Міністерство освіти і науки України

Харківський національний університет імені В. Н. Каразіна

Факультет (навчально-науковий інститут) Радіофізики, біомедичної електроніки та комп'ютерних систем

Кафедра ФБМЕ та КІТ

До захисту допущено

Кафедрою _____ *протокол №____ від _____*

завідувач кафедри _____ _____

(підпис) *(ім'я, прізвище)*

«___»_____ 202_ р.

Кваліфікаційна робота

здобувача _____другого (магістерського)_____ рівня вищої освіти

(першого (бакалаврського) / другого (магістерського))

МЕТОДИ ГЛИБИННОГО НАВЧАННЯ ДЛЯ КЛАСИФІКАЦІЇ ОФТАЛЬМОЛОГІЧНИХ ПАТОЛОГІЙ НА ОСНОВІ ФУНДУСНИХ ЗНІМКІВ ОКА

(назва роботи)

Спеціальність (спеціалізація) 176 Мікро- та наносистемна техніка

(код та найменування спеціальності; спеціалізації спеціальності - за наявності)

Освітня програма Фізична та біомедична електроніка

(назва освітньої програми)

Виконавець _____ Андрій Шегера

(підпис) *(ім'я, прізвище)*

Науковий керівник _____ *Ольга Величко*

(підпис) *(ім'я, прізвище)*

Харків – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Факультет Радіофізики, біомедичної електроніки та комп'ютерних систем

Кафедра ФБМЕ та КІТ

Спеціальність 176 Мікро- та наносистемна техніка

Освітньо-професійна програма Фізична та біомедична електроніка

Рівень вищої освіти другий (магістерський)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ _____

підпис ініціали, прізвище

"____" _____20___

року

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ

ШЕГЕРА Андрій Юрійович

(прізвище, ім'я, по батькові студента)

1. Тема роботи _Методи глибинного навчання для класифікації офтальмологічних патологій на основі фундусних знімків ока_

керівник роботи Величко Ольга Миколаївна, к.т.н., доц.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від " 24 "___10____2025 року № 1401-5/3957

2. Строк подання студентом роботи __08.12.2025_____

3. Перелік питань, які потрібно розробити:

1. Провести аналітичний огляд існуючих підходів до аналізу фундусних знімків ока.
2. Провести програмний аналіз набору даних ODIR-5K (https://www.kaggle.com/datasets/andrewmvd/ocular-disease-recognition-odir5k) для виявлення методологічних проблем оригінального маркування.
3. Створити валідований набір даних, придатний для мультилейблової класифікації окремого ока.
4. Обґрунтувати, спроектувати та реалізувати просунуту модель на основі трансферного навчання, використовуючи архітектуру ResNet50.

5. Провести якісний аналіз просунутої моделі трансферного навчання.4. План роботи

№ з/п	Назви етапів роботи
1	Аналітичний огляд джерел за темою роботи
2	Формулювання мети та задач, що повинні бути вирішені в дипломній роботі
3	Підготовка набору даних
4	Проектування, реалізація та аналіз базової моделі
5	Розробка та аналіз просунутої моделі на основі трансферного навчання
6	Аналіз та обговорення отриманих результатів
7	Підготовка пояснювальної записки
8	Підготовка доповіді та презентації до захисту

5. Дата видачі завдання 24.10.25

Студент _____ А.Ю. Шегера

підпис ініціали, прізвище

Керівник роботи _____ О.М. Величко

підпис ініціали, прізвище

АНОТАЦІЯ

ШЕГЕРА А. Ю. Методи глибинного навчання для класифікації офтальмологічних патологій на основі фундусних знімків ока. Кваліфікаційна робота магістра. Харківський національний університет імені В.Н. Каразіна, Харків, 2025. 289 с., 110 рис., 2 табл., 4 дод., 38 джерел.

ГЛИБИННЕ НАВЧАННЯ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, КЛАСИФІКАЦІЯ ЗОБРАЖЕНЬ, МЕДИЧНА ДІАГНОСТИКА, ОФТАЛЬМОЛОГІЯ, ТРАНСФЕРНЕ НАВЧАННЯ.

Об'єкт дослідження: процеси класифікації патологічних станів ока за фундусними знімками.

Предмет дослідження: методи глибинного навчання та архітектури згорткових нейронних мереж для мультилейблової класифікації.

Мета роботи: розробка та аналіз ефективної архітектури глибинного навчання для класифікації офтальмологічних патологій на основі фундусних знімків ока.

Методи дослідження: програмний аналіз даних, трансферне навчання на базі ResNet50, модифікована функція втрат Focal Loss, методи інтерпретації Grad-CAM.

Результати дослідження: Проведено аналіз та очищення набору даних ODIR-5K, на основі якого згенеровано новий валідований набір даних. Доведено, що застосування трансферного навчання та модифікованої функції втрат дозволило значно підвищити якість класифікації та навчити модель фокусуватися на клінічно релевантних анатомічних структурах.

Отримані результати можуть бути використані як основа для створення допоміжних інструментів підтримки прийняття рішень (CAD) або систем попереднього скринінгу в офтальмології.

ABSTRACT

SHEGERA A. Y. Deep learning methods for the classification of ophthalmological pathologies based on fundus eye images. Master's qualification thesis. V. N. Karazin Kharkiv National University, Kharkiv, 2025. 289 p., 110 fig., 2 table, 4 app., 38 ref.

CONVOLUTIONAL NEURAL NETWORKS, DEEP LEARNING, IMAGE CLASSIFICATION, MEDICAL DIAGNOSTICS, OPHTHALMOLOGY, TRANSFER LEARNING.

Object of research: processes of classifying pathological conditions of the eye based on fundus images.

Subject of research: deep learning methods and architectures of convolutional neural networks for multi-label classification.

Purpose of the work: to develop and analyze an effective deep learning architecture for the classification of ophthalmological pathologies based on fundus eye images.

Research methods: programmatic data analysis, transfer learning based on ResNet50, modified Focal Loss function, and Grad-CAM interpretation methods.

Research results: An analysis and cleaning of the ODIR-5K dataset were performed, which resulted in the generation of a new validated dataset. It has been demonstrated that the use of transfer learning and a modified loss function significantly improved classification quality and enabled the model to focus on clinically relevant anatomical structures.

The obtained results can be used as a basis for creating computer-aided diagnosis (CAD) systems or preliminary screening tools in ophthalmology.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ/ПОЗНАЧЕНЬ

Скорочення/ Позначення	Розшифрування
AdamW	Адаптивний оптимізатор з незалежним спадом ваги (Adam with Weight Decay)
AUC	Площа під кривою (Area Under the Curve)
AUC (PR)	Площа під кривою точність-повнота (Area Under the Precision-Recall Curve)
AUC (ROC)	Площа під кривою робочих характеристик (Area Under the Receiver Operating Characteristic Curve)
CAD	Система допомоги прийняття рішень (computer-aided diagnosis)
FPR	Частка хибно позитивних спрацьовувань (False Positive Rate)
FN	Хибно негативний (False Negative)
FP	Хибно позитивний (False Positive)
TN	Істинно негативний (True Negative)
TP	Істинно позитивний (True Positive)
MLP	Багатошаровий перцептрон (Multilayer Perceptron)
ODIR-5K	Інтелектуальне розпізнавання очних захворювань (Ocular Disease Intelligent Recognition)
ReLU	Випрямлений лінійний блок (Rectified Linear Unit)
ResNet	Залишкова мережа (Residual Network)

RGB	Червоний, зелений, синій (Red, Green, Blue)
TPR	Частка істинно позитивних спрацьовувань (True Positive Rate/Recall)
VRAM	Відеопам’ять (Video Random Access Memory)

ВСТУП

Стрімкий розвиток методів глибинного навчання (Deep Learning) у комп'ютерному зорі (Computer Vision) відкриває нові можливості для автоматизації та підвищення об'єктивності діагностики в офтальмології [1]. Існує нагальна потреба у надійних, швидких та об'єктивних інструментах для аналізу фундусних знімків (зображень очного дна), оскільки рутинна обробка великих обсягів даних є ресурсозатратною і залежить від експертної, але суб'єктивної оцінки спеціаліста. Наявність коморбідності (кількох патологій одночасно) у пацієнтів вимагає розробки архітектур, здатних до мультилейблової класифікації. Класичні архітектури нейронних мереж (наприклад, багатошаровий перцептрон) є неефективними для роботи з візуальною інформацією через надмірну кількість параметрів та втрату просторового контексту [6]. З огляду на необхідність надійного розпізнавання складних візуальних патернів, розробка спеціалізованих моделей на основі згорткових нейронних мереж (CNN) та трансферного навчання є актуальною та доцільною [11].

Тема дослідження пов'язана з напрямками Харківського національного університету імені В. Н. Каразіна, зокрема, з розробкою та аналізом комплексних інформаційних технологій та біомедичної електроніки, які передбачають застосування методів штучного інтелекту для вирішення аналітичних завдань у галузі медицини.

Мета роботи полягає у розробці та всебічному аналізі ефективної архітектури глибинного навчання для мультилейблової класифікації офтальмологічних патологій на основі фундусних знімків ока.

Для досягнення поставленої мети необхідно розв'язати наступні задачі:

1. Обрати та провести всебічний програмний аналіз вихідного набору даних, виявити методологічні проблеми маркування та сформувати новий, коректний набір мультилейблових цільових міток для кожного окремого ока [23].

2. Розробити, реалізувати та проаналізувати базову модель згорткової нейронної мережі як відправну точку (baseline), визначити її обмеження та сформулювати вимоги до вдосконалення архітектури [19].

3. обґрунтувати, спроектувати та реалізувати просунуту модель на основі трансферного навчання та сучасних методів регуляризації для подолання можливих проблеми перенавчання.

4. Розробити та застосувати модифіковану функцію втрат із клінічно обґрунтованими штрафами для ефективної боротьби з дисбалансом класів та нелогічними комбінаціями діагнозів.

5. Провести порівняльний аналіз продуктивності базової та просунутої моделей використовуючи метрики придатні для роботи з незбалансованими даними (AUC (PR), F-beta score), та надати якісний аналіз за допомогою Grad-CAM.

Об’єкт дослідження - процеси класифікації патологічних станів ока за візуальними даними (фундусними знімками).

Предмет дослідження - методи глибинного навчання, зокрема архітектури згорткових нейронних мереж, для автоматичної мультилейблової класифікації офтальмологічних патологій.

Для реалізації поставлених задач буде застосовано комплекс методів: глибоке навчання (Deep Learning) на основі згорткових нейронних мереж (CNN), трансферне навчання (Transfer Learning) з використанням зовнішньої архітектури для підвищення узагальнюючої здатності. Також будуть використані методи регуляризації (Dropout [29], аугментація даних [24] тощо), адаптивний оптимізатор AdamW [32] для стабілізації навчання, модифікована функція втрат Focal Loss [33] зі штрафами для подолання дисбалансу та методи оцінки та інтерпретації (Grad-CAM, [36] F-beta score) для обґрунтування достовірності результатів.

Очікувана наукова новизна полягатиме в:

1. Створенні нового, очищеного та валідованого набору даних final_dataset.csv, придатного для мультилейблової класифікації окремого ока, шляхом мапінгу 94

унікальних діагностичних термінів на 8 цільових класів на основі програмного аналізу оригінальної розмітки [23].

2. Розробці та впровадженні модифікованої функції втрат Focal Loss з додатковими інтегрованими штрафами, спрямованими на контроль надлишкового прогнозування кількості класів та запобігання нелогічним комбінаціям, що дозволить ефективніше працювати з незбалансованими мультилейбловими медичними даними.

3. Обґрунтуванні значної переваги трансферного навчання над навчанням з нуля в умовах обмеженої медичної вибірки, підтвердженої кількісними метриками, та доведенні здатності трансферної моделі фокусуватися на клінічно релевантних анатомічних структурах (за допомогою Grad-CAM [36]), а не на технічних артефактах, що продемонструє її потенційну надійність.

Отримані результати потенційно можуть бути використані як допоміжний інструмент підтримки прийняття рішень (CAD) або система попереднього скринінгу. Модель потенційно матиме потенціал для оптимізації роботи офтальмологів шляхом виявлення потенційно проблемних випадків у великому потоці знімків, що потребують першочергової уваги лікаря.

АНАЛІТИЧНИЙ ОГЛЯД

Протягом останніх років спостерігається постійне зростання інтересу до застосування алгоритмів штучного інтелекту (ШІ) та машинного навчання (МН) в медицині [1]. Стрімке зростання обсягів медичних даних, включно з електронними медичними картками [2] та великими масивами цифрових зображень, зокрема фотографій очного дна (фундусних знімків), вимагає розробки нових інструментів для ефективного аналізу. Класичні статистичні методи часто не здатні впоратися з значними за обсягом та складними за структурою наборами даних, тоді як методи машинного навчання можуть знаходити приховані закономірності, текстури, патерни, що э непомітними людському оку.

Основні підходи машинного навчання, які застосовуються в охороні здоров’я, включають регресію, класифікацію та глибокі нейронні мережі [6]. Ці методи можуть підвищити ефективність роботи лікарів у різних напрямках від прогнозування лікувальних результатів до підтримки прийняття клінічних рішень у реальному часі. Як зазначають дослідники, інтеграція машинного навчання є важливим кроком і сучасним трендом у рутинній медичній практиці, але вона вимагає високих обчислювальних ресурсів, а також чітких регуляторних норм у сфері збереження конфіденційності даних та етичних аспектів.

Концептуальні основи, що лежать в основі сучасного штучного інтелекту, сягають середини XX століття. У 1936 році А. М. Тюрінг (A. M. Turing) опублікував роботу про обчислювані числа [7]. У 1943 році В. С. Маккалох (W. S. McCulloch) і В. Піттс (W. Pitts) розробили логічне числення ідей, притаманних нервовій діяльності [8]. Роботи Я. ЛеКуна (Y. LeCun) та його колег кінця 1990-х років були присвячені градієнтному навчанню для розпізнавання документів [11].

Стрімкий розвиток глибокого навчання (Deep Learning, DL) у комп’ютерному зорі (Computer Vision) відкрив нові можливості для аналізу візуальної інформації. Глибоке навчання базується на використанні багатошарових архітектур, здатних вивчати ієрархію ознак.

Згорткові нейронні мережі (Convolutional Neural Networks, CNN) є ключовими архітектурами для роботи з візуальними даними. Вони вирішують

проблему надмірної кількості параметрів та втрати просторового контексту властиву класичним повнозв’язним мережам (MLP) [6]. Роботи О. Руссаковського (O. Russakovsky) та співавторів, а також Дж. Денга (J. Deng) та колег з розробки масштабної ієрархічної бази зображень ImageNet створили основу для розвитку глибокого навчання [12, 13]. У 2012 році робота А. Крижевського (A. Krizhevsky) зі співавторами продемонструвала прорив у класифікації ImageNet за допомогою глибоких згорткових мереж [14].

Для подолання проблеми зникаючого градієнта та побудови надзвичайно глибоких і ефективних мереж, які є менш схильними до деградації точності, були розроблені залишкові мережі (Residual Network, ResNet) [19]. Ключова ідея ResNet полягає у використанні залишкових блоків (residual blocks) зі “скороченими шляхами” (shortcut connections). Це дозволяє шарам вивчати залишкову функцію, спрощуючи оптимізацію та дозволяючи мережі ефективно визначати свою глибину.

У галузі медичної діагностики, включно з офтальмологією, глибоке навчання демонструє значний потенціал. Наприклад, якісні глибокі нейронні мережі здатні забезпечувати класифікацію патологій на рівні фахівців [3].

Трансферне навчання (Transfer Learning) є ключовою стратегією для роботи з медичними даними, особливо коли обсяги доступної вибірки обмежені [25]. Стратегія полягає у використанні ваг, попередньо навчених на масштабному наборі даних загального призначення, як-от ImageNet, а потім точній їхній настройці (Fine-Tuning) під вузькоспеціалізовану медичну задачу. Це дозволяє моделі ефективно перенести універсальні знання про візуальні ознаки (лінії, текстури) на специфічні медичні патерни. У медичних дослідженнях, зокрема в офтальмології, часто використовують архітектури, як-от ResNet50, навчені на ImageNet, як надійну основу.

У контексті класифікації офтальмологічних захворювань застосування глибокого навчання є вкрай актуальним для автоматизації діагностики на основі фундусних знімків ока. Як приклад, використовується публічний набір даних ODIR-5K (Ocular Disease Intelligent Recognition), який містить зображення очного

дна та відповідні мультилейблові діагностичні мітки для восьми класів патологій [23].

Незважаючи на значні успіхи, існують фундаментальні виклики, які перешкоджають надійному впровадженню глибоких систем у клінічну практику. Проблема дисбалансу класів (Class Imbalance) у медичних наборах даних, зокрема ODIR-5K полягає у тому, що рідкісні патології представлені значно меншою кількістю прикладів порівняно з нормою або поширеними захворюваннями. Це призводить до того, що модель, мінімізуючи загальну помилку, може ігнорувати рідкісні, але критично важливі класи. Для ефективної роботи необхідні спеціалізовані техніки компенсації дисбалансу, такі як зважування класів у функції втрат [33]. Також виникає проблема мультилейблової класифікації, адже у реальній клінічній практиці поширеною є коморбідність (наявність кількох патологій одночасно). Це вимагає розробки архітектур, здатних до мультилейблової класифікації, а не спрощеного мультикласового підходу, що призводить до втрати діагностичної інформації. Також варто враховувати ненадійність та неінтерпретованість моделей, адже моделі глибокого навчання часто є “чорними скриньками” [37]. Базова архітектура може страждати від надлишкової ємності (overcapacity) та навчання на хибних шляхах (shortcut learning), тобто покладатися на нерелевантні артефакти або шумові патерни замість справжніх біологічних ознак патологій [36].

Проведений аналіз демонструє, що існують значні прогалини у розробці якісних і стійких до проблем моделей для офтальмології які б ефективно працювали в умовах реального клінічного дисбалансу та мультилейблової природи даних. Хоча трансферне навчання на базі ResNet є перспективним, необхідно верифікувати та очистити вихідний набір даних для створення коректних мультилейблових міток. Обґрунтувати перевагу трансферного навчання (наприклад, ResNet [19]) над навчанням з нуля в умовах обмеженої медичної вибірки. Розробити та застосувати модифіковану функцію втрат із клінічно обґрунтованими штрафами для боротьби з дисбалансом і нелогічними комбінаціями діагнозів [33]. Підтвердити здатність трансферної моделі

фокусуватися на клінічно релевантних анатомічних структурах за допомогою методів інтерпретації, наприклад Grad-CAM [36].

Таким чином, розробка та всебічний аналіз ефективної архітектури глибинного навчання для мультилейблової класифікації офтальмологічних патологій на основі фундусних знімків ока є актуальною та доцільною. Це дозволить створити прототип допоміжного інструменту підтримки прийняття рішень (CAD) або систему попереднього скринінгу, здатну оптимізувати роботу офтальмологів.

1 ТЕОРЕТИЧНІ ОСНОВИ ГЛИБОКОГО НАВЧАННЯ

1.1 База про згорткові нейронні мережі

В контексті сучасних аналітичних завдань у комп'ютерному зорі (Computer Vision) особливо актуальною є здатність нейронних мереж до самостійного виявлення нетривіальних закономірностей в даних [6]. Як можна зрозуміти з назви, задачі реалізації комп'ютерного зору ставлять перед собою за основу інтерпретацію візуальної інформації, як правило у вигляді файлів зображень [11].

Для подібних проблем більш класичні методи побудови нейромереж (такі як, наприклад, багатошаровий перцептрон (MLP) [6, 9], в якому кожен нейрон одного шару пов'язаний з кожним нейроном сусіднього наступного) стикаються з фундаментальними недоліками [6] (рис. 1.1).

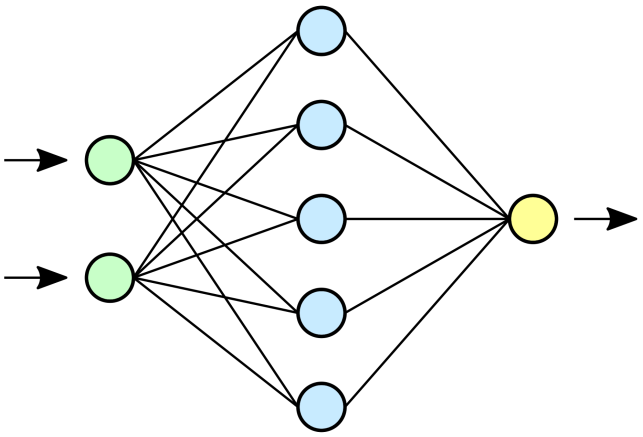


Рисунок 1.1 - Приклад нейронної мережі

Класичні нейроні мережі адаптовані під роботу з структурованими даними, наприклад табличними даними, або іншими видами даних в яких можна простежити чіткі закономірності [25]. Як можна зрозуміти, зображення це дані зовсім іншої, набагато менш детермінованої просторово-корельованої природи [11]. Спроби застосування класичних архітектур до них часто не мають сенсу і ефективних результатів. До проблем, що виникають під час подібних спроб можна віднести, наприклад, надмірну кількість параметрів [15]. Візьмемо до прикладу

зображення на рис. 1.2 розміром 224x224 пікселів з трьома класичними для зображень каналами кольорів - червоний, зелений, синій (RGB).

Рисунок 1.2 - Приклад кольорового зображення очного дна

В такому випадку, на кожний піксель приходиться три значення кольорів, і сама кількість пікселів дорівнює добутку ширини зображення на його висоту, тобто (1.1):

$$N_{features} = H \cdot W \cdot C; \quad (1.1)$$

де $N_{features}$ - кількість вхідних ознак;

H - висота зображення у пікселях;

W - ширина зображення у пікселях;

C - кількість каналів кольору на піксель.

Звідси:

$$224 \cdot 224 \cdot 3 = 150\,528.$$

Вихідних значень для такого зображення. Варто враховувати, що зображення такого розміру можна вважати доволі малим і не завжди достатнім для повної інформативності в певних задачах. Перший же прихований шар подібної

нейромережі містив бі в собі десятки або більше мільйонів вагових коефіцієнтів. У нашому випадку це виглядатиме як (1.2):

Кількість параметрів (ваг та зсувів) P для одного (в реальних випадках шарів часто більше) шару, що поєднує N_in входів з N_out нейронами:

$$P = (N_{in} \cdot N_{out}) + N_{out} ; \quad (1.2)$$

де

P - кількість параметрів (ваг та зсувів) для одного шару;

$(N_{in} \cdot N_{out})$ - кількість вагових коефіцієнтів,

а N_{out} - кількість параметрів зсуву (bias), по одному на кожен нейрон.

Зсув надає моделі додаткову гнучкість, дозволяючи нейрону регулювати свій поріг активації.

Візьмемо до прикладу шар з 1024 нейронами, звідси:

$$150528 \cdot 1024 + 1024 = 154\,141\,696 ;$$

параметрів що призводить до не стабільної роботи моделі та складностей в навчанні через схильність до перенавчання [29] і подібних негативних ефектів [26].

Під перенавчанням мається на увазі стан нейромережі, коли на тренувальних даних точність класифікації такої надвисока і наближається до 100%, але на реальних, раніше небачених моделлю, прикладах точність класифікації наднизька і наближається до 0% [6], тобто пряме запам’ятовування тренувальних даних, без практичної здібності до загальної апроксимації закономірностей. Перенавчання можна апроксимувати як розходження графіків навчання при аналізі подібно рис. 1.3.

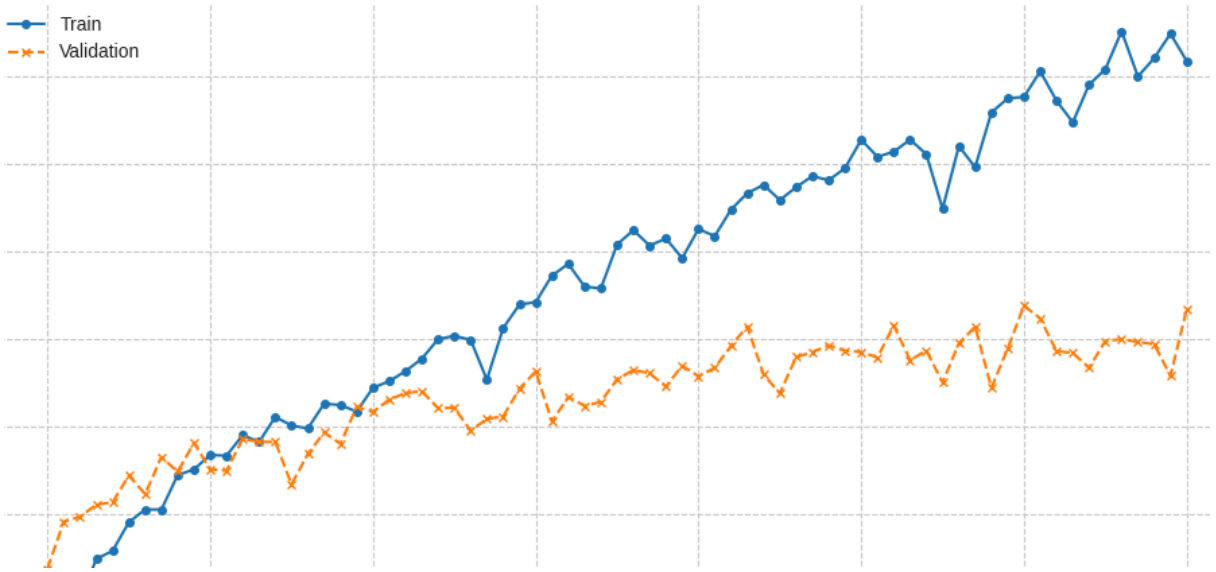


Рисунок 1.3 - Приклад візуального перенавчання на графіку

Також, повнозв’язні мережі вимагають перетворення вхідного зображення в єдиний одномірний вектор, по суті повністю ігноруючи його двовимірну структуру. Таким чином, такі нейромережі втрачають контекст позиціонування пікселів - для них немає різниці між пікселем знизу і всередині зображення, тому що все зображення представлене як один довгий вектор. Так втрачається інформація про те, які пікселі є сусідами, і внаслідок цього така мережа не здатна ефективно розпізнавати візуальні патерни і текстури на зображенні - базові елементи, з яких складається будь-яке зображення, такі як: лінії, криві, контури, градієнти тощо [16].

З метою вирішення цих та інших проблем були розроблені більш спеціалізовані архітектури [11, 14]. Золотим стандартом таких архітектур є так звані згорткові нейронні мережі або Convolutional Neural Networks (CNN) [11] (рис. 1.4).

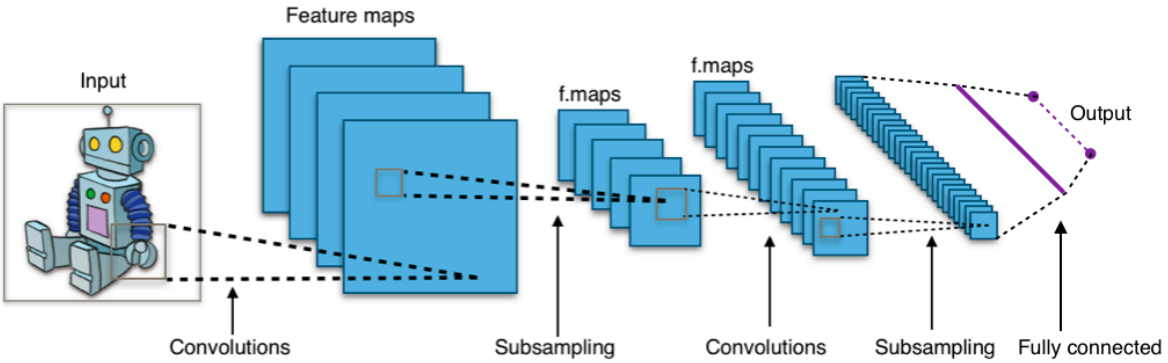


Рисунок 1.4 - Спрощений приклад роботи згорткової нейронної мережі

Такі мережі базуються на принципах, натхненних роботою зорової кори головного мозку людини з ціллю ефективного виявлення ієрархій візуальних ознак на зображеннях [8, 6, 25]. У цьому розділі детально розглянемо теоретичні основи технічної імплементації ЗНМ, починаючи з фундаменту побудови будь якої нейронної мережі - штучного нейрона [8]. Розгляд “знизу вверх” дозволить поступово прийти до розуміння складних механізмів, що лежать в основі таких моделей і дозволяють їм досягати високої ефективності в задачах комп’ютерного зору. Розгляд теоретичних основ дозволить зробити обґрунтований вибір архітектури для подальшої розробки і аналізу базової моделі класифікації патологій на зображеннях очного дна.

1.2 Штучний нейрон: від перцептрона до сучасної обчислювальної одиниці

Фундаментальною атомарною одиницею всіх нейронних мереж є штучний нейрон (рис. 1.5). Еволюція нейрону пройшла шлях від тривіальної порогової моделі до сучасних ефективних обчислювальних одиниць і активно продовжується і на момент написання роботи.

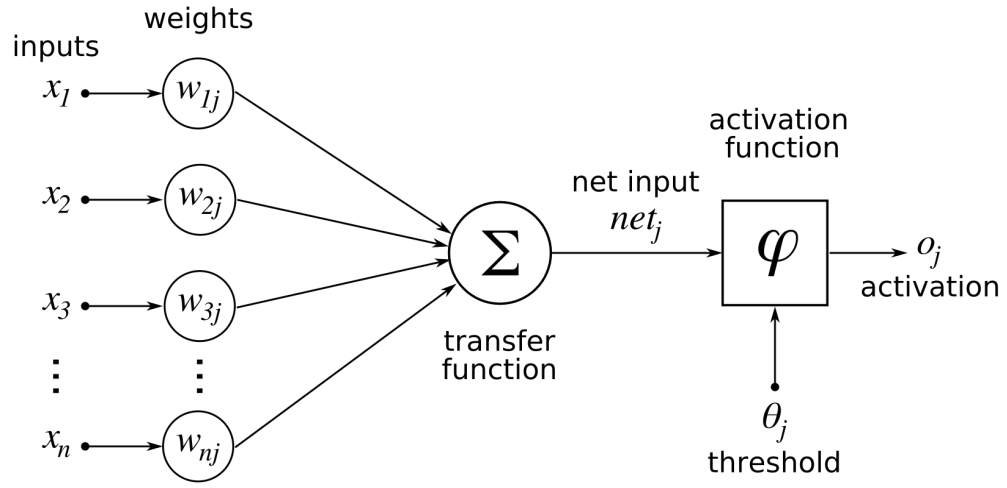


Рисунок 1.5 - Приклад штучного нейрона

Першою формалізованою моделью став перцептрон Френка Розенблатта у 1957р [9], який приймав бінарні рішення (на рівні “так або ні”) залежно від вхідних умов.

Принцип роботи схожий на операцію If в програмуванні. Припустимо, маємо декілька змінних вхідних показників і очікуємо бінарний вивід від системі на базі одного перцептрону. Тоді перцептрон обчислить зважену суму всіх своїх входів x_i з вагами w_i, порівнює результат з певним порогом і, у разі якщо сума більша за цей поріг, то вихід буде дорівнювати 1, а інакше 0.

Під вагами мається на увазі числові коефіцієнти в середині самих нейронів, що визначають важливість того чи іншого входу, те, наскільки сильно він впливає на кінцевий результат. Ваги підбираються під час ітеративного процесу навчання.

Така поведінка може бути описана пороговою функцією активації (step function) (1.3):

$$y = \begin{cases} 1 & \text{if } \sum_i w_i x_i > \theta \\ 0 & \text{otherwise} \end{cases} ; \qquad (1.3)$$

де θ - порогове значення активації встановлене вручну;

x_i - входи;

w_i - ваги;

Проте перцептрон має суттєвий недолік - власне, його функцію активації. Вона не є диференційованою, тобто має розрив у точці порогу, що, в свою чергу, робить неможливим застосування стандартних, сучасних, ефективних методів навчання, базованих на градієнті [6]. Для такої функції активації неможливо вирахувати градієнт (похідну), неможливо зрозуміти напрямок функції, адже вона схожа на сходинку, де у точці між 0 і 1 відбувається різка вертикальна зміна значення функції, а не плавна як, наприклад, для лінійної функції. Градієнт дає змогу зрозуміти, як невелика зміна ваги може вплинути на помилку моделі, що дозволяє застосовувати більш ефективні методи навчання.

У свою чергу, сучасний, або ReLU (Rectified Linear Unit) [27], нейрон являє собою гнучку модель, спеціально оптимізовану для навчання методом градієнтного спуску. Замість жорсткого бінарного виводу сучасні нейрони видають оцінку ймовірності в діапазоні від 0 до 1. Такі нейрони мають дві ключові математичні компоненти: лінійну і не лінійну частини.

Лінійна частина, або зважена сума, є першим етапом обчислень всередині нейрона. Вона агрегує вхідну інформацію і складається з двох типів навчаємих параметрів: ваг w_i та параметру зсуву b. Ваги визначають силу впливу кожного відповідного вхідного сигналу x_i. У процесі навчання модель підбирає їх таким чином, щоб надати більшої важливості тим входам, які є більш інформативними для вирішення задачі. Параметр зсуву, на відміну від ваг, не пов'язаний з жодним конкретним входом. Він є незалежним, вільним членом рівняння і виконує роль навчаємого порогу активації. Під “спрацьовуванням” (активацією) нейрона мається на увазі момент, коли результат його лінійної частини z стає достатньо великим, щоб функція активації (наприклад, ReLU) видала ненульове значення. Без зсуву, цей поріг є фіксованим залежно від функції активації (наприклад, z>0). Це обмежує нейрон. Уявімо ситуацію, коли для правильного вирішення задачі нейрон повинен активуватися, лише якщо зважена сума входів перевищує

значення 10. Без параметру зсуву, він буде спрацьовувати при будь-якому позитивному значенні, що є неправильною поведінкою. Параметр зсуву b вирішує цю проблему. У процесі навчання, який є ітеративним (тобто, параметри постійно коригуються відносно попереднього стану, а не обчислюються з нуля), модель може підібрати для цього нейрона негативне значення зсуву, наприклад, b = -10. Тоді для того щоб значення z стало позитивним, зважена сума входів повинна бути більшою за 10. І навпаки, позитивне значення b полегшує активацію, дозволяючи нейрону спрацьовувати навіть при слабких або негативних вхідних сигналах. Геометрично, це означає зміщення графіка функції активації вздовж горизонтальної осі (осі абсцис), де по цій осі відкладається значення зваженої суми z. Ця гнучкість дозволяє кожному нейрону в процесі ітеративного навчання знайти свій власний, оптимальний поріг спрацьовування для будь-якого розподілу даних. Зважену суму можна розрахувати як (1.4):

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b \quad ; \tag{1.4}$$

Нелінійна частина, або функція активації, це частина яка відповідає за пропускання результату z через диференційовану функцію активації f(z). Сучасним золотим стандартом такої функції є так звана ReLU (Rectified Linear Unit) [27] (рис 1.6), що повертає або копію входу у випадку якщо його значення позитивне, або нуль в іншому випадку (1.5):

$$y = f(z) = \max(0, z) \quad ; \tag{1.5}$$

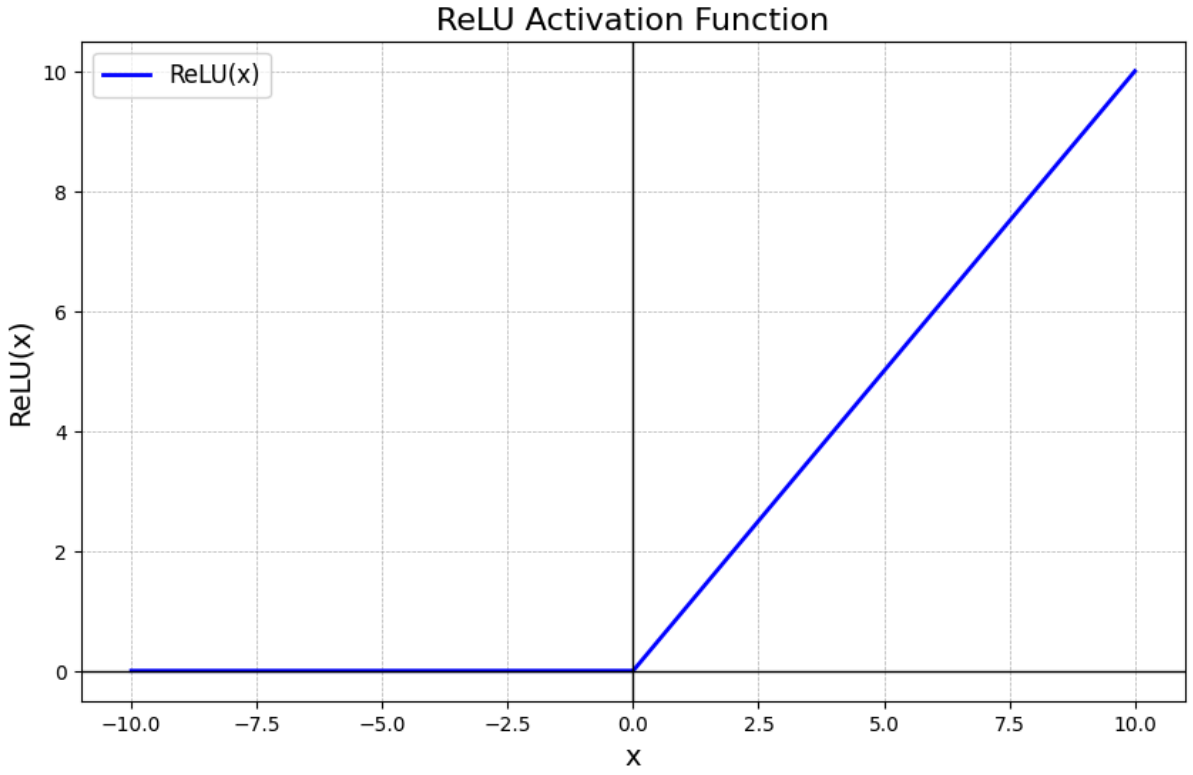


Рисунок 1.6 - Функція активації ReLU

До прикладу, розглянемо розрахунок виходу одного нейрона в межах вже навченої мережі. Припустимо, що у нас є декілька вхідних сигналів (входи від даних, або виходи попереднього шару тощо): x_1 = 0.1, x_2 = 2.2, x_3 = -0.55. Також знаємо параметри нейрона, що отримали в процесі навчання:

Зсув: b=0.5

Ваги: w_1 = 0.91, w_2 = -1.6, w_3 = 3.0

Розрахуємо зважену суму z (1.6):

$$z = (w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3) + b \quad ; \qquad (1.6)$$

Підставимо значення:

$$z = (0.91 \cdot 0.1 + (-1.6) \cdot 2.2 + 3.0 \cdot (-0.55)) + 0.5 \quad ;$$

$$z = (0,091 - 3,52 - 1,65) + 0.5 \text{ ;}$$

$$z = -5.079 + 0.5 = -4.579 \text{ ;}$$

Теперь застосуємо функцію активації ReLU (1.7):

$$y = \text{ReLU}(z) = \max(0, -4,579) \text{ ;} \quad (1.7)$$

$$y = 0 \text{ ;}$$

Як результат, бачимо, що вихідний сигнал цього нейрона буде дорівнювати 0, що означає, що для такої комбінації вхідних сигналів цей нейрон не буде активовано і, відповідно, він не передасть сигнал на наступний шар. Такий результат можна інтерпретувати як те, що нейрон не зміг розпізнати специфічний патерн, на який налаштовані його ваги. Проте те, на що не здатен один нейрон, може бути розпізнано іншим, можливо у глибших шарах, адже кожен нейрон являє собою атомарну одиницю яка навчається розпізнаванню вузькоспеціалізованої ознаки. Відповідно, наступний логічний крок - формування мереж пов’язуючи окремі нейрони між собою пошарово, що дозволяє мережі вивчати складні представлення даних.

1.3 Архітектура повнозв’язних мереж та процес навчання

Нейронні мережі є мережами саме через те, що включають в себе певну кількість раніше розглянутих нейронів, що пов’язані між собою і архітектурно сформовані у шари. Наприклад, класичні повнозв’язні (Fully Connected Network), або багатошарові перцептрони (MLP), нейронні мережі формуються за допомогою з’єднання виходу кожного нейрону кожного шару з входами кожного нейрону його сусіднього наступного шару (рис. 1.7) [6].

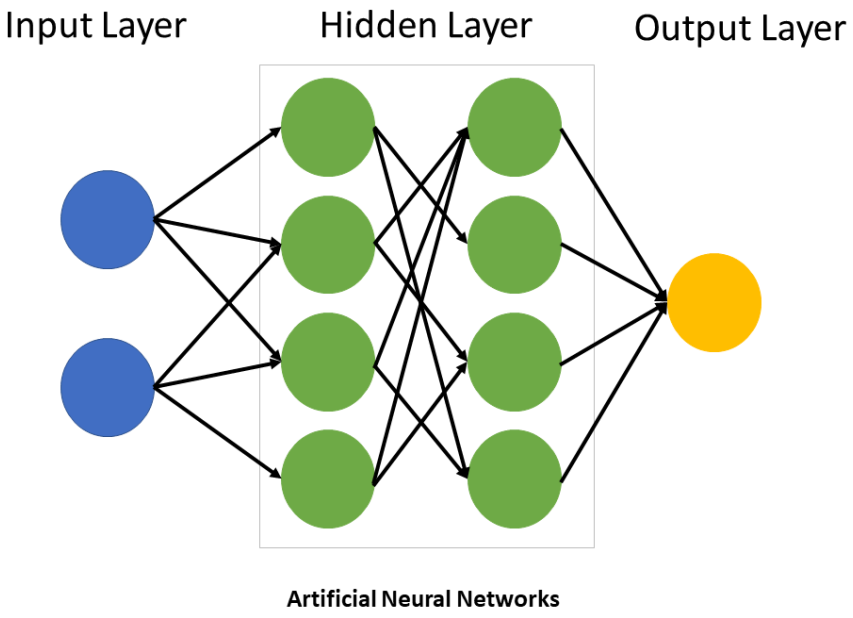


Рисунок 1.7 - Приклад спрощеної архітектури повнозвязної мережі

Такі мережі складаються з трьох базових типів шарів, а саме:

Вхідний шар (Input Layer) - кількість нейронів в ньому дорівнює розмірності вхідних даних (кількості пікселів, наприклад). Такий шар тільки один і відповідає за приймання і передачу даних на наступні шари, не проводячи обчислень. Його можна уявити як вхідний інтерфейс мережі. Вхідний шар (Input Layer) є особливим. Він не виконує жодних обчислень і не має параметрів, що навчаються. Його головна і єдина функція - декларативна. Він, по суті, є “вхідними воротами” мережі, що повідомляють їй, якого розміру та форми (shape) вхідні дані вона буде отримувати. Ця інформація є критично важливою для автоматичної побудови наступних шарів. Наприклад, щоб створити матрицю ваг W для першого прихованого шару, процесу необхідно знати дві величини: кількість нейронів у цьому шарі (наприклад, 1024) та розмірність вхідного вектора (150528). Саме розмірність вхідного вектора і визначається вхідним шаром. Без цієї інформації перший прихований шар не знав би, якого розміру матрицю ваг 1024 x 150528 йому потрібно створити, що призвело б до помилки конфігурації. Таким чином, вхідний шар не є обчислювальним прошарком, а скоріше обов'язковою

специфікацією, що дозволяє процесу автоматично і коректно ініціалізувати архітектуру мережі [6].

Прихований шар (Hidden Layer) - один або більше шарів нейронів між вхідним та вихідним шарами. Це і є основні шари, що навчаються за рахунок наявності параметрів для навчання (ваги і зсуви) і виявляють нелінійні залежності та абстрактні ознаки методом послідовного перетворення простору ознак у вхідних даних. Шари перетворювачі. Кожен такий шар перетворює вхідний вектор активацій a_{in} з попереднього шару у два етапи описані у блоці пояснення роботи окремого нейрону (лінійне перетворення і функція активації). Відмінність в тому, що в межах шару кожен нейрон працює паралельно з іншими і кожен з них у повнозв'язному шарі приймає на вхід весь вхідний вектор, але завдяки унікальному для себе набору параметрів, навчається розпізнавати свої власні специфічні патерни (ознаки) у вхідному векторі. Таким чином, разом, всі нейрони шару перетворюють вхідний вектор ознак на новий вихідний вектор, в якому кожен елемент показує міру наявності однієї з таких ознак, після чого новий вектор отримує наступний шар і так далі [25].

Вихідний шар (Output Layer) - відповідає за формування кінцевого результату і зазвичай має кількість нейронів відповідну кількості класів задачі. В своїй основі він працює аналогічно до прихованих шарів. Проте його конфігурація та функція активація підбирається так, щоб на виході отримувались інтерпритовані результати у вигляді ймовірностей. На вхід вихідного шару надходить вектор активацій a_{hidden} від останнього прихованого шару. Вихідний шар має свою власну матрицю ваг W_{out} та вектор зсувів b_{out}t і виконує стандартне лінійне перетворення вигляду (1.8):

$$z_{out} = W_{out} \cdot a_{hidden} + b_{out} \text{ ;} \qquad (1.8)$$

Результат z_{out} - це вектор сирих, тобто не інтерпретованих, значень - **логітів**. Логіти можуть приймати будь які значення в діапазоні (-∞, +∞) , наприклад 2;-13;8.23 тощо. Але, як бачимо, оцінювати ймовірність наявності класу по сирим логітам майже неможливо. Тому їх потрібно перетворити на ймовірності, а для цього пропустити через спеціальну функцію активації відповідно до контексту задачі класифікації. Такі функції можуть бути різними:

Для задач багатокласової класифікації (Multiclass) коли об'єкт може належати тільки до одного єдиного чітко детермінованого класу (квадрат, ні, кішка тощо) зазвичай використовують функцію Softmax (1.9) (рис 1.8). Ця функція збирає весь вектор логітів і перетворює його на вектор ймовірностей, сума яких завжди дорівнює 1, наприклад (0.2+0.3+0.5=1.0). Відповідно клас з найбільшою ймовірністю буде вважатися фінальним передбаченням моделі:

$$Softmax(z_i) = e^{z_i} / \Sigma(e^{z_j}) \text{ ;} \qquad (1.9)$$

де z_i - це логіт для і-го класу,

e - основа натурального логарифма (число Ейлера, ≈ 2.718),

$\sum(e^{z_j})$ - це сума експонент від усіх логітів у векторі [6].

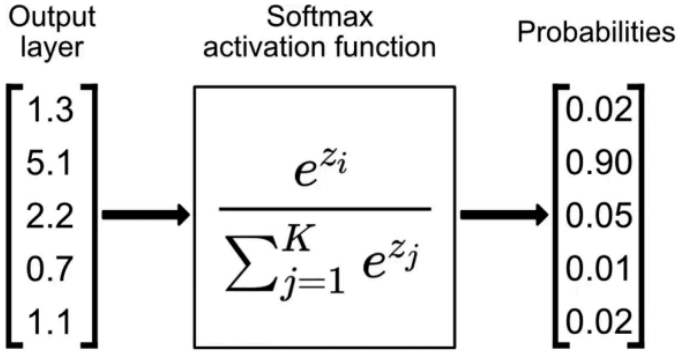


Рисунок 1.8 - Приклад роботи Softmax

Операція експоненціювання робить усі значення позитивними, а подальше ділення на суму нормалізує вектор так, щоб сума всіх його елементів дорівнювала одиниці.

Приведемо приклад розрахунку:

Припустимо, вихідний шар для 3 класів видав такі логіти: z = (2.0, 1.0, 0.1). Обчислюємо експоненти:

$$e^{2} \approx 7.39;$$

$$e^{1} \approx 2.72;$$

$$e^{0.1} \approx 1.11;$$

Обчислюємо їх суму:

$$\sum(e^{z_j}) = 7.39 + 2.72 + 1.11 = 11.22;$$

Обчислюємо фінальні ймовірності:

$$p_1 = 7.39 / 11.22 \approx 0.66;$$

$$p_2 = 2.72 / 11.22 \approx 0.24;$$

$$p_3 = 1.11 / 11.22 \approx 0.10;$$

Результат - вектор ймовірностей p=(0.66, 0.24, 0.10). Відповідно, сума:

$$0.66+0.24+0.10=1.0.$$

Модель прогнозує об'єкт першого класу.

Для задач багатолейблової класифікації (Multilabel) коли об'єкт може належати до кількох класів одночасно (наприклад, у пацієнта може бути і глаукома і катаракта) використовують функцію Sigmoid (1.10) (рис. 1.9). Ця функція застосовується до кожного логіта індивідуально і перетворює кожне число в ймовірність в діапазоні (0,1) незалежно від інших:

$$Sigmoid(z_i) = 1/(1 + e^{-z_i}) ; \quad (1.10)$$

де z_i- це логіт для і-го класу,

e - основа натурального логарифма [6].

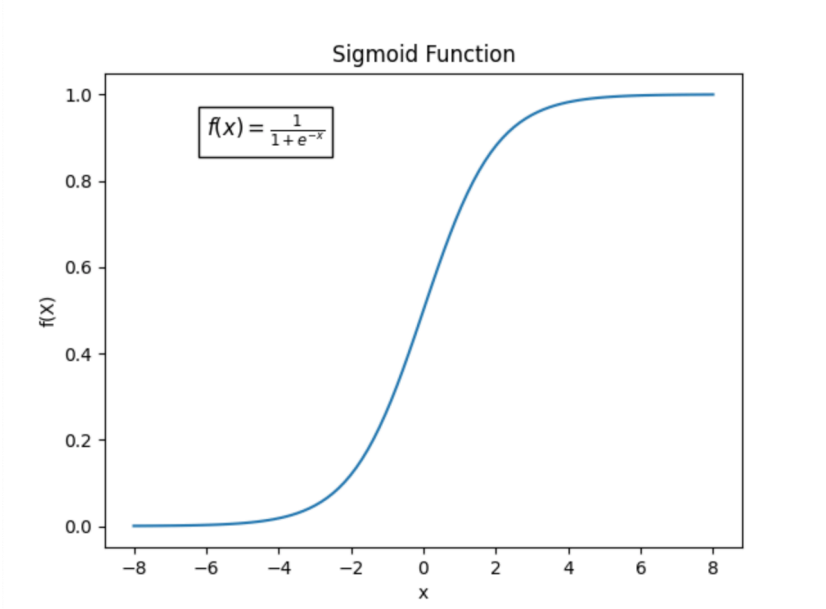


Рисунок 1.9 - Приклад графіку Sigmoid

Важливо пам’ятати, що Sigmoid застосовується до кожного логіта незалежно, і результат для одного класу ніяк не впливатиме на результати для інших.

Приведемо приклад розрахунку:

Припустимо, вихідний шар для 3 класів у задачі багатолейблової класифікації видав ті ж самі логіти: z = (2.0, 1.0, 0.1).

Обчислюємо Sigmoid для кожного елемента:

$$p_1 = \sigma(2.0) = \frac{1}{1+(e^{-2})} = \frac{1}{1+0.135} \approx 0.88;$$

$$p_2 = \sigma(1.0) = \frac{1}{1+(e^{-1})} = \frac{1}{1+0.368} \approx 0.73;$$

$$p_3 = \sigma(0.1) = \frac{1}{1+(e^{-0.1})} = \frac{1}{1+0.905} \approx 0.52.$$

Отримали вектор незалежних ймовірностей p=(0.88, 0.73, 0.52). Варто помітити, що їх сума вочевидь не дорівнює одиниці. Таким чином, модель на 88% впевнена в наявності першого класу, на 73% - другого і на 52% - третього. Це фінальний вихід моделі, але такі дані можна окремо інтерпретувати і надалі на базі індивідуальних порогів класів, наприклад отримавши вектор (0.9, 0.23, 0.51) з відповідними встановленими завчасно порогами (0.5, 0.2, 0.6) будемо вважати, що перший та другий клас наявні (0.9>05) та (0.23>0.2) що можна перевести в одиницю для цих класів, а третій не виявлено адже (0.51<0.6), що інтерпретуємо як нуль. Таким чином можна отримати вектор ймовірностей, який надалі можна перетворити в бінарний вектор (1,1,0), якщо потрібно відповідно до умов задачі.

У нейронних мережах процес обчислення і отримання прогнозу (класифікації) називається прямим поширенням (Forward Propagation) і, відповідно, він проходить послідовно починаючи зі входу і закінчуючи виходом, тобто виходи нейронів і-го шару це входи для (і+1)-го шару [10].

Саме навчання мережі є основою її роботи і представляє собою ітеративний процес, як правило автоматичного (в перших перцептронах ручного) налаштування її параметрів ваг та зсувів з метою мінімізації помилки. Тобто нейронна мережа робить свій прогноз, після чого її прогноз порівнюється з реальною відповіддю і ваги та зсуви окремих нейронів змінюються так, щоб в наступний раз помилка стала меншою, після чого цикл замикається, а сам такий цикл називають ітерацією навчання. Розглянемо процес навчання детальніше.

На самому початку навчання всі ваги та зсуви всіх нейронів встановлюються випадковим чином в межах відносно невеликих реалістичних значень, що

необхідно для того щоб кожен нейрон адаптувався індивідуально під свої індивідуальні реалії, тобто зміг знайти свою спеціалізацію [26]. Надалі буде йтися про градієнтний спуск, який можна спрощено уявити як кульку на площі. Якщо ініціалізувати систему з однаковими значеннями без випадковостей, це можна уявити як спробу покласти кульку на плоску поверхню сподіваючись що вона скотиться до найнижчої точки поверхні, що не має сенсу. Якщо ж використати випадкову ініціалізацію, то це можна уявити як текстурну поверхню з виступами і низинами - в такому випадку куди б не була покладена кулька вона все одно скотиться в відносну найнижчу точку. У першому випадку, всі нейрони є копіями одне одного і в процесі навчання змінюються однаково, видаючи однакові виводи і однаково впливаючи на помилку, однаково навчаючись тощо, що еквівалентно використанню одного єдиного нейрона, що не є ефективним для більшості практичних завдань.

Далі відбувається прямий прохід, тобто вхідні дані подаються на вхідний шар і проходять через всі шари мережі до вихідного, отримуючи на виході вектор прогнозованих ймовірностей p залежно від задачі.

Розрахунок помилки, в якому прогноз p порівнюється з істинною міткою (заздалегідь відомою відповіддю) здійснюється за допомогою спеціальної функції втрат (Loss Function). У випадку задач багатолейблової класифікації, тобто таких де кожному зразку може відповідати одночасно декілька класів, зазвичай використовується бінарна перехресна ентропія (Binary Cross-Entropy Loss), яка обчислює помилку для кожного класу класифікації окремо (1.11):

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)] \quad (1.11)$$

;

де N - кількість класів,

y_i - істинна мітка (0 або 1) для і-го класу,

p_i - прогнозована ймовірність для і-го класу.

Чим сильніше p_i відрізняється від y_i, тим більше значення помилки L.

Наприклад, припустимо що маємо такі дані:

N=2; y_1=1; y_2=0; p_1=0.8; p_2=0.25;

Тоді помилка буде розрахована таким чином:

Розглянемо кожен клас окремо (кожен елемент суми Σ):

Для класу 1 (i=1):

$$y_1 = 1, p_1 = 0.8;$$

$$\text{компонент}_1 = y_1 \cdot \log(p_1) + (1-y_1) \cdot \log(1-p_1);$$

$$\text{компонент}_1 = 1 \cdot \log(0.8) + 0 \cdot \log(0.2) = \log(0.8) \approx -0.223;$$

Для класу 2 (i=2):

$$y_2 = 0, p_2 = 0.25;$$

$$\text{компонент}_2 = y_2 \cdot \log(p_2) + (1-y_2) \cdot \log(1-p_2);$$

$$\text{компонент}_2 = 0 \cdot \log(0.25) + 1 \cdot \log(0.75) = \log(0.75) \approx -0.288;$$

Тепер підставимо все у фінальну формулу (1.12):

$$L = -1/2 \cdot (\text{компонент}_1 + \text{компонент}_2); \qquad (1.12)$$

$$L = -0.5 \cdot (-0.223 + (-0.288)) = -0.5 \cdot (-0.511) \approx 0.2555;$$

Тобто помилка для даного прогнозу становить приблизно 0.2555. Саме це значення алгоритм зворотного поширення помилки буде використовувати для обчислення градієнтів.

Чим ближче помилка наближається до 0 тим ефективніше працює нейромережа, проте помилка рівно 0 може вважатися аномалією і помилкою в роботі мережі. Щоб помилка дорівнювала 0 потрібно щоб кожен компонент дорівнював строго 0, тобто прогнози ймовірностей повинні бути строго такими як і реальна відповідь. В нормі модель ніколи не поверне абсолютну 100% впевненість і точність у всіх своїх відповідях, тому що для цього логіти повинні б були прямувати до нескінченності, що не є обчислювально стабільним. Якщо подібне відбувається, то це є ознакою надпотужного перенавчання [6]. Така модель завчила навчальні дані ідеально, але не здатна до пошуку закономірностей - в реальних випадках вхідні дані, зазвичай, так чи інакше зашумлені, відповідно якщо модель видає 0% помилки це означає, що модель вивчила навіть не інформативні шуми (артефакти, помилки розмітки, зміни освітлення тощо). В такому випадку будь які нові дані, які хоч трохи відрізняються від завчених будуть класифіковані невірно. Додатково, функція логарифма log(x) прямує до мінус нескінченності, коли x прямує до нуля. Якщо модель помилково поверне ймовірність $p_1 = 0$ для класу, де істинна мітка $y_1 = 1$, функція втрат прийме надвеликі значення, що може призвести до математичних проблем під час навчання (отримуємо NaN або inf в градієнтах). Саме тому при розробці моделей зазвичай потрібно уникати ситуацій коли на виході отримуємо екстремальні значення 0.0 або 1.0.

Далі робиться зворотний прохід (Backpropagation) і обчислюється індивідуальний внесок для кожної ваги чи зсуву у загальну помилку L. Обчислення відбувається за допомогою градієнту, що являє собою вектор часткових похідних функцій втрат по відношенню до всіх параметрів мережі ∇L, тобто градієнт вказує напрям найшвидшого зростання функції втрат. Приклад щодо градієнту було наведено раніше. Кулька в тому прикладі - це поточний стан мережі з усіма її значеннями параметрів навчання. Площа в прикладі являє собою функцію втрат і є складною і багатовимірною, а не тривимірною як у спрощеному представленні [10].

Розглянемо градієнт (рис. 1.10) і його роботу докладніше. Він представляє вектор, що в кожній точці нашого багатовимірного ландшафту вказує на напрямок найкрутішого підйому. Це можна уявити як відчуття напрямку підйому за допомогою ступнів ніг стоячи на схилі гори.

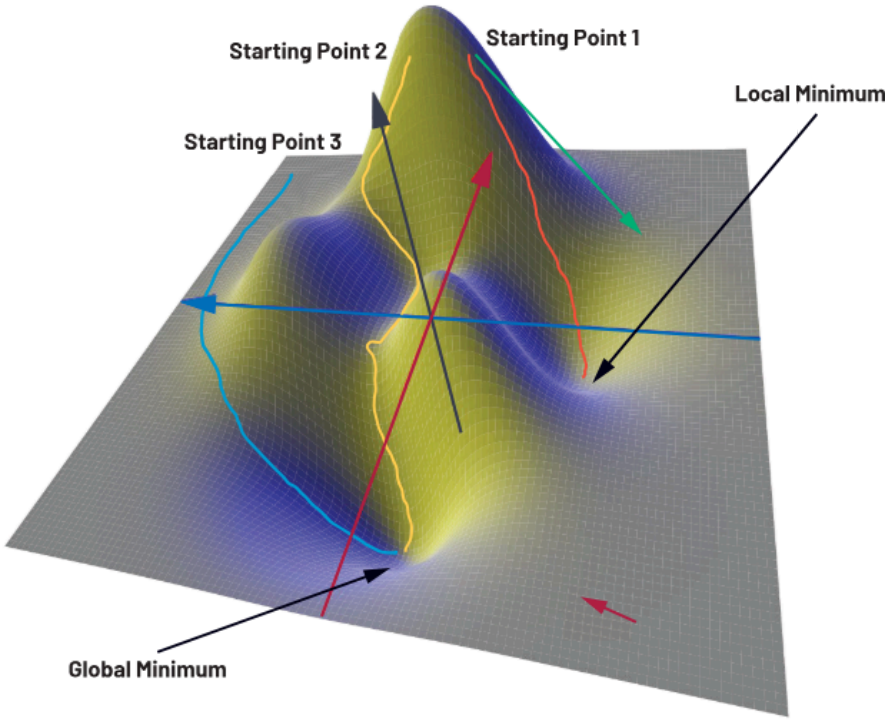


Рисунок 1.10 - Приклад спрощеної візуалізації градієнту

Градієнт складається з часткових похідних функції втрат відносно до кожного окремого параметру навчання θ (ваги w та зсуви b) (1.12):

$$\nabla L = \left[\frac{\partial L}{\partial \theta_1}, \frac{\partial L}{\partial \theta_2}, \ldots, \frac{\partial L}{\partial \theta_k} \right], \tag{1.12}$$

де кожна часткова похідна $\frac{\partial L}{\partial \theta_i}$ показує, наскільки сильно зміниться загальна помилка L при нескінченно малій зміні параметра θ_i.

Вочевидь, обчислити похідні напряму може бути складною задачею, через те що параметри перших шарів впливають на помилку через проміжні

обчислення, яких може бути велика кількість. Саме тому в машинному навчанні застосовують механізм зворотного поширення помилки, що працює за рахунок правила ланцюгової похідної (chain rule) і дозволяє ефективно обчислити градієнти [10]. Механізм називають зворотним через те, що обчислення починаються з вихідного шару і йдуть до вхідного, тобто в зворотному напрямку. Це робиться тому, що обчислення останнього шару відносно легкі адже його вплив на помилку прямий. Далі розраховується як передостанній шар вплинув на помилку враховуючи його вплив на останній шар за допомогою вже обчисленного градієнту. Таким чином алгоритм рекурсивно проходить по всім шарам мережі розраховуючи вплив параметрів кожного на фінальну помилку, звідки і пішла назва методу. Метод зворотного поширення помилки і досі є сучасним стандартом нейронних мереж на момент написання роботи через свою ефективність і обчислення всіх часткових похідних за один прохід по мережі.

Розглянемо приклад такого механізму:

Нехай помилка L залежить від виходу нейрона y, а y залежить від його ваги w (L(y(w))).

Щоб знайти, як зміна ваги w впливає на помилку L, потрібна похідна$\frac{\partial L}{\partial w}$;

За правилом ланцюгової похідної (1.13):

$$\frac{dL}{d\omega} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial \omega}; \qquad (1.13)$$

Тобто, внесок ваги w у загальну помилку дорівнює добутку внеску виходу нейрона y у помилку L на внесок самої ваги w у вихід y. Саме такий принцип, застосований до всієї мережі, і є суттю зворотного поширення помилки.

Після цього відбувається оновлення параметрів мережі. Вочевидь, для зменшення помилки потрібно рухатись у напрямку протилежному градієнту (адже він вказує на напрямок зростання помилки). Таке оновлення параметрів називають

градієнтним спуском (Gradient Descent). При цьому кожен параметр (ваги або зсуви) оновлюються як (1.14):

$$\nabla L\theta_{new} = \theta_{old} - \eta \cdot \nabla_{\theta} L \quad ; \qquad (1.14)$$

де η (ета) - це швидкість навчання (learning rate), що являє собою розмір кроку оновлення і може бути як заданою константою так і динамічною змінною в більш сучасних архітектурах, на приклад за допомогою алгоритмів оптимізаторів, золотим стандартом яких є оптимізатор Adam що автоматично на основі пам'яті про попередні стани підлаштовує швидкість навчання для найбільш ефективного пошуку мінімуму прискорюючи на "крутих схилах" і сповільнюючи у "пласких долинах" [31]. Швидкість навчання не повинна бути надто високою - тоді модель ніколи не зможе зійтись в оптимальному мінімумі і буде постійно "перестрибувати" його, і не може бути надто малою - тоді модель буде сходитись до мінімуму надто довго, витрачаючи зайві обчислювальні ресурси.

Наведемо приклад оновлення одного параметра:

Припустимо, для певної ваги w на поточній ітерації маємо:

Поточне значення ваги: $w_{old} = 0.8$

Розрахований градієнт по цій вазі: ∇w L = 1.2 (позитивний градієнт означає, що збільшення ваги призведе до збільшення помилки, отже, вагу потрібно зменшити).

Швидкість навчання: $\eta = 0.01$

Тоді нове значення ваги буде:

$$w_{new} = w_{old} - \eta \cdot \nabla \text{ w L};$$

$$w_{new} = 0.8 - 0.01 \cdot 1.2 = 0.8 - 0.012 = 0.788.$$

Як бачимо, нова вага w_{new} стала меншою, що є вірним кроком у напрямку зменшення загальної помилки.

Також варто розуміти, що ефективність градієнтного спуску залежить від величини (амплітуди) обчислених градієнтів. У глибоких нейронних мережах з багатьох шарів часто виникає проблема згасання градієнта (vanishing gradient). Коли градієнт має великі значення (амплітуду), то зміни ваг відбуваються відносно великими, помітними кроками, відповідно і фактична швидкість навчання у часі вища. Але під час процесу зворотного поширення помилки, градієнт для кожного шару обчислюється на базі градієнта наступного шару як ланцюгова похідна. Таким чином, градієнт для перших шарів мережі є добутком багатьох похідних від функцій активації всіх наступних шарів і якщо ці похідні будуть числами, що менші за одиницю (особливо характерно для сигмоїдальної функції), то у разі перемноження багатьох таких чисел кінцевий градієнт стане експоненційно малим і наближеним до нуля [26].

Для ілюстрації цього ефекту розглянемо спрощену мережу з 4 шарів, де кожен шар має один нейрон і використовує сигмоїдальну функцію активації σ(z). Похідна сигмоїдальної функції має максимальне значення 0.25 (у точці z=0).
Нехай L - функція втрат, а w_1, w_2, w_3, w_4 - ваги нейронів у шарах 1, 2, 3, 4 відповідно. За правилом ланцюгової похідної (1.13), градієнт для ваги першого шару w_1 буде виглядати так (1.15):

$$\frac{dL}{dw_1} = \frac{\partial L}{\partial a_4} \cdot \frac{\partial a_4}{\partial z_4} \cdot \frac{\partial z_4}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_1}; \qquad (1.15)$$

де a_i - вихід і-го шару, z_i - зважена сума. Ключовими компонентами тут є похідні функції активації $\frac{\partial a}{\partial z}$ = σ'(z).

Припустимо, що для стабільного навчання всі ваги близькі до 1, а похідні функції активації приймають своє максимальне значення 0.25. Тоді градієнт для першого шару буде пропорційний добутку похідних (1.16):

$$\nabla L_{w1} \propto \sigma'(z_4) \cdot \sigma'(z_3) \cdot \sigma'(z_2) \; ; \qquad (1.16)$$

Підставимо числові значення:

$$\nabla L_{w1} \propto 0.25 \times 0.25 \times 0.25 = 0.015625 \; .$$

Як бачимо, навіть у ідеалізованому сценарії градієнт для першого шару зменшився у $\frac{1}{0.015625} = 64$ рази порівняно з градієнтом останнього шару. У реальному випадку значно глибшої мережі, де похідні часто бувають ще меншими, цей ефект стає катастрофічним, а градієнт для перших шарів може легко досягти значень близько 10^{-10} або менше, що є обчислювальним нулем для оновлення ваг.

Як наслідок, відбудеться зупинка навчання перших шарів, адже оновлення ваг прямо пропорційне градієнту. Майже нульовий градієнт означає майже нульове оновлення ваг. Відповідно перші шари такої мережі майже не відійдуть від своїх початкових випадкових значень, нічому не навчаться і будуть, фактично, зайвими, при цьому все одно використовуючи додаткові обчислювальні ресурси. Це тягне за собою наступну проблему - саме перші шари відповідають за розпізнавання базових, низькорівневих ознак, таких як краї та градієнти. Відповідно, якщо вони не навчаються, то вся мережа майже не здатна побудувати ієрархію ознак, а це, в свою чергу, призводить до різкого зниження ефективності.

Існує також і прямо протилежна, проте менш поширена проблема, яку називають вибухом градієнта (exploding gradient). Коли похідні функції втрат по відношенню до ваг є стабільно більшими за одиницю, при використанні ланцюгового правила, множення таких значень призводить до експоненційного зростання величини градієнта. Відбувається чисельний вибух і як наслідок кроки оновлення вагових коефіцієнтів під час градієнтного спуску стають не адекватно великими. Це призводить до викидання моделі з області оптимальних значень параметрів і тотальної розбіжності у процесі навчання, але на практиці це частіше

проявляється як поява значень NaN (Not a Number) у функції втрат, що свідчить про арифметичне переповнення і призводить до помилки виконання програми. Один з ефективних методів боротьби з вибухами градієнту є просте обрізання градієнта (Gradient Clipping), при якому максимальне значення градієнта примусово обмежується певним порогом [26].

Саме проблеми з градієнтом виступають головним обмеженням, що заважає простому нарощуванню кількості шарів. Для їх подолання було розроблено низку ефективних методів та архітектурних рішень. Ці проблеми є актуальними, але вже існують деякі методи боротьби з ними. Наприклад, функції активації ReLU та її варіації для яких, як було зазначено, похідна стандартної функції ReLU дорівнює 1 для всіх додатних значень, що запобігає множенню градієнта на числа, менші за одиницю. Проте така функція має недолік, який називають “вмираючим ReLU”. Його суть полягає у тому, що у випадку якщо нейрон видасть від’ємне значення, його градієнт стане нулем і може перестати оновлюватися, тому існують модифіковані версії ReLU, такі як протікаючий (Leaky ReLU), що має невеликий фіксований нахил для від'ємних значень, та параметричний Parametric ReLU (PReLU), де цей нахил є параметром, що навчається [27].

Також використовуються залишкові зв’язки (Residual Connections). Наприклад, архітектура ResNet (Residual Network) створила революційне рішення у вигляді залишкових блоків. Ключова ідея залишкових блоків полягає у тому, щоб додати короткий шлях (shortcut connection), що буде передавати вхідний сигнал в обхід одного або декількох шарів і додасть його до їхнього виходу. Таким чином, шари навчаються не безпосередньо новому представленню H(x), а залишковій функції [19] (1.17)

$$F(x) = H(x) - x; \tag{1.17}$$

що дозволяє градієнтам поширюватися назад через ці коротки шляхи, ефективно долаючи згасання навіть у надглибоких мережах, що налічують сотні шарів.

Також можна використати нормалізацію по батчу (Batch Normalization), суть якого полягає в тому щоб стабілізувати процес навчання через нормалізацію виходів попереднього шару перед подачею їх на наступний шар. Це є стандартною практикою, адже для кожного такого міні-батчу даних обчислюється середнє значення і стандартне відхилення, і дані перетворюються так, щоб мати нульове середнє та одиничну дисперсію, що допомогає утримати входи функцій активації у їхній не насиченій області, в якій градієнти стабільні і не нульові, що значно прискорює навчання і знижує чутливість до ініціалізації ваг [28].

Нарешті можна використовувати альтернативні методи випадкової ініціалізації ваг, використовуючи методи, такі як Xavier або Glorot. Замість повної випадковості (псевдовипадковості у контексті обчислення компьютерами), вони встановлюють початкові ваги нейронів з урахуванням кількості вхідних та вихідних нейронів шару. Це дозволяє підтримувати дисперсію сигналу на стабільному рівні при його поширенні вглиб мережи та запобігає ранньому виникненню проблем із згасанням чи вибухом градієнтів [26].

Кроки починаючи з прямого проходу і до зворотного проходу повторюються для всіх (зазвичай тисяч і більше) прикладів з навчальної вибірки протягом багатьох епох (повних проходів по всій навчальній виборці) до тих пір поки помилка на валідаційний вибірці (тій на якій модель не була тренована, тобто можна сказати екзаменаційній, і не знає вірної відповіді) моделі не перестане відчутно зменшуватися і не вийде на плато. Проте проведення повного процесу навчання одразу на всій навчальній вибірці часто є обчислювально неефективним, адже часто навчальна вибірка може мати в собі більше десятків тисяч об’єктів. Тому процес навчання проходить з розбиттям на батчі - підвибірки навчальної вибірки, в які попадає відносно невелика кількість об’єктів (зазвичай 32 або 64). В рамках однієї епохи оброблюється кількість батчів що дорівнює (1.18):

$$N=\frac{X}{Y}; \qquad (1.18)$$

де X - загальна кількість об’єктів в навчальній вибірці;

У - встановлений розмір одного батчу.

Наприклад, якщо навчальна вибірка має в собі 5000 елементів, а розмір батчу встановлений як 32, то всього батчів на одну епоху буде:

$$5000/32=156.25.$$

Залишок в цьому випадку формує собою не повний батч (не 32 елементи, а 8) або відкидається.

1.4 Архітектура та принципи роботи згорткових нейронних мереж (CNN)

У вступі до розділу вже було згадано те, що повнозв’язні архітектури нейронних мереж, описані у попередньому розділі, не є ефективними для задач аналізу зображень через надмірну кількість параметрів навчання, що потребує значних обчислювальних ресурсів, та ігнорування просторової структури даних, що не дозволяє розрізняти позиційно залежні патерни і текстури. З метою вирішення цих фундаментальних проблем були розроблені **згорткові нейронні мережі (CNN) [11].**

Архітектура таких мереж заснована на двох ключових принципах, що були натхнені принципами роботи зорової кори головного мозку, тобто спільному використанні параметрів та локальному сприйнятті полів [6, 8].

Робота шарів в таких мережах дещо відрізняється як і самі шари:

Згортковий шар (Convolutional Layer) є основним елементом CNN і на відміну від повнозв’язного шару, в якому кожен нейрон аналізує всі вхідні дані одночасно, нейрони в згортковому шарі являють собою детектори локальних ознак. Кожен нейрон згорткового шару поєднаний не з кожним пікселем зображення, а лише з його відносно невеликою піксельною прямокутною областю. Така область називається рецептивним полем (receptive field). Якщо повнозв’язна

мережа аналізує все зображення одночасно то згортковий шар фокусує увагу на певних його частинах. Набір ваг, який відповідає з’єднанням рецептивного поля називається фільтром (filter) або ядром (kernel) і має відносно невеликі розміри (наприклад, 3x3, 5x5 пікселів), проте його глибина завжди дорівнює глибині (кількості каналів) вхідного зображення для першого згорткового шару і кількості фільтрів попереднього для наступного шару. Відповідно для фільтру 3x3 на RGB-зображенні фільтр матиме розмірність (кількість параметрів) 3x3x3=27 [6].

Цікаво, що хоча фільтр має фіксований розмір (наприклад, 3x3), його ефективне рецептивне поле - тобто область на вхідному зображенні, яка впливає на один нейрон на поточній карті ознак - зростає з кожним наступним шаром. Нейрон на другому шарі, дивлячись на ділянку 3x3 на попередній карті, опосередковано розглядає вже ділянку 5x5 на оригінальному зображенні і тд. Це дозволяє мережі поступово переходити від виявлення локальних текстур до розпізнавання глобальних об'єктів [16].

Суть роботи шару полягає в тому, що кожний фільтр шару ковзає по всьому вхідному зображенню з певним кроком (stride) і обчислює поелементний добуток між вагами фільтра та значеннями пікселів у відповідному рецептивному полі у кожній позиції фільтру (рис. 1.11) [6].

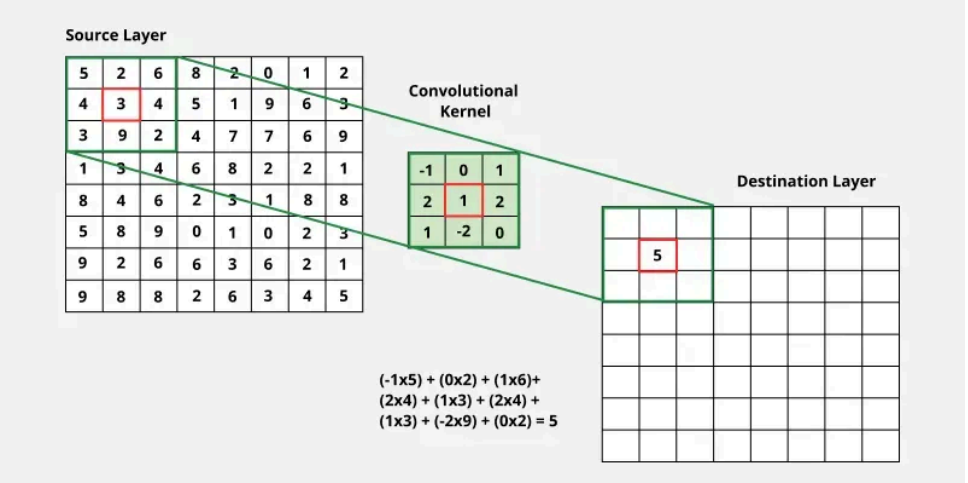


Рисунок 1.11 - Приклад роботи фільтру

Цей процес, який називають, власне, операція згортки, він схожий на “згортання” області ознак в одну оцінку і математично можна виразити наступною формулою для одного каналу (1.19):

$$Y_{ij} = f\left(\sum_{u=0}^{k-1}\sum_{v=0}^{k-1} W_{uv} \cdot X_{i\cdot S+u, j\cdot S+v} + b\right); \quad (1.19)$$

де Y_{ij}- значення вихідної карти ознак в позиції (i, j);

f - нелінійна функція активації (зазвичай ReLU), що застосовується до результату;

W_{uv} - ваги фільтра (ядра) розміром $k \times k$;

X - вхідні дані (ділянка вхідного зображення або попередньої карти ознак);

b - параметр зсуву (bias), єдиний для всього фільтра;

S - крок (stride), що визначає, на скільки пікселів фільтр зсувається за один раз.

Як результат, отримаємо одне число, що буде надалі записано у вихідну карту ознак (feature map). Така карта показує в яких областях зображення було знайдено патерн, на розпізнавання якого навчений даний фільтр.

Наприклад, уявімо вхідну матрицю 4x4 та фільтр 3x3 з кроком S=1, тоді:

$$X = \begin{pmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 1 \\ 7 & 8 & 9 & 2 \\ 1 & 0 & 2 & 3 \end{pmatrix};$$

$$W = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix};$$

$$b = 0;$$

Перший елемент вихідної карти буде розраховано як:

$$Y_{00} = (1 \cdot 1+2 \cdot 0+3 \cdot 1)+(4 \cdot 0+5 \cdot 1+6 \cdot 0)+(7 \cdot 1+8 \cdot 0+9 \cdot 1)+0=1+3+5+7+9=25;$$

Після застосування активації ReLU f(x)=max(0,x), результат залишиться 25 оскільки 25 є додатним числом. В такому вигляді результат і буде записано у перший елемент вихідної карти ознак цього фільтру. Далі фільтр зсунеться на 1 піксель праворуч, і процес повториться:

$$X_1 = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \rightarrow X_2 = \begin{pmatrix} 2 & 3 & 0 \\ 5 & 6 & 1 \\ 8 & 9 & 2 \end{pmatrix};$$

$$Y_{01} = (2 \cdot 1+3 \cdot 0+0 \cdot 1)+(5 \cdot 0+6 \cdot 1+1 \cdot 0)+(8 \cdot 1+9 \cdot 0+2 \cdot 1)+0=2+6+10=18;$$

Після застосування активації ReLU f(x)=max(0,x), результат залишиться 18 оскільки 18 є додатним числом. 18 запишеться у другий елемент карти. Якщо ж на якомусь з етапів отримаємо негативний результат, то після застосування активації ReLU він буде перезаписаний як 0 і буде інтерпретований як те, що фільтр не знайшов ознаку. Далі подібним чином фільтр проходить по всьому зображенню і формує матрицю значень, яка і називається карта ознак (feature map) і візуалізує в яких саме ділянках зображення наскільки сильно було активовано фільтр.

Проте при такому підході виникає проблема кількості використаних крайових значень. Коли піксель знаходиться всередині зображення, то, вочевидь, фільтр ковзне по ньому всіма своїми елементами рецептивного поля максимальну кількість разів. Якщо фільтр 3x3, то для серединного пікселя кожен елемент матриці провзаємодіє з ним один раз, відповідно піксель буде враховано 9 разів. Якщо ж піксель знаходиться на краю зображення, то він буде врахований лише 3 рази для 3х бокових значень матриці фільтру відповідно краю. У випадку ж кутових пікселів, вони будуть затронуті лише один раз крайніми значеннями фільтру. Таким чином бокові, а особливо кутові, пікселі менше впливають на

навчання фільтра, хоча навіть краї і кути зображень також можуть нести інформаційне навантаження для мережі.

Для боротьби з таким ефектом неуваги до крайових значень застосовується додатковий гіперпараметр доповнення (padding). Суть його роботи доволі проста і полягає у додаванні нульових пікселів (пікселів з однаковими, часто нульовими значеннями що не несуть в собі корисної інформації) по периметру вхідних даних (рис. 1.12). Існують також і інші варіанти додавання пікселів, такі як дзеркальні пікселі, доповнення копіюванням країв, циклічне доповнення, кожен з яких має специфічні застосування проте в роботі буде розглянуто саме доповнення нулями. Кількість пікселів, що додаються з одного боку, позначається як P. Відповідно, наприклад якщо P=1 для ширини зображення, то з лівого та правого краю зображення буде додано один нульовий піксель, тобто всього пікселів буде додано 2P. Це дозволяє центру фільтра відвідати кожен піксель вихідного зображення, забезпечуючи їх більш повну участь у формуванні вихідної карти ознак [6]. Кількість доданих пікселів менша і не дорівнює розмірності фільтру через те, що нульові пікселі не несуть інформації для мережі, тому якщо кількість доповнених пікселів з краю буде така ж як і розмір фільтра або більше, на певному етапі фільтр буде повністю поміщений у пікселі доповнення, що не має сенсу і додає зайвий крок розрахункового навантаження.

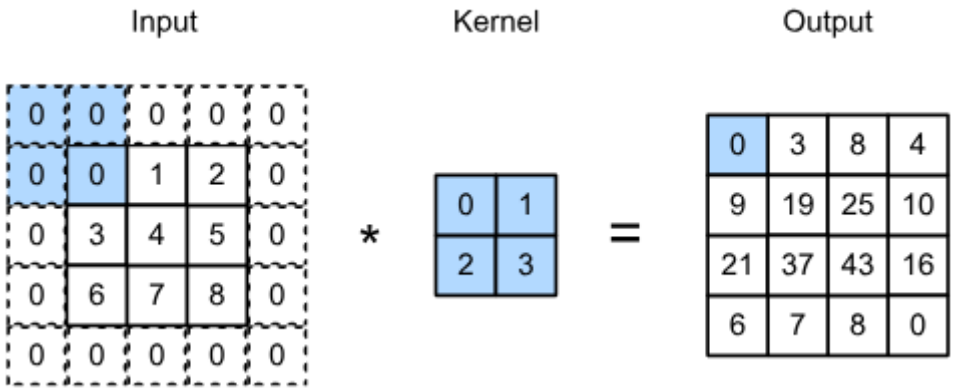


Рисунок 1.12 - Приклад застосування padding

Доповнення також використовується для збереження розмірів зображення. Без доповнення кожна операція згортки зменшує висоту та ширину карти ознак. Розрахувати розмірність вихідної карти ознак можна за формулою (1.20):

$$W_{out} = \frac{W_{in} - K + 2P}{S} + 1 \quad (1.20)$$

де W_{out}- ширина/висота виходу (розраховується окремо для зображень з несиметричними шириною та висотою);

W_{in}- ширина/висота входу;

K - розмір фільтра;

P - доповнення;

S - крок.

Наприклад, застосування фільтра 3x3 до зображення 28x28 з кроком S=1 і доповненням P=1 піксель зі всіх сторін:

$$W_{out} = \frac{28 - 3 + 2 * 1}{1} + 1 = 28;$$

дасть на виході карту розміром 28x28. Таким чином можна нівелювати проблему зменшення розмірності для згорткових шарів та зменшити проблему не уваги для крайових значень, що виникає, якщо встановити доповнення на 0:

$$W_{out} = \frac{28 - 3 + 2 * 0}{1} + 1 = 26.$$

Отримали зображення 26x26. Тобто без доповнення вихідна карта ознак поступово “згортається” з кожним наступним шаром. Це означає, що в глибоких мережах з великою кількістю шарів згортки, для зображення 28x28 достатньо

лише 14 шарів щоб зображення стало 0x0, що, вочевидь, неприпустимо адже ще до повної втрати зображення карта ознак стане замала для адекватної класифікації і обробки (катастрофічні помилки виконання можуть виникнути ще на 13 шарі, адже він матиме розмірність 2x2, а фільтр у прикладі 3x3) [14]. Часто, в реальних задачах, зображення можуть бути завеликі і вимагати відповідних обчислювальних ресурсів, проте, як показує практика, робити інформовані передбачення можна навіть на відносно невеликих зображеннях (часто 128x128, 256x256 або 512x512), особливо для відносно простих задач. В такому випадку зображення або приводяться до потрібної розмірності сторонніми методами заздалегідь, або не використовується доповнення з ціллю зменшення зображення у перших декількох шарах.

Крок фільтру є ключовим гіперпараметром. При S=1 фільтр рухається на один піксель вправо (якщо досягає краю зображення то повертається на початок поточного проходу але зміщується на 1 піксель вниз), зберігаючи високу деталізацію та розмір карти ознак зменшується незначно, а при S=2 розмір вихідної карти зменшується приблизно вдвічі, що дозволяє знизити обчислювальну складність і узагальнити ознаки:

$$W_{out} = \frac{28 - 3 + 2 * 1}{2} + 1 = 14,5$$

.

Допитливий читач може помітити парадокс використання доповнення. У нашому прикладі, використали доповнення P=1, щоб зберегти розміри зображення і побудувати інформативну карту ознак, але у такому випадку частково втрачаємо увагу до крайових значень через те, що фільтр 3x3 зможе врахувати крайові пікселі тільки відносно центру своєї матриці. Тобто додаючи лише 1 піксель отримуємо ситуацію, де фільтр, наприклад для крайнього лівого пікселя, зможе врахувати його тільки своїми крайніми лівими, та середніми значеннями але не крайніми правими. Таким чином, хоча і зменшили проблему уваги, але не позбулися неї. З іншого боку, якщо встановимо значення доповнення як P=2, то

буде додано по 2 пікселі і, у нашому прикладі, фільтр зможе повністю всіма своїми елементами врахувати крайові пікселі. Але у такому випадку виникає інша проблема:

$$W_{out} = \frac{28 - 3 + 2 * 2}{1} + 1 = 30 ;$$

ми отримуємо вихідну карту ознак розмірами 30x30 збільшуючи її. Це потребує більше обчислювальних ресурсів і може призвести до кумулятивного збільшення потреби в них з кожним наступним шаром. Виникає питання - що важливіше, абсолютна увага, або обчислювальна ефективність? Вочевидь, можна спробувати використати доповнення P=2 на одному шарі, а на наступному використати P=0 і чергувати їх таким чином отримуючи абсолютну увагу і стабільні вимоги до ресурсів.

Така ідея може виглядати логічною, але є хибною з оглядом на мету побудови згорткових мереж. Мета CNN це утворення ієрархії ознак переходячи від простих, локальних патернів до загальних абстрактних концепцій. Перші згорткові шари вчаться бачити базові елементи зображення - горизонтальні та вертикальні лінії, градієнти, кольорові плями. Середні шари комбінують елементарні елементи, щоб навчитися розпізнавати більш складні форми - контури ока, лінію носа, форму губ. На цьому етапі точне положення певної вертикальної лінії вже не так важливе, як її наявність у складі загальної структури ока. Глибокі шари комбінують очі, ніс та губи в єдину концепцію - обличчя. На цьому рівні для мережі вже байдуже, чи було око в пікселі (50, 60) чи (52, 62) - важливо лиш те, що два ока, ніс і рот знаходяться у правильному відносному положенні [16, 25].

Для досягнення такого ефекту потрібне поступове зменшення просторової розмірності (даунсемплінг). Зменшуючи карту ознак (за допомогою stride > 1 або, що частіше, шарів підвибірки (pooling), про які буде сказано далі), змушуємо

мережу узагальнювати інформацію і позбутися зайвої інформації, концентруючись на наявності самої ознаки а не її точних параметрів розташування.

Таким чином, підхід з постійним збереженням високої роздільної здатності (30x30) не тільки неефективний обчислювально, але й шкідливий для навчання, оскільки він заважає мережі будувати абстракції. Стандартний підхід з використанням padding P=1, або таким що гарантує що центр фільтру використає крайні пікселі але не весь фільтр, є загальноприйнятим оптимальним компромісом: він достатньо добре обробляє краї на ранніх етапах, зберігаючи стабільність архітектури, і дозволяє мережі ефективно зменшувати розмірність на глибших шарах для побудови якісних узагальнень.

У побудові згорткових нейромереж доповнення часто апроксимують у три основні режими роботи, суть яких була розглянута раніше: Valid (валідне) - режим без доповнення P=0; Same (таке саме) - P розраховується автоматично так, щоб розмір вхідного і вихідного зображення збігалися; Full (повне) - доповнення додається таким чином, щоб фільтр міг вийти за межі зображення доти, доки хоча б один піксель фільтра перекриває піксель зображення.

Терміни вихідне зображення та вихідна карта ознак у тексті часто використані як взаємозамінні. Не складно зрозуміти, що отримані значення у вигляді матриці після згорткового фільтру можна представити як зображення, де кожен піксель відповідає позиції числа в матриці, а його значення можна, наприклад, представити як яскравість. Такі методи називаються візуалізацією карт активації, серед них є наприклад Grad-CAM і вони дозволяють проводити глибший аналіз роботи кожного фільтру або декількох в мережі і знаходити, наприклад зайві не активні фільтри, аномалії, або формувати додаткові карти “уваги” які допомагають візуально оцінити на яких ділянках зображення фокусується нейромережа.

Щодо останнього, то існують просунуті методи інтерпретації, такі як Grad-CAM. Вони дозволяють візуалізувати, на які саме області вхідного зображення спиралась вся нейронна мережа для прийняття фінального рішення (наприклад, для класифікації об'єкта) і формують так звані “карти уваги”, що

допомагають оцінити, чи фокусується модель на релевантних ознаках (ніс, вуха, тулуб у випадку класифікації котів наприклад) [36].

Ключова економія параметрів, на відміну від повнозв’язних мереж, в згорткових мережах полягає в тому, що один і той самий фільтр з одним і тим самим набором параметрів використовується для аналізу всього зображення. Це має сенс тому що такий фільтр може навчитися розпізнавати, наприклад, вертикальні лінії або специфічні текстури поверхні очного дна, що можуть бути в різних частинах зображення, але їх важливість майже однакова в усіх частинах зображення, тому не потрібно аналізувати кожен елемент окремо (майже, тому що для згорткової мережі і відносно простих задач таке припущення є достатнім, але існують більш складні архітектури, що враховують окремий вплив кожного знайденого патерну, наприклад, механізми уваги (Attention Mechanisms)) [6]. Таким чином згорткові мережі використовують значно менше обчислювальних ресурсів маючи в собі менше параметрів. Наприклад, замість мільйонів параметрів для повнозв’язного шару, один фільтр розміром 3x3x3 має лише 3·3·3 + 1 = 28 параметрів (включаючи один зсув). Такий принцип називається спільним використанням параметрів.

Як вже було вказано, кожен фільтр навчається розпізнавати свою певну ознаку на зображенні, саме тому в згорткових нейронних мережах зазвичай використовується декілька фільтрів (часто 32, 64, 128 тощо), де кожен в процесі навчання спеціалізується на розпізнавані своєї унікальної ознаки зображення. Для підвищення ефективності, перші шари навчаються знаходити прості патерни (краї, градієнти, плями тощо), наступні шари отримують на вхід карти ознак від попередніх і навчаються комбінувати прості елементи у складні структури (текстури, прості форми), і так далі, формуючи ієрархію візуальних представлень [16]. Такий принцип називають ієрархією ознак. Теоретично, створюючи нескінченно глибоку мережу можна розпізнавати нескінчено комплексні та різноманітні патерни, що розпізнають цілі об’єкти в більш глибоких шарах, проте на практиці, настільки глибокі мережі не мають сенсу, адже виникають проблеми згасання градієнту і часто розпізнати об’єкт можна значно раніше за ключовими

ознаками [18]. Також, якщо завчасно відомо, що ключову інформаційну роль в розпізнаванні мають саме форми і прості градієнти та текстури ознак, а не колір, то у таких задачах має сенс використовувати монохромну кольорову палітру у зображенні, що надалі зменшить кількість параметрів, а відповідно потребу в обчислювальних ресурсах.

Процес навчання шарів згортки дещо відрізняється від повнозв'язного. Він базується на стандартному алгоритмі зворотного поширення помилки, але має особливості пов'язані з принципом спільного використання параметрів. Сенс в тому, що одна і та сама вага фільтра бере участь у формуванні багатьох значень на вихідній карті ознак, на відміну від повнозв'язніх шарів де кожен нейрон активується лише один раз за прохід. Через це, градієнт функції втрат L по відношенню до конкретної ваги фільтра w_{uv} розраховується як сума часткових градієнтів з усіх позицій (i,j), де був застосований цей фільтр (1.21):

$$\frac{\partial L}{\partial w_{uv}} = \sum_i \sum_j \frac{\partial L}{\partial Y_{ij}} \frac{\partial Y_{ij}}{\partial w_{uv}} ; \qquad (1.21)$$

де $\frac{\partial L}{\partial w_{uv}}$ - повний градієнт помилки відносно ваги w_{uv} у фільтрі;

$\frac{\partial L}{\partial Y_{ij}}$ - градієнт, що прийшов на вихідну карту ознак Y з наступного шару;

$\frac{\partial Y_{ij}}{\partial w_{uv}}$ - локальний градієнт, що показує, як зміна ваги w_{uv} впливає на вихідний елемент Y_{ij}.

З формули згортки знаємо, що цей градієнт дорівнює відповідному пікселю вхідних даних X, з яким взаємодіяла вага.

Уявімо мінімалістичний сценарій. Нехай у нас є вхідна матриця X 3x3 та фільтр W 2x2 (без зсуву та функції активації для простоти), тоді:

Прямий прохід:

$$X = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad W = \begin{pmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix};$$

Вихідна карта ознак Y буде розміром 2x2:

$$Y_{00} = (1 \cdot 1 + 2 \cdot 0) + (4 \cdot 2 + 5 \cdot 1) = 1 + 0 + 8 + 5 = 14;$$

$$Y_{01} = (2 \cdot 1 + 3 \cdot 0) + (5 \cdot 2 + 6 \cdot 1) = 2 + 0 + 10 + 6 = 18;$$

$$Y_{10} = (4 \cdot 1 + 5 \cdot 0) + (7 \cdot 2 + 8 \cdot 1) = 4 + 0 + 14 + 8 = 26;$$

$$Y_{11} = (5 \cdot 1 + 6 \cdot 0) + (8 \cdot 2 + 9 \cdot 1) = 5 + 0 + 16 + 9 = 30;$$

Зворотний прохід:

Припустимо, що з наступних шарів отримали градієнт помилки $\frac{\partial L}{\partial Y}$ для вихідної карти Y:

$$\frac{\partial L}{\partial Y} = \begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{pmatrix};$$

Розрахуємо градієнт для ваги фільтра. Візьмемо, наприклад, вагу w_{00}:

Частковий внесок в Y_{00}: вага w_{00} множилася на піксель $X_{00} = 1$.

Внесок у градієнт: $\frac{\partial L}{\partial Y_{00}} \cdot X_{00} = 0.1 \cdot 1 = 0.1$;

Частковий внесок в Y_{01}: вага w_{00} множилася на піксель $X_{01} = 1$.

Внесок у градієнт: $\frac{\partial L}{\partial Y_{01}} \cdot X_{01} = 0.2 \cdot 2 = 0.4$;

Частковий внесок в Y_{10}: вага w_{00} множилася на піксель $X_{10} = 1$.

Внесок у градієнт: $\frac{\partial L}{\partial Y_{10}} \cdot X_{10} = 0.3 \cdot 4 = 1.2$;

Частковий внесок в Y_{11}: вага w_{00} множилася на піксель $X_{11} = 1$.

Внесок у градієнт: $\frac{\partial L}{\partial Y_{11}} \cdot X_{11} = 0.4 \cdot 5 = 2.0$;

Сумарний градієнт для ваги w_{00} , буде сумою всіх цих внесків:

$$\frac{\partial L}{\partial w_{00}} = 0.1 + 0.4 + 1.2 + 2.0 = 3.7$$;

Якщо швидкість навчання η=0.01, то нова вага w_{00} буде:

$$w_{00\,new} = w_{00\,old} - \eta \cdot \frac{\partial L}{\partial w_{00}} = 1 - 0.01 \cdot 3.7 = 1 - 0.037 = 0.963;$$

Такий додатковий механізм агрегації помилок з усього зображення дозволяє фільтрам навчатися виявляти універсальні та позиційно-інваріантні патерни [11, 6].

Окрім стандартної операції згортки, в сучасних архітектурах існують інші її модифікації для вирішення певних специфічних завдань. Наприклад, розширена згортка (Dilated Convolution) дозволяє збільшити рецептивне поле без збільшення кількості параметрів, що корисно в задачах сегментації. Транспонована згортка (Transposed Convolution) використовується для збільшення просторової розмірності (апсемплінгу), що є ключовим у генеративних мережах (GAN) та автоенкодерах. Роздільна згортка (Depthwise Separable Convolution), популяризована в архітектурах MobileNet, значно зменшує кількість обчислень, що дозволяє створювати ефективні моделі для мобільних пристроїв. Проте в роботі розглядається здебільшого класичний варіант згортки.

Серед специфічних застосувань фільтрів згортки варто згадати застосування фільтрів розміром 1x1. Раніше в тексті вже було згадано стискання глибини (кількості каналів) зображення задля економії обчислювальних ресурсів в задачах які не потребують інформації про колір. Фільтри 1x1 працюють подібним, але

більш узагальненим чином. Кожен застосований фільтр згорткового шару створює свою карту ознак і, відповідно, додає глибину (канал) для наступного шару. Кожен шар може мати сотні і більше фільтрів, через що кожен наступний згортковий шар змушений витрачати додаткові ресурси. Наприклад, якщо у попередньому шарі було 100 фільтрів, а в поточному використовується 1 фільтр розміром 3x3, то один лише такий фільтр матиме 3x3x100=900 параметрів для навчання, а якщо таких фільтрів буде припустимо, також 100, то це вже 90000 параметрів для навчання. Часто виникають ситуації коли частина фільтрів реагує на схожі ознаки, створюючи надлишковість каналів, наприклад, один фільтр реагує на червоні вертикальні лінії, а другий на яскраво помаранчеві вертикальні лінії, в такому випадку може мати сенс апроксимувати такі дві карти ознак в одну, адже можна припустити, що такі лінії часто є частинами одного контуру. Саме для таких задач можна застосувати фільтр розміром 1x1. Він працює як навчений компресор інформації по глибині каналів. Не зачіпаючи просторову структуру, він навчається комбінувати існуючі канали у меншу кількість нових, більш інформативних. Принцип його роботи можна порівняти з алгоритмами стиснення зображень (наприклад JPEG) - часто можна апроксимувати, наприклад, градієнти на зображеннях зменшивши кількість кольорів, що призведе до “сходинок” градієнтів, але значно зменшить вагу зображення без значної втрати інформації. Проте фільтри 1x1 стискають не просторові, а глибинні ознаки (канали), комбінуючі схожі між собою карти ознак. Кількість каналів на виході згорткового шару буде дорівнювати кількості використаних 1x1 фільтрів у ньому. Такий підхід дозволяє створювати “вузькі місця” (bottlenecks) в архітектурі, що може бути ключовим для оптимізації [17]. Замість прямої обробки великої кількості каналів дорогим 3x3 ,або навіть більшим, фільтром, можемо спочатку стиснути їх за допомогою 1x1 фільтрів. Уявімо, що у нашому прикладі використали 25 фільтрів розміром 1x1, тоді зі 100 каналів отримаємо 25, відповідно в наступному шарі з 100 фільтрами розміром 3x3 буде використано (3x3x25)x100=225x100=22500 параметрів, що відповідно потребує в 4 рази менших обчислювальних ресурсів. Такі згорткові шари-компресори дозволяють створювати набагато більш ефективні

і швидкодійні мережі з відносно невеликими втратами точності. Навпаки, інтуїтивно зрозуміла логіка, що полягає в тому, що якщо шари з фільтрами 1x1 стискають інформацію, по суті втрачаючи її, призводить до погіршення результатів роботи моделі часто виявляється хибною. Такі фільтри можуть слугувати як регуляризатор. Регуляризація - це будь-який метод, який обмежує свободу моделі, не даючи їй занадто сильно підлаштовуватися під навчальні дані. Припущення про те, що втрата інформації гарантовано призводить до погіршення роботи мережі працює тільки в умовах ідеальних вхідних даних, що на практиці майже не можливо. В реальних застосуваннях у вхідних даних часто зустрічаються зашумленості і артефакти, враховувати при навчанні які може бути не тільки зайвим, а і активно шкідливим. Модель може навчитися розрізняти не інформативний артефакт зображення і пов’язати його з певним класом, але після навчання може виявитися, що такий артефакт був присутній тільки в навчальній вибірці, а в реальних даних зустрічається не часто або не зустрічається взагалі. Проте модель в такому випадку буде очікувати артефакт і спиратися на нього як на один з основних аспектів для фінального прогнозу, що призведе до погіршення фінальних результатів на етапі валідації через те що частина “мозку” моделі просто ніколи не буде активована. Таким чином, змушуючи мережу виражати інформацію через меншу кількість каналів, вузьке місце не дає їй приділяти надмірну увагу на другорядні або шумові ознаки, що присутні лише в навчальній вибірці. Це обмежує здатність моделі до перенавчання і покращує її здатність до узагальнення (generalization), тобто вміння коректно працювати на нових, раніше небачених даних. Як наслідок, такий підхід часто призводить не до втрати, а до зростання фінальної точності моделі, що може спочатку здаватися контрінтуїтивним [17].

1.5 Допоміжні та спеціалізовані шари

1.5.1 Шар підвибірки (Pooling Layer)

Окрім згорткового шару в згорткових нейронних мережах також існує інший критично важливий тип шару - шар підвибірки (Pooling Layer). Коли згортковий шар завершив роботу і сформував карту локальних ознак, шари підвибірки використовують для поступового зменшення просторової розмірності карт ознак (downsampling). Це дозволяє знизити обчислювальну складність, а також надає моделі певну ступінь інваріантності до відносно невеликих зсувів та деформацій частин зображення [6]. Якщо згортковий шар намагається детально описати зображення і всі елементи, то шар підвибірки апроксимує ці деталі змушуючи наступні згорткові шари звертати увагу на більш загальні концепції, а не кожну окрему лінію, пляму, тощо.

Технічно, робота шару підвибірки схожа на роботу згорткового шару. По вхідній карті ознак ковзає вікно підвибірки (pooling window) заданого розміру (часто 2x2 або 3x3) з певним кроком (stride, зазвичай 2). Проте, на відміну від згорткового шару, шар підвибірки не має жодного параметру, що навчається (ваги та зсуви) і являє собою відносно просту операцію апроксимації значень всередині вікна [6]. Фундаментальною відмінністю шару підвибірки є його робота з глибиною - він діє виключно у просторовій площині (висота та ширина). Операція пулінгу застосовується до кожного 2D-каналу (зрізу глибини) незалежно, відповідно якщо шар підвибірки отримав об’єм даних розміром 28×28×512, то на виході отримаємо 14×14×512, тобто глибина (кількість карт ознак) не змінюється і залишається 512.

Існує два основні типи шарів підвибірки. Вони відрізняються лише математичною операцією, що застосовується до вікна (рис. 1.13).

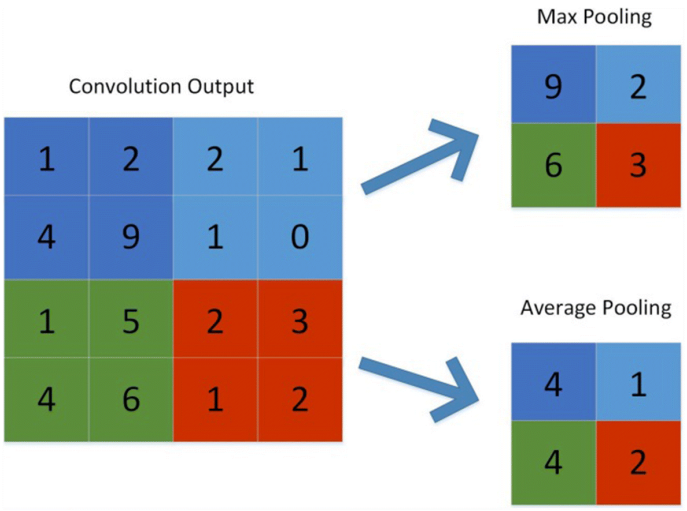


Рисунок 1.13 - Приклад застосування pooling

Max Pooling (Максимальна підвибірка) вважається класичним типом [16]. Для кожної ділянки значень, що знаходяться в межах вікна знаходиться максимальне значення і вихідне значення встановлюється як це максимальне значення. Математично це можна виразити як (1.22):

$$Y_{ij} = \max_{(u,v)\in R_{ij}} (X_{uv}); \tag{1.22}$$

де Y_{ij} - вихідний елемент;

R_{ij} - множина елементів вхідної карти X, що потрапили у вікно в позиції (i,j).

Уявімо, що на вхід шару підвибірки надійшла карта ознак розміром 4x4 і застосували до неї Max Pooling з розміром вікна 2x2 та кроком S=2, тоді:
Вхідна карта ознак X(4x4):

$$X = \begin{pmatrix} 6 & 8 & 1 & 2 \\ 3 & 4 & 5 & 7 \\ 9 & 2 & 0 & 1 \\ 5 & 6 & 4 & 8 \end{pmatrix};$$

Спочатку ковзне вікно накриє верхній лівий кут матриці

$$Y_{00} = \begin{pmatrix} 6 & 8 \\ 3 & 4 \end{pmatrix};$$

Знаходимо максимальне значення в межах вікна:

$$\max(6,8,3,4)=8;$$

На наступному кроці вікно буде зміщено на два кроки праворуч на ділянку

$$Y_{01} = \begin{pmatrix} 1 & 2 \\ 5 & 7 \end{pmatrix};$$

Знаходимо максимальне значення в межах вікна:

$$\max(1,2,5,7)=7;$$

Ми дісталися краю першого ряду тому повертаємося на початок але вікно зміщується на S вниз, отримуючи значення у вікні

$$Y_{10} = \begin{pmatrix} 9 & 2 \\ 5 & 6 \end{pmatrix};$$

Знаходимо максимальне значення в межах вікна:

$$\max(9,2,5,6)=9.$$

Знову зсувається праворуч у тому ж ряду отримуючи

$$Y_{11} = \begin{pmatrix} 0 & 1 \\ 4 & 8 \end{pmatrix};$$

Знаходимо максимальне значення в межах вікна:

$$\max(0,1,4,8)=8.$$

Відповідно отримуємо вихідну карту ознак розмірами 2x2 (в 4 рази меншу за вхідну):

$$Y = \begin{pmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{pmatrix};$$

$$Y = \begin{pmatrix} 8 & 7 \\ 9 & 8 \end{pmatrix};$$

Зменшення карти ознак в 4 рази (в 2 рази по кожній осі) скоротило кількість даних для наступного шару, при цьому зберігаючи найважливішу інформацію у вигляді найсильніших активацій з кожної ділянки. Таким чином отримали ефект даунсемплінгу і апроксимації, через що наступні шари зможуть навчатися більш загальним концепціям і будуть фокусувати увагу, наприклад, на кроні дерева в цілому, а не кожному окремому листку. Також таким чином значно зменшили вимоги до обчислювальних ресурсів, адже сам шар підвибірки можна вважати відносно простою операцією через те, що в ньому відсутні параметри для навчання, а наступний згортковий шар отримає меншу карту і зробить меншу кількість розрахунків.

Max Pooling ефективно забезпечує відповідь на питання присутності ознаки в межах вікна, зберігаючи найсильнішу активацію. Таким чином забезпечується інваріантність до зсувів, адже не важливо, де саме в межах вікна було знайдено ознаку - на виході фіксується сам факт її наявності.

Другий тип це Average Pooling (Середня підвибірка), в якому для кожної ділянки замість максимального значення знаходиться середнє арифметичне всіх значень в ній (1.23) [11].

$$Y_{ij} = \frac{1}{|R_{ij}|} \sum_{(u,v)\in R_{ij}} X_{uv} \quad ; \tag{1.23}$$

де $|R_{ij}|$- кількість елементів у вікні.

Для прикладу розрахунку візьмемо попередню вхідну матрицю:

$$X = \begin{pmatrix} 6 & 8 & 1 & 2 \\ 3 & 4 & 5 & 7 \\ 9 & 2 & 0 & 1 \\ 5 & 6 & 4 & 8 \end{pmatrix};$$

і застосуємо до неї Average Pooling з вікном 2×2 та кроком S=2, тоді:

Подібним до попереднього прикладу ковзаємо по матриці і розраховуємо середні арифметичні значення у кожному випадку.

Верхній лівий квадрат:

$$Y_{00} = \frac{6+8+3+4}{4} = \frac{21}{4} = 5.25;$$

Верхній правий квадрат:

$$Y_{01} = \frac{1+2+5+7}{4} = \frac{15}{4} = 3.75;$$

Нижній лівий квадрат:

$$Y_{10} = \frac{9+2+5+6}{4} = \frac{22}{4} = 5.5;$$

Нижній правий квадрат:

$$Y_{11} = \frac{0+1+4+8}{4} = \frac{13}{4} = 3.25;$$

Отримуємо вихідну карту ознак:

$$Y = \begin{pmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{pmatrix};$$

$$Y = \begin{pmatrix} 5.25 & 3.75 \\ 5.5 & 3.25 \end{pmatrix}.$$

Тобто Average Pooling також зменшує розмірність, але враховуючи внесок кожної ознаки в межах вікна, а не тільки найсильнішої. Цей тип застосовується рідше всередині мережі, але його варіація Global Average Pooling, часто використовується в кінці мережі як заміна повнозв'язним шарам з ціллю агрегації фінальних карт ознак [17]. Average Pooling використовується менш часто через те, що часто в задачах класифікації зображень важливіше знати, чи присутня ключова ознака фактично, ніж дізнатися її середню інтенсивність у межах вікна. У випадку використання Max Pooling зберігається тільки найсильніший і відповідно найінформативніший сигнал, тоді як при усередненні сильні і важливі активації можуть змішуватись з більш слабкими, по суті, розмиваючи інформативність ділянки. Задачі класифікації зображень вимагають точності, що можна порівняти з задачею знаходження найяскравішого пікселя на зображенні - Max Pooling одразу вкаже на нього, так як він найяскравіший, а Average Pooling покаже лише те, що певна ділянка на зображенні яскравіша за інші, що менш інформативно і не допоможе знайти певний піксель. Подібним чином працює і задача класифікації зображень і саме тому Average Pooling використовується рідше.

Проте Global Average Pooling (GAP) стала революційною ідею в проектуванні фінальних, класифікуючих частин згорткових нейронних мереж. Наприклад після останнього шару отримали фінальну карту ознак розмірами 7×7×512 (тобто 512 карт розміром 7x7). За умов класичного підходу з розгортанням такої карти в один одномірний вектор за допомогою операції випрямлення (flatten), отримуємо вектор розміром

$$7 \cdot 7 \cdot 512 = 25{,}088;$$

елементів, після чого подаємо його на вхід повнозв'язному шару з, наприклад 4096 нейронів, що є реальною кількістю нейронів в такому шарі в межах класичних архітектур, таких як VGG16, відповідно отримуємо

$$25{,}088 \cdot 4096 = 102{,}760{,}448;$$

ваг навчання. Також врахуємо зміщення по одному на нейрон, тоді

$$102{,}760{,}448 + 4096 = 102{,}764{,}544;$$

параметрів навчання, що вимагає значних обчислювальних ресурсів, тільки для одного повнозв'язного шару, хоча на практиці, особливо у більш глибоких архітектурах, може бути використано декілька повнозв'язних шарів. Більше одного шару тут використовується як механізм переходу від формування ознак до формування абстрактних концепцій - перші повнозв'язні шари вчаться виділяти з отриманих карт загальні концепції (хутро, кігті, гострі трикутники), наступні шари апроксимують ці концепції в ще більш абстрактні концепції (4 лапи, гострі вуха, великий хвіст, руде хутро), а останній шар класифікації приводить все до фінальної відповіді (можливо це лисиця або кіт або лев або тигр). Відповідно, в реальних випадках часто параметрів навіть більше ніж в наведеному прикладі. Надмірна кількість параметрів навчання може зробити шар схильним до проблем

перенавчання. Суть GAP, з іншого боку, полягає в застосуванні Average Pooling з розміром вікна, що дорівнює розміру всієї карти, тобто 7×7 у нашому прикладі. Відповідно обчислюємо середнє для всіх 49 значень першої карти і отримуємо одне єдине число, потім для другої карти і так далі для всіх 512 карт. Не складно зрозуміти, що на виході отримаємо простий вектор довжиною у 512 елементи, де кожен представляє собою усереднене значення відповідної карти ознак. Допитливий читач одразу помітить паралелі з використанням вікна згортки 1x1 в згортковому шарі для апроксимації, адже принцип роботи такого вікна і GAP дещо схожі - обидва виконують функцію агрегації інформації, але якщо згортка 1x1 робить це для кожного пікселя окремо з урахуванням глибини (всіх карт одночасно), то GAP враховує весь розмір карти з усіма пікселями одночасно і кожний канал (карту) окремо. Відповідно, як і використання вікна згортки 1x1, так і використання вікна підвибірки розміром з зображення, дозволяє радикальним чином зменшити кількість параметрів наступного повнозв'язного шару. Для нашого прикладу виходить

$$512 \cdot 4096 + 4096 = 2{,}101{,}248;$$

параметрів навчання. Це на

$$\frac{102{,}764{,}544 - 2{,}101{,}248}{102{,}764{,}544} \cdot 100\% = 97.955\%;$$

менше ніж у випадку застосування операції випрямлення (flatten). Враховуючи те, що у випадку використання повнозв'язного класифікатора параметрів може бути ще більше за рахунок додаткових шарів, а у випадку використання GAP можна одразу перейти до фінального повнозв'язного шару класифікатора при цьому не маючи додаткових параметрів навчання (шари підвибірки не мають ваг та зсувів), то відносна економія стає ресурсів стає ще більш значною. Використання GAP може легко зменшити кількість параметрів у 200 разів. У такому випадку значно

зменшується ризик виникнення проблем перенавчання і зменшуються вимоги до обчислювальних ресурсів [17]. Також таким чином легше провести інтерпретацію адже кожен вихідний нейрон фінального вектора відповідає одній карті ознак, що дозволяє легше зрозуміти яке фінальне рішення відноситься до яких конкретних візуальних патернів, що були знайдені мережею. GAP також дозволяє не використовувати додаткові повнозв’язні шари абстракції через те, що архітектура з використанням GAP припускає, що кожна карта ознак відповідає за свою певну ознаку, наприклад карта під номером 42 показує де на зображенні наявні ознаки очей. Відповідно, застосувавши до такої карти GAP отримаємо значення “впевненості” карти в наявності очей, де малі значення будуть пов’язані з аномаліями або хибними реакціями на округлі об’єкти, а високі значення будуть вказувати на те, що на зображенні майже гарантовано наявні чіткі очі. Отримуємо вектор ознак з 512 елементів, саме таку задачу виконують і декілька повнозв’язних шарів, але у випадку GAP можемо отримати цей вектор за допомогою одного шару, який не має параметрів для навчання. Таким чином, GAP може слугувати потужною альтернативою більш класичним повнозв’язним класифікаторам.

Окрім Max та Average Pooling існують і інші архітектури, створені для вирішення більш специфічних завдань. Наприклад стохастична підвибірка (Stochastic Pooling) [16] вносить елемент випадковості, розраховуючи ймовірність вибору для кожної активації пропорційно до її величини і залежно від цього вибирає випадковим чином вихідне значення де більш значні активації будуть обрані з більшою ймовірністю, але не у 100% випадків як в Max Pooling. Такий метод, за рахунок випадковості, значно ускладнює для моделі процес перенавчання. Проте, цей процес є обчислювально більш складним і на етапі валідації та тестування (inference) його поведінку додатково потрібно усереднювати. Max Pooling у комбінаціями з іншими класичними методами здатний досягти схожого ефекту регуляризації при менших затратах обчислювальних ресурсів.

Також існує Lp Підвибірка (Lp Pooling), що являє собою математичне узагальнення, а Max та Average Pooling можна представити як його крайні випадки роботи [6]. Формула Lp підвибірки (1.24):

$$Y_{ij} = \left(\sum_{(u,v) \in R_{ij}} (X_{uv})^p \right)^{1/p} ; \quad (1.24)$$

де p - це гіперпараметр.

При p=1 отримуємо поведінку аналогічну Average Pooling, а при p→∞ отримуємо поведінку аналогічну Max Pooling.

Окремим випадком використання є L2 Pooling (p=2), що обчислює корінь із суми квадратів активацій. L2 Pooling це компромісний метод, що завдяки піднесенню до квадрату враховує всі значення у вікні подібно Average Pooling але і надає більшу вагу сильнішим активаціям подібно Max Pooling, беручи найкраще з обох методів. На практиці ж, виграш у точності порівняно з Max Pooling часто є незначним, але все ще вимагає додаткових обчислень піднесення до ступіня та взяття кореня, що вважаються відносно складними комп'ютерними операціями.

Також існує Змішана підвибірка (Mixed Pooling), яка дозволяє моделі самостійно навчитися обирати оптимальну комбінацію з Max та Average Pooling. Вихідне значення обчислюється як зважена сума (1.25):

$$Y = \alpha \cdot \mathrm{MaxPool}(X) + (1 - \alpha) \cdot \mathrm{AvgPool}(X) ; \quad (1.25)$$

де α - це параметр, що навчається і приймає значення від 0 до 1.

Змішана підвибірка, вочевидь, гнучкий метод, який теоретично здатний адаптуватися під конкретну задачу навчившись вирішувати зберігати домінуючі ознаки або узагальнену інформацію. Таким чином можна отримати модель, що може самостійно обирати потрібний метод, що в особливо складних архітектурах

може мати сенс. Але такий метод додає складності моделі та процесу навчання використовуючи більше обчислювальних ресурсів. Для більшості відносно простих задач переваги не виправдовують такі ускладнення.

Існують і інші специфічні види шарів підвибірки, кожен з яких вузькоспеціалізований під свою задачі і не буде розглянутий в роботі.

Розуміючи принципи роботи шару підвибірки та його призначення залишилося лише зрозуміти як ефективність роботи шару залежить від його гіперпараметрів (таких що встановлюються завчасно), а саме розміру вікна (K) та кроку (S). Стандартні і перевірені часом конфігурації вже були розглянуті раніше, тепер розглянемо що станеться при їх подальших змінах за межі стандартів щоб глибше зрозуміти роль гіперпараметрів та шарів підвибірки загалом в архітектурі нейромережі. Для розрахунків вихідних розмірів карти ознак можна використати раніше розглянуту формулу для згорткового шару (за умови P=0).

Стандартна конфігурація називається підвибіркою що не перекривається, адже вікно ніколи не захватує значення які вже були оброблені, при цьому рухаючись “стик в стик” зі своїм попереднім положенням. Щоб створити підвибірку що не перекривається достатньо встановити крок рівним розміру вікна підвибірки, тоді всі розрахунки будуть проводитися як в вищенаведених прикладах. У найпоширенішому випадку підвибірки що не перекривається, де крок дорівнює розміру вікна (K=S), розмір карти ознак по кожній осі зменшується рівно в K разів. Наприклад, для K=2,S=2 карта 28x28 перетвориться на 14x14, зменшившись у 4 рази за площею.

Варто згадати, що ковзне вікно не завжди здатне повністю покрити зображення, наприклад парне вікно 2x2 з парним кроком 2 не врахує останній правий стовпець та нижній ряд непарного зображення розмірами 5x5, як і непарне вікно 3x3 з непарним кроком 3 не врахує останні два правих стовпця та нижніх ряди парного зображення 8x8, адже інакше вікно вийде за діапазон зображення. Така поведінка (Floor / Valid Padding) у більшості реалізацій є реалізацією за замовчуванням, дозволяє зберігти обчислювальні ресурси, але частково втрачає інформацію країв. Якщо ж стоїть задача зберегти інформацію ціною використання

додаткових ресурсів, то, подібно до шарів згортки, можна використати доповнення з правого та нижнього країв зображення потрібною кількістю нульових пікселів, проте доповнення нулями (Ceil / Same Padding) може призвести до гіршої роботи Average Pooling, адже нулі будуть враховані в усередненні. Саме через такі нюанси при проектуванні архітектур CNN перевагу часто віддають розмірам карт ознак, що є кратними степеням двійки (наприклад, 256, 128, 64, ...), що спрощує розрахунки та забезпечує стабільну роботу шарів підвибірки.

Якщо ж встановити крок меншим за розмір вікна подібно до того як це роблять для згорткового шару (K>S), то вже оброблені значення почнуть потрапляти повторно у межі матриці підвибірки. Наприклад, K=3, S=2, при цьому крайній правий ряд/стовпець першого вікна стає крайнім лівим для наступного, тобто вікна перекриваються на один стовпець. Це створить більш узагальнену карту ознак, через те що сусідні вихідні значення базуються на частково однаковій вхідній інформації, що може надати моделі певної стійкості до деформації ознак. Такий підхід використовувався в деяких ранніх архітектурах (наприклад, AlexNet) [14]. Зараз таке використання не вважається стандартом, адже вводить більшу кількість розрахунків зберігаючи надлишкову інформацію і не так сильно зменшуючи розміри вихідної карти ознак, оскільки на практиці сусідні значення карт ознак часто частково дублюють одне одного.

При S>K відбувається підвибірка з пропусками, адже вікно не захватує проміжні значення між попереднім та поточним положенням. Наприклад при K=2, S=3, вікно перестрибне один ряд значень і не врахує його створюючи сліпу зону, що призводить до втрати інформації (в сліпій зоні могла залишитися ключова ознака) і надмірно агресивного зменшення зображення, що в комбінації являє собою майже повну гарантію не вірної роботи нейромережі, адже у такому випадку модель просто не врахує критично важливу інформацію і не зможе видати релевантні результати.

Також можна спробувати використати підвибірку з розміром вікна і кроком K=1,S=1, але тоді операція не має сенсу, адже максимальне число у вибірці з одного числа це і є це число, рівно як і середнє значення у вибірці з одного числа і

є це число. Якщо в шарі згортки фільтр 1x1 є потужним інструментом для маніпуляції каналами, то в шарах підвибірки, які працюють з кожним каналом незалежно, вікно 1x1 не виконує жодної корисної функції, призводячи до точної копії вхідної карти, марно витрачаючи обчислювальні ресурси при цьому.

Таким чином, вибір гіперпараметрів шару підвибірки є важливим архітектурним рішенням. Стандартна конфігурація (K=2,S=2 або K=3,S=3) забезпечує найкращий баланс між ефективним зменшенням розмірності та збереженням інформації. Будь-які відхилення від неї є компромісом, наприклад перекриття (K>S) покращує плавність ціною обчислень, пропуски (S>K) ведуть до ризикованої втрати даних, великі розміри вікна ведуть до надмірного зменшення вихідної карти. Розуміння цих компромісів є ключовим при проектуванні ефективних згорткових нейронних мереж.

1.5.2 Шар пакетної нормалізації (Batch Normalization)

Ми розглянули основні невід'ємні шари, без яких складно уявити роботу згорткової нейромережі, адже вони виявляють та агрегують ознаки.

Такі шари, без яких робота мережі неможлива або практично неможлива будемо вважати основними і віднесемо до них шар підвибірки, шар згортки, шар активації, вхідний шар і блок або шар класифікації (вихідний). Шари, без яких робота нейромережі можлива, але з ними вона буде більш швидкою, ефективною, стабільною та зручною в побудові та налаштуванні будемо вважати службовими. Як правило, їх мета зводиться до забезпечення ефективності та стабільності процесу навчання замість пошуку нових ознак.

До таких допоміжних шарів можна віднести шар пакетної нормалізації (Batch Normalization), що насправді має дещо спірний статус в межах нашої бінарної класифікації шарів, адже без нього були побудовані деякі глибокі нейронні мережі, наприклад VGG. Проте саме поява пакетної стандартизації у 2015 році дозволила ефективно навчати надглибокі архітектури, такі як ResNet, успіх якої значною мірою завдячує саме цьому шару [19]. Де-факто, в сучасних

архітектурах він присутній у переважній більшості випадків навіть поза межами згорткових мереж.

Розглянемо цей шар. Як вже відомо, в процесі навчання ваги та зсуви в кожному попередньому шарі постійно оновлюються. Це призводить до того, що розподіл логітів, які отримує на вхід наступний шар, невпинно змінюється. Це відбувається через те, що алгоритм зворотного поширення помилки на кожній ітерації вносить зміни у ваги всіх попередніх шарів. Через це, навіть відносно незначне коригування ваг у першому шарі може призвести до лавиноподібного ефекту, що змінює розподіл активацій на десятому шарі в значній мірі. В результаті, на одній ітерації шар може отримувати значення в діапазоні [-2, 2], а вже через десять - в діапазоні [-100, 100]. Це явище, також відоме як внутрішній коваріантний зсув (Internal Covariate Shift), створює для мережі значні проблеми [28]. До таких проблем, наприклад, можна віднести сповільнення навчання, адже наступний шар змушений витрачати ресурси на те щоб і навчатися розпізнавати ознаки і постійно адаптуватися до плаваючого входу; або у випадку якщо логіти стають стабільно негативними значеннями, то функція активації ReLU буде завжди видавати нуль, а відповідно градієнт через такі нейрони проходити не буде що призведе до їх зупинки навчання; або навпаки, якщо логіти стають великими позитивними значеннями, то значення активації будуть відповідними, що створить надмірно великі градієнти і привиде до проблеми вибуху градієнту, що повністю паралізує модель, для боротьби з чим потрібно радикально зменшувати швидкість навчання. Шар пакетної нормалізації був створений саме для вирішення проблеми внутрішнього коваріантного зсуву. Він розташовується між шаром, що навчається (лінійна функція), та шаром активації (не лінійна функція) і приводить значення логітів до певного діапазону, щоб забезпечити стабільність розподілу для наступних шарів [28]. Варто зазначити, що хоча назва шару містить термін “нормалізація”, з математичної точки зору він виконує операцію стандартизації (приведення до середнього 0 та стандартного відхилення 1), а не нормалізації у вузькому сенсі (приведення до діапазону) [28]. Сам по собі шар стандартизації не приводить значення до фіксованого діапазону (як функція tanh до [-1, 1],

наприклад), але він стандартизує розподіл логітів, а потім дозволяє мережі самій навчитися оптимальному діапазону для кожного окремого каналу. Тобто, як буде наведено далі, завдяки параметрам γ та β, що навчаються, мережа може вирішити, що для певного каналу оптимальним є розподіл з середнім 0.5 та стандартним відхиленням 1.2, що може відповідати діапазону, наприклад, [-2, 3] [28].

Раніше в роботі згадувалась функція активації як невід'ємна частина шарів що навчаються, проте через те, що функція активації застосовується до кожного окремого нейрона і для зручності пояснення роботи Batch Normalization корисно уявити функцію активації саме як шар активації, адже стандартизація зазвичай встановлюється після отримання логітів, але до застосування активації. Тобто conv (згортковий шар) - Batch Normalization - ReLU. Таке розташування необхідне, щоб стандартизувати сирі значення логітів до того як вони потраплять до шару активації і призведуть до проблем зникаючих та/або вибухаючих градієнтів.

Процес стандартизації відбувається для кожного каналу в межах однієї підвибірки (mini-batch) даних, що складається з декількох карт ознак, окремо. Подібне визначення може бути не інтуїтивним, тому варто уточнити, що тут мається на увазі один канал (наприклад, 42-й канал вертикальних червоних ліній), але обробляється кілька карт ознак (відповідно розміру міні-вибірки) відповідно до кількості зображень. Тобто, наприклад 32 карти ознак з 32 різних зображень, але для одного каналу вертикальних ліній. Розмір "міні-партії" встановлюється як гіперпараметр відповідно до можливостей і доступного об'єму пам'яті GPU (зазвичай як ступінь двійки). Тобто для одного каналу шар враховує всі логіти з усіх прикладів в межах партії та обчислює середнє значення (μ) та дисперсію (σ^2), де середнє являє собою центральну точку, навколо якої розподілені значення, а дисперсія показує міру розкиду цих значень навколо неї. Дисперсія розраховується як середнє значення квадратів відхилень від середнього, де використання квадрату дозволяє уникати проблеми взаємного знищення позитивних та негативних відхилень. Далі кожен окремий логіт x_i стандартизується за формулою стандартизації (1.26):

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} ; \quad (1.26)$$

де є - дуже мале число для уникнення ділення на нуль, наприклад 10^{-5} (і відповідно уникнення проблеми нульового градієнту).

Ця операція показує, на скільки стандартних відхилень дане значення відрізняється від середнього. В результаті отримуємо значення з гарантованими властивостями - всі логіти приводяться до вигляду де середнє значення буде дорівнювати 0, а стандартне відхилення та дисперсія 1, що зручно для подальших перетворень.

Для кращого розуміння механіки шару, розрахуємо всі кроки на конкретному прикладі. Наприклад, навчаємо мережу з розміром партії (mini-batch size) 2, і на вхід шару пакетної нормалізації для одного каналу надійшли дві карти ознак (від першого та другого зображення в партії) розміром 2×2, а саме:

$$X_1 = \begin{pmatrix} 10 & 20 \\ -6 & -4 \end{pmatrix} \quad X_2 = \begin{pmatrix} 30 & -10 \\ 0 & 10 \end{pmatrix};$$

Відповідно, всього в межах цього каналу по всій партії маємо

$$2\times(2\times2)=8;$$

значень: [10, 20, -6, -4, 30, -10, 0, 10].

Обчислимо середнє значення (μ) по всій партії:

$$\mu = \frac{10 + 20 - 6 - 4 + 30 - 10 + 0 + 10}{8} = \frac{50}{8} = 6.25 ;$$

Обчислимо дисперсію (σ^2) по всій партії:

$$\begin{aligned}\sigma^2 &= \frac{1}{8}\sum_{i=1}^{8}(x_i-\mu)^2 \\ &= \frac{1}{8}\big((10-6.25)^2+(20-6.25)^2+(-6-6.25)^2+(-4-6.25)^2 \\ &\quad +(30-6.25)^2+(-10-6.25)^2+(0-6.25)^2+(10-6.25)^2\big) \\ &= \frac{1}{8}(14.0625+189.0625+150.0625+105.0625+564.0625+264.0625+39.0625+14.0625) \\ &= \frac{1339.5}{8} = 167.4375\end{aligned};$$

Обчислимо стандартне відхилення (σ):

$$\sigma = \sqrt{\sigma^2+\epsilon} = \sqrt{167.4375+10^{-5}} \approx 12.9397 ;$$

Застосуємо формулу стандартизації до кожного логіта:

$$\begin{aligned}\hat{x}_{1,00} &= \frac{10-6.25}{12.9397} \approx 0.290 & \hat{x}_{2,00} &= \frac{30-6.25}{12.9397} \approx 1.835 \\ \hat{x}_{1,01} &= \frac{20-6.25}{12.9397} \approx 1.063 & \hat{x}_{2,01} &= \frac{-10-6.25}{12.9397} \approx -1.256 \\ \hat{x}_{1,10} &= \frac{-6-6.25}{12.9397} \approx -0.947 & \hat{x}_{2,10} &= \frac{0-6.25}{12.9397} \approx -0.483 \\ \hat{x}_{1,11} &= \frac{-4-6.25}{12.9397} \approx -0.792 & \hat{x}_{2,11} &= \frac{10-6.25}{12.9397} \approx 0.290\end{aligned};$$

Отримуємо стандартизовані значення логітів у вигляді:

$$\hat{X}_1 = \begin{pmatrix}0.290 & 1.063 \\ -0.947 & -0.792\end{pmatrix} \quad \hat{X}_2 = \begin{pmatrix}1.835 & -1.256 \\ -0.483 & 0.290\end{pmatrix} ;$$

Після цього потрібно застосувати масштабування та зсув, адже повна стандартизація може бути шкідливою, обмежуючи репрезентативну силу мережі. Якщо примусово зводити всі значення до середнього 0 та дисперсії 1, то у деяких

випадках, наприклад при застосуванні функції активації Sigmoid, може виникнути ситуація, коли важлива інформація знаходиться в її насиченій області, а приводимо всі значення в межі близькі до нуля. Задля уникнення цього, шар має два параметри, що навчаються для кожного каналу, а саме - гамма (γ), що відповідає за масштаб (нове стандартне відхилення), та бета (β), що відповідає за зсув (нове середнє значення). За допомогою цих параметрів, шар пакетної нормалізації виконує фінальне перетворення після стандартизації, дозволяючи мережі самій за допомогою градієнтного спуску навчитися оптимальному масштабу та зсуву для кожного каналу (1.27):

$$y_i = \gamma \hat{x}_i + \beta \quad ; \qquad (1.27)$$

Цікаво помітити, що у випадку якщо мережа навчиться γ=σ та β=μ, то вона може відновити початкові дані скасувавши стандартизацію, що робить шар абсолютно гнучким і здатним стандартизувати дані тільки тоді, коли це справді потрібно для досягнення кращих результатів [28].

Відповідно, у нашому прикладі (припустимо, що значення параметрів γ=2.0 та β=1.5), за формулою (1.27):

$$\begin{aligned} y_{1,00} &= 2.0 \cdot 0.290 + 1.5 = 2.08 \\ y_{1,01} &= 2.0 \cdot 1.063 + 1.5 = 3.626 \\ y_{1,10} &= 2.0 \cdot (-0.947) + 1.5 = -0.394 \\ y_{1,11} &= 2.0 \cdot (-0.792) + 1.5 = -0.084 \end{aligned} \qquad \begin{aligned} y_{2,00} &= 2.0 \cdot 1.835 + 1.5 = 5.17 \\ y_{2,01} &= 2.0 \cdot (-1.256) + 1.5 = -1.012 \\ y_{2,10} &= 2.0 \cdot (-0.483) + 1.5 = 0.534 \\ y_{2,11} &= 2.0 \cdot 0.290 + 1.5 = 2.08 \end{aligned} \quad ;$$

Таким чином, фінальний вихід шару пакетної нормалізації буде:

$$Y_1 = \begin{pmatrix} 2.08 & 3.626 \\ -0.394 & -0.084 \end{pmatrix} \quad Y_2 = \begin{pmatrix} 5.17 & -1.012 \\ 0.534 & 2.08 \end{pmatrix} ;$$

Ці отримані значення будуть передані до функції/шару активації.

Шар пакетної нормалізації гарантує, що наступний шар завжди отримує набір даних з стабільним та передбачуваним розподілом, що, незважаючи на те що цей шар також навчається і вимагає додаткових ресурсів, дозволяє використовувати значно вищі швидкості навчання [28]. Також, використання шару стандартизації знижує чутливість до ініціалізації ваг - без пакетної стандартизації успіх навчання глибокої мережі залежить від того, якими саме початковими випадковими значеннями були ініціалізовані ваги [28]. “Невдала” випадкова ініціалізація може викликати екстремальні значення логітів майже одразу, що унеможливлює навчання і зводить вдале навчання моделі до лотереї, а через те, що Batch Normalization центрує та стискає розподіл логітів на кожному шарі, ефект від невдалого початкового вибору ваг нівелюється, роблячи процес навчання значно надійнішим. Також, він вирішує проблеми градієнтів і таким чином, вся сукупність факторів дозволяє створювати навіть швидші та надійніші мережі [6].

1.5.3 Шар виключення (Dropout)

Одна з ключових проблем при навчанні глибоких нейронних мереж, в яких, як правило, відносно велика кількість параметрів є проблема перенавчання (overfitting), що виникає у випадку, коли модель надмірно адаптується до аномалій, специфічних особливостей, шуму тощо, що можуть бути присутніми в межах навчальної вибірки (наприклад, навчається приймати випадково потрапивши в виборку набору даних зображень частки пилу за реальну ознаку певного класу) [29]. При цьому втрачається здатність до узагальнення на нових, раніше небачених даних, проте результати під час тренування виглядають ідеально, що може збивати інтуїтивне розуміння результатів моделі. Класичним проявом перенавчання є розходження кривих навчання - випадок, коли помилка моделі на тренувальних даних (наприклад, training loss) з кожною епохою продовжує наближатися до нуля, але на валідаційних даних (відповідно, наприклад, validation loss) з певного моменту починає зростати, або майже одразу значно відхиляється від тренувальної помилки [6]. Така точка розходження наочно

демонструє момент, коли модель перестає навчатися узагальнюючим патернам і починає заучувати навчальні дані. Саме для моніторингу цього явища і використовується валідаційна (екзаменаційна) вибірка, а досягнення мінімуму на кривій валідаційної помилки часто є критерієм для зупинки навчання. При цьому зупинка навчання може відбуватися або по досягненню встановленого максимума епох навчання, або більш гнучко за допомогою, наприклад, алгоритма ранньої зупинки (Early stopping), який автоматично зупиняє навчання у випадку, якщо валідаційна помилка не спадає нижче раніше зафіксованого мінімуму на протязі встановленої кількості епох [6]. Проблема перенавчання проявляє себе тим сильніше чим вища щільність зв'язків, тобто у повнозв'язних шарах, наприклад. Це відбувається тому, що нейрони можуть навчитися формувати складні співзалежності, наприклад, навчитися виправляти помилки інших нейронів, що спирається на нестабільні комбінації активацій [29]. Наприклад, уявімо задачу класифікації птахів. При перенавчанні, перший нейрон може навчитися розпізнавати довгу трикутну форму дзьоба птаха, але робити це з помилками (що є природним для нейронних мереж і майже неможливо позбутися помилок повністю), реагуючи також на продовгуваті форми літаків у небі на фоні зображення, що іноді випадково попадали в кадр. Тоді другий нейрон може навчитися тому, що хибні активації першого нейрона часто корелюють з наявністю хмар на зображенні. Тоді другий нейрон може навчитися видавати сильний позитивний, підкріплюючий сигнал коли "бачить" хмари на зображенні. Таким чином мережа навчається правилу "це птах, якщо перший та другий нейрони активні". Це дуже крихке правило, адже якщо потім показати мережі зображення птаха у фотостудії, другий нейрон не активується адже не побачить хмар і як результат впевненість передбачень нейромережі катастрофічно впаде. Подібний ефект перенавчання можна уявити як списування між студентами (нейронами) - студенти що списують майже нічому не навчаються, ідеально копіюючи колег або готові відповіді, але не мають розуміння предмету, від чого на екзамені (нових даних) не будуть здатні набрати високий бал. Для боротьби з проблемою перенавчання було запропоновано багато методів, деякі з яких вже

розглянули раніше, такі як L2-регуляризація або використання шару пакетної нормалізації. З метою примусового розриву таких співзалежностей і змушення кожного нейрону навчатися бути більш самостійним, у 2014 році було запропоновано шар виключення (Dropout) (рис. 1.14) [29].

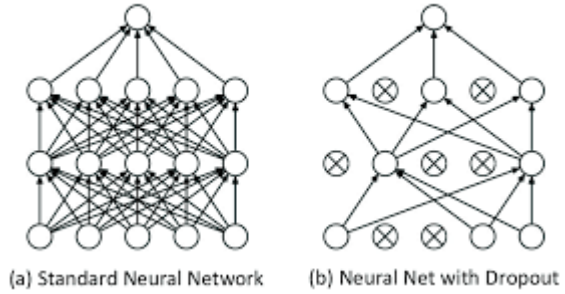


Рисунок 1.14 - Приклад застосування dropout

У суті своїй, шар Dropout працює радикально просто і його роботу можна описати одним реченням - він просто випадковим чином під час кожної епохи вимикає певну кількість (у відсотках) нейронів. Проте розглянемо принцип роботи цього службового шару стабілізатору детальніше. Задля того, щоб розірвати складні співзалежності між нейронами, шар примусовим чином на кожній ітерації встановлює значення випадково обраної частини нейронів з попереднього шару на нуль і передає отримані, модифіковані дані далі на наступні шари. Таким чином створюється стан постійної невизначеності при навчанні, у якому кожен окремий нейрон просто не здатен покладатися на сусідні нейрони і тому він змушений вивчати індивідуальні ознаки, що корисні самі по собі (студент почне вивчати матеріал якщо знатиме, що ті у кого він би міг списати часто можуть просто нічого не робити всю пару або навіть не прийти на неї). Тобто замість однієї великої мережі, тепер можна уявити процес навчання як навчання ансамблю з великої кількості менших підмереж, що значно покращує здатність мережі до узагальнення [29].

Як вже було вказано, Dropout застосовується до виходів (активацій) попереднього шару. Робота шару визначається одним гіперпараметром p (rate), що

відповідає за те, який відсоток нейронів буде занулено за ітерацію. Цей параметр на практиці зазвичай приймає значення від 0.1 (10%) до 0.5 (50%) [29]. Менші значення не мають сенсу через слабкий регуляризаційний ефект, а більші можуть лише нашкодити навчанню мережі через те, що при > 0.5 мережа під час кожної ітерації втрачає більше половини нейронів, що може призвести до проблеми недонавчання (underfitting), яка виникає коли модель занадто слабка для того щоб вивчити навіть базові закономірності в даних, в цьому випадку через те, що їй не вистачить активних нейронів для передачі комплексної інформації. Під час навчання для кожного прикладу з отриманої партії створюється бінарна маска. Тобто випадковим (а точніше кажучи, часто псевдо-випадковим чином) створюється бінарна матриця d того ж розміру, що й вектор активацій y з попереднього шару, де кожен елемент маски di є незалежною випадковою величиною з розподілу Бернуллі (1.28) [29]:

$$d_i \sim \text{Bernoulli}(1-p) \text{ ;} \qquad (1.28)$$

що означає, що з ймовірністю 1-p елемент маски дорівнює 1 (значення нейрону пропускається далі без змін), і з імовірністю p-0 елемент маски дорівнює 0 (значення нейрона примусово перезаписується на нуль).

Цікаво згадати, що цей процес не має пам’яті, тобто вибір нейронів на поточній ітерації ніяк не залежить від того, які нейрони були обрані для вимикання на попередній. Відповідно, теоретично, може виникнути ситуація, коли один і той самий нейрон буде вимкнено декілька разів поспіль, що може призвести до проблеми затухаючої активації і відключити нейрон назавжди. Але ймовірність такої ситуації на практиці мізерна і навіть якщо вона виникне для одного нейрону, велика ймовірність що це майже не вплине на загальну роботу мережі. Тому не має сенсу додавати додаткову логіку і витрачати додаткові обчислювальні ресурси на введення алгоритму запам'ятовування попередніх обраних нейронів і їх відкидання в випадковій виборці.

Відповідно, після формування маски відкидань, вихідний вектор y поелементно множиться на маску d, обнуляючи частину активацій (добуток Адамара, що позначається символом ⊙ і означає, що кожен елемент вектора y множиться на відповідний елемент маски d, що стоїть на тій же позиції) (1.29):

$$y_{\text{dropped}} = y \odot d \ ; \tag{1.29}$$

Допитливий читач може помітити, що при цьому значення активацій вимкнених нейронів встановлюються на нуль, що повинно призвести до проблеми вмираючого ReLU. Проте таке твердження є хибним, адже при вмираючих активаціях стан нейрону перманентний і виникає через стабільно негативні логіти, що надходять на функцію активації такого нейрону, що призводить до нульового градієнту ваг цього нейрона. Dropout же обнуляє нейрони вже після активації і лише на одну ітерацію (в більшості випадків), а на наступній нейрон працює вже як і до цього і його ваги зможуть оновитися. Навіть декількох таких короткоплинних вимкнень за процес навчання не буде достатньо для виникнення проблем, адже вони (вимкнення) тимчасові. Проте важливо розуміти, що обнулення активації нейрона також призводить до нульового градієнта для всіх ваг попереднього шару, пов'язаних із цим нейроном, під час зворотного поширення помилки. Таким чином, Dropout тимчасово припиняє оновлення ваг попереднього шару, що є контрольованою формою вимирання градієнта на даній ітерації.

Далі виникає проблема того, що під час навчання, в середньому, використовується лише (1-p)·100% нейронів, а на етапі тестування - всі 100%. Таким чином, сумарна активація, що надійде на наступний шар під час тестування буде значно вищою, що може порушити роботу мережі. Задля того щоб нівелювати такий ефект, активації, які не були занулені, діляться на коефіцієнт 1-p (зворотне масштабування) (1.30) [29]:

$$y' = \frac{y_{\text{dropped}}}{1-p} \quad ; \tag{1.30}$$

Таким чином, гарантуємо, що математичне очікування (середнє значення, яке б отримали, якби повторили випадковий процес - у нашому випадку, генерацію маски Dropout, нескінченну кількість разів. На кожній окремій ітерації результат буде випадковим, але в довгостроковій перспективі середнє цього шуму буде прагнути до початкового значення.) суми активацій залишиться таким же на етапі навчання як і на етапі тестування (inference). Відповідно на етапі тестування шар Dropout вимикається (p автоматично встановлюється на 0.0) і ніяк не впливає на роботу мережі, просто пропускаючи через себе дані наскрізь, оскільки ефект масштабування вже врахований під час навчання.

Приведемо приклад розрахунків. Нехай на вхід шару Dropout з p = 0.5 надійшов вектор активацій з 4 нейронів:

$$y = [10 \quad 20 \quad 30 \quad 40] \quad .$$

Наприклад, було згенеровано випадкову маску:

$$d = [1 \quad 0 \quad 1 \quad 0] \quad .$$

Застосуємо маску до матриці активацій:

$$y_{\text{dropped}} = y \odot d = [10 \cdot 1 \quad 20 \cdot 0 \quad 30 \cdot 1 \quad 40 \cdot 0] \quad ;$$

$$y_{\text{dropped}} = [10 \quad 0 \quad 30 \quad 0] \quad .$$

Бачимо, що сума значень початкової матриці дорівнює 100, а отриманої лише 40. Застосуємо зворотне масштабування:

Ділимо отриману матрицю на 1-p:

$$y' = \frac{y_{\text{dropped}}}{1-p} = \frac{[10 \quad 0 \quad 30 \quad 0]}{1-0.5} = \frac{[10 \quad 0 \quad 30 \quad 0]}{0.5};$$

$$y' = [20 \quad 0 \quad 60 \quad 0].$$

Саме цей вектор з загальною сумою 80 буде передано на наступний шар. Хоча на даній конкретній ітерації сума активацій (80) не дорівнює початковій (100) через випадковість вибору, математичне очікування виходу після масштабування залишається рівним входу. Щоб довести це, візьмемо до прикладу один нейрон з виходом y=10 та p=0.5. В такому випадку, існує лише два можливі сценарії: нейрон занулюється з 50% ймовірністю, або нейрон не занулюється з 50% ймовірністю і його вихід після зворотного масштабування стане

$$\frac{10}{(1-0.5)} = 20.$$

Математичне очікування для цього нейрона розраховується як сума добутків кожного можливого результату на його ймовірність:

$$E[y'] = (0 \times 0.5) + (20 \times 0.5) = 0 + 10 = 10.$$

Звідси, середнє очікуване значення точно дорівнює початковому. Це гарантує, що в середньому по багатьох ітераціях сигнал не згасає і не вибухає, що стабілізує процес навчання.

Як вже було згадано, навчання мережі з використанням Dropout можна розглянути як навчання експоненційно великого ансамблю різних підмереж, що ділять між собою ваги. На етапі тестування використовується повна мережа, що фактично являє собою апроксимацію усереднення прогнозів від усього ансамблю

підмоделей. Власне, така потужна техніка підвищення точності і називається ансамблювання. Ансамблювання через Dropout ефективніше, хоча і менш точне, ніж пряме ансамблювання через навчання кількох незалежних моделей і усереднення їх передбачень [29].

Використання Dropout зазвичай збільшує кількість епох (через те що не всі нейрони навчаються одночасно), необхідних для збіжності моделі, через що час навчання моделі в цілому збільшується в обмін на кращі результати моделі і менші ризики перенавчання [29].

Dropout найчастіше застосовується між повнозв'язними шарами в кінці мережі, де ризик перенавчання є найвищим через велику кількість зв’язків, і ставиться після шарів активації повнозв’язних шарів.

Використання Dropout шару не є панацеєю і не може бути застосоване до відносно простих, малих, слабких мереж, що не схильні до перенавчання і самі по собі не мають достатню потужність. Використання Dropout у такому випадку може навіть зашкодити, обмежуючи і без того слабку модель.

1.6 Загальна архітектура згорткової нейронної мережі та її компоненти

Тепер, розглянувши всі базові будівельні блоки згорткових нейронних мереж, можемо зібрати їх разом, щоб побудувати та приблизно розрахувати архітектуру і потреби типової згорткової нейронної мережі (CNN).

Класичну побудову можна умовно поділити на два основних функціональних блоки - блок вилучення ознак (Feature Extractor) та блок класифікації (Classifier) [6]. Часто тип мережі визначається останнім блоком - якщо останній блок це блок класифікації, приймаючий рішення на основі виявлених ознак, то і мережу називають класифікатором. Існують також і інші типи блоків, такі як блоки генератори (декодери), що генерують зображення, блоки сегментації, що класифікують кожен піксель окремо дозволяючи наглядно побачити де саме модель побачила ознаки класу, тощо, але такі механізми в роботі не розглядаються адже потребують роз’яснення принципово нових механізмів

обробки і декодування. Блок вилучення ознак відповідає за обробку просторових даних у той час як блок класифікації відповідає за формування остаточного прогнозу на базі цих даних. За будовою, блок вилучення ознак зазвичай становить собою послідовність шарів, що повторюються і кожен шар складається за згорткових шарів (Convolutional або Conv2D), шарів активації (ReLU або інші), шарів пакетної нормалізації (Batch Normalization), шарів підвибірки (Pooling), тощо. Суть у тому, що в межах блоку карта ознак стає меншою за висотою та шириною, але глибшою, що, як вже неодноразово було згадано, дозволяє моделі переходити від знаходження простих патернів до складних концепцій [16]. Часто, з кожним наступним блоком відбувається збільшення кількості згорткових фільтрів, що дозволяє виявляти все більш складні комбінації ознак, тоді як просторова розмірність зменшується за допомогою шарів підвибірки [15]. Блок вилучення ознак також у менш формальній формі називають тілом нейромережі. Відповідно, блок класифікації називають головою нейромережі. Він приймає оброблений набір ознак з попередніх шарів, випрямляє їх (Flatten) з багатомірної карти ознак на одновимірний вектор і обробляє цей вектор одним або декількома повнозв’язними шарами (Dense або Fully Connected), що навчаються аналізувати комбінації абстрактних ознак для прийняття фінальних рішень. Найчастіше, застосування шару виключення (Dropout) для боротьби з перенавчанням відбувається саме у цьому блоці [29]. На виході блоку встановлюється вихідний шар з кількістю нейронів та функцією активації, що відповідають задачі. Для згорткових нейронних мереж задачі зазвичай поділяються на бінарну класифікацію, суть задачі якої полягає в визначенні належності або не належності до певного класу (є патологія або немає патології), і вихідний шар має один нейрон з функцією активації Sigmoid або її варіацією; багатокласову класифікацію (Multiclass Classification), при якій суть задачі у визначенні одного і тільки одного класу з багатьох (на зображенні об’єкт X або У або Z), при цьому кількість вихідних нейронів відповідає кількості класів, а функція активації зазвичай Softmax або її варіація і сума ймовірностей завжди дорівнює одиниці і клас з найбільшим значенням впевненості обирається як фінальна відповідь;

багатолейблова класифікація (Multilabel Classification), при якій суть задачі в визначенні набору класів, до яких може належати об’єкт одночасно (на зображенні є і кіт і птах але не собака), при цьому кількість нейронів також відповідна кількості класів і для кожного використовується Sigmoid або її варіації, але кожен нейрон видає свою впевненість окремо і їх сума не зобов’язана дорівнювати одиниці.

Тут доречно глибше розглянути саме етап прийняття фінального рішення. Як вже було розглянуто раніше, для перетворення логітів вихідного шару на ймовірності використовуються спеціальні функції Sigmoid або Softmax залежно від задачі. Sigmoid краще підходить до задач бінарної та багатолейблової класифікації і перетворює кожен логіт на незалежну ймовірність в діапазоні (0,1). Але щоб остаточно перетворити цю імовірність на кінцеве бінарне рішення (належить / не належить) застосовується порог прийняття рішення (decision threshold). Стандартним значенням вважається 0.5 (50% впевненості) і якщо вихід нейрона перевищіть це значення, то об’єкт буде вважатися належним до класу. Значення порогів кожного класу можна обирати вручну як гіперпараметри або автоматично за допомогою спеціалізованих алгоритмів оптимізаторів. Значення порогів можуть впливати на результати мережі залежно від потреб. Наприклад, якщо від мережі вимагається велика точність і, в межах задачі, приблизність і неточність неприпустимі (наприклад, відбір максимально ідеальних співпадінь), то пороги можна встановити вищими, щоб мережа видавала належність до класу лише у випадку високої впевненості у цьому. І навпаки, якщо стоїть задача, наприклад, діагностики захворювань, то кращим вибором може стати зниження порогів, щоб мережа відправляла пацієнтів на додаткові обстеження частіше навіть якщо не дуже впевнена, адже краще провести додаткове обстеження здорової людини ніж відпустити додому хвору сказавши, що вона не хвора. Таким чином, для задачі багатолейблової класифікації (бінарна працює аналогічно але на одному нейроні), якщо для трьох класів модель видала вектор ймовірностей як [0.9, 0.35, 0.51], то при порозі 0.5 фінальним прогнозом буде вектор [1, 0, 1], де 1 вказує на наявність класу, а 0 на його відсутність. У випадку багатокласової

класифікації використовується функція Softmax, що враховує всі логіти разом і приводить їх до розподілу ймовірностей, сума яких завжди дорівнює 1. Тут поріг не має сенсу, адже фінальним класом обирається нейрон з максимальною передбаченою ймовірністю за допомогою операції argmax. Тобто на векторі [0.5, 0.3, 0.2], що, наприклад, отримали після застосування Softmax на сирих логітах, при застосуванні після цього argmax отримаємо значення 0 - argmax повертає індекс найбільшого значення (нумерація починається з 0), що умовно можна уявити як вектор [1, 0, 0], адже 0.5 це найбільше значення в векторі.

Варто підкреслити, що галузь машинного навчання розвивається швидкоплинно і експоненційно на момент створення роботи. Розглянуті шари, механізми і техніки покращуються майже кожного місяця і створюються альтернативи або модифікації для вирішення специфічних проблем. На момент написання роботи, наприклад, було представлено новий алгоритм оптимізації демонструючий покращені результати порівняно з іншими сучасними варіаціями Adam оптимізатору. Розглянуті в роботі елементи можна вважати базовими та перевіреними часом. Серед альтернатив можна навести:

Альтернативи згортковому шару:

- Роздільна згортка (Depthwise Separable Convolution), що розділяє стандартну згортку на два етапи (просторову та глибинну). Це, у свою чергу, дозволяє радикально зменшити кількість обчислювальних операцій та параметрів. Є основою сучасних компактних мобільних архітектур, таких як EfficientNet [20].
- Транспонована згортка (Transposed Convolution), це операція, що дозволяє збільшувати просторову розмірність карт ознак (апсемплінг) і є ключовим компонентом у генеративних моделях та архітектурах сегментації [16].
- Розширена згортка (Dilated / Atrous Convolution), це модифікована згортка, що дозволяє експоненційно збільшувати рецептивне поле без збільшення кількості параметрів за допомогою введення пропусків у ядрі фільтра [6].

Альтернативи функції активації ReLU:

- Leaky ReLU та PReLU (Parametric ReLU), це функції, що допомагають вирішити проблему вмираючих нейронів шляхом введення невеликого фіксованого (Leaky) або навченого (Parametric) значення для негативних значень, що дозволяє градієнтам поширюватися навіть у цій області через те, що значення після функції ніколи не дорівнює нулю [27].
- ELU (Exponential Linear Unit), це метод, що використовує експоненційну функцію для негативних значень, що може прискорити навчання, наближаючи середнє значення активацій до нуля [6].
- GeLU та Swish (SiLU), це більш сучасні, гладкі функції активації що демонструють кращу продуктивність у глибоких архітектурах [21].

Альтернативи шару підвибірки (Pooling):

- Згортка з кроком (Strided Convolution), це використання згорткового шару з кроком S>1 у якості альтернативи шару Max Pooling, що дозволяє моделі самій навчатися оптимальному способу зменшення розмірності [19].
- Стохастичний пулінг (Stochastic Pooling), це метод, що замість детермінованого вибору виконує випадкову вибірку активації з вікна на основі їхніх величин, діючи як метод регуляризації [6].

Альтернативи пакетній нормалізації (Batch Normalization):

- Layer Normalization, Instance Normalization, Group Normalization, все це альтернативні методи нормалізації, які обчислюють статистики не по всій партії, а в межах одного прикладу, що робить їх ефективними при малих розмірах партії та в рекурентних архітектурах [28].

Альтернативи оптимізатору Adam (докладніше про оптимізатори при побудові практичної частини нейромереж):

- AdamW, це поширена модифікація Adam з більш коректним механізмом регуляризації ваг (weight decay), що часто призводить до кращої узагальнюючої здатності моделі [32].
- Сучасні оптимізатори, такі як Lion та інші показують, що дослідження в області оптимізації тривають, і періодично з'являються нові

алгоритми, що пропонують покращену швидкість збіжності або менше споживання пам'яті.

Цей далеко неповний перелік демонструє той факт, що дисципліна глибокого машинного навчання являє собою високо динамічну і актуальну галузь досліджень. Проте в рамках роботи більше уваги буде приділено побудові, застосуванню та аналізу моделей, що побудовані на основі більш класичних та перевірених часом компонентів. Цей підхід дозволяє створити фундаментальне розуміння в межах задачі. До того ж таким чином можна буде зрозуміти чи достатньо класичних методів для вирішення поставленої задачі і на скільки можна покращити результати застосовуючи відносно сучасні методи і чи варте воно того, особливо у випадку додавання елементів або технік збільшуючих потреби у розрахункових ресурсах і часу.

1.7 Проектування і реалізація тривіальної моделі

Побудуємо план архітектури моделі для вирішення задачі класифікації захворювань очного дна. Почнемо з першої і найпростішої, на прикладі якої сформуємо практичне уявлення про реалізацію, будову та роботу нейромережі. Ця модель потенційно стане відправною точкою подальших досліджень (baseline), або доведе нездатність надпростих архітектур до виконання задач подібної складності.

Для початку, перша модель буде побудована з трьох згорткових блоків та блоку класифікації. Тобто:

1. Вхід (Input): Зображення з набору даних ODIR [23] (модифіковане), розміром 256×256×3
2. Conv2D (32 фільтрів, ядро 3×3, same padding, активація ReLU) [27]. Ядро 3×3 та same padding є стандартним підходом для збереження просторової розмірності на ранніх етапах [15]. Відповідно, кількість параметрів (ваг та зсувів) шару дорівнює:

$$\mathrm{P}_{\mathrm{Conv1}} = ((3 \times 3 \times 3) + 1) \times 32 = 896 \text{ ;}$$

3. MaxPooling2D (вікно 2×2, крок 2). Не має параметрів, але зменшує просторову розмірність в 2 (по кожній осі, тобто в 4 рази загалом) рази до 128×128, зберігаючи глибину у 32 канали.
4. Conv2D (64 фільтри, ядро 3×3, same padding, ReLU). Після Conv2D кількість фільтрів (відповідно і глибина також) у наступному згортковому шарі збільшується до 64, що є стандартною практикою, що компенсує зменшення просторової розмірності збільшенням глибини для вивчення складніших комбінацій ознак [15]. Кількість параметрів:

$$\mathrm{P_{Conv2}} = ((3 \times 3 \times 32) + 1) \times 64 = 18,496 \text{ ;}$$

5. MaxPooling2D (вікно 2×2, крок 2). зменшує карту ознак до 64×64.
6. Conv2D (128 фільтри, ядро 3×3, same padding, ReLU). Кількість параметрів:

$$\mathrm{P_{Conv3}} = ((3 \times 3 \times 64) + 1) \times 128 = 73,856 \text{ ;}$$

7. MaxPooling2D (вікно 2×2, крок 2). Остаточне зменшення до 32×32×128.
8. Flatten. Перетворює тривимірну карту ознак в одновимірний вектор розмірністю 32×32×128=131,072 елементи. Не має параметрів для навчання.
9. Dense (128 нейрони, ReLU). Повнозв’язний шар. Кількість параметрів:

$$\mathrm{P_{Dense1}} = (131,072 \times 128) + 128 = 16,777,344 \text{ ;}$$

10. Dense (8 нейронів, Sigmoid). Вихідний шар для багатолейблової класифікації 8 класів набору даних ODIR [6]. Кількість параметрів:

$$\mathrm{P_{Dense2}} = (128 \times 8) + 8 = 1,032 \text{ ;}$$

Всього параметрів моделі:

$$P_{Total} = 896 + 18,496 + 73,856 + 16,777,344 + 1,032 = 16,871,624 ;$$

Кількість параметрів, що навчаються, є важливою метрикою для розуміння складності моделі, але вона не дає повної картини щодо практичної реалізованості такої моделі. Щоб це зрозуміти потрібно оцінити ключову характеристику - вимоги до відеопам'яті (VRAM).

Всі подальші експерименти в даній роботі будуть проводитися на системі з графічним прискорювачем (GPU) NVIDIA GeForce RTX 3060 Ti, що має 8 ГБ відеопам'яті GDDR6 (VRAM). Цей об'єм будемо вважати основним апаратним обмеженням.

Найчастіше для роботи з нейромережами використовують GPU через значно вищу здатність до паралелізації простих обчислень за рахунок унікальної архітектури, що необхідно для навчання нейромереж адже оновлення ваг і зсувів вимагає обчислення часто мільйонів або більше однакових і відносно простих операцій одночасно, з ціллю значного прискорення навчання, процессор, в свою чергу, більш підходить до складних і послідовних задач і, як правило, не здатен демонструвати можливості паралелізації навіть приблизно подібні GPU [6].

Щоб більш точно оцінити практичну реалізованість моделі на доступному обладнанні, проведемо аналіз її статичних вимог до відеопам'яті (VRAM) під час навчання. Пам'ять, здебільшого, витрачається на зберігання ваг, градієнтів та змінних оптимізатора (Adam або аналоги [31]), використовуючи 32-бітний формат (4 байти) для чисел з плаваючою комою. 32-бітний формат є золотим стандартом і дозволяє створювати моделі з високою точністю роботи за рахунок більшої кількості доступних нулів після коми, але, з метою економії пам’яті, можна зустріти архітектури, що застосовують техніки, які дозволяють створювати мережі на базі 16-бітних, 8-бітних і навіть менших форматів. Ефективність роботи і

навчання мережі при цьому згасає (для 16-бітного варіанту згасання відносно незначні), але економія пам’яті кратно зростає (в 2 рази для 16-бітної моделі). Таким чином, ваги і зсуви моделі потребуватимуть:

$$16,871,624 \text{ параметрів} \times 4 \text{ байти/параметр} \approx 67.5 \text{ МБ}$$

Під час зворотного поширення помилки для кожного параметра обчислюється відповідний йому градієнт, що є числом того ж типу. Таким чином, градієнти потребують аналогічний об'єм пам'яті.

Моменти оптимізатора Adam (зберігає 2 додаткові змінні на параметр):

$$16,871,624 \text{ параметрів} \times 4 \text{ байти/параметр} \times 2 \text{ моменти} \approx 135 \text{ МБ}$$

Відповідно, сумарні мінімальні теоретичні вимоги до VRAM такої моделі на базі 32-бітного формату для зберігання стану моделі складають:

$$\text{VRAM}_{\text{model}} = 67.5 + 67.5 + 135 = 270 \text{ МБ}$$

Важливо зазначити, що цей розрахунок представляє статичну, тобто мінімально необхідну, частину пам'яті. Загальне споживання VRAM під час навчання буде значно вищим, оскільки воно включає динамічні компоненти, головним з яких є пам'ять для збереження проміжних активацій для одного батчу даних, розмір якого залежить від обраного розміру батчу (batch size).

Тим не менш, отримане значення у 270 МБ є незначним у порівнянні з доступними 8 ГБ (8192 МБ). Це залишає значний запас для динамічних компонентів, витрат операційної системи та інших фонових процесів. Таким чином, розрахунок показує, що навчання та розгортання моделей подібної архітектури є цілком реалістичним навіть на відносно доступному споживчому обладнанні. Це відкриває широкі можливості для їх практичного застосування у

медицині, де подібні ефективні інструменти можуть значно підвищити якість та доступність діагностики [1].

Однак, будь-яка, навіть найдосконаліша архітектура нейронної мережі, є лише частиною фінальної системи класифікації. Якість її роботи та здатність до узагальнення напряму залежать від якості даних, на яких вона навчається [5]. Некоректні, зашумлені або не репрезентативні дані можуть повністю нівелювати переваги складної архітектури. Тому, перш ніж переходити до практичної реалізації та навчання моделі, необхідно провести всебічний аналіз та підготовку вихідного набору даних.

2 ПІДГОТОВКА ТА АНАЛІЗ ДАНИХ

2.1 Теорія формування набору даних

2.1.1 Основні вимоги до якості, обсягу та репрезентативності медичних даних

Навчання будь-якої моделі неможливе якщо немає на чому навчатися. Для навчання нейронних мереж використовуються набори даних (dataset), що, для задач комп’ютерного зору, являють собою структурований набір зображень, кожне з яких, тим чи іншим чином (зазвичай окремим файлом розмітки) супроводжується міткою (label) або їх набором, що описують зміст зображення (наприклад, діагноз, стать, вік тощо). У контексті медичної діагностики головні мітки, це, як правило, мітки діагнозів [1]. Сама по собі мережа не має “досвіду” і “знань” з будь якої предметної області, саме тому розмічений набір даних з набору даних слугує джерелом інформації для навчання. У процесі навчання модель ітеративно аналізує, як правило, тисячі прикладів, поступово знаходячи статистичні закономірності, що допомагають пов’язати певні візуальні патерни на зображеннях з відповідними діагнозами. Саме тому обсяг та різноманітність даних мають значний вплив на кінцевий результат навчання. Модель, що була навчена на недостатній кількості або неякісних або неякісно розмічених даних буде більш схильна до проблеми перенавчання, адже буде здатна запам’ятати тренувальні дані замість визначення закономірностей [24]. Така модель не зможе узагальнювати через брак прикладів (навіть людині буває важко зрозуміти нову для неї концепцію виходячи тільки з декількох прикладів без попередніх пояснень). З іншого боку, різноманітна і велика вибірка на тисячі, десятки тисяч або більше елементів змушує модель ігнорувати випадкові артефакти і дає достатньо інформації щоб винайти стабільні, повторювані ознаки, що є ознакою здатності до узагальнення (generalization), тобто здатності працювати з новими, раніше не баченими даними так же як і з тренувальними, адже модель знаходить загальні ознаки класів [6].

Нейромережі, це детерміновані математичні структури, тому вони висувають чіткі вимоги до формату вхідних даних, ігнорування яких може

призвести до катастрофічного погіршення результатів або повної неможливості навіть розпочати процес навчання. Ця проблема схожа на раніше описану проблему нормалізації по батчу з стабілізацією розмірності даних - якщо на вхід мережі будуть приходити не нормовані дані довільних розмірів і інших властивостей, то мережа буде вимушена адаптуватися до вхідних випадковостей в першу чергу і якщо в неї це вийде на певному рівні, то тільки тоді модель зможе розпочати навчання, все ще витрачаючи більшість ресурсів на адаптацію. Саме тому в межах набору даних, або додатковими програмними методами дані завжди приводяться до певного обраного стандарту розміру зображення і даних на ньому. Для стабільної роботи, розмір зображення на вхідному шарі будь-якої моделі повинен бути фіксованим (наприклад 256×256×3), якщо подати іншу розмірність, то в межах сучасних бібліотек може виникнути помилка невідповідності розмірностей (dimension mismatch error). Система не може працювати з зображеннями розміру 512×512×3 якщо запрограмована працювати з 256×256×3 тощо. Привести зображення до потрібного розміру можна за допомогою операцій масштабування або обрізки. Також, зазвичай значення яскравості пікселів на зображеннях подаються у вигляді діапазону, наприклад (0;255). Якщо подати такі сирі великі і різнорідні за масштабом дані на мережу, то це може дестабілізувати процес навчання через вже знайомі проблеми вибуху градієнтів. Тому, стандартною практикою є своєчасна нормалізація - операція перетворення значень пікселів у значно менший і стабільний діапазон розділивши їх на 255 щоб звести діапазон до (0;1), або інші перетворення, наприклад до діапазону (-1;1), що зробить механізм градієнтного спуску більш стабільним та прискорить збіжність моделі [4, 28].

На практиці, набори даних (особливо медичної сфери) рідко бувають ідеальними (надалі побачимо це наочно при аналізі використаного набору даних зображень очного дна) і тому існує низка типових проблем, ігнорування яких може призвести до неефективної або упередженої моделі.

2.1.2 Проблема дисбалансу класів та методи її компенсації

Однією з таких проблем є проблема дисбалансу класів (Class Imbalance) [33]. Вона виникає у тому випадку, коли кількість прикладів для різних класів у межах навчальної вибірки є не рівномірною. Наприклад, логічно, що в медичній практиці, клас “норма” зустрічається в декілька десятків разів частіше ніж клас рідкісного захворювання. Якщо ігнорувати такий ефект, то модель, в процесі оптимізації своєї функції втрат, як правило, приходить до того, що досягає високої точності, наприклад 90%, але взагалі не здатна знаходити рідкісний клас. В такому випадку нейромережа “обирає” шлях найменшого опору і починає передбачати найчастіший клас отримуючи низьку помилку, що є її основною і єдиною задачею і є технічно вірно, але практично не має сенсу якщо модель майже кожного пацієнта класифікує як здорового незважаючи на реальний стан [35]. Серед можливих вирішень дисбалансу можна виділити, наприклад, зважування класів у функції втрат (Weighted Loss) [33]. Цей потужний підхід дозволяє, під час обчислення помилки, надавати більшу вагу (множник коефіцієнт впливу) більш рідкісним класам, або навпаки зменшувати вагу більш частих класів. Відповідно, модель буде отримувати значно більші значення помилки на рідкісних класах, що змусить її приділяти таким класам більшу увагу. Ще одним можливим рішенням можна назвати методи ресемплінгу (Resampling) (рис 2.1) [6].

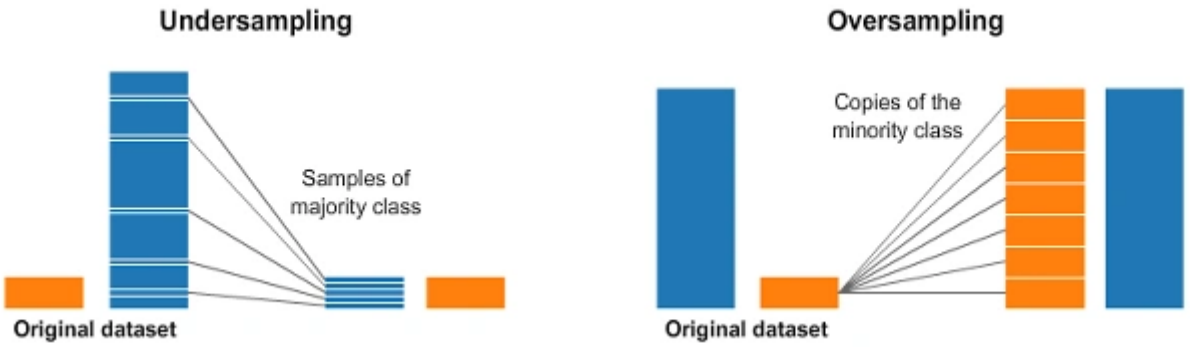


Рисунок 2.1 - Приклад застосування методів ресемплінгу

Тобто змінювати самі дані. Oversampling - штучне дублювання прикладів рідкісних класів. Undersampling - видалення частини прикладів з найчастіших

класів. Проте такий метод може призвести до перенавчання на дублікатах і втрат корисної інформації через видалення. Також можна застосовувати динамічну аугментацію - створювати копії-альтернативні зображення (з трохи іншим масштабом, контрастом, зміщенням, тощо) для рідкісних класів так, щоб вихідна кількість зображень класу порівнювалася з найчастішими [24]. Проте такий метод має свої обмеження через те, що аугментації не створюють принципово нову інформацію, а лише її варіації, через що надмірне використання аугментацій також може призвести до проблем перенавчання.

2.1.3 Аугментація даних

Проблеми недостатнього обсягу даних для навчання можуть виникати не лише в межах певних класів, а і в межах всього набору даних в цілому. Також, залежно від задач, але у випадку надмірної стандартизації, наприклад якщо об’єкт на зображеннях завжди строго посередині також можуть виникати проблеми робастності моделі. Виникає специфічне перенавчання, адже тепер якщо продемонструвати моделі таке ж зображення, але об’єкт знаходиться зліва, а не по центру, то нейромережа не зможе ефективно класифікувати це зображення через те, що ніколи не бачила, що подібне взагалі можливо в межах своєї тренувальної вибірки [6]. Ці проблеми також може допомогти вирішити аугментація даних (рис. 2.2) у випадку якщо збільшувати вибірку не має можливості через дороговизну або брак часу або брак можливостей тощо [24].

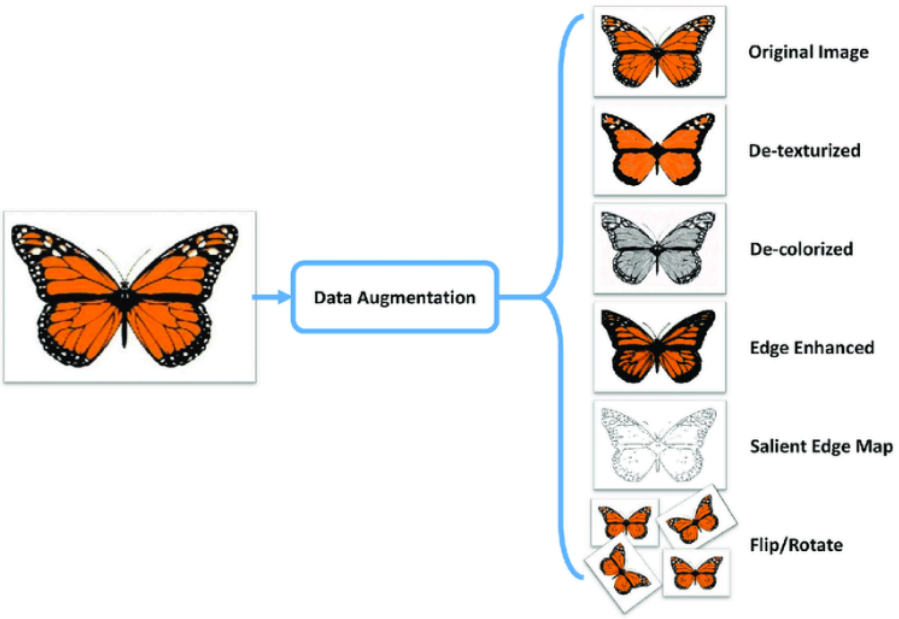


Рисунок 2.2 - Приклад застосування методів аугментації

Аугментація даних, це процес штучного створення нових, відносно унікальних тренувальних прикладів, методом програмного застосування до вже існуючих зображень набору випадкових, але реалістичних трансформацій і збереженням аугментованих прикладів в тренувальну вибірку. До таких трансформацій можна віднести: випадкові повороти, зміни контрасту, яскравості, насиченості, масштабу, зміщення тощо [14]. Таким чином, в межах кожної епохи модель працює з трохи видозміненими версіями оригінальних зображень разом з оригіналами, що змушує її навчатися ігнорувати несуттєві варіації і шуми та фокусуватися на значущих ознаках патологій. Аугментація набору даних вважається однією з технік регуляризації і здатна значно підвищити стійкість та узагальнюючу здатність моделі [6].

2.1.4 Стратегії поділу вибірок

Одна з найфундаментальніших і катастрофічних помилок яку доволі легко допустити під час роботи з нейромережами це кросс-забруднення тренувальних, валідаційних і тестувальних даних. Неможливо оцінити якість моделі на тих самих даних, на яких вона навчалася. В якості аналогії можно привести екзамен на

якому студент завчасно знає всі відповіді адже екзаменаційне завдання в точності дублює домашні. Оцінка моделі у такому випадку не здатна дати жодного уявлення про те, як модель буде поводити себе з новими, раніше не баченими даними. Для адекватної оцінки якості роботи нейромереж використовують вже згадані тренувальну, валідаційну та тестову вибірки, що ніколи не перетинаються (рис. 2.3) [6].

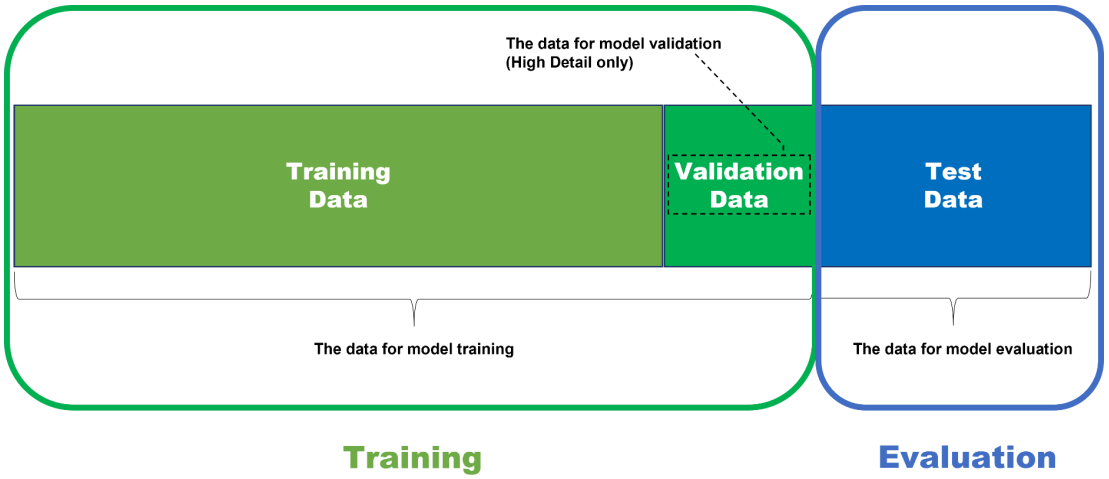


Рисунок 2.3 - Приклад поділу вибірок

В тренувальну вибірку потрапляє найбільша частина даних (часто приблизно 80%) і саме на цих даних модель безпосередньо навчається і налаштовує свої параметри. Валідаційна вибірка займає 10-15% даних, що залишилися після формування тренувальної, для неї виконується лише прямий прохід (forward pass) для оцінки, але не виконується зворотний прохід (backpropagation) і не оновлюються ваги - вона виступає у ролі перевірки проміжкових результатів моделі. Модель при навчанні не враховує валідаційну вибірку для навчання, вона виступає у ролі контролю і проміжкової оцінки помилки подібно контрольним роботам в межах семестру. Саме валідаційні дані роботи мережі часто вважаються основними і використовуються, наприклад, для завчасної зупинки навчання у випадку початку перенавчання коли помилка на навчальній вибірці продовжує падати, а на валідаційний починає зростати. Вона також дозволяє оцінити загальну якість роботи мережі, її архітектури,

налаштувань гіперпараметрів тощо [6]. Тестова вибірка може бути опціональною з точки зору навчання і роботи моделі, адже прямо не впливає на них, але зазвичай використовується для фінального тестування моделі і підтвердження результатів валідації в реальній практиці на реальних даних, особливо в межах дослідницьких робіт подібних цій. Вона обирається як залишок після створення тренувальної і валідаційної виборок. Тестова вибірка ніяк не використовується при навчанні - тільки після нього коли мережа вже повністю готова до роботи кожне зображення тестової вибірки використовується лише один раз для об’єктивної і остаточної перевірки ефективності мережі. Якщо валідація це контрольні, то тестова вибірка це екзаменаційна робота. Формувати вибірки можна як починаючи з тренувальної, так і починаючи з тестової. Якщо починати формування вибірок з тестової, то полегшується контроль відбору даних для тестової та валідаційної вибірок, наприклад якщо в наборі даних мала кількість даних для рідкісного класу і хочемо гарантувати що хоча б одне попаде у тестову вибірку. Формування від тестової вибірки більш доречне при роботі з відносно малими наборами даних, проте, на практиці, обидва варіанти формування вибірок майже однаково ефективні.

Особливу увагу при поділі даних слід приділяти уникненню проблеми витоку даних (data leakage), особливо в медичних задачах. Класичним прикладом такого витоку є ситуація, коли знімки, що належать одному й тому ж пацієнту (наприклад, ліве та праве око, або рентген-знімки, зроблені з різницею в кілька днів), потрапляють у різні вибірки (навчальну та тестову). Оскільки такі дані можуть бути не повністю незалежними (часто маючи певні патології на одному оці вони також є і на іншому, що робить два знімки схожими), модель може “підглянути” інформацію про пацієнта з навчальної вибірки і показати штучно завищені результати на тестовій через надмірну схожість даних (одна і та сама людина зі схожими або майже однаковими ознаками). Тому коректний поділ даних повинен виконуватися на рівні пацієнтів, а не на рівні зображень: всі зображення одного пацієнта повинні знаходитись виключно в одній з вибірок (або в тренувальній, або в валідаційній, або в тестовій) [3].

2.2 Аналіз та підготовка набору даних ODIR

2.2.1 Загальні відомості

Після визначення основних теоретичних принципів роботи з даними, розглянемо та застосуємо їх на практиці до конкретного набору даних, що використовується в цій роботі - Ocular Disease Intelligent Recognition (ODIR), що є публічно доступною на платформі Kaggle [23].

Важливим аспектом при роботі з будь-якими медичними даними є питання анонімності пацієнтів та етики збору даних. Згідно з документацією, набір даних ODIR є анонімізованим, тобто вся інформація, що могла б ідентифікувати особу (імена, точні дати, тощо), була вилучена, а пацієнтам присвоєно унікальні числові ідентифікатори (ID). Це є стандартною і обов'язковою практикою, що дозволяє проводити дослідження, не порушуючи конфіденційності пацієнтів [1]. Окрім того, той факт, що дані були зібрані в різних клініках на різному обладнанні, хоч і створює технічні виклики (варіативність якості), є перевагою з точки зору узагальнюючої здатності. Модель, навчена на таких різноманітних даних, має потенціал бути більш стійкою до варіацій, що зустрічаються в реальній клінічній практиці, ніж модель, навчена на ідеалізованих знімках з одного апарату [6].

З загального огляду можна зрозуміти, що набір даних ODIR-5K є структурованою офтальмологічною базою даних, що містить інформацію про близько 5000 пацієнтів з різних медичних центрів Китаю [22]. Згідно документації, набір даних було сформовано, в тому числі, з метою репрезентації реальних клінічних умов, на що вказують детальність даних, наявність заключень лікарів, той факт що зображення були зроблені на різні камери (Canon, Zeiss, Kowa), тощо (рис. 2.4) [23]. В основі роботи фундус-камери лежить принцип фіксації зображення на цифровий сенсор, подібний до звичайного фотоапарата, вона є вузькоспеціалізованим медичним приладом створеним для вирішення складного оптичного завдання: освітити й сфотографувати очне дно через отвір зіниці, уникаючи при цьому засліплюючих відблисків від рогівки та кришталика. Суть її роботи полягає в унікальній системі освітлення, яка спрямовує світло в око через зовнішнє кільце розширеної зіниці, тоді як зображення, відбите від сітківки,

повертається до камери через її центральну частину. Цей принцип розділення світлових шляхів, поєднаний з мікроскопом низького збільшення, дозволяє отримати чіткі, деталізовані знімки сітківки, диска зорового нерва та судин. Сам процес для пацієнта є швидким та безконтактним, адже голова фіксується на спеціальній підставці для стабільності, а знімок робиться за долі секунди за допомогою яскравого, але безпечного спалаху світла.

Рисунок 2.4 - Приклад апарату для отримання фотографій очного дна

Все це призводить до варіативності в роздільній здатності, якості, точності знімків і навіть наявності неточностей, наприклад, пилу на зображенні або повністю непридатних зображень.

2.2.2 Попередній аналіз

Вся необхідна для роботи інформація міститься у двох основних папках та файлі анотацій. У кореневій папці ODIR-5K знаходяться підпапки тренувальної та валідаційної вибірки зображень та data.xlsx файл анотацій. Проте у конкретній

обраній для роботи варіації цього набору даних, що була завантажена на платформу автором andrewmvd Larxel (Andrew Maranhão), Старшим спеціалістом з обробки даних у лікарні Israelita Albert Einstein, дані додатково структуровані і оброблені [23]. В межах набору даних існує папка preprocessed_images, що супроводжується файлом full_df.csv. В межах папки, зображення, як можна зрозуміти з назви, попередньо оброблені, відсортовані і зведені до розмірів 512x512x3, що значною мірою полегшує подальшу роботу з нейромережами заощаджуючи час на зведенні до стандартів [24]. Однак, як було показано в попередньому розділі при аналізі вимог до VRAM, для нашої базової архітектури такий розмір є надлишковим та обчислювально невиправданим, тому на етапі перед-обробки будемо додатково зменшувати зображення до 256x256x3. Додатково, файл розмітки full_df.csv більш зручний для роботи через код і містить детальну розмітку для кожного пацієнта і ока. Файл розмітки складається зі стовпчиків ID номеру пацієнта, його віку, статі, назв зображень для лівого та правого ока, діагнози лікарів для лівого та правого ока, стовпчики наявності класу патологій для пацієнта (норма, діабетична ретинопатія, глаукома, катаракта, вікова макулярна дегенерація, гіпертонія, патологічна міопія, інші хвороби/аномалії), точний шлях до файлу окремого ока, точна мультикласова символьна мітка, її векторний аналог, назва файлу окремого ока [22]. Звідси можуть виникнути непорозуміння адже кожен рядок має в собі і дані пацієнта і його окремого ока. Це тому, що файл структуровано як дві частини - перша частина враховує стовпчики після стовпчиків міток для пацієнта як інформацію для його правого ока. Перша частина файлу закінчується на ID під номером 4784, після чого знову з 0 починається друга частина, але вже для лівого ока. Для кращого розуміння розберемо два рядки з обох частин файлу для одного пацієнта:

Таблиця 2.1 - Приклад даних для пацієнта 0

Дані з файлу	**Перша частина**	**Друга частина**
ID	0	0
Patient Age	69	69
Patient Sex	female	female
Left-Fundus	*0_left.jpg*	*0_left.jpg*
Right-Fundus	*0_right.jpg*	*0_right.jpg*
Left-Diagnostic Keywords	cataract	cataract
Right-Diagnostic Keywords	normal fundus	normal fundus
N	0	0
D	0	0
G	0	0
C	1	1
A	0	0
H	0	0
M	0	0
O	0	0
filepath	*../input/ocular-disease-recognition-odir5k/ODIR-5K/Training Images/0_right.jpg*	*../input/ocular-disease-recognition-odir5k/ODIR-5K/Training Images/0_left.jpg*

Продовження таблиці 2.1

labels	['N']	['C']
target	"[1, 0, 0, 0, 0, 0, 0, 0]"	"[0, 0, 0, 1, 0, 0, 0, 0]"
filename	*0_right.jpg*	*0_left.jpg*

Як бачимо з таблиці (табл. 2.1): починаючи з стовпчика filepath включно і далі дані відносяться до кожного окремого ока пацієнта, а попередні стовпчики відносяться до самого пацієнта в цілому і співпадають між собою. При цьому, вже на цьому етапі поверхневого аналізу можна помітити неточності. Наприклад конкретно для пацієнта 0 бачимо, що в загальних мітках пацієнта не позначено норму, що дивлячись на інші приклади відповідає певному правилу - якщо у пацієнта є патологія, то навіть при наявності норми в іншому оці норма не помічається, якщо ж у пацієнта немає патологій то помічається норма. Проте той факт, що загальні мітки відповідають пацієнту в цілому, навіть з таким правилом, можна довести якщо поглянути на приклади пацієнтів з різними патологіями на обидвох очах - в такому випадку в загальних мітках помічені обидва класи. Відповідно до такого поверхневого аналізу, можна констатувати, що задача класифікації однозначно мультилейблова [22]. Проте вона також може бути і мультикласовую якщо користуватися наступними стовпчиками. Так, labels та target завжди демонструють лише один клас, навіть за наявності декількох в межах одного ока. В роботі буде поставлена задача мультилейблової класифікації, адже на практиці, що і призваний репрезентувати набір даних, існують випадки одразу декількох патологій в межах одного ока. Проте, як можна побачити, поверхневого аналізу недостатньо для того щоб зробити остаточні висновки по всьому набору даних. Виникають питання: чи точно правило не відображення норми працює на весь набір даних? чи точно колонка target завжди вказує лише на один клас? Чи немає в наборі даних даних з артефактами, що будуть тільки заважати навчанню? Яка точна кількість пацієнтів і зображень в обробленій версії? Чи є в межах

вибірки пацієнти з одним оком? Чи присутній дисбаланс класів? На ці, та інші питання зможемо відповісти створивши скрипт-аналізатор [33].

2.3 Програмний аналіз набору даних та результати

2.3.1 Загальна інформація

Для отримання об'єктивних відповідей на поставлені питання та для фінальної підготовки даних був розроблений скрипт на мові Python з використанням бібліотеки Pandas для аналізу анотацій та OpenCV/Pillow для роботи з зображеннями.

Почнемо з базового огляду файлу за допомогою блоку коду згідно рис. А.1. Результатом виконання цього коду є наступний вивід (рис. 2.5):

```
Файл /home/foleps/Study/Diploma/archive (1)/full_df.csv успішно завантажено.
------------------------------
Базова інформація про структуру даних (df.info()):
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6392 entries, 0 to 6391
Data columns (total 19 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   ID                        6392 non-null   int64
 1   Patient Age               6392 non-null   int64
 2   Patient Sex               6392 non-null   object
 3   Left-Fundus               6392 non-null   object
 4   Right-Fundus              6392 non-null   object
 5   Left-Diagnostic Keywords  6392 non-null   object
 6   Right-Diagnostic Keywords 6392 non-null   object
 7   N                         6392 non-null   int64
 8   D                         6392 non-null   int64
 9   G                         6392 non-null   int64
 10  C                         6392 non-null   int64
 11  A                         6392 non-null   int64
 12  H                         6392 non-null   int64
 13  M                         6392 non-null   int64
 14  O                         6392 non-null   int64
 15  filepath                  6392 non-null   object
 16  labels                    6392 non-null   object
 17  target                    6392 non-null   object
 18  filename                  6392 non-null   object
dtypes: int64(10), object(9)
memory usage: 948.9+ KB
```

Рисунок 2.5 - Результат роботи коду базового огляду файлу

Звідси бачимо, що з наданого виводу df.info() можемо зробити кілька ключових, фундаментальних висновків про структуру та якість нашого набору даних [5]. А саме - у файлі 6392 рядки, що підтверджує той факт, що дані пройшли попередню обробку і були переведені з пацієнт-орієнтованого формату в зображення-орієнтований; Для всіх 19 колонок значення Non-Null Count дорівнює 6392, що означає що у наборі даних немає жодного пропущеного значення, це свідчить про високу якість попередньої підготовки даних набору даних; Числові дані (int64), до яких відносяться ID, Patient Age, а також всі вісім діагностичних колонок (N, D, G, C, A, H, M, O) [22] вже представлені у вигляді чисел (0 або 1), а це означає, що вони готові для використання в моделі без додаткового кодування; Текстові/Об'єктні дані (object), до яких відносяться всі інші колонки, такі як Patient Sex, діагностичні ключові слова, шляхи до файлів, для моделі незрозумілі у сирому вигляді. До того ж, стовпчики labels та target (які виглядають як списки) також мають тип object. Таким чином розуміємо, що Pandas зчитав їх як текст (рядок) і якщо постане задача їх використати, то доведеться додатково їх привести до потрібного вигляду (наприклад, перетворити з рядка "[1, 0, 0, ...]" на реальний масив чисел); Об'єм пам'яті memory usage: 948.9+ KB, підтверджує, що сам файл анотацій є дуже маленьким і легко завантажується в оперативну пам'ять для аналізу.

Таким чином бачимо, що здебільшого набір даних можна вважати якісним і зручним для подальшої роботи, що додатково підтверджується високими оцінками використовуваності на сервісі Kaggle [23].

2.3.2 Аналіз віку та статі пацієнтів

Тепер розглянемо дані стосовно віку і статі пацієнтів за допомогою коду рис А.2. Результатом виконання цього коду є наступний вивід (рис. 2.6 - 2.8):

```
Загальна кількість унікальних пацієнтів: 3358
------------------------------
Графік 'gender_distribution.png' збережено.
------------------------------
Графік 'age_distribution.png' збережено.
------------------------------
Статистичний опис віку пацієнтів (для текстового аналізу):
count    3358.000000
mean       57.889815
std        11.730862
min         1.000000
25%        51.000000
50%        59.000000
75%        66.000000
max        91.000000
Name: Patient Age, dtype: float64
```

Рисунок 2.6 - Результат роботи коду огляду даних стосовно віку і статі пацієнтів текстовий

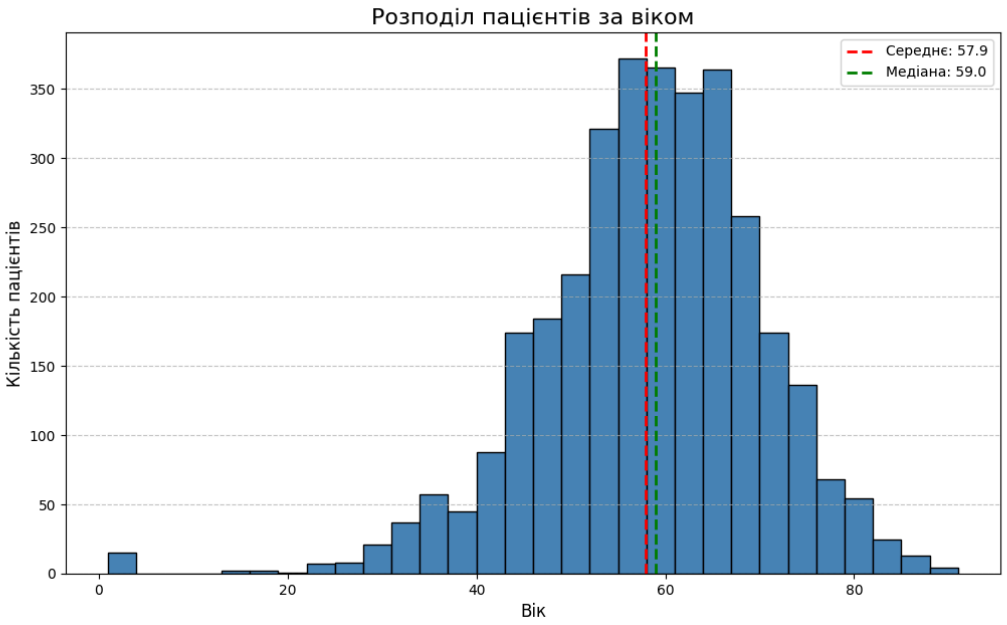


Рисунок 2.7 - Результат роботи коду огляду даних стосовно віку пацієнтів

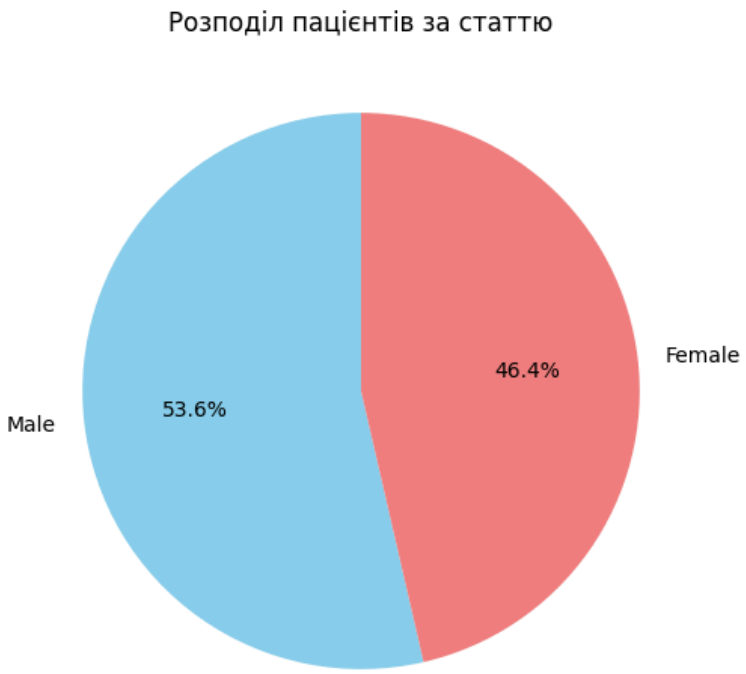


Рисунок 2.8 - Результат роботи коду огляду даних стосовно статі пацієнтів.

Звідси бачимо, що програмний аналіз виявляє розбіжність між заявленою та реальною кількістю пацієнтів. В оригінальному описі набору даних ODIR-5K згадується близько 5000 пацієнтів, а аналіз показує 3358 унікальних пацієнтів [22]. Також варто помітити, що максимальний ID пацієнта у файлі розмітки дорівнює 4784, але при цьому в послідовності ID є пропуски (наприклад, після ID=1 може йти ID=3), що свідчить про те, що автор версії набору даних на Kaggle провів значну роботу з очищення та фільтрації даних [23]. Були вилучені пацієнти, для яких зображення обох очей були непридатними для аналізу (наприклад, через низьку якість) або мали суперечливі анотації, при цьому індексація ID була збережена, тощо. Така фільтрація даних свідчить про те, що працюємо з більш надійним та очищеним набором даних, ніж оригінальний.

Розподіл за статтю, представлений на круговій діаграмі, є майже збалансованим: 53.6% чоловіків та 46.4% жінок. Це свідчить про відсутність

значного гендерного перекосу у вибірці. В задачах медицини, нерівномірна за статтю вибірка може призводити до менш якісних результатів класифікації патологій для менш представленої статі через можливі розбіжності особливостей будови і функціонування організмів різних статей. Боротися з нерівною вибіркою потрібно перш за все на етапі формування такої, адже програмно маємо можливість лише аугментувати дані або використати дублікації і відкидання подібно роботи з проблемами дисбалансу класів, що неможливо застосовувати ефективно при значних розбіжностях [24].

Гістограма розподілу за віком та таблиця статистичного опису здатні додатково поглибити розуміння вибірки. Класичне середнє арифметичне по віку (mean) значення дозволяє оцінити середній вік пацієнта і дорівнює 57.9 років. Але середнє значення може бути чутливим до аномалій викидів (дуже молодих або дуже старих пацієнтів), тому додатково доречно застосувати і інші методи [5]. Наприклад медіана (median, або 50% квартиль), це значення, яке ділить всю вибірку навпіл. Тобто, 59 років - це вік, молодше і старше якого знаходиться рівно по 50% пацієнтів. Медіана є більш стійкою до викидів, ніж середнє, і краще відображає "типового" пацієнта. Додатково застосуємо і пояснимо квантилі. Ква**н**тилі - це точки, що ділять відсортовану вибірку на частини рівного розміру. У стандартному статистичному аналізі найчастіше використовують ква**р**тилі (25%, 50% та 75%), які є трьома точками, що ділять вибірку на чотири рівні частини: 25% квантиль (51 рік) означає, що 25% пацієнтів молодші за 51 рік. 75% квантиль (66 років) - що 75% пацієнтів молодші за 66 років. Додатково, інтерквартильний розмах (Interquartile Range, IQR), здатен вказати на діапазон між 25% та 75% квантилями (від 51 до 66 років). Він означає, в якому віковому діапазоні знаходиться ядро вибірки, тобто центральні 50% пацієнтів. Також графік розподілу кількості пацієнтів по віку демонструє схожий на нормальний розподіл графік, що додатково вказує на правильність даних з огляду на типові вікові групи для розглянутих хвороб.

З огляду на розглянуті показники і графік розподілу бачимо, що основна маса пацієнтів знаходиться у віковому діапазоні від 51 до 66 років, що є

очікуваним і гарною репрезентацію загальної вибірки для набору даних, орієнтованого на вікові офтальмологічні патології. Але окремої уваги заслуговує наявність аномалії в даних - мінімальний вік пацієнта становить 1 рік, що візуально підтверджується невеликим стовпчиком на гістограмі біля нуля і є статистичним викидом. Щоб зрозуміти, чи є це помилкою при внесенні даних, чи реальним клінічним випадком, проведемо його детальний аналіз у наступному блоці.

Проаналізуємо аномалію за допомогою блоку коду згідно рис. А.3. Результатом виконання цього коду є наступний вивід (рис. 2.9):

ID	Patient Age	Patient Sex	Left-Fundus	Right-Fundus	Left-Diagnostic Keywords	Right-Diagnostic Keywords	N	D	G	C	A	H	M	O	filepath	labels	target	filename
1242	1	Female	1242_left.jpg	1242_right.jpg	chorioretinal atrophy	normal fundus	0	0	0	0	0	0	0	1	ir5k/ODI	['N']	[1, 0, 0, 0, 0, 0, 0, 0]	1242_right.jpg
1563	1	Female	1563_left.jpg	1563_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	ir5k/ODI	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1563_right.jpg
1564	1	Female	1564_left.jpg	1564_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	ir5k/ODI	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1564_right.jpg
1565	1	Female	1565_left.jpg	1565_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	ir5k/ODI	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1565_right.jpg
1568	1	Female	1568_left.jpg	1568_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	ir5k/ODI	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1568_right.jpg
1583	1	Female	1583_left.jpg	1583_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	ir5k/ODI	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1583_right.jpg
1588	1	Female	1588_left.jpg	1588_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	ir5k/ODI	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1588_right.jpg
1589	1	Female	1589_left.jpg	1589_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	ir5k/ODI	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1589_right.jpg
1590	1	Female	1590_left.jpg	1590_right.jpg	normal fundus	pathological myopia	0	0	0	0	0	0	1	0	ir5k/ODI	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1590_right.jpg
1595	1	Female	1595_left.jpg	1595_right.jpg	tessellated fundus , peripapillary atrophy	pathological myopia	0	0	0	0	0	0	1	1	ir5k/ODI	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1595_right.jpg
1599	1	Female	1599_left.jpg	1599_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	ir5k/ODI	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1599_right.jpg
1600	1	Female	1600_left.jpg	1600_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	ir5k/ODI	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1600_right.jpg
1603	1	Female	1603_left.jpg	1603_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	ir5k/ODI	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1603_right.jpg
1617	1	Female	1617_left.jpg	1617_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	ir5k/ODI	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1617_right.jpg
1829	1	Female	1829_left.jpg	1829_right.jpg	dry age-related macular degeneration	ıge-related macular degener	0	0	0	0	1	0	0	0	ir5k/ODI	['A']	[0, 0, 0, 0, 1, 0, 0, 0]	1829_right.jpg
1563	1	Female	1563_left.jpg	1563_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	dir5k/OD	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1563_left.jpg
1564	1	Female	1564_left.jpg	1564_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	dir5k/OD	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1564_left.jpg
1565	1	Female	1565_left.jpg	1565_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	dir5k/OD	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1565_left.jpg
1568	1	Female	1568_left.jpg	1568_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	dir5k/OD	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1568_left.jpg
1583	1	Female	1583_left.jpg	1583_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	dir5k/OD	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1583_left.jpg
1588	1	Female	1588_left.jpg	1588_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	dir5k/OD	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1588_left.jpg
1589	1	Female	1589_left.jpg	1589_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	dir5k/OD	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1589_left.jpg
1590	1	Female	1590_left.jpg	1590_right.jpg	normal fundus	pathological myopia	0	0	0	0	0	0	1	0	dir5k/OD	['N']	[1, 0, 0, 0, 0, 0, 0, 0]	1590_left.jpg
1599	1	Female	1599_left.jpg	1599_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	dir5k/OD	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1599_left.jpg
1600	1	Female	1600_left.jpg	1600_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	dir5k/OD	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1600_left.jpg
1603	1	Female	1603_left.jpg	1603_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	dir5k/OD	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1603_left.jpg
1617	1	Female	1617_left.jpg	1617_right.jpg	pathological myopia	pathological myopia	0	0	0	0	0	0	1	0	dir5k/OD	['M']	[0, 0, 0, 0, 0, 0, 1, 0]	1617_left.jpg
1829	1	Female	1829_left.jpg	1829_right.jpg	dry age-related macular degeneration	ıge-related macular degener	0	0	0	0	1	0	0	0	dir5k/OD	['A']	[0, 0, 0, 0, 1, 0, 0, 0]	1829_left.jpg

Рисунок 2.9 - Результат роботи коду аналізу вікової аномалії

Звідси бачимо, що програмний аналіз виявив 28 записів, що належать пацієнтам з віком в 1 рік. Детальний розгляд цих записів (представлений у таблиці) дозволяє зробити однозначний висновок щодо природи цієї аномалії. У переважній більшості цих випадків пацієнтам поставлені діагнози “патологічна міопія” (pathological myopia) та “вікова макулярна дегенерація” (age-related macular degeneration). Обидва ці діагнози є медично несумісними з настільки

малим віком [23]. Вікова макулярна дегенерація (ВМД), як випливає з її назви, є захворюванням, що розвивається внаслідок старіння сітківки. Зазвичай вона діагностується у людей старше 50 років і є однією з основних причин втрати зору в літньому віці, а патологічна міопія є важкою формою короткозорості, що характеризується прогресуючим розтягненням очного яблука та дегенеративними змінами на очному дні. І хоча міопія може бути вродженою, її патологічна форма розвивається протягом багатьох років, зазвичай у підлітковому та/або дорослому віці.

Наявність таких діагнозів у записах з віком в 1 рік може вказувати на те, що це є однозначною помилкою при внесенні даних. Найбільш імовірним сценарієм є помилка вводу, наприклад, була пропущена друга цифра або невірно введена перша тощо.

Оскільки вік пацієнта не використовується як вхідна ознака (feature) для навчання нашої моделі, а служить лише для загального опису набору даних, ця помилка не може вплинути на технічну якість навчання. До того ж, навіть з використанням помилкових даних, кількість в 28 записів не здатна створити помітного дегенеративного ефекту для моделі через раніше розглянуті принципи роботи нейромереж і наборів даних. Всі інші дані в цих записах (зображення, діагностичні ключові слова, бінарні мітки тощо) виглядають консистентними, тому було прийнято рішення не вилучати ці записи з набору даних, але враховувати можливу наявність таких чи інших помилок при загальній оцінці якості даних [5].

2.3.3 Пошук пацієнтів з записом про лише одне око

Тепер знайдемо пацієнтів, що мають запис лише для одного ока (лише один ID), якщо такі є, за допомогою коду рис. А.4. Результатом виконання цього коду є наступний вивід (рис. 2.10):

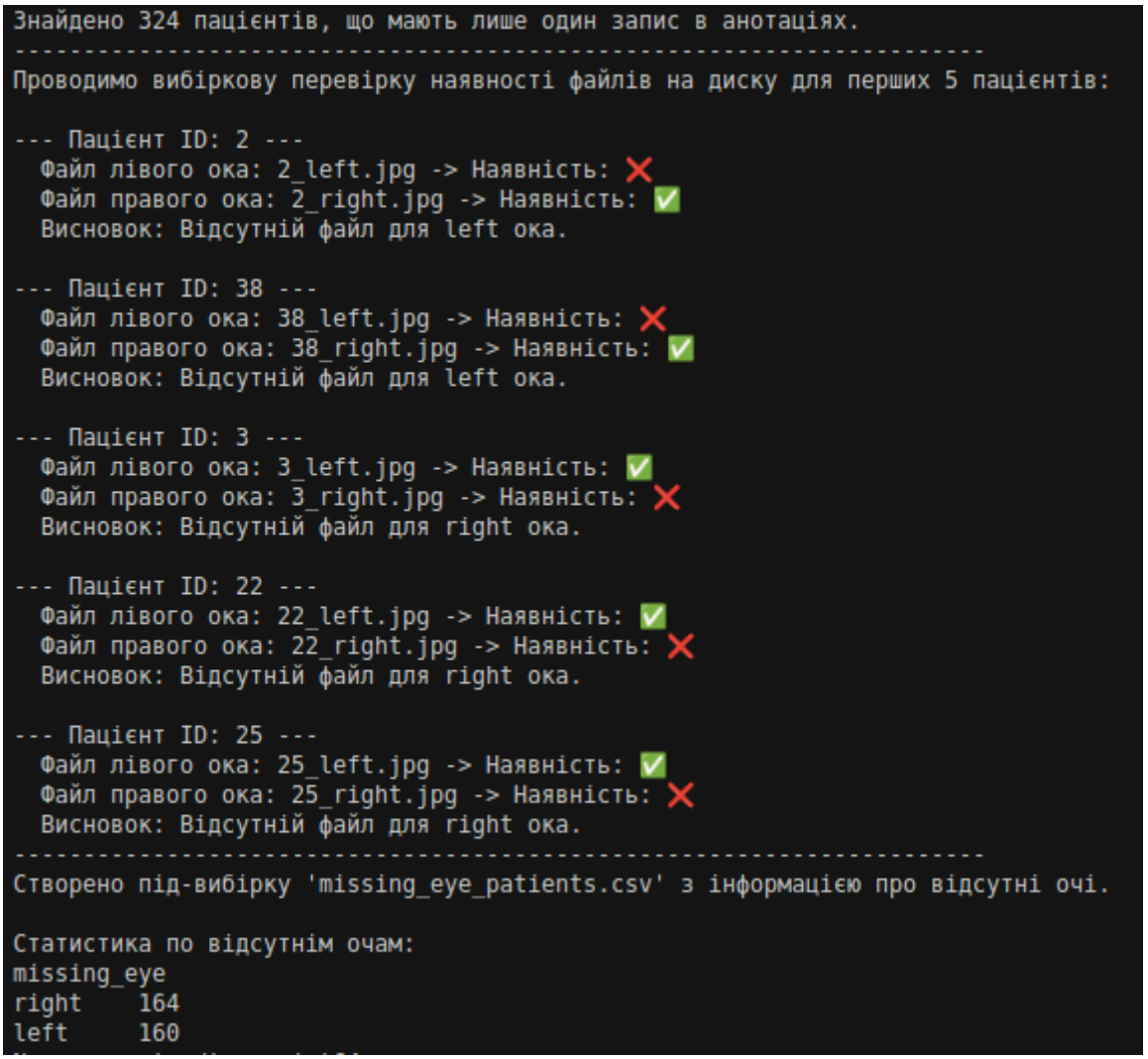


Рисунок 2.10 - Результат роботи коду аналізу пацієнтів, що мають запис лише для одного ока

Звідси бачимо, що програмний аналіз підтверджує наявність в наборі даних 324 пацієнтів, для яких в файлі анотацій full_df.csv присутній лише один запис [23]. Це складає значні 9.6% від загальної кількості унікальних пацієнтів.

Для з'ясування причини цієї особливості було проведено додаткову перевірку, а саме - для кожного з цих 324 пацієнтів скрипт перевірив наявність

файлів зображень для обох очей на диску в папці попередньо оброблених даних (preprocessed_images), з якою працюємо. Результати вибіркової перевірки демонструють чітку закономірність: для цих пацієнтів на диску фізично присутній лише один з двох файлів у обраній папці. Наприклад, для пацієнта ID=2 існує файл 2_right.jpg, але відсутній 2_left.jpg. Якщо звернутися до папки оригінальних не оброблених зображень (Training Images), то у файлі 2_left.jpg (рис. 2.11)можна побачити, що зображення значно затемнене, має значну зашумленість, плями і закриваючі частину зображення затемнення, що робить його не читаємим для нейромереж і важко читаємим для людей.

Рисунок 2.11 - Приклад відкинутого в оригінальному наборі даних ока

Це дозволяє зробити висновок, що “одноокі” пацієнти в нашому наборі даних не є помилкою в анотаціях, а є результатом свідомого процесу очищення даних, проведеного автором [23]. В ході цього процесу зображення, що були визнані неінформативними або повністю нечитабельними (як раніше бачили на прикладі оригінального 2_left.jpg), були фізично видалені з фінального обробленого набору даних. Статистика показує, що було видалено 164 лівих та 160 правих очей, що свідчить про відсутність системного перекосу.

Це відкриття значно підвищує довіру до якості всього набору даних, оскільки підтверджує, що він пройшов ретельну кураторську перевірку. Це також остаточно обґрунтовує підхід до роботи з кожним зображенням як з окремим зразком і підкреслює важливість поділу даних на рівні пацієнтів, з метою уникнення витоку даних [3].

2.3.4 “Правило норми”

Тепер перевіримо гіпотезу раніше згаданого “правила норми” - якщо в мітках патологій пацієнта хоча б для одного ока є хоча б одна мітка патології, то мітка норми для пацієнта не ставиться, навіть якщо інше око має запис “норма” [22]. Для цього скористаємося блоком коду рис. А.5. Результатом виконання цього коду є наступний вивід (рис. 2.12):

```
Перевірка 'Правила Норми'...
--------------------------------------------------
Результат: Правило підтверджено для всього датасету.
У жодного пацієнта з наявністю патології (D,G,C,A,H,M,O) не стоїть загальна мітка 'Норма' (N=1).
```

Рисунок 2.12 - Результат роботи коду аналізу “правила норми”

Звідси бачимо, що програмний аналіз повністю підтвердив гіпотезу для всього набору даних [23]. Не було знайдено жодного випадку, де у пацієнта одночасно була б мітка будь-якої патології (D, G, C, A, H, M, O) і загальна мітка “норма” (N). Тобто, з цього випливає, що клас “норма” (N) є взаємовиключним по

відношенню до всіх інших класів патологій. Що означає, що мітка N=1 використовується для позначення виключно тих пацієнтів, у яких відсутні будь-які патології на обох очах [22]. Відповідно, тепер розуміємо, що мітки класів пацієнта формуються за власними правилами, що робить їх менш надійними для задач мультилейблової класифікації [4].

2.3.5 Визначення типу задачі

Для того, щоб остаточно визначити природу задачі (мультикласова чи мультилейблова) на рівні окремого ока, необхідно проаналізувати безпосередньо текстові діагнози лікарів, для кожного ока. Якщо існують випадки, коли для одного ока вказано кілька патологій, це однозначно свідчить про мультилейблову природу задачі.

2.3.5.1 Аналіз діагностичних описів

Для перевірки цієї гіпотези створимо скрипт (рис. А.6), що аналізує діагностичні описи кожного з 6392 зображень на наявність коми, яка використовується для перерахування кількох станів [23]. Результатом виконання цього коду є наступний вивід (рис. 2.13):

```
Аналіз наявності кількох діагнозів для одного ока
Знайдено 548 зображень (з 6392 проаналізованих, що складає 8.57%), де в текстовому описі є кома.

Перші 10 прикладів таких записів:
  - moderate non proliferative retinopathy, epiretinal membrane
  - moderate non proliferative retinopathy, hypertensive retinopathy
  - moderate non proliferative retinopathy, abnormal pigment
  - macular epiretinal membrane, moderate non proliferative retinopathy
  - dry age-related macular degeneration, glaucoma
  - moderate non proliferative retinopathy, laser spot
  - moderate non proliferative retinopathy, pathological myopia
  - macular epiretinal membrane, diabetic retinopathy
  - wet age-related macular degeneration, diabetic retinopathy
  - moderate non proliferative retinopathy, cataract
----------------------------------------------------------------------
```

Рисунок 2.13 - Результат роботи коду аналізу наявності декількох патологій в межах одного ока

Звідси бачимо, що програмний аналіз виявив 548 зображень (що складає 8.57% від усіх проаналізованих), де в текстовому описі для одного ока присутня кома, яка розділяє кілька діагностичних термінів. Приклади, виведені скриптом, демонструють випадки, такі як dry age-related macular degeneration, glaucoma або moderate non proliferative retinopathy, hypertensive retinopathy, що вказує на мультилейбловий характер. Цей результат є прямим доказом того, що задача класифікації окремого ока в даному наборі даних є мультилейбловою. Спрощення задачі до мультикласової призвело б до втрати значної частини діагностичної інформації, що є неприпустимим в межах реальної медичної практики [1]. Варто зазначити, що метод підрахунку за наявністю коми не є надійною оцінкою. Теоретично, частина з цих 548 випадків може бути перерахуванням синонімічних термінів або різних стадій однієї й тієї ж хвороби (наприклад, diabetic retinopathy, mild nonproliferative retinopathy). Не маючи карти правил, не можемо точно це перевірити. Проте, навіть приблизний аналіз прикладів показує наявність великої кількості випадків з різними патологіями. Це дозволяє обґрунтувати вибір розробляти саме мультилейблову архітектуру. Подальший аналіз дозволить отримати точну кількість реальних мультилейблових випадків після очищення даних.

2.3.5.2 Стовпчики label та target

Розглянемо стовпчики label та target. припускаємо, що ці стовпчики відповідають лише одному класу незалежно від того, чи наявні кілька класів одночасно чи ні. Якщо ця гіпотеза підтвердиться, то ці стовпчики будемо вважати відкинутими, адже вони не підходять під обрану задачу мультилейблової класифікації. Перевіримо за допомогою блоку коду рис. А.7. Результатом виконання цього коду є наступний вивід (рис. 2.14):

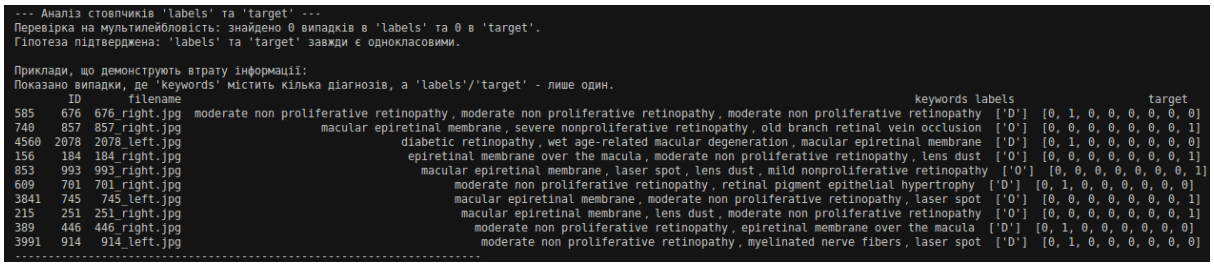


Рисунок 2.14 - Результат роботи коду аналізу стовпчиків label та target

Звідси бачимо, що програмний аналіз показує, що гіпотеза про їхню однокласову (single-label) природу була повністю підтверджена - перевірка не виявила жодного випадку, де б ці стовпчики містили більше однієї мітки [23]. Це справедливо навіть для тих зображень, де відповідний текстовий опис keywords містить перерахування кількох діагностичних термінів. Таким чином, ці стовпчики представляють собою спрощену, однокласову інтерпретацію потенційно мультилейблових даних [5]. У всіх наведених випадках, значення в labels та target відповідає першому діагностичному терміну, вказаному в рядку keywords. Наприклад, для зображення 857_right.jpg з текстовим описом macular epiretinal membrane, severe nonproliferative retinopathy, old branch retinal vein occlusion, у фінальну мітку потрапляє лише macular epiretinal membrane. Аналогічно, для 2078_left.jpg з опису diabetic retinopathy, wet age-related macular degeneration, macular epiretinal membrane обирається лише diabetic retinopathy. Хоча логіка вибору є детермінованою, її застосування для навчання моделі є методологічно невірним. Такий підхід вибору першого діагнозу і відкидання інших призводить до систематичної втрати діагностичної інформації по всім іншим патологіям, що присутні на зображенні. У мультилейблових випадках, модель ніколи не отримає сигнал про наявність інших класів для одного і того ж зображення, що є значним спотворенням клінічної картини [1]. Для створення ефективного діагностичного інструменту, здатного розпізнавати стани з декількома патологіями одночасно, такий підхід є шкідливим. Тобто, враховуючи однокласову природу та логіку, що призводить до втрати даних, єдиним

методологічно коректним рішенням є повна відмова від використання стовпчиків labels та target для навчання моделі.

З іншого боку використання міток пацієнта не дозволить працювати з окремим оком і має проблему “правила норми”. Це правило може бути корисним у задачах класифікації відносно пацієнта, адже тоді логічно припустити, що якщо хоча б одне око має патологію, то навіть якщо інше здорове весь пацієнт все одно не може вважатися здоровим, тобто клас норми не проставляється. У випадку задачі класифікації по окремому оку, це правило, як і використання пацієнт-орієнтованих даних в цілому, може бути шкідливим через зашумлення і втрату інформації - мережа почне отримувати мітки патологій для очей, в яких їх немає, наприклад [4].

2.3.6 Аналіз письмових діагнозів

Спробуємо звернутися до письмових описів патологій для кожного ока в стовпчиках diagnostic keywords. Ці стовпчики являють собою не оброблені діагнози лікарів для кожного окремого ока, тому для початку потрібно віднайти і розглянути всі унікальні записи, що існують в наборі даних за допомогою блоку коду рис А.8 [23]. Результатом виконання цього коду є наступний вивід (рис. 2.15):

```
Аналіз унікальних діагнозів, вписаних лікарями...
------------------------------------------------------------
Знайдено 94 унікальних записів.
------------------------------------------------------------
Повний словник унікальних діагнозів:
abnormal pigment                                 age-related macular degeneration                 anterior segment image
arteriosclerosis                                 atrophic change                                  atrophy
branch retinal artery occlusion                  branch retinal vein occlusion                    cataract
central retinal artery occlusion                 central retinal vein occlusion                   central serous chorioretinopathy
chorioretinal atrophy                            chorioretinal atrophy with pigmentation proliferation  choroidal nevus
congenital choroidal coloboma                    depigmentation of the retinal pigment epithelium  diabetic retinopathy
diffuse chorioretinal atrophy                    diffuse retinal atrophy                          drusen
dry age-related macular degeneration             epiretinal membrane                              epiretinal membrane over the macula
fundus laser photocoagulation spots              glaucoma                                         glial remnants anterior to the optic disc
hypertensive retinopathy                         idiopathic choroidal neovascularization          image offset
intraretinal hemorrhage                          intraretinal microvascular abnormality           laser spot
lens dust                                        low image quality                                macular coloboma
macular epiretinal membrane                      macular hole                                     macular pigmentation disorder
maculopathy                                      mild nonproliferative retinopathy                moderate non proliferative retinopathy
morning glory syndrome                           myelinated nerve fibers                          myopia retinopathy
myopic maculopathy                               no fundus image                                  normal fundus
old branch retinal vein occlusion                old central retinal vein occlusion               old chorioretinopathy
old choroiditis                                  optic disc edema                                 optic discitis
optic disk epiretinal membrane                   optic disk photographically invisible            optic nerve atrophy
oval yellow-white atrophy                        pathological myopia                              peripapillary atrophy
pigment epithelium proliferation                 pigmentation disorder                            post laser photocoagulation
post retinal laser surgery                       proliferative diabetic retinopathy               refractive media opacity
retina fold                                      retinal detachment                               retinal pigment epithelial hypertrophy
retinal pigment epithelium atrophy               retinal pigmentation                             retinal vascular sheathing
retinitis pigmentosa                             retinochoroidal coloboma                         severe nonproliferative retinopathy
severe proliferative diabetic retinopathy        silicone oil eye                                 spotted membranous change
suspected abnormal color of  optic disc          suspected cataract                               suspected diabetic retinopathy
suspected glaucoma                               suspected macular epimacular membrane            suspected microvascular anomalies
suspected moderate non proliferative retinopathy  suspected retinal vascular sheathing            suspicious diabetic retinopathy
tessellated fundus                               vessel tortuosity                                vitreous degeneration
vitreous opacity                                 wedge white line change                          wet age-related macular degeneration
white vessel
```

Рисунок 2.15 - Результат роботи коду аналізу кількості унікальних діагностичних записів

Звідси бачимо, що маємо 94 унікальних записів відповідно до яких формувалися мітки класів. Цей список містить як чіткі назви патологій (наприклад, cataract, glaucoma), так і більш описові терміни (vessel tortuosity), ознаки лікування (laser spot) та технічні артефакти (low image quality, no fundus image). Така кількість типів є надто детальною і зашумленою для прямого використання в якості класів для моделі, наприклад, терміни mild nonproliferative retinopathy, moderate non proliferative retinopathy та proliferative diabetic retinopathy по суті відносяться до одного й того ж захворювання діабетичної ретинопатії (D), але вказують на різні його стадії [22]. Окремої уваги в цьому списку заслуговують записи, що не є діагнозами, а описують технічний стан зображення, наприклад: low image quality, no fundus image, lens dust, image offset. Це створює важливе методологічне питання: чи варто включати такі зображення в процес навчання? З одного боку, їх виключення дозволить працювати з більш чистими даними, змушуючи модель фокусуватися виключно на біологічних патернах захворювань підвищуючи ефективність моделі на якісних даних [4]. А з іншого боку, якщо таких зображень відносно небагато, то їх можна включити до класу “інше” (O), що потенційно здатне зробити модель більш стійкою до неідеальних зображень у реальній практиці продемонструвавши моделі, що не ідеальні зображення існують і повинні бути віднесені до класу (O), замість того щоб навчатися класифікувати нерозбірливі зображення по основним класам. Прийняття обґрунтованого рішення неможливе без кількісної оцінки. Якщо кількість зображень, що містять виключно технічні артефакти (і не мають інших діагностичних ознак), є значною, їх включення може забруднити клас “інше” і навчити модель хибним кореляціям для інших класів. Якщо ж їх кількість відносно невелика, то їх вплив буде або незначним або навіть корисним. Тому, перш ніж створювати фінальну карту правил, наступним кроком буде програмний аналіз для відповіді на це питання: скільки зображень у наборі даних містять виключно технічні артефакти? Для цього спочатку визначимо список таких слів-артефактів, а потім підрахуємо їхню кількість у наборі даних. Для цього скористаємося блоком коду згідно рис. А.9.1 та рис. А.9.2. Результатом виконання цього коду є наступний вивід (рис. 2.16):

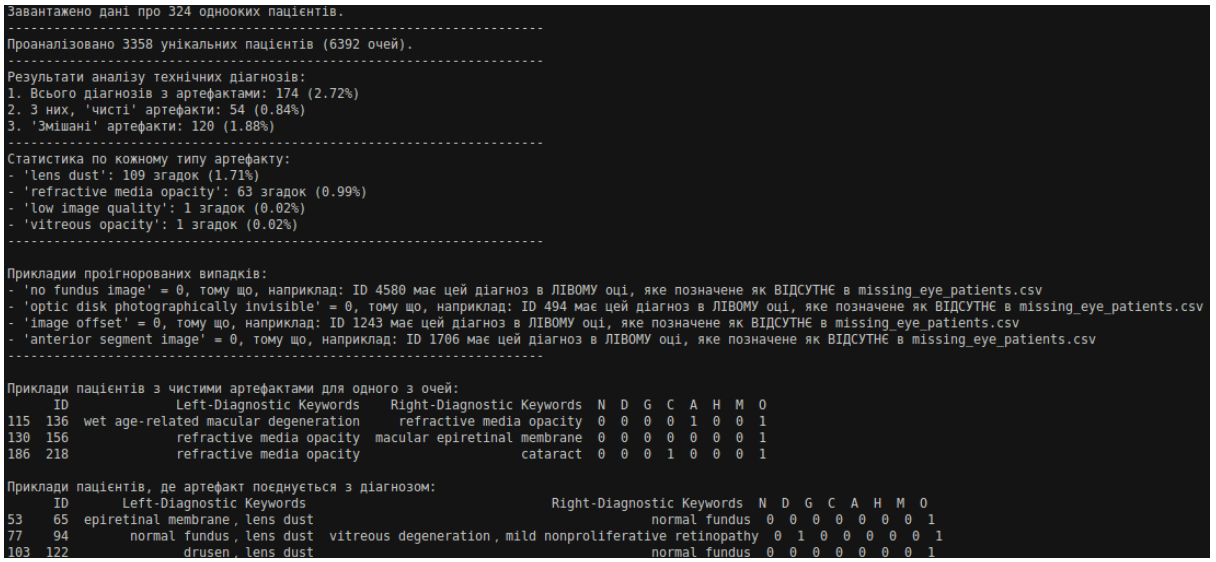


Рисунок 2.16 - Результат роботи коду аналізу технічних артефактів

Звідси бачимо, що програмний аналіз, що враховує лише фізично наявні зображення в обробленій частині набору даних, дає фінальну картину поширеності технічних артефактів. Загалом було виявлено 174 зображення (2.72% від проаналізованих 6392 очей), в діагностичних описах яких присутні терміни, що вказують на проблеми з якістю знімку. Ці 174 випадки розподіляються на дві ключові групи, а саме зображення, що містять виключно технічні артефакти (було виявлено 54 (0.84%) таких випадків). Як видно з прикладів, це випадки, де єдиним описом є, наприклад, refractive media opacity, тобто суто технічні мітки. Такі зображення є діагностично неінформативними щодо патологій. Інша група включає в себе зображення, на яких артефакт йде разом з реальним діагнозом (було виявлено 120 (1.88%) таких випадків). Приклади показують, що, незважаючи на наявність артефакту (наприклад, lens dust), лікар все ж зміг поставити діагноз (наприклад, epiretinal membrane). Такі зображення є діагностично цінними. Окремо варто звернути увагу на статистику по кожному типу артефактів. Були знайдені згадки лише про lens dust (109 випадків), refractive media opacity (63), low image quality (1) та vitreous opacity (1). Інші діагнози були присвоєні тим очам, які були видалені з набору даних автором під час очищення, і скрипт їх коректно проігнорував щоб не спотворювати статистичні дані, адже працювати будемо з передобробленими даними. Виходячи з цього аналізу, було

прийнято рішення про те, що 54 зображення з суто технічними мітками будуть виключені з навчальної вибірки, адже їх кількість відносно мала і включення їх у навчання було б шкідливим, оскільки це змусило б модель вчитися на діагностично неінформативних прикладах [5]. До того ж 54 випадка не достатньо для того щоб надійно навчити мережу відносити аномалії до певного класу. Проте 120 змішаних за мітками зображень будуть залишені в наборі даних, оскільки вплив на робастність моделі завдяки таким зображенням також буде не значним, але вони містять цінну медичну інформацію.

2.3.7 Створення карти правил (мапінг)

Наступним кроком є створення фінальної карти правил (мапінгу) для перетворення діагностичних описів у цільові мультилейблові мітки.

Проведений аналіз продемонстрував, що існуючі в наборі даних мітки є непридатними для вирішення поставленої задачі мультилейблової класифікації окремого зображення [23]. Єдиним достовірним джерелом інформації залишаються текстові описи лікарів, які раніше були декомпозивані на 94 унікальні терміни. Наступним логічним етапом є перетворення цього неструктурованого набору термінів у формалізований, придатний для машинного навчання вигляд. Для цього необхідно виконати мапінг (mapping), тобто процедуру встановлення відповідності між кожним діагностичним терміном та одним з восьми цільових класів (N, D, G, C, A, H, M, O) [22]. Процедура мапінгу базується на принципі агрегації за нозологічною сутністю, тобто віднесенням різних проявів та стадій в рамках єдиного захворювання до одного класу. Наприклад, терміни, що описують різні стадії діабетичної ретинопатії (mild/moderate/severe nonproliferative retinopathy), агрегуються під єдиним узагальнюючим класом D (діабет). Аналогічно, терміни glaucoma та cataract (та їхні похідні з suspected) прямо відносяться до класів G та C. До класу A (вікова макулярна дегенерація) відносяться її різні форми (dry/wet). Класи H (Гіпертонія) та M (Міопія) формуються за тим же принципом, об'єднуючи терміни hypertensive

retinopathy та pathological myopia з їхніми синонімами [22]. Єдиний термін normal fundus однозначно відповідає класу N (норма). Інші терміни, що описують чітко визначені патології, але не входять до основних семи класів (наприклад, macular hole, retinal detachment), а також неспецифічні морфологічні знахідки (vessel tortuosity), наслідки лікування (laser spot) та інші стани, відносяться до узагальнюючого класу O (інше). Окремої уваги потребує обробка термінів, що описують технічні артефакти. Як було встановлено, існує дві групи таких зображень: чисто технічні та змішані. Відповідно, при генерації фінальних міток будемо вилучати і ігнорувати зображення якщо воно є чисто технічним (містить лише артефакти і не містить жодних патологій). І навпаки, якщо зображення є змішаним (артефакт супроводжує реальний діагноз), воно залишається в наборі даних, а сам термін-артефакт при мапінгу буде віднесено до класу “інше” (O). Такий підхід дозволяє очистити дані від діагностично неінформативних прикладів, зберігши при цьому цінні зображення, зроблені в неідеальних умовах [5].

Програмна реалізація даної карти правил дозволить перейти до фінального етапу підготовки даних, а саме автоматизованої генерації коректного мультилейблового вектору цільових міток для кожного зображення в наборі даних. Створимо таку карту (рис. 2.17):

```python
# Створення та автоматична перевірка фінальної карти правил
keyword_to_label_map = {
    'N': [
        'normal fundus'
    ],
    'D': [
        'diabetic retinopathy', 'mild nonproliferative retinopathy',
        'moderate non proliferative retinopathy', 'severe nonproliferative retinopathy',
        'proliferative diabetic retinopathy', 'suspicious diabetic retinopathy',
        'suspected diabetic retinopathy', 'severe proliferative diabetic retinopathy',
        'suspected moderate non proliferative retinopathy'
    ],
    'G': [
        'glaucoma', 'suspected glaucoma'
    ],
    'C': [
        'cataract', 'suspected cataract'
    ],
    'A': [
        'age-related macular degeneration', 'dry age-related macular degeneration',
        'wet age-related macular degeneration'
    ],
    'H': [
        'hypertensive retinopathy'
    ],
    'M': [
        'pathological myopia', 'myopia retinopathy', 'myopic maculopathy'
    ],
    'O': [
        'epiretinal membrane', 'macular hole', 'maculopathy', 'retinal detachment',
        'chorioretinal atrophy', 'retinitis pigmentosa',
        'branch retinal vein occlusion', 'central retinal vein occlusion', 'old branch retinal vein occlusion',
        'old central retinal vein occlusion', 'branch retinal artery occlusion', 'central retinal artery occlusion',
        'vitreous degeneration', 'myelinated nerve fibers', 'optic nerve atrophy',
        'optic discitis', 'morning glory syndrome',
        'retinal vascular sheathing', 'vessel tortuosity', 'pigmentation disorder',
        'retinal pigmentation', 'abnormal pigment', 'tessellated fundus', 'atrophy',
        'old chorioretinopathy', 'intraretinal hemorrhage', 'retina fold', 'choroidal nevus',
        'macular epiretinal membrane', 'old choroiditis', 'optic disc edema',
        'retinal pigment epithelial hypertrophy',
        'spotted membranous change', 'atrophic change', 'chorioretinal atrophy with pigmentation proliferation',
        'congenital choroidal coloboma', 'depigmentation of the retinal pigment epithelium',
        'glial remnants anterior to the optic disc', 'idiopathic choroidal neovascularization',
        'intraretinal microvascular abnormality', 'macular coloboma', 'macular pigmentation disorder',
        'optic disk epiretinal membrane', 'oval yellow-white atrophy', 'peripapillary atrophy',
        'pigment epithelium proliferation', 'retinochoroidal coloboma', 'suspected macular epimacular membrane',
        'suspected microvascular anomalies',
        'suspected retinal vascular sheathing', 'wedge white line change', 'white vessel',
        'arteriosclerosis', 'central serous chorioretinopathy', 'retinal pigment epithelium atrophy',
        'diffuse chorioretinal atrophy', 'diffuse retinal atrophy', 'epiretinal membrane over the macula',
        'suspected abnormal color of  optic disc', 'laser spot', 'post retinal laser surgery',
        'silicone oil eye', 'fundus laser photocoagulation spots', 'post laser photocoagulation',
        'low image quality', 'image offset', 'lens dust', 'no fundus image',
        'optic disk photographically invisible', 'refractive media opacity',
        'anterior segment image', 'vitreous opacity', 'drusen'
    ]
}
```

Рисунок 2.17 - Карта правил відношення діагностичного запису до мітки класу

Так як карта була створена вручну, було проведено додаткову автоматизовану перевірку її повноти та консистентності. Для цього було написано скрипт (рис. А.10), що порівнює терміни з карти з оригінальним списком 94 унікальних діагнозів, перевіряє наявність дублікатів, відсутність елементів, наявність зайвих елементів. Результатом виконання цього коду є наступний вивід (рис. 2.18):

```
Перевірка карти правил пройшла успішно.
Статистика розподілу діагнозів по класах:
--------------------------------------------
Клас 'N': віднесено 1 унікальних термінів.
Клас 'D': віднесено 9 унікальних термінів.
Клас 'G': віднесено 2 унікальних термінів.
Клас 'C': віднесено 2 унікальних термінів.
Клас 'A': віднесено 3 унікальних термінів.
Клас 'H': віднесено 1 унікальних термінів.
Клас 'M': віднесено 3 унікальних термінів.
Клас 'O': віднесено 73 унікальних термінів.
--------------------------------------------
Загальна кількість правил у карті: 94
```

Рисунок 2.18 - Результат роботи коду аналізу порівняння карти правил

Звідси бачимо, що перевірка пройшла успішно, що підтверджує повноту та коректність створеної карти правил. Загальна кількість правил у карті (94) точно відповідає кількості унікальних діагнозів, знайдених у даних, що гарантує врахування кожного терміну. Статистика показує розподіл цих 94 термінів по восьми цільових класах. Очікувано, найбільша кількість унікальних діагнозів (73) була віднесена до узагальнюючого класу “інше” (O), тоді як інші класи об'єднали від 1 до 9 синонімічних термінів.

2.4 Створення власного набору міток

Тепер, маючи карту правил, можемо автоматично генерувати цільові мітки та створити чистий набір даних для навчання моделі відповідно до поставлених задач. Для цього створимо скрипт (рис. А.11.1 та рис. А.11.2), що буде для кожного запису в full_df.csv, вибирати ID пацієнта, стать, вік, назву файла відповідного ока і формувати вектор ознак відповідно до карти на основі діагнозів відповідного ока у оригінальному файлі, після чого збереже новий сформований файл даних. Додатково, не врахуємо “суто технічні” зображення [5]. Результатом виконання цього коду є новий файл даних final_dataset.csv та наступний вивід (рис. 2.19):

```
Файл 'final_dataset.csv' успішно створено.
--------------------------------------------------
Статистика процесу:
Початкова кількість зображень: 6392
Відфільтровано 'чисто технічних' зображень: 54
Фінальна кількість зображень у датасеті: 6338
--------------------------------------------------
```

Рисунок 2.19 - Результат роботи коду створення нового набору даних

Звідси бачимо, що процес генерації фінального набору даних пройшов успішно. З початкових 6392 записів було ідентифіковано та відфільтровано 54 зображення, що містили виключно технічні артефакти і були визнані діагностично неінформативними. Таким чином, фінальний, очищений набір даних final_dataset.csv тепер містить 6338 записів, кожен з яких представляє собою валідне зображення з коректно згенерованими мітками. Структура нового набору даних була свідомо розроблена для вирішення як поточної, так і потенційних майбутніх задач. Вона включає наступні поля:

filename як назва файлу зображення, що є основним ідентифікатором, який пов'язує запис у таблиці з конкретним файлом на диску, що є необхідним для завантаження даних у нейронну мережу.

N, D, G, C, A, H, M, O - вісім бінарних колонок міток, що представляють згенерований мультилейбловий вектор. Це цільові змінні (targets), які модель буде вчитися передбачати [23].

ID (Ідентифікатор пацієнта), що є критично важливим для коректного поділу даних на навчальну, валідаційну та тестову вибірки [6]. Як було обґрунтовано раніше, поділ повинен відбуватися на рівні пацієнтів, щоб уникнути витоку даних, і так як ID дублюється до одного і того ж пацієнта (за виключенням пацієнтів з записом лише про одне око), то саме ця колонка буде використана для коректного поділу.

Демографічні дані пацієнта Patient Age, Patient Sex хоча і не використовуються як вхідні ознаки (features) в рамках нашої базової моделі, проте збереження цих даних в фінальному наборі даних є важливим з точки зору майбутніх досліджень. Наприклад, у більш просунутих архітектурах ці дані можна

використовувати як додаткові входи для моделі, або для проведення аналізу якості роботи моделі на різних демографічних групах (наприклад, відповідаючи на питання “чи є точність однаковою для чоловіків і жінок?” або “яка вікова група найбільш схильна до певного діагнозу?”) [1].

Таким чином, створений final_dataset.csv можна вважати гнучким інструментом для подальших експериментів.

2.4.1 Валідація нового набору даних

Але, перш ніж допускати новий файл анотацій у роботу, необхідно провести повторну валідацію цього нового набору даних, щоб переконатися у його цілісності та відповідності оригінальним даним та нашим очікуванням [5]. Для цього скористаємося скриптом (рис. А.12), що порівняє дані нового файлу з оригінальним, відносно міток будемо використовувати раніше сформовані карти і діагностичні слова в оригінальному файлі. Результатом виконання цього коду є наступний вивід (рис. 2.20):

```
--- Початок повної валідації 'final_dataset.csv' ---
Файли 'final_dataset.csv' та 'full_df.csv' успішно завантажені.

Базова перевірка цілісності
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6338 entries, 0 to 6337
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   ID           6338 non-null   int64
 1   Patient Age  6338 non-null   int64
 2   Patient Sex  6338 non-null   object
 3   filename     6338 non-null   object
 4   N            6338 non-null   int64
 5   D            6338 non-null   int64
 6   G            6338 non-null   int64
 7   C            6338 non-null   int64
 8   A            6338 non-null   int64
 9   H            6338 non-null   int64
 10  M            6338 non-null   int64
 11  O            6338 non-null   int64
dtypes: int64(10), object(2)
memory usage: 594.3+ KB
------------------------------------------------------------
```

Рисунок 2.20 - Результат роботи коду базової валідації створеного набору даних

Звідси бачимо, що базова перевірка цілісності файлу пройшла успішно. Аналіз за допомогою df.info() показує, що новостворений набір даних final_dataset.csv містить 6338 записів, що точно відповідає очікуваній кількості після фільтрації (6392 початкових - 54 відфільтрованих) [23]. Структура файлу також є коректною: він містить 12 колонок, де 10 з них (ID, вік та 8 колонок з мітками) мають числовий тип int64, а 2 (Patient Sex, filename) - об'єктний/текстовий. Важливо, що у жодній з колонок немає пропущених значень (Non-Null Count для всіх дорівнює 6338), що свідчить про коректність роботи скрипта генерації.

Базова перевірка підтверджує отримання цілісного та повного файлу даних.

2.4.2 Аналіз демографічних даних нового набору даних

Наступним кроком є більш глибокий аналіз його демографічного складу та порівняння зі змінами в оригінальному наборі даних (рис. А.13) [1]. Результатом виконання цього коду є наступний вивід (рис. 2.21):

```
Детальний аналіз демографії
Кількість унікальних пацієнтів: 3349 (було 3358, втрачено 9)

Перевірка втрачених пацієнтів:
  - ID: 352 (Існує L: ✅, R: ✅), L_kw: 'refractive media opacity', R_kw: 'refractive media opacity'
  - ID: 388 (Існує L: ✅, R: ✅), L_kw: 'refractive media opacity', R_kw: 'refractive media opacity'
  - ID: 585 (Існує L: ✅, R: ✅), L_kw: 'refractive media opacity', R_kw: 'refractive media opacity'
  - ID: 1865 (Існує L: ❌, R: ✅), L_kw: 'laser spot, chorioretinal atrophy, glaucoma', R_kw: 'refractive media opacity'
  - ID: 1516 (Існує L: ✅, R: ✅), L_kw: 'refractive media opacity', R_kw: 'refractive media opacity'
  - ID: 973 (Існує L: ✅, R: ✅), L_kw: 'refractive media opacity', R_kw: 'refractive media opacity'
  - ID: 283 (Існує L: ❌, R: ✅), L_kw: 'spotted membranous change', R_kw: 'refractive media opacity'
  - ID: 1052 (Існує L: ✅, R: ✅), L_kw: 'refractive media opacity', R_kw: 'refractive media opacity'
  - ID: 829 (Існує L: ✅, R: ✅), L_kw: 'refractive media opacity', R_kw: 'refractive media opacity'

Знайдено 9 пацієнтів, у яких всі існуючі зображення були чистими артефактами. Це пояснює їх втрату.

Кількість 'одноокіх' пацієнтів: 360 (було 324)

Розподіл за статтю у новому датасеті:
Patient Sex
Male      53.57
Female    46.43
Name: proportion, dtype: float64

Статистичний опис віку у новому датасеті:
count    3349.000000
mean       57.859361
std        11.731295
min         1.000000
25%        51.000000
50%        59.000000
75%        66.000000
max        91.000000
Name: Patient Age, dtype: float64
--------------------------------------------------------------------------------
```

Рисунок 2.21 - Результат роботи коду аналізу демографічних даних створеного набору даних

Звідси бачимо, що програмний аналіз підтверджує, що кількість унікальних пацієнтів у фінальному наборі даних зменшилася з 3358 до 3349, тобто було повністю виключено 9 пацієнтів. Для верифікації причини їх виключення був проведений детальний аналіз. Скрипт підтвердив, що це саме ті 9 пацієнтів, у яких всі наявні в наборі даних зображення (одне або два) були класифіковані як “чисто технічні артефакти”. Наприклад, для пацієнта ID=352 обидва ока мали діагноз refractive media opacity. Для пацієнта ID=283, який в оригінальному наборі даних був представлений лише одним зображенням, цей єдиний запис також містив виключно технічний діагноз. Це пояснює їх відсутність у фінальному наборі даних, адже було прийнято рішення відкидати діагностично не інформативні зображення [5].

Також, спостерігається очікувана зміна у кількості пацієнтів, представлених одним зображенням - “однооких”. Їхня кількість зросла з 324 до 360. Ця різниця в +36 пацієнтів є прямим наслідком фільтрації. Математична перевірка повністю підтверджує коректність цього процесу [4]. З 54 відфільтрованих зображень, 16 належали 9 пацієнтам, які були повністю виключені з набору даних (з них 7 були двоокими, що втратили 14 зображень відповідно, та 2 були одноокими, що втратили 2 зображення відповідно). Решта 38 відфільтрованих зображень (54−16=38) належали пацієнтам, що залишилися в наборі даних і стали новими одноокими. Це пояснює, чому кількість однооких пацієнтів змінилася наступним чином:

324 (початкова кількість) - 2 (видалені оригінальні одноокі) + 38 (нові одноокі) = 360;

Нарешті, загальні демографічні показники, такі як розподіл за статтю (53.6% чоловіків) та віком (середній вік ~57.9 років), у новому наборі даних залишилися практично незмінними порівняно з оригінальним. Це свідчить про те, що процес фільтрації, спрямований на видалення неінформативних даних, не вніс значних статистичних викривлень у репрезентативність вибірки [5].

Відповідно, даний етап валідації підтверджує, що зміни у складі набору даних є логічними, математично пояснюваними і відповідають меті очищення даних зі збереженням їхніх основних характеристик.

2.4.3 Перехресна верифікація згенерованих міток

Наступним кроком є перехресна верифікація згенерованих міток (рис. А.14). Результатом виконання цього коду є наступний вивід (рис. 2.22):

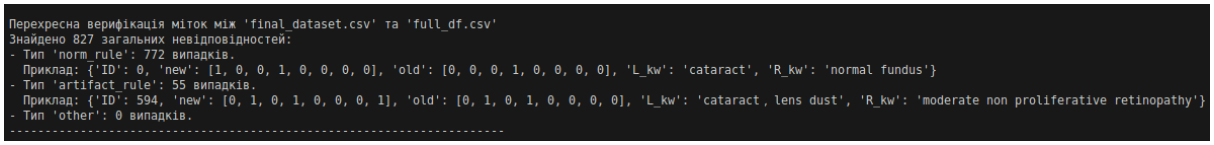


Рисунок 2.22 - Результат роботи коду перехресної верифікації створеного набору даних

Звідси бачимо, що був реалізований алгоритм, що для кожного пацієнта порівнює сумарний вектор міток, згенерований для його наявних очей, з оригінальним вектором міток для пацієнта в цілому [23]. Програмний аналіз виявив 827 загальних невідповідностей, що на перший погляд може здатися ознакою помилки. Однак, детальна класифікація цих розбіжностей показує, що всі вони є очікуваними і є прямим наслідком двох свідомих методологічних рішень, прийнятих для покращення якості та інформативності даних [4]. Очікувано, найбільша група розбіжностей являє собою розбіжності через “правило норми” (772 випадки). Вона виникає у випадках, подібних до пацієнта ID=0, де одне око здорове (normal fundus), а на іншому є патологія (cataract). Оригінальний набір даних, слідуючи “правилу норми”, присвоював такому пацієнту вектор N=0, C=1. Алгоритм, працюючи на рівні окремих очей, коректно генерує мітки для обох, що при сумації дає вектор N=1, C=1. Ця розбіжність не є помилкою, а навпаки, доводить, що успішно подолали обмеження оригінальної логіки маркування і зберегли інформацію про здорове око, працюючи з окремими зображеннями, а не

пацієнтами в цілому [3]. Також, у ході аналізу було знайдено інше правило оригинального набору даних - часто, технічні мітки не враховуються як клас “інше”. Назвемо таке правило “правилом артефактів”. Було прийнято рішення завжди відносити технічні артефакти до класу “інше”. Таким чином клас “інше” в новому наборі даних являє собою клас для будь чого, що не відповідає основним 7 класам. Також, якщо в оригінальному наборі даних у випадку, наприклад, катаракта та пил на лінзі враховується тільки клас C, то в сгенерованому наборі даних будуть проставлені мітки C та O як “катаракта з особливістю”, адже таким чином отримуємо більшу інформативність даних для моделі і потенційно більшу робастність (надійність) моделі [5]. Розбіжності через наше правило для артефактів (55 випадків) виникає через наше рішення перетворити клас O на універсальний детектор аномалій. Наприклад, для пацієнта ID=594 з діагнозами cataract, lens dust та retinopathy, оригінальний набір даних ігнорував технічний артефакт lens dust і присвоював вектор D=1, C=1, O=0. Наша ж карта правил, навпаки, відносить lens dust до класу O, генеруючи більш інформативний вектор D=1, C=1, O=1.

Найважливішим результатом цього аналізу є те, що кількість інших (непередбачених) невідповідностей дорівнює нулю. Цей факт можна вважати прямим доказом того, що наша карта правил (keyword_to_label_map) та алгоритм генерації працюють коректно в рамках визначеної нової логіки. Всі розбіжності з оригіналом є не помилками, а очікуваними наслідками свідомих методологічних рішень.

2.4.4 Перевірка “правила норми” та “правила артефактів” на новому наборі даних

Більш детально перевіримо статистику по “правилу норми” та “правилу артефактів” (рис. А.15). Результатом виконання цього коду є наступний вивід (рис. 2.23):

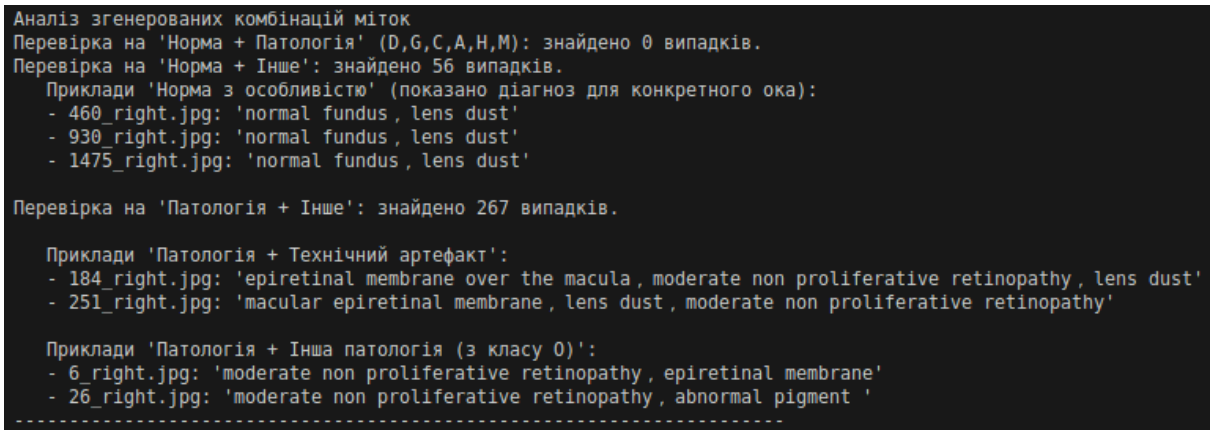


Рисунок 2.23 - Результат роботи коду аналізу “правила норми” та “правила артефактів” створеного набору даних

Звідси бачимо, що було проведено програмний аналіз згенерованих комбінацій міток для перевірки коректності реалізації розроблених правил мапінгу. Перевірка на одночасну наявність мітки “норма” (N) та будь-якої з основних патологій (D, G, C, A, H, M) показала відсутність таких випадків. Це підтверджує, що алгоритм генерації зберіг логічний принцип взаємовиключності стану “норма” та наявності чітко визначеного захворювання [22]. Було виявлено 56 випадків, де мітка N=1 співіснує з міткою O=1 (“інше”). Аналіз прикладів, наведених скриптом, демонструє, що ці випадки відповідають діагностичним описам normal fundus, lens dust і подібним, що свідчить про коректну реалізацію правила для збереження інформації про здорові очі, зображення яких містять технічні артефакти [5]. Аналіз комбінації “Патологія + Інше” виявив 267 таких випадків. Розгляд прикладів дозволяє виділити два основні сценарії, що призводять до такої комбінації міток, а саме наявність технічного артефакту разом з основною патологією (moderate nonproliferative retinopathy, lens dust, показують, що технічні дефекти на діагностично цінних зображеннях були коректно віднесені до класу O [23]), та наявність кількох патологій (moderate nonproliferative retinopathy, epiretinal membrane, демонструють, що клас O також використовується для позначення рідкісних захворювань, що супроводжують основну патологію [1]).

Таким чином, даний етап валідації підтверджує, що алгоритм генерації міток працює у повній відповідності до розроблених правил, створюючи логічно консистентні та інформативні мультилейблові вектори [4].

2.4.5 Перевірка задачі нового набору даних

Додатково впевнимося в тому, що сгенерований набір даних все ще залишається мультилейбловим (рис. А.16). Результатом виконання цього коду є наступний вивід (рис. 2.24):

```
Фінальний аналіз мультилейбловості з діагностикою
Аналіз фінального датасету (df_new):
- Знайдено 497 мультилейблових зображень.
--------------------------------------------------------------------------------
Аналіз оригінального датасету (df_old):
- Знайдено 548 мультилейблових записів через кому.
- З урахуванням карти правил, мультилейблів в оригіналі: 497.
--------------------------------------------------------------------------------
Діагностика різниці:

Пояснення розбіжності між методом по комі та методом з картою:
Різниця складає 51 випадків (548 - 497).
Це записи, де кілька термінів відносяться до одного класу.
Приклади:
                                keywords generated_labels
69                drusen, atrophic change              {0}
171  macular epiretinal membrane, lens dust              {0}
189        vitreous degeneration, lens dust              {0}
199          epiretinal membrane, lens dust              {0}
227  macular epiretinal membrane, lens dust              {0}

Фінальний розподіл зображень за кількістю міток:
num_labels
1    5841
2     490
3       7
Name: count, dtype: int64

Приклади зображень з 2 мітками з фінального датасету:
    ID  Patient Age Patient Sex      filename  N  D  G  C  A  H  M  O                                                relevant_keywords  num_labels
5    6           60        Male   6_right.jpg  0  1  0  0  0  0  0  1      moderate non proliferative retinopathy, epiretinal membrane           2
10  11           60      Female  11_right.jpg  0  1  0  0  0  1  0  0  moderate non proliferative retinopathy, hypertensive retinopathy           2
21  26           63      Female  26_right.jpg  0  1  0  0  0  0  0  1         moderate non proliferative retinopathy, abnormal pigment           2

Приклади зображень з 3 мітками з фінального датасету:
        ID  Patient Age Patient Sex       filename  N  D  G  C  A  H  M  O                                                                                relevant_keywords  num_labels
1083  1409           79      Female  1409_right.jpg  0  1  1  0  0  0  0  1                    moderate non proliferative retinopathy, lens dust, suspected glaucoma           3
1333  2030           66        Male  2030_right.jpg  0  1  1  0  0  0  0  1                       mild nonproliferative retinopathy, glaucoma, vitreous degeneration           3
1338  2063           77        Male  2063_right.jpg  0  1  1  0  0  0  0  1                            glaucoma, moderate non proliferative retinopathy, laser spot           3
1411  2168           64      Female  2168_right.jpg  0  1  0  1  0  0  0  1                            cataract, laser spot, moderate non proliferative retinopathy           3
4282  1442           63        Male   1442_left.jpg  0  1  1  0  0  0  0  1                glaucoma, mild nonproliferative retinopathy, macular pigmentation disorder           3
4489  1994           67        Male   1994_left.jpg  0  1  0  1  0  0  0  1                  cataract, myelinated nerve fibers, moderate non proliferative retinopathy           3
4508  2078           42      Female   2078_left.jpg  0  1  0  0  1  0  0  1  diabetic retinopathy, wet age-related macular degeneration, macular epiretinal membrane           3
```

Рисунок 2.24 - Результат роботи коду аналізу задачі створеного набору даних

Звідси бачимо, що аналіз розподілу зображень за кількістю міток дає кількісну характеристику складності очищеного набору даних і остаточно обґрунтовує вибір мультилейблового підходу. З результатів бачимо, що з 6338 зображень, переважна більшість - 5841 (приблизно 92%) дійсно мають лише одну мітку. Проте ігнорування решти випадків можна вважати грубою методологічною помилкою [5]. Аналіз показує наявність 497 зображень (близько 7.8%) з двома або трьома мітками одночасно, що є значною статистично важливою підмножиною.

Інтуїтивно, може здатися, що через домінування однокласових випадків можна спростити задачу до мультикласової. Але таке твердження є хибним з огляду на практичне застосування. Метою даної роботи є розробка прототипу інструменту, що здатен працювати в умовах, наближених до реальної клінічної практики, в якій коморбідність (наявність декількох патологій одночасно) є поширеним явищем [1]. Тому 497 випадків варто вважати не статистичною похибкою, а прямим відображенням реальності.

Додатково, проведений діагностичний аналіз пояснив різницю між підрахунком мультилейблів (548) за наявністю коми, що був проведений раніше та (497) після застосування карти правил. Було доведено, що 51 випадок є перерахуванням синонімічних термінів для одного класу захворювання, що доводить не вірність врахування лише через кому. Це підтверджує, що мультилейблові випадки не є артефактом нашого процесу обробки даних, а їхня кількість була лише уточнена. Обраний процес генерації міток не змінив кількість вже існуючих мультилейблових випадків (приклади, виведені скриптом, демонструють випадки реальної коморбідності, такі як diabetic retinopathy, hypertensive retinopathy (D=1, H=1), що існували в даних від самого початку) через те, що обрана нова логіка формування даних базується на тих же текстових діагнозах для кожного ока окремо, що і були в оригінальному наборі даних [23].

Таким чином, вибір мультилейблової архітектури є необхідністю, яка продиктована природою обраного набору даних та метою роботи. Хоча модель дійсно буде навчатися переважно на однокласових прикладах, її здатність коректно обробляти значущі 7.8% мультилейблових випадків додатково буде визначати її практичну цінність.

2.5 Аналіз балансу класів

Маючи готовий набір даних, проаналізуємо баланс класів в ньому за допомогою коду рис. А.17. Результатом виконання цього коду є наступний вивід (рис. 2.25 - 2.26):

```
Аналіз дисбалансу класів у 'final_dataset.csv'
Кількість зображень для кожного діагностичного класу у фінальному датасеті:
--------------------------------------------------------------------------
   Кількість  Відсоток (%)
N       2874         45.35
D       1722         27.17
G        313          4.94
C        301          4.75
A        279          4.40
H        192          3.03
M        256          4.04
O        905         14.28
--------------------------------------------------------------------------
Графік 'final_class_imbalance.png' збережено.
```

Рисунок 2.25 - Результат роботи коду аналізу балансу класів створеного набору даних. Текстовий

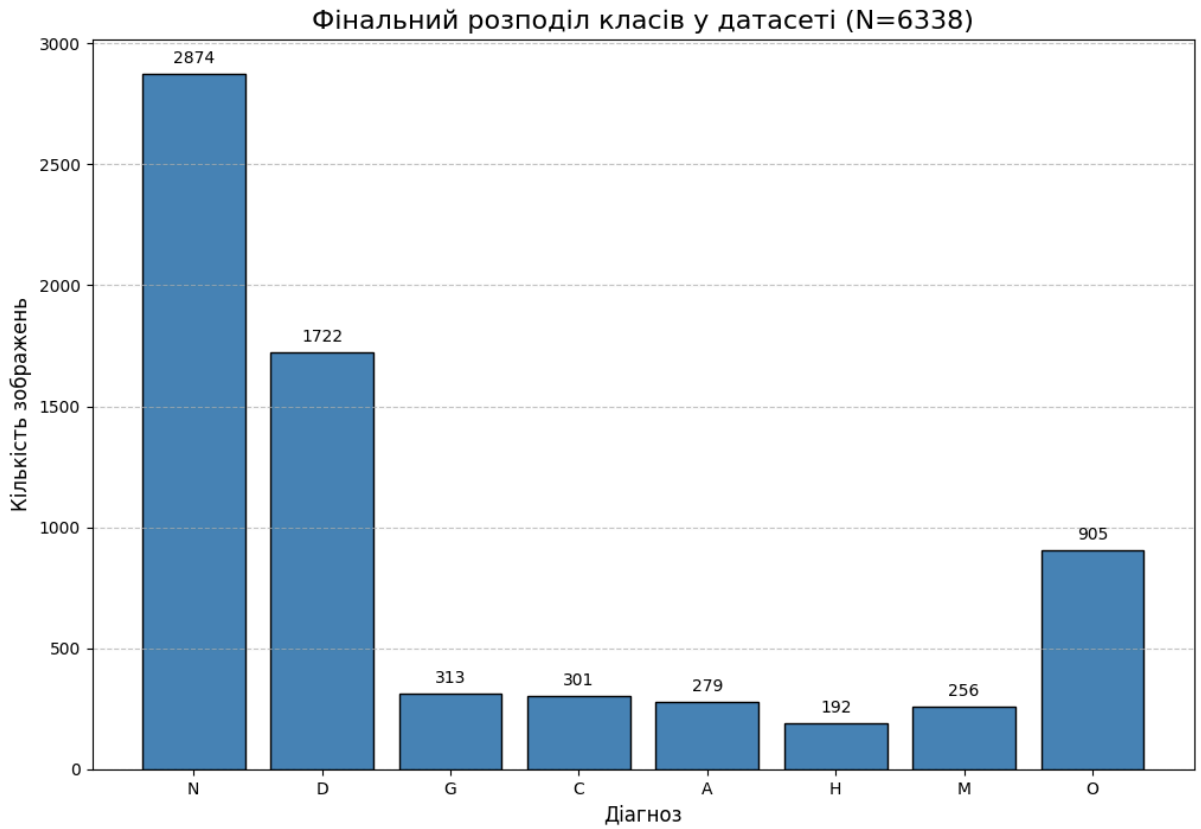


Рисунок 2.26 - Результат роботи коду аналізу балансу класів створеного набору даних.

Звідси бачимо, що аналіз розподілу класів у очищеному наборі даних final_dataset.csv дає кількісну оцінку дисбалансу, з яким доведеться працювати

моделі. Як наочно демонструє стовпчаста діаграма, дисбаланс є значно вираженим. Клас “норма” (N) є домінуючим, охоплюючи 2874 зображення, що складає 45.35% від усього набору даних. На другому місці за поширеністю знаходиться “діабет” (D) з 1722 зображеннями (27.17%). Представленість інших класів коливається від 905 зображень для класу “інше” (O) до 192 зображень для “гіпертонії” (H), що складає лише 3.03% від вибірки. Співвідношення між найчастішим класом (N) та найрідкіснішим (H) становить приблизно 15:1. Такий сильний дисбаланс є типовим для медичних наборів даних, адже в більшості випадків діагностується норма або найчастіші хвороби, і становить серйозний виклик для процесу навчання [35]. Без застосування спеціальних технік, модель буде схильна ігнорувати рідкісні класи, оскільки оптимізація функції втрат буде в першу чергу визначатися домінуючими класами [33]. Це підтверджує критичну необхідність застосування методів для боротьби з дисбалансом на етапі навчання, наприклад, шляхом зважування класів у функції втрат (Weighted Loss) [33]. Також можна використати комплекс методик зведення ваг класів. Наприклад, можна відкинути частину зображень значно переважуючих класів (undersampling), одночасно додати аугментації для слабко репрезентованих класів та застосувати зважування класів вже до такого, додатково обробленого набору даних. Проте комплексні методики доречно використовувати тільки у випадку, якщо зважування класів не допомогло, через те, що відкидання частини зображень (undersampling) призводить до втрати потенційно ключової для узагальнення інформації [5], а надмірні аугментації для рідкісних класів також можуть призвести до перенавчання моделі на однотипних, хоч і видозмінених, прикладах [24]. Тому в даній роботі основним методом боротьби з дисбалансом буде обрано саме зважування класів у функції втрат як найбільш збалансований та безпечний підхід.

2.6 Висновки щодо аналізу набору даних

Як висновок, був проведений всебічний, покроковий аналіз та процес очищення офтальмологічного набору даних ODIR-5K з метою підготовки якісного набору даних для навчання моделі глибокого навчання. В ході аналізу було вирішено низку методологічних та технічних завдань. Було проведено детальне дослідження структури та анотацій вихідного набору даних full_df.csv [23]. Цей аналіз виявив низку не явних особливостей формування набору даних, таких як “правило норми” та наявність однокласових міток labels/target, що зробило їх непридатними для поставленої задачі мультилейблової класифікації окремого ока [22]. Було проведено декомпозицію та мапінг 94 унікальних текстових діагнозів лікарів. На основі розробленої карти правил було згенеровано новий, чистий набір даних final_dataset.csv, що містить 6338 валідних зображень з коректними мультилейбловими мітками. В процесі генерації, на основі кількісного аналізу, було прийнято обґрунтоване рішення про виключення 54 діагностично неінформативних зображень (чистих артефактів), що підвищило загальну якість даних [5]. Була проведена вичерпна валідація новоствореного набору даних. Математична перевірка повністю пояснила всі зміни в демографічному складі [1], а перехресна верифікація міток підтвердила, що всі розбіжності з оригінальними анотаціями є очікуваними і є наслідком свідомо прийнятих рішень щодо покращення логіки маркування. Було проведено фінальний аналіз розподілу класів, який кількісно охарактеризував сильний дисбаланс у фінальному наборі даних (співвідношення ~15:1) і обґрунтував критичну необхідність застосування технік для його компенсації, зокрема, зважування класів у функції втрат [33].

Таким чином, в результаті проведеної роботи отримано повністю досліджений, очищений, валідований та готовий до використання набір даних. Це дозволяє перейти до наступного етапу - практичної реалізації та навчання архітектури нейронної мережі, описаної в попередньому розділі, на цих якісно підготовлених даних.

3 ПРОЕКТУВАННЯ, РЕАЛІЗАЦІЯ ТА АНАЛІЗ БАЗОВОЇ МОДЕЛІ

Після детального розгляду теоретичних основ, що лежать в основі згорткових нейронних мереж та підготовки оновленої розмітки, наступним логічним кроком є перехід до практичної реалізації. Основою для роботи слугуватиме якісно підготовлений та очищений набір даних, процес створення якого було детально описано в попередньому аналітичному блоці [23].

Для початку, буде побудовано та проаналізовано відносно просту, базову архітектуру згорткової нейронної мережі. Важливо підкреслити, що основна мета цього етапу - не створення моделі з максимально можливою точністю, а розробка подоби навчального стенду. Цю модель можна порівняти з інженерним прототипом, побудованим для демонстрації фундаментальних принципів, а не для реальних навантажень. Вона слугуватиме зрозумілою відправною точкою (baseline), з якою буде легко порівнювати більш складні архітектури в подальшому, та дозволить на практичному прикладі зрозуміти механізми, що були описані раніше лише теоретично [6].

Саме на прикладі цієї тривіальної моделі продемонструємо повний цикл роботи: від фінальної підготовки даних та їх поділу на вибірки до навчання, тестування та інтерпретації результатів. Будуть детально розглянуті ключові метрики оцінки якості, такі як точність (precision) та повнота (recall) [35]. Окрім цього, будуть представлені методи візуалізації процесу навчання, включаючи графіки втрат, та способи інтерпретації роботи моделі за допомогою методу Grad-CAM, що дозволяє побачити, на які саме ділянки зображення нейромережа звертає увагу при прийнятті рішення [36]. Також буде проведено аналіз впливу порогу активації на фінальний результат класифікації.

Всебічний аналіз результатів та виявлених обмежень цієї базової моделі дозволить сформулювати обґрунтовані вимоги до більш просунутих архітектур.

3.1 Формування вибірок для навчання та валідації

Як згадувалося в теоретичній частині, для об'єктивної оцінки здатності моделі до узагальнення (generalization), тобто її вміння працювати з новими, раніше небаченими даними, необхідно розділити вихідний набір даних на три незалежні частини: тренувальну, валідаційну та тестову вибірки [6]. Проте, просте випадкове перемішування та поділ даних на рівні зображень в нашому випадку є методологічно невірним підходом і може призвести до проблеми витоку даних (data leakage), адже входи даних не є абсолютно незалежними [5]. В контексті нашого набору даних це може статися, якщо зображення лівого та правого ока одного й того ж пацієнта опиняться в різних вибірках. Щоб уникнути цієї проблеми, поділ даних необхідно виконувати строго на рівні пацієнтів обираючи для внесення всі записи з одним ID пацієнта [3].

Тестова вибірка була цілеспрямовано сформована як невелика, але максимально різноманітна екзаменаційна група, що включає в себе заздалегідь відібрані унікальні клінічні випадки: пацієнти з одним оком, зображення з двома та трьома патологіями одночасно, комбінації “норма + патологія” для різних очей одного пацієнта, пацієнти тільки з нормою, тільки патологіями тощо. Валідаційна вибірка була сформована з 15% пацієнтів, що залишилися, для моніторингу процесу навчання. Вся решта даних була віднесена до тренувальної вибірки для забезпечення максимального обсягу інформації для навчання моделі.

Для реалізації описаного підходу було створено скрипт (рис. Б.1.1 - рис. Б.1.4), що виконує покроковий аналіз та розподіл пацієнтів. Результатом виконання коду є наступний вивід (рис. 3.1):

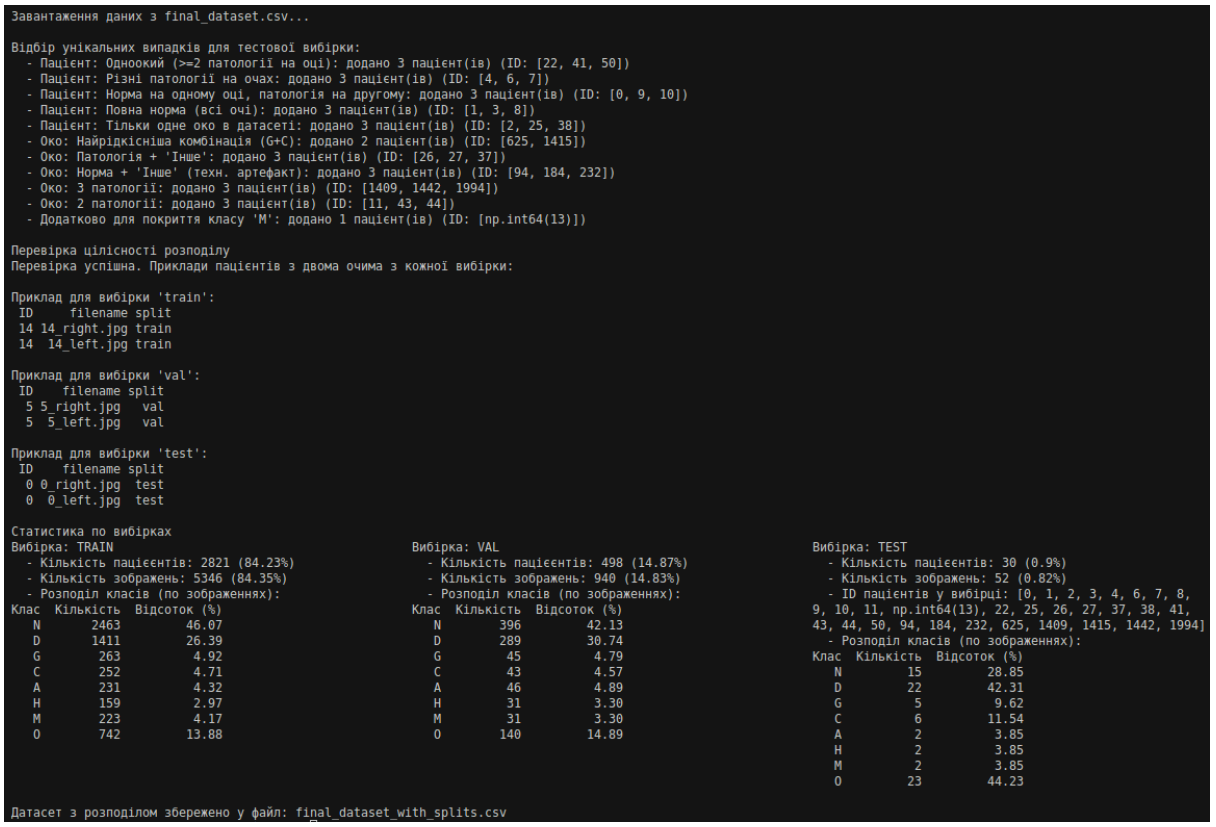


Рисунок 3.1 - Результат роботи коду створення вибірок

Звідси бачимо, що вихідний набір даних, який містить 3349 унікальних пацієнтів та 6338 зображень, було успішно розділено на три вибірки з дотриманням принципу поділу на рівні пацієнтів [23]. Процедура перевірки підтвердила відсутність витоку даних, продемонструвавши на прикладах для кожної з вибірок, що зображення обох очей одного пацієнта завжди знаходяться в межах однієї групи.

Тренувальна вибірка є найбільшою і містить дані про 2821 пацієнта (84.23%), що становить 5346 зображень (84.35%). Валідаційна вибірка, сформована з 498 пацієнтів (14.87%), містить 940 зображень (14.83%). Незначна розбіжність між відсотком пацієнтів та відсотком зображень в межах однієї вибірки є очікуваною і пояснюється наявністю в наборі даних як пацієнтів з одним оком, так і з двома. Оскільки поділ виконувався на рівні пацієнтів, випадкове потрапляння у вибірку трохи різної пропорції одноких та двоких пацієнтів призводить до цих невеликих коливань у фінальному відсотку зображень.

Важливо відзначити, що розподіл класів в обох цих вибірках залишається відповідним оригіналу і відображає загальний дисбаланс у вихідних даних [33].

Особливу увагу було приділено формуванню тестової вибірки. Вона є найменшою за розміром, включаючи 30 пацієнтів (0.9%) та 52 зображення (0.82%), і її склад було сформовано не випадково. До неї увійшли пацієнти з заздалегідь визначеними унікальними характеристиками, такі як асиметричні діагнози між очима, наявність кількох патологій на одному оці, рідкісні комбінації захворювань тощо. Цим пояснюється її атиповий розподіл класів, де відсоток класів (наприклад, N - 28.85%, O - 44.23%) значно відрізняється від інших виборок. Такий підхід дозволяє провести перевірку моделі на максимально повному спектрі можливих випадків, враховуючи рідкісні і не тривіальні для моделі задачі.

Таким чином, підготовлені набори даних є повністю готовими для подальшого використання на етапі побудови та навчання моделі.

3.2 Практична реалізація та навчання базової моделі

Після ретельної підготовки та валідації набору даних, наступним етапом є практична реалізація, навчання та всебічний аналіз базової моделі нейронної мережі. Всі експерименти, описані в цьому розділі, проводилися в єдиному програмному середовищі для забезпечення відтворюваності результатів. В якості операційної системи використовувалася Linux Mint 22.2 на базі Ubuntu 24.04, з ядром Linux версії 6.8. Основною мовою програмування було обрано Python, а в якості середовища розробки виступив Visual Studio Code. Обчислення виконувалися на апаратному забезпеченні, що складалося з 8-ядерного процесора AMD Ryzen 7 5700X та графічного прискорювача NVIDIA GeForce RTX 3060 Ti з 8 ГБ відеопам'яті GDDR6. Для реалізації було створено єдиний скрипт, що використовує набір спеціалізованих бібліотек. Цей підхід дозволяє зібрати всю логіку від завантаження даних до фінального аналізу в одному місці, що спрощує

проведення та відтворення експериментів. Першим кроком є налаштування глобальних параметрів та імпорт необхідних інструментів.

Розглянемо блок коду на зображенні детальніше (рис. В.1). На самому початку скрипта встановлюється рівень логування для бібліотеки TensorFlow. Це технічний крок, що дозволяє приховати службові повідомлення (INFO, WARNING) від TensorFlow та CUDA, які не стосуються безпосередньо процесу навчання, і таким чином зробити вивід програми більш чистим та читабельним. Далі відбувається імпорт бібліотек, що є фундаментом для роботи. Основним інструментом є TensorFlow - відкрита програмна бібліотека для машинного навчання, розроблена компанією Google. Вона надає потужну та гнучку екосистему для створення та навчання нейронних мереж. Разом з нею використовується Keras - високорівневий API всередині TensorFlow, що дозволяє описувати архітектуру моделі та процес навчання у значно простішому та більш інтуїтивному вигляді. Для роботи з табличними даними, такими як файл розмітки final_dataset_with_splits.csv, використовується бібліотека Pandas, що є стандартом де-факто для маніпуляції даними в Python. Для наукових обчислень та роботи з багатовимірними масивами (тензорами) використовується NumPy. За візуалізацію результатів, побудову графіків та діаграм відповідає бібліотека Matplotlib разом з надбудовою Seaborn для покращення естетичного вигляду. Бібліотека Scikit-learn надає широкий набір інструментів для аналізу метрик якості моделі, таких як classification_report. Нарешті, OpenCV (cv2) використовується для операцій з обробки зображень, зокрема при генерації візуалізацій Grad-CAM [36].

Центральною частиною цього блоку є секція налаштування експерименту. Винесення всіх ключових параметрів в окремий блок на початку коду є практикою, що дозволяє легко керувати поведінкою моделі та відтворювати результати. Параметр TARGET_IMG_SIZE встановлює цільовий розмір зображення (256x256 пікселів), до якого будуть приводитись усі вхідні дані, як це було обґрунтовано раніше при аналізі вимог до відеопам'яті. BATCH_SIZE визначає кількість зображень, що обробляються одночасно за один крок навчання. Значення 32 є поширеним компромісом між швидкістю навчання та стабільністю

градієнтів. Менші значення зменшують потребу у відеопам’яті оскільки на кожному кроці оброблюється менша кількість зображень, але також зменшує стабільність градієнту, подібно намаганням знайти середній зріст людей в країні опитавши лише 8 людей. Більші значення працюють прямо навпаки. Екстремальні значення, наприклад 1, називаються стохастичним градієнтним спуском (Stochastic Gradient Descent, SGD) і призводять до повільного, не стабільного навчання, але водночас такого, при якому проблеми локальних мінімумів майже не відбуваються. USE_EARLY_STOPPING є перемикачем, що активує механізм ранньої зупинки, який автоматично завершує навчання і повертається до стану останньої найкращої епохи, якщо якість моделі на валідаційній вибірці перестає покращуватися протягом PATIENCE епох. LEARNING_RATE (швидкість навчання) є одним з найважливіших гіперпараметрів, що контролює розмір кроку при оновленні ваг моделі під час градієнтного спуску. Він визначає на яку частку від розрахованого вектора градієнта будуть змінені ваги моделі. Невелике значення безрозмірного коефіцієнту у 0.0001 було обрано для забезпечення більш стабільного та точного навчання. Нарешті, GRADCAM_IMAGE_COUNT визначає, для скількох зображень з тестової вибірки будуть створені візуалізації “карт уваги” моделі, деякі з яких розглянемо надалі.

Наступним логічним блоком у скрипті є визначення набору функцій-помічників та кастомних класів, які інкапсулюють специфічну логіку та роблять основний процес навчання більш чистим та організованим (рис. В.2).

Першим визначено клас PerformanceMonitor. Це спеціальний інструмент, що наслідується від базового класу Callback бібліотеки Keras. Колбеки - це об'єкти, які можна підключити до процесу навчання моделі, щоб виконувати певні дії на різних його етапах (наприклад, на початку чи в кінці кожної епохи). PerformanceMonitor призначений для збору даних про використання системних ресурсів і фіксує завантаження центрального процесора (CPU), використання оперативної пам'яті (RAM) та відеопам'яті (VRAM) після завершення кожної епохи. Накопичена історія цих показників згодом використовується для побудови графіків, що дозволяє візуально оцінити апаратні вимоги моделі.

Далі йде клас CustomModelCheckpoint, який також є колбеком. Його основне завдання - відстежувати ключову метрику якості на валідаційній вибірці (в нашому випадку val_auc) і зберігати на диск тільки ту версію моделі, яка показала найкращий результат за весь час навчання. Це критично важливий механізм, оскільки, як побачимо на графіках, модель схильна до перенавчання, і її продуктивність на нових даних може погіршуватися на пізніх епохах. Цей колбек гарантує, що збережемо модель саме в момент її пікової ефективності.

Функція create_dataset є серцем конвеєра даних (Data Pipeline). Вона приймає на вхід DataFrame бібліотеки Pandas з інформацією про зображення та їхні мітки, і перетворює його на високоефективний об'єкт tf.data.Dataset. Цей об'єкт виконує функцію розумного генератора, що в реальному часі завантажує зображення з диска, виконує їх перед-обробку (зміну розміру та нормалізацію значень пікселів до діапазону) та групує їх у батчі. Використання tf.data.Dataset є сучасною практикою, яка дозволяє значно прискорити навчання завдяки паралельній обробці даних та попередньому завантаженню (prefetching), що мінімізує простої графічного процесора в очікуванні нових даних.

Функція calculate_class_weights реалізує стратегію боротьби з дисбалансом класів, який виявили на етапі аналізу даних. Як було показано, кількість зображень для різних класів відрізняється в десятки разів. Без компенсації цього ефекту модель швидко навчилася б ігнорувати рідкісні патології, оскільки це дозволило б їй мінімізувати загальну помилку, просто фокусуючись на найчастіших класах. Ця функція аналізує тренувальну вибірку і розраховує для кожного класу вагу, обернено пропорційну його частоті. Рідкісніші класи отримують значно більшу вагу. Під час розрахунку функції втрат помилка для рідкісного класу множиться на його велику вагу, що змушує модель приділяти таким помилкам значно більше уваги, ніж помилкам на частих класах (збільшується внесок помилки цього класу в загальну функцію втрат, що, в свою чергу, призводить до більшого за величиною градієнта для ваг, відповідальних за розпізнавання цього класу). Важливо підкреслити, що для цього розрахунку

використовуються дані тільки з тренувальної вибірки, щоб уникнути витоку інформації про розподіл класів у валідаційному та тестовому наборах даних [33].

Наступний блок коду визначає архітектуру нейронної мережі та реалізує механізм Grad-CAM для її подальшої візуалізації (рис. В.3).

Функція build_model відповідає за програмне створення архітектури базової моделі. Як було спроектовано раніше, вона складається з трьох основних частин: блоку аугментації даних, тіла моделі (feature extractor) та голови (classifier).

Першим шаром є блок аугментації (data_augmentation), який реалізовано за допомогою tf.keras.Sequential. Sequential, це найпростіший спосіб створення моделі в Keras, який представляє собою лінійний контейнер для шарів. Усі шари, додані до Sequential моделі, виконуються строго послідовно, один за одним. В даному випадку він містить набір шарів для випадкових трансформацій зображень, що застосовуються в реальному часі (по мірі роботи моделі): RandomFlip(“horizontal”) виконує горизонтальне віддзеркалення зображення з імовірністю 50%; RandomRotation(0.1) повертає зображення на випадковий кут в діапазоні від -10% до +10% від повного кола (тобто від -36° до +36°); RandomZoom(0.1) випадковим чином збільшує або зменшує масштаб зображення на величину до 10%; RandomContrast(0.1) змінює контрастність зображення також в межах +-10%. Важливою особливістю є те, що ці шари активні лише під час навчання і автоматично вимикаються під час валідації та тестування. Це дозволяє моделі на кожній епосі бачити дещо змінені версії одних і тих самих зображень, що значно ускладнює завчання та змушує її фокусуватися на стабільних біологічних ознаках, а не на випадкових артефактах.

Далі йде тіло моделі, яке складається з трьох послідовних блоків Conv2D - MaxPooling2D. Ця структура точно відповідає тій, що була розрахована в теоретичній частині [15]. Кількість фільтрів у згорткових шарах послідовно збільшується (32 - 64 - 128), що дозволяє моделі будувати все більш складну ієрархію візуальних ознак, починаючи від простих ліній та градієнтів на перших шарах до складних текстур та форм на глибших. Кожен шар MaxPooling2D зменшує просторову розмірність карти ознак вдвічі, що допомагає моделі

узагальнювати інформацію та робить її менш чутливою до точного розташування ознак на зображенні.

Останньою частиною є голова моделі, що складається з шару Flatten, який випрямляє тривимірну карту ознак в одновимірний вектор, та двох повнозв'язних шарів Dense. Перший Dense шар на 128 нейронів виступає в ролі аналізатора комбінацій виявлених ознак. Фінальний Dense шар має 8 нейронів (по одному для кожного класу) та функцію активації sigmoid. Вибір sigmoid є ключовим для задачі мультилейблової класифікації, оскільки вона розраховує ймовірність для кожного класу незалежно від інших (на відміну від softmax, де сума ймовірностей завжди дорівнює одиниці) [6].

Функції make_gradcam_heatmap, format_labels, save_gradcam_with_info та generate_gradcam_visualizations разом реалізують механізм візуалізації Grad-CAM (Gradient-weighted Class Activation Mapping). Цей метод дозволяє зазирнути всередину моделі і зрозуміти, на які саме ділянки вхідного зображення вона звертала найбільше уваги при прийнятті рішення. Алгоритм аналізує градієнти, що надходять до останнього згорткового шару (conv2d_3), і на їх основі створює теплову карту (heatmap), яка показує області, що зробили найбільший внесок у фінальне передбачення. Фінальна функція накладає цю карту на оригінальне зображення та додає текстову інформацію про істинний та передбачений діагнози, створюючи таким чином вичерпний візуальний звіт про роботу моделі на конкретному прикладі.

Наступний блок коду визначає набір функцій, призначених для всебічного аналізу та візуалізації результатів навчання моделі. Кожна функція інкапсулює логіку для створення певного звіту або графічного представлення даних (рис. В.4.1 та рис. В.4.2).

Функція plot_training_history використовує дані з логу навчання, що зберігається колбеком CSVLogger, для побудови графіків ключових метрик (loss, auc, precision, recall, а також додатково розрахованого F1-score) в динаміці по епохах. Вона генерує окремі файли зображень для кожної метрики, порівнюючи на них показники для тренувальної та валідаційної вибірок [35]. Це є стандартним

інструментом для візуальної діагностики процесу навчання та виявлення перенавчання. Візуально, на графіках, перенавчання відображається як розбіжність між графіком навчання і графіком валідації.

Наступний набір функцій призначений для глибшого аналізу передбачень моделі, зроблених на валідаційній вибірці. Функція plot_probability_histograms створює гістограми, що показують розподіл ймовірностей, які модель видала для кожного класу. Вона розділяє передбачення на дві групи: для об'єктів, що дійсно належать до класу (позитивні), і для тих, що не належать (негативні), що дозволяє візуально оцінити впевненість та розділову здатність моделі.

Функція plot_threshold_metrics реалізує програмний перебір різних значень порогу бінаризації (від 0.05 до 0.95). Для кожного значення порогу вона розраховує усереднені (macro) метрики F1-score, precision та recall, а також загальну точність (accuracy), і будує графік їх залежності від порогу.

Функція plot_per_class_pr_curves генерує окрему Precision-Recall криву для кожного з восьми класів, що візуалізує компроміс між цими двома метриками при зміні порогу, та розраховує площу під кривою (AUC-PR).

Функція plot_confusion_matrices створює набір з восьми матриць помилок, по одній для кожного класу, що дозволяє детально проаналізувати типи помилок (TP, TN, FP, FN) для кожної патології окремо.

Функція analyze_test_groups відповідає за фінальний аналіз на тестовій вибірці. Вона використовує заздалегідь підготовлений розподіл даних, групує зображення за ID пацієнта і для кожного унікального тестового випадку генерує текстовий звіт, що порівнює істинні та передбачені мітки для кожного ока.

Останньою йде функція plot_resource_usage. Вона використовує дані, зібрані раніше визначеним колбеком PerformanceMonitor, для побудови графіків використання CPU, RAM та відеопам'яті протягом усього процесу навчання, що дозволяє оцінити апаратні вимоги моделі.

Останній блок коду є головною точкою входу в програму (*if __name__=="__main__":*). Він послідовно виконує всі етапи експерименту від

підготовки середовища до фінального аналізу та збереження результатів (рис. В.5.1 та рис В.5.2).

На самому початку відбувається підготовка робочої директорії training_results_base_model. Скрипт перевіряє її існування і, якщо вона вже є, повністю видаляє її разом з усім вмістом, після чого створює наново. Такий підхід гарантує, що результати кожного нового запуску є повністю чистими і не змішуються з артефактами попередніх експериментів. Далі виводиться службова інформація про версію TensorFlow та назву використовуваного графічного процесора, що є важливою частиною документування умов експерименту.

Після цього завантажується файл final_dataset_with_splits.csv і на основі колонки split дані розділяються на три окремі DataFrame: train_df, val_df та test_df. Потім ці DataFrame передаються у функцію create_dataset для створення трьох відповідних об'єктів tf.data.Dataset, які будуть подавати дані в модель.

Далі, використовуючи раніше визначені функції, відбувається розрахунок ваг класів, побудова архітектури моделі та її компіляція. На етапі компіляції модель зв'язується з оптимізатором Adam, якому передається швидкість навчання LEARNING_RATE з блоку налаштувань. Також визначається функція втрат binary_crossentropy, що є стандартом для мультилейблової бінарної класифікації, та набір метрик (AUC, Precision, Recall), які будуть відстежуватися під час навчання. Після компіляції виводиться зведена таблиця архітектури моделі (model.summary()), що детально показує послідовність шарів, розмірність їх виходів та кількість параметрів.

Наступним кроком є налаштування колбеків. Створюється список, що включає CustomModelCheckpoint для збереження найкращої моделі, CSVLogger для запису історії навчання у файл, та PerformanceMonitor для збору даних про використання ресурсів. Якщо в налаштуваннях активовано USE_EARLY_STOPPING, до цього списку також додається колбек EarlyStopping.

Після завершення всіх підготовчих етапів запускається процес навчання викликом model.fit(). Цьому методу передаються тренувальний та валідаційний набори даних, розраховані ваги класів для боротьби з дисбалансом, та

налаштовані колбеки. Після завершення навчання (або по закінченню всіх епох, або через спрацювання ранньої зупинки), викликаються функції для побудови та збереження графіків процесу навчання та використання ресурсів.

Фінальна частина скрипта присвячена всебічній оцінці навченої моделі. Спочатку завантажуються ваги найкращої епохи, збережені CustomModelCheckpoint. Потім модель оцінюється на тестовій вибірці: розраховуються загальні метрики, генерується детальний classification_report по класах, а також викликається функція analyze_test_groups для аналізу результатів на специфічних тестових випадках. Уся ця текстова інформація збирається та зберігається у єдиний файл test_report.txt. Після цього запускається генерація візуалізацій Grad-CAM.

В самому кінці скрипт проводить розширений аналіз на валідаційній вибірці (оскільки вона є більшою та більш репрезентативною для статистичних вимірювань), генеруючи гістограми ймовірностей, графік залежності метрик від порогу, PR-криві та матриці помилок. Наостанок вимірюється та виводиться розмір збереженого файлу моделі на диску та середній час, що витрачається на одне передбачення (inference time) для готової моделі.

3.3 Технічні аспекти процесу навчання

3.3.1 Оптимізатор (Optimizer)

В основі процесу навчання будь-якої нейронної мережі лежить ітеративне коригування її внутрішніх параметрів (ваг та зсувів) з метою мінімізації помилки. Цей процес керується алгоритмом, що називається оптимізатором. Якщо уявити всі можливі комбінації параметрів моделі як великий, багатовимірний ландшафт, а помилку моделі як висоту в кожній точці цього ландшафту, то завдання оптимізатора знайти в цьому ландшафті найнижчу точку - глобальний мінімум.

Для простоти розуміння, цей процес можна представити як рух кульки по складній гірській місцевості. Де кулька це нейромережа, позиція кульки на площині це її поточний набір ваг та зсувів, рух кульки це їх оновлення, ландшафт

це функція втрат, а мета кульки - закотитися в найнижчу точку (змінити параметри так, щоб отримати найнижчу помилку). Початковий стан моделі, це випадкова точка на схилі, з якої відпускаємо кульку (за рахунок випадкової ініціалізації параметрів). На кожному кроці кулька піддається гравітації відносно нахилу поверхні під собою (це градієнт, тобто вектор, що вказує напрям найшвидшого зростання помилки) і робить невеликий крок у її напрямку (насправді у протилежному напрямку адже градієнт за замовчуванням вказує напрям зростання помилки), тобто вниз по схилу. Розмір такого кроку регулюється швидкістю навчання (learning rate) [6].

Однак такий простий підхід (градієнтний спуск) має суттєві недоліки. Наприклад, якщо кулька потрапить у невелику локальну западину (локальний мінімум), вона може в ній залишитися, так і не діставшись до значно глибшої долини глобального мінімуму. Також, якщо схил дуже пологий, рух буде надто повільним, а на дуже крутих схилах кулька може робити завеликі кроки і перестрибувати мінімум, безкінечно коливаючись навколо нього [6].

Для вирішення цих проблем були розроблені більш просунуті алгоритми, одним з яких є Adam (Adaptive Moment Estimation), що використовується в даній роботі [31]. Adam є одним з найпоширеніших оптимізаторів і вважається надійним вибором для широкого кола задач, оскільки він поєднує в собі дві потужні ідеї.

Імпульс (Momentum), це механізм призначений для прискорення руху оптимізатора та подолання неглибоких локальних мінімумів. Технічно, він реалізується шляхом розрахунку експоненційно ковзного середнього попередніх градієнтів, яке називається першим моментом. На кожному кроці t цей момент m_t оновлюється за формулою (3.1) [31]:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \text{ ;} \qquad (3.1)$$

де g_t - це градієнт на поточному кроці,

m_{t-1} - значення моменту з попереднього кроку,

β_1 - гіперпараметр (зазвичай близько 0.9), що контролює затухання впливу старих градієнтів.

Це надає кульці інерцію. Оптимізатор не просто дивиться на нахил у поточній точці (g_t), а й враховує, куди він котився до цього (m_{t-1}). Якщо він довго рухався в одному напрямку, m_t накопичує значення, і наступний крок буде більшим. Наприклад, якщо градієнти були [10, 11, 9], то момент буде накопичувати це стабільне значення, і навіть якщо наступний градієнт раптом стане 1 (пологий схил), накопичений імпульс все одно проштовхне кульку вперед зі значною швидкістю.

Уявімо, що наша модель котиться по стабільному схилу, і градієнти протягом трьох кроків були g_1=10, g_2=9, g_3=8. Нехай β_1=0.9 і початковий момент m_0=0. Тоді:

$$\text{Крок 1: } m_1 = 0.9 \cdot 0 + 0.1 \cdot 10 = 1;$$
$$\text{Крок 2: } m_2 = 0.9 \cdot 1 + 0.1 \cdot 9 = 0.9 + 0.9 = 1.8;$$
$$\text{Крок 3: } m_3 = 0.9 \cdot 1.8 + 0.1 \cdot 8 = 1.62 + 0.8 = 2.42;$$

Тобто m_t поступово набирає інерцію. Тепер, якщо на 4-му кроці кулька потрапляє на майже пласку ділянку з g_4=1, новий момент буде:

$$\text{Крок 4: } m_4 = 0.9 \cdot 2.42 + 0.1 \cdot 1 = 2.178 + 0.1 = 2.278;$$

Замість того, щоб майже зупинитися (як зробив би простий градієнтний спуск, крок якого був би пропорційний 1), Adam зробить крок, пропорційний 2.278, легко долаючи пологу ділянку завдяки накопиченій інерції.

Адаптивна швидкість навчання (Adaptive Learning Rate), дозволяє Adam підбирати індивідуальну швидкість навчання для кожного окремого параметра.

Технічно, це досягається шляхом розрахунку експоненційно ковзного середнього квадратів попередніх градієнтів, яке називається другим моментом (v_t) [31]:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \; ; \qquad (3.2)$$

де g_t^2 - це по-елементний квадрат градієнта,

β_2 - гіперпараметр (зазвичай близько 0.999). При оновленні ваг, крок ділиться на корінь з v_t.

Якщо для якогось параметра градієнт стабільний (наприклад, завжди [2, 2.1, 1.9]), то v_t буде малим, і крок оновлення буде великим. Якщо ж градієнт не стабільний (наприклад, [10, -8, 22]), то його квадрати будуть великими ([100, 64, 484]), v_t накопичить велике значення, і крок оновлення для цього параметра автоматично зменшиться. Це дозволяє оптимізатору робити великі, впевнені кроки на рівнинах і сповільнюватися на крутих, кам'янистих схилах.

Розглянемо два параметри. Один має стабільний градієнт (рівний схил), інший дуже шумний (кам'янистий схил). Нехай β_2=0.9.

Параметр А (стабільний): градієнти [2, 2.1, 1.9]. Тоді:

$$v_1 = 0.9 \cdot 0 + 0.1 \cdot 2^2 = 0.4;$$
$$v_2 = 0.9 \cdot 0.4 + 0.1 \cdot 2.1^2 = 0.36 + 0.441 = 0.801;$$

Значення v_t залишається малим.

Параметр Б (нерівний): градієнти [10, -8, 12]. Тоді:

$$v_1 = 0.9 \cdot 0 + 0.1 \cdot 10^2 = 10;$$
$$v_2 = 0.9 \cdot 10 + 0.1 \cdot (-8)^2 = 9 + 6.4 = 15.4;$$

Значення v_t швидко зростає.

Оскільки фінальний крок ділиться на $\sqrt{v_t}$, для стабільного параметра А знаменник буде малим ($\sqrt{0.801} \approx 0.9$), і крок буде великим. Для нерівного параметра Б знаменник буде великим ($\sqrt{15.4} \approx 3.9$), і крок автоматично зменшиться.

Фінальне оновлення ваги θ на кроці t виглядає так (3.3) [31]:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \quad ; \tag{3.3}$$

де α - це швидкість навчання,

$\hat{m}_t$ та $\hat{v}_t$ - це скориговані моменти для усунення зміщення на початкових етапах,

ε - дуже мале число для уникнення ділення на нуль.

Варто окремо пояснити корекцію. Оскільки на самому початку навчання моменти m_t та v_t ініціалізуються нулями, на перших ітераціях їхні значення будуть штучно заниженими, тобто зміщеними до нуля. Щоб компенсувати це початкове зміщення і зробити оптимізатор більш стабільним з перших же кроків, Adam застосовує до них коригувальні коефіцієнти (3.4) [31]:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^{\,t}}; \tag{3.4}$$

та (3.5)

$$\hat{v}_t = \frac{v_t}{1-\beta_2^{\,t}}; \tag{3.5}$$

На перших кроках, коли t мале, цей дільник значно збільшує значення моментів. Зі збільшенням кількості ітерацій t, знаменник $1 - \beta_2^{\,t}$ швидко прямує до одиниці, і дія корекції плавно зникає, оскільки вона більше не потрібна.

Таким чином, вибір оптимізатора Adam є обґрунтованим рішенням, що дозволяє забезпечити стабільне та відносно швидке навчання, ефективно долаючи проблеми складного ландшафту функції втрат та зменшуючи ризик застрягання в невдалих локальних мінімумах.

3.3.2 Функція втрат (Loss Function)

Функція втрат є ключовим компонентом процесу навчання, що кількісно оцінює розбіжність між передбаченнями моделі та істинними мітками. На кожному кроці навчання оптимізатор використовує значення цієї функції для розрахунку вектора градієнта, який вказує напрямок для коригування параметрів моделі з метою мінімізації помилки [6].

Для даної задачі мультилейблової класифікації було обрано функцію binary_crossentropy (бінарна перехресна ентропія). Як було детально розглянуто в теоретичній частині, її фундаментальна відмінність від categorical_crossentropy полягає в тому, що вона розглядає кожен вихідний нейрон (що відповідає за окремий клас) як незалежну задачу бінарної класифікації [6]. Вона обчислює помилку для кожного класу окремо, а потім усереднює ці значення. Такий підхід є методологічно коректним для нашої проблеми, оскільки одне зображення ока може містити ознаки кількох патологій одночасно, і наявність однієї з них не виключає наявності інших. Таким чином, вибір binary_crossentropy забезпечує адекватну оцінку помилки моделі відповідно до мультилейблової природи даних [33].

3.3.3 Метрики (Metrics)

На відміну від функції втрат, яка є єдиним сигналом для оптимізатора, метрики є набором показників, призначених для інтерпретації якості моделі людиною. Вони не впливають на процес оновлення ваг, а лише відстежуються наприкінці кожної епохи для моніторингу процесу навчання [5].

Вибір правильних метрик є критично важливим, особливо в умовах сильного дисбалансу класів, як у нашому наборі даних. За таких умов стандартна метрика точності (accuracy) може бути оманливою, оскільки високі її значення можуть бути досягнуті моделлю, що просто ігнорує рідкісні класи і концентрує всю увагу лише на найчастіших класах [33]. Тому для моніторингу було обрано набір більш спеціалізованих та інформативних показників: Precision (Точність), Recall (Повнота) та AUC (PR) (Площа під Precision-Recall кривою) [35].

Разом ці метрики дозволяють отримати повну та збалансовану картину про поведінку моделі, її здатність знаходити патології та кількість помилок, які вона при цьому робить. Детальний аналіз кожної з цих метрик буде проведено в наступних розділах при розгляді графіків процесу навчання та фінальних результатів тестування.

3.4 Результати навчання та їх аналіз

Послідовно розглянемо та проаналізуємо вивід програми, а також згенеровані нею графічні та текстові звіти.

3.4.1 Початковий етап, підготовка, навчання та аналіз попередніх результатів

Перший блок виводу програми, представлений на зображенні 3.2, демонструє результати початкової ініціалізації середовища та підготовчих розрахунків перед запуском основного циклу навчання.

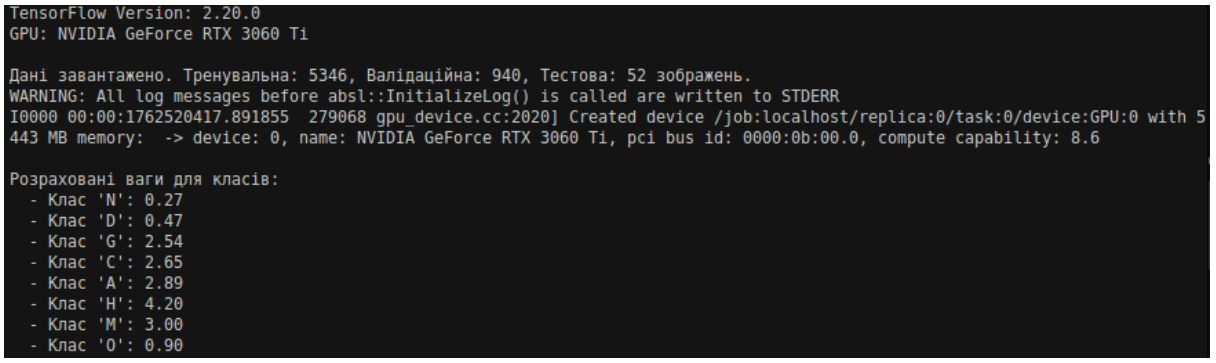


Рисунок 3.2 - Результат роботи коду попередньої підготовки

На початку виводу зафіксовано ключові компоненти середовища: версія TensorFlow (2.20.0) та назва графічного процесора (NVIDIA GeForce RTX 3060 Ti), що є важливим для відтворюваності експерименту. Далі підтверджено успішне завантаження та поділ даних на три вибірки: 5346 зображень для тренувальної, 940 для валідаційної та 52 для тестової. Наступним кроком є розрахунок ваг класів, що є стратегією боротьби з дисбалансом даних. Логіка розрахунку полягає в наданні кожному класу ваги, обернено пропорційної його частоті в тренувальній вибірці. Згідно з результатами, найчастіші класи 'N' (Норма) та 'D' (Діабет) мають найменші ваги (0.27 та 0.47 відповідно), тоді як рідкісні патології, такі як 'H' (Гіпертонія) та 'M' (Міопія), мають найбільші (4.20 та 3.00). Такий підхід означає, що під час навчання помилка моделі для кожного прикладу примножується на вагу його класу. Таким чином, одна помилка на зображенні з гіпертонією має для оптимізатора майже в 15 разів більший вплив на зміну градієнтів ($\frac{4.2}{0.27} \approx 15.5$), ніж одна помилка на здоровому оці. Це змушує оптимізатор приділяти значно більше уваги правильній класифікації рідкісних класів [33].

Наступним кроком є вивід зведеної таблиці архітектури моделі (model.summary()), що детально описує її структуру (рис. 3.3).

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 256, 256, 3)	0
augmentation (Sequential)	(None, 256, 256, 3)	0
conv2d_1 (Conv2D)	(None, 256, 256, 32)	896
max_pooling2d (MaxPooling2D)	(None, 128, 128, 32)	0
conv2d_2 (Conv2D)	(None, 128, 128, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 64)	0
conv2d_3 (Conv2D)	(None, 64, 64, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 128)	0
flatten (Flatten)	(None, 131072)	0
dense (Dense)	(None, 128)	16,777,344
predictions (Dense)	(None, 8)	1,032

Total params: 16,871,624 (64.36 MB)
Trainable params: 16,871,624 (64.36 MB)
Non-trainable params: 0 (0.00 B)

Рисунок 3.3 - Результат роботи коду виводу структури та параметрів базової мережі

Таблиця на зображенні демонструє послідовність шарів, з яких складається нейронна мережа зверху вниз, розмірність тензорів даних на виході кожного шару (Output Shape) та кількість параметрів, що навчаються (Param), для кожного з них. Дана архітектура повністю відповідає тій, що була спроектована в теоретичній частині, і складається з блоку аугментації, трьох згорткових блоків та голови-класифікатора.

Колонка Output Shape показує, як змінюється розмірність даних при проходженні через мережу. Перший елемент (None, ...) у кожному кортежі відповідає розміру батча. None тут означає, що модель є гнучкою і може приймати на вхід будь-яку кількість зображень одночасно, від одного до повного батча розміром 32. Як видно з таблиці, кожен шар MaxPooling2D зменшує просторову розмірність (висоту та ширину) тензора вдвічі, в той час як кожен згортковий шар Conv2D збільшує його глибину (кількість каналів/фільтрів).

Колонка Param показує кількість параметрів, що навчаються, для кожного шару. Ці значення повністю збігаються з розрахунками, проведеними в теоретичній частині. Наприклад, для шару conv2d_2 кількість параметрів становить 18,496, а для dense 16,777,344. Загальна кількість параметрів моделі становить 16,871,624. Важливо відзначити, що абсолютна більшість цих параметрів (майже 99.5%) зосереджена саме в повнозв'язному шарі dense, що є характерною ознакою простих архітектур і, як побачимо далі, є однією з головних причин схильності моделі до перенавчання [5].

Усі параметри моделі є такими, що навчаються (Trainable params), оскільки в ній відсутні шари, які могли б бути замороженими (наприклад, при використанні технік transfer learning - навчання на базі вже навченої моделі, певна кількість шарів навмисне заморожується, тобто не змінюється під час навчання [25]). Кількість параметрів, що не навчаються (Non-trainable params), дорівнює нулю. Хоча шари аугментації та пулінгу не мають ваг, які потрібно було б оптимізувати, деякі сучасні шари (наприклад, BatchNormalization) мають внутрішні змінні, які оновлюються під час навчання, але не через градієнтний спуск, і саме вони відносяться до категорії Non-trainable [28]. У нашій базовій моделі таких шарів немає.

Наступний етап, це безпосередньо процес навчання моделі (рис. 3.4). На зображенні представлено фрагмент виводу програми під час виконання методу model.fit().

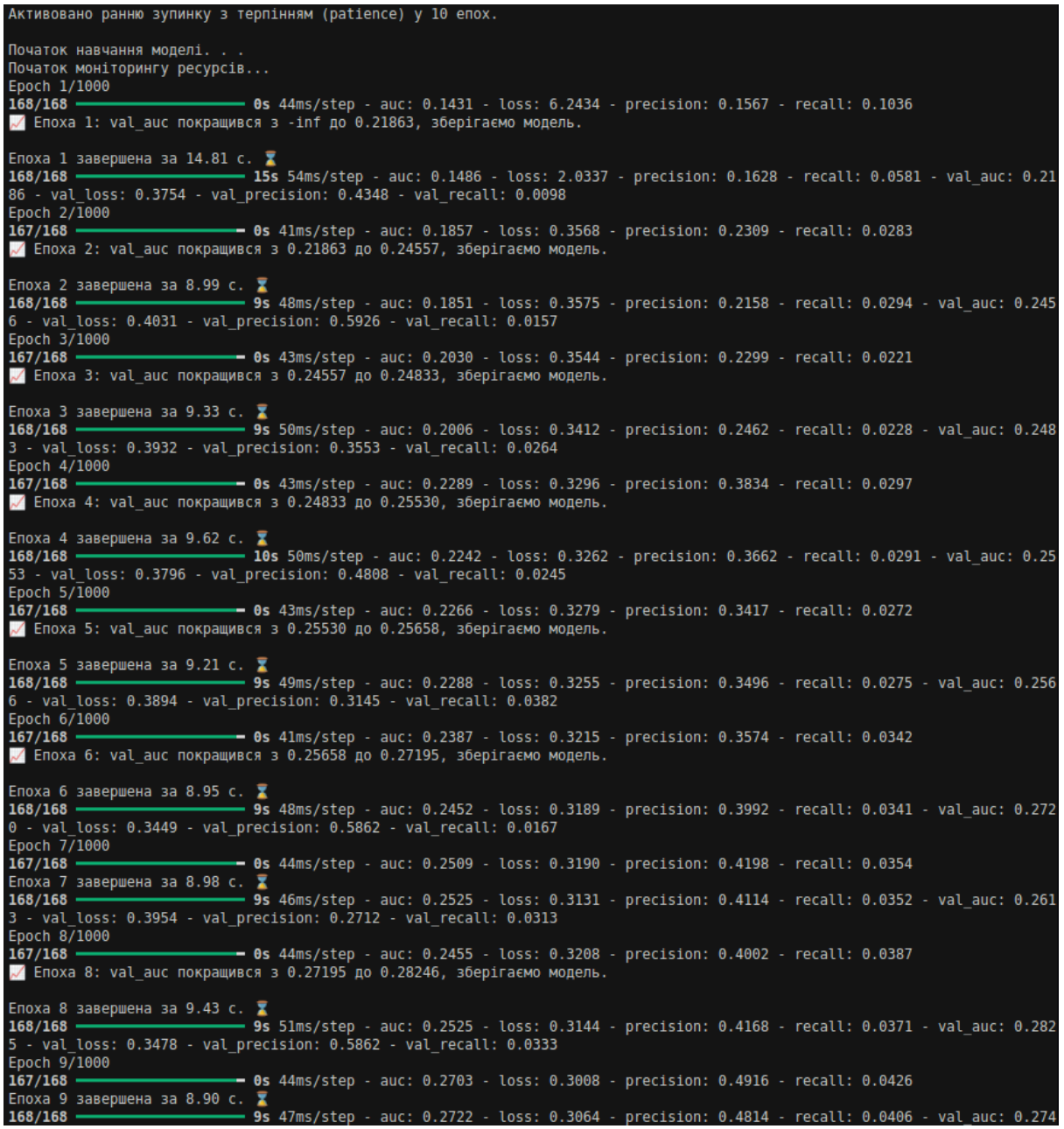


Рисунок 3.4 - Приклад виводу під час навчання мережі

Вивід демонструє покроковий прогрес навчання по епохах. Для кожної епохи відображається її номер, час виконання, а також фінальні значення функції втрат та метрик як для тренувальної, так і для валідаційної вибірки. Рядки, що вказують на покращення, генеруються спеціально створеним колбеком CustomModelCheckpoint. Вони з'являються лише тоді, коли значення ключової метрики val_auc на валідаційній вибірці перевищує найкращий результат,

досягнутий на попередніх епохах. У цей момент скрипт зберігає поточний стан ваг моделі на диск.

На зображенні 3.5 представлено завершення навчання, оцінка найкращої версії моделі на тестовій вибірці та детальний аналіз частини її передбачень.

```
✅ Загальний час навчання: 10 хвилин(а) 49 секунд(и).
Epoch 71: early stopping
Restoring model weights from the end of the best epoch: 61.
Графіки навчання збережено.
Графіки використання ресурсів збережено.

Оцінка найкращої моделі на тестовій вибірці
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step

Загальні метрики на тестовій вибірці:
  - loss: 0.5197
  - compile_metrics: 0.3002

Детальний звіт по класах (поріг 0.5):
              precision    recall  f1-score   support

           N       0.56      0.33      0.42        15
           D       0.20      0.05      0.07        22
           G       0.00      0.00      0.00         5
           C       0.50      0.17      0.25         6
           A       0.25      0.50      0.33         2
           H       0.00      0.00      0.00         2
           M       0.00      0.00      0.00         2
           O       0.67      0.09      0.15        23

   micro avg       0.38      0.13      0.19        77
   macro avg       0.27      0.14      0.15        77
weighted avg       0.41      0.13      0.18        77
 samples avg       0.18      0.15      0.15        77


Аналіз результатів по тестових групах

Пацієнт ID: 0 (2 зображення)
  - Успішно вгадано зображень: 1 з 2 (Точність: 50.00%)
    - 0_right.jpg: True='N', Pred='No prediction'
    - 0_left.jpg: True='C', Pred='C'

Пацієнт ID: 1 (2 зображення)
  - Успішно вгадано зображень: 0 з 2 (Точність: 0.00%)
    - 1_right.jpg: True='N', Pred='D'
    - 1_left.jpg: True='N', Pred='No prediction'

Пацієнт ID: 2 (1 зображення)
  - Успішно вгадано зображень: 0 з 1 (Точність: 0.00%)
    - 2_right.jpg: True='D', Pred='No prediction'

Пацієнт ID: 4 (2 зображення)
  - Успішно вгадано зображень: 0 з 2 (Точність: 0.00%)
    - 4_right.jpg: True='D', Pred='H'
    - 4_left.jpg: True='O', Pred='No prediction'

Пацієнт ID: 6 (2 зображення)
  - Успішно вгадано зображень: 0 з 2 (Точність: 0.00%)
    - 6_right.jpg: True='D, O', Pred='No prediction'
    - 6_left.jpg: True='O', Pred='No prediction'

Пацієнт ID: 7 (2 зображення)
  - Успішно вгадано зображень: 0 з 2 (Точність: 0.00%)
    - 7_right.jpg: True='D', Pred='No prediction'
    - 7_left.jpg: True='O', Pred='No prediction'

Пацієнт ID: 8 (2 зображення)
  - Успішно вгадано зображень: 2 з 2 (Точність: 100.00%)
    - 8_right.jpg: True='N', Pred='N'
```

Рисунок 3.5 - Результат роботи коду виводу оцінки найкращої моделі на тестовій вибірці

Процес навчання був автоматично зупинений на 71-й епосі завдяки механізму ранньої зупинки (EarlyStopping). Це сталося тому, що протягом 10 епох поспіль модель не змогла покращити свій найкращий показник val_auc (0.36907), досягнутий на 61-й епосі. Після зупинки модель автоматично повернулася до стану з найкращими вагами, зафіксованими на 61-й епосі. Загальний час навчання склав 10 хвилин 49 секунд.

Загальні метрики показують усереднену якість по всіх класах. Значення loss (0.5197) є відносно високим, а AUC (PR) становить 0.3002, що свідчить про невисоку загальну якість моделі [35].

Детальний звіт по класах розкриває цю картину глибше, надаючи значення precision, recall та f1-score для кожної патології при порозі класифікації 0.5. Для рідкісних класів G (Глаукома), H (Гіпертонія) та M (Міопія) всі метрики дорівнюють нулю. Це вказує на те, що модель не зробила жодного позитивного передбачення для цих класів на тестовій виборці. Для інших класів результати варіюються. Наприклад, для класу A (Вікова макулярна дегенерація) recall становить 0.50, що означає, що модель знайшла половину (1 з 2) реальних випадків. Водночас precision 0.25 показує, що з 4 разів, коли модель передбачила клас A в цілому, лише 1 раз це було правильне передбачення.

Аналіз результатів по тестових групах надає якісну оцінку роботи моделі на кожному окремому зображенні. Звіт демонструє, що модель здатна коректно класифікувати деякі прості випадки (наприклад, чисту норму у пацієнта ID 8), але часто помиляється на складних прикладах. Часто зустрічається результат No prediction, що означає, що для всіх восьми класів передбачена ймовірність виявилася нижчою за поріг 0.5. Це свідчить про загальну невпевненість моделі, що є прямим наслідком сильного перенавчання, яке буде детально розглянуто при аналізі графіків.

На фінальному етапі роботи скрипт також проводить аналіз фізичних та продуктивних характеристик навченої моделі, результати якого представлено на зображенні 3.6.

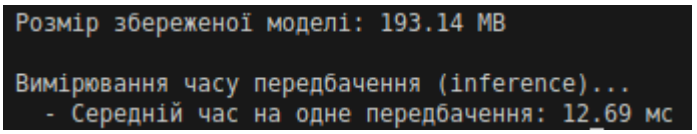


Рисунок 3.6 - Результат роботи коду виводу оцінки розміру та швидкодії моделі

Першим показником є розмір збереженого файлу моделі (best_model.keras), який становить 193.14 МБ. Це значення суттєво відрізняється від теоретичного розміру самих параметрів (64.36 МБ), вказаного в model.summary(). Ця різниця пояснюється тим, що збережений файл у форматі Keras містить не тільки ваги моделі, але й її архітектуру та, що найважливіше, стан оптимізатора Adam. Оскільки Adam для кожного параметра, що навчається, зберігає два додаткових значення (перший та другий моменти), загальний об'єм файлу приблизно втричі перевищує об'єм самих ваг (64.36 МБ · 3 ≈ 193.08 МБ) [31]. Цей показник є важливою практичною характеристикою, що визначає вимоги до дискового простору для зберігання та розгортання моделі. Варто зазначити, що для розгортання на пристроях з обмеженими ресурсами (наприклад, мобільних діагностичних апаратах) існують техніки постобробки, такі як квантизація (quantization), які дозволяють значно зменшити розмір моделі та прискорити її роботу шляхом перетворення 32-бітних ваг у менш точні 8-бітні цілі числа, часто з мінімальними втратами якості. Проте навіть у такому вигляді, модель можна вважати відносно малою.

Іншим ключовим показником є час передбачення (inference time), тобто швидкість, з якою модель може обробити одне зображення. Для отримання стабільного та усередненого результату, вимірювання проводилося шляхом виконання 100 послідовних передбачень на одному зображенні, при цьому час виконання першого виклику model.predict() ігнорувався. Це є стандартною практикою при вимірюванні продуктивності (бенчмаркінгу). Перший запуск є аномально повільним, оскільки в цей момент відбувається низка одноразових операцій, наприклад, компіляція обчислювального графа “на льоту” (Just-In-Time Compilation). Під обчислювальним графом розуміється статична структура, що

описує всі операції моделі та зв'язки між ними. TensorFlow не виконує команди Python послідовно, а спершу будує такий граф. Процес JIT-компіляції полягає в тому, що в момент першого виклику цей високорівневий граф аналізується та перетворюється на оптимізований низькорівневий код, зрозумілий для графічного процесора. Всі наступні виклики використовують вже готовий, скомпільований граф і тому виконуються значно швидше. Ігнорування першого запуску дозволяє виміряти усталену швидкість роботи моделі, виключивши з розрахунків витрати на ініціалізацію. Таким чином, середній час на одне передбачення склав 12.69 мс. Це високий показник продуктивності, що демонструє здатність моделі на тестовому обладнанні (NVIDIA GeForce RTX 3060 Ti) обробляти приблизно 78 зображень на секунду ($\frac{1000ms}{12.69ms}$). Така швидкість є більш ніж достатньою для використання моделі в практичних застосунках, включаючи системи аналізу в реальному часі.

Висока швидкість передбачення відкриває цікаві гіпотетичні можливості для практичного застосування. Замість аналізу одного статичного знімка, теоретично можна було б аналізувати відеопотік з камери очного дна в реальному часі. Оскільки пацієнт неминуче буде робити мікрорухи, а на зображенні завжди присутній певний рівень шуму, кожен кадр у потоці буде дещо унікальним [23]. Виконавши передбачення для, наприклад, 50-100 послідовних кадрів і усереднивши отримані ймовірності, можна було б отримати значно більш стабільний та надійний результат. Такий підхід, по суті, є формою усереднення (ensembling) і дозволив би нівелювати вплив випадкових артефактів або невдалого ракурсу на одному окремому знімку, що могло б призвести до помилкової класифікації. Таким чином, висока продуктивність моделі є не тільки технічною характеристикою, але й фактором, що уможливлює розробку більш надійних діагностичних протоколів у майбутньому.

3.5 Аналіз графіків процесу навчання

Найважливішим інструментом для діагностики поведінки моделі є графіки, що відображають динаміку зміни ключових показників протягом епох. Розглянемо їх послідовно.

3.5.1 Функція втрат (Loss)

На графіку 3.7 представлено динаміку зміни функції втрат (loss), яка є головним індикатором помилки моделі та тим числовим значенням, яке оптимізатор намагається мінімізувати [6].

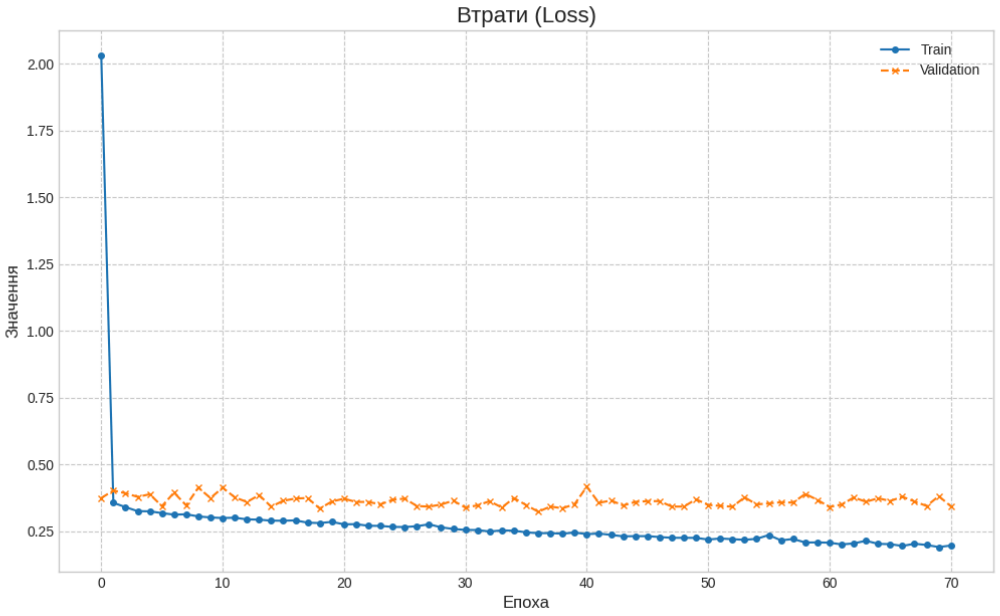


Рисунок 3.7 - Графік функції втрат (Loss) базової моделі

В даній роботі використовується binary_crossentropy, що розраховується шляхом порівняння передбачених моделлю ймовірностей з істинними мітками для кожного зображення. В ідеальному сценарії криві для тренувальної (Train loss, синя лінія) та валідаційної (Validation loss, помаранчева лінія) вибірок мали б відносно синхронно і плавно знижуватися до певного мінімуму, що свідчило б про успішне навчання та узагальнення знань. Однак спостережувана картина

демонструє класичні ознаки перенавчання (overfitting) [6]. Крива тренувальної вибірки очікувано знижується протягом усього процесу навчання, що вказує на успішну адаптацію моделі до тренувальних даних. На противагу цьому, крива валідаційної вибірки після різкого падіння на перших епохах швидко виходить на плато в районі значення ~0.35-0.40 і надалі лише незначно коливається навколо цього рівня. Значна і зростаюча з епохами розбіжність між цими двома кривими є ключовим діагностичним сигналом того, що модель перестає покращувати свою здатність до узагальнення на нових даних і переходить до процесу завчання тренувальних прикладів [29]. Велика кількість параметрів у повнозв'язному шарі дозволяє їй знаходити унікальні, не узагальнюючі патерни для конкретних зображень, які виявляються марними при зіткненні з валідаційною вибіркою. Враховуючи, що кількість параметрів моделі (16.9 млн) в тисячі разів перевищує кількість тренувальних зображень (5346), теоретично вона має достатню ємність, щоб виділити окрему групу нейронів для запам'ятовування кожного окремого зображення, замість того щоб вчитися виділяти загальні біологічні ознаки [6]. Той факт, що loss не падає до нуля навіть на тренувальній вибірці, пояснюється складністю самої функції втрат в умовах дисбалансу класів та використанням ваг, однак головним показником є саме динаміка та розбіжність кривих, які чітко діагностують перенавчання як головну проблему даної архітектури [33].

3.5.2 AUC (PR)

На графіку 3.8 представлено AUC (PR) базової моделі.

Рисунок 3.8 - Графік AUC (PR) базової моделі

Абревіатура AUC (PR) розшифровується як “Area Under the Precision-Recall Curve” - площа під кривою “точність-повнота”. Щоб зрозуміти її суть, необхідно спершу розглянути, що таке PR-крива. Кожне передбачення моделі це не бінарна відповідь “так/ні”, а ймовірність від 0 до 1. Щоб перетворити її на бінарну, використовується поріг класифікації (за замовчуванням 0.5). Змінюючи цей поріг, можна керувати балансом між двома ключовими метриками precision (точність) та recall (повнота). Наприклад, при дуже високому порозі (наприклад, 0.99) модель буде робити передбачення лише тоді, коли вона абсолютно впевнена, що призведе до високої точності, але низької повноти (більшість реальних випадків буде пропущено через недостатню впевненість). Навпаки, при дуже низькому порозі (0.01) модель буде знаходити майже всі реальні випадки (висока повнота), але ціною значної кількості хибних спрацьовувань (низька точність). PR-крива являє

собою графік, що візуалізує цей компроміс, показуючи всі можливі пари значень (recall, precision), які можна отримати, перебираючи поріг від 1.0 до 0.0 [35].

AUC (PR) є узагальнюючою, інтегральною (такою, що узагальнює результати у повному діапазоні значень, у нашому випадку у діапазоні значень порогу) метрикою, яка виражає якість цієї кривої одним числом площі під нею. Ідеальна модель, яка може досягти 100% повноти та 100% точності одночасно, мала б AUC (PR) рівний 1.0, проте на практиці такі значення, хоча і бажані, але їх майже неможливо досягти і вони можуть означати аномалію (наприклад, витік даних). Важливою особливістю цієї метрики є те, що для моделі, що робить випадкові передбачення на незбалансованих даних, базовий рівень AUC (PR) не дорівнює 0.5, як може здатися логічним інтуїтивно, а є близьким до частки позитивних прикладів у вибірці. Наприклад, якщо у вибірці лише 5% позитивних прикладів для класу або загалом (залежно від того що розглядається), то випадкове вгадування буде давати precision близько 0.05, і PR-крива буде притиснута до цього низького рівня, а її площа буде відповідною. Це робить AUC (PR) відносно чесною метрикою для незбалансованих наборів даних, адже будь-яке значення, що є суттєво вищим за цю базову лінію, свідчить про реальну предиктивну силу моделі [35].

Варто також пояснити, чому було обрано саме AUC (PR), а не AUC-ROC (площа під кривою робочих характеристик) [34]. ROC-крива будується в координатах частки істинно позитивних спрацьовувань (TPR, що є синонімом recall, вісь У) проти частки хибно позитивних спрацьовувань (FPR, вісь X). Формула для FPR (3.6):

$$FPR = \frac{FP}{FP+TN}; \quad (3.6)$$

де FP - хибні спрацювання,

TN - істинно негативні.

Для розуміння цієї та наступних формул необхідно визначити чотири базові результати бінарної класифікації для будь-якого класу [34]:

- True Positive (TP, істинно позитивний) - модель правильно передбачила наявність класу. Наприклад, модель вказала діабет, і у пацієнта дійсно є діабет.
- True Negative (TN, істинно негативний) - модель правильно передбачила відсутність класу. Наприклад, модель вказала відсутність глаукоми, і у пацієнта дійсно немає глаукоми.
- False Positive (FP, хибно позитивний) - модель помилково передбачила наявність класу, якого насправді немає. Наприклад, модель вказала глаукому, але пацієнт здоровий.
- False Negative (FN, хибно негативний) - модель помилково передбачила відсутність класу, який насправді є. Наприклад, модель вказала відсутність діабету, але у пацієнта він є.

У задачах з сильним дисбалансом, як наша, кількість TN (здорові пацієнти) є переважаючою. Це призводить до того, що знаменник у формулі FPR приймає високі значення, через що метрика стає майже нечутливою до значного зростання кількості хибних спрацьовувань FP. ROC-крива може показувати оптимістично високі значення AUC навіть для слабких моделей, які генерують велику кількість хибних тривог [35]. PR-крива, навпаки, використовує (3.7) замість FPR, (precision як вісь Y, recall як вісь X):

$$precision = \frac{TP}{TP+FP}; \quad (3.7)$$

де знаменник чутливий до зростання FP, що робить її значно більш надійним індикатором якості для дисбалансованих задач, де важливий контроль над хибними спрацюваннями.

Отриманий графік AUC (PR) підтверджує та доповнює висновки, зроблені з аналізу функції втрат. Крива тренувальної вибірки (Train AUC) стабільно зростає,

досягаючи значення ~0.52. Це свідчить про те, що на знайомих їй даних модель впевнено навчається знаходити правильний баланс між точністю та повнотою. Крива валідаційної вибірки (Validation AUC) також зростає на початкових етапах, що вказує на те, що модель дійсно вивчає певні узагальнюючі закономірності. Вона досягає свого піку ~0.36-0.38 в районі 60-70 епох. Це значення, хоч і далеке від ідеального, є кращим за випадкове вгадування і показує, що модель має певну, хоч і невисоку, предиктивну силу.

Розбіжність кривих тут також чітко виражена. Після ~20-ї епохи крива Train AUC починає значно випереджати Validation AUC. Це ще одне візуальне підтвердження перенавчання [6]. Здатність моделі розрізняти класи на тренувальних даних продовжує покращуватися, в той час як її здатність до узагальнення на нових даних виходить на плато. Саме за цією метрикою (val_auc) працював механізм ранньої зупинки, який перервав навчання, коли подальше завчання тренувальних даних перестало давати будь-яке покращення на валідаційній вибірці.

3.5.3 Точність (Precision)

Наступним для аналізу є графік 3.9 метрики Precision (Точність).

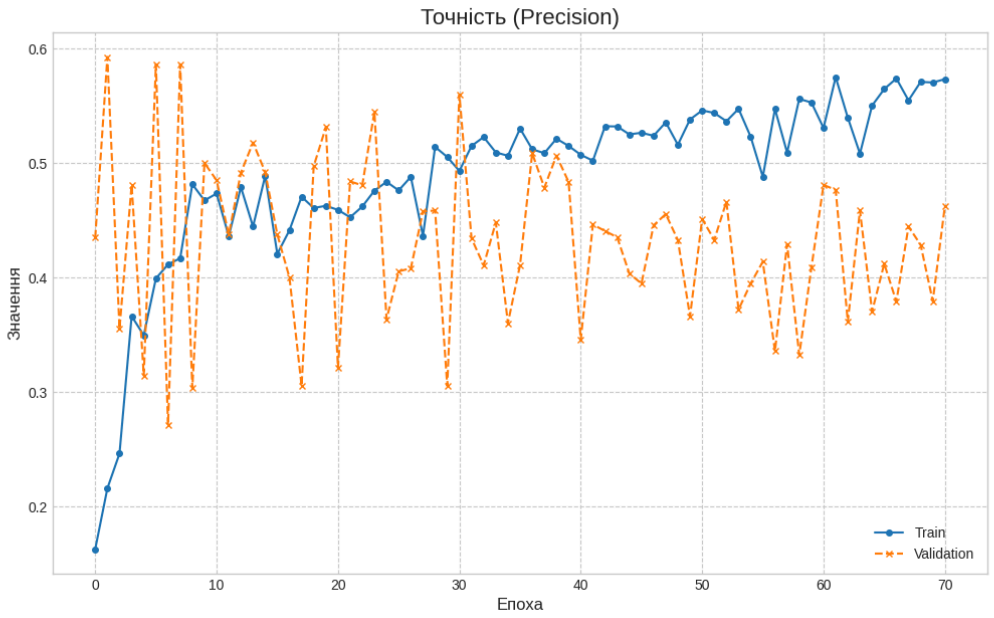


Рисунок 3.9 - Графік точності (Precision) базової моделі

Цей показник відповідає на питання: “Яка частка з передбачень, зроблених моделлю для певного класу як позитивні, дійсно були позитивними?” і розраховується за формулою (3.7) [34].

Високе значення точності свідчить про низький рівень хибних спрацьовувань (False Positives), тобто модель рідко помічає патологію там, де її немає. В ідеалі обидві криві мали б стабільно прагнути до значення 1.0.

Графік демонструє вкрай високу нестабільність метрики, особливо для валідаційної вибірки (помаранчева лінія). Крива тренувальної вибірки (Train Precision), хоч і має певні коливання, демонструє загальну тенденцію до зростання, досягаючи значень ~0.5-0.55. Це означає, що на знайомих даних модель навчається робити менше хибних спрацювань. Крива валідаційної вибірки (Validation Precision) поводиться хаотично. Наприклад, на 5-й епосі точність сягає майже 0.59, а вже на 7-й падає до ~0.26. Така висока волатильність (мінливість) є

прямим наслідком сильного дисбалансу класів та невпевненості моделі. На кожній епосі модель, через аугментацію та стохастичність (випадковість) навчання, може трохи змінити свою поведінку [24]. Припустимо, на одній епосі вона зробила лише два передбачення для рідкісного класу H на валідаційній вибірці, і одне з них виявилося правильним. Precision для цього класу буде 0.5. На наступній епосі вона може зробити три передбачення, і всі виявляться хибними, відповідно precision впаде до 0.0. Оскільки фінальна метрика precision, що відображається на графіку, є усередненням по всіх класах, навіть незначна зміна в поведінці на одному з рідкісних класів призводить до сильних коливань загального показника.

Таким чином, графік яскраво ілюструє, що precision сама по собі є недостатньою метрикою для оцінки прогресу навчання в подібних задачах. Її висока нестабільність робить її ненадійним індикатором. Саме тому для моніторингу та ранньої зупинки було обрано більш стабільну та узагальнюючу метрику AUC (PR) [35].

3.5.4 Повнота (Recall)

Наступним для аналізу є графік 3.10 метрики Recall (Повнота).

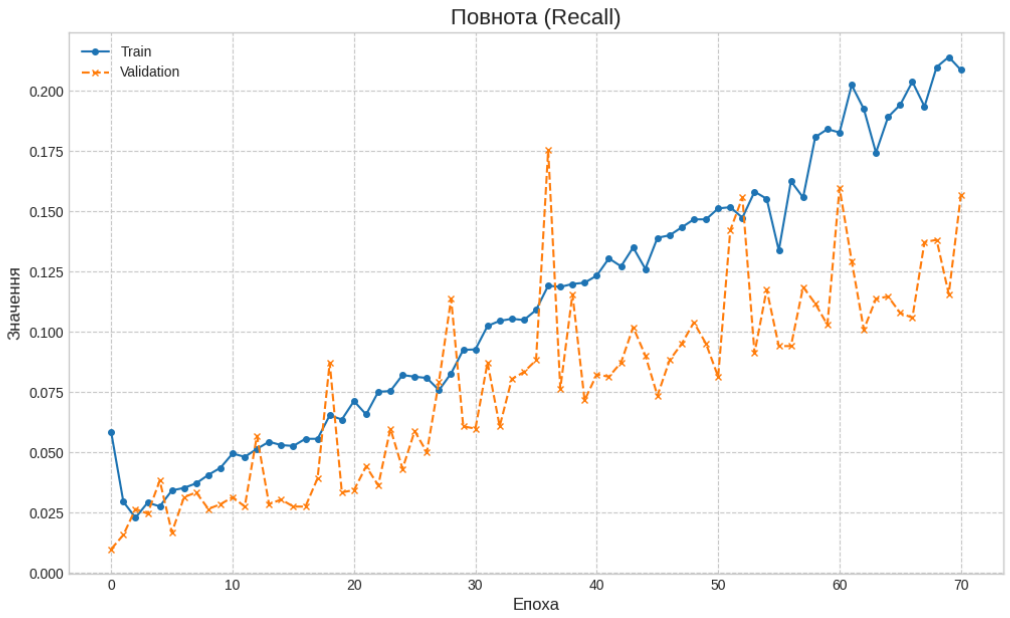


Рисунок 3.10 - Графік повноти (Recall) базової моделі

Цей показник відповідає на питання: “Яку частку з усіх реальних випадків певного класу, що були присутні у вибірці, модель змогла успішно знайти?” і розраховується за формулою (3.8):

$$recall = \frac{TP}{TP+FN}; \quad (3.8)$$

де TP (True Positives) - кількість істинно позитивних спрацьовувань,
FN (False Negatives) - кількість хибно негативних.

Високе значення повноти свідчить про низький рівень пропусків реальних захворювань (False Negatives), що є критично важливим у медичному контексті. Часто краще відправити здорову людину на додаткове обстеження (зробити хибне спрацьовування, що знизить precision), ніж пропустити реальне захворювання, яке може призвести до серйозних наслідків (зробити хибне негативне спрацьовування,

що знизить recall). Тому recall часто розглядається як одна з пріоритетних метрик для подібних задач медичної діагностики [1].

Графік повноти демонструє більш стабільну позитивну динаміку порівняно з графіком точності. Крива тренувальної вибірки (Train Recall) стабільно і майже монотонно (лінійно) зростає протягом усього процесу навчання. Це очікувана поведінка, яка показує, що завдяки використанню ваг класів, модель дійсно навчається знаходити все більше і більше прикладів патологій у тренувальних даних, поступово долаючи ефект дисбалансу [33]. Крива валідаційної вибірки (Validation Recall) також демонструє загальну тенденцію до зростання, хоч і з більшими коливаннями. Вона не виходить на чітке плато, а продовжує повільно йти за тренувальною кривою сповільнюючись у процесі відносно тренувальної створюючи розбіжність, що також є ознакою перенавчання [6].

На відміну від precision, яка може різко впасти, якщо модель зробить хоча б одне хибне передбачення для рідкісного класу, recall є більш стабільною метрикою. Її значення змінюється лише тоді, коли модель починає або перестає знаходити реальні позитивні випадки, що відбувається більш плавно.

Незважаючи на позитивну динаміку, абсолютні значення recall залишаються надто низькими. Навіть наприкінці навчання усереднене значення val_recall ледь перевищує 0.1, що означає, що модель знаходить лише близько 10% усіх наявних патологій на нових даних. Це підтверджує попередній висновок про сильне перенавчання та низьку узагальнюючу здатність базової моделі. Хоча модель і демонструє ознаки прогресу, її поточна продуктивність є абсолютно недостатньою для будь-якого практичного застосування.

3.5.5 F1-Score

На зображенні 3.11 представлено Графік F1-Score базової моделі.

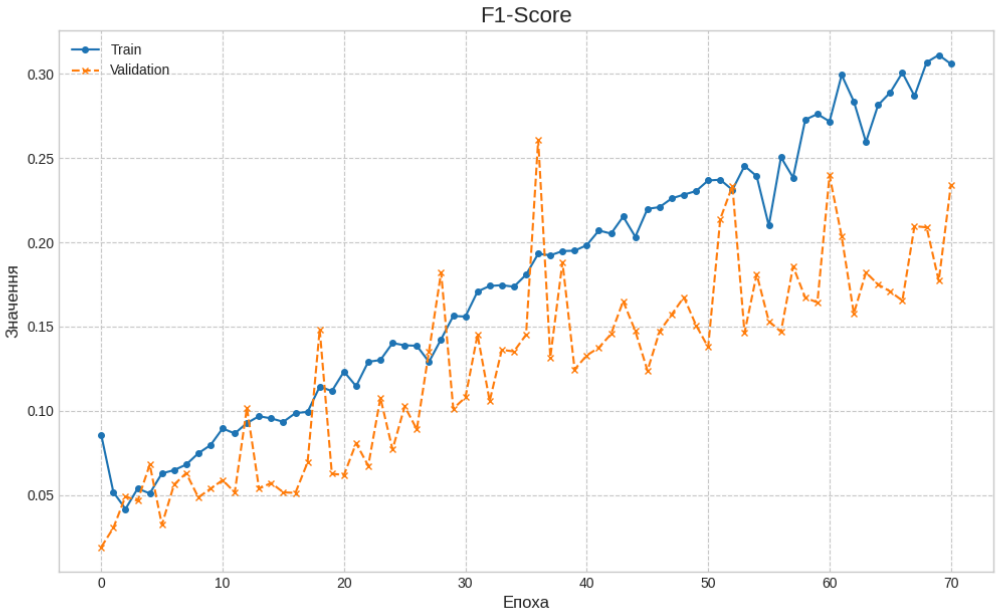


Рисунок 3.11 - Графік F1-Score базової моделі

Оскільки precision та recall часто є конкуруючими метриками (покращення однієї може призвести до погіршення іншої), для оцінки їхнього балансу використовується F1-Score. Він є гармонійним середнім між точністю та повнотою. На відміну від звичайного арифметичного середнього, яке може маскувати проблеми, якщо одна метрика дуже висока, а інша дуже низька, гармонійне середнє тяжіє до меншого зі значень. Це означає, що F1-score отримує високе значення лише тоді, коли обидва показники, precision та recall, є збалансованими та високими. Якщо хоча б одна з цих метрик близька до нуля, F1-score також буде низьким. Таким чином, F1-score можна інтерпретувати не тільки як усереднену метрику, але і як показник балансу між точністю та повнотою - чим ближчі значення precision та recall одне до одного, тим ближчим до їхнього середнього буде F1-score. Його можна розрахувати за формулою (3.9) [35]:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}; \quad (3.9)$$

Головна відмінність F1 від PR AUC полягає в тому, що F1-score, як і precision та recall, є точковою метрикою. Його значення розраховується для одного фіксованого порогу класифікації (в нашому випадку, 0.5). AUC (PR), навпаки, є інтегральною метрикою, що оцінює якість моделі по всьому діапазону можливих порогів [35]. Саме тому val_auc є більш надійним та стабільним показником для моніторингу загальної здатності моделі до навчання, в той час як val_f1_score показує її продуктивність при конкретному, не обов'язково оптимальному, порозі. Саме значення F1-score, розраховане при оптимально підібраному порозі (який максимізує цей показник), часто слугує фінальною оцінкою збалансованої якості класифікатора.

Як можна побачити, графік F1-Score є, по суті, комбінацією двох попередніх графіків і відображає їхні ключові риси. Крива валідаційної вибірки (помаранчева лінія) успадкувала високу нестабільність від метрики precision. Водночас, загальна тенденція до повільного зростання обох кривих є наслідком більш стабільного росту метрики recall. Оскільки протягом усього навчання значення повноти (Recall) було значно нижчим за значення точності (Precision), саме воно виступало обмежуючим фактором для гармонійного середнього. Через це динаміка F1-score значною мірою повторює динаміку Recall. Таким чином, F1-score візуалізує постійну боротьбу між поступовим покращенням здатності моделі знаходити патології (recall) та її нестабільною здатністю робити це без хибних спрацьовувань (precision). Також, як і у випадку з recall, фінальні значення val_f1_score є незадовільно низькими (~0.15-0.20). Це ще раз підкреслює, що, незважаючи на наявність позитивної динаміки, загальний баланс між точністю та повнотою для базової моделі залишається на незадовільному рівні.

Таким чином, аналіз трьох основних метрик (precision, recall, f1-score) дає достатньо повну і невтішну, картину. Базова модель демонструє ознаки навчання,

але страждає від сильного перенавчання, що призводить до низької та нестабільної продуктивності на нових даних [6].

3.5.6 Гістограми розподілу ймовірностей

Наступним кроком є глибокий аналіз поведінки моделі на рівні окремих передбачень. Для цього використовуються, в тому числі, гістограми розподілу ймовірностей (рис. 3.12), побудовані для кожного класу на валідаційній вибірці.

Рисунок 3.12 - Гістограми розподілу ймовірностей базової моделі

Кожен з восьми графіків візуалізує розподіл передбачених моделлю ймовірностей. Вісь X (Передбачена ймовірність) представляє вихідне значення нейрона для конкретного класу, тобто число від 0 до 1, розраховане для кожного зображення у валідаційній вибірці. Це значення інтерпретується як впевненість моделі в наявності даної патології. Вісь Y (Щільність) відображає щільність ймовірності (Probability Density). На відміну від самої ймовірності, яка за визначенням не може перевищувати 1.0, значення щільності може бути більшим за одиницю. Щоб зрозуміти, як розраховується цей показник і який його фізичний сенс, необхідно розглянути процес побудови гістограми. Спочатку весь діапазон значень на осі X (від 0 до 1) розбивається на фіксовану кількість рівних, непересічних інтервалів, також відомих як біни (bins). На графіку ширина кожного стовпчика візуально представляє один такий інтервал. Потім для кожного і-го інтервалу підраховується кількість N_i значень з вибірки, що в нього потрапили. Висота стовпчика h(i), що відповідає щільності, розраховується за формулою (3.10) [5]:

$$h(i) = \frac{N_i}{N \cdot w}; \qquad (3.10)$$

де N_i - кількість значень в і-му інтервалі,

N - загальна кількість значень у вибірці,

w - ширина одного інтервалу.

Ключовим результатом такої нормалізації є те, що сумарна площа всіх стовпчиків гістограми дорівнює одиниці. Це дозволяє коректно порівнювати два розподіли (синій та червоний) на одному графіку, навіть якщо кількість прикладів у них кардинально різна, наприклад, 900 негативних випадків і лише 40 позитивних. Без нормалізації гістограма для негативних випадків була б у десятки разів вищою, що зробило б візуальне порівняння їхніх форм неможливим. Таким чином, значення на осі Y це відносна концентрація передбачень у даному

вузькому діапазоні ймовірностей. Високе значення щільності (наприклад, 16) означає, що велика частка передбачень сконцентрована в вузькому інтервалі. Наприклад, при ширині інтервалу w=0.02, щільність h=16 означає, що в цьому інтервалі знаходиться $16 \cdot 0.02 = 0.32$, тобто 32% всіх значень даного розподілу.

Кожен графік містить два таких розподіли. Синій (Справжній Негатив) показує, які ймовірності модель видавала для зображень, де цього класу насправді не було, а червоний (Справжній Позитив), відповідно, для зображень, де цей клас дійсно був присутній. В ідеальної моделі ці два розподіли були б повністю розділені - синя гістограма була б сконцентрована у вигляді високого вузького піку біля значення 0.0, а червона біля 1.0. Чим сильніше ці два розподіли перекриваються, тим гірше роздільна здатність моделі [5].

Отримані графіки є візуальним підтвердженням та деталізацією всіх попередніх висновків. Для частих класів (N, D, O): Спостерігається значне перекриття синього та червоного розподілів. Хоча можна помітити, що центри мас червоних гістограм (позитивні випадки) все ж зміщені вправо відносно синіх, їхні розподіли здебільшого злилися в одну масу. Це візуальна демонстрація невпевненості моделі. Вона не може чітко розділити класи, і більшість її передбачень (як для позитивних, так і для негативних випадків) мають низьку ймовірність. Саме тому при порозі 0.5 модель робить так мало позитивних передбачень (recall дуже низький) і часто видає No prediction (жодне з передбачень не є вище порогу) [33].

Для рідкісних класів (G, C, A, H, M) візуальний аналіз є більш різноманітним, але загалом підтверджує нездатність моделі ефективно їх розрізняти. Для класу G (Глаукома) розподіл для негативних випадків (синя гістограма) має очікуваний високий пік біля нуля, що означає, що модель переважно правильно видає низьку ймовірність, коли глаукоми немає. Однак розподіл для позитивних випадків (червона гістограма) є майже рівномірним по всьому діапазону ймовірностей. Це свідчить про те, що модель не знайшла жодних стабільних ознак глаукоми і її передбачення для хворих очей є практично випадковими. Для класу A (Вікова макулярна дегенерація) та H (Гіпертонія)

ситуація схожа на глаукому. Сині гістограми сконцентровані біля нуля, але червоні розподіли знову є дуже розрідженими, хоча і з певною концентрацією біля зони низьких ймовірностей. Це також вказує на те, що модель не здатна впевнено ідентифікувати ці патології. Для класу M (Міопія) випадок є унікальним. Через дуже малу кількість позитивних прикладів у валідаційній вибірці, червона гістограма складається з кількох окремих стовпчиків. Цікаво, що один з них знаходиться біля значення 1.0, що свідчить про наявність певного одного, дуже характерного прикладу, який модель змогла запам'ятати. Водночас, синя гістограма також має дуже чіткий і вузький пік біля нуля. Для класу C (катаракта) випадок також є винятком, адже як і для міопії, тут можна побачити невелике, але чітке червоне скупчення біля значення 1.0 і синій пік біля значення 0. Це означає, що існують деякі специфічні, можливо, дуже явні випадки катаракти, які модель навчилася розпізнавати з високою впевненістю.

Загалом, аналіз цих гістограм для рідкісних класів пояснює, чому у фінальному звіті для більшості з них метрики дорівнювали нулю - модель майже ніколи не видає для них ймовірність, що перевищує поріг 0.5, оскільки її передбачення є або випадковими, або сконцентрованими в зоні дуже низької впевненості.

Окрім діагностики загальної поведінки моделі, ці гістограми є потужним інструментом для візуального підбору оптимального порогу класифікації для кожного класу окремо. Точка, де синій та червоний розподіли перетинаються, є точкою максимальної невизначеності для моделі. В таких точках ймовірність того, що передбачення є хибним спрацьовуванням (FP), приблизно дорівнює ймовірності того, що воно є істинно позитивним (TP). Вибір порогу відносно цієї точки дозволяє керувати балансом між точністю та повнотою. Встановлення порогу правіше від точки перетину (ближче до 1.0) означає, що модель буде робити позитивне передбачення лише при дуже високій впевненості. Це призведе до зменшення кількості хибних спрацьовувань (FP), а отже, до високої точності (precision). Однак, ціною цього буде збільшення кількості пропусків (FN), що призведе до низької повноти (recall). Такий підхід може бути виправданим у

задачах, де хибне спрацювання є дуже дорогим, наприклад, перед призначенням інвазивної процедури. Встановлення порогу лівіше від точки перетину (ближче до 0.0) означає, що модель буде вважати патологією навіть ті випадки, в яких вона не дуже впевнена. Це призведе до максимального виявлення реальних випадків (висока повнота) і зменшення кількості пропущених захворювань. Проте, це неминуче призведе до зростання кількості хибних спрацювань (низька точність). Це є типовою стратегією для скринінгових систем з високою чутливістю, де пріоритетом є не пропустити хворобу на ранній стадії [35].

Таким чином, хоча гістограми і надають інструмент для тонкого налаштування балансу між precision та recall, значне перекриття розподілів на більшості графіків свідчить про те, що для даної моделі жоден поріг не дозволить досягти одночасно високих значень обох метрик. Це ще раз підтверджує, що головною проблемою є не якість порогу, а низька розділова здатність самої моделі. Загалом, ці гістограми надають вичерпне візуальне пояснення, чому попередня оцінка моделі була настільки низькою. Вони демонструють, що головною проблемою є нездатність моделі генерувати впевнені, добре розділені передбачення, що є прямим наслідком перенавчання.

3.5.7 Залежність метрик від порогу бінаризації

Після аналізу гістограм, які візуально продемонстрували невпевненість моделі та сильне перекриття розподілів ймовірностей, логічним кроком є кількісна оцінка того, як зміна порогу класифікації впливає на фінальні метрики. Для цього було побудовано графік залежності усереднених по класах (macro) F1-score, Precision, Recall та загальної Accuracy від значення порогу бінаризації на валідаційній вибірці (рис. 3.13).

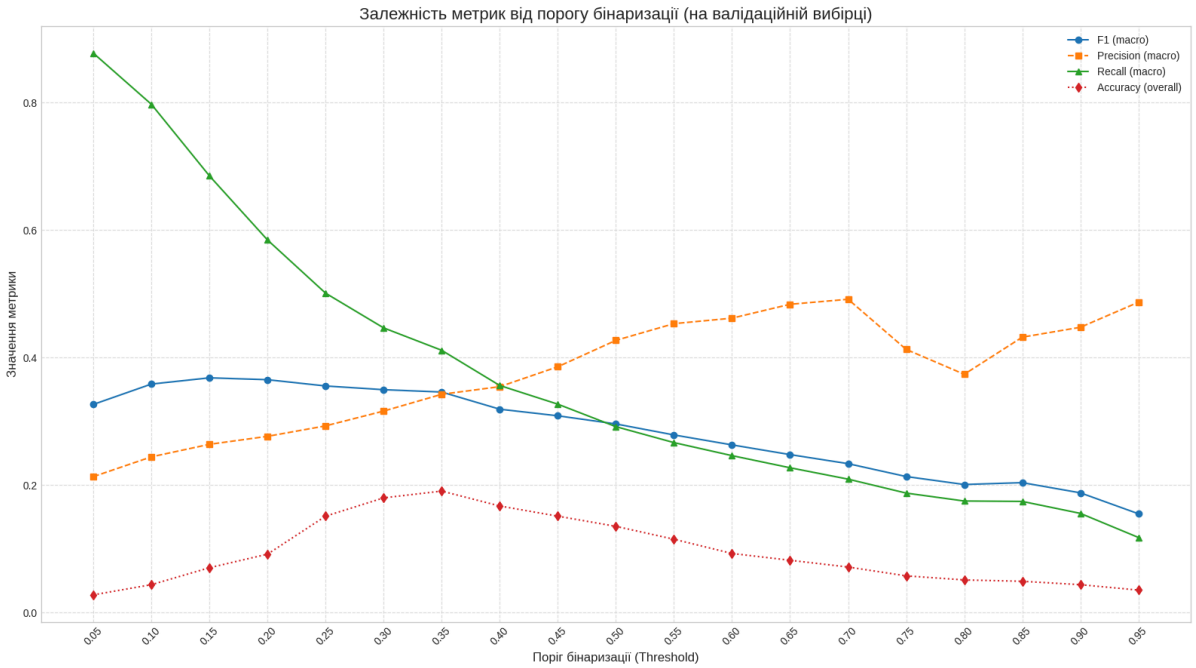


Рисунок 3.13 - Залежність метрик від порогу бінаризації базової моделі

Графік є візуалізацією компромісу між різними аспектами якості моделі. Вісь X представляє поріг класифікації, який програмно змінювався від 0.05 до 0.95 з кроком 0.05. Для кожного значення порогу всі ймовірності, передбачені моделлю для валідаційної вибірки, були перетворені на бінарні класи (0 або 1), після чого були розраховані фінальні метрики [34]. Графік наочно демонструє зворотну пропорційність між точністю та повнотою. Крива Recall (зелена), як і очікувалося, має найвище значення при найнижчому порозі (~0.88 при 0.05) і майже монотонно спадає зі збільшенням порогу, що логічно, адже чим нижчий поріг, тим більше передбачень модель робить як позитивні, а отже, тим вища ймовірність випадково винайти всі реальні позитивні випадки. Крива Precision (помаранчева) навпаки демонструє загальну тенденцію до зростання. Чим вищий поріг, тим більш впевненою має бути модель, щоб зробити позитивне передбачення, що зменшує кількість хибних спрацьовувань і підвищує точність. Крива F1-score (синя), що є гармонійним середнім між precision та recall, показує точку найкращого балансу і досягає свого максимального значення ~0.37 в при порозі ~0.15. Це і можна вважати оптимальним порогом для даної моделі, що забезпечує найкращий компроміс між знаходженням патологій та кількістю

хибних тривог [35]. Крива Accuracy (червона) досягає свого піку близько 0.35. Ця метрика показує загальну частку правильних передбачень (як позитивних, так і негативних) по всіх класах відносно всіх передбачень. Незважаючи на свою простоту та інтуїтивну зрозумілість, Accuracy (загальна точність) є оманливою для задач з сильним дисбалансом класів [33]. Модель може досягти високої загальної точності, просто навчившись завжди передбачати найчастіші клас (наприклад, “норма” та “діабет”), і при цьому повністю ігнорувати всі рідкісні патології, отримуючи при цьому значення точності близькі до ідеальних, але все ще бути не здатною до вірної класифікації більш рідкісних класів. Саме тому Accuracy не використовувалася як основна метрика для моніторингу процесу навчання, однак її аналіз в залежності від порогу може бути корисним для розуміння загальної поведінки моделі. Зокрема, він показує, що навіть при оптимальному порозі ~0.35, який максимізує загальну кількість правильних класифікацій, ця кількість все одно залишається дуже низькою (~19%), що додатково підкреслює слабкість узагальнюючої здатності моделі.

Графік підтверджує, що стандартний поріг 0.5 не є оптимальним для даної моделі. Зниження порогу до ~0.30 або нижче дозволило б суттєво покращити її збалансовану якість (F1-score). Навіть при оптимально підібраному порозі, максимальне значення F1-score (~0.37) все ще є дуже низьким. Отже, Головна проблема полягає не у виборі порогу, а в низькій предиктивній силі самої моделі, спричиненій перенавчанням [29].

3.5.8 PR-криві для кожного класу

Якщо попередні графіки показували усереднені по всіх класах метрики, то PR-криві дозволяють детально проаналізувати продуктивність моделі для кожної патології окремо (рис. 3.14).

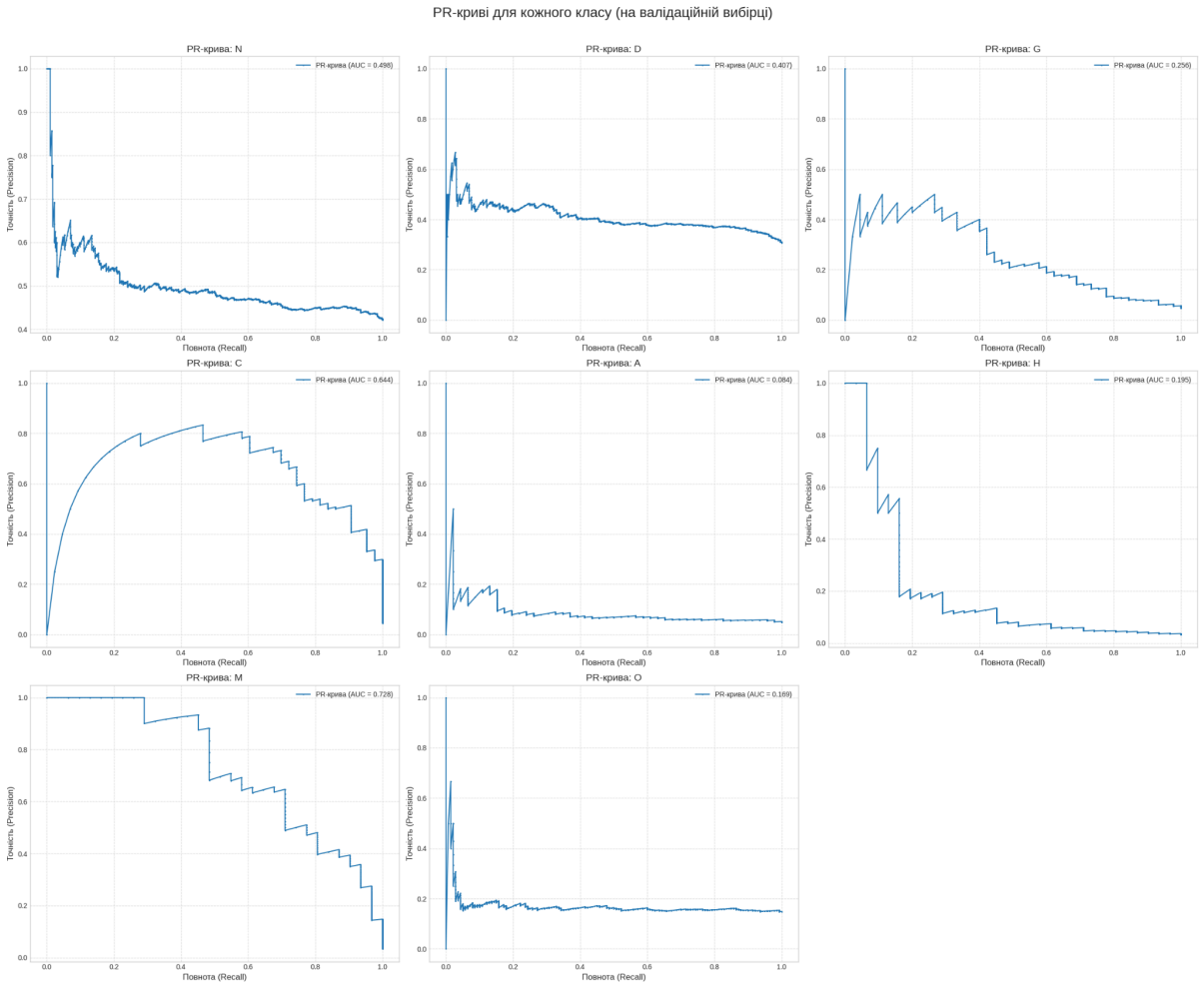


Рисунок 3.14 - PR-криві для кожного класу базової моделі

Кожен з восьми графіків візуалізує компроміс між точністю (Precision) та повнотою (Recall) для конкретного класу на валідаційній вибірці. Площа під цією кривою (AUC) є узагальнюючою оцінкою якості класифікатора для даного класу. Чим вище розташована крива і чим більше її AUC, тим краще модель здатна відрізняти цей клас від інших [35].

Окремої уваги заслуговує характерна форма деяких кривих (наприклад, для класів C, D, A, G), що починається з різкого падіння. Згідно встановленої

практики, при Recall=0 (коли ще не зроблено жодного позитивного передбачення) Precision вважається рівним 1.0, що вирішує математичну неоднозначність випадку коли обидві метрики дорівнюють 0 і встановлює початкову точку кривої в (0, 1.0). Це також забезпечує, що AUC ідеального класифікатора дорівнюватиме 1.0. Стрімке падіння кривої до нуля на самому початку означає, що одне або кілька найбільш впевнених передбачень моделі (ті, що мають найвищу ймовірність) насправді є хибними спрацюваннями (False Positives) [34]. Відростання кривої починається лише тоді, коли модель починає робити перші істинно позитивні передбачення (True Positive) в зоні менш впевнених ймовірностей. Такий патерн є важливим діагностичним сигналом, який вказує на те, що модель впевнено помиляється. Це може бути наслідком того, що модель могла сфокусуватися на хибних, нерелевантних ознаках (наприклад, артефактах на зображенні), які вона плутає з ознаками реальної патології [36]. Подальша зубчаста поведінка кривої відображає постійний компроміс між знаходженням правильних випадків та новими хибними спрацюваннями при поступовому зниженні порогу впевненості.

Ці графіки дають дуже детальну і неоднорідну картину, яка підтверджує попередні висновки. Найкращу продуктивність модель демонструє для класів C (Катаракта) та M (Міопія), з показниками AUC 0.644 та 0.728 відповідно. Їхні криві після початкових коливань досягають плато з відносно високою точністю, перш ніж почати спадати при збільшенні повноти. Це свідчить про те, що для цих патологій модель змогла вивчити певні стабільні та унікальні візуальні ознаки [23]. Високий AUC для міопії, незважаючи на її рідкісність, може пояснюватися тим, що її прояви (наприклад, зміни форми очного дна) є дуже характерними.

Для найчастіших класів N (Норма) та D (Діабет) модель показує середню продуктивність (AUC 0.498 та 0.407). Їхні криві швидко опускаються до рівня precision ~0.4-0.5 і далі йдуть майже горизонтально, що означає, що модель може досягти певної повноти, але ціною значної кількості хибних спрацьовувань.

Для решти класів модель демонструє дуже низьку продуктивність. Їхні PR-криві притиснуті до осі X, а значення AUC є низькими (від 0.084 для A до 0.256 для G). Це свідчить про те, що модель практично не навчилася розрізняти ці

патології. Для будь-якого значення повноти, вищого за нуль, точність передбачень для цих класів є дуже низькою.

Загалом, ці графіки дозволяють зробити висновок, що продуктивність базової моделі є вкрай неоднорідною. Вона змогла частково вивчити ознаки лише для кількох класів з найбільш явними та унікальними візуальними патернами, в той час як для більшості патологій її передбачення залишаються на рівні, не набагато кращому за випадковий [35].

3.5.9 Матриці помилок (Confusion Matrices)

Якщо попередні графіки показували усереднені та інтегральні метрики, то матриці помилок дозволяють детально проаналізувати абсолютну кількість правильних та неправильних передбачень для кожного класу окремо (рис. 3.15).

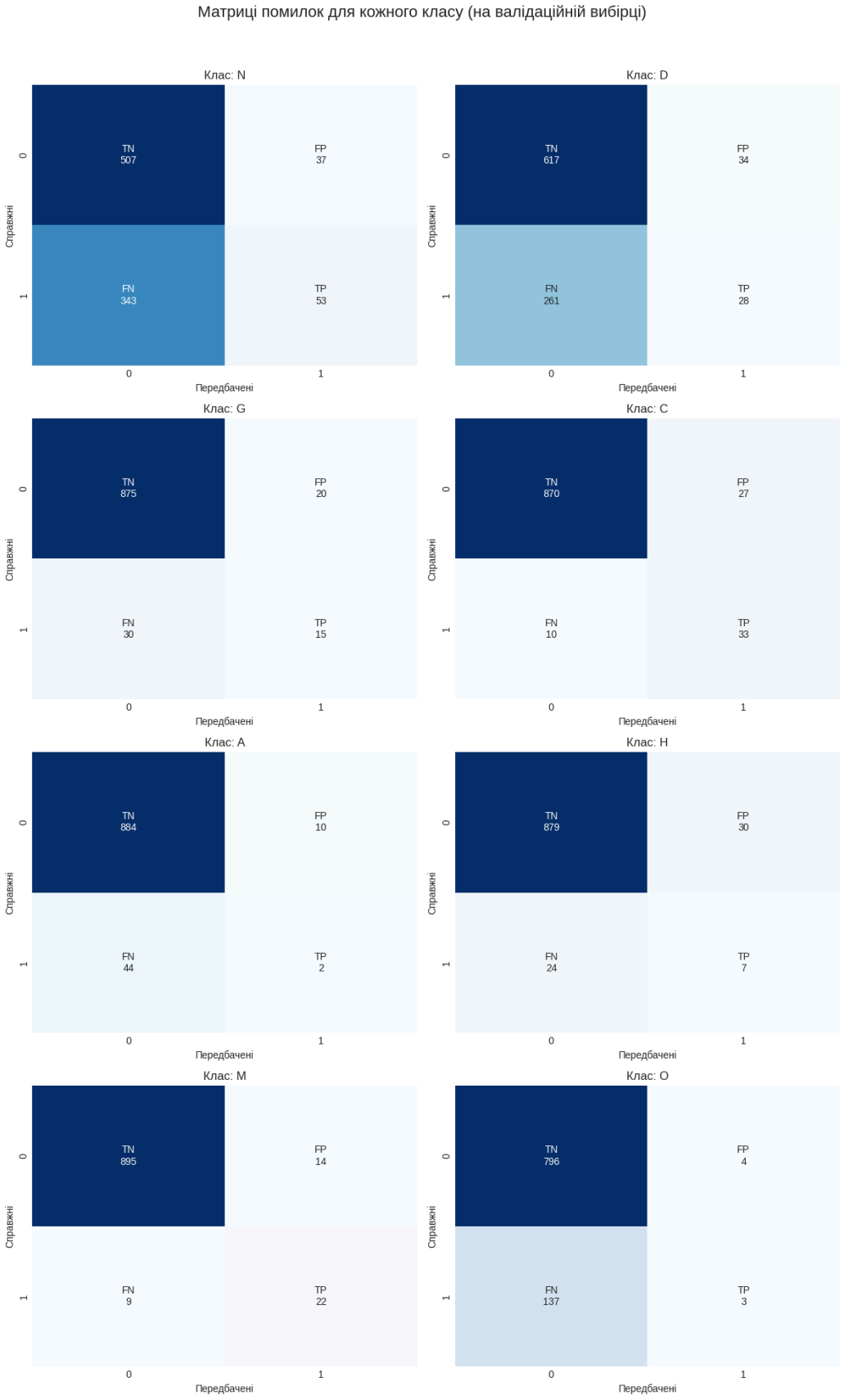


Рисунок 3.15 - Матриці помилок (Confusion Matrices) базової моделі

На зображенні представлено вісім матриць помилок, по одній для кожної патології, розрахованих на валідаційній вибірці при порозі класифікації 0.5. Кожна матриця 2x2 візуалізує чотири можливі результати класифікації: TN (True Negative) - істинно негативні, FP (False Positive) - хибно позитивні (помилка I роду), FN (False Negative) - хибно негативні (помилка II роду) та TP (True Positive) - істинно позитивні спрацювання [34].

Ідеальна матриця помилок мала б великі значення лише на головній діагоналі (TN та TP) і нулі в інших квадрантах. Натомість спостережувані матриці надають вичерпне пояснення низьких значень precision та recall, зафіксованих раніше. Вони дозволяють наочно побачити, як абсолютні числа помилок формують ці відносні метрики.

Проблема низького Recall чітко простежується через домінування значень FN над TP для більшості класів. Це демонструє, що модель пропускає переважну кількість реальних випадків. Наприклад, для класу “діабет” (D), де модель правильно знайшла лише 28 випадків (TP), а 261 пропустила (FN), можна безпосередньо розрахувати recall, що підтверджує низькі значення на графіках [34]:

$$\text{Recall (D)} = \frac{TP}{TP+FN} = \frac{28}{28+261} \approx 0.097;$$

Аналогічно, проблема низького Precision пояснюється тим, що для багатьох класів кількість хибних спрацьовувань (FP) є співмірною або навіть більшою за кількість правильних (TP). На прикладі класу G (Глаукома), де модель зробила 15 правильних передбачень (TP), але при цьому 20 разів помилково знайшла глаукому там, де її не було (FP), розрахунок precision виглядає так [35]:

$$\text{Precision (G)} = \frac{TP}{TP+FP} = \frac{15}{15+20} \approx 0.428;$$

Ці розрахунки, зроблені на основі валідаційної вибірки, є прямим числовим підтвердженням загальних тенденцій, що спостерігалися на графіках навчання. Матриці також підтверджують неоднорідну продуктивність моделі. адже для класів C (Катаракта) та M (Міопія) співвідношення TP до FN є найкращим серед усіх патологій, що збігається з висновками, зробленими з аналізу PR-кривих.

Загалом, матриці помилок є найбільш детальним доказом того, що базова модель є слабкою. Вона одночасно робить занадто багато помилок другого роду (пропускає хвороби) і помилок першого роду (вигадує їх), що робить її непридатною для практичного використання без суттєвих архітектурних покращень [1].

3.6 Візуалізація роботи моделі (Grad-CAM)

Якщо попередні метрики давали кількісну оцінку якості моделі, то для якісного аналізу її поведінки необхідно зрозуміти, на основі яких візуальних ознак вона приймає рішення [37]. Для цього використовується метод Grad-CAM (Gradient-weighted Class Activation Mapping) [36]. Цей підхід дозволяє отримати візуальну інтерпретацію роботи згорткової нейронної мережі шляхом створення теплової карти, що локалізує важливі для передбачення ділянки на вхідному зображенні.

Принцип роботи Grad-CAM ґрунтується на тому факті, що глибокі згорткові шари мережі навчаються розпізнавати складні, значущі патерни. Кожен канал (фільтр) в останньому згортковому шарі можна розглядати як детектор певної високорівневої ознаки [16]. Grad-CAM використовує інформацію про градієнти, щоб визначити, які з цих каналів зробили найбільший позитивний внесок у фінальне рішення моделі для конкретного класу.

Процес побудови теплової карти є багатоетапним. Спочатку зображення проходить через мережу (прямий прохід), в результаті чого розраховуються активації всіх шарів та фінальний вектор ймовірностей. Далі, для цільового класу c (наприклад, “діабет”), розраховується градієнт оцінки цього класу y^c по

відношенню до карт ознак $A^к$ k-го каналу в останньому згортковому шарі. Цей градієнт, що має ту ж розмірність, що і карта ознак, показує, наскільки кожен піксель цієї карти впливає на фінальне рішення. Щоб отримати єдину "вагу важливості" $\alpha_к^с$ для всього k-го каналу, ці градієнти усереднюються по всій їхній площі (Global Average Pooling) (3.11) [36]:

$$\alpha_k^c = (1/Z) * \Sigma_i \Sigma_j (\partial y^c / \partial A_{ij}^k) \text{ ;} \quad (3.11)$$

де Z - кількість пікселів у карті ознак.

Цей крок є ключовим, адже він дозволяє оцінити, наскільки важливим є k-й канал в цілому для розпізнавання класу с. Якби ми, наприклад, використовували не усереднення, а максимальне значення градієнта, то вага каналу визначалася б лише однією, можливо, шумовою точкою, що зробило б результат менш стабільним.

Фінальна теплова карта $L_{Grad-CAM}^c$ є лінійною комбінацією всіх карт активації $A^к$ з останнього згорткового шару, зважених на їхні розраховані ваги важливості $\alpha_к^с$. Після цього до неї застосовується функція активації ReLU, щоб відфільтрувати негативні та нульові внески, залишаючи лише ті ознаки, які позитивно вплинули на рішення (3.12) [36]:

$$L_{Grad-CAM}^c = ReLU(\Sigma_k \alpha_k^c A^k) \text{ .} \quad (3.12)$$

Таким чином, теплова карта візуалізує, які саме вивчені моделлю патерни були активовані і де на зображенні вона їх побачила. Для фінальної візуалізації ця карта нормалізується, забарвлюється (де гарячі кольори відповідають високій важливості, а холодні меншій) і накладається на оригінальне зображення. Це

дозволяє підтвердити і глибше зрозуміти причини успіхів та невдач моделі, що буде продемонстровано на прикладах з тестової вибірки.

Розглянемо п'ять характерних прикладів з тестової вибірки.

3.6.1 Пацієнт ID 0

На зображенні 3.16 представлено результати візуалізації Grad-CAM для пацієнта ID 0.

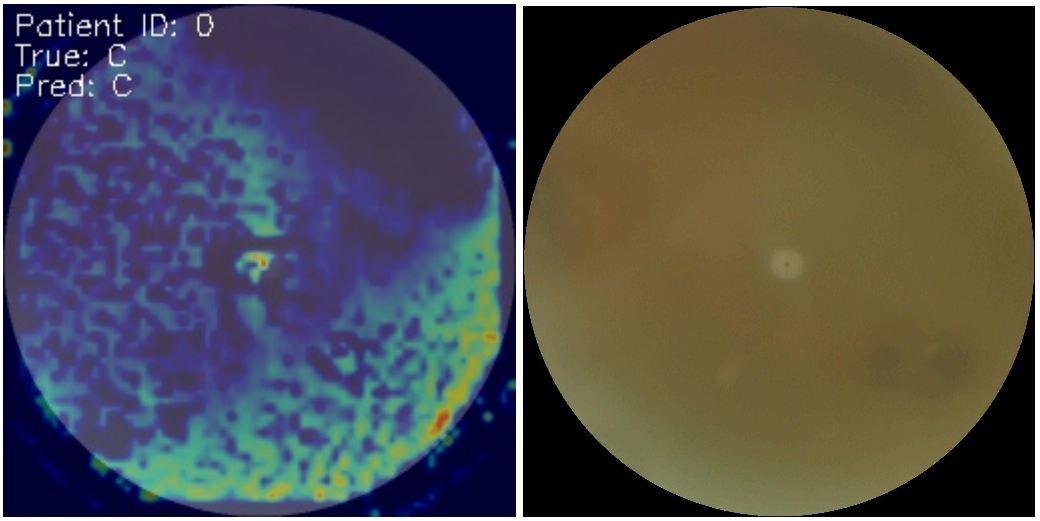


Рисунок 3.16 - Візуалізація Grad-CAM та оригінальне зображення для базової моделі для пацієнта ID 0

У першому випадку, правильне розпізнавання (ID 0, True='C', Pred='C') являє собою приклад рідкісного, але формально правильного передбачення. Оригінальне зображення є дуже розмитим, темним, без чітких анатомічних деталей, що є класичним візуальним проявом катаракти, яка заважає фокусуванню камери [23]. Теплова карта показує, що увага моделі дифузно розподілена по всій площі зображення. Це свідчить про те, що модель навчилася асоціювати не конкретні патологічні зміни, а загальну розмитість та відсутність чітких структур з класом “катаракта”. Хоча ця ознака є коректною для даної патології, покладання виключно на такий глобальний патерн, без локалізації специфічних змін, може

робити модель вразливою до хибних спрацьовувань на зображеннях, розмитих з технічних причин (наприклад, через помилку фокусування камери) [36].

3.6.2 Пацієнт ID 1

На зображенні 3.17 представлено результати візуалізації Grad-CAM для пацієнта ID 1.

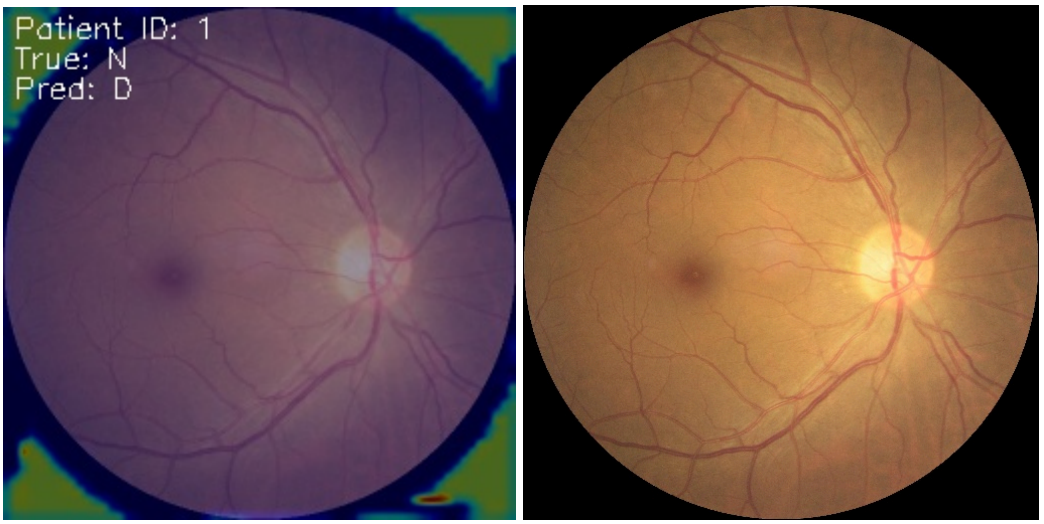


Рисунок 3.17 - Візуалізація Grad-CAM та оригінальне зображення для базової моделі для пацієнта ID 1

У другому випадку, хибне спрацювання (ID 1, True='N', Pred='D') означає, що модель помилково класифікувала діабетичну ретинопатію на абсолютно здоровому оці. Карта уваги показує, що модель сконцентрувалася виключно на артефактах по краях зображення, які є технічною особливістю обробки, а не біологічною ознакою. Модель повністю проігнорувала здорові анатомічні структури (диск зорового нерва, макулу, судини) і прийняла рішення на основі нерелевантного шуму. Це є прямим доказом того, що модель навчилася покладатися на хибні патерни [36].

3.6.3 Пацієнт ID 2

На зображенні 3.18 представлено результати візуалізації Grad-CAM для пацієнта ID 2.

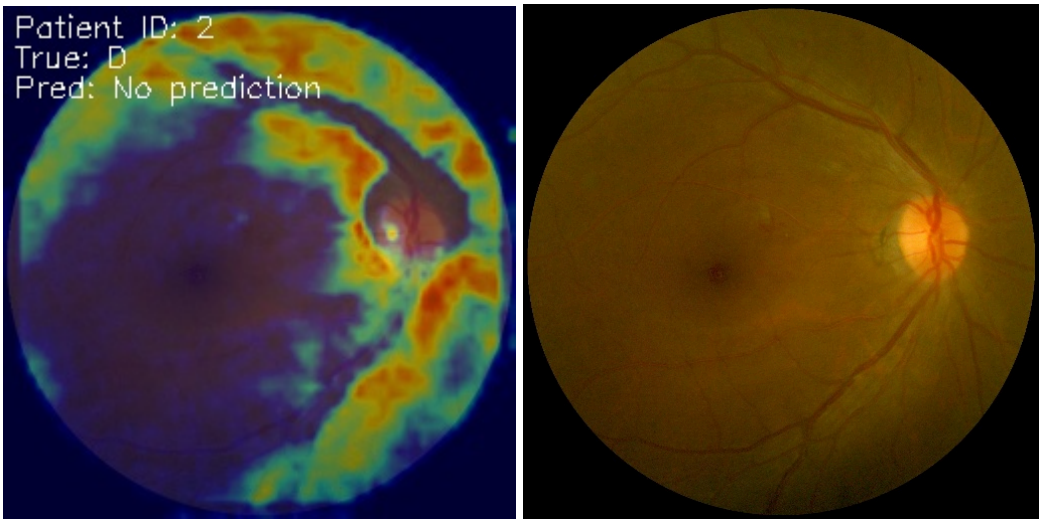


Рисунок 3.18 - Візуалізація Grad-CAM та оригінальне зображення для базової моделі для пацієнта ID 2

У третьому випадку, відбувся пропуск патології (ID 2, True='D', Pred='No prediction') незважаючи на те, що на зображенні присутні патологічні зміни і теплова карта показує, що модель здебільшого правильно сконцентрувала свою увагу на аномальних ділянках. Однак, незважаючи на коректне фокусування, модель не змогла зробити впевненого передбачення (No prediction). Це демонструє глибоку проблему перенавчання коли модель вірно визначає інформативні ділянки, але не може правильно класифікувати побачене, оскільки не вивчила узагальнюючих ознак патології [36].

3.6.4 Пацієнт ID 4

На зображенні 3.19 представлено результати візуалізації Grad-CAM для пацієнта ID 4.

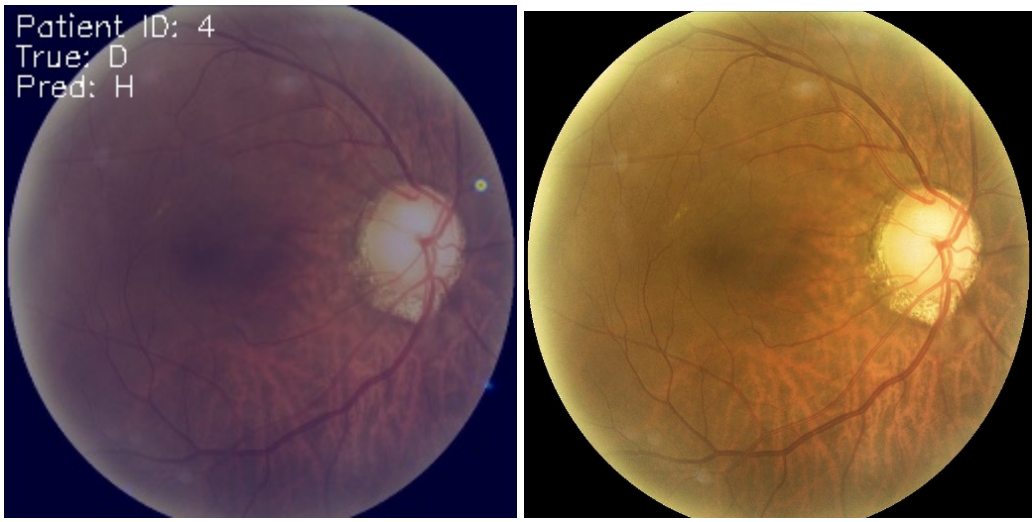


Рисунок 3.19 - Візуалізація Grad-CAM та оригінальне зображення для базової моделі для пацієнта ID 4

У четвертому випадку, плутанина між класами (ID 4, True='D', Pred='H') викликана тим, що модель переплутала діабет з гіпертонією. Оригінальне зображення має певні аномалії, що вказують на діабет, але карта уваги показує, що модель сфокусувалася на одній невеликій точці на периферії, повністю ігноруючи основні анатомічні структури. Це свідчить про повну нездатність моделі до узагальнення у цьому прикладі. Не знайшовши звичних для неї хибних патернів, вона не впоралася з задачею класифікації, взявши до уваги випадковий шум і зробила на його основі хибне передбачення.

3.6.5 Пацієнт ID 6

На зображенні 3.20 представлено результати візуалізації Grad-CAM для пацієнта ID 6.

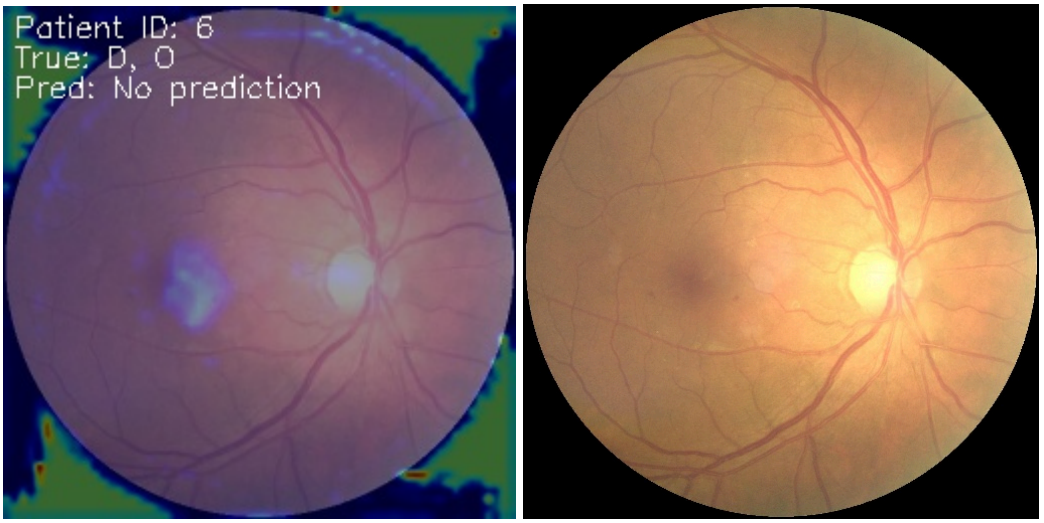


Рисунок 3.20 - Візуалізація Grad-CAM та оригінальне зображення для базової моделі для пацієнта ID 6

У п’ятому випадку, відбувся пропуск мультилейблового випадку (ID 6, True='D, O', Pred='No prediction'). На оригінальному зображенні наявні кілька класів одночасно. Однак теплова карта Grad-CAM демонструє, що модель майже повністю проігнорувала їх [36]. Її увага знову сконцентрована переважно на артефактах по краях зображення, з лише ледь помітною, слабкою активацією в районі макули очного дна. Це можна вважати ще одним доказом перенавчання, адже зіткнувшись зі складним мультилейбловим випадком, модель почала пошук патернів на межі зображення, замість аналізу біологічних ознак. Це демонструє, що здатність моделі до виділення релевантних ознак є вкрай низькою та нестабільною.

Як висновок, бачимо, що базова модель страждає від сильного перенавчання. Вона або покладається на нерелевантні артефакти, або, навіть у тих випадках, коли її увага спрямована на реальну патологію, її здатність до правильної класифікації залишається на вкрай низькому рівні.

3.7 Аналіз використання ресурсів

Окрім метрик якості, важливим аспектом оцінки моделі є аналіз її вимог до апаратних ресурсів під час навчання. На зображенні 3.21 представлено графіки використання центрального процесора (CPU), оперативної пам'яті (RAM) та відеопам'яті (VRAM) протягом усього процесу навчання.

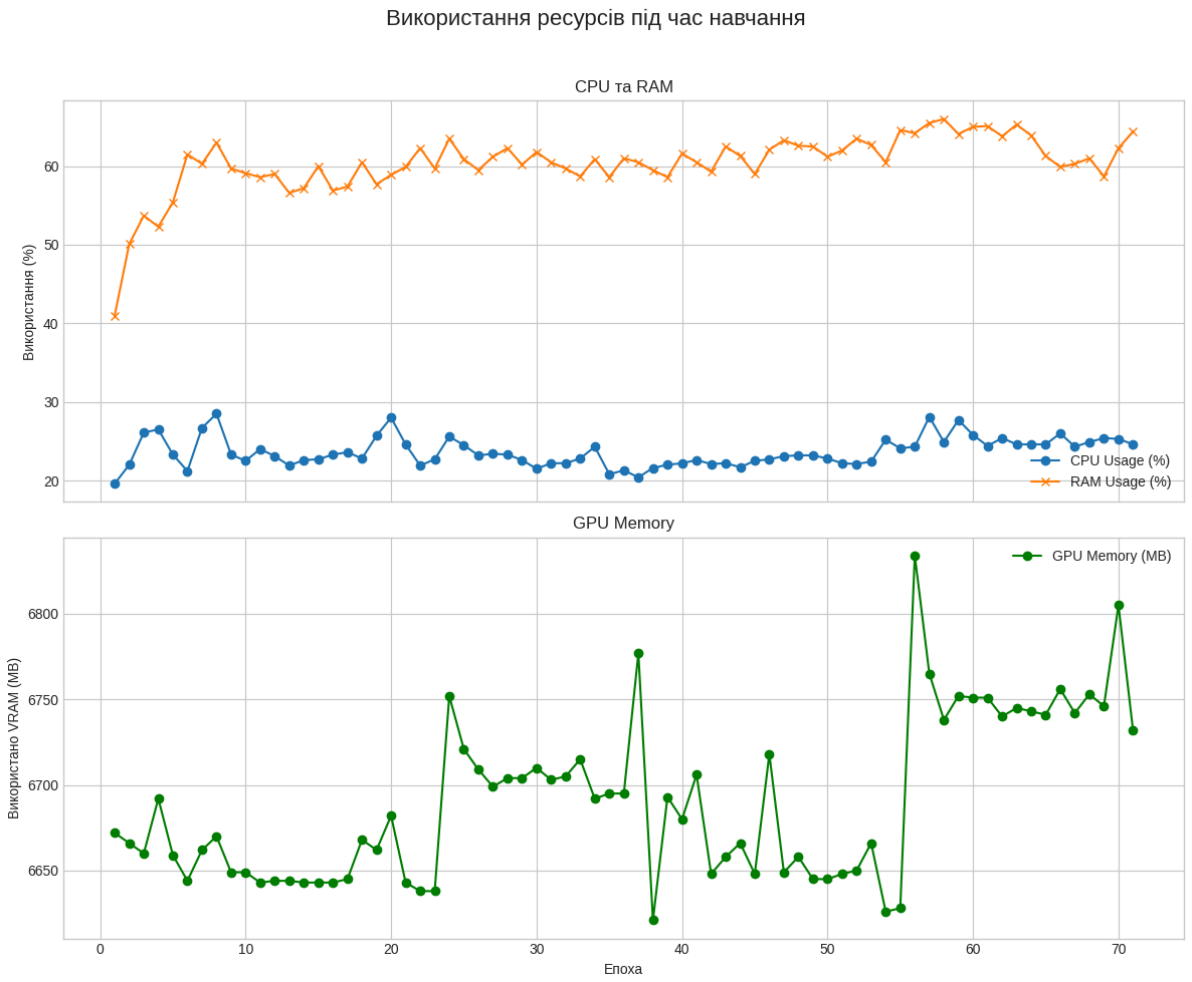


Рисунок 3.21 - Графіки використаних ресурсів базової моделі

Графіки демонструють типовий розподіл навантаження для задач глибокого навчання, де основні обчислення виконуються на графічному процесорі [6]. Верхній графік показує, що навантаження на CPU (синя лінія) залишалося відносно низьким і стабільним, коливаючись в межах 20-30%. Використання RAM (помаранчева лінія) після початкового зростання стабілізувалося на рівні ~60%. Це очікувана поведінка, оскільки CPU в даному випадку відповідає за допоміжні

задачі, такі як підготовка даних у фоновому режимі та керування процесом, в той час як GPU виконує основну обчислювальну роботу.

Нижній графік є найбільш показовим. Використання відеопам'яті (VRAM) після завантаження моделі та даних одразу стрибнуло до високого значення ~6650 МБ і залишалося на цьому рівні протягом усього навчання з незначними коливаннями. Пікове споживання досягло ~6900 МБ.

Варто детально порівняти ці практичні результати з теоретичними розрахунками. Як було показано раніше, модель має 16,871,624 параметрів, що навчаються. При використанні 32-бітного формату (float32), де кожне число займає 4 байти, можна розрахувати статичні вимоги до VRAM, що включають [6]:
Пам'ять для ваг моделі:

$$16{,}871{,}624 \cdot 4 \text{ байти} \approx 64.36 \text{ МБ};$$

Пам'ять для градієнтів:

$$16{,}871{,}624 \cdot 4 \text{ байти} \approx 64.36 \text{ МБ};$$

Пам'ять для стану оптимізатора Adam (2 моменти):

$$16{,}871{,}624 \cdot 4 \text{ байти} \cdot 2 \approx 128.72 \text{ МБ};$$

Сумарні мінімальні теоретичні вимоги складають ~257 МБ. Однак, як показав моніторинг ресурсів, пікове споживання VRAM під час навчання сягало ~6900 МБ. Така значна різниця є очікуваною і демонструє домінуючий вплив динамічних компонентів, головним з яких є пам'ять, що виділяється для зберігання проміжних активацій для кожного батча даних. Об'єм пам'яті, необхідний для зберігання вихідних тензорів (активацій) для одного батча з 32 зображень, можна розрахувати для кожного шару окремо:

input_layer (вихід: 32x256x256x3): $32 \cdot 256 \cdot 256 \cdot 3 \cdot 4 \approx 24.0$ МБ;

augmentation (вихід: 32x256x256x3): $32 \cdot 256 \cdot 256 \cdot 3 \cdot 4 \approx 24.0$ МБ;

conv2d_1 (вихід: 32x256x256x32): $32 \cdot 256 \cdot 256 \cdot 32 \cdot 4 \approx 256.0$ МБ;

max_pooling2d (вихід: 32x128x128x32): $32 \cdot 128 \cdot 128 \cdot 32 \cdot 4 \approx 64.0$ МБ;

conv2d_2 (вихід: 32x128x128x64): $32 \cdot 128 \cdot 128 \cdot 64 \cdot 4 \approx 128.0$ МБ;

max_pooling2d_1 (вихід: 32x64x64x64): $32 \cdot 64 \cdot 64 \cdot 64 \cdot 4 \approx 32.0$ МБ;

conv2d_3 (вихід: 32x64x64x128): $32 \cdot 64 \cdot 64 \cdot 128 \cdot 4 \approx 64.0$ МБ;

max_pooling2d_2 (вихід: 32x32x32x128): $32 \cdot 32 \cdot 32 \cdot 128 \cdot 4 \approx 16.0$ МБ;

flatten (вихід: 32x131072): $32 \cdot 131072 \cdot 4 \approx 16.0$ МБ;

dense (вихід: 32x128): $32 \cdot 128 \cdot 4 \approx 0.016$ МБ;

predictions (вихід: 32x8): $32 \cdot 8 \cdot 4 \approx 0.001$ МБ;

Сумарний об'єм пам'яті, необхідний лише для зберігання активацій під час одного прямого проходу, складає приблизно 624 МБ. Решта пам'яті

$$\sim 6.8 \text{ ГБ} - 257 \text{ МБ} - 624 \text{ МБ} \approx 5.9 \text{ ГБ};$$

використовується для інших цілей. Наприклад, під час зворотного поширення помилки TensorFlow повинен зберігати не тільки самі активації, але й градієнти по відношенню до них, що може майже подвоїти динамічні вимоги [6]. Також бібліотека cuDNN, що використовується для прискорення згорткових операцій, часто виділяє великі тимчасові буфери пам'яті (workspaces) для своїх алгоритмів. Об'єм цих буферів може бути значним і залежить від версії бібліотеки та архітектури GPU. Також TensorFlow не запитує пам'ять у драйвера для кожного окремого тензора. Натомість, при старті він резервує відносно великий пул пам'яті (наприклад, кілька гігабайт) і далі керує ним самостійно, щоб уникнути затримок на виділення пам'яті.

Таким чином, практика повністю підтверджує, що саме динамічні компоненти є головним обмежуючим фактором. Стабільно високе використання VRAM свідчить про те, що ресурси графічного процесора були задіяні ефективно.

3.8 Висновки по базовій моделі

Проведений всебічний аналіз побудови, навчання та тестування базової згорткової нейронної мережі дозволяє зробити низку важливих висновків. Було успішно реалізовано повний цикл роботи від підготовки даних до навчання та детального аналізу результатів. Застосування таких технік, як аугментація даних та зважування класів, продемонструвало свою ефективність у боротьбі з дисбалансом та дозволило моделі почати процес навчання, що підтверджується стабільним покращенням метрик на тренувальній вибірці [24, 33]. Однак, незважаючи на ці успіхи, головним результатом експерименту є чітка діагностика феномену сильного перенавчання (overfitting). Аналіз графіків loss та AUC (PR) показав значну розбіжність між показниками на тренувальній та валідаційній вибірках. Модель швидко завчила тренувальні дані, але її здатність до узагальнення на нових, небачених зображеннях залишилася на вкрай низькому рівні, що підтверджується низькими фінальними метриками на тестовій вибірці [6]. Причини такої поведінки є фундаментальними. Одна з них, це надлишкова ємність (overcapacity) базової архітектури. Маючи у своєму розпорядженні 16.9 мільйонів параметрів для навчання на вибірці з 5346 зображень, модель теоретично має достатню кількість штучних нейронів, щоб виділити окрему групу нейронів для запам'ятовування кожного окремого тренувального прикладу, замість того щоб вчитися виділяти загальні біологічні ознаки [6]. Також, як показав аналіз Grad-CAM, модель вдалася до навчання на хибних шляхах (shortcut learning), навчившись покладатися на нерелевантні технічні артефакти (наприклад, на межі зображення або специфічні патерни шумів), які виявилися для неї простішим патерном, ніж складні та варіативні ознаки реальних патологій [37].

Для вирішення цих проблем у теорії глибокого навчання існує набір стандартних підходів. Для боротьби з надлишковою ємністю та змушення моделі до узагальнення застосовуються методи регуляризації. Одним з найефективніших є Dropout, який згадувався раніше, тобто техніка, що випадковим чином вимикає частину нейронів під час навчання, що не дозволяє їм формувати складні співзалежності для запам'ятовування конкретних прикладів [29]. Більш фундаментальним рішенням є використання архітектур, що мають значно кращу узагальнюючу здатність за своєю будовою. Ключовою стратегією тут є трансферне навчання (transfer learning). Замість того, щоб навчати модель з нуля, можна взяти за основу потужну, попередньо навчену на мільйонах різноманітних зображень (наприклад, на наборі даних ImageNet) архітектуру [12]. Така модель вже вміє розпізнавати базові візуальні примітиви (текстури, форми, градієнти тощо) і її потрібно лише донавчити для нашої специфічної задачі (fine-tune) [25]. Це методологічно швидше та ефективніше з точки зору реалізації ніж навчання з нуля, також прискорює навчання й суттєво покращує узагальнення. Також існують більш досконалі оптимізатори, наприклад AdamW, який реалізує більш ефективний механізм регуляризації ваг (weight decay) порівняно зі стандартним Adam [32]. Окрім цього, продуктивність моделі може бути додатково покращена шляхом систематичного підбору гіперпараметрів (наприклад, швидкості навчання або архітектурних параметрів), що є окремим етапом оптимізації.

Таким чином, базова модель є непридатною для практичного використання, але за допомогою неї було розглянуто базові принципи і елементи побудови згорткових нейронних мереж та аналізу їх роботи і результатів. На її прикладі було виявлено ключові проблеми та застосовано широкий спектр інструментів для їх аналізу. Отримані результати та зроблені висновки є надійною відправною точкою і формують чітке обґрунтування для переходу до наступного розділу, де буде реалізована та проаналізована просунута модель на основі трансферного навчання з використанням технік регуляризації тощо.

4 РОЗРОБКА ТА АНАЛІЗ ПРОСУНУТОЇ МОДЕЛІ НА ОСНОВІ ТРАНСФЕРНОГО НАВЧАННЯ

Проведений у попередньому розділі аналіз базової моделі згорткової нейронної мережі дозволив зробити низку важливих висновків. Незважаючи на успішну реалізацію повного циклу роботи від підготовки даних до навчання, ключовим результатом експерименту стала чітка діагностика феномену сильного перенавчання (overfitting). Модель продемонструвала здатність до адаптації під тренувальні дані, однак її здатність до узагальнення на нових, небачених даних залишилася на вкрай низькому рівні. Це підтверджується фінальними метриками на тестовій вибірці, де усереднений показник F1-score для рідкісних класів патологій, таких як глаукома (G) та гіпертонія (H), дорівнював нулю, а загальний показник AUC (PR) не перевищував 0.3002. Аналіз візуалізацій Grad-CAM показав, що однією з фундаментальних причин такої поведінки є схильність моделі до навчання на хибних шляхах (shortcut learning), тобто використання нерелевантних артефактів (межі зображення, шумові патерни) замість справжніх біологічних ознак патологій [36]. Основною причиною цих проблем є поєднання двох факторів - надлишкової ємності архітектури та відносно невеликого обсягу тренувальної вибірки [6]. Для вирішення цих фундаментальних проблем у сучасному глибокому навчанні існує набір стандартних та ефективних підходів.

Наступним логічним кроком у дослідженні є побудова та аналіз більш просунутої моделі, яка буде спроектована спеціально для подолання виявлених обмежень базової архітектури. Ключовою стратегією для цього стане використання трансферного навчання (transfer learning) [25]. Цей підхід дозволяє використати потужну, попередньо навчену на мільйонах зображень архітектуру, що вже вміє розпізнавати базові візуальні примітиви [12, 13]. Для подальшого підвищення ефективності та боротьби з перенавчанням інтегровано сучасні методи регуляризації [29, 30], більш досконалий оптимізатор [32] та модифікована функція втрат [33] тощо. Метою цього розділу є покращення метрик якості і побудова моделі з кращою здатністю до узагальнення, що є критично важливим для її практичного застосування.

4.1 Теоретичне обґрунтування підходу

Навчання глибоких нейронних мереж з нуля є ресурсомістким процесом, що вимагає значних обсягів даних та обчислювальних потужностей. У медичній сфері, де збір та розмітка даних є дорогим і складним процесом, набори даних часто обмежені тисячами, а не мільйонами зразків. Трансферне навчання є ефективною стратегією для подолання цієї проблеми [25].

Трансферне навчання (Transfer Learning) - це методологія машинного навчання, при якій модель, розроблена для вирішення однієї задачі, повторно використовується як відправна точка для моделі в другій задачі.

В контексті комп'ютерного зору, це означає використання моделі, попередньо навченої на великому загальному наборі даних, і адаптації її для специфічної, більш вузької задачі. Основою для більшості моделей у трансферному навчанні є набір даних ImageNet [12]. Це масштабний дослідницький проєкт, що містить понад 14 мільйонів зображень, анотованих відповідно до ієрархії лексичної бази даних WordNet, яка групує об'єкти в набори синонімів (синсети) та встановлює семантичні зв'язки між ними. Для щорічного змагання ILSVRC (ImageNet Large Scale Visual Recognition Challenge) використовується його підмножина, що складається з приблизно 1.2 мільйона зображень для навчання, 50 тисяч для валідації та 100 тисяч для тестування, розподілених по 1000 категоріях об'єктів [13]. Зображення в ImageNet, хоч і мають різний розмір, для навчання моделей зазвичай приводяться до стандартного розміру, наприклад, 224x224 пікселів. Моделі, навчені на ImageNet, вивчають багату ієрархію візуальних ознак від простих (лінії, градієнти, текстури) на ранніх шарах до складних (форми, об'єкти, патерни) на глибоких шарах. Ці знання є універсальними і можуть бути успішно перенесені на інші задачі, включаючи медичну діагностику [3].

Існує дві основні стратегії застосування трансферного навчання. Перша, це вилучення ознак (Feature Extraction), в якій використовується заморожена (параметри під час навчання примусово не змінюються) згорткова основа попередньо навченої моделі як фіксований екстрактор ознак. Поверх неї додається нова, не заморожена голова (зазвичай кілька повнозв'язних шарів), яка навчається

з нуля на новому наборі даних. Друга, це тонка настройка (Fine-Tuning), в якій розморожується (повертається можливість навчання) кілька останніх шарів згорткової основи, і вони продовжують навчання на нових даних, але з дуже низькою швидкістю навчання, щоб не зруйнувати вже вивчені ваги, а лише скоригувати їх під потреби задачі [25].

Для досягнення найкращих результатів у даній роботі застосовується гібридний двоетапний підхід. На першому етапі використовується стратегія вилучення ознак, тобто усі шари базової моделі заморожуються, і навчається лише новостворена голова. Це дозволяє вагам голови стабільно адаптуватися до нових даних, не вносячи хаотичних градієнтів у предобучену основу. На другому етапі застосовується тонка настройка (Fine-Tuning). Для цього верхня частина згорткової основи розморожується, дозволяючи оновлення її ваг, і вся модель продовжує навчання з суттєво зниженою швидкістю. Логіка такого підходу полягає в тому, що більш ранні шари мережі вивчають загальні та універсальні ознаки (краї, текстури), тоді як більш глибокі шари складні та специфічні для конкретного набору даних патерни. Адаптація саме цих глибоких шарів дозволяє моделі краще налаштуватися на специфіку нової задачі, не руйнуючи при цьому фундаментальні знання, вивчені на ранніх шарах. Це дозволяє моделі тонко адаптувати свої високорівневі ознаки під специфіку нової задачі.

Також варто врахувати, що світ архітектур згорткових нейронних мереж стрімко розвивається. Після ResNet з'явилися такі потужні архітектури, як EfficientNet, ConvNeXt, і навіть абсолютно нова парадигма у вигляді Vision Transformers (ViT) [21]. Проте вибір архітектури для даної роботи керується методологічною доцільністю та логічною послідовністю дослідження. Архітектури на основі трансформерів (ViT, Swin Transformer) використовують механізм уваги (self-attention), що дозволяє моделі зважувати важливість різних частин зображення одна відносно одної, та позиційне кодування для збереження просторової інформації. Це є фундаментально іншим підходом порівняно зі згортковими мережами, і його детальний розгляд виходить за межі логічної послідовності даної роботи. Архітектура ConvNeXt, хоч і є згортковою, інтегрує

принципи, запозичені саме з трансформерів, що також потребувало б їх попереднього детального опису. Натомість, архітектура ResNet (Residual Network), представлена у 2015 році, є ідеальним кандидатом, оскільки вона вирішує фундаментальну проблему затухання градієнта, вже описану раніше, послідовно та може вважатися наступним логічним кроком в роботі [19]. До появи ResNet існувало припущення, що чим глибша мережа, тим кращою є її продуктивність. Однак на практиці дуже глибокі мережі (наприклад, VGG) страждали від деградації, адже з додаванням нових шарів точність спочатку виходила на плато, а потім починала стрімко падати [15]. Це було викликано тим, що під час зворотного поширення помилки градієнт, проходячи через безліч шарів, експоненційно зменшувався і ставав майже нульовим, унеможливлюючи ефективне навчання ранніх шарів мережі.

Загальну структуру архітектури ResNet50 можна представити у вигляді послідовних етапів або стадій. Вхідний блок (Stage 1) виконує початкове агресивне зменшення просторової розмірності зображення (з 256x256 до 64x64) та виділення базових низькорівневих ознак. Далі йдуть чотири стадії (Stages 2-5), кожна з яких побудована з декількох залишкових блоків. При переході між стадіями просторова розмірність карти ознак зменшується вдвічі, а глибина (кількість каналів) подвоюється, починаючи з 64 каналів на Stage 2 і закінчуючи 2048 каналами на виході з Stage 5. Така архітектура дозволяє мережі поступово вивчати все більш складну та абстрактну ієрархію візуальних патернів.

Ключовим нововведенням ResNet є залишкові блоки (residual blocks). Ідея полягає у створенні “коротких шляхів” або ідентифікаційних з'єднань (identity shortcut connections), які дозволяють сигналу проходити через один або кілька шарів в обхід. Таким чином, замість того, щоб змушувати шари вивчати безпосереднє відображення H(x) з входу x, вони змушені вивчати залишкову функцію F(x) = H(x) - x [19]. Фінальний вивід блоку тоді стає (4.1):

$$y = \mathcal{F}(x, \{W_i\}) + x \quad ; \tag{4.1}$$

де y - вихідний вектор ознак залишкового блоку;

x - вхідний вектор ознак блоку;

F(x, $\{W_i\}$) - залишкова функція, що вивчається шарами всередині блоку (наприклад, два згорткових шари з вагами $\{W_i\}$).

Додавання x має фундаментальне значення. Воно гарантує, що навіть якщо шари всередині блоку виявляються марними (і F(x) стане нульовим), сигнал x безперешкодно пройде далі. Під час зворотного поширення помилки градієнт може так само безперешкодно поширюватися назад через ці ідентифікаційні з'єднання, що повністю вирішує проблему затухання і дозволяє будувати надзвичайно глибокі та ефективні мережі. Крім того, така структура дозволяє мережі легко вивчити ідентифікаційне відображення (identity mapping). Якщо для оптимізації виявляється, що певний блок шарів є зайвим, оптимізатор може просто обнулити ваги W_i всередині блоку, зводячи функцію F(x) до нуля. У такому випадку вихід блоку y стає просто x, і блок перетворюється на прозорий шар, що не змінює даних. Це значно спрощує оптимізацію дуже глибоких мереж, дозволяючи їм самостійно визначати свою ефективну глибину. В архітектурі ResNet50 використовуються два типи таких залишкових блоків. Identity Block (ідентифікаційний блок) застосовується, коли розмірність вхідного та вихідного тензорів збігається. Його внутрішня структура має архітектуру вузького горла (bottleneck), збірка з трьох згорткових шарів, де перший та останній шари з ядром 1x1 відповідно зменшують та відновлюють кількість каналів, а середній шар з ядром 3x3 виконує основну просторову згортку. Convolutional Block (згортковий блок) використовується, коли необхідно змінити розмірність тензора. Його відмінність полягає в тому, що короткий шлях також містить згортковий шар, який виконує необхідне перетворення розмірності вхідного тензора перед операцією додавання Для даної роботи буде використано архітектуру ResNet50, що складається з 50 шарів (175 враховуючи службові, але не враховуючи голову і

аугментацію) і є золотим стандартом у трансферному навчанні, поєднуючи високу продуктивність та відносно помірні обчислювальні вимоги.

Важливим архітектурним елементом, що використовується для переходу від згорткової основи до повнозв'язної голови моделі, є шар GlobalAveragePooling2D (Глобальне усереднююче об'єднання). Він є сучасною та більш ефективною альтернативою шару Flatten, що використовувався у базовій моделі. Шар Flatten перетворює тривимірний тензор ознак (висота x ширина x канали) в один довгий одновимірний вектор, зберігаючи всі значення. Це призводить до значної кількості параметрів у першому повнозв'язному шарі, що є однією з причин перенавчання. Натомість, шар GlobalAveragePooling2D працює інакше беручи кожну карту ознак (кожен канал) і усереднюючи всі значення на ній до одного-єдиного числа [17]. Таким чином, тензор розмірністю (H x W x C) перетворюється на вектор довжиною C. Таким чином кількість входів для наступного повнозв'язного шару дорівнює лише кількості каналів, що робить модель значно легшою та менш схильною до перенавчання. Також усереднення робить модель більш стійкою до того, в якій саме частині зображення була знайдена ознака, що покращує її здатність до узагальнення.

Допитливий читач може помітити, що на перший погляд, вибір архітектури ResNet50, що містить близько 23 мільйонів параметрів, може здаватися контрінтуїтивним на фоні висновку про надлишкову ємність базової моделі. Однак ключова відмінність полягає не в загальній кількості параметрів, а в кількості параметрів, що навчаються з нуля. У базовій моделі всі параметри були ініціалізовані випадково і не несли жодної апріорної інформації. У випадку з ResNet50, переважна більшість її параметрів вже є навченими на наборі даних ImageNet і несуть у собі фундаментальні знання про візуальні ознаки. На першому етапі навчання (вилучення ознак), кількість параметрів, що навчаються, буде обмежена лише новою головою, що становить значно меншу частину від загальної кількості. Це кардинально знижує свободу моделі, не дозволяючи їй запамятовувати тренувальні дані, і змушує її використовувати вже існуючі знання, що є потужним механізмом регуляризації.

Окрім вибору потужної архітектури, для досягнення високих результатів необхідно використовувати сучасні оптимізатори, функції втрат та активації, що дозволяють ефективніше навчати модель та боротися з перенавчанням.

4.1.1 Оптимізатор AdamW

Adam (Adaptive Moment Estimation) є одним з найпопулярніших оптимізаторів [31], однак його стандартна реалізація L2-регуляризації є неефективною. AdamW пропонує рішення цієї проблеми через роз’єднаний спад ваги (decoupled weight decay) [32]. У стандартному оптимізаторі Adam з L2-регуляризацією, штраф за великі ваги додається безпосередньо до функції втрат. Це означає, що член регуляризації стає частиною градієнта, який потім використовується для розрахунку адаптивних моментів (ковзних середніх градієнта та його квадрата). Виникає небажаний зв’язок (coupling), через який параметри з великими вагами створюють великий градієнт регуляризації, що, в свою чергу, впливає на адаптивну швидкість навчання для всіх параметрів. Це може призводити до того, що великі ваги регуляризуються менше, ніж потрібно, особливо на пізніх етапах навчання. AdamW вирішує цю проблему через роз'єднання (decoupling) спаду ваги від кроку градієнтної оптимізації. Спочатку оптимізатор розраховує крок оновлення ваг, базуючись виключно на градієнті функції втрат. І лише після цього, окремим кроком, він зменшує ваги на величину, пропорційну їх поточному значенню. Такий підхід робить процес регуляризації більш стабільним та передбачуваним, і дозволяє спаду ваги та швидкості навчання працювати більш незалежно [32]. Оновлення ваги θ на кроці t виглядає так:
Крок оптимізатора Adam (4.2):

$$\theta_t \leftarrow \theta_{t-1} - \frac{\alpha \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}; \quad (4.2)$$

де θ_t та θ_{t-1} - параметри моделі на поточному та попередньому кроці;

α - швидкість навчання (learning rate);

$\hat{m}_t$ - скоригована оцінка першого моменту (ковзне середнє градієнтів);

$\hat{v}_t$ - скоригована оцінка другого моменту (ковзне середнє квадратів градієнтів);

ε - мале число для запобігання діленню на нуль.

Крок спаду ваги (4.3):

$$\boldsymbol{\theta_t \leftarrow \theta_t \cdot (1 - \lambda)}; \tag{4.3}$$

де λ - коефіцієнт спаду ваги (weight decay).

Такий підхід робить регуляризацію більш стабільною та передбачуваною.

4.1.2 Функція активації GELU

Проблема класичної функції ReLU полягає у вмиранні нейронів. якщо на вхід нейрона стабільно надходять значення, що призводять до негативного виходу, то градієнт для цього нейрона завжди буде дорівнювати нулю, і його ваги перестануть оновлюватися [6]. Leaky ReLU (Rectified Linear Unit з “протіканням”) є першою спробою вирішити цю проблему. Ця функція активації вводить невеликий позитивний нахил (зазвичай 0.01) для негативних вхідних значень, що гарантує наявність невеликого, але ненульового градієнта (4.4).

$$\mathbf{LeakyReLU}(\boldsymbol{x}) = \mathbf{max}(\boldsymbol{\alpha x}, \boldsymbol{x}); \tag{4.4}$$

де α - малий коефіцієнт, наприклад, 0.01.

GELU (Gaussian Error Linear Unit) є ще більш сучасною, гладкою функцією активації, що використовується у багатьох передових архітектурах [21]. Її ідея

полягає у стохастичному підході. Вихід нейрона множиться на ймовірність того, що він не буде обнулений. Ця ймовірність моделюється за допомогою кумулятивної функції розподілу Гаусса (Φ(x)) (4.5).

$$\mathbf{GELU}(x) = x \cdot \Phi(x) \; ; \tag{4.5}$$

де x - вхідне значення;

Φ(x) - кумулятивна функція розподілу стандартного нормального розподілу (Гаусса).

Це створює нелінійність, яка є гладкою і більш ефективною для навчання, оскільки вона дозволяє градієнтам краще поширюватися, особливо для значень входу близьких до нуля. Гладкість функції GELU є її ключовою перевагою. На відміну від гострого кута функції ReLU в точці x=0, GELU має нелінійну, але плавну криву, що дозволяє градієнтам більш стабільно поширюватися під час навчання.

Розглянемо кілька прикладів:

При великому позитивному значенні x=3, Φ(3) ≈ 0.998, отже

$$\text{GELU}(3) \approx 3 \cdot 0.998 = 2.994;$$

Функція працює майже як ідентифікатор.

При значенні, близькому до нуля, x=0.5, Φ(0.5) ≈ 0.691, отже

$$\text{GELU}(0.5) \approx 0.5 \cdot 0.691 = 0.345;$$

Функція плавно приглушує активацію.

При великому негативному значенні x=-3, Φ(-3) ≈ 0.0013, отже

$$\text{GELU}(-3) \approx -3 \cdot 0.0013 = -0.0039;$$

Вихідне значення близьке до нуля, але не дорівнює йому, що також запобігає вмиранню нейронів.

4.1.3 Функція втрат Focal Loss

Як було встановлено під час аналізу даних, набір даних є сильно незбалансованим. Простий метод зважування класів, хоч і корисний, не розрізняє “легкі” та “складні” приклади всередині класу. Focal Loss є динамічною функцією втрат, розробленою для вирішення цієї проблеми. Вона модифікує стандартну перехресну ентропію, додаючи до неї модулюючий множник $(1 - p_t)^{\gamma}$ (4.6) [33]:

$$\mathbf{FL}(p_t) = -\alpha_t(1 - p_t)^{\gamma}\log(p_t) \; ; \qquad (4.6)$$

де p_t - ймовірність, розрахована моделлю для істинного класу;

γ (гамма) - фокусуючий параметр, що контролює ступінь фокусування на складних прикладах (зазвичай $\gamma \geq 0$);

a_t (альфа) - опціональний ваговий коефіцієнт для балансування класів.

Коли модель дуже впевнена у правильній відповіді (наприклад, p_t наближається до 1), модулюючий множник прямує до нуля, і внесок цього “легкого” прикладу у загальну помилку значно зменшується. Коли ж модель помиляється (p_t наближається до 0), множник прямує до 1, і помилка розраховується як зазвичай. Це змушує модель фокусуватися на складних прикладах, ігноруючи ті, які вона вже добре класифікує, що значно покращує якість навчання на незбалансованих даних.

Ефект фокусуючого параметра γ можна продемонструвати на числовому прикладі. Нехай p_t - ймовірність, яку модель присвоїла правильному класу.

Легкий приклад: Модель дуже впевнена, p_t = 0.9.

При γ = 0 (стандартна перехресна ентропія), модулюючий множник дорівнює $(1\text{-}0.9)^0 = 1$.

При γ = 2, множник стає $(1\text{-}0.9)^2 = 0.01$. Внесок цього прикладу у функцію втрат зменшується у 100 разів [33].

Складний приклад: Модель не впевнена, p_t = 0.1.

При γ = 0, множник дорівнює $(1\text{-}0.1)^0 = 1$.

При γ = 2, множник стає $(1\text{-}0.1)^2 = 0.81$. Внесок цього прикладу зменшується незначно.

Таким чином, Focal Loss ефективно змушує модель ігнорувати масу легких прикладів і концентрувати обчислювальні ресурси на невеликій кількості складних випадків.

Для подальшої адаптації процесу навчання під специфіку медичних даних та вирішення додаткових проблем у роботі, виявлених під час аналізу, стандартна функція Focal Loss була модифікована. До її складу було інтегровано два додаткові компоненти, що дозволило створити комплексну функцію втрат, яка краще відповідає вимогам поставленої задачі. Перший доданий компонент, це штраф за кількість класів. Його впровадження обумовлено необхідністю контролювати схильність моделі до надмірних передбачень, коли одному зображенню може бути присвоєно велику кількість патологій одночасно. Цей механізм розраховує різницю між фактичною та прогнозованою кількістю патологій для кожного зразка і додає до загальної помилки значення, пропорційне квадрату цієї різниці. Математично цей штраф можна виразити як (4.7):

$$L_{\text{count}} = \left(\sum_{i=1}^{N} y_i - \sum_{i=1}^{N} p_i \right)^2 \cdot C \quad ; \tag{4.7}$$

де N - загальна кількість класів;

y_i - істинна бінарна мітка для і-го класу (0 або 1);

p_i - прогнозована ймовірність для і-го класу;

C - коефіцієнт, що регулює силу штрафу.

Такий підхід змушує модель бути більш консервативною, прагнучи не лише до правильної ідентифікації класів, але й до їх коректної кількості.

Другим компонентом став штраф за нелогічні комбінації, призначений для впровадження в модель базової клінічної логіки. Його мета, це запобігання ситуаціям, коли модель одночасно класифікує око як здорове (клас N) та присвоює йому будь-яку патологію. Цей механізм додає значну помилку, якщо модель одночасно з високою ймовірністю прогнозує клас 'N' та будь-який інший патологічний клас (окрім узагальнюючого класу 'O').

Таким чином, у даній роботі використовується гібридна функція втрат, що поєднує три стратегії: Focal Loss бореться з дисбалансом класів, штраф за кількість класів контролює надмірні прогнози, а штраф за нелогічні комбінації підвищує клінічну узгодженість моделі. Цей комплексний підхід дозволяє більш гнучко та цілеспрямовано керувати процесом навчання.

4.1.4 Пошук порогів та F-beta score

Стандартний поріг бінаризації 0.5, який використовується за замовчуванням, рідко є оптимальним для задач з незбалансованими класами [35]. Для підвищення практичної цінності моделі застосовується підхід з пошуком індивідуальних оптимальних порогів для кожного класу. Цей процес виконується на валідаційній вибірці після завершення навчання. Для кожного класу програмно перебираються різні значення порогу з метою знайти таке, що максимізує цільову метрику. Оскільки метрики точності (Precision) та повноти (Recall) часто є конкуруючими, для їх збалансованої оцінки використовується F1-score. Однак у медичній діагностиці пропуск реального захворювання (низький Recall) є значно більш критичною помилкою, ніж хибна тривога (низький Precision) [1]. Для врахування

цього пріоритету використовується F-beta score, як узагальнена версія F1-score (4.8) [35].

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}}; \quad (4.8)$$

де β (бета) - параметр, що визначає, у скільки разів повнота (Recall) є важливішою за точність (Precision).

При β = 1 формула зводиться до стандартного F1-score. При β > 1 перевага надається повноті і навпаки. У даній роботі для пошуку оптимальних порогів використовується значення β = 1.5, що надає більший пріоритет здатності моделі виявляти максимальну кількість реальних патологій.

4.2 Практична реалізація та архітектура трансферної моделі

Для практичної реалізації просунутої моделі було використано фреймворк TensorFlow з високорівневим API Keras. Вибір архітектури ResNet50 та стратегії трансферного навчання був реалізований шляхом завантаження попередньо навченої моделі з бібліотеки tf.keras.applications, що забезпечує доступ до стандартних, перевірених часом архітектур [19].

Зведена таблиця архітектури (табл. Г.1), яку надає метод summary(), є детальним планом побудови моделі. Вона містить чотири основні стовпчики: Layer (type) (назва та тип шару), Output Shape (розмірність тензора на виході шару), Param (кількість параметрів, що навчаються) та Connected to (шар або шари, з яких поточний шар отримує вхідні дані).

Розглянемо послідовність шарів та їх зв'язки.

Блок аугментації та підготовки даних складається з:

input_layer (InputLayer) - декларативний шар, що визначає очікувану розмірність вхідних даних (None, 256, 256, 3), де None позначає гнучкий розмір батчу.

augmentation (Sequential) - контейнер для шарів аугментації, що застосовує випадкові трансформації до вхідних зображень.

get_item, stack, add - шари, що є частиною внутрішньої реалізації функції tf.keras.applications.resnet50.preprocess_input. Вона очікує дані у форматі BGR, а не RGB, і для конвертації виконує перестановку каналів. Шари get_item виділяють кожен канал окремо, stack збирає їх у новому порядку, а add виконує віднімання середніх значень ImageNet для кожного каналу. Це є технічним кроком для приведення даних до формату, на якому навчалася ResNet50 [13].

Згорткова основа (Тіло моделі) будується як:

conv1_pad (ZeroPadding2D) - шар, що додає нульові рамки до зображення, що дозволяє ядру згортки коректно обробляти крайові пікселі [6].

Послідовність Conv2D - BatchNormalization - Activationt є стандартним будівельним блоком у сучасних CNN. Conv2D виконує операцію згортки, BatchNormalization стабілізує активації, а Activation (у даному випадку relu) вносить нелінійність [28]. Стовпчик Connected to чітко демонструє цю послідовність, де вихід шару conv1_conv подається на вхід conv1_bn, а його вихід на вхід conv1_relu.

Реалізація залишкових блоків створюється через зв'язки шарів. Наприклад, для шару conv2_block1_add (Add) стовпчик Connected to показує, що він отримує вхідні дані з двох джерел: conv2_block1_0_bn[0][0] (вихід згортки на короткому шляху) та conv2_block1_3_bn[0][0] (вихід основного набору шарів). Це є програмна реалізація операції y = F(x) + x [19]. Шар conv2_block1_out (Activation) застосовує фінальну активацію ReLU до результату цього додавання.

global_average_pooling2d (GlobalAveragePooling2D) в голові моделі отримує на вхід тензор з останнього згорткового блоку (conv5_block3_out) і виконує операцію усереднення, значно зменшуючи кількість параметрів для наступних шарів [17].

dense (Dense) та predictions (Dense) створюють послідовність повнозв'язних шарів, що формують фінальний прогноз. Шар dropout застосовується між GlobalAveragePooling2D та першим Dense шаром для регуляризації [29].

Таким чином, аналіз model.summary() дозволяє побачити, як теоретичні концепції, такі як залишкові блоки та вузьке горло, реалізуються на практиці через послідовне з'єднання та розгалуження потоків даних між шарами.

4.3 Налаштування експерименту та процес навчання трансферної моделі

Експеримент з навчання та оцінки трансферної моделі проводився в тому ж програмно-апаратному середовищі та на тому ж підготовленому наборі даних final_dataset_with_splits.csv, що й для базової моделі. Це забезпечило консистентність умов для коректного порівняння результатів. Розмір зображень (256x256) та розмір батчу (32) також залишилися незмінними.

Ключові зміни стосувалися гіперпараметрів, що безпосередньо пов'язані з новою архітектурою та стратегією навчання. Для оптимізації використовувався AdamW з коефіцієнтом спаду ваги (weight decay) 0.0001 [32]. Це невелике, але стандартне значення для weight decay, яке забезпечує м'яку L2-регуляризацію штрафуючи модель за надто великі ваги, але не настільки сильно, щоб перешкоджати навчанню складних залежностей. Процес навчання був розділений на два етапи з різними швидкостями навчання (learning rate). На етапі вилучення ознак було встановлено значення 0.001, яку можна вважати відносно високою швидкістю, що дозволяє новостворених голов швидко адаптуватися до розподілу даних. Для етапу тонкої настройки швидкість була значно знижена до 0.00001 (на два порядки), що є критично важливим для збереження попередньо навчених ваг від руйнування та дозволяє лише обережно їх коригувати.

Для реалізації модифікованої функції втрат Focal Loss були встановлені значення alpha=0.25 та gamma=2.0. Ці параметри є стандартними і рекомендовані в оригінальній статті по Focal Loss як ефективна відправна точка [33]. Значення

gamma=2.0 значно зменшує вплив легких, впевнено класифікованих прикладів, змушуючи модель фокусуватися на складних випадках. Значення alpha=0.25 надає більшої ваги рідкісним (позитивним) класам. Початкові коефіцієнти для додаткових компонентів штрафу, N_PLUS_PENALTY=0.4 та COUNT_PENALTY_FACTOR=0.1, були підібрані емпірично як компроміс між наданням достатньо сильного штрафного сигналу та уникненням дестабілізації процесу навчання. Для регуляризації класифікаційної голови використовувався шар Dropout з коефіцієнтом 0.55. Таке значення було обрано цілеспрямовано для більш агресивної боротьби з перенавчанням, враховуючи велику кількість параметрів у повнозв'язних шарах та обмежений розмір набору даних [29].

Додатковим і потужним методом боротьби з перенавчанням та підвищення узагальнюючої здатності моделі слугувала аугментація даних в реальному часі. Перед подачею в модель, кожне зображення з навчальної вибірки піддавалося набору випадкових перетворень. У даній роботі використовувався алгоритмічний конвеєр, що включав геометричні трансформації, такі як випадкові горизонтальні віддзеркалення (RandomFlip), повороти в діапазоні ±10% від повного кола (RandomRotation), масштабування до 10% (RandomZoom), зміщення ±10% по горизонталі та вертикалі а також фотометричні трансформації, що змінювали яскравість та контрастність зображення (RandomBrightness, RandomContrast) [24]. Такий підхід штучно розширює навчальну вибірку, змушуючи модель на кожній епосі бачити дещо нові варіації одних і тих же даних. Це спонукає мережу навчатися інваріантності до несуттєвих змін (таких як освітлення, ракурс, невеликі зміщення) і фокусуватися на стабільних біологічних ознаках патологій, що є ключовим для її робастності на нових, раніше не бачених даних.

Процес навчання контролювався механізмом ранньої зупинки (Early Stopping) з терпінням (patience) у 10 епох, що відстежував метрику val_auc (PR-AUC) на валідаційній вибірці. Значення 10 є компромісом, що дозволяє моделі пережити невеликі випадкові коливання якості на валідаційній вибірці, але зупинити навчання, якщо стійкого покращення не спостерігається [6].

Фінальна оцінка моделі проводилася на тестовій вибірці з використанням ваг, що показали найкраще значення val_auc протягом усього процесу навчання. Для перетворення ймовірнісних виходів моделі в бінарні класифікації застосовувалися індивідуальні оптимальні пороги. Ці пороги були знайдені на валідаційній вибірці шляхом максимізації метрики F-beta score з параметром β=1.5 [35]. Таке значення було обрано свідомо, щоб надати більший пріоритет повноті (Recall) у порівнянні з точністю (Precision), що є критично важливим для медичних задач діагностики, де пропуск реального захворювання є більш серйозною помилкою, ніж хибне спрацювання.

4.4 Результати навчання та їх аналіз

Послідовно розглянемо та проаналізуємо вивід програми, а також згенеровані нею графічні та текстові звіти, порівнюючи їх з результатами базової моделі для оцінки ефективності застосованих удосконалень.

4.4.1 Ініціалізація середовища та аналіз базових показників

Перший блок виводу програми фіксує ключові компоненти середовища (рис. 4.1).

```
TensorFlow Version: 2.20.0
GPU: NVIDIA GeForce RTX 3060 Ti

Дані завантажено. Тренувальна: 5346, Валідаційна: 940, Тестова: 52 зображень.

Розрахунок базової лінії для AUC-PR (Prevalence):
  - Клас 'N': 0.4607
  - Клас 'D': 0.2639
  - Клас 'G': 0.0492
  - Клас 'C': 0.0471
  - Клас 'A': 0.0432
  - Клас 'H': 0.0297
  - Клас 'M': 0.0417
  - Клас 'O': 0.1388
```

Рисунок 4.1 - Результат роботи коду попередньої підготовки трансферної моделі

Як і в попередньому експерименті, використовувалася версія TensorFlow 2.20.0 та графічний процесор NVIDIA GeForce RTX 3060 Ti, а дані були успішно

завантажені та розділені на вибірки (5346 тренувальних, 940 валідаційних та 52 тестових зображення), що підтверджує консистентність умов проведення обох експериментів.

Важливим доповненням на цьому етапі став додатковий розрахунок базової лінії для метрики AUC-PR, яка визначається поширеністю (prevalence) кожного класу. На відміну від метрики AUC-ROC, де базовий рівень для випадкового класифікатора завжди дорівнює 0.5 [34], для AUC-PR ця базова лінія є плаваючою і прямо залежить від дисбалансу класів. Вона розраховується як частка позитивних прикладів у вибірці (4.9) [35]:

$$B_c = \frac{N_{\text{pos},c}}{N_{\text{total}}}; \tag{4.9}$$

де B_c - базова лінія (Prevalence) для класу c;

$N_{pos,c}$ - кількість позитивних прикладів для класу c у тренувальній вибірці; N_{total} - загальна кількість прикладів у тренувальній вибірці.

Це значення показує, якої середньої точності (Precision) досяг би наївний класифікатор, що робить передбачення випадковим чином. Таким чином, модель можна вважати ефективною лише в тому випадку, якщо її показник AUC-PR для певного класу значно перевищує цю базову лінію [35].

Як видно з результатів, базові лінії суттєво відрізняються для різних класів. Для найчастішого класу 'N' (Норма) вона становить 0.4607, тоді як для найрідкіснішого класу 'H' (Гіпертонія) лише 0.0297. Це означає, що для визнання моделі ефективною, її показник AUC-PR для класу 'H' має бути значно вищим за ~0.03, тоді як для класу 'N' навіть результат 0.5 вже свідчитиме про відносно невеликий прогрес у порівнянні з випадковим вгадуванням. Середнє значення базової лінії для всіх восьми класів становить приблизно 0.134. Це усереднене значення дає загальне уявлення про складність задачі в умовах сильного

дисбалансу і слугуватиме важливим орієнтиром при подальшій оцінці усереднених метрик моделі.

4.4.2 Процес навчання трансферної моделі

Процес навчання трансферної моделі, як і було заплановано в методології, проходив у два етапи початкового навчання голови (Feature Extraction) та подальшої тонкї настройки (Fine-Tuning) частини базової архітектури [3].

На першому етапі, коли навчалися лише новостворені повнозв'язні шари, а основа ResNet50 була заморожена, спостерігалася стрімка позитивна динаміка. Вже протягом перших епох ключова метрика val_auc (PR-AUC) значно перевищила пікові значення, досягнуті базовою моделлю за весь час її навчання. Це свідчить про значну ефективність трансферного навчання: попередньо навчені фільтри ResNet50 одразу почали постачати класифікаційній голові якісні та інформативні ознаки, що дозволило моделі швидко адаптуватися до нової задачі [16].

Після стабілізації метрик на першому етапі був здійснений перехід до тонкої настройки. Останні шари згорткової основи були розморожені, а швидкість навчання значно знижена. Цей перехід супроводжувався подальшим, хоч і менш стрімким, але стабільним покращенням показника val_auc, що демонструє, що модель змогла успішно адаптувати свої високорівневі фільтри під специфіку офтальмологічних зображень, не руйнуючи при цьому вже вивчені базові патерни [25].

Весь процес навчання контролювався механізмом ранньої зупинки [6]. Загалом навчання тривало 84 епохи, після чого було автоматично зупинено, оскільки метрика val_auc на валідаційній вибірці перестала демонструвати значуще покращення. Для фінальної оцінки була використана версія моделі з тієї епохи, де був зафіксований найвищий показник val_auc (0.60753) протягом усього процесу навчання.

4.5 Оцінка моделі на тестовій вибірці

Після завершення процесу навчання була проведена фінальна оцінка найкращої версії моделі на тестовій вибірці.

4.5.1 Загальні метрики та оптимальні пороги

Перший блок звіту надає загальні метрики та перелік оптимальних порогів класифікації, знайдених на валідаційній вибірці (рис. 4.2).

```
Загальні метрики на тестовій вибірці:
  - loss: 3.0075
  - compile_metrics: 0.5297

Оптимальні пороги (максимізація F-beta, beta=1.5):
  - Клас 'N': 0.2648 (F-beta: 0.7653)
  - Клас 'D': 0.2005 (F-beta: 0.6629)
  - Клас 'G': 0.2321 (F-beta: 0.4425)
  - Клас 'C': 0.1287 (F-beta: 0.8464)
  - Клас 'A': 0.2468 (F-beta: 0.4914)
  - Клас 'H': 0.2214 (F-beta: 0.2633)
  - Клас 'M': 0.1450 (F-beta: 0.8966)
  - Клас 'O': 0.2282 (F-beta: 0.4554)
```

Рисунок 4.2 - Результат роботи коду оцінки найкращої моделі на тестовій вибірці для трансферної моделі

Загальне значення функції втрат (loss) на тестових даних склало 3.0075. Важливо відзначити, що пряме порівняння цього значення з loss базової моделі (0.5197) є некоректним, оскільки для навчання просунутої моделі використовувалася значно складніша кастомна функція втрат з додатковими штрафами, що природно призводить до вищих числових значень [33].

Ключовим елементом постобробки є використання індивідуальних порогів для кожного класу. На відміну від стандартного порогу 0.5, ці значення (наприклад, 0.2648 для класу 'N' та 0.2005 для класу 'D') були підібрані для максимізації метрики F-beta [35]. Це демонструє, що для досягнення оптимального балансу між точністю та повнотою, модель не вимагає високого рівня впевненості у своїх прогнозах [5]. Для класів з високим показником F-beta (C, M), оптимальний поріг виявився ще нижчим (~0.13-0.14), що вказує на

наявність чітких, але слабких за амплітудою сигналів, які модель навчилася розпізнавати. Водночас для класів, з якими у моделі виникли труднощі (G, H, A), оптимальний поріг знаходиться в діапазоні ~0.22-0.25, що свідчить про невпевненість моделі та відсутність чітко виражених патернів.

4.5.2 Оцінка продуктивності на тестовій вибірці

Аналіз звіту показує як сильні, так і слабкі сторони просунутої моделі (рис. 4.3).

```
Детальний звіт по класах (оптимальні пороги, beta=1.5):
              precision    recall  f1-score   support

           N       0.34      1.00      0.51        15
           D       0.61      0.91      0.73        22
           G       0.00      0.00      0.00         5
           C       1.00      0.33      0.50         6
           A       0.33      1.00      0.50         2
           H       0.00      0.00      0.00         2
           M       0.67      1.00      0.80         2
           O       0.65      0.57      0.60        23

   micro avg       0.50      0.70      0.58        77
   macro avg       0.45      0.60      0.46        77
weighted avg       0.54      0.70      0.56        77
 samples avg       0.52      0.75      0.58        77

Аналіз результатів по тестових групах

Пацієнт ID: 0 (2 зображення)
  - Середня схожість (Jaccard): 100.00%
    - 0_right.jpg: True='N', Pred='N' (Схожість: 100.00%)
    - 0_left.jpg: True='C', Pred='C' (Схожість: 100.00%)

Пацієнт ID: 1 (2 зображення)
  - Середня схожість (Jaccard): 50.00%
    - 1_right.jpg: True='N', Pred='N, D' (Схожість: 50.00%)
    - 1_left.jpg: True='N', Pred='N, D' (Схожість: 50.00%)

Пацієнт ID: 2 (1 зображення)
  - Середня схожість (Jaccard): 50.00%
    - 2_right.jpg: True='D', Pred='N, D' (Схожість: 50.00%)

Пацієнт ID: 4 (2 зображення)
  - Середня схожість (Jaccard): 50.00%
    - 4_right.jpg: True='D', Pred='D' (Схожість: 100.00%)
    - 4_left.jpg: True='O', Pred='D, H' (Схожість: 0.00%)

Пацієнт ID: 6 (2 зображення)
  - Середня схожість (Jaccard): 50.00%
    - 6_right.jpg: True='D, O', Pred='N, D, O' (Схожість: 66.67%)
    - 6_left.jpg: True='O', Pred='N, A, O' (Схожість: 33.33%)

Пацієнт ID: 7 (2 зображення)
  - Середня схожість (Jaccard): 41.67%
    - 7_right.jpg: True='D', Pred='N, D' (Схожість: 50.00%)
    - 7_left.jpg: True='O', Pred='N, D, O' (Схожість: 33.33%)

Пацієнт ID: 8 (2 зображення)
  - Середня схожість (Jaccard): 75.00%
    - 8_right.jpg: True='N', Pred='N, D' (Схожість: 50.00%)
    - 8_left.jpg: True='N', Pred='N' (Схожість: 100.00%)

Пацієнт ID: 9 (2 зображення)
  - Середня схожість (Jaccard): 41.67%
    - 9_right.jpg: True='O', Pred='N, D, O' (Схожість: 33.33%)
```

Рисунок 4.3 - Результат роботи коду детального звітування по тестовій вибірці для трансферної моделі

Найбільш помітний прогрес спостерігається для найчастіших патологічних класів. Для класу 'D' (Діабет) метрика F1-score зросла більш ніж у 10 разів (з 0.07 до 0.73). Для класу 'O' (Інше) F1-score покращився в 4 рази (з 0.15 до 0.60). Також значно покращилася продуктивність для класу 'M' (Міопія), де F1-score досяг 0.80 (проти 0.00 у базової моделі). Це є прямим доказом того, що трансферне навчання та досконаліші методи регуляризації дозволили моделі вивчити стабільні та узагальнюючі ознаки цих патологій [25]. Водночас, як і базова модель, трансферна архітектура повністю не впоралася з найрідкіснішими класами 'G' (Глаукома) та 'H' (Гіпертонія), де всі метрики дорівнюють нулю, що є очікуваним наслідком екстремального дисбалансу даних [33]. Усереднені метрики, такі як macro avg F1-score (0.46), показують значне покращення порівняно з базовою моделлю (0.15), однак все ще залишаються на середньому рівні.

Для більш якісної оцінки поведінки моделі на індивідуальних прикладах був проведений аналіз по тестових групах пацієнтів з використанням індексу схожості Жаккара (Jaccard Index). Ця метрика оцінює ступінь перетину між множиною істинних міток (A) та множиною прогнозованих (B), і розраховується за формулою (4.10) [5]:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}; \tag{4.10}$$

Значення 1.0 означає повний збіг, а 0.0 повну розбіжність. Середня схожість по всіх тестових зразках склала 46.44%. Цей показник слід інтерпретувати як те, що в середньому модель правильно визначає близько половини аспектів діагнозу для кожного зображення, що є значним прогресом у порівнянні з базовою моделлю.

Детальний аналіз по пацієнтах виявляє характерні патерни поведінки моделі. Спостерігаються як випадки ідеально точних прогнозів (Пацієнт ID: 0, 50), так і типові помилки, що дозволяє оцінити ефективність кастомної функції втрат.

Штраф за кількість класів виявився ефективним, адже модель утримується від надмірних прогнозів, і максимальна кількість одночасно передбачених класів не перевищує трьох. Однак штраф за норму і патологію у випадку коли вони прогнозовані одночасно виявився менш дієвим. Модель часто вдається до паттернів комбінованих прогнозів, таких як N, D або N, O (пацієнт ID: 1, 8), або D, O (пацієнт ID: 26, 50, 1994) або N, D, O (пацієнт ID: 6, 7, 9, 11, 26). Оскільки 'N', 'D' та 'O' є найчастішими класами, модель, ймовірно, знайшла локальний мінімум, де їх комбінації дозволяють мінімізувати загальну помилку, незважаючи на штраф [6].

Незважаючи на ці обмеження, поведінка моделі вказує на її потенціал як допоміжного інструменту. Вона часто правильно ідентифікує основну патологію або декілька, хоча й додає до неї шумові прогнози. З огляду на високі показники повноти (Recall) для ключових класів, таку модель обмежено (через нездатність роботи з окремими класами та схильність до патернів передбачень), але можна розглядати як інструмент “другої думки” або попереднього скринінгу для звуження кола пошуку, який хоч і не є безпомилковим, але здатний привернути увагу лікаря до потенційно проблемних випадків [1].

4.6 Аналіз графіків процесу навчання

На відміну від фінальних метрик, ці графіки дозволяють оцінити сам процес навчання та ефективність застосованих стратегій.

4.6.1 Функція втрат (Loss)

Графік функції втрат 4.4 наочно демонструє вплив двоетапної стратегії навчання на стабільність моделі. Процес чітко розділений на два періоди

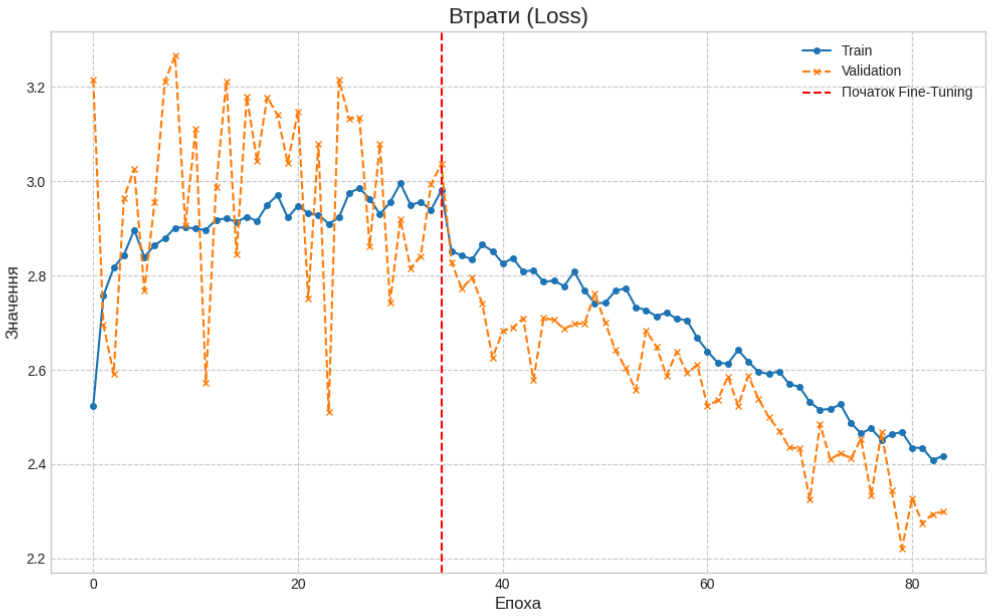


Рисунок 4.4 - Графік функції втрат (Loss) трансферної моделі

На етапі вилучення ознак (епохи 0-36) коли навчалася лише голова моделі, спостерігається висока волатильність, особливо для валідаційної кривої втрат (Validation loss). Крива тренувальних втрат (Train loss) показує тенденцію до зростання, що є очікуваною поведінкою на початкових етапах, коли модель переходить від випадкових прогнозів до осмислених, а складна функція втрат починає активно штрафувати за помилки. Висока швидкість навчання на цьому етапі призводить до значних коливань, оскільки модель робить великі кроки в просторі параметрів, шукаючи оптимальне рішення.

Одразу після переходу до тонкої настройки (епохи 36-84), позначеного вертикальною лінією, поведінка моделі кардинально змінюється. Обидві криві, як тренувальна, так і валідаційна, починають стабільно і синхронно знижуватися. Це є прямим наслідком значного зниження швидкості навчання, що дозволило моделі обережно коригувати ваги предобученої основи [3]. Важливо відзначити, що розрив між двома кривими залишається мінімальним і не зростає. Більше того, валідаційна крива втрат стабільно знаходиться нижче тренувальної. Це явище є індикатором ефективної регуляризації та відсутності перенавчання. Воно пояснюється тим, що під час тренування модель працює в ускладнених умовах, адже частина її нейронів деактивована шаром Dropout [29], а вхідні дані піддаються аугментації [24]. На етапі валідації, навпаки, Dropout вимикається, і модель працює на повну потужність з оригінальними, чистими даними, що і призводить до нижчого значення помилки. На відміну від базової моделі, де криві loss помітно розходилися, тут покращення на тренувальних даних призводить до аналогічного покращення на нових, валідаційних даних, що свідчить про високу узагальнюючу здатність моделі [6].

4.6.2 AUC (PR)

Графік метрики AUC (PR) 4.5, яка є площею під кривою точність-повнота на етапі вилучення ознак (епохи 0-35) демонструє, що тренувальна крива стрімко зростає, що свідчить про швидке початкове навчання моделі.

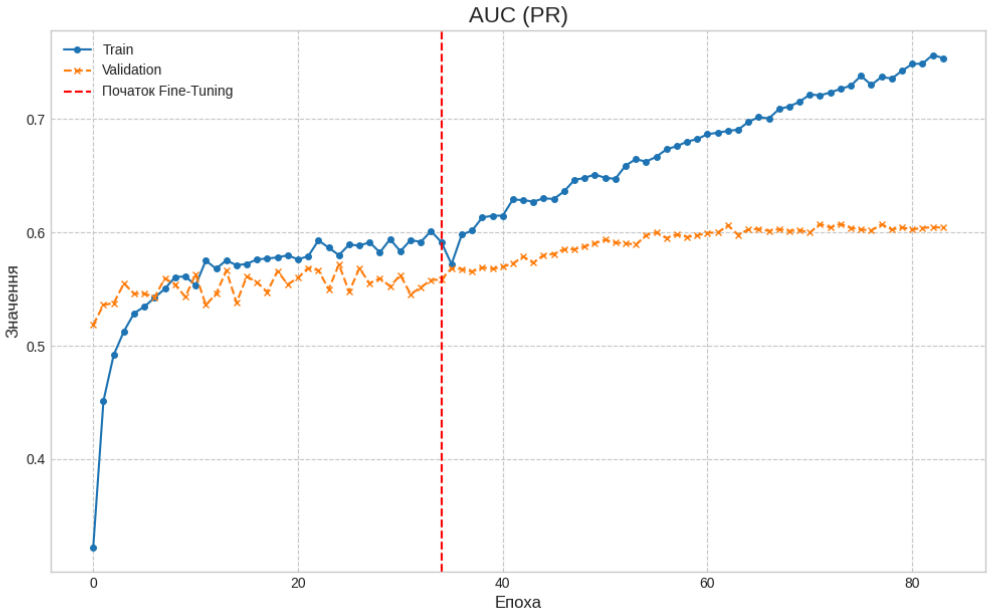


Рисунок 4.5 - Графік AUC (PR) трансферної моделі

Важливо відзначити, що крива валідації (Validation) знаходиться дуже близько до тренувальної (Train), з мінімальним відставанням, але не демонструє значного росту після перших епох досягаючи значень у 0.58. Це кардинально відрізняється від поведінки базової моделі, де розходження кривих починалося значно раніше і було більш вираженим. Така близькість кривих на початковому етапі підтверджує ефективність замороженої основи ResNet50 як потужного екстрактора ознак [25].

Після переходу до етапу тонкої настройки (з 36-ї епохи) динаміка змінюється. Крива валідації знову починає зростання, але більш плавно за рахунок меншої швидкості навчання, поступово виходячи на плато в діапазоні ~0.60. Водночас тренувальна крива продовжує впевнено зростати, що призводить до поступового, але помітного розходження між ними. Це розходження є очікуваним ефектом перенавчання, яке неминуче виникає при навчанні глибоких мереж та схоже на поведінку базової моделі [6].

Фінальне значення val_auc на рівні 0.608 можна вважати прогресом у порівнянні з піковим значенням базової моделі (~0.38). Також, воно демонструє

значну перевагу над рівнем випадкового вгадування. Порівнюючи цей результат з раніше розрахованими базовими лініями (prevalence), можна зробити наступні висновки [35]. Досягнутий показник у ~4.5 рази перевищує середню базову лінію (0.134), що свідчить про високу узагальнюючу предиктивну силу моделі в цілому. Відносно найчастішого класу 'N' (норма), де базова лінія становила 0.461, покращення є помірним (приблизно в 1.3 рази). Однак для найрідкіснішого класу 'H' (гіпертонія) з базовою лінією 0.030, досягнутий середній показник є більш ніж у 20 разів вищим, що вказує на потенційну здатність моделі виявляти навіть рідкісні патології значно краще за випадковий рівень [3]. Це демонструє, що просунута архітектура та стратегія навчання дозволили досягти значно кращого балансу між точністю та повнотою на нових даних.

4.6.3 AUC (ROC)

Для повноти аналізу також розглянемо графік метрики AUC (ROC) 4.6, яка вимірює площу під кривою робочих характеристик.

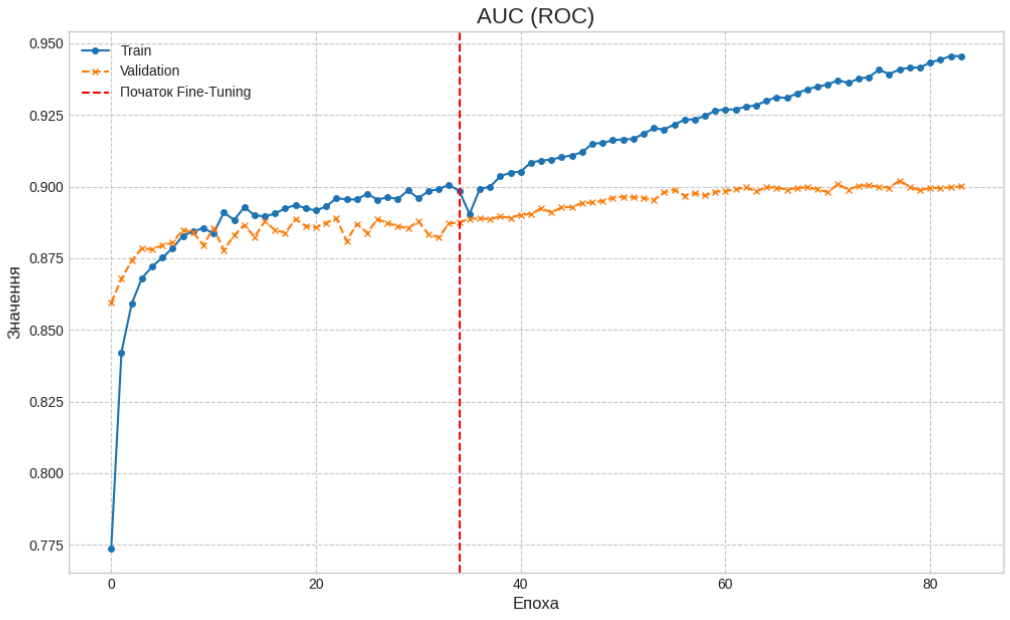


Рисунок 4.6 - Графік AUC (ROC) трансферної моделі

Як було теоретично обґрунтовано раніше, ця метрика може бути оманливою в умовах сильного дисбалансу класів, оскільки її розрахунок (співвідношення TPR до FPR) значною мірою залежить від кількості істинно-негативних випадків (True Negatives) [34], яких у медичних даних переважна більшість [33]. Даний графік є практичним підтвердженням цієї тези. На перший погляд, динаміка кривих AUC (ROC) дуже схожа на динаміку AUC (PR), однак ключова відмінність полягає в абсолютних значеннях. Модель досягає високих показників val_auc_roc, виходячи на плато в районі ~0.90. Це значення може створити хибне враження про надзвичайно високу якість моделі. Проте, як показав фінальний звіт по класах, модель не здатна розпізнавати рідкісні патології ('G', 'H') і має інші проблеми. Високий показник AUC (ROC) пояснюється нечутливістю метрики FPR (False Positive Rate) до збільшення кількості хибно-позитивних спрацьовувань в умовах сильного дисбалансу [35]. Розглянемо це на гіпотетичному прикладі для рідкісного класу, де є 10 позитивних прикладів (хворих) та 990 негативних (здорових).

Припустимо, модель робить 10 хибно-позитивних прогнозів (FP=10). В цьому випадку кількість істинно-негативних прогнозів (TN) буде

$$990 - 10 = 980.$$

Тоді

$$\text{FPR} = \frac{FP}{FP+TN} = \frac{10}{10+980} = 0.01.$$

Тепер припустимо, що модель почала працювати гірше і зробила вже 50 хибно-позитивних прогнозів (FP=50), що є п'ятикратним погіршенням. Кількість TN тепер

$$990 - 50 = 940.$$

Тоді

$$\text{FPR} = \frac{50}{50+940} \approx 0.05.$$

Як видно з прикладу, навіть п’ятикратне зростання кількості хибних тривог призводить лише до незначного збільшення FPR (з 0.01 до 0.05) через домінуючу кількість TN у знаменнику. Метрика AUC (ROC), яка будується на основі FPR, відповідно, також залишається оптимістично високою, маскуючи реальні проблеми моделі [35].

Таким чином, цей графік наочно демонструє, чому AUC (ROC) не був обраний як основна метрика для моніторингу процесу навчання. Хоча він і підтверджує загальну здатність моделі до навчання через що може слугувати додатковою метрикою, його високі значення не відображають реальних проблем з класифікацією рідкісних класів. На противагу цьому, метрика AUC (PR), знаменник якої (TP + FP) є чутливим до кількості хибних спрацювань, є значно більш надійним індикатором продуктивності.

4.6.4 Точність (Precision)

На зображенні 4.7 наведено графік точності (Precision) для трансферної моделі.

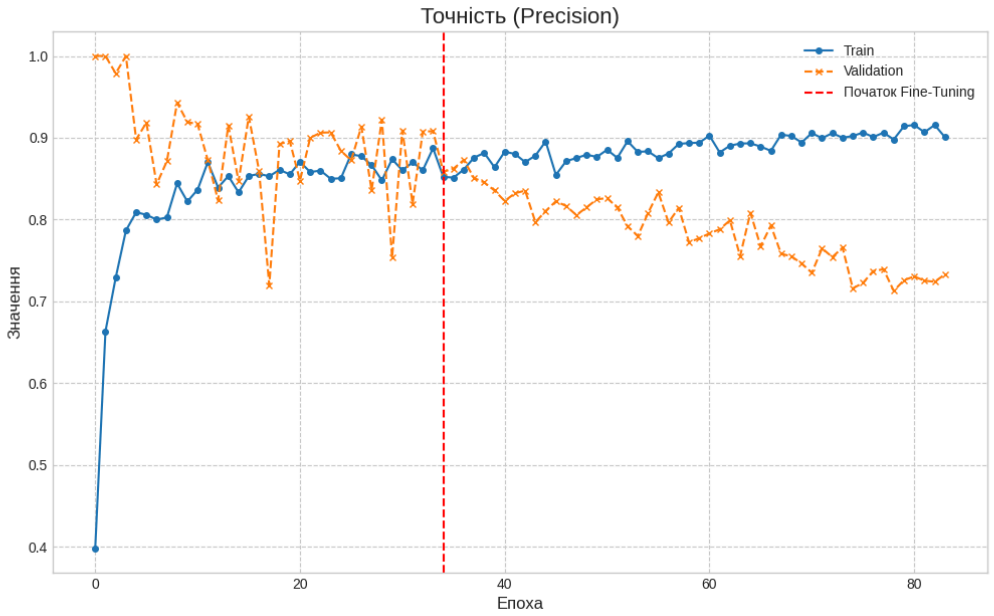


Рисунок 4.7 - Графік точності (Precision) трансферної моделі

На етапі вилучення ознак (епохи 0-36) спостерігається висока нестабільність валідаційної кривої (Validation precision). Вона починається з високих значень, а потім демонструє хаотичні коливання. Така поведінка є наслідком сильного дисбалансу класів, де навіть незначна зміна в прогнозах для рідкісних класів може призвести до різких стрибків усередненої метрики [35]. Тренувальна крива (Train precision) при цьому поводиться більш стабільно, поступово зростаючи до рівня ~0.88.

Після переходу до етапу тонкої настройки динаміка змінюється. Тренувальна крива продовжує повільно зростати, наближаючись до значення 0.9, що свідчить про підвищення впевненості моделі на знайомих даних. На противагу цьому, валідаційна крива починає демонструвати стабільну тенденцію до зниження, опускаючись з ~0.88 до ~0.73. Це явище є класичним проявом

компромісу між точністю та повнотою. У процесі донавчання модель намагається ідентифікувати більше позитивних випадків (що підвищує повноту, Recall), однак це неминуче призводить до збільшення кількості хибно-позитивних спрацьовувань (FP), що, в свою чергу, знижує точність [34]. Зростаючий розрив між тренувальною та валідаційною кривими на цьому етапі також є ознакою поступового перенавчання моделі [6].

4.6.5 Повнота (Recall)

На зображенні 4.8 наведено графік повноти (Recall) для трансферної моделі.

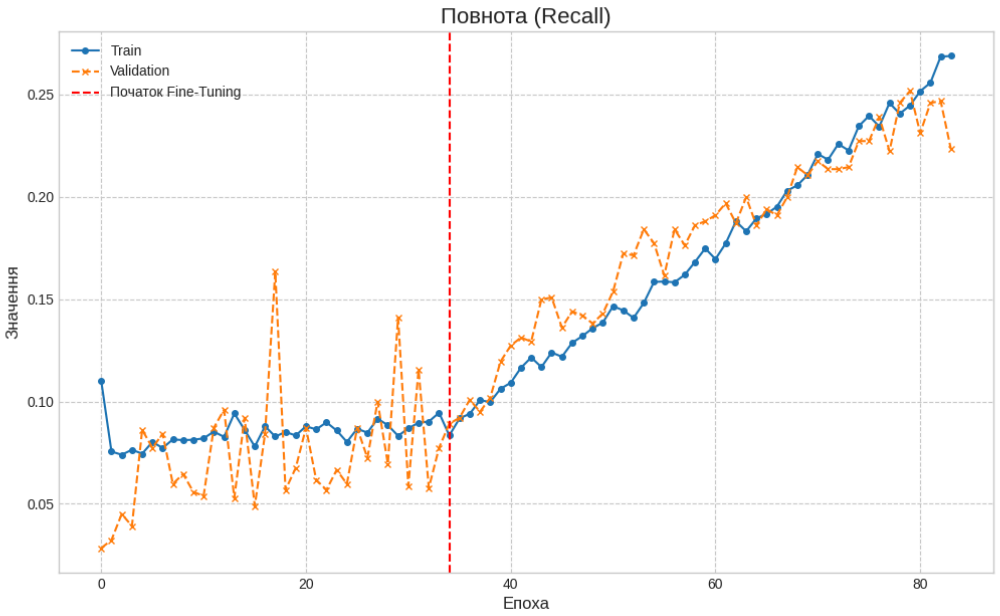


Рисунок 4.8 - Графік повноти (Recall) трансферної моделі

Динаміка повноти також чітко відображає два етапи навчання. На етапі вилучення ознак (епохи 0-36) крива Validation recall демонструє високу волатильність з різкими піками, але при цьому загальна тенденція залишається на низькому рівні, близько 0.05-0.1. Це вказує на те, що на початковому етапі модель

робить мало позитивних прогнозів, через що її здатність виявляти всі наявні патології є низькою та нестабільною.

Поведінка радикально змінюється після переходу до етапу тонкої настройки (з 36-ї епохи). Обидві криві, тренувальна та валідаційна, починають демонструвати впевнене і майже синхронне зростання. Це є прямим підтвердженням того, що в процесі донавчання модель починає краще розпізнавати ознаки захворювань на нових даних. Зростання Recall на цьому етапі пояснює одночасне падіння Precision, що було відзначено на попередньому графіку, адже модель знаходить більше реальних випадків, але ціною збільшення кількості хибних спрацьовувань [34].

Важливо відзначити, що на момент зупинки навчання (84-а епоха) криві повноти все ще мали виражену тенденцію до зростання і не вийшли на плато. Це свідчить про те, що потенціал моделі щодо знаходження патологій ще не був повністю вичерпаний. Однак, оскільки загальний баланс між точністю та повнотою, що відображається метрикою val_auc, перестав покращуватися, механізм ранньої зупинки коректно завершив навчання, зафіксувавши модель в точці оптимального компромісу [6]. Фінальне значення val_recall на рівні ~0.25, хоч і значно краще за базову модель, все ще залишається невисоким в абсолютному вираженні, що підкреслює складність задачі. На перший погляд може здатися не інтуітивним той факкт, що навчання було зупинено, якщо крива val_recall все ще демонструвала тенденцію до зростання. Це пояснюється природою метрики val_auc, яка використовувалася для моніторингу. На відміну від val_recall, що є точковою метрикою, розрахованою при фіксованому порозі, val_auc є інтегральною оцінкою, що враховує баланс точності та повноти по всьому діапазону можливих порогів. Вихід val_auc на плато означає, що подальше збільшення повноти досягалося ціною настільки значного падіння точності, що загальна якість моделі (площа під PR-кривою) перестала покращуватися [35]. Таким чином, механізм ранньої зупинки коректно зафіксував модель в точці її оптимального балансу між цими двома конкуруючими метриками.

4.6.6 F1-Score

З графіку 4.9 видно, що динаміка F1-Score значною мірою повторює динаміку метрики Recall, розглянутої раніше.

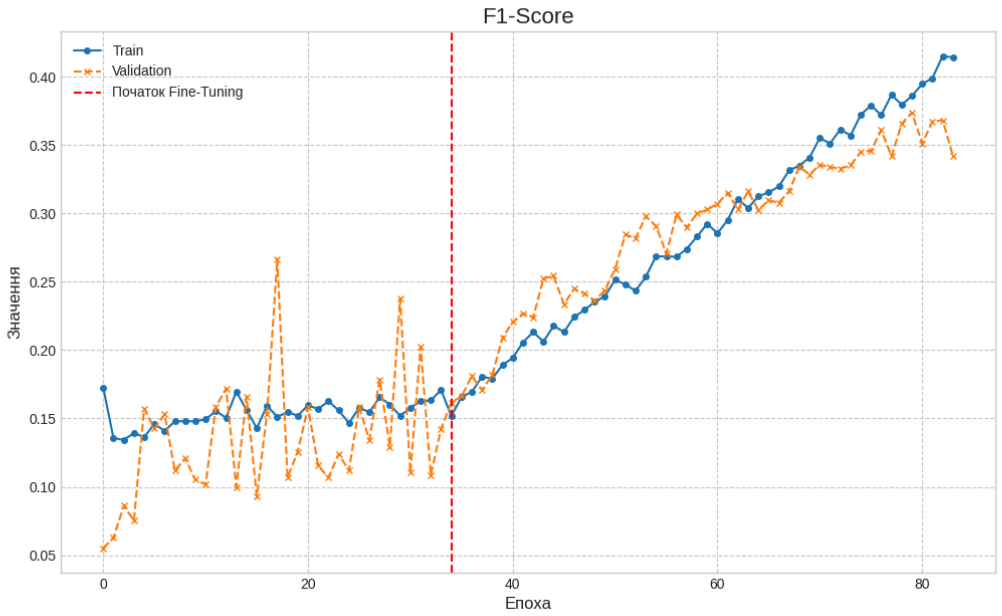


Рисунок 4.9 - Графік F1-Score трансферної моделі

Це не є випадковістю, а прямим наслідком математичної природи F1-score як гармонійного середнього, яке завжди тяжіє до меншого зі своїх компонентів [35]. Оскільки протягом усього навчання показник повноти (Recall) був значно нижчим за показник точності (Precision), саме він виступав обмежуючим фактором для загального F1-score.

На етапі вилучення ознак (епохи 0-35) графік val_f1_score успадковує високу волатильність від метрики Recall, залишаючись на стабільно низькому рівні. Після переходу до тонкої настройки (з 36-ї епохи) крива val_f1_score починає впевнено зростати, слідуючи за трендом val_recall. Це підтверджує, що загальний баланс метрик покращувався переважно за рахунок того, що модель навчалася знаходити більше реальних випадків патологій [25].

Фінальне значення val_f1_score на рівні ~0.35 є незадовільним в абсолютному вираженні, проте воно майже вдвічі перевищує пікове значення базової моделі (~0.20). Це ще раз підкреслює, що, незважаючи на збереження сильного перенавчання [6], трансферна модель демонструє значно кращий, хоч і все ще недостатній для практичного застосування, баланс між точністю та повнотою.

4.6.7 Гістограми розподілу впевненості по класах

У порівнянні з базовою моделлю, де розподіли були майже повністю змішані або дисперсні, трансферна модель демонструє кращу розділову здатність (рис. 4.10) [25].

Рисунок 4.10 - Гістограми розподілу впевненості по класах трансферної моделі

Для більшості класів спостерігається зміщення центрів мас червоних гістограм (позитивні випадки) вправо, в зону вищих ймовірностей, тоді як сині гістограми (негативні випадки) переважно концентруються біля низьких значень. Це свідчить про те, що модель стала більш впевненою у своїх прогнозах.

Особливо показовим є аналіз цих гістограм у контексті оптимальних порогів, знайдених для максимізації метрики F-beta [35]. Для частих класів ('N', 'D', 'O'), де розподіли все ще значно перекриваються, оптимальні пороги (для 'N' - 0.2648, для 'D' - 0.2005, для 'O' - 0.2282) розташовані таким чином, щоб захопити основну масу позитивних випадків, яка знаходиться в зоні відносно низьких ймовірностей. Такий вибір порогу дозволяє досягти високої повноти (Recall) ціною зниження точності, що є прямим наслідком пріоритету, заданого параметром $\beta=1.5$ [1].

Для рідкісних класів 'G' (Глаукома) та 'H' (Гіпертонія) гістограми візуально пояснюють нульові показники у фінальному звіті. Розподіли позитивних випадків для цих класів є дуже розрідженими, майже не відрізняючись від базової моделі. Навіть низькі оптимальні пороги (для 'G' - 0.2321, для 'H' - 0.2214) не змогли вірно розділити гістограми для цих прогнозів, що свідчить про повну нездатність моделі вивчити їхні ознаки [33].

На противагу цьому, для класів 'C' (Катаракта) та 'M' (Міопія) видно, що червоні гістограми змістилися до зон високих ймовірностей, але відмінність від базової моделі не значна. Це вказує на те, що модель навчилася розпізнавати деякі специфічні ознаки цих захворювань з високою впевненістю краще ніж базова модель [16]. Завдяки цьому, низькі оптимальні пороги (для 'C' - 0.1287, для 'M' - 0.1450) дозволяють ефективно відокремити майже всі позитивні випадки, що і призвело до досягнення 100% повноти (Recall=1.00) для цих класів у фінальному звіті.

4.6.8 Залежність метрик від порогу бінаризації

Графік 4.11 наочно демонструє класичний компроміс між точністю та повнотою.

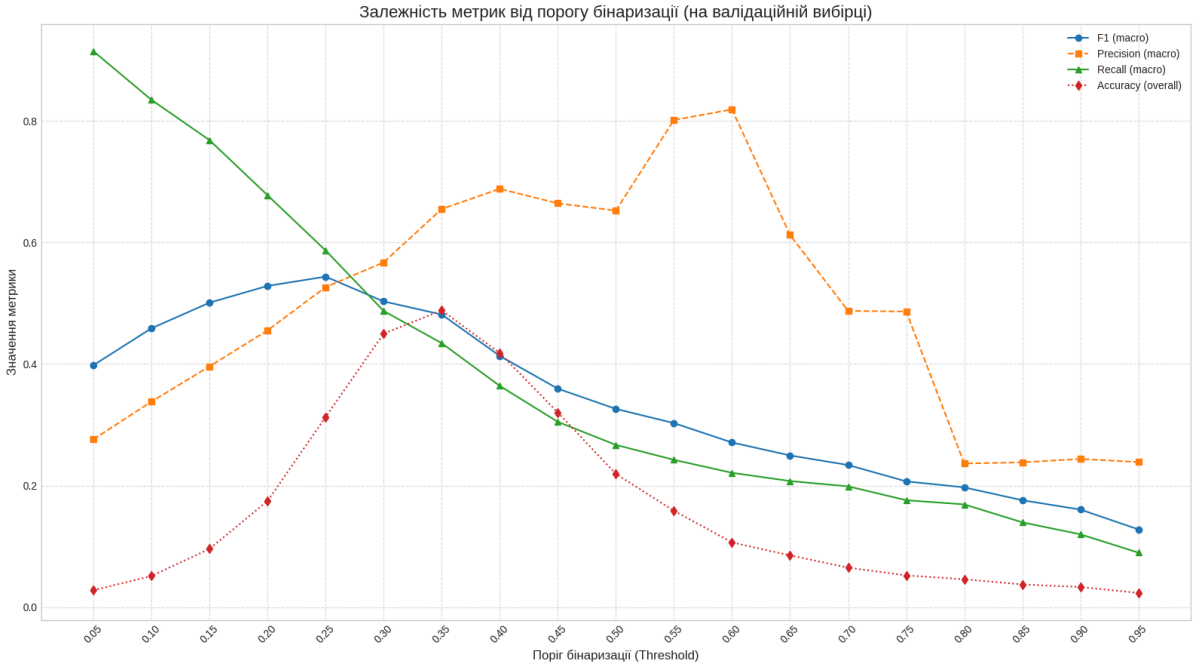


Рисунок 4.11 - Залежність метрик від порогу бінаризації трансферної моделі

Крива Recall (зелена лінія) очікувано має найвище значення при найнижчому порозі і монотонно спадає, тоді як крива Precision (помаранчева лінія) демонструє загальну тенденцію до зростання.

Ключовим індикатором на цьому графіку є крива F1-score (синя), яка відображає гармонійний баланс між точністю та повнотою. Вона досягає свого максимального значення ~0.54 в діапазоні порогів від 0.20 до 0.25. Це і є зона оптимального компромісу для даної моделі [35]. Важливо відзначити, що цей показник є значно вищим за пікове значення F1-score базової моделі (~0.37), що кількісно підтверджує перевагу трансферної архітектури. Той факт, що оптимальні пороги, знайдені раніше для кожного класу окремо, також переважно знаходяться в цьому діапазоні, підтверджує коректність обраного підходу до їх пошуку [5].

Крива загальної точності Accuracy (червона) досягає свого піку при значеннях ~0.35 та її значення дещо нижчі, що ще раз підкреслює оманливість цієї

метрики для задач з сильним дисбалансом класів [33]. Таким чином, даний графік підтверджує, що головна проблема моделі полягає не стільки у виборі порогу, скільки в її недостатній предиктивній силі, хоча ця сила і є значно вищою, ніж у базової архітектури.

Окремої уваги заслуговує аномальна поведінка кривої Precision (macro) після порогу 0.60, де вона демонструє різке падіння. Це явище пояснюється впливом рідкісних класів на усереднену метрику. При високих значеннях порогу модель перестає робити правильні позитивні прогнози для деяких рідкісних патологій, але все ще робить поодинокі хибні спрацювання. В результаті, точність для цих класів падає, що різко знижує загальний усереднений показник. Цей ефект є ще одним візуальним підтвердженням труднощів, з якими стикається модель при роботі з класами, що мають недостатню кількість прикладів у вибірці.

4.6.9 PR-криві для кожного класу

Аналіз графіків 4.12 демонструє значне, хоч і неоднорідне, покращення продуктивності для переважної більшості класів.

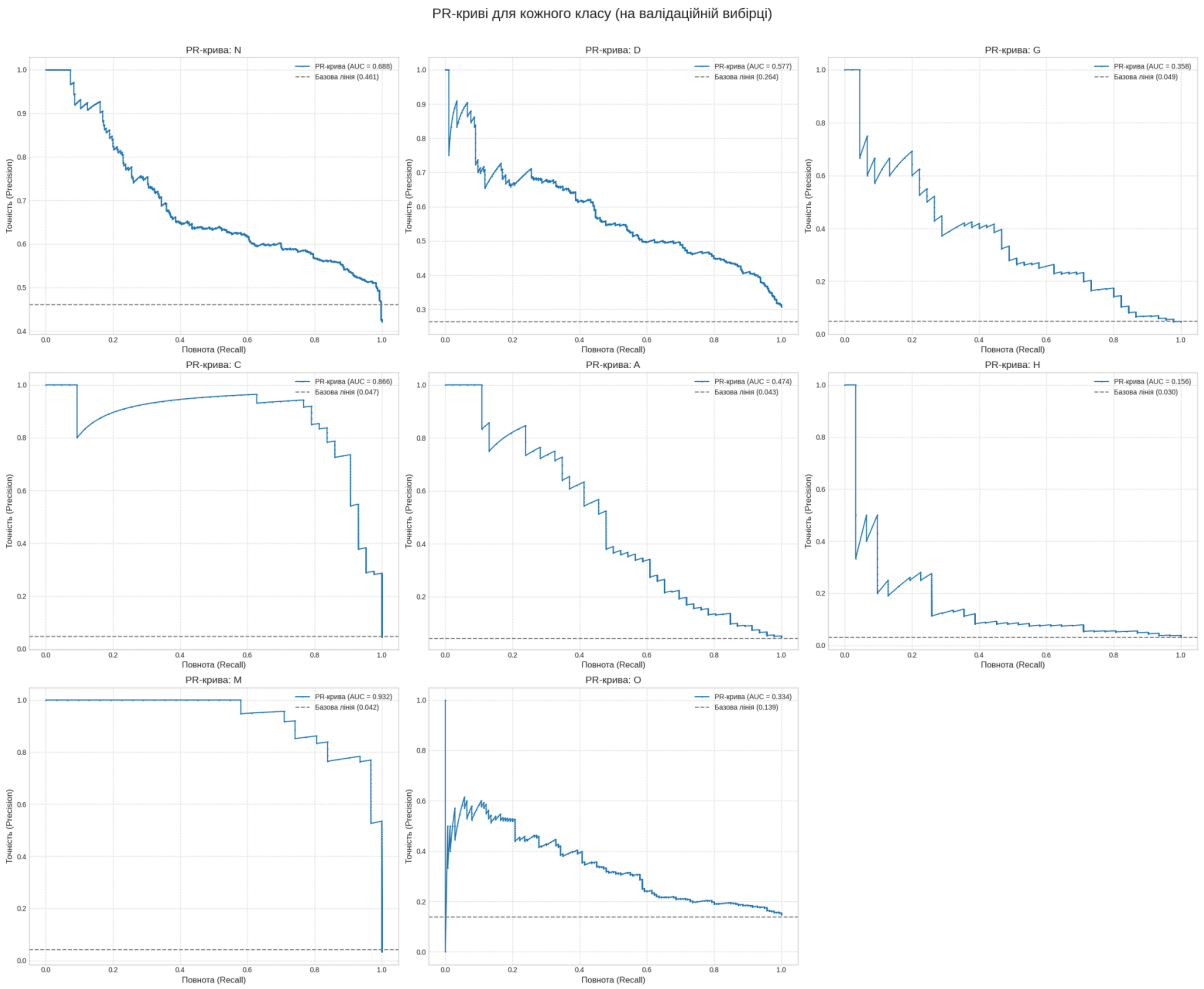


Рисунок 4.12 - PR-криві для кожного класу трансферної моделі

Найбільший прогрес спостерігається для класу 'A', де показник AUC зріс більш ніж у 5.6 рази (з 0.084 до 0.474). Це свідчить про те, що трансферна модель змогла вивчити візуальні патерни, які були абсолютно невидимими для базової архітектури [25]. Стабільно високі результати були покращені для класів 'M' та 'C'. Їхні показники AUC зросли відповідно з 0.728 до 0.932 (в ~1.3 рази) та з 0.644 до 0.866 (в ~1.3 рази), наблизившись до дуже високих значень [3]. Це вказує на те, що модель не лише зберегла, але й покращила здатність розпізнавати патології з чіткими візуальними ознаками. Помірне, але важливе зростання спостерігається

для найчастіших класів. Для класу 'N' AUC зріс з 0.498 до 0.688 (в ~1.4 рази), для 'D' з 0.407 до 0.577 (в ~1.4 рази), а для 'O' з 0.168 до 0.334 (в ~2 рази). Навіть для рідкісного класу 'G' (Глаукома) показник AUC зріс з 0.256 до 0.358 (в ~1.4 рази). Це демонструє загальну та системну перевагу просунутої архітектури над базовою.

Єдиний випадок деградації зафіксовано для найрідкіснішого класу 'H', де показник AUC знизився приблизно на 20% (з 0.195 до 0.156). Це може свідчити про те, що процес тонкої настройки, оптимізуючи ознаки для більш представлених класів, міг негативно вплинути на здатність моделі розпізнавати слабкі та рідкісні патерни, що належать до класу 'H' [33].

Таким чином, детальний аналіз PR-кривих підтверджує загальну ефективність обраного підходу. Трансферна модель демонструє значне або помірне покращення для 7 з 8 класів, хоча проблема ефективної класифікації найрідкісніших патологій залишається невирішеною.

4.6.10 Матриці помилок (Confusion Matrices)

Зображення 4.13 демонструє матриці помилок (Confusion Matrices) для трансферної моделі.

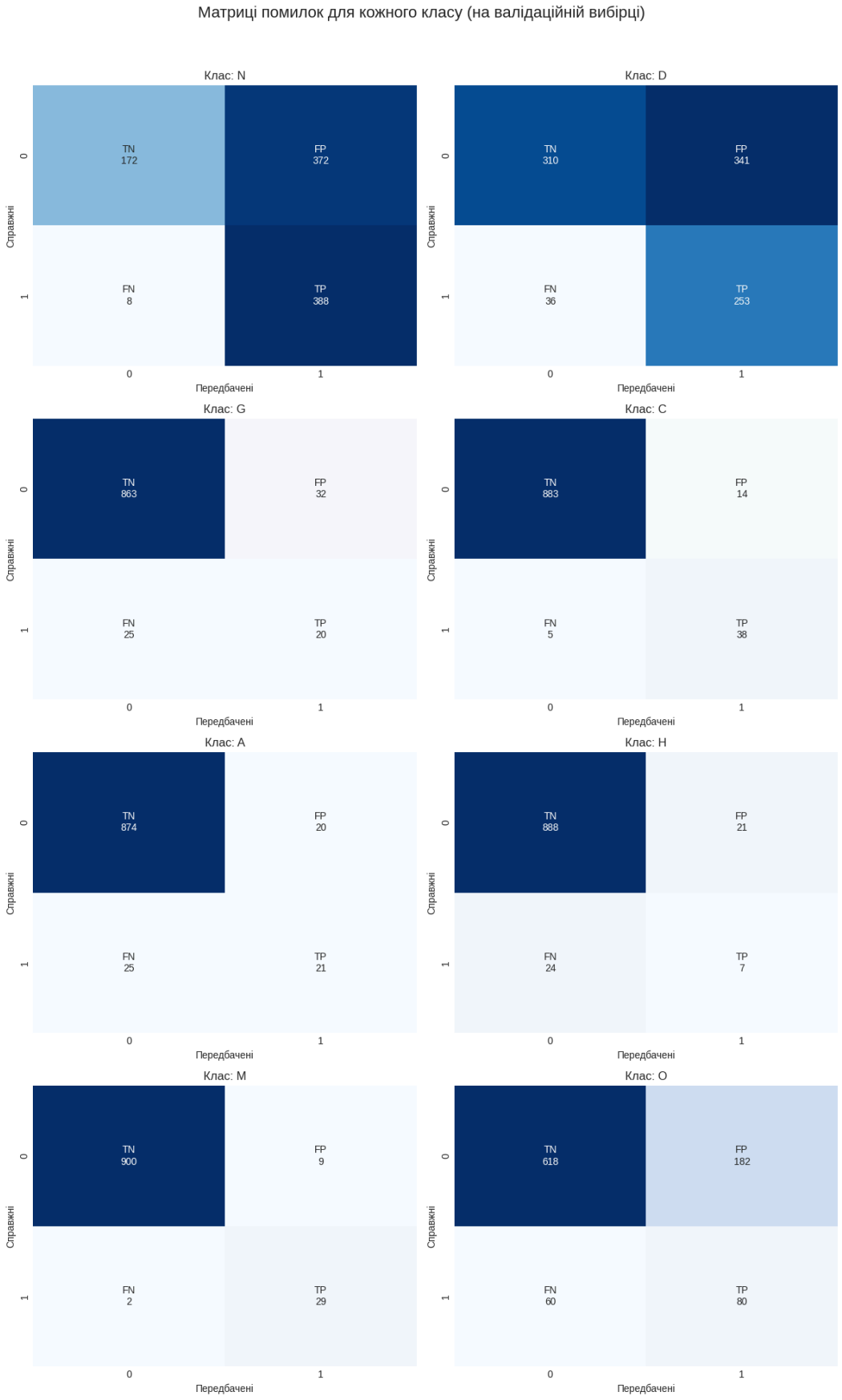


Рисунок 4.13 - Матриці помилок (Confusion Matrices) трансферної моделі

Як видно з матриці, для класу N просунута модель знайшла 388 правильних випадків (TP), пропустивши лише 8 (FN), тоді як базова модель знайшла 53, пропустивши 343. Це семиразове збільшення кількості правильно визначених здорових очей та радикальне скорочення (в 42 рази) кількості помилково пропущених. Для класу D продуктивність зросла в 9 разів, адже кількість TP збільшилася з 28 до 253, а кількість FN скоротилася з 261 до 36. Для класу C спостерігається стабільне покращення, адже кількість TP зросла з 33 до 38, а FN зменшилася вдвічі (з 10 до 5). Для класу M прогрес також очевидний, адже TP зросли з 22 до 29, а FN скоротилися більш ніж в 4 рази (з 9 до 2). Для класу O трансферна модель продемонструвала значне покращення, збільшивши кількість TP з 3 до 80, хоча і ціною зростання хибно-позитивних спрацьовувань (FP) з 4 до 182. Для класу 'G' (Глаукома) відбувся помірний прогрес, кількість TP зросла з 15 до 20, акількість FN впала з 30 до 25. Цікаво, що для класу 'H' показники залишилися ідентичними (TP=7, FN=24), що підтверджує нездатність обох моделей вивчити ознаки цієї патології [33].

Загалом, порівняльний аналіз матриць помилок наочно демонструє переваги просунутої моделі. Вона значно ефективніше знаходить випадки захворювань для більшості класів (зменшення FN) і є набагато більш придатною для практичного використання, ніж базова модель, яка пропускала переважну більшість патологій [1].

4.7 Аналіз фізичних та продуктивних характеристик

Зображення 4.14 демонструє фізичні та продуктивні характеристики для трансферної моделі.

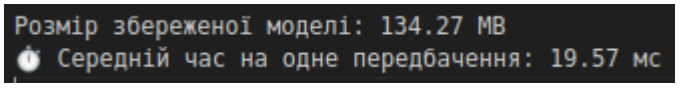


Рисунок 4.14 - Результат роботи коду аналізу фізичних та продуктивних характеристик трансферної моделі

Розмір файлу найкращої версії просунутої моделі склав 134.27 МБ. На перший погляд, цей результат є контрінтуїтивним, оскільки він на ~30% менший за розмір базової моделі (193.14 МБ), незважаючи на те, що архітектура ResNet50 є значно глибшою і містить більше загальних параметрів (~24 млн проти ~17 млн). Ця різниця пояснюється не кількістю ваг самої моделі, а станом оптимізатора, який зберігається разом з ваговими коефіцієнтами у файлі .keras. Оптимізатор Adam (і його варіація AdamW) для кожного параметра, що навчається, зберігає два додаткові значення, перший та другий моменти (moments), що фактично потроює об'єм пам'яті, необхідний для зберігання стану моделі під час навчання [31]. У базовій моделі всі ~16.9 мільйонів параметрів були тренованими (trainable). Таким чином, розмір файлу складався з ваг (64.36 МБ) та стану оптимізатора для всіх цих параметрів (128.72 МБ), що в сумі дає 193.08 МБ. У трансферній моделі на етапі тонкої настройки, коли була досягнута найкраща якість, більшість шарів ResNet50 були заморожені. Навчалися лише параметри голови та верхніх згорткових блоків, що складало значно меншу кількість тренованих параметрів (Trainable params: 5,653,064 (21.56 МБ)). Відповідно, стан оптимізатора зберігався лише тренованої частини ваг (43.12МБ), що і призвело до значно меншого фінального розміру файлу. Враховуючи що загальна кількість параметрів та їх вага дорівнює Total params = 23,719,368 (90.48 МБ), згідно виводу логу, то не складно підрахувати, що

$$(21.56 \cdot 2) + 90.48 = 133.6 \text{МБ};$$

(різниця в межах похибки через можливі накладні збереження інших процесів).

Цей результат демонструє важливу перевагу трансферного навчання, адже воно не лише покращує якість, але й може призводити до створення більш компактних артефактів моделі за рахунок зменшення кількості параметрів, що потребують тренування з нуля [25].

Середній час, необхідний для виконання одного передбачення на тестовому обладнанні, склав 19.57 мс. Цей показник є дещо вищим, ніж у базової моделі

(12.69 мс), що є очікуваним результатом. Збільшення часу обчислень приблизно на 54% пояснюється значно більшою глибиною та обчислювальною складністю архітектури ResNet50 порівняно з простою тришаровою згортковою мережею [19]. Незважаючи на це збільшення, продуктивність моделі залишається на дуже високому рівні. Час у 19.57 мс еквівалентний здатності обробляти приблизно 51 кадр на секунду ($\frac{1000 \text{ мс}}{19.57 \text{ мс}}$). Така швидкість є більш ніж достатньою для переважної більшості практичних застосувань, включаючи системи аналізу зображень в реальному часі [1]. Таким чином, досягнуте значне покращення якості класифікації було отримано ціною лише незначного і прийнятного зменшення швидкодії.

4.8 Візуалізація роботи моделі (Grad-CAM)

Були згенеровані матриці карт уваги, що показують, які області зображення були найбільш впливовими для активації нейронів на різних рівнях глибини мережі від ранніх згорткових шарів (conv2_block3_out) до глибоких (conv5_block3_out) [36].

Варто відзначити, що виникло технічне обмеження застосованого методу візуалізації, адже Grad-CAM генерувався для даних, що проходять через повний конвеєр обробки моделі, включаючи блок аугментації, через що теплові карти можуть мати невеликі геометричні зміщення відносно оригінального зображення ($\pm$ 10%) [24]. Однак, через відносно невеликі значення випадкових аугментацій, це не впливає на загальну якісну інтерпретацію областей, на які модель звертає увагу.

4.8.1 Пацієнт ID 1

На зображенні 4.15 представлено результати візуалізації Grad-CAM для пацієнта ID 1.

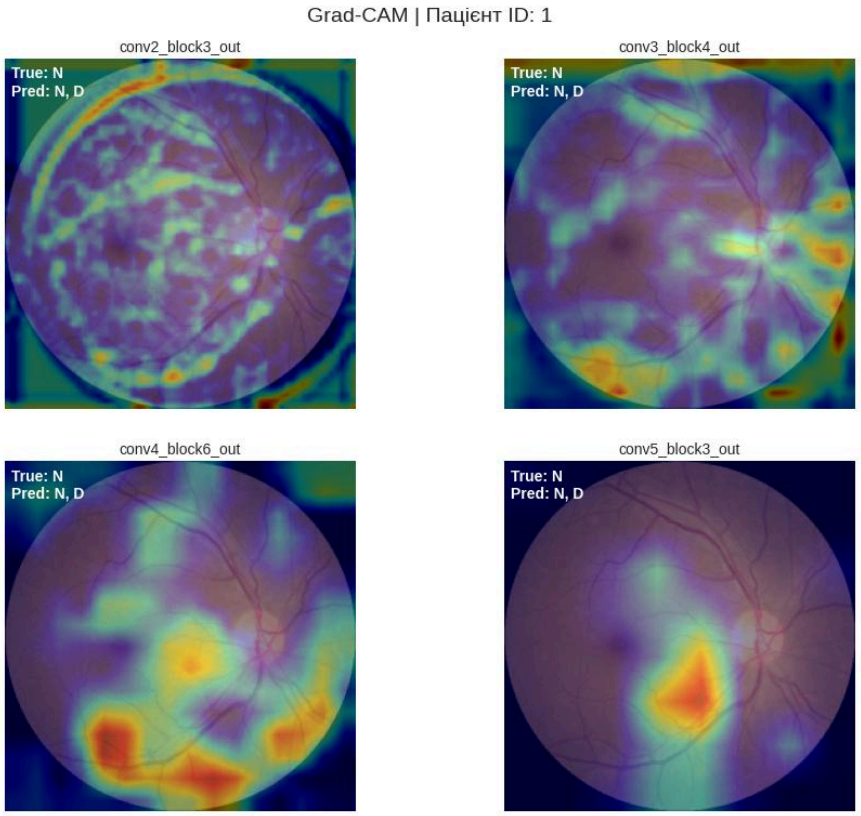


Рисунок 4.15 - Візуалізація Grad-CAM матриці по шарам різної глибини для трансферної для пацієнта ID 1

Перший розглянутий випадок демонструє схильність до навчання на нерелевантних ознаках та шаблонну поведінку при прогнозуванні. Модель частково правильно визначила, що око є здоровим (True: 'N'), але помилково додала до прогнозу клас 'D'.

На ранніх згорткових шарах (conv2_block3_out) увага моделі охоплює майже все зображення, з особливим акцентом на контрастних краях, зокрема на межі самого зображення та чорного фону. Це типова поведінка для ранніх шарів, що навчаються розпізнавати базові контури. На середніх шарах (conv3_block4_out, conv4_block6_out) фокус починає зміщуватися. Увага з чорних країв поступово

зникає, натомість з'являються зони активації в нижній частині сітківки та в області між диском зорового нерва та макулою. На найглибшому шарі (conv5_block3_out), який є найбільш впливовим для фінального рішення, увага моделі чітко концентрується на одній області трохи нижче центральної точки між макулою та диском.

Цей випадок демонструє що модель не сфокусувалася на явних патологічних змінах (яких немає), що дозволило їй правильно включити клас 'N' у прогноз, але фінальна точка фокусування не є очевидною ознакою діабету, що вказує на те, що модель, можливо, навчилася асоціювати певні неспецифічні патерни освітлення або текстури з класом 'D'. Помилкове додавання діагнозу до норми є також проявом шаблонної поведінки, виявленої раніше оскільки 'N' та 'D' є двома найчастішими класами в наборі даних, модель схильна прогнозувати їх разом, щоб статистично збільшити ймовірність правильної відповіді (підвищити recall), навіть якщо для цього немає чітких візуальних підстав.

4.8.2 Пацієнт ID 2

На зображенні 4.16 представлено результати візуалізації Grad-CAM для пацієнта ID 2.

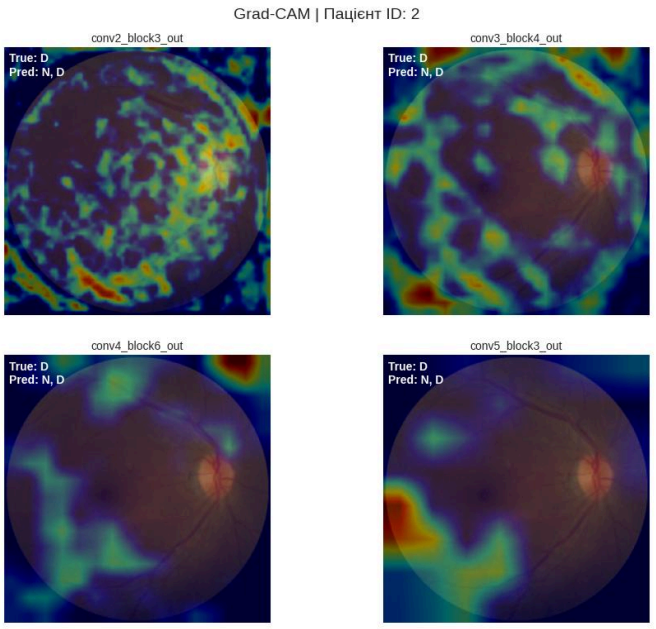


Рисунок 4.16 - Візуалізація Grad-CAM матриці по шарам різної глибини для трансферної для пацієнта ID 2

Цей випадок ілюструє більш складну поведінку моделі, де початкове розпізнавання релевантних ознак згодом нівелюється фокусуванням на менш інформативних ділянках. Модель правильно ідентифікувала наявність діабету ('D'), але, як і в попередньому прикладі, помилково додала клас ('N'), знову вдаючись до шаблонного прогнозу.

На ранніх згорткових шарах (conv2_block3_out, conv3_block4_out) модель демонструє перспективну поведінку. Окрім очікуваної уваги до країв, активації хаотично, але чітко розподілені вздовж області основних судин, де візуально спостерігаються ознаки діабетичної ретинопатії. Це свідчить про те, що низькорівневі фільтри моделі успішно виявили аномальні текстури, пов'язані з патологією [3]. На середньому шарі (conv4_block6_out) починається зміщення фокусу. Увага з центральної області судин та диска зміщується до лівої нижньої частини зображення, огинаючи макулу. При цьому з’являється значна зона активації в правому верхньому куті, що знаходиться за межами сітківки, вказуючи на реакцію на артефакт або шум. На найглибшому шарі (conv5_block3_out), що визначає фінальний прогноз, увага остаточно концентрується на лівій нижній частині зображення, частково захоплюючи і чорну область за його межами.

Цей випадок демонструє, що модель здатна на ранніх етапах виявляти релевантні патологічні ознаки. Однак, по мірі заглиблення в архітектуру, більш абстрактні фільтри не змогли коректно інтерпретувати цю інформацію. Замість того, щоб посилити фокус на судинах, модель змістила свою увагу на менш інформативну область, що, ймовірно, містила певний більш зручний для неї патерн. Це призвело до правильного, але невпевненого прогнозу, який модель додатково “посилила”, додавши до нього найчастіший клас 'N'.

4.8.3 Пацієнт ID 6

На зображенні 4.17 представлено результати візуалізації Grad-CAM для пацієнта ID 6.

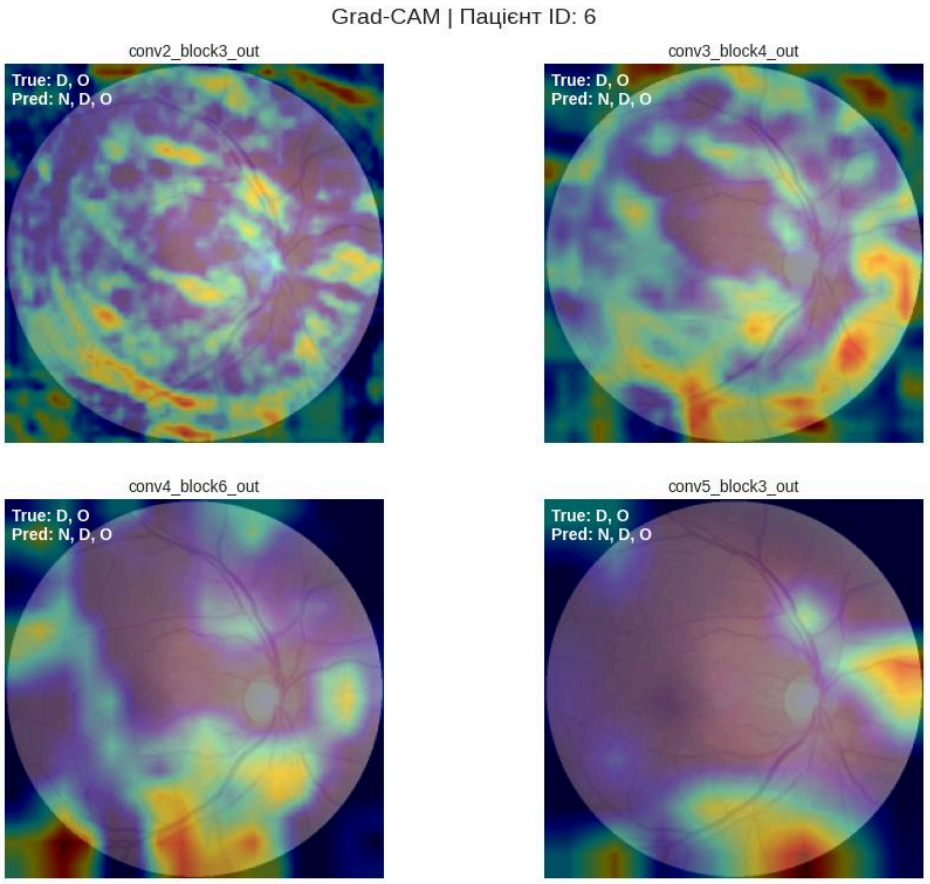


Рисунок 4.17 - Візуалізація Grad-CAM матриці по шарам різної глибини для трансферної для пацієнта ID 6

Цей приклад демонструє, як продуктивність моделі знижується при роботі зі складними випадками, що містять більше двох патологій одночасно. Модель правильно ідентифікувала наявність класів 'D' та 'O', але знову додала до прогнозу шумовий клас 'N'. Аналіз карт уваги показує поступову втрату фокусу. На ранніх згорткових шарах (conv2_block3_out, conv3_block4_out) увага моделі є дифузною і розподілена по всьому зображенню у вигляді хаотичних пучків. На відміну від попередніх прикладів, тут відсутня чітка концентрація на краях або судинах, що може свідчити про те, що наявність кількох типів візуальних ознак одночасно

заплутала низькорівневі фільтри. На середньому шарі (conv4_block6_out) увага починає зміщуватися до нижньої частини зображення, однак значна її частина припадає на чорну область за межами сітківки, що вказує на реакцію на артефакти. На найглибшому шарі (conv5_block3_out) увага моделі остаточно розпадається на кілька окремих плям. Частина з них знаходиться на анатомічних структурах (невелика пляма на судинах зверху від диска), але більшість на периферії та навіть за межами ока.

Отже зіткнувшись зі складним мультилейбловим випадком, модель не змогла виділити єдиний домінантний патерн. Її увага розпорошилася по кількох дрібних, частково нерелевантних ділянках. Це призвело до невпевненого прогнозу, який, хоч і виявився частково правильним (завдяки виявленню 'D' та 'O'), був доповнений шаблонним класом 'N', що є найчастішими класами. Цей приклад демонструє, що здатність моделі до впевненого фокусування ознак знижується при збільшенні складності клінічної картини на одному зображенні.

4.8.4 Пацієнт ID 13

На зображенні 4.18 представлено результати візуалізації Grad-CAM для пацієнта ID 13.

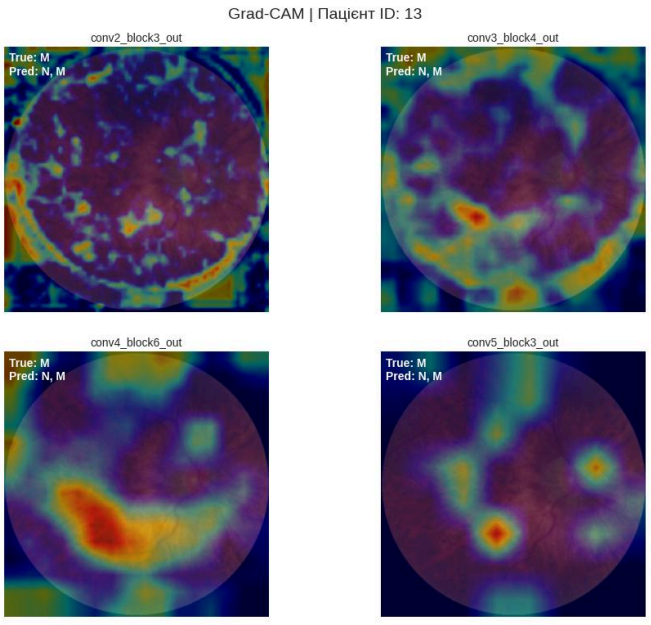


Рисунок 4.18 - Візуалізація Grad-CAM матриці по шарам різної глибини для трансферної для пацієнта ID 13

Цей приклад є демонстрацією значного прогресу трансферної моделі, а саме її здатності розпізнавати відносно рідкісний клас 'M' (патологічна міопія), який повністю ігнорувався базовою архітектурою. Модель правильно ідентифікувала патологію, хоча і додала до прогнозу клас 'N', що вказує на часткову невпевненість.

На ранніх згорткових шарах (conv2_block3_out, conv3_block4_out) увага моделі, окрім країв, концентрується у невеликих, але яскравих точках всередині зображення. Це може бути реакцією на початкові ознаки структурних аномалій, таких як лакові тріщини або зміни в судинах, характерні для міопії. На середньому шарі (conv4_block6_out) відбувається ключова зміна. Фокус моделі концентрується у велику, інтенсивну зону активації в лівій нижній частині сітківки. Ця область відповідає зоні перипапілярної атрофії та деформації диска зорового нерва, що є класичними ознаками патологічної міопії. На найглибшому шарі (conv5_block3_out) увага чітко локалізується на диску зорового нерва та на нижній перипапілярній області. Звивиста смуга меншої уваги, що йде вгору, ймовірно, слідує за аномально викривленими судинами.

На відміну від попередніх випадків, де модель реагувала на текстури або артефакти, тут вона продемонструвала здатність до розпізнавання складних структурних деформацій очного дна. Еволюція фокусу від дрібних точок до великої зони атрофії і, нарешті, до конкретних елементів (диск та судини) ілюструє успішний ієрархічний аналіз. Цей випадок доводить, що трансферна модель здатна вивчати нетривіальні та клінічно значущі патерни [16], що є значним кроком уперед у порівнянні з базовою архітектурою.

4.8.5 Пацієнт ID 8

На зображенні 4.19 представлено результати візуалізації Grad-CAM для пацієнта ID 8.

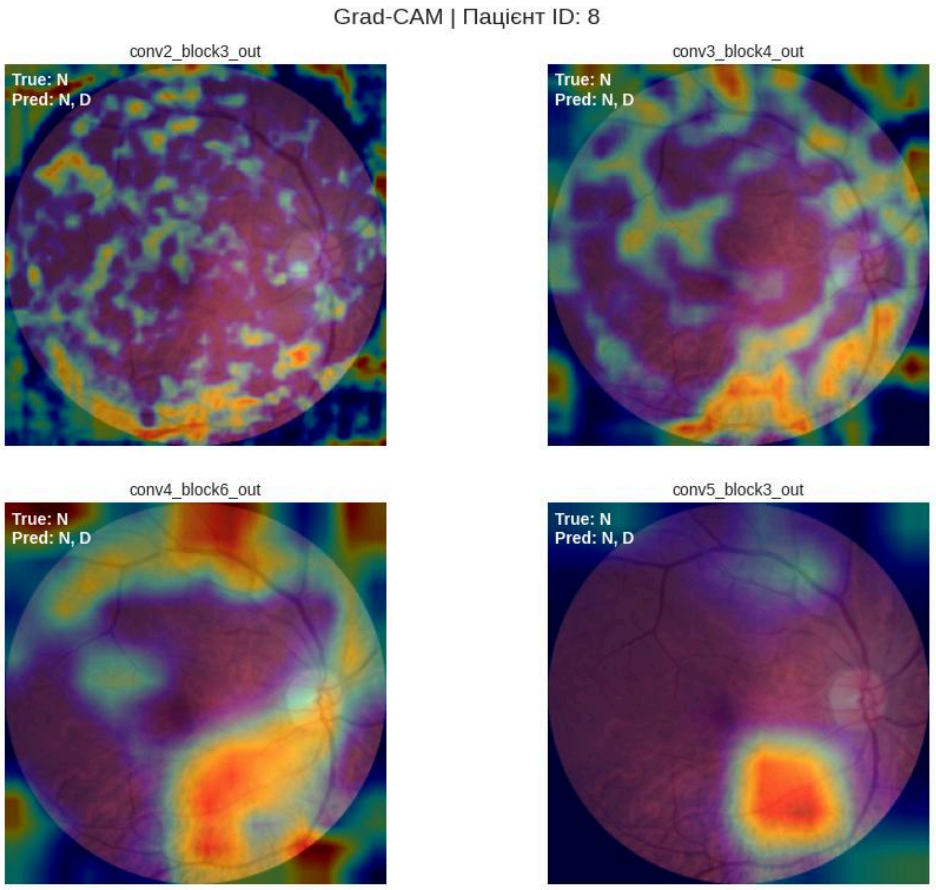


Рисунок 4.19 - Візуалізація Grad-CAM матриці по шарам різної глибини для трансферної для пацієнта ID 8

Останній розглянутий випадок є найбільш неоднозначним та показовим. Згідно з розміткою набору даних, дане зображення належить до класу 'N' (Норма). Однак модель спрогнозувала комбінацію 'N, D', помилково додавши клас “діабет”. При візуальному огляді оригінального зображення можна помітити ознаки, що не є типовими для здорового ока: наявність білувато-жовтих відкладень (подібних до твердих ексудатів) та аномальну товщину судин, що можуть вказувати на ознаки діабетичної ретинопатії.

На ранніх згорткових шарах (conv2_block3_out) увага моделі є дифузною, але вже помітні зони підвищеної активації в нижній частині зображення, що збігаються з областями найбільш виражених ниткоподібних структур. На середніх шарах (conv3_block4_out, conv4_block6_out) фокус уваги поступово звужується і концентрується на нижній правій частині сітківки, слідуючи за вигнутою дугою аномальних відкладень у напрямку до диска зорового нерва. На найглибшому шарі (conv5_block3_out) увага моделі чітко сконцентрована на великій зоні в нижній правій частині сітківки, там, де візуальні ознаки, схожі на патологію, є найбільш вираженими.

Таким чином, цей випадок демонструє, що модель впоралася з поставленою задачею, коректно ідентифікувавши клас “норма” ('N'). Аналіз карт уваги Grad-CAM підтверджує, що її фокус був сконцентрований на ділянках, які візуально містять неоднозначні ознаки, схожі на патологічні зміни. Помилкове додавання класу “діабет” ('D') у даному контексті є очікуваним проявом раніше описаної шаблонної поведінки моделі. Через сильну упередженість в бік найчастіших класів ('N' та 'D'), модель, зіткнувшись з візуально неоднозначним випадком, видала комбінацію найбільш статистично ймовірних прогнозів. На відміну від випадків, де модель реагувала на технічні артефакти, тут її поведінка була спровокована реальними особливостями зображення, що робить її помилку хоч і передбачуваною, але менш критичною.

4.9 Висновки по трансферній моделі

Для подолання фундаментальних обмежень базової архітектури, зокрема сильного перенавчання та не здатності до узагальнення, був застосований комплексний підхід. В якості основи була обрана архітектура ResNet50 з ваговими коефіцієнтами, попередньо навченими на наборі даних ImageNet [12, 19]. Була реалізована двоетапна стратегія навчання, що поєднувала вилучення ознак (Feature Extraction) та тонку настройку (Fine-Tuning). Для подальшої боротьби з перенавчанням та дисбалансом класів були інтегровані сучасні інструменти:

оптимізатор AdamW з механізмом decoupled weight decay, агресивна регуляризація за допомогою Dropout, функція активації GELU та модифікована функція втрат Focal Loss з додатковими штрафами [29, 32, 33]. Застосовані удосконалення призвели до значного покращення як процесу навчання, так і фінальних метрик продуктивності. Аналіз графіків навчання продемонстрував значно кращу стабільність та здатність до узагальнення, адже розходження між тренувальною та валідаційною кривими було зменшено, а ключова метрика val_auc (PR) досягла пікового значення ~0.61, що на 60% вище, ніж у базової моделі. На тестовій вибірці усереднений показник Macro Avg F1-score зріс більш ніж утричі (з 0.15 до 0.46). Модель продемонструвала високу швидкість роботи, із середнім часом передбачення 19.57 мс. Найбільшим успіхом є те, що модель навчилася впевнено ідентифікувати найбільш поширені та деякі рідкісні патології. Продуктивність для класів “діабет” (D), “міопія” (M) та “інше” (O) зросла на порядок, а F1-score для них досяг високих значень (0.73, 0.80 та 0.60 відповідно). Аналіз Grad-CAM підтвердив, що, на відміну від базової архітектури, просунута модель навчилася фокусуватися на клінічно релевантних анатомічних структурах, а не на технічних артефактах [36]. Водночас, головною невдачею залишається нездатність моделі розпізнавати найрідкісніші класи, такі як “глаукома” (G) та “гіпертонія” (H), де всі метрики залишилися на нульовому рівні. Це є очікуваним наслідком екстремального дисбалансу класів у вихідних даних, який не вдалося повністю компенсувати навіть за допомогою зважених функцій втрат. Також, модель схильна до передбачень у вигляді патернів що складаються з найчастіших класів, та часто передбачає більше класів ніж присутньо насправді на зображенні, при цьому часто в межах передбачень присутні і реальні класи. Враховуючи отримані результати, розроблену модель не можна використовувати як самостійний та повністю автоматичний інструмент для клінічної діагностики. Її нездатність виявляти такі критично важливі захворювання, як глаукома, та схильність до шаблонних прогнозів роблять її ненадійною для постановки остаточного діагнозу. Однак, модель демонструє значний потенціал як допоміжний інструмент підтримки прийняття рішень або система попереднього скринінгу [4]. З огляду на

високі показники повноти (Recall) для ключових патологій (наприклад, 1.0 для 'M' та 'A', 0.91 для 'D' при оптимальних порогах), її можна використовувати для первинного огляду великого потоку знімків з метою виявлення потенційно проблемних випадків, які потребують першочергової уваги лікаря. Модель демонструє високу ефективність у виявленні найпоширеніших патологій, що дозволяє використовувати її для первинного сортування, звужувати коло пошуку для спеціаліста, тим самим оптимізуючи його час. Використання моделі можливе лише за умови, що її прогнози розглядаються не як остаточний діагноз, а як рекомендація, що підлягає обов'язковій верифікації кваліфікованим офтальмологом [1].

ВИСНОВКИ

У роботі було досліджено та розроблено архітектуру глибинного навчання для мультилейблової класифікації офтальмологічних патологій на основі фундусних знімків ока, застосовано комплекс сучасних методологічних підходів для підвищення узагальнюючої здатності моделі в умовах сильного дисбалансу класів та обмеженої вибірки.

Проведено всебічний програмний аналіз та підготовку вихідного набору даних (ODIR-5K) [23, 38]. Детальне дослідження структури та оригінальних анотацій підтвердило непридатність наявних міток для мультилейблової класифікації окремого ока. На основі декомпозиції та мапінгу 94 унікальних текстових діагнозів лікарів було створено новий, очищений та валідований набір даних (final_dataset.csv), що містить 6338 зображень з коректними мультилейбловими векторами міток. Це забезпечило надійну основу для подальшого навчання моделі.

Розроблено та проаналізовано базову модель згорткової нейронної мережі (CNN) як відправну точку (baseline) [6]. Проста тришарова архітектура дозволила продемонструвати базові принципи роботи згорткових нейронних мереж та виявила ключові проблеми, а саме: сильне перенавчання (overfitting) через надлишкову ємність (16.9 млн параметрів) та низьку узагальнюючу здатність, що призвело до незадовільних фінальних метрик (Macro Avg F1-score ≈0.15 та PR-AUC ≈0.38).

Обґрунтовано, спроектовано та реалізовано просунуту модель на основі трансферного навчання на базі ResNet50 [19, 25]. Для подолання проблем базової моделі було застосовано попередньо навчену на ImageNet архітектуру ResNet50 [12]. Реалізовано двоетапну стратегію навчання (Feature Extraction та Fine-Tuning), що дозволило значно підвищити стабільність процесу навчання та збільшити ключову метрику val_auc (PR) на 60% (з ≈0.38 до ≈0.61).

Розроблено та впроваджено модифіковану гібридну функцію втрат (Focal Loss) [33]. Для компенсації сильного дисбалансу класів (~15:1) була застосована функція Focal Loss з параметрами γ=2.0 та α=0.25. Додатково впроваджено

штрафи за надмірну кількість прогнозованих класів та за нелогічні комбінації (класи “норма” та “патологія” (окрім класу “інше”) одночасно).

Проведено порівняльний та якісний аналіз продуктивності моделей. Фінальна оцінка на тестовій вибірці підтвердила трикратне зростання усередненої якості, де показник Macro Avg F1-score зріс з 0.15 до 0.46. Аналіз Grad-CAM показав, що, на відміну від базової моделі, трансферна архітектура навчилася фокусуватися на клінічно релевантних анатомічних структурах, що свідчить про успішне подолання схильності до навчання на технічних артефактах [36].

Щодо наукових та практичних результатів (новизна), то:

• Для обраного набору даних було проведено детальний програмний аналіз структури оригінальних анотацій, що дозволило виявити неявні правила маркування та обґрунтувати відмову від використання оригінальних міток на користь генерації нових, коректних мультилейблових векторів на основі текстових описів лікарів [23].

• Доведено, що трансферне навчання (ResNet50) дозволило моделі навчитися впевнено ідентифікувати найбільш поширені та деякі рідкісні патології, про що свідчить значне підвищення метрик: F1-score для Діабету (D) зріс з 0.07 до 0.73, для Міопії (M) зріс з 0.00 до 0.80, а для Вікової макулярної дегенерації (A) та Міопії (M) Recall досяг 1.0 [19].

• Продемонстровано, що модифікована функція втрат Focal Loss дозволила контролювати надмірні прогнози, але не змогла повністю компенсувати ефект екстремального дисбалансу для найрідкісніших класів, зокрема Глаукоми (G) та Гіпертонії (H), які залишилися на нульовому рівні ефективності [33].

Розроблена модель може бути використана як допоміжний інструмент підтримки прийняття рішень (CAD) або система попереднього скринінгу [4]. Завдяки високим показникам повноти (Recall $\geq$0.91) для ключових патологій (D, C, M), модель може ефективно виявляти потенційно проблемні випадки у великому потоці знімків, що потребують першочергової уваги лікаря тим самим оптимізуючи час спеціаліста.

Для підвищення надійності класифікації рідкісних класів (G, H) необхідні додаткові дослідження в напрямку компенсації екстремального дисбалансу. Це може включати застосування комбінованих методик аугментації та ускладнених функцій втрат, які динамічно адаптують вагові коефіцієнти в залежності від якості прогнозу [24]. Подальші дослідження мають включати в себе оптимізацію гіперпараметрів, пошук оптимальної архітектури під поставлені задачі, ансамблювання, навчання різних моделей на по різному оброблених зображеннях, усереднення передбачень у часі для роботи з відео тощо.

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Topol E. J. High-performance medicine: the convergence of human and artificial intelligence / E. J. Topol // Nature Medicine. – 2019. – Vol. 25, no. 1. – P. 44–56. – DOI: https://doi.org/10.1038/s41591-018-0300-7.
2. Deep EHR: A Survey of Recent Advances in Deep Learning Techniques for Electronic Health Record (EHR) Analysis / B. Shickel, P. J. Tighe, A. Bihorac, P. Rashidi // IEEE Journal of Biomedical and Health Informatics. – 2018. – Vol. 22, no. 5. – P. 1589–1604.
3. Dermatologist-level classification of skin cancer with deep neural networks / A. Esteva, B. Kuprel, R. A. Novoa [et al.] // Nature. – 2017. – Vol. 542, no. 7639. – P. 115–118.
4. Perova I. Adaptive Human Machine Interaction Approach for Feature Selection-Extraction Task in Medical Data Mining / I. Perova, Y. Bodyanskiy // International Journal of Computing. – 2018. – Vol. 17, no. 2. – P. 113–119.
5. Flach P. Machine Learning: The Art and Science of Algorithms that Make Sense of Data / P. Flach. – Cambridge : Cambridge University Press, 2012. – 409 p.
6. Goodfellow I. Deep Learning / I. Goodfellow, Y. Bengio, A. Courville. – Cambridge, MA : MIT Press, 2016. – 800 p.
7. Turing A. M. On computable numbers, with an application to the Entscheidungsproblem / A. M. Turing // Proceedings of the London Mathematical Society. – 1936. – Vol. s2-42, no. 1. – P. 230–265.
8. McCulloch W. S. A logical calculus of the ideas immanent in nervous activity / W. S. McCulloch, W. Pitts // The bulletin of mathematical biophysics. – 1943. – Vol. 5, no. 4. – P. 115–133.
9. Rosenblatt F. The perceptron: A probabilistic model for information storage and organization in the brain / F. Rosenblatt // Psychological Review. – 1958. – Vol. 65, no. 6. – P. 386–408.
10. LeCun Y. Backpropagation Applied to Handwritten Zip Code Recognition / Y. LeCun, B. Boser, J. S. Denker // Neural Computation. – 1989. – Vol. 1, no. 4. – P. 541–551.

11. Gradient-Based Learning Applied to Document Recognition / Y. LeCun, L. Bottou, Y. Bengio, P. Haffner // Proceedings of the IEEE. – 1998. – Vol. 86, no. 11. – P. 2278–2324.
12. Deng J. ImageNet: A large-scale hierarchical image database / J. Deng, W. Dong, R. Socher // 2009 IEEE Conference on Computer Vision and Pattern Recognition. – Miami, FL, USA : IEEE, 2009. – P. 248–255.
13. ImageNet Large Scale Visual Recognition Challenge / O. Russakovsky [et al.] // International Journal of Computer Vision. – 2015. – Vol. 115, no. 3. – P. 211–252.
14. Krizhevsky A. ImageNet Classification with Deep Convolutional Neural Networks / A. Krizhevsky, I. Sutskever, G. E. Hinton // Advances in Neural Information Processing Systems 25 (NIPS 2012). – Lake Tahoe, NV, USA, 2012. – P. 1097–1105.
15. Simonyan K. Very Deep Convolutional Networks for Large-Scale Image Recognition / K. Simonyan, A. Zisserman // 3rd International Conference on Learning Representations (ICLR 2015). – San Diego, CA, USA, 2015. – URL: https://arxiv.org/abs/1409.1556 (дата звернення: 20.08.2025).
16. Zeiler M. D. Visualizing and Understanding Convolutional Networks / M. D. Zeiler, R. Fergus // European Conference on Computer Vision (ECCV). – Zurich, Switzerland : Springer, 2014. – P. 818–833.
17. Lin M. Network In Network / M. Lin, Q. Chen, S. Yan // International Conference on Learning Representations (ICLR). – Banff, Canada, 2014. – URL: https://arxiv.org/abs/1312.4400 (дата звернення: 25.08.2025).
18. Going deeper with convolutions / C. Szegedy [et al.] // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – Boston, MA, USA, 2015. – P. 1–9.
19. Deep Residual Learning for Image Recognition / K. He, X. Zhang, S. Ren, J. Sun // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – Las Vegas, NV, USA, 2016. – P. 770–778.

20. Tan M. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks / M. Tan, Q. V. Le // Proceedings of the 36th International Conference on Machine Learning (ICML). – Long Beach, CA, USA, 2019. – P. 6105–6114.
21. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale / A. Dosovitskiy [et al.] // International Conference on Learning Representations (ICLR). – 2021. – URL: https://openreview.net/forum?id=YicbFdNTTy (дата звернення: 02.09.2025).
22. ODIR-2019 Grand Challenge : Ocular Disease Intelligent Recognition. – 2019. – URL: https://odir2019.grand-challenge.org/ (дата звернення: 10.09.2025).
23. ODIR-5K: Ocular Disease Intelligent Recognition // Kaggle. – 2019. – URL: https://www.kaggle.com/datasets/andrewmvd/ocular-disease-recognition-odir5k (дата звернення: 18.09.2025).
24. Yang S. Image Data Augmentation for Deep Learning: A Survey / S. Yang, W. Xiao, M. Zhang // arXiv preprint arXiv:2204.08610. – 2023. – URL: https://arxiv.org/abs/2204.08610 (дата звернення: 29.09.2025).
25. Bengio Y. Representation Learning: A Review and New Perspectives / Y. Bengio, A. Courville, P. Vincent // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2013. – Vol. 35, no. 8. – P. 1798–1828.
26. Glorot X. Understanding the difficulty of training deep feedforward neural networks / X. Glorot, Y. Bengio // Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS). – Chia Laguna Resort, Sardinia, Italy, 2010. – Vol. 9. – P. 249–256.
27. Nair V. Rectified Linear Units Improve Restricted Boltzmann Machines / V. Nair, G. E. Hinton // Proceedings of the 27th International Conference on Machine Learning (ICML-10). – Haifa, Israel, 2010. – P. 807–814.
28. Ioffe S. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift / S. Ioffe, C. Szegedy // Proceedings of the 32nd International Conference on Machine Learning (ICML). – Lille, France, 2015. – Vol. 37. – P. 448–456.

29. Dropout: A Simple Way to Prevent Neural Networks from Overfitting / N. Srivastava [et al.] // Journal of Machine Learning Research. – 2014. – Vol. 15, no. 1. – P. 1929–1958.
30. Regularization of Neural Networks using DropConnect / L. Wan [et al.] // Proceedings of the 30th International Conference on Machine Learning (ICML). – Atlanta, GA, USA, 2013. – Vol. 28, no. 3. – P. 1058–1066.
31. Kingma D. P. Adam: A Method for Stochastic Optimization / D. P. Kingma, J. Ba // 3rd International Conference on Learning Representations (ICLR 2015). – San Diego, CA, USA, 2015. – URL: https://arxiv.org/abs/1412.6980 (дата звернення: 14.10.2025).
32. Loshchilov I. Decoupled Weight Decay Regularization / I. Loshchilov, F. Hutter // 7th International Conference on Learning Representations (ICLR 2019). – New Orleans, LA, USA, 2019. – URL: https://openreview.net/forum?id=Bkg6RiCqY7 (дата звернення: 01.11.2025).
33. Focal Loss for Dense Object Detection / T.-Y. Lin, P. Goyal, R. Girshick, K. He, P. Dollár // Proceedings of the IEEE International Conference on Computer Vision (ICCV). – Venice, Italy, 2017. – P. 2980–2988.
34. Fawcett T. An introduction to ROC analysis / T. Fawcett // Pattern Recognition Letters. – 2006. – Vol. 27, no. 8. – P. 861–874.
35. Saito T. The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets / T. Saito, M. Rehmsmeier // PLoS ONE. – 2015. – Vol. 10, no. 3. – e0118432.
36. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization / R. R. Selvaraju [et al.] // Proceedings of the IEEE International Conference on Computer Vision (ICCV). – Venice, Italy, 2017. – P. 618–626.
37. Castelvecchi D. Can we open the black box of AI? / D. Castelvecchi // Nature. – 2016. – Vol. 538, no. 7623. – P. 20–23.
38. Шегера А. Ю. Застосування згорткової нейронної мережі для класифікації офтальмологічних захворювань за зображеннями очного дна / А. Ю. Шегера, О. М. Величко, О. М. Дацок // Вісник Національного технічного

університету «ХПІ». Серія: Інформатика та моделювання. – 2025. – № 2 (14). – С. 172–183. – DOI: 10.20998/2411-0558.2025.02.12.

ДОДАТОК А - Програмна реалізація аналізу вхідного набору даних

Data_tester.py > ...

```python
# Завантаження та початковий огляд даних
import pandas as pd
import numpy as np
import os

# Вказуємо повний шлях до файлу
file_path = os.path.expanduser('~/Study/Diploma/archive (1)/full_df.csv')

# Завантажуємо дані в DataFrame
try:
    df = pd.read_csv(file_path)
    print(f"Файл {file_path} успішно завантажено.")
    print("-" * 30)

    # Виводимо базову інформацію про DataFrame
    print("Базова інформація про структуру даних (df.info()):")
    df.info()

except FileNotFoundError:
    print(f"Помилка: файл {file_path} не знайдено. Перевірте шлях до файлу.")
```

Рисунок А.1 - Код базового огляду файлу

```python
# Аналіз демографії пацієнтів
import matplotlib.pyplot as plt

if 'df' in locals():
    # Створюємо DataFrame з унікальними пацієнтами, щоб уникнути подвійного підрахунку
    unique_patients_df = df.drop_duplicates(subset=['ID'])

    # Загальна кількість унікальних пацієнтів
    num_patients = unique_patients_df.shape[0]
    print(f"Загальна кількість унікальних пацієнтів: {num_patients}")
    print("-" * 30)

    # Візуалізація розподілу за статтю
    gender_percentage = unique_patients_df['Patient Sex'].value_counts(normalize=True) * 100
    plt.figure(figsize=(8, 6))
    plt.pie(gender_percentage, labels=gender_percentage.index, autopct='%1.1f%%', startangle=90, colors=['skyblue', 'lightcoral'])
    plt.title('Розподіл пацієнтів за статтю')
    plt.ylabel('') # Прибираємо текстову мітку 'Patient Sex' збоку

    # Зберігаємо графік у файл
    plt.savefig('gender_distribution.png')
    print("Графік 'gender_distribution.png' збережено.")
    plt.close() # Закриваємо, щоб не заважав наступному

    print("-" * 30)

    # Візуалізація розподілу за віком
    plt.figure(figsize=(12, 7))
    # Створюємо гістограму
    plt.hist(unique_patients_df['Patient Age'], bins=30, edgecolor='black', color='steelblue')
    plt.title('Розподіл пацієнтів за віком', fontsize=16)
    plt.xlabel('Вік', fontsize=12)
    plt.ylabel('Кількість пацієнтів', fontsize=12)
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    # Додамо вертикальні лінії для середнього та медіани
    mean_age = unique_patients_df['Patient Age'].mean()
    median_age = unique_patients_df['Patient Age'].median()
    plt.axvline(mean_age, color='red', linestyle='dashed', linewidth=2, label=f'Середнє: {mean_age:.1f}')
    plt.axvline(median_age, color='green', linestyle='dashed', linewidth=2, label=f'Медіана: {median_age:.1f}')
    plt.legend()

    # Зберігаємо графік у файл
    plt.savefig('age_distribution.png')
    print("Графік 'age_distribution.png' збережено.")
    plt.close()

    # Виводимо текстовий вивід describe() для детального аналізу в роботі
    print("-" * 30)
    print("Статистичний опис віку пацієнтів (для текстового аналізу):")
    print(unique_patients_df['Patient Age'].describe())

else:
    print("Помилка: DataFrame 'df' не створено. Виконайте Блок 1.")

```

Рисунок А.2 - Код аналізу статі і віку пацієнтів

```python
# Аналіз аномалії (пацієнти з віком < 10 років)

if 'df' in locals():
    # Знаходимо всі записи, де вік пацієнта менший за 10 років.
    anomalous_age_df = df[df['Patient Age'] < 10].copy()

    if not anomalous_age_df.empty:
        num_anomalies = len(anomalous_age_df)
        print(f"Знайдено {num_anomalies} записів з віком < 10 років.")

        # Визначаємо шлях для збереження файлу
        output_csv_path = 'anomalous_age_records.csv'

        # Зберігаємо знайдені аномальні записи у новий CSV-файл
        try:
            anomalous_age_df.to_csv(output_csv_path, index=False, encoding='utf-8-sig')
            print(f"Таблиця з аномаліями збережена у файл '{output_csv_path}'.")
        except Exception as e:
            print(f"Помилка при збереженні файлу: {e}")

    else:
        print("Записів з аномально низьким віком (< 10 років) не знайдено.")

else:
    print("Помилка: DataFrame 'df' не створено. Виконайте Блок 1.")

```

Рисунок А.3 - Код аналізу вікової аномалії

```python
# Перевірка наявності пацієнтів з зображенням лише одного ока

if 'df' in locals():
    # Знаходимо пацієнтів з одним записом у .csv
    eye_counts_per_patient = df.groupby('ID').size()
    one_record_patient_ids = eye_counts_per_patient[eye_counts_per_patient == 1].index
    num_one_record_patients = len(one_record_patient_ids)

    print(f"Знайдено {num_one_record_patients} пацієнтів, що мають лише один запис в анотаціях.")
    print("-" * 70)

    # Перевірка файлів на диску для цих пацієнтів
    # Шлях до папки з обробленими зображеннями
    image_dir = os.path.expanduser('~/Study/Diploma/archive (1)/preprocessed_images/')

    missing_eye_data = [] # Тут будемо зберігати дані про відсутні очі

    # Беремо перші 5 для детального аналізу та виводу
    print("Проводимо вибіркову перевірку наявності файлів на диску для перших 5 пацієнтів:")

    # Знаходимо повні дані для всіх одноких
    one_record_df = df[df['ID'].isin(one_record_patient_ids)]

    for index, patient_row in one_record_df.iterrows():
        patient_id = patient_row['ID']
        # Отримуємо імена обох файлів з таблиці
        left_eye_filename = patient_row['Left-Fundus']
        right_eye_filename = patient_row['Right-Fundus']
        # Перевіряємо наявність обох файлів на диску
        left_eye_exists = os.path.exists(os.path.join(image_dir, left_eye_filename))
        right_eye_exists = os.path.exists(os.path.join(image_dir, right_eye_filename))
        # Визначаємо, який файл відсутній
        if not left_eye_exists and right_eye_exists:
            missing_eye = 'left'
        elif left_eye_exists and not right_eye_exists:
            missing_eye = 'right'
        elif not left_eye_exists and not right_eye_exists:
            missing_eye = 'both' # Аномальний випадок, якщо обидва відсутні
        else:
            missing_eye = 'none' # Аномальний випадок, якщо обидва присутні
        missing_eye_data.append({'ID': patient_id, 'missing_eye': missing_eye})

        # Виводимо детальний лог для перших 5 прикладів
        if patient_id in one_record_patient_ids[:5]:
            print(f"\n--- Пацієнт ID: {patient_id} ---")
            print(f"  Файл лівого ока: {left_eye_filename} -> Наявність: {'✅' if left_eye_exists else '❌'}")
            print(f"  Файл правого ока: {right_eye_filename} -> Наявність: {'✅' if right_eye_exists else '❌'}")

    # Створюємо та зберігаємо під-вибірку
    missing_eye_df = pd.DataFrame(missing_eye_data)
    missing_eye_df.to_csv('missing_eye_patients.csv', index=False)

    print("-" * 70)
    print("Створено під-вибірку 'missing_eye_patients.csv' з інформацією про відсутні очі.")
    # Виводимо статистику по відсутнім очам
    print("\nСтатистика по відсутнім очам:")
    print(missing_eye_df['missing_eye'].value_counts())
else:
    print("Помилка: DataFrame 'df' не створено. Виконайте Блок 1.")
```

Рисунок А.4 - Код аналізу пацієнтів з записом лише про одне око

```python
# Перевірка "Правила Норми"
# Гіпотеза: Якщо у пацієнта є хоча б одна патологія, то загальна мітка N=0.

if 'df' in locals():
    # Створюємо DataFrame з унікальними пацієнтами
    unique_patients_df = df.drop_duplicates(subset=['ID']).set_index('ID')

    # Визначаємо колонки патологій
    pathology_cols = ['D', 'G', 'C', 'A', 'H', 'M', 'O']

    # Додаємо колонку, яка показує, чи є у пацієнта хоча б одна патологія
    unique_patients_df['has_pathology'] = (unique_patients_df[pathology_cols].sum(axis=1) > 0)

    # Знаходимо випадки, де є патологія, АЛЕ загальна мітка N=1.
    # Це будуть випадки, що спростовують гіпотезу.
    rule_breakers = unique_patients_df[(unique_patients_df['has_pathology'] == True) & (unique_patients_df['N'] == 1)]

    num_rule_breakers = len(rule_breakers)

    print("Перевірка 'Правила Норми'...")
    print("-" * 50)

    if num_rule_breakers == 0:
        print("Результат: Правило підтверджено для всього датасету.")
        print("У жодного пацієнта з наявністю патології (D,G,C,A,H,M,O) не стоїть загальна мітка 'Норма' (N=1).")
    else:
        print(f"Результат: Правило спростовано. Знайдено {num_rule_breakers} випадків-виключень.")
        print("Це означає, що логіка маркування більш складна.")
        print("Перші 5 випадків-виключень:")
        print(rule_breakers.head())

else:
    print("Помилка: DataFrame 'df' не створено. Виконайте Блок 1.")
```

Рисунок А.5 - Код аналізу “правила норми”

```python
# Аналіз мультилейбловості (пошук коми)

if 'df' in locals():
    # Створення чистої таблиці очей
    # Це гарантує, що ми аналізуємо кожне око окремо і не рахуємо нічого двічі
    df_left = df[['filename', 'Left-Diagnostic Keywords']].rename(columns={'Left-Diagnostic Keywords': 'keywords'})
    df_left = df_left[df_left['filename'].str.contains('_left')]

    df_right = df[['filename', 'Right-Diagnostic Keywords']].rename(columns={'Right-Diagnostic Keywords': 'keywords'})
    df_right = df_right[df_right['filename'].str.contains('_right')]

    df_eyes = pd.concat([df_left, df_right]).sort_index().dropna(subset=['keywords'])

    # Пошук записів з комою
    multilabel_cases = df_eyes[df_eyes['keywords'].str.contains('[,]', regex=True, na=False)]

    num_multilabel_cases = len(multilabel_cases)
    total_eyes = len(df_eyes)
    percentage = (num_multilabel_cases / total_eyes) * 100

    # Вивід результатів
    print("Аналіз наявності кількох діагнозів для одного ока")
    print(f"Знайдено {num_multilabel_cases} зображень (з {total_eyes} проаналізованих, що складає {percentage:.2f}%), де в текстовому описі є кома.")

    if num_multilabel_cases > 0:
        print("\nПерші 10 прикладів таких записів:")
        # Виводимо повний текст без обрізання
        for index, row in multilabel_cases.head(10).iterrows():
            print(f"  - {row['keywords']}")

    print("-" * 70)
else:
    print("Помилка: DataFrame 'df' не створено.")
```

Рисунок А.6 - Код аналізу діагностичних описів зображень на наявність коми

```python
# Аналіз 'labels' та 'target'
if 'df' in locals():
    print("\n🔍 Аналіз стовпчиків 'labels' та 'target'")

    # Створення чистої таблиці очей
    df_left = df[['ID', 'filename', 'Left-Diagnostic Keywords', 'labels', 'target']].rename(columns={'Left-Diagnostic Keywords': 'keywords'})
    df_left = df_left[df_left['filename'].str.contains('_left')].copy()

    df_right = df[['ID', 'filename', 'Right-Diagnostic Keywords', 'labels', 'target']].rename(columns={'Right-Diagnostic Keywords': 'keywords'})
    df_right = df_right[df_right['filename'].str.contains('_right')].copy()

    df_eyes = pd.concat([df_left, df_right]).sort_index().dropna(subset=['keywords'])

    # Перевірка гіпотези про однокласовість 'labels' та 'target'
    df_eyes['labels_count'] = df_eyes['labels'].str.count("'") // 2
    df_eyes['target_count'] = df_eyes['target'].str.count("1")
    exceptions_labels = df_eyes[df_eyes['labels_count'] > 1]
    exceptions_target = df_eyes[df_eyes['target_count'] > 1]

    print(f"Перевірка на мультилейбловість: знайдено {len(exceptions_labels)} випадків в 'labels' та {len(exceptions_target)} в 'target'.")

    # Демонстрація неповноти даних
    if len(exceptions_labels) == 0 and len(exceptions_target) == 0:
        print("Гіпотеза підтверджена: 'labels' та 'target' завжди є однокласовими.")
        # Знаходимо випадки, де в keywords є кома
        df_multilabel_eyes = df_eyes[df_eyes['keywords'].str.contains('[,،]', regex=True, na=False)]

        if not df_multilabel_eyes.empty:
            print("\nПриклади, що демонструють втрату інформації:")
            cols_to_show = ['ID', 'filename', 'keywords', 'labels', 'target']
            examples_to_show = df_multilabel_eyes.sort_values(by='keywords', key=lambda col: col.str.len(), ascending=False).head(10)
            print(examples_to_show[cols_to_show].to_string())
    print("-" * 70)
else:
    print("Помилка: DataFrame 'df' не створено.")
```

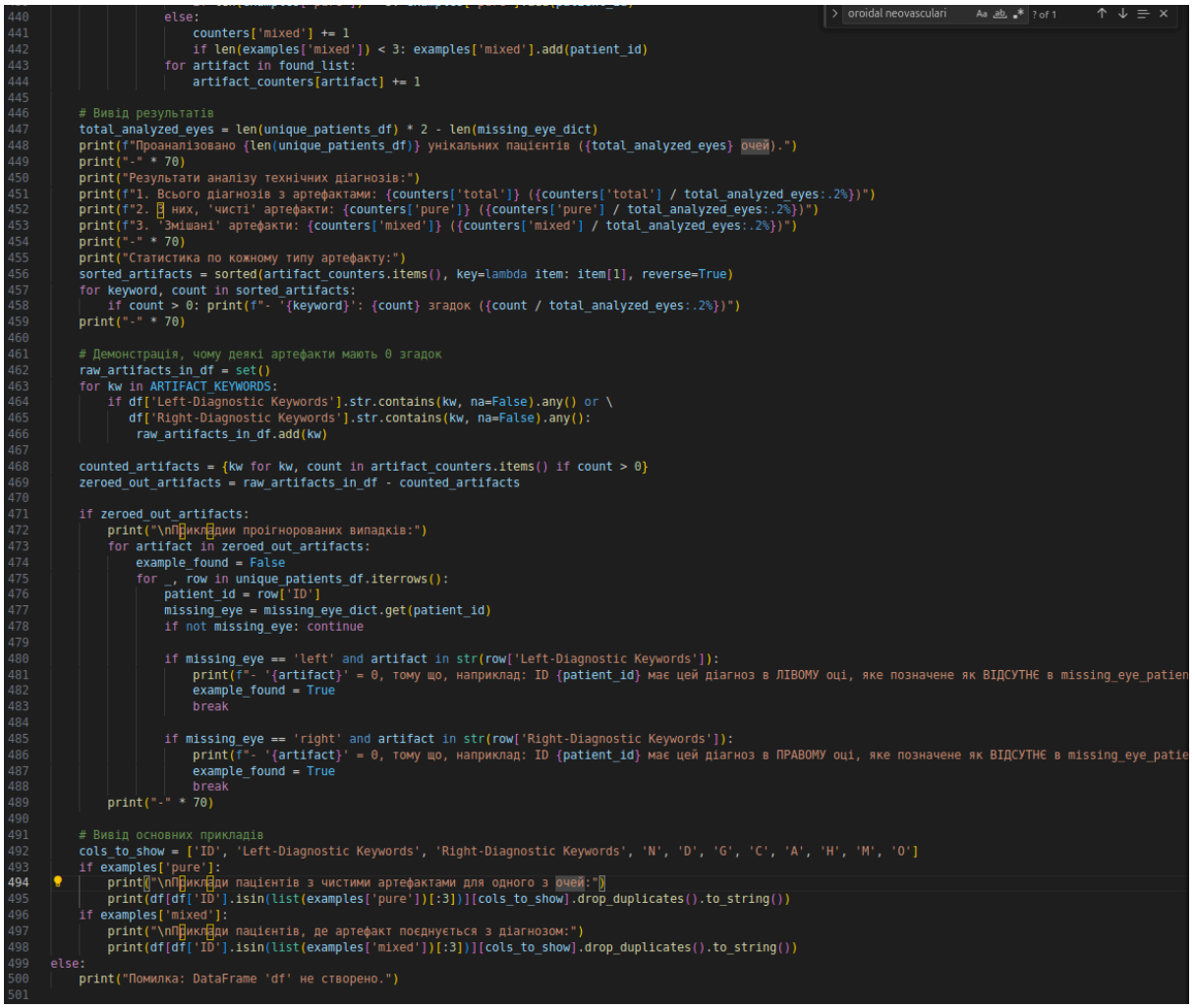
Рисунок А.7 - Код аналізу стовпчиків label та target

```python
# Аналіз унікальних діагнозів

if 'df' in locals():
    # Об'єднуємо діагнози з обох колонок
    all_keywords_series = pd.concat([df['Left-Diagnostic Keywords'], df['Right-Diagnostic Keywords']], ignore_index=True)
    # Створюємо множину для унікальних діагнозів
    unique_atomic_diagnoses = set()
    # Ітеруємо і розділяємо по комам
    for record in all_keywords_series:
        atomic_diagnoses = [diag.strip() for diag in str(record).replace(',', '،').split('،')]
        unique_atomic_diagnoses.update(atomic_diagnoses)

    # Виводимо результати
    # Видаляємо порожні рядки, якщо є
    unique_atomic_diagnoses.discard('')
    num_unique_diagnoses = len(unique_atomic_diagnoses)
    sorted_diagnoses = sorted(list(unique_atomic_diagnoses))
    print("Аналіз унікальних діагнозів, вписаних лікарями...")
    print("-" * 60)
    print(f"Знайдено {num_unique_diagnoses} унікальних записів.")
    print("-" * 60)
    # Виводимо у кілька стовпчиків
    print("Повний словник унікальних діагнозів:")
    num_columns = 3
    # Розраховуємо ширину, додаючи перевірку на випадок порожнього списку
    if sorted_diagnoses:
        col_width = max(len(word) for word in sorted_diagnoses) + 4
        for i in range(0, num_unique_diagnoses, num_columns):
            row_diagnoses = sorted_diagnoses[i:i+num_columns]
            formatted_row = "".join(f"{word:<{col_width}}" for word in row_diagnoses)
            print(formatted_row)
else:
    print("Помилка: DataFrame 'df' не створено.")

```

Рисунок А.8 - Код аналізу унікальних діагностичних записів

```python
if 'df' in locals():
    # Підготовка
    ARTIFACT_KEYWORDS = [
        'low image quality', 'image offset', 'lens dust', 'no fundus image',
        'optic disk photographically invisible', 'refractive media opacity',
        'anterior segment image', 'vitreous opacity'
    ]
    NON_PATHOLOGY_TERMS = ARTIFACT_KEYWORDS

    MISSING_EYE_FILE = 'missing_eye_patients.csv'
    missing_eye_dict = {}
    if os.path.exists(MISSING_EYE_FILE):
        missing_eye_df = pd.read_csv(MISSING_EYE_FILE)
        missing_eye_dict = missing_eye_df.set_index('ID')['missing_eye'].to_dict()
        print(f"Завантажено дані про {len(missing_eye_dict)} однооких пацієнтів.")
    else:
        print(f"Файл '{MISSING_EYE_FILE}' не знайдено.")
    print("-" * 70)

    # Функція для аналізу
    def analyze_eye_keywords(keywords_str):
        if not isinstance(keywords_str, str): return False, False, []
        keywords_lower = keywords_str.lower()
        found_artifacts = [kw for kw in ARTIFACT_KEYWORDS if kw in keywords_lower]
        if not found_artifacts: return False, False, []
        temp_str = keywords_lower
        for term in NON_PATHOLOGY_TERMS:
            temp_str = temp_str.replace(term, '')
        temp_str = temp_str.replace('', '').strip()
        is_pure = len(temp_str) == 0
        return True, is_pure, found_artifacts

    # Аналіз унікальних пацієнтів
    counters = {'total': 0, 'pure': 0, 'mixed': 0}
    artifact_counters = {kw: 0 for kw in ARTIFACT_KEYWORDS}
    examples = {'pure': set(), 'mixed': set()}

    unique_patients_df = df.drop_duplicates(subset=['ID'])

    for _, patient_row in unique_patients_df.iterrows():
        patient_id = patient_row['ID']
        missing_eye = missing_eye_dict.get(patient_id)

        if missing_eye != 'left':
            has_artifact, is_pure, found_list = analyze_eye_keywords(patient_row['Left-Diagnostic Keywords'])
            if has_artifact:
                counters['total'] += 1
                if is_pure:
                    counters['pure'] += 1
                    if len(examples['pure']) < 3: examples['pure'].add(patient_id)
                else:
                    counters['mixed'] += 1
                    if len(examples['mixed']) < 3: examples['mixed'].add(patient_id)
                for artifact in found_list:
                    artifact_counters[artifact] += 1

        if missing_eye != 'right':
            has_artifact, is_pure, found_list = analyze_eye_keywords(patient_row['Right-Diagnostic Keywords'])
            if has_artifact:
                counters['total'] += 1
                if is_pure:
                    counters['pure'] += 1
                    if len(examples['pure']) < 3: examples['pure'].add(patient_id)
                else:
```

Рисунок А.9.1 - Код аналізу технічних артефактів




```
                else:
                    counters['mixed'] += 1
                    if len(examples['mixed']) < 3: examples['mixed'].add(patient_id)
                for artifact in found_list:
                    artifact_counters[artifact] += 1

    # Вивід результатів
    total_analyzed_eyes = len(unique_patients_df) * 2 - len(missing_eye_dict)
    print(f"Проаналізовано {len(unique_patients_df)} унікальних пацієнтів ({total_analyzed_eyes} очей).")
    print("-" * 70)
    print("Результати аналізу технічних діагнозів:")
    print(f"1. Всього діагнозів з артефактами: {counters['total']} ({counters['total'] / total_analyzed_eyes:.2%})")
    print(f"2. З них, 'чисті' артефакти: {counters['pure']} ({counters['pure'] / total_analyzed_eyes:.2%})")
    print(f"3. 'Змішані' артефакти: {counters['mixed']} ({counters['mixed'] / total_analyzed_eyes:.2%})")
    print("-" * 70)
    print("Статистика по кожному типу артефакту:")
    sorted_artifacts = sorted(artifact_counters.items(), key=lambda item: item[1], reverse=True)
    for keyword, count in sorted_artifacts:
        if count > 0: print(f"- '{keyword}': {count} згадок ({count / total_analyzed_eyes:.2%})")
    print("-" * 70)

    # Демонстрація, чому деякі артефакти мають 0 згадок
    raw_artifacts_in_df = set()
    for kw in ARTIFACT_KEYWORDS:
        if df['Left-Diagnostic Keywords'].str.contains(kw, na=False).any() or \
           df['Right-Diagnostic Keywords'].str.contains(kw, na=False).any():
            raw_artifacts_in_df.add(kw)

    counted_artifacts = {kw for kw, count in artifact_counters.items() if count > 0}
    zeroed_out_artifacts = raw_artifacts_in_df - counted_artifacts

    if zeroed_out_artifacts:
        print("\nПриклади проігнорованих випадків:")
        for artifact in zeroed_out_artifacts:
            example_found = False
            for _, row in unique_patients_df.iterrows():
                patient_id = row['ID']
                missing_eye = missing_eye_dict.get(patient_id)
                if not missing_eye: continue

                if missing_eye == 'left' and artifact in str(row['Left-Diagnostic Keywords']):
                    print(f"- '{artifact}' = 0, тому що, наприклад: ID {patient_id} має цей діагноз в ЛІВОМУ оці, яке позначене як ВІДСУТНЄ в missing_eye_patien
                    example_found = True
                    break

                if missing_eye == 'right' and artifact in str(row['Right-Diagnostic Keywords']):
                    print(f"- '{artifact}' = 0, тому що, наприклад: ID {patient_id} має цей діагноз в ПРАВОМУ оці, яке позначене як ВІДСУТНЄ в missing_eye_patie
                    example_found = True
                    break
            print("-" * 70)

    # Вивід основних прикладів
    cols_to_show = ['ID', 'Left-Diagnostic Keywords', 'Right-Diagnostic Keywords', 'N', 'D', 'G', 'C', 'A', 'H', 'M', 'O']
    if examples['pure']:
        print("\nПриклади пацієнтів з чистими артефактами для одного з очей:")
        print(df[df['ID'].isin(list(examples['pure'])[:3])][cols_to_show].drop_duplicates().to_string())
    if examples['mixed']:
        print("\nПриклади пацієнтів, де артефакт поєднується з діагнозом:")
        print(df[df['ID'].isin(list(examples['mixed'])[:3])][cols_to_show].drop_duplicates().to_string())
else:
    print("Помилка: DataFrame 'df' не створено.")
```

Рисунок А.9.2 - код аналізу технічних артефактів

```
# Тиха перевірка карти
validation_errors = []
# Перевірка на дублікати
all_rules_list = [rule for sublist in keyword_to_label_map.values() for rule in sublist]
if len(all_rules_list) != len(set(all_rules_list)):
    seen = set()
    duplicates = {x for x in all_rules_list if x in seen or seen.add(x)}
    validation_errors.append(f"Знайдено дублікати в карті: {list(duplicates)}")

# Перевірка на повноту (порівняння з попереднім аналізом унікальних діагнозів)
if 'sorted_diagnoses' in locals():
    original_diagnoses_set = set(sorted_diagnoses)
    map_set = set(all_rules_list)
    unmapped = original_diagnoses_set - map_set
    if unmapped:
        validation_errors.append(f"{len(unmapped)} діагнозів НЕ потрапили в карту: {unmapped}")
    mistakes = map_set - original_diagnoses_set
    if mistakes:
        validation_errors.append(f"{len(mistakes)} правил з карти НЕ було в оригіналі: {mistakes}")
    if len(all_rules_list) != len(original_diagnoses_set):
        validation_errors.append(f"Невідповідність кількості: {len(all_rules_list)} в карті проти {len(original_diagnoses_set)} в оригіналі.")
else:
    validation_errors.append("Неможливо перевірити повноту карти.")

# Вивід результатів
if validation_errors:
    # Якщо є помилки, виводимо їх всі
    print("УВАГА: ЗНАЙДЕНО ПОМИЛКИ ПРИ ПЕРЕВІРЦІ КАРТИ ПРАВИЛ")
    for error in validation_errors:
        print(f"- {error}")
else:
    # Якщо все добре, виводимо статистику
    print("Перевірка карти правил пройшла успішно.")
    print("Статистика розподілу діагнозів по класах:")
    print("-" * 70)
    total_rules = 0
    for label, keywords in keyword_to_label_map.items():
        count = len(keywords)
        total_rules += count
        print(f"Клас '{label}': віднесено {count} унікальних термінів.")
    print("-" * 70)
    print(f"Загальна кількість правил у карті: {total_rules}")
```

Рисунок А.10 - Код верифікації створеної карти правил

```python
# Генерація фінального, чистого датасету
if 'df' in locals() and 'keyword_to_label_map' in locals():
    # Підготовка
    # Створюємо карту для швидкого пошуку: {'термін': 'КЛАС'}
    label_lookup = {
        keyword: label
        for label, keywords in keyword_to_label_map.items()
        for keyword in keywords
    }
    # Список слів-артефактів
    ARTIFACT_KEYWORDS = [
        'low image quality', 'image offset', 'lens dust', 'no fundus image',
        'optic disk photographically invisible', 'refractive media opacity',
        'anterior segment image', 'vitreous opacity'
    ]

    # Функції-фільтр та перетворювач
    def is_pure_artifact(keywords_str):
        if not isinstance(keywords_str, str): return False
        parts = [p.strip().lower() for p in keywords_str.replace(',', '，').split('，')]
        # Повертає True, тільки якщо всі частини діагнозу є артефактами
        return all(p in ARTIFACT_KEYWORDS for p in parts if p)

    def generate_label_vector(keywords_str):
        labels = {'N': 0, 'D': 0, 'G': 0, 'C': 0, 'A': 0, 'H': 0, 'M': 0, 'O': 0}
        if not isinstance(keywords_str, str): return list(labels.values())
        parts = [p.strip().lower() for p in keywords_str.replace(',', '，').split('，')]
        for part in parts:
            label = label_lookup.get(part)
            if label:
                labels[label] = 1
        return list(labels.values())

    # Головний цикл
    final_data = []
    filtered_count = 0
    initial_count = len(df)

    for _, row in df.iterrows():
        # Визначаємо, з яким оком працюємо
        if '_left.jpg' in row['filename']:
            keywords = row['Left-Diagnostic Keywords']
        else:
            keywords = row['Right-Diagnostic Keywords']

        # Фільтруємо "чисто технічні" діагнози
        if is_pure_artifact(keywords):
            filtered_count += 1
            continue # Пропускаємо цей рядок
        # Генеруємо вектор міток
        label_vector = generate_label_vector(keywords)
        # Збираємо фінальний запис
        record = {
            'ID': row['ID'],
```

Рисунок А.11.1 - Код створення нового набору даних

```
682             # Збираємо фінальний запис
683             record = {
684                 'ID': row['ID'],
685                 'Patient Age': row['Patient Age'],
686                 'Patient Sex': row['Patient Sex'],
687                 'filename': row['filename']
688             }
689             record.update(zip(['N', 'D', 'G', 'C', 'A', 'H', 'M', 'O'], label_vector))
690             final_data.append(record)
691 
692         # Збереження та статистика
693         final_df = pd.DataFrame(final_data)
694         output_filename = 'final_dataset.csv'
695         final_df.to_csv(output_filename, index=False)
696 
697         print(f"Файл '{output_filename}' успішно створено.")
698         print("-" * 70)
699         print("Статистика процесу:")
700         print(f"Початкова кількість зображень: {initial_count}")
701         print(f"Відфільтровано 'чисто технічних' зображень: {filtered_count}")
702         print(f"Фінальна кількість зображень у датасеті: {len(final_df)}")
703         print("-" * 70)
704     else:
705         print("Помилка: Необхідні змінні не знайдено.")
```

Рисунок А.11.2 - Код створення нового набору даних

```
737 # Валідація фінального датасету
738 print("\n\n--- Початок повної валідації 'final_dataset.csv' ---")
739 # Крок 1: Завантаження даних
740 try:
741     df_new = pd.read_csv('final_dataset.csv')
742     df_old = df # Використовуємо df, вже завантажений раніше
743     print("Файли 'final_dataset.csv' та 'full_df.csv' успішно завантажені.")
744 except FileNotFoundError:
745     print("Увага: Не вдалося завантажити дані!")
746     exit()
747 
748 # Крок 2: Базова перевірка цілісності
749 print("\nБазова перевірка цілісності")
750 df_new.info()
751 print("-" * 70)
752 
```

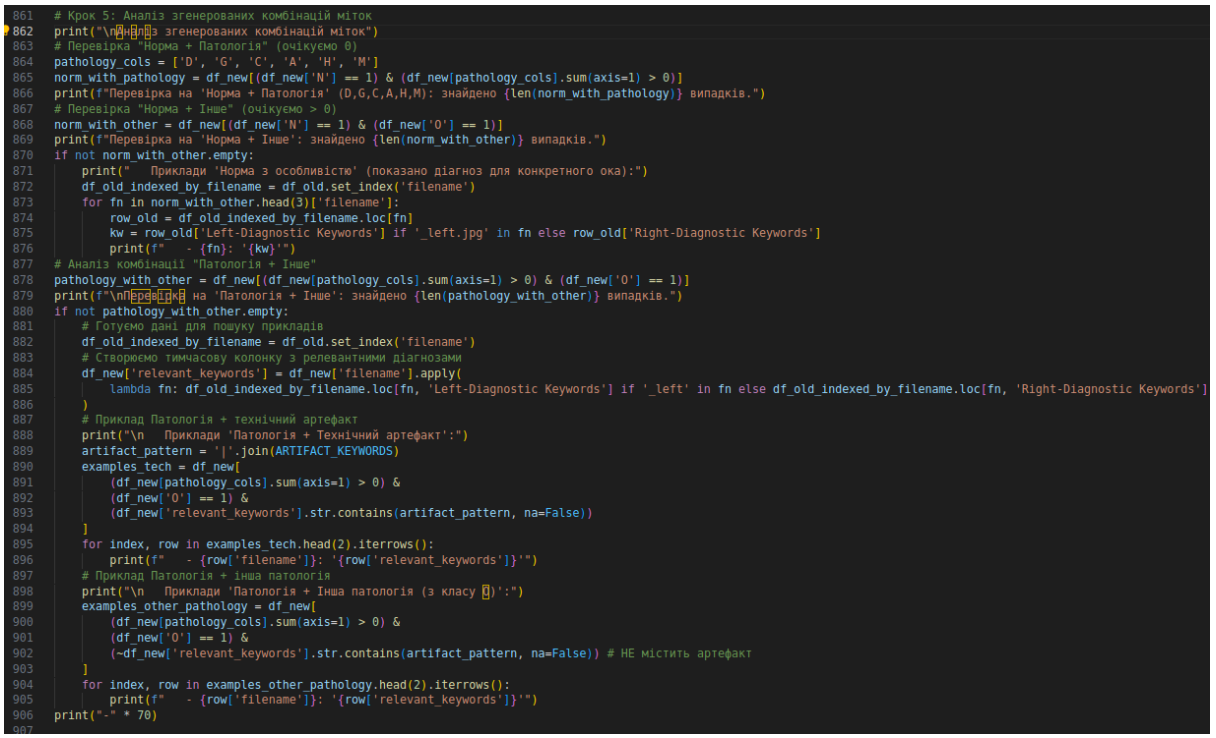
Рисунок А.12 - Код валідації нового набору даних

```python
# Крок 3: Детальний аналіз демографії
print("\nДетальний аналіз демографії")
old_patient_ids = set(df_old['ID'].unique())
new_patient_ids = set(df_new['ID'].unique())
lost_patient_ids = old_patient_ids - new_patient_ids
print(f"Кількість унікальних пацієнтів: {len(new_patient_ids)} (було {len(old_patient_ids)}, втрачено {len(lost_patient_ids)})")

# Перевірка втрачених пацієнтів
if lost_patient_ids:
    artifact_keywords_for_check = ['low image quality', 'image offset', 'lens dust', 'no fundus image',
                                   'optic disk photographically invisible', 'refractive media opacity',
                                   'anterior segment image', 'vitreous opacity']
    def is_pure_artifact_validator(keywords_str):
        if not isinstance(keywords_str, str): return False
        parts = [p.strip().lower() for p in keywords_str.replace(',', '，').split('，')]
        return all(p in artifact_keywords_for_check for p in parts if p)

    lost_patients_proof_count = 0
    unique_old_df_for_check = df_old.drop_duplicates(subset=['ID']).set_index('ID')
    image_dir = os.path.expanduser('~/Study/Diploma/archive (1)/preprocessed_images/')

    print("\nПеревірка втрачених пацієнтів:")
    for patient_id in lost_patient_ids:
        row = unique_old_df_for_check.loc[patient_id]

        left_exists = os.path.exists(os.path.join(image_dir, row['Left-Fundus']))
        right_exists = os.path.exists(os.path.join(image_dir, row['Right-Fundus']))

        is_pure_l = is_pure_artifact_validator(row['Left-Diagnostic Keywords'])
        is_pure_r = is_pure_artifact_validator(row['Right-Diagnostic Keywords'])

        lost_due_to_artifacts = False

        if left_exists and right_exists:
            if is_pure_l and is_pure_r: lost_due_to_artifacts = True
        elif left_exists and not right_exists:
            if is_pure_l: lost_due_to_artifacts = True
        elif not left_exists and right_exists:
            if is_pure_r: lost_due_to_artifacts = True
        elif not left_exists and not right_exists:
            lost_due_to_artifacts = True

        if lost_due_to_artifacts:
            lost_patients_proof_count += 1
            print(f"  - ID: {patient_id} (Існує L: {'✅' if left_exists else '❌'}, R: {'✅' if right_exists else '❌'}), L_kw: '{row['Left-Diagnos

    print(f"\nЗнайдено {lost_patients_proof_count} пацієнтів, у яких всі існуючі зображення були чистими артефактами. Це пояснює їх втрату.")
```

Рисунок А.13 - Код аналізу демографічних даних нового набору даних

```python
# Крок 4: Перехресна верифікація міток
print("\nПерехресна верифікація міток між 'final_dataset.csv' та 'full_df.csv'")
mismatches = {'norm_rule': [], 'artifact_rule': [], 'other': []}
df_old_patient_indexed = df_old.drop_duplicates(subset=['ID']).set_index('ID')
df_new_grouped = df_new.groupby('ID')
one_eye_patient_ids_in_new = set(one_eye_patients_new.index) # Беремо ID з нового датасету

for patient_id, group in df_new_grouped:
    new_vector = group[['N', 'D', 'G', 'C', 'A', 'H', 'M', 'O']].sum().clip(upper=1).astype(int)
    old_vector = df_old_patient_indexed.loc[patient_id][['N', 'D', 'G', 'C', 'A', 'H', 'M', 'O']].astype(int)

    is_mismatch = False

    # Визначаємо, чи є розбіжність
    if patient_id in one_eye_patient_ids_in_new:
        # Логіка для одноokих
        if any(val == 1 and old_vector[label] == 0 for label, val in new_vector.items()):
            is_mismatch = True
    else:
        # Логіка для двоokих
        if not new_vector.equals(old_vector):
            is_mismatch = True

    if is_mismatch:
        example = {
            'ID': patient_id, 'new': list(new_vector), 'old': list(old_vector),
            'L_kw': df_old_patient_indexed.loc[patient_id, 'Left-Diagnostic Keywords'],
            'R_kw': df_old_patient_indexed.loc[patient_id, 'Right-Diagnostic Keywords']
        }

        # Класифікуємо невідповідність
        is_norm_rule = new_vector['N'] == 1 and old_vector['N'] == 0
        is_artifact_rule = new_vector['O'] == 1 and old_vector['O'] == 0 and any(kw in (str(example['L_kw']) + str(example['R_kw'])) for kw in ARTIFACT_KEYWORDS)

        if is_norm_rule:
            mismatches['norm_rule'].append(example)
        elif is_artifact_rule:
            mismatches['artifact_rule'].append(example)
        else:
            mismatches['other'].append(example)

# Вивід результатів діагностики
total_mismatches = sum(len(v) for v in mismatches.values())
print(f"Знайдено {total_mismatches} загальних невідповідностей:")
for key, value in mismatches.items():
    print(f"- Тип '{key}': {len(value)} випадків.")
    if value:
        print(f"  Приклад: {value[0]}")
print("-" * 70)
```

Рисунок А.14 - Код перехресної верифікації згенерованих міток нового набору даних



```
# Крок 5: Аналіз згенерованих комбінацій міток
print("\n📊 Аналіз згенерованих комбінацій міток")
# Перевірка "Норма + Патологія" (очікуємо 0)
pathology_cols = ['D', 'G', 'C', 'A', 'H', 'M']
norm_with_pathology = df_new[(df_new['N'] == 1) & (df_new[pathology_cols].sum(axis=1) > 0)]
print(f"Перевірка на 'Норма + Патологія' (D,G,C,A,H,M): знайдено {len(norm_with_pathology)} випадків.")
# Перевірка "Норма + Інше" (очікуємо > 0)
norm_with_other = df_new[(df_new['N'] == 1) & (df_new['O'] == 1)]
print(f"Перевірка на 'Норма + Інше': знайдено {len(norm_with_other)} випадків.")
if not norm_with_other.empty:
    print("   Приклади 'Норма з особливістю' (показано діагноз для конкретного ока):")
    df_old_indexed_by_filename = df_old.set_index('filename')
    for fn in norm_with_other.head(3)['filename']:
        row_old = df_old_indexed_by_filename.loc[fn]
        kw = row_old['Left-Diagnostic Keywords'] if '_left.jpg' in fn else row_old['Right-Diagnostic Keywords']
        print(f"    - {fn}: '{kw}'")
# Аналіз комбінації "Патологія + Інше"
pathology_with_other = df_new[(df_new[pathology_cols].sum(axis=1) > 0) & (df_new['O'] == 1)]
print(f"\n🔍 Перевірка на 'Патологія + Інше': знайдено {len(pathology_with_other)} випадків.")
if not pathology_with_other.empty:
    # Готуємо дані для пошуку прикладів
    df_old_indexed_by_filename = df_old.set_index('filename')
    # Створюємо тимчасову колонку з релевантними діагнозами
    df_new['relevant_keywords'] = df_new['filename'].apply(
        lambda fn: df_old_indexed_by_filename.loc[fn, 'Left-Diagnostic Keywords'] if '_left' in fn else df_old_indexed_by_filename.loc[fn, 'Right-Diagnostic Keywords']
    )
    # Приклад Патологія + технічний артефакт
    print("\n   Приклади 'Патологія + Технічний артефакт':")
    artifact_pattern = '|'.join(ARTIFACT_KEYWORDS)
    examples_tech = df_new[
        (df_new[pathology_cols].sum(axis=1) > 0) &
        (df_new['O'] == 1) &
        (df_new['relevant_keywords'].str.contains(artifact_pattern, na=False))
    ]
    for index, row in examples_tech.head(2).iterrows():
        print(f"    - {row['filename']}: '{row['relevant_keywords']}'")
    # Приклад Патологія + інша патологія
    print("\n   Приклади 'Патологія + Інша патологія (з класу D)':")
    examples_other_pathology = df_new[
        (df_new[pathology_cols].sum(axis=1) > 0) &
        (df_new['O'] == 1) &
        (~df_new['relevant_keywords'].str.contains(artifact_pattern, na=False)) # НЕ містить артефакт
    ]
    for index, row in examples_other_pathology.head(2).iterrows():
        print(f"    - {row['filename']}: '{row['relevant_keywords']}'")
print("-" * 70)
```

Рисунок А.15 - Код аналізу “правила норми” та “правила артефактів” нового набору даних

```python
# Крок 6: Фінальний аналіз мультилейбловості з повною діагностикою
print("\nФінальний аналіз мультилейбловості з діагностикою")
# Аналіз фінального датасету (df_new)
label_cols = ['N', 'D', 'G', 'C', 'A', 'H', 'M', 'O']
if 'num_labels' in df_new.columns: df_new = df_new.drop(columns=['num_labels'])
df_new['num_labels'] = df_new[label_cols].sum(axis=1)
label_distribution = df_new['num_labels'].value_counts().sort_index()
num_real_multilabel_clean = label_distribution[label_distribution.index >= 2].sum()
print("Аналіз фінального датасету (df_new):")
print(f"- Знайдено {num_real_multilabel_clean} мультилейблових зображень.")
print("-" * 70)

# Аналіз оригінального датасету (df_old)
# Створюємо чисту таблицю очей
df_left = df_old[['filename', 'Left-Diagnostic Keywords']].rename(columns={'Left-Diagnostic Keywords': 'keywords'})
df_left = df_left[df_left['filename'].str.contains('_left')]
df_right = df_old[['filename', 'Right-Diagnostic Keywords']].rename(columns={'Right-Diagnostic Keywords': 'keywords'})
df_right = df_right[df_right['filename'].str.contains('_right')]
df_eyes_old = pd.concat([df_left, df_right]).sort_index().dropna(subset=['keywords'])
# Рахуємо мультилейбли (по комі)
raw_multilabel_mask_by_comma = df_eyes_old['keywords'].str.contains('[,]', regex=True, na=False)
num_raw_multilabel = raw_multilabel_mask_by_comma.sum()
# Рахуємо мультилейбли з урахуванням карти
label_lookup = {kw: lbl for lbl, kws in keyword_to_label_map.items() for kw in kws}
def count_unique_labels(keywords_str):
    labels = {label_lookup.get(p.strip().lower()) for p in str(keywords_str).replace(',', '.').split('.') if label_lookup.get(p.strip().lower())}
    return len(labels)
df_eyes_old['num_generated_labels'] = df_eyes_old['keywords'].apply(count_unique_labels)
num_real_multilabel_raw = (df_eyes_old['num_generated_labels'] >= 2).sum()
print("Аналіз оригінального датасету (df_old):")
print(f"- Знайдено {num_raw_multilabel} мультилейблових записів через кому.")
print(f"- З урахуванням карти правил, мультилейблів в оригіналі: {num_real_multilabel_raw}.")
print("-" * 70)

# Діагностика різниці
print("Діагностика різниці:")
# Пояснюємо різницю між через кому і через карту підрахунком
num_pseudo = num_raw_multilabel - num_real_multilabel_raw
print(f"\nПояснення розбіжності між методом по комі та методом з картою:")
print(f"Різниця складає {num_pseudo} випадків ({num_raw_multilabel} - {num_real_multilabel_raw}).")
print("Це записи, де кілька термінів відносяться до одного класу.")
# Доказ: знаходимо і виводимо приклади
pseudo_multilabel_df = df_eyes_old[raw_multilabel_mask_by_comma & (df_eyes_old['num_generated_labels'] < 2)].copy()
if not pseudo_multilabel_df.empty:
    pseudo_multilabel_df['generated_labels'] = pseudo_multilabel_df['keywords'].apply(
        lambda x: {label_lookup.get(p.strip().lower()) for p in str(x).replace(',', '.').split('.') if label_lookup.get(p.strip().lower())}
    )
    print("Приклади:")
    print(pseudo_multilabel_df[['keywords', 'generated_labels']].head(5).to_string())

# Вивід фінальної статистики
print("\nФінальний розподіл зображень за кількістю міток:")
print(label_distribution)

if 2 in label_distribution.index:
    print("\nПриклади зображень з 2 мітками з фінального датасету:")
    print(df_new[df_new['num_labels'] == 2].head(3).to_string())
if 3 in label_distribution.index:
    print("\nПриклади зображень з 3 мітками з фінального датасету:")
    print(df_new[df_new['num_labels'] == 3].to_string())
```

Рисунок А.16 - Код аналізу типу задачі нового набору даних

```python
# Аналіз дисбалансу класів
if 'df_new' in locals():
    print("\n\n--- Аналіз дисбалансу класів у 'final_dataset.csv'")

    # Визначаємо колонки з мітками класів у правильному порядку (як в оригіналі)
    label_columns = ['N', 'D', 'G', 'C', 'A', 'H', 'M', 'O']

    # Рахуємо кількість зображень для кожного класу
    class_distribution = df_new[label_columns].sum()

    # Розраховуємо відсотки
    total_images = len(df_new)
    class_percentages = (class_distribution / total_images) * 100

    print("Кількість зображень для кожного діагностичного класу у фінальному датасеті:")
    print("-" * 70)
    # Створюємо DataFrame для виводу
    dist_df = pd.DataFrame({'Кількість': class_distribution, 'Відсоток (%)': class_percentages.round(2)})
    print(dist_df)
    print("-" * 70)

    # Візуалізація
    plt.figure(figsize=(12, 8))

    # Використовуємо .loc[label_columns] щоб гарантувати правильний порядок
    bars = plt.bar(class_distribution.loc[label_columns].index, class_distribution.loc[label_columns].values, color='steelblue', edgecolor='black')

    plt.title(f'Фінальний розподіл класів у датасеті (N={total_images})', fontsize=16)
    plt.xlabel('Діагноз', fontsize=12)
    plt.ylabel('Кількість зображень', fontsize=12)
    plt.xticks(rotation=0)
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    # Додаємо текстові мітки з кількістю над кожним стовпчиком
    for bar in bars:
        yval = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2.0, yval + 30, int(yval), ha='center', va='bottom')

    # Зберігаємо графік у файл
    plt.savefig('final_class_imbalance.png', bbox_inches='tight')
    print("Графік 'final_class_imbalance.png' збережено.")
    plt.close()
else:
    print("ПОМИЛКА: DataFrame 'df_new' не знайдено. Запустіть Блок 11.")
```

Рисунок А.17 - Код аналізу балансу класів новогонабору даних

ДОДАТОК Б - Формування вибірок для навчання та валідації

```python
import pandas as pd
from sklearn.model_selection import train_test_split
import os
from itertools import combinations
from collections import OrderedDict
import textwrap

# Налаштування
DATASET_PATH = 'final_dataset.csv'
OUTPUT_PATH = 'final_dataset_with_splits.csv'
VAL_SIZE = 0.15
RANDOM_STATE = 42
N_SAMPLES_PER_CRITERION = 3
LABEL_COLS = ['N', 'D', 'G', 'C', 'A', 'H', 'M', 'O']
PRIMARY_PATHOLOGY_COLS = ['D', 'G', 'C', 'A', 'H', 'M']
COLUMN_WIDTH = 55 # Ширина стовпчика для виводу статистики

def print_side_by_side(blocks, col_width):
    """Допоміжна функція для виводу блоків тексту у стовпчиках."""
    lines_per_block = [b.split('\n') for b in blocks]
    max_lines = max(len(lines) for lines in lines_per_block)
    for lines in lines_per_block:
        lines.extend([""] * (max_lines - len(lines)))
    for i in range(max_lines):
        row_str = "".join([lines[i].ljust(col_width) for lines in lines_per_block])
        print(row_str)

print(f"Завантаження даних з {DATASET_PATH}...")
if not os.path.exists(DATASET_PATH):
    print(f"ПОМИЛКА: Файл '{DATASET_PATH}' не знайдено.")
else:
    df = pd.read_csv(DATASET_PATH)
```

Рисунок Б.1.1 - Код створення та аналізу вибірок

```python
else:
    df = pd.read_csv(DATASET_PATH)
    # Поглиблений аналіз
    eye_labels = df[PRIMARY_PATHOLOGY_COLS]
    co_occurrence = {}
    for col1, col2 in combinations(PRIMARY_PATHOLOGY_COLS, 2):
        count = ((eye_labels[col1] > 0) & (eye_labels[col2] > 0)).sum()
        if count > 0:
            co_occurrence[f'{col1}+{col2}'] = count
    rarest_combo_str = min(co_occurrence, key=co_occurrence.get) if co_occurrence else None
    rarest_combo_cols = rarest_combo_str.split('+') if rarest_combo_str else []
    patient_analysis = []
    for patient_id, group in df.groupby('ID'):
        num_eyes = len(group)
        labels_sum = group[LABEL_COLS].sum(axis=0)
        eye_label_counts = group[LABEL_COLS].sum(axis=1).values
        analysis = {'ID': patient_id}
        analysis['is_one_eyed'] = num_eyes == 1
        analysis['is_pure_normal'] = (labels_sum['N'] == num_eyes) and (labels_sum.drop('N').sum() == 0)
        analysis['max_labels_on_eye'] = eye_label_counts.max() if len(eye_label_counts) > 0 else 0
        analysis['one_eyed_with_multiple_pathologies'] = analysis['is_one_eyed'] and analysis['max_labels_on_eye'] >= 2
        has_pathology_plus_other, has_normal_plus_other, has_rarest_combo_on_eye = False, False, False
        for _, eye_row in group.iterrows():
            if eye_row[PRIMARY_PATHOLOGY_COLS].sum() > 0 and eye_row['O'] > 0: has_pathology_plus_other = True
            if eye_row['N'] > 0 and eye_row['O'] > 0: has_normal_plus_other = True
            if rarest_combo_cols and eye_row[rarest_combo_cols[0]] > 0 and eye_row[rarest_combo_cols[1]] > 0: has_rarest_combo_on_eye = True
        analysis['has_pathology_plus_other_on_eye'] = has_pathology_plus_other
        analysis['has_normal_plus_other_on_eye'] = has_normal_plus_other
        analysis['has_rarest_combo_on_eye'] = has_rarest_combo_on_eye
        if num_eyes == 2:
            eye1_labels, eye2_labels = group.iloc[0][LABEL_COLS], group.iloc[1][LABEL_COLS]
            is_eye1_pure_normal, is_eye2_pure_normal = (eye1_labels['N'] == 1 and eye1_labels.drop('N').sum() == 0), (eye2_labels['N'] == 1 and eye2_labels.drop('N').sum() == 0)
            is_eye1_pure_other, is_eye2_pure_other = (eye1_labels['O'] == 1 and eye1_labels.drop('N').sum() == 0), (eye2_labels['O'] == 1 and eye2_labels.drop('N').sum() == 0)
            analysis['is_mixed_norm_pathology'] = (is_eye1_pure_normal and not is_eye2_pure_normal) or (is_eye2_pure_normal and not is_eye1_pure_normal)
            analysis['has_different_pathologies'] = not (eye1_labels.equals(eye2_labels)) and not is_eye1_pure_normal and not is_eye2_pure_normal
            analysis['is_mixed_norm_other'] = (is_eye1_pure_normal and is_eye2_pure_other) or (is_eye2_pure_normal and is_eye1_pure_other)
        else:
            analysis['is_mixed_norm_pathology'] = analysis['has_different_pathologies'] = analysis['is_mixed_norm_other'] = False
        patient_analysis.append(analysis)
    patients_df = pd.DataFrame(patient_analysis)
```

Рисунок Б.1.2 - Код створення та аналізу вибірок

```python
    # Формування тестової вибірки
    print("\nВідбір унікальних випадків для тестової вибірки:")
    test_patient_ids = []
    test_selection_details = OrderedDict()
    criteria = OrderedDict([
        ("Пацієнт: Одноокий (>=2 патології на оці)", lambda r: r['one_eyed_with_multiple_pathologies']),
        ("Пацієнт: Різні патології на очах", lambda r: r['has_different_pathologies']),
        ("Пацієнт: Норма на одному оці, 'Інше' на другому", lambda r: r['is_mixed_norm_other']),
        ("Пацієнт: Норма на одному оці, патологія на другому", lambda r: r['is_mixed_norm_pathology']),
        ("Пацієнт: Повна норма (всі очі)", lambda r: r['is_pure_normal']),
        ("Пацієнт: Тільки одне око в датасеті", lambda r: r['is_one_eyed']),
        (f"Око: Найрідкісніша комбінація ({rarest_combo_str})", lambda r: r['has_rarest_combo_on_eye']),
        ("Око: Патологія + 'Інше'", lambda r: r['has_pathology_plus_other_on_eye']),
        ("Око: Норма + 'Інше' (техн. артефакт)", lambda r: r['has_normal_plus_other_on_eye']),
        ("Око: 3 патології", lambda r: r['max_labels_on_eye'] == 3),
        ("Око: 2 патології", lambda r: r['max_labels_on_eye'] == 2),
    ])
    for reason, criterion in criteria.items():
        candidates = patients_df[patients_df.apply(criterion, axis=1) & ~patients_df['ID'].isin(test_patient_ids)]
        if not candidates.empty:
            selected_ids = candidates.head(N_SAMPLES_PER_CRITERION)['ID'].tolist()
            test_patient_ids.extend(selected_ids)
            test_selection_details[reason] = selected_ids
    current_test_df = df[df['ID'].isin(test_patient_ids)]
    missing_classes = current_test_df[LABEL_COLS].sum()[lambda x: x == 0].index.tolist()
    if missing_classes:
        for missing_class in missing_classes:
            candidate_df = df[(df[missing_class] > 0) & (~df['ID'].isin(test_patient_ids))]
            if not candidate_df.empty:
                patient_id = candidate_df.iloc[0]['ID']
                test_patient_ids.append(patient_id)
                reason = f"Додатково для покриття класу '{missing_class}'"
                if reason not in test_selection_details: test_selection_details[reason] = []
                test_selection_details[reason].append(patient_id)
    for reason, ids in test_selection_details.items():
        print(f"  - {reason}: додано {len(ids)} пацієнт(ів) (ID: {ids})")

```

Рисунок Б.1.3 - Код створення та аналізу вибірок

```python
    # Розділення решти пацієнтів
    test_patient_ids = sorted(list(set(test_patient_ids)))
    remaining_ids = patients_df[~patients_df['ID'].isin(test_patient_ids)]['ID'].values
    train_ids, val_ids = train_test_split(remaining_ids, test_size=VAL_SIZE, random_state=RANDOM_STATE)

    # Формування фінального датасету з розподілом
    df['split'] = 'train'
    df.loc[df['ID'].isin(val_ids), 'split'] = 'val'
    df.loc[df['ID'].isin(test_patient_ids), 'split'] = 'test'

    # Перевірка
    print("\nПеревірка цілісності розподілу")
    leakage_check = df.groupby('ID')['split'].nunique()
    if (leakage_check > 1).any(): print("ПОМИЛКА: ЗНАЙДЕНО ВИТІК ДАНИХ!")
    else:
        print("Перевірка успішна. Приклади пацієнтів з двома очима з кожної вибірки:")
        for split_name, ids_list in [('train', train_ids), ('val', val_ids), ('test', test_patient_ids)]:
            sample_df = df[(df['ID'].isin(ids_list)) & (df.duplicated(subset=['ID'], keep=False))]
            if not sample_df.empty:
                sample_id = sample_df.iloc[0]['ID']
                print(f"\nПриклад для вибірки '{split_name}':")
                print(df[df['ID'] == sample_id][['ID', 'filename', 'split']].to_string(index=False))

    # Статистика
    stats_blocks = []
    total_patients, total_images = df['ID'].nunique(), len(df)
    for split_name in ['train', 'val', 'test']:
        split_df = df[df['split'] == split_name]
        num_patients, num_images = split_df['ID'].nunique(), len(split_df)
        patient_perc, image_perc = round((num_patients / total_patients * 100), 2), round((num_images / total_images * 100), 2)
        block_str = f"Вибірка: {split_name.upper()}\n"
        block_str += f"  - Кількість пацієєнтів: {num_patients} ({patient_perc}%)\n"
        block_str += f"  - Кількість зображень: {num_images} ({image_perc}%)\n"
        if split_name == 'test':
            id_line = f"  - ID пацієнтів у вибірці: {test_patient_ids}"
            # Обгортаємо рядок, якщо він занадто довгий для стовпчика
            wrapped_lines = textwrap.wrap(id_line, width=COLUMN_WIDTH - 1)
            block_str += "\n".join(wrapped_lines) + "\n"
        block_str += "  - Розподіл класів (по зображеннях):\n"
        if num_images > 0:
            class_distribution = split_df[LABEL_COLS].sum().reset_index()
            class_distribution.columns = ['Клас', 'Кількість']
            class_distribution['Відсоток (%)'] = (class_distribution['Кількість'] / num_images * 100).round(2)
            block_str += class_distribution.to_string(index=False)
        else:
            block_str += "  Вибірка порожня."
        stats_blocks.append(block_str)

    print("\nСтатистика по вибірках")
    print_side_by_side(stats_blocks, col_width=COLUMN_WIDTH)

    df.to_csv(OUTPUT_PATH, index=False)
    print(f"\nДатасет з розподілом збережено у файл: {OUTPUT_PATH}")
```

Рисунок Б.1.4 - Код створення та аналізу вибірок

ДОДАТОК В - Реалізація та навчання базової моделі

```python
# Налаштування логування та імпорти
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, RandomFlip, RandomRotation, RandomZoom, RandomContrast
from tensorflow.keras.callbacks import CSVLogger, EarlyStopping, Callback
from tensorflow.keras.optimizers import Adam
import time
import psutil
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, precision_recall_curve, auc, f1_score, precision_score, recall_score, accuracy_score, multilabel_confusion_matrix
import seaborn as sns
import shutil
import cv2

# Моніторинг GPU
try:
    import pynvml
    pynvml.nvmlInit()
    GPU_MONITORING_ENABLED = True
except (ImportError, pynvml.NVMLError):
    GPU_MONITORING_ENABLED = False

# Налаштування
RESIZE_IMAGES = True
TARGET_IMG_SIZE = 256
BATCH_SIZE = 32
USE_EARLY_STOPPING = True
PATIENCE = 10
EPOCHS = 1000 if USE_EARLY_STOPPING else 100
LEARNING_RATE = 0.0001
GRADCAM_IMAGE_COUNT = 30

# Глобальні константи
DATA_SPLITS_FILE = 'final_dataset_with_splits.csv'
IMAGES_DIR = 'archive (1)/preprocessed_images'
LABEL_COLS = ['N', 'D', 'G', 'C', 'A', 'H', 'M', 'O']
RESULTS_DIR = "training_results_base_model"
MODEL_SAVE_PATH = os.path.join(RESULTS_DIR, "best_model.keras")
TRAINING_LOG_FILE = os.path.join(RESULTS_DIR, "training_log.csv")
TEST_REPORT_FILE = os.path.join(RESULTS_DIR, "test_report.txt")
GRADCAM_DIR = os.path.join(RESULTS_DIR, "gradcam_visualizations")

```

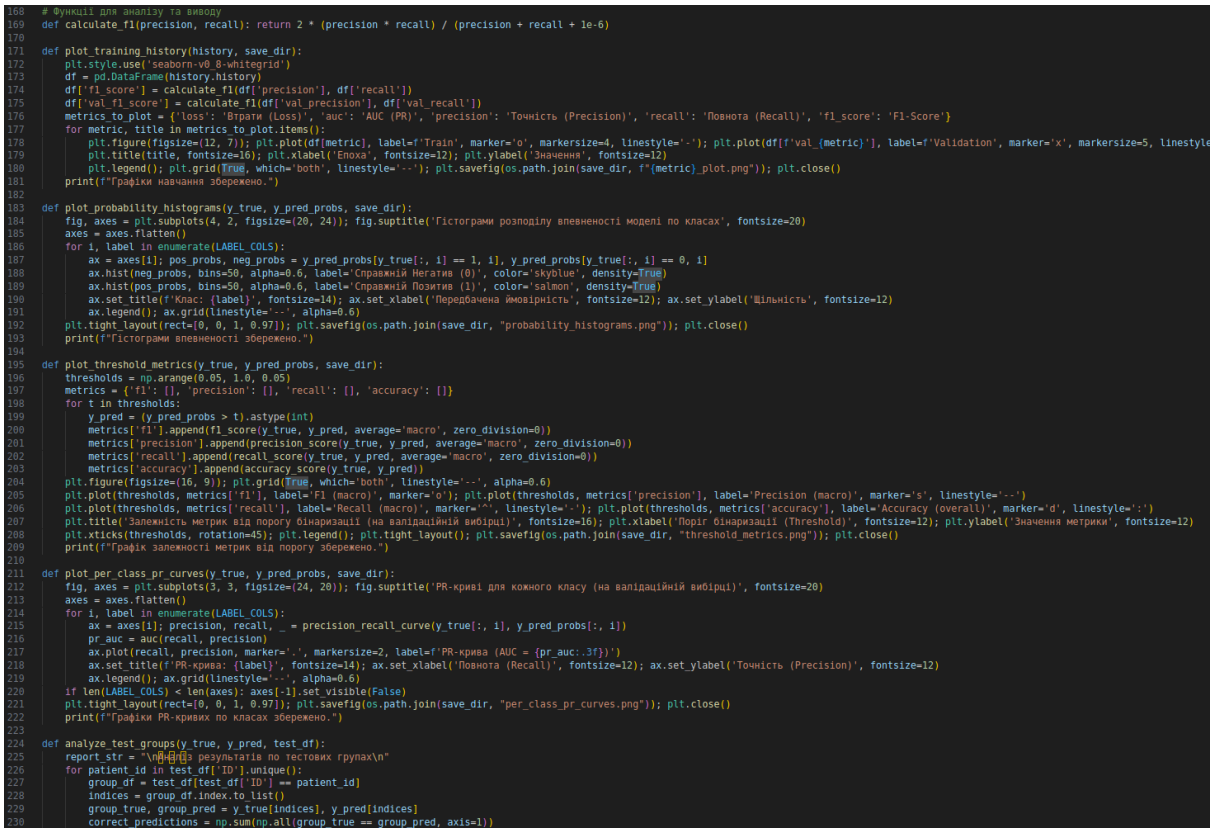
Рисунок В.1 - Код налаштування глобальних параметрів та імпорт необхідних інструментів

```python
# Функції-помічники
class PerformanceMonitor(Callback):
    def on_train_begin(self, logs=None):
        self.train_start_time = time.time()
        self.history = {'cpu': [], 'ram': [], 'gpu_mem_mb': []}
        self.max_gpu_mem = 0
        print("Початок моніторингу ресурсів...")
    def on_epoch_begin(self, epoch, logs=None): self.epoch_start_time = time.time()
    def on_epoch_end(self, epoch, logs=None):
        self.history['cpu'].append(psutil.cpu_percent())
        self.history['ram'].append(psutil.virtual_memory().percent)
        if GPU_MONITORING_ENABLED:
            handle = pynvml.nvmlDeviceGetHandleByIndex(0)
            gpu_info = pynvml.nvmlDeviceGetMemoryInfo(handle)
            used_mem_mb = gpu_info.used // 1024**2
            self.history['gpu_mem_mb'].append(used_mem_mb)
            self.max_gpu_mem = max(self.max_gpu_mem, used_mem_mb)
        print(f"\n□□□□ {epoch + 1} завершена за {time.time() - self.epoch_start_time:.2f} □. ⏳")
    def on_train_end(self, logs=None):
        mins, secs = divmod(time.time() - self.train_start_time, 60)
        print(f"\n✅ Загальний час навчання: {int(mins)} хвилин(□) {secs:.0f} секунд(и).")

class CustomModelCheckpoint(Callback):
    def __init__(self, filepath, monitor='val_loss', mode='auto'):
        super(CustomModelCheckpoint, self).__init__()
        self.monitor, self.filepath = monitor, filepath
        self.monitor_op = np.greater if mode == 'max' else np.less
        self.best = -np.inf if mode == 'max' else np.inf
    def on_epoch_end(self, epoch, logs=None):
        current = logs.get(self.monitor)
        if current is None: return
        if self.monitor_op(current, self.best):
            print(f"\n📈 Епоха {epoch + 1}: {self.monitor} покращився з {self.best:.5f} до {current:.5f}, зберігаємо модель.")
            self.best = current
            self.model.save(self.filepath, overwrite=True)

def create_dataset(df, image_dir, label_cols, target_size, is_training=False):
    dataset = tf.data.Dataset.from_tensor_slices((df['filename'].apply(lambda x: os.path.join(image_dir, x)).values, df[label_cols].values.astype('float32')))
    def parse_image(filepath, labels):
        image = tf.io.read_file(filepath)
        image = tf.image.decode_jpeg(image, channels=3)
        current_size = 512 if not RESIZE_IMAGES else target_size
        image = tf.image.resize(image, [current_size, current_size])
        return tf.image.convert_image_dtype(image, tf.float32), labels
    dataset = dataset.map(parse_image, num_parallel_calls=tf.data.AUTOTUNE)
    if is_training: dataset = dataset.shuffle(buffer_size=len(df))
    return dataset.batch(BATCH_SIZE).prefetch(buffer_size=tf.data.AUTOTUNE)

def calculate_class_weights(df):
    num_samples, num_classes = len(df), len(LABEL_COLS)
    pos_counts = df[LABEL_COLS].sum()
    weights = {i: (num_samples) / (num_classes * pos_counts[col] + 1e-6) for i, col in enumerate(LABEL_COLS)}
    print("\n□□□□□□□□□ ваги для класів:"); [print(f"  - Клас '{col}': {weights[i]:.2f}") for i, col in enumerate(LABEL_COLS)]
    return weights

```

Рисунок В.2 - Код визначення набору функцій-помічників та кастомних класів

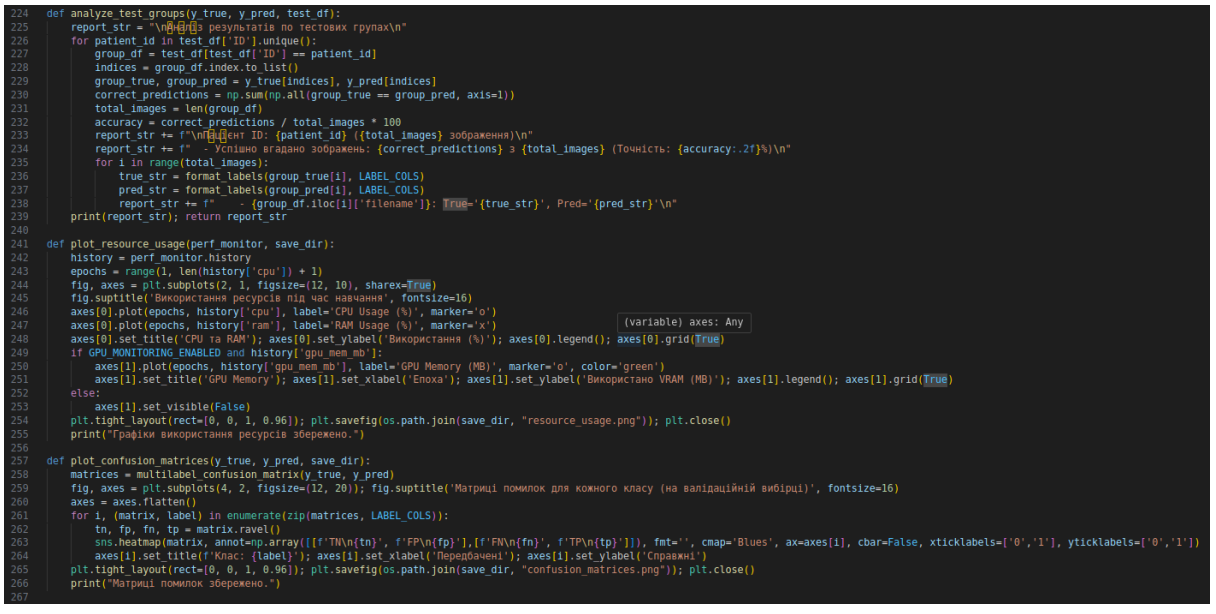
```python
# Побудова моделі та Grad-CAM
def build_model(input_shape, num_classes):
    data_augmentation = tf.keras.Sequential([RandomFlip("horizontal"), RandomRotation(0.1), RandomZoom(0.1), RandomContrast(0.1)], name="augmentation")
    inputs = Input(shape=input_shape)
    x = data_augmentation(inputs)
    x = Conv2D(32, (3, 3), activation='relu', padding='same', name='conv2d_1')(x)
    x = MaxPooling2D((2, 2))(x)
    x = Conv2D(64, (3, 3), activation='relu', padding='same', name='conv2d_2')(x)
    x = MaxPooling2D((2, 2))(x)
    x = Conv2D(128, (3, 3), activation='relu', padding='same', name="conv2d_3")(x)
    x = MaxPooling2D((2, 2))(x)
    x = Flatten()(x)
    x = Dense(128, activation='relu')(x)
    outputs = Dense(num_classes, activation='sigmoid', name='predictions')(x)
    return Model(inputs=inputs, outputs=outputs)

def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_index=None):
    grad_model = Model(model.inputs, [model.get_layer(last_conv_layer_name).output, model.output])
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None: pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]
    grads = tape.gradient(class_channel, last_conv_layer_output)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    heatmap = last_conv_layer_output[0] @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)
    return tf.maximum(heatmap, 0) / (tf.math.reduce_max(heatmap) + 1e-8)

def format_labels(labels_vector, all_labels):
    present_labels = [all_labels[i] for i, val in enumerate(labels_vector) if val == 1]
    return ", ".join(present_labels) if present_labels else "No prediction"

def save_gradcam_with_info(img_path, heatmap, save_path, patient_id, true_labels, pred_labels):
    img = cv2.imread(img_path)
    if img is None: print(f"  - ПОМИЛКА Grad-CAM: не вдалося завантажити зображення: {img_path}"); return
    img = cv2.resize(img, (TARGET_IMG_SIZE, TARGET_IMG_SIZE))
    heatmap = np.uint8(255 * cv2.resize(heatmap, (img.shape[1], img.shape[0])))
    heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
    superimposed_img = cv2.addWeighted(img, 0.6, heatmap, 0.4, 0)
    font, scale, color, thickness = cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1
    cv2.putText(superimposed_img, f"Patient ID: {patient_id}", (5, 15), font, scale, color, thickness)
    cv2.putText(superimposed_img, f"True: {true_labels}", (5, 30), font, scale, color, thickness)
    cv2.putText(superimposed_img, f"Pred: {pred_labels}", (5, 45), font, scale, color, thickness)
    cv2.imwrite(save_path, superimposed_img)

def generate_gradcam_visualizations(model, test_df, image_dir, save_dir, num_images):
    print("\nГенерація візуалізацій Grad-CAM...")
    vis_model = Model(inputs=model.input, outputs=model.output)
    for index, row in test_df.head(num_images).iterrows():
        img_path = os.path.join(image_dir, row['filename'])
        if not os.path.exists(img_path):
            print(f"  - ПОМИЛКА Grad-CAM: файл не знайдено: {img_path}"); continue

        img_tensor, true_labels_vec = next(iter(create_dataset(row.to_frame().T, image_dir, LABEL_COLS, TARGET_IMG_SIZE)))
        pred_probs = vis_model.predict(img_tensor, verbose=0)
        pred_labels_vec = (pred_probs > 0.5).astype(int)[0]
        pred_index = np.argmax(pred_probs[0]) if np.any(pred_labels_vec) else np.argmax(true_labels_vec.numpy()[0])
        heatmap = make_gradcam_heatmap(img_tensor, vis_model, "conv2d_3", pred_index=pred_index)
        true_labels_str = format_labels(true_labels_vec.numpy()[0], LABEL_COLS)
        pred_labels_str = format_labels(pred_labels_vec, LABEL_COLS)

        save_gradcam_with_info(img_path, heatmap.numpy(), os.path.join(save_dir, f"gradcam_{row['filename']}"),
                               row['ID'], true_labels_str, pred_labels_str)
        print(f"  - Збережено візуалізацію для: {row['filename']}")
```

Рисунок В.3 - Код визначення архітектури нейронної мережі та реалізація механізму Grad-CAM



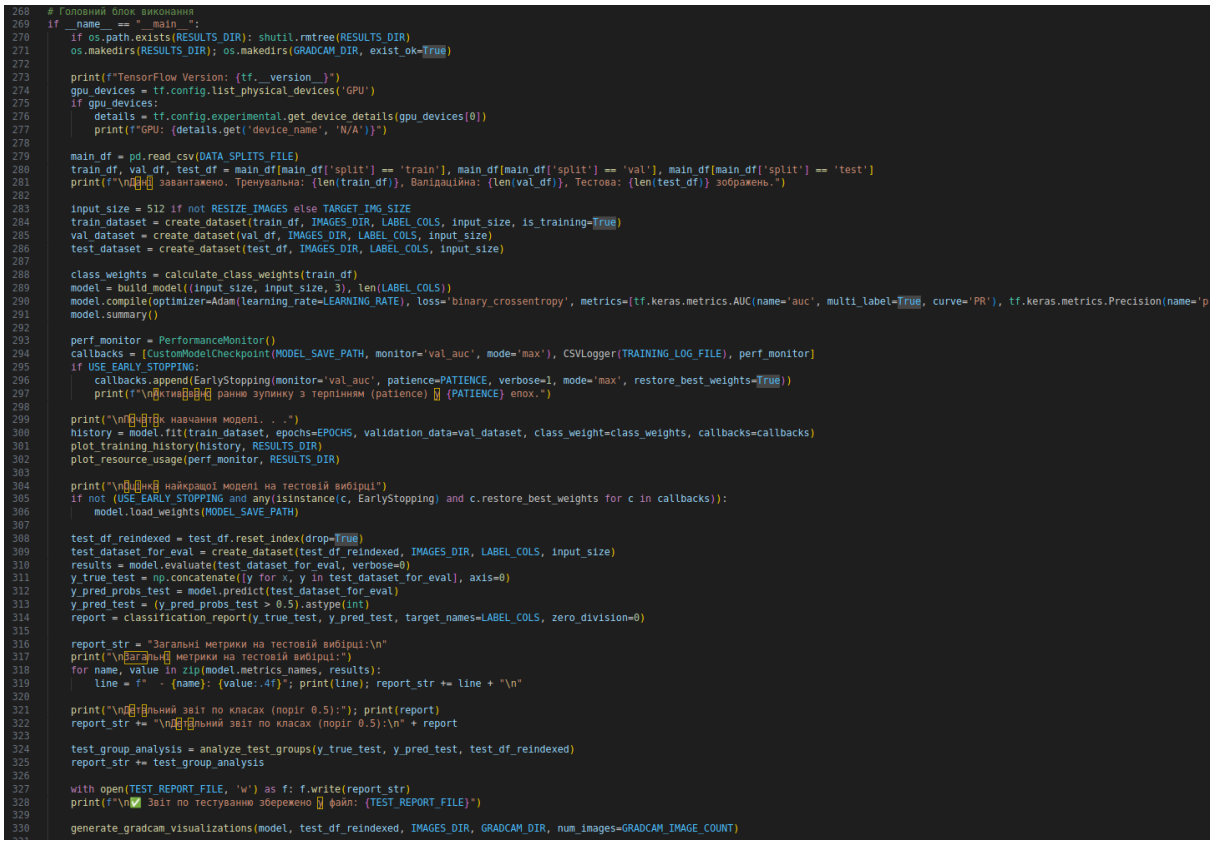
```python
# Функції для аналізу та виводу
def calculate_f1(precision, recall): return 2 * (precision * recall) / (precision + recall + 1e-6)

def plot_training_history(history, save_dir):
    plt.style.use('seaborn-v0_8-whitegrid')
    df = pd.DataFrame(history.history)
    df['f1_score'] = calculate_f1(df['precision'], df['recall'])
    df['val_f1_score'] = calculate_f1(df['val_precision'], df['val_recall'])
    metrics_to_plot = {'loss': 'Втрати (Loss)', 'auc': 'AUC (PR)', 'precision': 'Точність (Precision)', 'recall': 'Повнота (Recall)', 'f1_score': 'F1-Score'}
    for metric, title in metrics_to_plot.items():
        plt.figure(figsize=(12, 7)); plt.plot(df[metric], label=f'Train', marker='o', markersize=4, linestyle='-'); plt.plot(df[f'val_{metric}'], label=f'Validation', marker='x', markersize=5, linestyle
        plt.title(title, fontsize=16); plt.xlabel('Епоха', fontsize=12); plt.ylabel('Значення', fontsize=12)
        plt.legend(); plt.grid(True, which='both', linestyle='--'); plt.savefig(os.path.join(save_dir, f"{metric}_plot.png")); plt.close()
    print(f"Графіки навчання збережено.")

def plot_probability_histograms(y_true, y_pred_probs, save_dir):
    fig, axes = plt.subplots(4, 2, figsize=(20, 24)); fig.suptitle('Гістограми розподілу впевненості моделі по класах', fontsize=20)
    axes = axes.flatten()
    for i, label in enumerate(LABEL_COLS):
        ax = axes[i]; pos_probs, neg_probs = y_pred_probs[y_true[:, i] == 1, i], y_pred_probs[y_true[:, i] == 0, i]
        ax.hist(neg_probs, bins=50, alpha=0.6, label='Справжній Негатив (0)', color='skyblue', density=True)
        ax.hist(pos_probs, bins=50, alpha=0.6, label='Справжній Позитив (1)', color='salmon', density=True)
        ax.set_title(f'Клас: {label}', fontsize=14); ax.set_xlabel('Передбачена ймовірність', fontsize=12); ax.set_ylabel('Щільність', fontsize=12)
        ax.legend(); ax.grid(linestyle='--', alpha=0.6)
    plt.tight_layout(rect=[0, 0, 1, 0.97]); plt.savefig(os.path.join(save_dir, "probability_histograms.png")); plt.close()
    print(f"Гістограми впевненості збережено.")

def plot_threshold_metrics(y_true, y_pred_probs, save_dir):
    thresholds = np.arange(0.05, 1.0, 0.05)
    metrics = {'f1': [], 'precision': [], 'recall': [], 'accuracy': []}
    for t in thresholds:
        y_pred = (y_pred_probs > t).astype(int)
        metrics['f1'].append(f1_score(y_true, y_pred, average='macro', zero_division=0))
        metrics['precision'].append(precision_score(y_true, y_pred, average='macro', zero_division=0))
        metrics['recall'].append(recall_score(y_true, y_pred, average='macro', zero_division=0))
        metrics['accuracy'].append(accuracy_score(y_true, y_pred))
    plt.figure(figsize=(16, 9)); plt.grid(True, which='both', linestyle='--', alpha=0.6)
    plt.plot(thresholds, metrics['f1'], label='F1 (macro)', marker='o'); plt.plot(thresholds, metrics['precision'], label='Precision (macro)', marker='s', linestyle='--')
    plt.plot(thresholds, metrics['recall'], label='Recall (macro)', marker='^', linestyle='--'); plt.plot(thresholds, metrics['accuracy'], label='Accuracy (overall)', marker='d', linestyle=':')
    plt.title('Залежність метрик від порогу бінаризації (на валідаційній вибірці)', fontsize=16); plt.xlabel('Поріг бінаризації (Threshold)', fontsize=12); plt.ylabel('Значення метрики', fontsize=12)
    plt.xticks(thresholds, rotation=45); plt.legend(); plt.tight_layout(); plt.savefig(os.path.join(save_dir, "threshold_metrics.png")); plt.close()
    print(f"Графік залежності метрик від порогу збережено.")

def plot_per_class_pr_curves(y_true, y_pred_probs, save_dir):
    fig, axes = plt.subplots(3, 3, figsize=(24, 20)); fig.suptitle('PR-криві для кожного класу (на валідаційній вибірці)', fontsize=20)
    axes = axes.flatten()
    for i, label in enumerate(LABEL_COLS):
        ax = axes[i]; precision, recall, _ = precision_recall_curve(y_true[:, i], y_pred_probs[:, i])
        pr_auc = auc(recall, precision)
        ax.plot(recall, precision, marker='.', markersize=2, label=f'PR-крива (AUC = {pr_auc:.3f})')
        ax.set_title(f'PR-крива: {label}', fontsize=14); ax.set_xlabel('Повнота (Recall)', fontsize=12); ax.set_ylabel('Точність (Precision)', fontsize=12)
        ax.legend(); ax.grid(linestyle='--', alpha=0.6)
    if len(LABEL_COLS) < len(axes): axes[-1].set_visible(False)
    plt.tight_layout(rect=[0, 0, 1, 0.97]); plt.savefig(os.path.join(save_dir, "per_class_pr_curves.png")); plt.close()
    print(f"Графіки PR-кривих по класах збережено.")

def analyze_test_groups(y_true, y_pred, test_df):
    report_str = "\n[illegible] результатів по тестових групах\n"
    for patient_id in test_df['ID'].unique():
        group_df = test_df[test_df['ID'] == patient_id]
        indices = group_df.index.to_list()
        group_true, group_pred = y_true[indices], y_pred[indices]
        correct_predictions = np.sum(np.all(group_true == group_pred, axis=1))
```

Рисунок В.4.1 - Код визначення набіру функцій, призначених для всебічного аналізу та візуалізації результатів навчання моделі



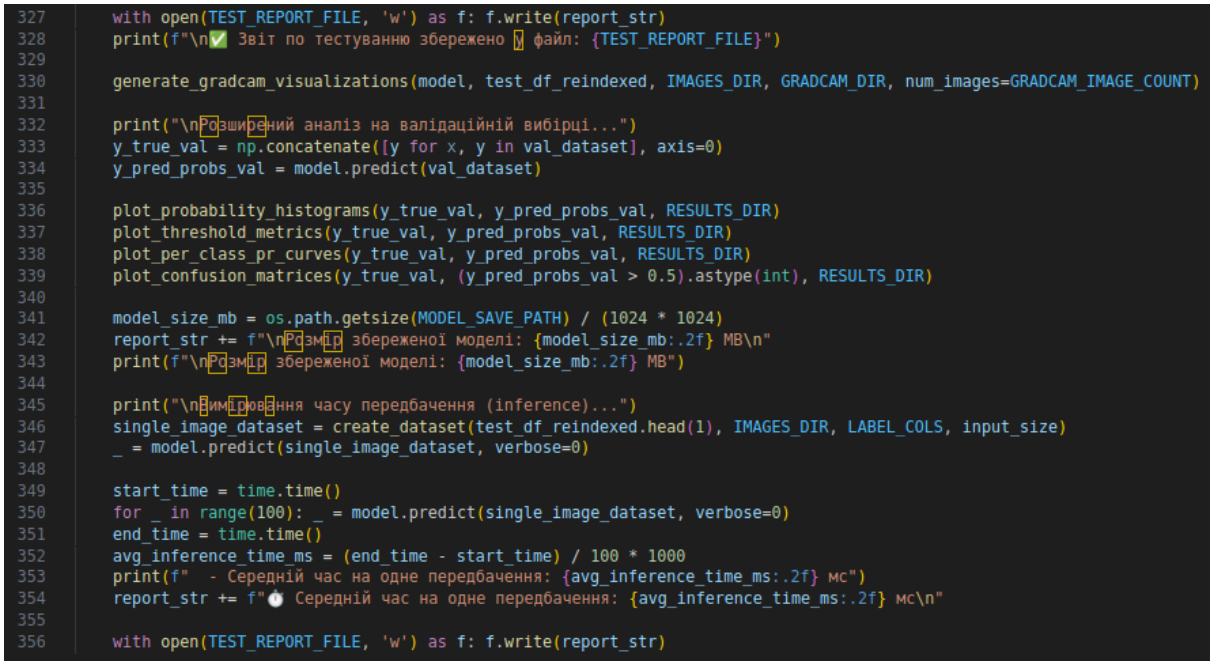
```python
def analyze_test_groups(y_true, y_pred, test_df):
    report_str = "\n[illegible] результатів по тестових групах\n"
    for patient_id in test_df['ID'].unique():
        group_df = test_df[test_df['ID'] == patient_id]
        indices = group_df.index.to_list()
        group_true, group_pred = y_true[indices], y_pred[indices]
        correct_predictions = np.sum(np.all(group_true == group_pred, axis=1))
        total_images = len(group_df)
        accuracy = correct_predictions / total_images * 100
        report_str += f"\n[illegible]ент ID: {patient_id} ({total_images} зображення)\n"
        report_str += f"  - Успішно вгадано зображень: {correct_predictions} з {total_images} (Точність: {accuracy:.2f}%)\n"
        for i in range(total_images):
            true_str = format_labels(group_true[i], LABEL_COLS)
            pred_str = format_labels(group_pred[i], LABEL_COLS)
            report_str += f"    - {group_df.iloc[i]['filename']}: True='{true_str}', Pred='{pred_str}'\n"
    print(report_str); return report_str

def plot_resource_usage(perf_monitor, save_dir):
    history = perf_monitor.history
    epochs = range(1, len(history['cpu']) + 1)
    fig, axes = plt.subplots(2, 1, figsize=(12, 10), sharex=True)
    fig.suptitle('Використання ресурсів під час навчання', fontsize=16)
    axes[0].plot(epochs, history['cpu'], label='CPU Usage (%)', marker='o')
    axes[0].plot(epochs, history['ram'], label='RAM Usage (%)', marker='x')
    axes[0].set_title('CPU та RAM'); axes[0].set_ylabel('Використання (%)'); axes[0].legend(); axes[0].grid(True)
    if GPU_MONITORING_ENABLED and history['gpu_mem_mb']:
        axes[1].plot(epochs, history['gpu_mem_mb'], label='GPU Memory (MB)', marker='o', color='green')
        axes[1].set_title('GPU Memory'); axes[1].set_xlabel('Епоха'); axes[1].set_ylabel('Використано VRAM (MB)'); axes[1].legend(); axes[1].grid(True)
    else:
        axes[1].set_visible(False)
    plt.tight_layout(rect=[0, 0, 1, 0.96]); plt.savefig(os.path.join(save_dir, "resource_usage.png")); plt.close()
    print("Графіки використання ресурсів збережено.")

def plot_confusion_matrices(y_true, y_pred, save_dir):
    matrices = multilabel_confusion_matrix(y_true, y_pred)
    fig, axes = plt.subplots(4, 2, figsize=(12, 20)); fig.suptitle('Матриці помилок для кожного класу (на валідаційній вибірці)', fontsize=16)
    axes = axes.flatten()
    for i, (matrix, label) in enumerate(zip(matrices, LABEL_COLS)):
        tn, fp, fn, tp = matrix.ravel()
        sns.heatmap(matrix, annot=np.array([[f'TN\n{tn}', f'FP\n{fp}'],[f'FN\n{fn}', f'TP\n{tp}']]), fmt='', cmap='Blues', ax=axes[i], cbar=False, xticklabels=['0','1'], yticklabels=['0','1'])
        axes[i].set_title(f'Клас: {label}'); axes[i].set_xlabel('Передбачені'); axes[i].set_ylabel('Справжні')
    plt.tight_layout(rect=[0, 0, 1, 0.96]); plt.savefig(os.path.join(save_dir, "confusion_matrices.png")); plt.close()
    print("Матриці помилок збережено.")
```

Рисунок В.4.2 - Код визначення набіру функцій, призначених для всебічного аналізу та візуалізації результатів навчання моделі



```
# Головний блок виконання
if __name__ == "__main__":
    if os.path.exists(RESULTS_DIR): shutil.rmtree(RESULTS_DIR)
    os.makedirs(RESULTS_DIR); os.makedirs(GRADCAM_DIR, exist_ok=True)

    print(f"TensorFlow Version: {tf.__version__}")
    gpu_devices = tf.config.list_physical_devices('GPU')
    if gpu_devices:
        details = tf.config.experimental.get_device_details(gpu_devices[0])
        print(f"GPU: {details.get('device_name', 'N/A')}")

    main_df = pd.read_csv(DATA_SPLITS_FILE)
    train_df, val_df, test_df = main_df[main_df['split'] == 'train'], main_df[main_df['split'] == 'val'], main_df[main_df['split'] == 'test']
    print(f"\nДані завантажено. Тренувальна: {len(train_df)}, Валідаційна: {len(val_df)}, Тестова: {len(test_df)} зображень.")

    input_size = 512 if not RESIZE_IMAGES else TARGET_IMG_SIZE
    train_dataset = create_dataset(train_df, IMAGES_DIR, LABEL_COLS, input_size, is_training=True)
    val_dataset = create_dataset(val_df, IMAGES_DIR, LABEL_COLS, input_size)
    test_dataset = create_dataset(test_df, IMAGES_DIR, LABEL_COLS, input_size)

    class_weights = calculate_class_weights(train_df)
    model = build_model((input_size, input_size, 3), len(LABEL_COLS))
    model.compile(optimizer=Adam(learning_rate=LEARNING_RATE), loss='binary_crossentropy', metrics=[tf.keras.metrics.AUC(name='auc', multi_label=True, curve='PR'), tf.keras.metrics.Precision(name='p
    model.summary()

    perf_monitor = PerformanceMonitor()
    callbacks = [CustomModelCheckpoint(MODEL_SAVE_PATH, monitor='val_auc', mode='max'), CSVLogger(TRAINING_LOG_FILE), perf_monitor]
    if USE_EARLY_STOPPING:
        callbacks.append(EarlyStopping(monitor='val_auc', patience=PATIENCE, verbose=1, mode='max', restore_best_weights=True))
        print(f"\nАктивовано ранню зупинку з терпінням (patience) у {PATIENCE} епох.")

    print("\nПочаток навчання моделі. . .")
    history = model.fit(train_dataset, epochs=EPOCHS, validation_data=val_dataset, class_weight=class_weights, callbacks=callbacks)
    plot_training_history(history, RESULTS_DIR)
    plot_resource_usage(perf_monitor, RESULTS_DIR)

    print("\nЗавантаження найкращої моделі на тестовій вибірці")
    if not (USE_EARLY_STOPPING and any(isinstance(c, EarlyStopping) and c.restore_best_weights for c in callbacks)):
        model.load_weights(MODEL_SAVE_PATH)

    test_df_reindexed = test_df.reset_index(drop=True)
    test_dataset_for_eval = create_dataset(test_df_reindexed, IMAGES_DIR, LABEL_COLS, input_size)
    results = model.evaluate(test_dataset_for_eval, verbose=0)
    y_true_test = np.concatenate([y for x, y in test_dataset_for_eval], axis=0)
    y_pred_probs_test = model.predict(test_dataset_for_eval)
    y_pred_test = (y_pred_probs_test > 0.5).astype(int)
    report = classification_report(y_true_test, y_pred_test, target_names=LABEL_COLS, zero_division=0)

    report_str = "Загальні метрики на тестовій вибірці:\n"
    print("\nЗагальні метрики на тестовій вибірці:")
    for name, value in zip(model.metrics_names, results):
        line = f"  - {name}: {value:.4f}"; print(line); report_str += line + "\n"

    print("\nДетальний звіт по класах (поріг 0.5):"); print(report)
    report_str += "\nДетальний звіт по класах (поріг 0.5):\n" + report

    test_group_analysis = analyze_test_groups(y_true_test, y_pred_test, test_df_reindexed)
    report_str += test_group_analysis

    with open(TEST_REPORT_FILE, 'w') as f: f.write(report_str)
    print(f"\n✅ Звіт по тестуванню збережено у файл: {TEST_REPORT_FILE}")

    generate_gradcam_visualizations(model, test_df_reindexed, IMAGES_DIR, GRADCAM_DIR, num_images=GRADCAM_IMAGE_COUNT)
```

Рисунок В.5.1 - Код точки входу в програму



```
    with open(TEST_REPORT_FILE, 'w') as f: f.write(report_str)
    print(f"\n✅ Звіт по тестуванню збережено у файл: {TEST_REPORT_FILE}")

    generate_gradcam_visualizations(model, test_df_reindexed, IMAGES_DIR, GRADCAM_DIR, num_images=GRADCAM_IMAGE_COUNT)

    print("\nРозширений аналіз на валідаційній вибірці...")
    y_true_val = np.concatenate([y for x, y in val_dataset], axis=0)
    y_pred_probs_val = model.predict(val_dataset)

    plot_probability_histograms(y_true_val, y_pred_probs_val, RESULTS_DIR)
    plot_threshold_metrics(y_true_val, y_pred_probs_val, RESULTS_DIR)
    plot_per_class_pr_curves(y_true_val, y_pred_probs_val, RESULTS_DIR)
    plot_confusion_matrices(y_true_val, (y_pred_probs_val > 0.5).astype(int), RESULTS_DIR)

    model_size_mb = os.path.getsize(MODEL_SAVE_PATH) / (1024 * 1024)
    report_str += f"\nРозмір збереженої моделі: {model_size_mb:.2f} MB\n"
    print(f"\nРозмір збереженої моделі: {model_size_mb:.2f} MB")

    print("\nВимірювання часу передбачення (inference)...")
    single_image_dataset = create_dataset(test_df_reindexed.head(1), IMAGES_DIR, LABEL_COLS, input_size)
    _ = model.predict(single_image_dataset, verbose=0)

    start_time = time.time()
    for _ in range(100): _ = model.predict(single_image_dataset, verbose=0)
    end_time = time.time()
    avg_inference_time_ms = (end_time - start_time) / 100 * 1000
    print(f"  - Середній час на одне передбачення: {avg_inference_time_ms:.2f} мс")
    report_str += f"⏱️ Середній час на одне передбачення: {avg_inference_time_ms:.2f} мс\n"

    with open(TEST_REPORT_FILE, 'w') as f: f.write(report_str)
```

Рисунок В.5.2 - Код точки входу в програму

ДОДАТОК Г - Архітектура трансферної моделі

Таблиця Г.1 - Архітектура трансферної згорткової нейронної мережі

Layer (type)	Output Shape	Param	Connected to
input_layer (InputLayer)	(None, 256, 256, 3)	0	-
augmentation (Sequential)	(None, 256, 256, 3)	0	input_layer[0][0]
get_item (GetItem)	(None, 256, 256)	0	augmentation[0][0]
get_item_1 (GetItem)	(None, 256, 256)	0	augmentation[0][0]
get_item_2 (GetItem)	(None, 256, 256)	0	augmentation[0][0]
stack (Stack)	(None, 256, 256, 3)	0	get_item[0][0], get_item_1[0][0], get_item_2[0][0]
add (Add)	(None, 256, 256, 3)	0	stack[0][0]
conv1_pad (ZeroPadding2D)	(None, 262, 262, 3)	0	add[0][0]
conv1_conv (Conv2D)	(None, 128, 128, 64)	9,472	conv1_pad[0][0]
conv1_bn (BatchNormalization)	(None, 128, 128, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 128, 128, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 130, 130, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 64, 64, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 64, 64, 64)	4,16	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormalization)	(None, 64, 64, 64)	256	conv2_block1_1_conv[0][0]
conv2_block1_1_relu (Activation)	(None, 64, 64, 64)	0	conv2_block1_1_bn[0][0]
conv2_block1_2_conv (Conv2D)	(None, 64, 64, 64)	36,928	conv2_block1_1_relu[0][0]
conv2_block1_2_bn (BatchNormalization)	(None, 64, 64, 64)	256	conv2_block1_2_conv[0][0]
conv2_block1_2_relu (Activation)	(None, 64, 64, 64)	0	conv2_block1_2_bn[0][0]
conv2_block1_0_conv (Conv2D)	(None, 64, 64, 256)	16,64	pool1_pool[0][0]
conv2_block1_3_conv (Conv2D)	(None, 64, 64, 256)	16,64	conv2_block1_2_relu[0][0]
conv2_block1_0_bn (BatchNormalization)	(None, 64, 64, 256)	1,024	conv2_block1_0_conv[0][0]
conv2_block1_3_bn (BatchNormalization)	(None, 64, 64, 256)	1,024	conv2_block1_3_conv[0][0]
conv2_block1_add (Add)	(None, 64, 64, 256)	0	conv2_block1_0_bn[0][0], conv2_block1_3_bn[0][0]
conv2_block1_out (Activation)	(None, 64, 64, 256)	0	conv2_block1_add[0][0]
conv2_block2_1_conv (Conv2D)	(None, 64, 64, 64)	16,448	conv2_block1_out[0][0]
conv2_block2_1_bn (BatchNormalization)	(None, 64, 64, 64)	256	conv2_block2_1_conv[0][0]
conv2_block2_1_relu (Activation)	(None, 64, 64, 64)	0	conv2_block2_1_bn[0][0]
conv2_block2_2_conv (Conv2D)	(None, 64, 64, 64)	36,928	conv2_block2_1_relu[0][0]
conv2_block2_2_bn (BatchNormalization)	(None, 64, 64, 64)	256	conv2_block2_2_conv[0][0]
conv2_block2_2_relu (Activation)	(None, 64, 64, 64)	0	conv2_block2_2_bn[0][0]
conv2_block2_3_conv (Conv2D)	(None, 64, 64, 256)	16,64	conv2_block2_2_relu[0][0]
conv2_block2_3_bn (BatchNormalization)	(None, 64, 64, 256)	1,024	conv2_block2_3_conv[0][0]
conv2_block2_add (Add)	(None, 64, 64, 256)	0	conv2_block1_out[0][0], conv2_block2_3_bn[0][0]
conv2_block2_out (Activation)	(None, 64, 64, 256)	0	conv2_block2_add[0][0]
conv2_block3_1_conv (Conv2D)	(None, 64, 64, 64)	16,448	conv2_block2_out[0][0]
conv2_block3_1_bn (BatchNormalization)	(None, 64, 64, 64)	256	conv2_block3_1_conv[0][0]

Продовження 1 таблиці Г1

conv2_block3_1_relu (Activation)	(None, 64, 64, 64)	0	conv2_block3_1_bn[0][0]
conv2_block3_2_conv (Conv2D)	(None, 64, 64, 64)	36,928	conv2_block3_1_relu[0][0]
conv2_block3_2_bn (BatchNormalization)	(None, 64, 64, 64)	256	conv2_block3_2_conv[0][0]
conv2_block3_2_relu (Activation)	(None, 64, 64, 64)	0	conv2_block3_2_bn[0][0]
conv2_block3_3_conv (Conv2D)	(None, 64, 64, 256)	16,64	conv2_block3_2_relu[0][0]
conv2_block3_3_bn (BatchNormalization)	(None, 64, 64, 256)	1,024	conv2_block3_3_conv[0][0]
conv2_block3_add (Add)	(None, 64, 64, 256)	0	conv2_block2_out[0][0], conv2_block3_3_bn[0][0]
conv2_block3_out (Activation)	(None, 64, 64, 256)	0	conv2_block3_add[0][0]
conv3_block1_1_conv (Conv2D)	(None, 32, 32, 128)	32,896	conv2_block3_out[0][0]
conv3_block1_1_bn (BatchNormalization)	(None, 32, 32, 128)	512	conv3_block1_1_conv[0][0]
conv3_block1_1_relu (Activation)	(None, 32, 32, 128)	0	conv3_block1_1_bn[0][0]
conv3_block1_2_conv (Conv2D)	(None, 32, 32, 128)	147,584	conv3_block1_1_relu[0][0]
conv3_block1_2_bn (BatchNormalization)	(None, 32, 32, 128)	512	conv3_block1_2_conv[0][0]
conv3_block1_2_relu (Activation)	(None, 32, 32, 128)	0	conv3_block1_2_bn[0][0]
conv3_block1_0_conv (Conv2D)	(None, 32, 32, 512)	131,584	conv2_block3_out[0][0]
conv3_block1_3_conv (Conv2D)	(None, 32, 32, 512)	66,048	conv3_block1_2_relu[0][0]
conv3_block1_0_bn (BatchNormalization)	(None, 32, 32, 512)	2,048	conv3_block1_0_conv[0][0]
conv3_block1_3_bn (BatchNormalization)	(None, 32, 32, 512)	2,048	conv3_block1_3_conv[0][0]
conv3_block1_add (Add)	(None, 32, 32, 512)	0	conv3_block1_0_bn[0][0], conv3_block1_3_bn[0][0]
conv3_block1_out (Activation)	(None, 32, 32, 512)	0	conv3_block1_add[0][0]
conv3_block2_1_conv (Conv2D)	(None, 32, 32, 128)	65,664	conv3_block1_out[0][0]
conv3_block2_1_bn (BatchNormalization)	(None, 32, 32, 128)	512	conv3_block2_1_conv[0][0]
conv3_block2_1_relu (Activation)	(None, 32, 32, 128)	0	conv3_block2_1_bn[0][0]
conv3_block2_2_conv (Conv2D)	(None, 32, 32, 128)	147,584	conv3_block2_1_relu[0][0]
conv3_block2_2_bn (BatchNormalization)	(None, 32, 32, 128)	512	conv3_block2_2_conv[0][0]
conv3_block2_2_relu (Activation)	(None, 32, 32, 128)	0	conv3_block2_2_bn[0][0]
conv3_block2_3_conv (Conv2D)	(None, 32, 32, 512)	66,048	conv3_block2_2_relu[0][0]
conv3_block2_3_bn (BatchNormalization)	(None, 32, 32, 512)	2,048	conv3_block2_3_conv[0][0]
conv3_block2_add (Add)	(None, 32, 32, 512)	0	conv3_block1_out[0][0], conv3_block2_3_bn[0][0]
conv3_block2_out (Activation)	(None, 32, 32, 512)	0	conv3_block2_add[0][0]
conv3_block3_1_conv (Conv2D)	(None, 32, 32, 128)	65,664	conv3_block2_out[0][0]
conv3_block3_1_bn (BatchNormalization)	(None, 32, 32, 128)	512	conv3_block3_1_conv[0][0]
conv3_block3_1_relu (Activation)	(None, 32, 32, 128)	0	conv3_block3_1_bn[0][0]
conv3_block3_2_conv (Conv2D)	(None, 32, 32, 128)	147,584	conv3_block3_1_relu[0][0]
conv3_block3_2_bn (BatchNormalization)	(None, 32, 32, 128)	512	conv3_block3_2_conv[0][0]
conv3_block3_2_relu (Activation)	(None, 32, 32, 128)	0	conv3_block3_2_bn[0][0]
conv3_block3_3_conv (Conv2D)	(None, 32, 32, 512)	66,048	conv3_block3_2_relu[0][0]
conv3_block3_3_bn (BatchNormalization)	(None, 32, 32, 512)	2,048	conv3_block3_3_conv[0][0]
conv3_block3_add (Add)	(None, 32, 32, 512)	0	conv3_block2_out[0][0], conv3_block3_3_bn[0][0]
conv3_block3_out (Activation)	(None, 32, 32, 512)	0	conv3_block3_add[0][0]

Продовження 2 таблиці Г1

conv3_block4_1_conv (Conv2D)	(None, 32, 32, 128)	65,664	conv3_block3_out[0][0]
conv3_block4_1_bn (BatchNormalization)	(None, 32, 32, 128)	512	conv3_block4_1_conv[0][0]
conv3_block4_1_relu (Activation)	(None, 32, 32, 128)	0	conv3_block4_1_bn[0][0]
conv3_block4_2_conv (Conv2D)	(None, 32, 32, 128)	147,584	conv3_block4_1_relu[0][0]
conv3_block4_2_bn (BatchNormalization)	(None, 32, 32, 128)	512	conv3_block4_2_conv[0][0]
conv3_block4_2_relu (Activation)	(None, 32, 32, 128)	0	conv3_block4_2_bn[0][0]
conv3_block4_3_conv (Conv2D)	(None, 32, 32, 512)	66,048	conv3_block4_2_relu[0][0]
conv3_block4_3_bn (BatchNormalization)	(None, 32, 32, 512)	2,048	conv3_block4_3_conv[0][0]
conv3_block4_add (Add)	(None, 32, 32, 512)	0	conv3_block3_out[0][0], conv3_block4_3_bn[0][0]
conv3_block4_out (Activation)	(None, 32, 32, 512)	0	conv3_block4_add[0][0]
conv4_block1_1_conv (Conv2D)	(None, 16, 16, 256)	131,328	conv3_block4_out[0][0]
conv4_block1_1_bn (BatchNormalization)	(None, 16, 16, 256)	1,024	conv4_block1_1_conv[0][0]
conv4_block1_1_relu (Activation)	(None, 16, 16, 256)	0	conv4_block1_1_bn[0][0]
conv4_block1_2_conv (Conv2D)	(None, 16, 16, 256)	590,08	conv4_block1_1_relu[0][0]
conv4_block1_2_bn (BatchNormalization)	(None, 16, 16, 256)	1,024	conv4_block1_2_conv[0][0]
conv4_block1_2_relu (Activation)	(None, 16, 16, 256)	0	conv4_block1_2_bn[0][0]
conv4_block1_0_conv (Conv2D)	(None, 16, 16, 1024)	525,312	conv3_block4_out[0][0]
conv4_block1_3_conv (Conv2D)	(None, 16, 16, 1024)	263,168	conv4_block1_2_relu[0][0]
conv4_block1_0_bn (BatchNormalization)	(None, 16, 16, 1024)	4,096	conv4_block1_0_conv[0][0]
conv4_block1_3_bn (BatchNormalization)	(None, 16, 16, 1024)	4,096	conv4_block1_3_conv[0][0]
conv4_block1_add (Add)	(None, 16, 16, 1024)	0	conv4_block1_0_bn[0][0], conv4_block1_3_bn[0][0]
conv4_block1_out (Activation)	(None, 16, 16, 1024)	0	conv4_block1_add[0][0]
conv4_block2_1_conv (Conv2D)	(None, 16, 16, 256)	262,4	conv4_block1_out[0][0]
conv4_block2_1_bn (BatchNormalization)	(None, 16, 16, 256)	1,024	conv4_block2_1_conv[0][0]
conv4_block2_1_relu (Activation)	(None, 16, 16, 256)	0	conv4_block2_1_bn[0][0]
conv4_block2_2_conv (Conv2D)	(None, 16, 16, 256)	590,08	conv4_block2_1_relu[0][0]
conv4_block2_2_bn (BatchNormalization)	(None, 16, 16, 256)	1,024	conv4_block2_2_conv[0][0]
conv4_block2_2_relu (Activation)	(None, 16, 16, 256)	0	conv4_block2_2_bn[0][0]
conv4_block2_3_conv (Conv2D)	(None, 16, 16, 1024)	263,168	conv4_block2_2_relu[0][0]
conv4_block2_3_bn (BatchNormalization)	(None, 16, 16, 1024)	4,096	conv4_block2_3_conv[0][0]
conv4_block2_add (Add)	(None, 16, 16, 1024)	0	conv4_block1_out[0][0], conv4_block2_3_bn[0][0]
conv4_block2_out (Activation)	(None, 16, 16, 1024)	0	conv4_block2_add[0][0]
conv4_block3_1_conv (Conv2D)	(None, 16, 16, 256)	262,4	conv4_block2_out[0][0]
conv4_block3_1_bn (BatchNormalization)	(None, 16, 16, 256)	1,024	conv4_block3_1_conv[0][0]
conv4_block3_1_relu (Activation)	(None, 16, 16, 256)	0	conv4_block3_1_bn[0][0]
conv4_block3_2_conv (Conv2D)	(None, 16, 16, 256)	590,08	conv4_block3_1_relu[0][0]
conv4_block3_2_bn (BatchNormalization)	(None, 16, 16, 256)	1,024	conv4_block3_2_conv[0][0]
conv4_block3_2_relu (Activation)	(None, 16, 16, 256)	0	conv4_block3_2_bn[0][0]
conv4_block3_3_conv (Conv2D)	(None, 16, 16, 1024)	263,168	conv4_block3_2_relu[0][0]
conv4_block3_3_bn (BatchNormalization)	(None, 16, 16, 1024)	4,096	conv4_block3_3_conv[0][0]

Продовження 3 таблиці Г1

conv4_block3_add (Add)	(None, 16, 16, 1024)	0	conv4_block2_out[0][0], conv4_block3_3_bn[0][0]
conv4_block3_out (Activation)	(None, 16, 16, 1024)	0	conv4_block3_add[0][0]
conv4_block4_1_conv (Conv2D)	(None, 16, 16, 256)	262,4	conv4_block3_out[0][0]
conv4_block4_1_bn (BatchNormalization)	(None, 16, 16, 256)	1,024	conv4_block4_1_conv[0][0]
conv4_block4_1_relu (Activation)	(None, 16, 16, 256)	0	conv4_block4_1_bn[0][0]
conv4_block4_2_conv (Conv2D)	(None, 16, 16, 256)	590,08	conv4_block4_1_relu[0][0]
conv4_block4_2_bn (BatchNormalization)	(None, 16, 16, 256)	1,024	conv4_block4_2_conv[0][0]
conv4_block4_2_relu (Activation)	(None, 16, 16, 256)	0	conv4_block4_2_bn[0][0]
conv4_block4_3_conv (Conv2D)	(None, 16, 16, 1024)	263,168	conv4_block4_2_relu[0][0]
conv4_block4_3_bn (BatchNormalization)	(None, 16, 16, 1024)	4,096	conv4_block4_3_conv[0][0]
conv4_block4_add (Add)	(None, 16, 16, 1024)	0	conv4_block3_out[0][0], conv4_block4_3_bn[0][0]
conv4_block4_out (Activation)	(None, 16, 16, 1024)	0	conv4_block4_add[0][0]
conv4_block5_1_conv (Conv2D)	(None, 16, 16, 256)	262,4	conv4_block4_out[0][0]
conv4_block5_1_bn (BatchNormalization)	(None, 16, 16, 256)	1,024	conv4_block5_1_conv[0][0]
conv4_block5_1_relu (Activation)	(None, 16, 16, 256)	0	conv4_block5_1_bn[0][0]
conv4_block5_2_conv (Conv2D)	(None, 16, 16, 256)	590,08	conv4_block5_1_relu[0][0]
conv4_block5_2_bn (BatchNormalization)	(None, 16, 16, 256)	1,024	conv4_block5_2_conv[0][0]
conv4_block5_2_relu (Activation)	(None, 16, 16, 256)	0	conv4_block5_2_bn[0][0]
conv4_block5_3_conv (Conv2D)	(None, 16, 16, 1024)	263,168	conv4_block5_2_relu[0][0]
conv4_block5_3_bn (BatchNormalization)	(None, 16, 16, 1024)	4,096	conv4_block5_3_conv[0][0]
conv4_block5_add (Add)	(None, 16, 16, 1024)	0	conv4_block4_out[0][0], conv4_block5_3_bn[0][0]
conv4_block5_out (Activation)	(None, 16, 16, 1024)	0	conv4_block5_add[0][0]
conv4_block6_1_conv (Conv2D)	(None, 16, 16, 256)	262,4	conv4_block5_out[0][0]
conv4_block6_1_bn (BatchNormalization)	(None, 16, 16, 256)	1,024	conv4_block6_1_conv[0][0]
conv4_block6_1_relu (Activation)	(None, 16, 16, 256)	0	conv4_block6_1_bn[0][0]
conv4_block6_2_conv (Conv2D)	(None, 16, 16, 256)	590,08	conv4_block6_1_relu[0][0]
conv4_block6_2_bn (BatchNormalization)	(None, 16, 16, 256)	1,024	conv4_block6_2_conv[0][0]
conv4_block6_2_relu (Activation)	(None, 16, 16, 256)	0	conv4_block6_2_bn[0][0]
conv4_block6_3_conv (Conv2D)	(None, 16, 16, 1024)	263,168	conv4_block6_2_relu[0][0]
conv4_block6_3_bn (BatchNormalization)	(None, 16, 16, 1024)	4,096	conv4_block6_3_conv[0][0]
conv4_block6_add (Add)	(None, 16, 16, 1024)	0	conv4_block5_out[0][0], conv4_block6_3_bn[0][0]
conv4_block6_out (Activation)	(None, 16, 16, 1024)	0	conv4_block6_add[0][0]
conv5_block1_1_conv (Conv2D)	(None, 8, 8, 512)	524,8	conv4_block6_out[0][0]
conv5_block1_1_bn (BatchNormalization)	(None, 8, 8, 512)	2,048	conv5_block1_1_conv[0][0]
conv5_block1_1_relu (Activation)	(None, 8, 8, 512)	0	conv5_block1_1_bn[0][0]
conv5_block1_2_conv (Conv2D)	(None, 8, 8, 512)	2,359,808	conv5_block1_1_relu[0][0]
conv5_block1_2_bn (BatchNormalization)	(None, 8, 8, 512)	2,048	conv5_block1_2_conv[0][0]
conv5_block1_2_relu (Activation)	(None, 8, 8, 512)	0	conv5_block1_2_bn[0][0]
conv5_block1_0_conv (Conv2D)	(None, 8, 8, 2048)	2,099,200	conv4_block6_out[0][0]
conv5_block1_3_conv (Conv2D)	(None, 8, 8, 2048)	1,050,624	conv5_block1_2_relu[0][0]

Продовження 4 таблиці Г1

conv5_block1_0_bn (BatchNormalization)	(None, 8, 8, 2048)	8,192	conv5_block1_0_conv[0][0]
conv5_block1_3_bn (BatchNormalization)	(None, 8, 8, 2048)	8,192	conv5_block1_3_conv[0][0]
conv5_block1_add (Add)	(None, 8, 8, 2048)	0	conv5_block1_0_bn[0][0], conv5_block1_3_bn[0][0]
conv5_block1_out (Activation)	(None, 8, 8, 2048)	0	conv5_block1_add[0][0]
conv5_block2_1_conv (Conv2D)	(None, 8, 8, 512)	1,049,088	conv5_block1_out[0][0]
conv5_block2_1_bn (BatchNormalization)	(None, 8, 8, 512)	2,048	conv5_block2_1_conv[0][0]
conv5_block2_1_relu (Activation)	(None, 8, 8, 512)	0	conv5_block2_1_bn[0][0]
conv5_block2_2_conv (Conv2D)	(None, 8, 8, 512)	2,359,808	conv5_block2_1_relu[0][0]
conv5_block2_2_bn (BatchNormalization)	(None, 8, 8, 512)	2,048	conv5_block2_2_conv[0][0]
conv5_block2_2_relu (Activation)	(None, 8, 8, 512)	0	conv5_block2_2_bn[0][0]
conv5_block2_3_conv (Conv2D)	(None, 8, 8, 2048)	1,050,624	conv5_block2_2_relu[0][0]
conv5_block2_3_bn (BatchNormalization)	(None, 8, 8, 2048)	8,192	conv5_block2_3_conv[0][0]
conv5_block2_add (Add)	(None, 8, 8, 2048)	0	conv5_block1_out[0][0], conv5_block2_3_bn[0][0]
conv5_block2_out (Activation)	(None, 8, 8, 2048)	0	conv5_block2_add[0][0]
conv5_block3_1_conv (Conv2D)	(None, 8, 8, 512)	1,049,088	conv5_block2_out[0][0]
conv5_block3_1_bn (BatchNormalization)	(None, 8, 8, 512)	2,048	conv5_block3_1_conv[0][0]
conv5_block3_1_relu (Activation)	(None, 8, 8, 512)	0	conv5_block3_1_bn[0][0]
conv5_block3_2_conv (Conv2D)	(None, 8, 8, 512)	2,359,808	conv5_block3_1_relu[0][0]
conv5_block3_2_bn (BatchNormalization)	(None, 8, 8, 512)	2,048	conv5_block3_2_conv[0][0]
conv5_block3_2_relu (Activation)	(None, 8, 8, 512)	0	conv5_block3_2_bn[0][0]
conv5_block3_3_conv (Conv2D)	(None, 8, 8, 2048)	1,050,624	conv5_block3_2_relu[0][0]
conv5_block3_3_bn (BatchNormalization)	(None, 8, 8, 2048)	8,192	conv5_block3_3_conv[0][0]
conv5_block3_add (Add)	(None, 8, 8, 2048)	0	conv5_block2_out[0][0], conv5_block3_3_bn[0][0]
conv5_block3_out (Activation)	(None, 8, 8, 2048)	0	conv5_block3_add[0][0]
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0	conv5_block3_out[0][0]
dropout (Dropout)	(None, 2048)	0	global_average_pooling2d[0][0]
dense (Dense)	(None, 64)	131,136	dropout[0][0]
predictions (Dense)	(None, 8)	520	dense[0][0]