Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

«Затверджую»
в.о. завідуючого кафедри
комп'ютерних систем та робототехніки
_____ к. ф.-м. н., доцент Максим ХРУСЛОВ
«___» червня 2025 р.

Пояснювальна записка

до кваліфікаційної роботи
бакалавра
на тему **«МОДЕЛЬ КЛАСИФІКАЦІЇ СТИЛІВ АРХІТЕКТУРНИХ СПОРУД»**

Спеціальність 123 – Комп'ютерна інженерія.
Галузь знань: 12 – Інформаційні технології.
Освітня програма «Комп'ютерна інженерія».

Захищено на засіданні

Екзаменаційної комісії № 44

протокол № __ від __.06.2025 р.

Оцінка _____ / _____

Голова Екзаменаційної комісії

_____ ЧУГАЙ А.М.

Виконала:
Студентка групи КІ– 41
ЄРЬОМІНА Надія Володимирівна__

Керівник: доцент кафедри комп'ютерних систем та робототехніки, к.т.н.
СТРІЛЕЦЬ Вікторія Євгенівна ____

Рецензент: д.т.н., професор; професор кафедри математичного моделювання та аналізу даних
КІРІЧЕНКО Людмила Олегівна_____

Харків – 2025

АНОТАЦІЯ

Пояснювальна записка до бакалаврської кваліфікаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел і п’ятьох додатків. Загальний обсяг роботи складає 76 сторінок, із яких 50 сторінок основної частини з 29 рисунками, 1 таблицею, списку використаних джерел із 34 найменувань та 5 додатками.

У роботі розглянуто застосування методів глибокого навчання для задачі класифікації архітектурних стилів за зображеннями фасадів. Розроблено програмну модель на основі ResNet50, яка демонструє високу точність розпізнавання та придатна для використання у практичних застосунках.

Мета кваліфікаційної роботи – підвищення якості автоматизованого розпізнавання стилів архітектурних споруд на зображеннях за допомогою моделей і методів штучного інтелекту.

Об’єкт дослідження – процес розпізнавання стилів архітектурних споруд на зображеннях.

Предмет дослідження – моделі і методи штучного інтелекту для розпізнавання і аналізу зображень, зокрема методи машинного навчання і глибокого навчання.

Проблема, яка вирішується у даній кваліфікаційній роботі, полягає у створенні ефективної моделі, що дозволяє автоматично класифікувати архітектурні стилі будівель за зображеннями, враховуючи складність розрізнення схожих стилів та різноманітність візуальних ознак.

Область застосування розробленої моделі охоплює автоматизований візуальний аналіз міського середовища, цифрову архівацію й класифікацію об’єктів культурної спадщини, підтримку містобудівних і реставраційних проєктів, а також використання у навчальних і спеціалізованих додатках для архітекторів, урбаністів і музейних працівників.

***Ключові слова*:** класифікація архітектурних стилів, глибоке навчання, convolutional neural network, ResNet50, transfer learning, комп’ютерний зір.

ABSTRACT

The thesis for the bachelor's qualification work consists of an introduction, three chapters, conclusions, a list of references, and five appendices. The total length of the work is 76 pages, including 50 pages of the main part with 29 figures, 1 table, a reference list of 34 sources, and 5 appendices.

The work investigates the application of deep learning methods to the task of classifying architectural styles based on images of building facades. A deep learning model based on ResNet50 was developed, demonstrating high recognition accuracy and suitability for practical implementation.

The qualification work aims to improve the quality of automated recognition of architectural styles in images using artificial intelligence models and methods.

The object of the study is the process of recognizing architectural styles in images.

The subject of the study is artificial intelligence models and methods for image recognition and analysis, particularly machine learning and deep learning techniques.

The problem addressed in this qualification work is the creation of an effective model that enables automatic classification of architectural styles of buildings from images, considering the challenge of distinguishing similar styles and the diversity of visual features.

The area of application of the developed model includes automated visual analysis of urban environments, digital archiving and classification of cultural heritage objects, support for urban planning and restoration projects, as well as use in educational and specialized applications for architects, urban planners, and museum professionals.

Keywords**:** architectural style classification, deep learning, convolutional neural network, ResNet50, transfer learning, computer vision.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

ПК – укр. персональний комп’ютер

AI – англ. Artificial Intelligence, укр. штучний інтелект

Adam – англ. Adaptive Moment Estimation, укр. адаптивна оцінка моментів

BoVW – англ. Bag-of-Visual-Words, укр. мішок візуальних слів

CNN – англ. Convolutional Neural Network, укр. згорткова нейронна мережа

DPM – англ. Deformable Part Model, укр. модель деформованих частин

GAP – англ. Global Average Pooling, укр. глобальне середнє згортання

GPU – англ. Graphics Processing Unit, укр. графічний процесор

HOG – англ. Histogram of Oriented Gradients, укр. гістограма орієнтованих градієнтів

LBP – англ. Local Binary Patterns, укр. локальні бінарні шаблони

LSVM – англ. Latent Support Vector Machine, укр. латентна машина опорних векторів

MLLR – англ. Multinomial Latent Logistic Regression, укр. багатономіальна латентна логістична регресія

NBNN – англ. Naive-Bayes Nearest Neighbor, укр. наївно-байєсівський найближчий сусід

ONNX – англ. Open Neural Network Exchange, укр. відкритий формат обміну нейронними мережами

OpenCV – англ. Open Source Computer Vision Library, укр. бібліотека комп’ютерного зору

ReLU – англ. Rectified Linear Unit, укр. функція активації випрямленого лінійного вузла

SIFT – англ. Scale-Invariant Feature Transform, укр. масштабно-інваріантне перетворення ознак

SPM – англ. Spatial Pyramid Matching, укр. просторове пірамідальне зіставлення

SURF – англ. Speeded-Up Robust Features, укр. прискорені стійкі ознаки

SVM – англ. Support Vector Machine, укр. машина опорних векторів

VGG – англ. Visual Geometry Group, укр. згорткова нейронна мережа VGG

venv – англ. virtual environment, укр. віртуальне середовище

WSL – англ. Windows Subsystem for Linux, укр. підсистема Windows для Linux

ВСТУП

Актуальність роботи. У сучасному світі, де застосування цифрових технологій охоплює всі сфери життя, автоматичне розпізнавання та класифікація зображень стають ключовими технологіями для розвитку штучного інтелекту та автоматизації процесів. Комп'ютерний зір, що поєднує методи машинного та глибокого навчання з аналізом зображень, є основою для впровадження інтелектуальних систем у промисловості, медицині, безпеці та багатьох інших галузях.

Особливої актуальності набуває задача автоматичної класифікації архітектурних стилів, оскільки вона дозволяє ефективно аналізувати великі масиви візуальних даних, сприяє збереженню культурної спадщини та розвитку цифрових архівів. Сучасні підходи, зокрема використання згорткових нейронних мереж (CNN), моделей ResNet, VGG та методів transfer learning, значно підвищують точність і швидкість розпізнавання.

Розробка та дослідження ефективних моделей для класифікації архітектурних стилів є важливим внеском у розвиток комп'ютерного зору, оскільки дозволяє вирішувати складні завдання fine-grained класифікації та відкриває нові можливості для міждисциплінарних досліджень, що робить представлену роботу актуальною як з наукової, так і з практичної точки зору.

Метою дослідження є підвищення якості автоматизованого розпізнавання стилів архітектурних споруд на зображеннях за допомогою моделей і методів штучного інтелекту.

Об’єкт дослідження – процес розпізнавання стилів архітектурних споруд на зображеннях.

Предмет дослідження – моделі і методи штучного інтелекту для розпізнавання і аналізу зображень, зокрема методи машинного навчання і глибокого навчання.

Методи дослідження: У ході дослідження було використано системний підхід для формалізації задачі та встановлення логічних зв’язків між її

складовими елементами. З метою теоретичного обґрунтування проведено аналітичний огляд сучасних методів класифікації зображень. Для розробки та навчання моделі застосовано методи глибокого навчання, зокрема згорткову нейронну мережу ResNet50 у поєднанні з підходом transfer learning. Для оцінювання ефективності побудованої моделі проведено статистичний аналіз результатів класифікації.

Для реалізації поставленої мети необхідно вирішити такі *завдання*:

1. Провести аналітичний огляд сучасних методів класифікації зображень, зокрема архітектурних об’єктів.

2. Підготувати набір зображень фасадів будівель із відповідною розміткою архітектурних стилів.

3. Виконати попередню обробку набору даних і провести аугментацію для підвищення узагальнюючої здатності моделі.

4. Адаптувати архітектуру моделі ResNet50 до задачі багатокласового розпізнавання зображень та навчити модель за допомогою фреймворку PyTorch.

5. Провести оцінку якості класифікації на основі метрик ефективності.

6. Проаналізувати результати, визначити переваги й обмеження створеної моделі та її програмної реалізації.

За результатами виконання зазначених завдань буде створений ефективний інструмент для автоматичної класифікації архітектурних стилів за зображеннями фасадів, придатний для використання у візуальному аналізі міського середовища, цифровій архівації об’єктів культурної спадщини, а також у навчальних системах і додатках в галузі архітектури.

РОЗДІЛ 1.
АНАЛІЗ ПІДХОДІВ ДО КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ

Класифікація зображень – це одне з фундаментальних завдань комп’ютерного зору, яке полягає у віднесенні зображення до певного класу або категорії з попередньо визначеного набору на основі його візуального вмісту.

Розвиток методів обробки візуальної інформації тісно пов'язаний з еволюцією штучного інтелекту (AI). У цьому контексті доцільно згадати популярне визначення стосовно машинного навчання, сформульоване дослідником Томом М. Мітчеллом [1]:

«Кажуть, що комп'ютерна програма навчається на досвіді D стосовно певного класу завдань T та міри виконання P, якщо її ефективність у виконанні завдань T, виміряна за допомогою P, покращується з досвідом D».

Застосовуючи це визначення до проблематики класифікації зображень, можна встановити таку відповідність: завдання *T* являє собою точне віднесення зображення до відповідної категорії, досвідом *D* виступає корпус розмічених прикладів, а показник *P* відображає кількісні метрики успішності розпізнавання. Саме на цих засадах відбувається розробка та вдосконалення алгоритмів для автоматичної інтерпретації візуальних даних.

Задача класифікації зображень належить до проблем, які занадто складні для безпосереднього програмування, подібно до розпізнавання мови, образів чи аналізу великих і складних даних. Вона потребує застосування алгоритмів машинного навчання, які здатні виявляти закономірності у вхідних даних та використовувати їх для прийняття рішень щодо нових, раніше не бачених зразків [2].

У загальному випадку підходи до класифікації зображень можна поділити на дві основні категорії: *традиційні методи машинного навчання* та *методи глибокого навчання*. Кожен із цих підходів має свої переваги та обмеження, а вибір конкретного методу залежить від специфіки задачі та типу зображень, обсягу доступних даних та наявних обчислювальних ресурсів.

1.1 Огляд підходів до класифікації зображень будівель

Розпізнавання архітектурного стилю будівель є складною задачею комп'ютерного зору, яка вимагає врахування як глобальних, так і локальних ознак зображення. Класифікація зображень будівель за архітектурними стилями є прикладом *тонкої* (fine-grained) категоризації, у якій багато спільних ознак поширюються одразу на кілька стилів, а також існують внутрішньокласові варіації залежно від регіону та епохи зведення споруд [3].

На сьогодні розроблено низку методів для розв’язання цієї задачі: від класичних підходів із розробленими ознаками (локальні ключові точки та дескриптори з подальшою класифікацією із використанням BoVW і SVM [4]) до складних латентних моделей (latent SVM, DPM та MLLR) [5], а також сучасних методів глибокого навчання (згорткові нейронні мережі, transfer learning із ResNet/VGG та ін.) [6]. Найкращі результати демонструють CNN-архітектури, які автоматично вивчають ієрархічні ознаки зображень (розділ 1.3).

На основі аналізу літературний джерел можна виділити кілька ключових груп методів, що використовуються для вирішення задачі класифікації зображень, які будуть розглянуті далі.

1.1.1 Методи дискримінативного виділення локальних шаблонів

У класичних підходах до класифікації зображень важливу роль відіграє детекція локальних ознак – ключових точок або фрагментів, що відображають структуру об’єкта незалежно від масштабу, ракурсу чи освітлення. Найпоширенішими дескрипторами є: SIFT (Scale-Invariant Feature Transform) – інваріантний до поворотів і масштабування; SURF (Speeded-Up Robust Features) – оптимізує обчислення ознак; HOG (Histogram of Oriented Gradients) – описує напрямки градієнтів; LBP (Local Binary Patterns) – аналізує текстурні шаблони.

Ці дескриптори зазвичай перетворюються у векторне представлення за допомогою моделі Bag-of-Visual-Words (BoVW), коли ознаки квантуються у «візуальні слова» (тобто усі дескриптори з навчального набору зображень

збираються разом, а потім кластеризуються), і зображення подається як гістограма частотності цих слів [4].

Одним із традиційних підходів є використання дискримінативних локальних шаблонів (discriminative local-based patches or patterns) – фрагментів, що часто повторюються в одному класі й мають високу розпізнавальну здатність. У класифікації архітектурних стилів це можуть бути арки, карнизи, орнаменти.

У роботі «Visual pattern discovery for architecture image classification and product image search» [7] було запропоновано метод виявлення візуальних шаблонів, стійких до масштабування, обертань і деформацій, що дозволив класифікувати зображення за чотирма стилями з високою точністю. Інше дослідження [8] поєднало BoVW із семантичними шаблонами (наприклад, тип вікна або балкона), що дало змогу класифікувати п’ять стилів, хоча ефективність знижувалась зі зростанням кількості класів чи складності фасадів.

Переваги підходу – інтерпретованість, стійкість до локальних змін і ефективність при малих даних. Недоліки – чутливість до варіативності фасаду, обмежена масштабованість і слабке узагальнення порівняно з глибинними моделями. Нині такі шаблони мають здебільшого допоміжну або пояснювальну функцію, основне навантаження несе згорткова мережа.

1.1.2 Методи Multinomial Latent Logistic Regression та Latent SVM

Для класифікації архітектурних стилів зручно застосовувати моделі з латентними змінними, які дозволяють моделювати частини об’єктів. У цьому контексті Multinomial Latent Logistic Regression (MLLR) і Latent Support Vector Machine (LSVM) є потужними методами, особливо в аналізі зображень. Обидва підходи виявляють менш виражені, але важливі ознаки за допомогою прихованих змінних, проте мають свої особливості.

MLLR – розширення логістичної регресії з латентними змінними, що враховує контекст менш виражених ознак поруч із домінантними. Важливо для архітектури, де значущість елементів (арки, кути, декор) залежить від просторового контексту. Метод запропонований у роботі Z. Xu [9]. Модель працює в багатомасштабному режимі з HOG-пірамідою та базується на

частинній деформованій моделі з трьома компонентами: кореневий фільтр (загальний контур будівлі), фільтри деталей з удвічі вищою роздільною здатністю (локальні елементи), і витрати на деформацію (моделювання варіативності розміщення частин).

MLLR, як і LSVM, працює з деформованими моделями, але має переваги: тренує всі класи одночасно в єдиній цільовій функції, зберігаючи кореляції між класами й уникаючи проблем калібрування. Модель підтримує ймовірнісний висновок для багатостильової приналежності об'єкта. MLLR ефективний при класифікації великої кількості стилів з малопомітними регулярними деталями, але обмежений для слабо структурованих стилів або незвичних ракурсів.

LSVM – модифікація класичного SVM з використанням латентних змінних, яка також враховує деформації й варіативність об’єктів. Однак на відміну від MLLR, LSVM чутливий до дисбалансу у вибірках, коли негативних прикладів значно більше, ніж позитивних (типова ситуація в задачах класифікації архітектурних стилів, де часто домінує клас «фон»).

У багатокласових задачах LSVM застосовує стратегії «one-vs-one» або «one-vs-rest», де кожен клас розглядається окремо. Це обмежує здатність моделі враховувати взаємозв’язки й може призводити до труднощів з калібруванням. LSVM добре працює при невеликій кількості чітко структурованих класів, але менш ефективний у задачах з багатьма класами або сильним дисбалансом [10].

Отже, обидва методи використовують латентні змінні для врахування структурної варіативності. Проте MLLR виявляється гнучкішим і ефективнішим для складних багатокласових задач завдяки здатності зберігати міжкласові зв’язки та реалізовувати ймовірнісний підхід. Натомість LSVM доцільно застосовувати для задач із малою кількістю добре визначених класів.

1.1.3 Алгоритм Naive-Bayes Nearest-Neighbor

NBNN (Naive-Bayes Nearest Neighbor) – це безтренувальний, непараметричний метод класифікації, який розглядає зображення як набір локальних дескрипторів і класифікує їх на основі сумарних відстаней до найближчих сусідів у кожному класі. На відміну від методів, що квантують

ознаки, NBNN безпосередньо порівнює дескриптори, використовуючи евклідову відстань і припущення наївного байєсівського класифікатора [4].

У праці «In Defense of Nearest-Neighbor Based Image Classification» [4] метод представлено як такий, що визначає клас тестового зображення через мінімізацію сумарної відстані до найближчих дескрипторів з навчального набору. Клас, який має найменшу суму таких відстаней, і вважається правильним.

До переваг NBNN належить простота реалізації, відсутність потреби в навчанні складних моделей (наприклад, глибоких нейронних мереж), низькі вимоги до обчислювальних ресурсів і стійкість до перенавчання.

Однак метод менш ефективний у задачах із високою візуальною подібністю між класами – як-от при класифікації архітектурних стилів. У таких випадках евклідова відстань може не відображати реальну схожість. Також метод чутливий до нерепрезентативної або зашумленої навчальної вибірки. Ще один недолік – висока обчислювальна складність при великому обсязі даних, оскільки відстані обчислюються до кожного дескриптора навчального набору.

Отже, NBNN добре працює в умовах чітко розмежованих класів та якісного навчального набору, але має обмеження при високій міжкласовій подібності або масштабних даних.

1.1.4 Підхід «Bag-of-Visual-Words»

Модель Bag-of-Visual-Words (BoVW) є стандартним підходом у класичних системах візуального розпізнавання. Локальні дескриптори (зазвичай SIFT) кластеризуються (наприклад, методом K-means) для формування візуального словника. Кожне зображення далі представляється у вигляді гістограми частот поява візуальних слів, без урахування просторової структури. Отримана векторна гістограма подається на вхід класифікатора, наприклад, SVM. Простота та ефективність BoVW забезпечили йому широку популярність, однак жорстке квантування ознак призводить до втрати частини інформації.

З метою часткової компенсації цієї втрати застосовується метод Spatial Pyramid Matching (SPM), який передбачає розбиття зображення на області різних

масштабів і обчислення гістограм для кожної з них. Це дозволяє зберегти просторову інформацію та значно підвищує дискримінативність моделі [11]. Загалом, підхід BoVW із SVM є ефективним для багатьох задач класифікації, однак у випадку з класифікацією складних архітектурних стилів може поступатися глибоким нейронним мережам через спрощене подання ознак.

1.1.5 Методи глибокого навчання

Підхід глибокого навчання (deep learning) до класифікації архітектурних споруд базується на використанні згорткових нейронних мереж (CNN), що здатні автоматично вивчати багаторівневі представлення візуальних даних. На початкових шарах мережі виділяються ознаки низького рівня (контури, текстури), у середніх –патерни середнього рівня (геометричні форми, ритмічні елементи фасаду), а на глибоких – концептуальні ознаки високого рівня, характерні для певного архітектурного стилю. Використання таких ієрархічних ознак значно підвищує точність класифікації порівняно з традиційними методами, адже враховує як локальні деталі, так і глобальну структуру фасаду [12].

У задачах класифікації архітектурних стилів найпоширенішими є два основні підходи до застосування CNN:

1. Transfer learning.
2. Навчання «з нуля» власної архітектури:

Побудова і навчання мережі починається з випадкових ваг, без попередньої ініціалізації, із підбором оптимальної глибини, кількості фільтрів і гіперпараметрів для специфічного корпусу зображень фасадів.

Обидва підходи – transfer learning та навчання «з нуля» власної архітектури – демонструють високу гнучкість у врахуванні специфіки корпусу фасадних зображень і дозволяють досягати точності класифікації, недоступної традиційним методам. Transfer learning скорочує час розробки і знижує вимоги до обсягу даних, тоді як навчання «з нуля» дає змогу тонко налаштовувати мережу на особливості архітектурних стилів. У поєднанні ці стратегії забезпечують баланс між швидкістю адаптації та глибиною моделювання

візуальних патернів. Завдяки цьому deep learning стає ключовим інструментом тонкої категоризації архітектурних споруд, здатним одночасно оптимізувати продуктивність і адаптивність моделі.

1.2 Загальний огляд моделей нейронних мереж

Перед тим як будувати нейронну мережу, необхідно визначити її архітектуру, тобто спосіб організації її компонентів. На відміну від традиційних алгоритмів, які мають фіксовані етапи обробки даних, у нейронних мережах потрібно самостійно задавати схему руху інформації. Передбачається визначення кількості нейронів, їхнього розподілу по шарах, а також способу їхнього з'єднання між собою, як показано на рис. 1.1, що ілюструє узагальнену структуру штучної нейронної мережі [13].

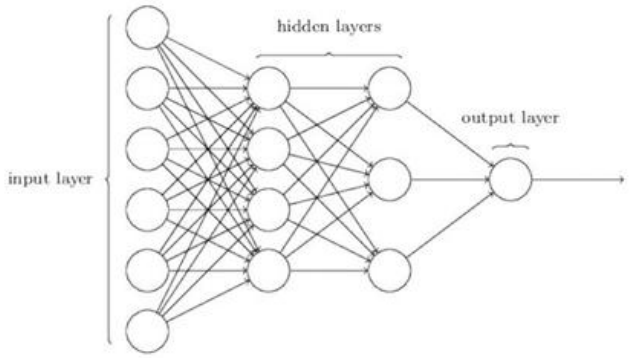


Рисунок 1.1 – Загальна структура нейронної мережі

Нейронні мережі складаються з різних *шарів*, кожен з яких має власний набір нейронів і ваг. Крайній лівий шар – це вхідний шар (input layer), що містить вхідні нейрони. Крайній правий або вихідний шар (output layer) містить вихідні нейрони, наприклад, єдиний вихідний нейрон на рис. 1.1. Середні шари називаються прихованими (hidden layer), оскільки їх нейрони не є ні вхідними, ні вихідними. Наведена вище чотиришарова мережа (рис. 1.1) має два прихованих шари. Залежно від реалізації моделі, глибока нейронна мережа може мати від двох до сотень шарів. Кожен шар отримує інформацію про різні

характеристики вхідних даних від попереднього шару, розбиває її на більш детальну інформацію і передає її наступному шару.

Основним елементом нейронної мережі є, звичайно, нейрон. Взаємопов'язані нейрони формують нейромережу, обмінюючись вхідними та вихідними сигналами. Штучний нейрон отримує певні вхідні значення $x = [x_1, x_2, \ldots, x_n]$, кожне з яких множиться на відповідну вагу $w = [w_1, w_2, \ldots, w_n]$. Ці зважені вхідні дані підсумовуються та утворюють логіти (вихідні значення нейронної мережі до застосування функції активації) нейрона z, який визначається як:

$$z = \sum_{i=0}^{n} w_i \cdot x_i. \tag{1.1}$$

Логіт нейрона потім передається через функцію f, також відому як функція активації. Функція активації визначає активність нейрона та формує вихідне значення [9]:

$$y = f(z + b), \tag{1.2}$$

де b – це зміщення (bias). Це вихідне значення у може бути передано іншому нейрону. Існують різні типи функцій активації, такі як Sigmoid, Tanh, ReLU і Softmax [7].

Розенблат запропонував правило для обчислення вихідного значення нейрона, увівши вагові коефіцієнти, $w1, w2, \ldots w1, w2, \ldots$, які є дійсними числами та відображають важливість відповідних вхідних даних для вихідного результату. Вихід нейрона (0 або 1) визначається тим, чи менша зважена сума від якогось порогового значення (threshold value). Аналогічно з ваговими коефіцієнтами, порогове значення є дійсним числом, яке є параметром нейрону [14].

$$output = \begin{cases} 0 \; if \; \sum_j w_j * x_j \leq treshold \\ 1 \; if \; \sum_j w_j * x_j > treshold \end{cases} \tag{1.3}$$

Ваги (weights) визначають силу зв'язку між нейронами в нейронній мережі. Якщо вага між двома шарами мала, то це означає, що передані значення слабо впливають на результат, і цей шлях майже не впливає на фінальний прогноз. Навпаки, велика позитивна або негативна вага значно змінює передані дані, впливаючи на роботу наступного шару та формуючи кінцевий результат. Вагові коефіцієнти зазвичай передаються у вигляді матриць (рис. 1.2). Кількість синаптичних ваг, які існують для кожного вузла, пропорційна кількості його вхідних векторів [15].

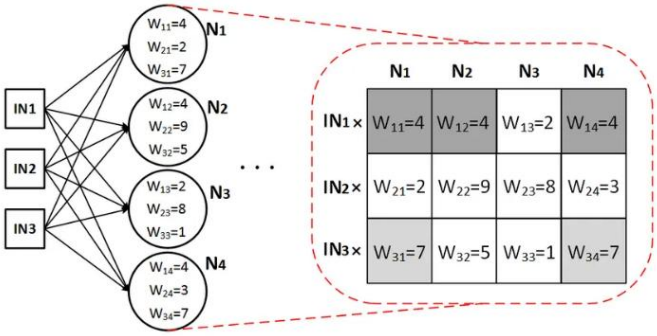


Рисунок 1.2 – Приклад матриці вагових коефіцієнтів

У нейронних мережах намагаються врахувати непередбачувані або невидимі фактори, які можуть впливати на результат. Це й є *bias* (упередженість, зсув). Кожен нейрон, що не належить до вхідного шару, має прикріплений bias. Bias має вхідне значення, яке зазвичай дорівнює 1. Це дозволяє зсунути функцію активації вліво або вправо. Рис. 1.3 ілюструє цей процес [14].

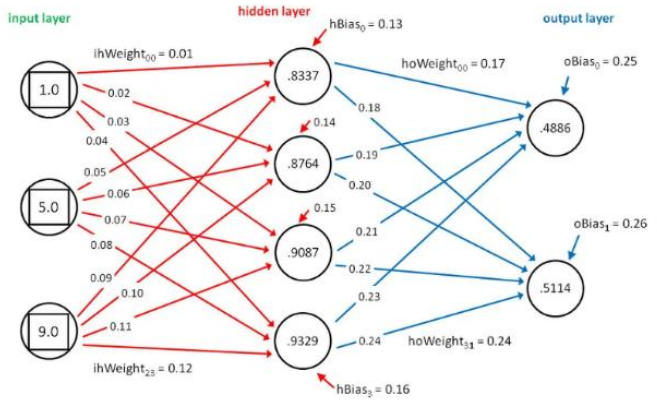


Рисунок 1.3 – Процес проходу по нейронній мережі з використанням bias

У нейронних мережах, *активаційні функції* використовуються для обчислення зваженої суми вхідних значень і зсувів, після чого приймається рішення, чи має бути активований нейрон чи ні [16]. Якщо розглянути частину рівняння (1.2) без активаційної функції, вона буде виглядати наступним чином:

$$y = b + \sum_{i=0}^{n} (x_i \cdot w_i). \quad (1.4)$$

Можна побачити, що в рівнянні (1.4) значення *y* може бути будь-яким, від -∞ до +∞. Оскільки нейрон не знає меж цього значення, використовуються активаційні функції для встановлення порогу, який визначає, чи слід активувати нейрон і передати значення далі. Нейронні мережі застосовують різні активаційні функції. Лінійна функція не здійснює трансформації й рідко використовується, оскільки зводить нейронну мережу до регресії з поліноміальними перетвореннями. Найпоширеніші — сигмоїдальна та гіперболічна тангенс функції, що ефективні в певних діапазонах *x*. З цієї причини необхідно завжди здійснювати масштабування вхідних даних для нейронної мережі за допомогою статистичної стандартизації (середнє значення 0 і одинична дисперсія) або нормалізації в межах від 0 до 1 або від -1 до 1.

Нейронні мережі використовують різні активаційні функції для введення нелінійностей, що дозволяє моделі розпізнавати складні патерни. Кожна функція має свої властивості й застосовується залежно від задачі: сигмоїдна — для бінарної класифікації, softmax — для багатокласових прогнозів, ReLU — для уникнення зникнення градієнтів. Вдалий вибір функції прискорює навчання й підвищує ефективність [17, 18].

Після проходження через активаційні функції нейронна мережа формує прогноз, який порівнюється з очікуваним значенням — ground truth — за допомогою *функції втрат* (loss function). Вона кількісно оцінює похибку моделі: чим точніше прогноз, тим менше значення втрати. Під час навчання параметри мережі змінюються так, щоб мінімізувати цю втрату, що формалізує мету

навчання — зменшити відхилення між прогнозами та правильними відповідями. У задачах з учителем loss function є ключовим інструментом для оптимізації моделі, тоді як у самонавчанні роль ground truth виконує частина самих вхідних даних [19].

1.2.1 Алгоритми навчання нейронний мереж

У глибоких мережах кожен нейрон має свою нелінійну активаційну функцію, і для диференціації всієї мережі потрібно обчислити часткові похідні численних змінних. Це здійснюється методом зворотного поширення (backpropagation), який після прямого проходу (forward pass), що завершується прогнозом, обчислює градієнт функції втрат. Зворотне поширення — ключовий етап навчання, мета якого — коригування ваг і зсувів мережі на основі помилок між передбаченими і реальними значеннями. Це досягається через ***метод зворотного поширення*** (the Backpropagation Algorithm), де зміни в параметрах мережі здійснюються в напрямку зменшення загальної помилки [20].

Процес починається з обчислення функції втрат, потім під час зворотного проходу від вихідного до вхідного шару оцінюється вплив кожного параметра на помилку для корекції ваг і зсувів. Вхідні значення z для кожного шару l для прямого поширення обчислюються наступним чином:

$$z_j^i = \sum_k w_{jk}^l * a_k^{l-1} + b_j^l, \tag{1.5}$$

де a_k^{l-1} – активація попереднього шару, w_{jk}^l– ваги шару, який використовується, b_j^l – зсув нейрону.

Активація шару обчислюється застосуванням активаційної функції до вхідних значень:

$$a^l = \sigma(w^l \cdot a^{l-1} + b^l), \tag{1.6}$$

де σ – функція активації

Для зворотного поширення, щоб скорегувати ваги, обчислюють, як змінна кожного параметра впливає на помилку.

$$\frac{\partial C}{\partial w_{jk}^{l}} = \delta_j^l \cdot a_k^{l-1}, \tag{1.7}$$

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}. \tag{1.8}$$

Формула 1.8 визначає похідну помилки по входу нейрона.

Зворотне поширення використовує правило ланцюга для обчислення градієнтів часткових похідних кожного параметра відносно функції втрат.

Градієнти, отримані під час зворотного поширення, застосовуються в алгоритмах оптимізації для покрокового налаштування параметрів мережі, що дозволяє зменшувати помилки й покращувати точність моделей. Цей метод, розроблений у 1970-х роках, є ключовим для ефективного навчання нейронних мереж [20].

Градієнтний спуск (Gradient Descent) – це алгоритм оптимізації, який використовується для знаходження оптимальних значень параметрів моделі (наприклад, у лінійній чи логістичній регресії) шляхом ітеративного оновлення параметрів з метою мінімізації функції втрат [21].

Формально, процес оновлення ваг у градієнтному спуску можна записати як:

$$\theta = \theta - \alpha \cdot \nabla J(\theta_t), \tag{1.10}$$

де θ – параметри моделі, α – швидкість навчання (learning rate), ∇J(θ) – градієнт функції втрат J(θ) за параметрами.

Цей метод є основним для навчання нейронних мереж і працює разом із backpropagation для коригування ваг.

1.2.2 Проблема згасання градієнтів

Глибина нейронної мережі визначає її здатність апроксимувати складні функції. Проте збільшення кількості шарів не завжди покращує навчання, а іноді навіть погіршує результати, збільшуючи навчальну похибку без ознак перенавчання. Навіть при наявності конструктивного розв’язання, наприклад, коли додаткові шари виконують ідентичне відображення (identity mapping), глибша модель все одно може навчатися гірше. Однією з головних причин цього

є проблема *згасання градієнтів* (vanishing gradients), яка ускладнює ефективне навчання глибоких моделей.

Під час зворотного поширення похибки градієнти обчислюються через послідовне множення похідних активаційних функцій. Чим більше шарів у моделі, тим більше малих похідних перемножується, що посилює згасання градієнта і ускладнює оновлення ваг на початкових шарах, погіршуючи навчання всієї моделі загалом.

Для часткового розв'язання цієї проблеми застосовуються активаційні функції з кращими градієнтними властивостями, такі як ReLU, яка зберігає градієнт на рівні 1 для позитивних значень вхідного сигналу. Проте навіть ReLU не усуває проблему повністю. Потрібні архітектурні інновації, розробляються архітектури з прямими з'єднаннями між шарами, які допомагають уникнути згасання градієнтів [13].

1.3 Згорткові нейронні мережі

Дослідницька праця Губеля та Візеля, здійснена у 1950–1960-х роках, стала підґрунтям для подальшого розвитку згорткових (конволюційних) нейронних мереж. У ході вивчення зорового кортексу котів вони виокремили два основні типи візуальних клітин: прості та комплексні. На основі цього відкриття була розроблена модель, що дозволяла застосовувати ці типи клітин у задачах візуального розпізнавання образів. Архітектура Neocognitron, яку у 1980 році запропонував Куніхіко Фукусіма, була безпосередньо натхненна згаданим дослідженням. Саме в цій роботі були вперше введені два ключові шари згорткової мережі — згортковий та шар зменшення розмірності. [22].

Зараз конволюційні нейронні мережі (CNN) – це тип глибоких нейронних мереж, розроблений для аналізу даних із просторовою структурою, таких як зображення та відео. Вони використовують операцію згортки (convolution), яка дозволяє автоматично виявляти значущі особливості, такі як контури, текстури та форми, без потреби в ручному проектуванні ознак [18].

Цей процес відбувається з менш локальних ознак зображення до більш загальних, та математично описується згортковою функцією:

$$f\ [\ x,y\ ] * g\ [\ x,y\ ] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n1,n2] * g[x-n_1, y-n_2]. \quad (1.11)$$

У конволюційних нейроних мережах є три основні типи шарів [23].

Конволюційні шари (Convolutional Layers) – застосовують метод Кернел для виділення характерних особливостей. Згортка – це лінійна операція, що перемножує фрагмент зображення на 2D-фільтр, формуючи карту ознак (Feature Map). Фільтр ковзає по зображенню, виявляючи задані патерни незалежно від їхнього розташування.

Шари *підвибірки* (Pooling Layers) – зменшують розмірність карти ознак, зберігаючи суттєву інформацію. Наприклад, Max Pooling та Average Pooling.

Вихід згорткового шару надмірно чутливий до дрібних деталей вхідного зображення, через що навіть незначні спотворення можуть знижувати точність розпізнавання. Щоб підвищити стійкість моделі, застосовується pooling, який зменшує розмірність карти ознак (Feature Map), зберігаючи ключову інформацію.

Max Pooling — найпоширеніший метод зменшення розмірності карти ознак, який розділяє її на області та вибирає максимальне значення в кожній. Це зменшує параметри й обчислення, покращує узагальнення ознак і підвищує стійкість моделі до варіацій вхідних зображень. Завдяки цьому Max Pooling також допомагає запобігати перенавчанню, забезпечуючи більш абстрактне представлення даних [24].

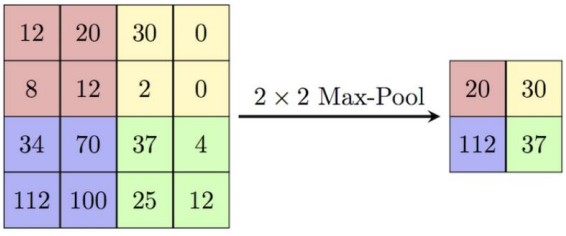


Рисунок 1.4 – Приклад операції Max-Pooling

Повнозв'язні шари (Fully Connected Layers) – в конволюційних нейронних мережах складаються з вагових коефіцієнтів (weights) і зсувів (biases) разом із нейронами. Вони виконують роль з'єднання між двома різними шарами мережі. У CNN ці шари зазвичай розташовані перед вихідним шаром і формують фінальні рівні архітектури. Вони виконують остаточну обробку та класифікацію [25].

Ще однією характерною особливістю CNN є шар *Dropout*. Він виконує функцію маски, випадковим чином вимикаючи деякі нейрони під час навчання, щоб запобігти перенавчанню. При цьому інші нейрони залишаються активними та без змін передають інформацію до наступного шару.

Пакетна норма (BatchNormalization) – це метод у глибокому навчанні, що нормалізує вихід кожного шару нейронної мережі, запропонований С. Йоффе та К. Сегеді у 2015 році для прискорення та стабілізації навчання. Цей підхід приводить вхідні дані кожного шару до нульового середнього значення та одиничного стандартного відхилення, що робить тренування мережі ефективнішим. Batch Norm часто додають до лінійних або згорткових шарів для покращення стабільності мережі [26].

1.4 Трансферне навчання

Трансферне навчання (Transfer Learning) – це метод машинного навчання, який дозволяє використовувати вже натреновану модель для вирішення нової задачі. Цей підхід особливо корисний для роботи з неструктурованими даними, такими як зображення, відео чи аудіо [27].

Головна перевага трансферного навчання полягає в тому, що воно скорочує час і обчислювальні ресурси, необхідні для навчання глибоких нейронних мереж, які зазвичай потребують великих обсягів даних та довгого тренування. Завдяки цьому підходу можна адаптувати вже натреновану модель до нової задачі, зберігаючи попередньо отримані знання та коригуючи лише останні шари мережі для специфічного завдання.

У CNN кожен шар містить фільтри, що аналізують різні аспекти вхідного зображення. Початкові шари розпізнають базові особливості, такі як краї, кольори та текстури, тоді як глибші шари формують більш складні об'єкти. При трансферному навчанні початкові шари залишаються незмінними, оскільки вони містять загальні ознаки, а останні шари підлаштовуються під нову задачу, що дозволяє ефективно застосовувати вже накопичений досвід.

Висновки за розділом 1

Підсумовуючи викладене в попередніх підрозділах, можна зробити висновок, що класифікація архітектурних стилів на основі зображень фасадів є складною задачею комп’ютерного зору, яка потребує врахування міжкласової схожості, внутрішньокласової варіативності, впливу умов зйомки та, найчастіше, обмеженої кількості розмічених даних для навчання моделей. Вирішення таких задач вимагає поєднання сучасних методів глибинного навчання з оптимізацією архітектури нейронних мереж під специфіку вхідних даних.

У цьому контексті основним завданням бакалаврської кваліфікаційної роботи є розробка програмної моделі для автоматичної класифікації архітектурних стилів будівель за зображеннями фасадів з використанням підходів глибинного навчання. Основою запропонованої моделі є згорткова нейронна мережа ResNet50, попередньо навчена на наборі зображень ImageNetV2 – покращеній версії класичного ImageNet, яка забезпечує кращу відповідність до сучасних реальних даних і дає змогу більш об'єктивно оцінювати здатність моделей до узагальнення.

Застосування підходу transfer learning дозволяє адаптувати попередні знання моделі до нової, вузькоспеціалізованої задачі класифікації стилів архітектури за допомогою донавчання на обмеженій вибірці зображень. Такий підхід дає змогу ефективно працювати навіть за умови недостатньої кількості навчальних прикладів та високої візуальної схожості між окремими класами.

РОЗДІЛ 2.
МОДЕЛЬ КЛАСИФІКАЦІЇ СТИЛІВ БУДІВЕЛЬ НА ЗОБРАЖЕННЯХ

2.1 Залишкове навчання й архітектура ResNet

У 2015 році дослідники з *Microsoft Research Asia* [28], представили новаторську роботу "Deep Residual Learning for Image Recognition", у якій було запропоновано архітектуру ResNet на основі ідеї залишкового (резидуального) навчання (*Residual Learning*). Residual у перекладі з англійської означає похибка. Похибка або залишок знаходиться віднявши від дійсного значення, значення яке було передбачено:

$$e = y - \hat{y}. \tag{2.1}$$

Значення цього залишку показує наскільки прогноз був далеким від дійсності.

Вчені виявили, що просте збільшення кількості згорткових шарів не завжди призводить до покращення точності: хоча на початку глибші моделі демонструють кращі результати, з певної глибини починається деградація якості – точність знижується, а навчання ускладняється.

Щоб подолати цю проблему, автори запропонували підхід, у якому замість прямої апроксимації складної цільової функції *H(x)*, мережа навчається апроксимувати залишкову функцію $F(x) = H(x) - x$. Таким чином, вихід обчислюється як:

$$H(x) = F(x) + x, \tag{2.2}$$

де *F(x)* – залишкове перетворення (*residual mapping*), яке виконується послідовністю згорткових операцій, *x* – вхідний тензор, *H(x)* – вихід залишкового блоку.

ResNet (Residual Network) – це глибока згорткова нейронна мережа, яка використовує залишкові блоки (residual blocks) як основні будівельні елементи. Стандартна модель ResNet будується з повторення таких блоків, кожен з яких

виконує операції згортки, нормалізації та нелінійної активації, а також містить прямий (shortcut) шлях для передачі вхідного сигналу (рис. 2.1).

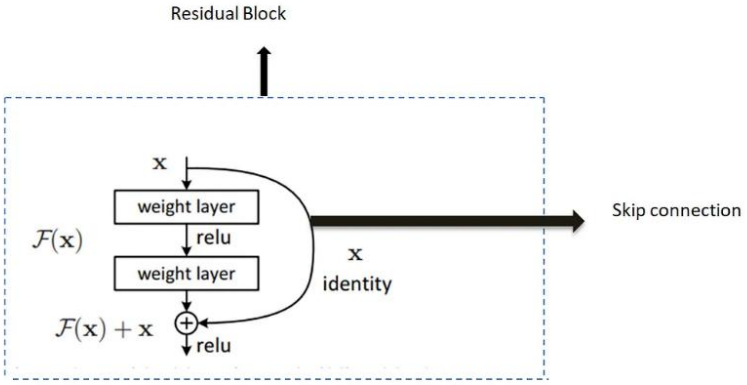


Рисунок 2.1 – Архітектура ResNet

Основу архітектури ResNet становлять *залишкові блоки*, які дозволяють ефективно навчати дуже глибокі згорткові нейронні мережі. Ключова ідея полягає в додаванні вхідного сигналу x до виходу залишкового перетворення $F(x)$, що реалізується у вигляді прямого з'єднання (skip connection або identity mapping). Така структура полегшує поширення градієнтів у зворотному проходженні, зменшуючи проблему затухання градієнтів у глибоких мережах [29].

У типовому залишковому блоці, наприклад у ResNet-34, $F(x)$ визначається як:

$$F(x) = W_2 \cdot \sigma(W_1 \cdot x), \tag{2.3}$$

де W_1, W_2 – матриці ваг згорткових шарів, σ – функція активації (зазвичай ReLu).

Якщо розмірність вхідного тензора змінюється (наприклад, через зміну кількості каналів), пряме додавання x до $F(x)$ неможливе. У таких випадках використовується лінійне перетворення (projection shortcut) за допомогою додаткового згорткового шару з вагами W_s:

$$H(x) = F(x) + W_s \cdot x. \quad (2.4)$$

Це дозволяє адаптувати залишкове з'єднання до зміненої розмірності тензорів, зберігаючи ключову ідею ResNet.

Також ключовим елементом ResNet є *з'єднання швидкого доступу (skip connection*) – пряме з'єднання між вхідним сигналом та виходом шару, яке реалізується як тотожне відображення (identity mapping). Тобто вхід x передається паралельно через стік звичайних згорткових шарів та напряму додається до їх виходу F(x). Якщо новий шар не приносить користі, його вплив нівелюється, і глибока модель не деградує. Така стратегія дозволяє ефективно тренувати мережі з понад сотнею шарів. Такий підхід дозволяє зберігати та передавати інформацію без спотворення навіть у дуже глибоких мережах, зменшуючи вплив згасаючих градієнтів і спрощуючи оптимізацію

Архітектура ResNet створюється шляхом послідовного накладання великої кількості залишкових блоків (*Stacked layers*). Така шарова структура забезпечує можливість ефективного конструювання надзвичайно глибоких нейронних мереж. Завдяки цій модульній побудові були реалізовані глибокі варіанти ResNet, зокрема ResNet-50, ResNet-101 і ResNet-152, які містять відповідно 50, 101 і 152 шари.

У ResNet-архітектурах перед вихідним повнозв'язним шаром зазвичай застосовується глобальний усереднюючий pooling (*Global Average Pooling, GAP*). Ця операція замінює кожну карту ознак її середнім значенням, зменшуючи просторові розміри до одного скалярного значення на канал. Таким чином, модель отримує компактне узагальнення інформації без необхідності використання великої кількості параметрів.

На відміну від Max Pooling, який зберігає лише найбільше значення в області, GAP враховує усі значення пікселів, що сприяє стабільнішому навчанню та зменшує ризик перенавчання.

2.2 Приклад архітектури залишкової мережі

На схемі (Додаток Д, рис. Д.1) зображено порівняння мереж VGG-19, звичайної 34-шарової нейронної мережі та залишкової мережі з 34 шарами. Загалом залишкова мережа включає 16 залишкових блоків.

Розглянемо кожний набір блоків окремо. Перший набір (рис. 2.2) містить 3 залишкові блоки. Кожен з них включає дві згорткові операції з 64 фільтрами розміром 3×3 та прямим з'єднанням (skip connection), що виконує тотожне відображення (identity mapping).

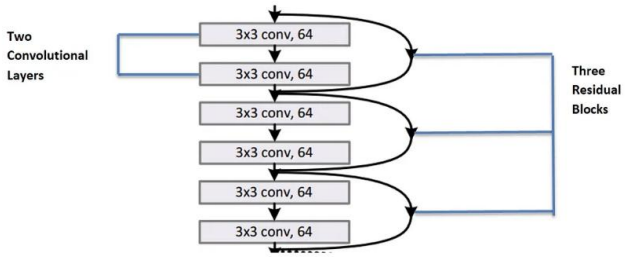


Рисунок 2.2 – Три залишкові блоки

Другий набір (рис. 2.3) складається з 4 залишкових блоків, кожен з яких також має по два згорткові шари, але з 128 фільтрами 3×3, і аналогічним прямим з'єднанням. Пунктирні з'єднання на схемі позначають ті випадки, коли змінюється розмірність (наприклад, кількість каналів зростає). У таких ситуаціях необхідно узгодити розміри вхідного та вихідного тензора. Якщо розмірність змінюється, згортка виконується зі страйдом 2 (тобто посунеться на два пікселі при зчитувані зображення).

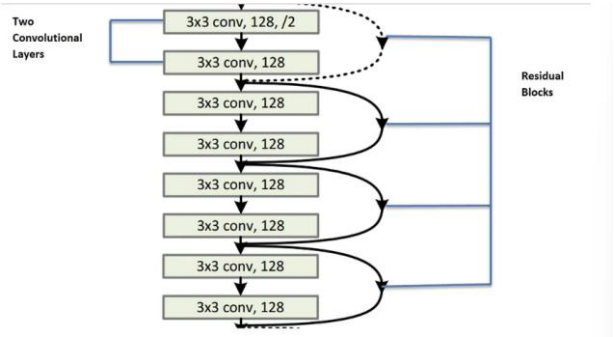


Рисунок 2.3 – Чотири залишкові блоки

Третій набір (рис. 2.4) складається з 6 залишкових блоків, кожен з яких має по два згорткових шари з 256 фільтрами розміром 3×3.

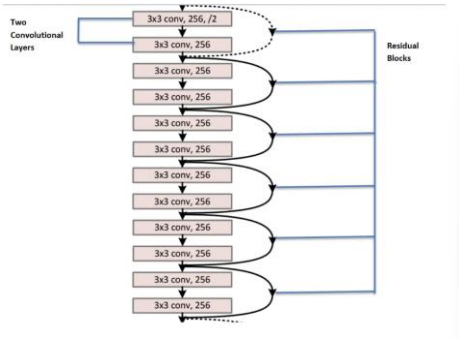


Рисунок 2.4 – Шість залишкові блоки

Четвертий набір (рис. 2.5) включає 3 залишкові блоки, де кожен містить по два згорткових шари з 512 фільтрами 3×3.

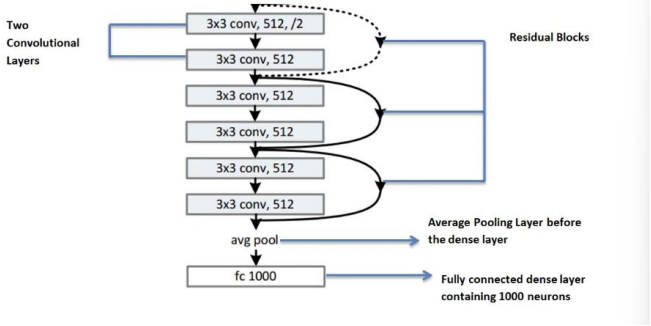


Рисунок 2.5 – Останні залишкові блоки

Після завершення всіх залишкових блоків, отримана карта ознак проходить через шар глобального усереднюючого пулінгу (Global Average Pooling), а далі – через повнозв’язний шар із 1000 нейронів, який виконує класифікацію по 1000 класах.

2.3 Архітектура ResNet-50

Для створення архітектури версії ResNet-50 було замінено кожний 2-х шаровий блок на 34-шарову мережу з трьома боттлнек-блоками (bottleneck blocks).

Bottleneck-блок – це модифікований тип залишкового блоку, в якому застосовуються згортки 1×1 для зменшення розмірності перед основною згорткою 3×3. Таке звуження дозволяє значно скоротити кількість параметрів і обчислень без втрати виразності моделі. Головна ідея – зробити блоки вузькими всередині, щоб збільшити загальну глибину мережі при обмеженій складності.

Ці блоки були вперше запропоновані в оригінальній архітектурі ResNet і стали ключовими для побудови глибших моделей, таких як ResNet-50 та ResNet-101.

Архітектура ResNet-50 [30] умовно поділяється на шість логічних етапів і охоплює три рівні глибини (рис. 2.6):

– попередня обробка вхідного зображення (Input Pre-processing);
– Cfg[0] blocks;
– Cfg[1] blocks;
– Cfg[2] blocks;
– Cfg[3] blocks;
– завершальний повнозв’язний шар (Fully-connected layer) для класифікації.

Cfg[i] (від "configuration") – умовне позначення групи боттлнек-блоків із фіксованими характеристиками, що використовуються в певному сегменті архітектури ResNet-50.

Кожна така група має однакову кількість фільтрів на всіх згортках у межах групи, однакову кількість боттлнек-блоків, фіксовану стратегію зміни розмірності.

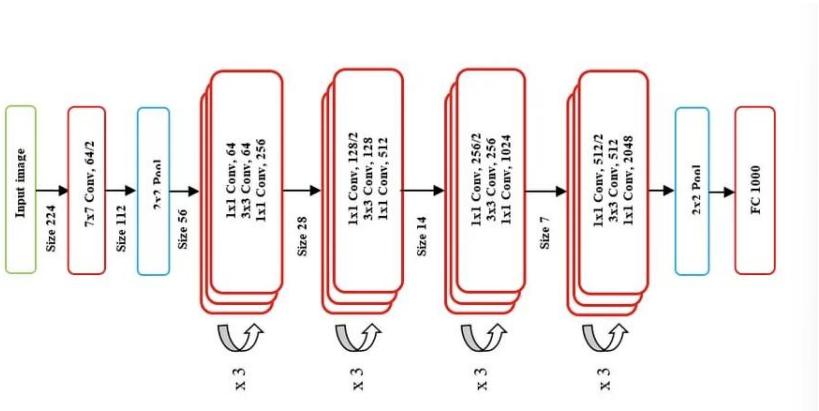


Рисунок 2.6 – Схема архітектури ResNet-50

Різні версії ResNet використовують різну кількість блоків Cfg та на різних рівнях. На рис. 2.7 наведена таблиця архітектур кожної версії.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				

Рисунок 2.7 – Архітектури версій ResNet

2.4 Порівняння звичайної та залишкової нейронної мережі

У звичайній багатошаровій мережі послідовність шарів породжує складене відображення. Наприклад, три шари з функціями h, g, f дають $y = f(g(h(x)))$. У ResNet кожен блок додає до цього відображення ідентичний шлях: вихід блоку часто описують як $y = x + f(g(x))$. Тобто в ResNet результат шару складається з суми вузлової функції $f(g(x))$ та самого входу x.

Теоретично це означає, що ResNet-модель здатна відобразити ширший простір функцій, причому цей простір є вкладеним: ResNet може повторювати поведінку менш глибокої мережі, навчивши блоки тотожності. Як зазначається у книзі "Dive into Deep Learning", якщо нові шари навчаться ідентичності, то збільшення глибини не погіршує апроксимацію і навіть спрощує зниження похибки [31].

На рис. 2.9 [28] наведено порівняльну таблицю результатів класифікації зображень для звичайної (plain) та залишкової (ResNet) мереж із різною кількістю шарів. З таблиці видно, що для звичайної мережі збільшення кількості шарів призводить до погіршення результату (зростає частка неправильних відповідей), тоді як для ResNet — навпаки, збільшення глибини дозволяє зменшити кількість помилок.

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Рисунок 2.8 – Порівняння частки неправильних відповідей на зображеннях для звичайної та залишкової мережі при різній глибині

Висновки до розділу 2

Архітектурні стилі мають складні локальні та глобальні ознаки: форма арок, текстура фасаду, орнамент, ритм вікон, співвідношення поверхів. ResNet50, завдяки глибокій ієрархії фільтрів та збереженню ідентичності через shortcut-з’єднання, здатна ефективно виділяти як низькорівневі, так і високорівневі ознаки. Додатково, її поширене використання у комп’ютерному зорі (ImageNet, COCO) свідчить про її надійність у класифікаційних задачах.

Таким чином, обраний підхід на основі ResNet50 дозволяє досягти балансу між точністю класифікації, стійкістю до глибини мережі та можливістю подальшого розширення архітектури при потребі.

РОЗДІЛ 3.
РЕАЛІЗАЦІЯ МОДЕЛІ КЛАСИФІКАЦІЇ СТИЛІВ БУДІВЕЛЬ

3.1 Аналіз вимог до програмного продукту

Вимоги до функціональних характеристик. Система має надавати користувачу такі функціональні можливості:

а) програмний виріб повинен автоматично виконувати класифікацію архітектурних стилів на основі вхідного зображення;

б) програмний виріб має підтримувати обробку зображень у форматах .jpg, .jpeg та .png;

в) користувач повинен мати можливість завантажити зображення, вказавши шлях до файлу або надавши шлях до папки з зображеннями, якщо йдеться про обробку великої кількості зображень;

г) система має забезпечувати класифікацію зображення за одним із 10 архітектурних стилів: American Craftsman, Art Deco, Art Nouveau, Baroque, Beaux-Arts, Georgian, Japanese Traditional, Queen Anne's, Russian Revival, Tudor Revival;

д) після класифікації має виводитись назва визначеного стилю;

е) максимальний обсяг оперативної пам'яті, що використовується під час класифікації одного зображення, не повинен перевищувати 500 МБ;

є) час inference (обробки) одного зображення розміром 224×224 пікселів не перевищує 3 с на ПК із GPU NVIDIA GTX 1050 та відповідними оптимізаціями.

Вимоги до надійності. Система має відповідати наступним вимогам надійності:

а) модель має бути стійкою до невеликих змін у зображенні (варіації освітлення, шум);

б) обробка винятків у випадку пошкоджених файлів або некоректного формату вхідних даних;

в) система не повинна аварійно завершуватись у випадку проблеми із зображенням;

г) програмний виріб має зберігати цілісність моделей та вихідних даних навіть у разі збоїв у роботі.

Вимоги до складу і параметрів технічних засобів:

а) програма повинна підтримувати виконання на персональних комп’ютерах під операційними системами Windows (7 і вище), Linux (будь-які поширені дистрибутиви) та macOS (10.13 і вище). Для запуску необхідний інтерпретатор Python версії 3.6 або вище;

б) для користування програмою мінімальними вимогами є підключення комп'ютера до електромережі, мінімальна оперативна пам'ять 8 ГБ, дисковий простір 1 ГБ, процесор 3.0-3.5 ГГц;

в) для прискорення обчислень рекомендується використовувати дискретний графічний процесор (GPU) з підтримкою CUDA (версія 10.2 або новіша), наприклад NVIDIA GTX 1050 або вище.

3.2 Засоби програмної розробки

При реалізації моделі було використано мову Python, яка є однією з популярних мов для задач машинного навчання. Python забезпечує високий рівень абстракції та широкий спектр спеціалізованих бібліотек, що є критично важливим для ефективного впровадження алгоритмів комп'ютерного зору та глибокого навчання.

Для ефективної розробки та функціонування системи класифікації архітектурних стилів на основі архітектури ResNet50 було задіяно набір інструментальних засобів та технологій, який формує єдине інтегроване середовище, що забезпечує повний цикл розробки та функціонування системи автоматизованої класифікації архітектурних стилів, дозволяючи досягти високої точності розпізнавання при прийнятних обчислювальних витратах. Роботу виконано в ізольованому середовищі (venv) з установленими необхідними пакетами. Робоче середовище налаштовано в редакторі коду Visual Studio Code з використанням WSL (Windows Subsystem for Linux) — компонента Windows, який дає змогу запускати повноцінне середовище Linux безпосередньо у

Windows, що забезпечує зручність роботи з Linux-інструментами в межах однієї операційної системи.

3.2.1 Фреймворки глибокого навчання

В якості основного фреймворку глибокого навчання було обрано PyTorch, що обумовлено його гнучкістю при конструюванні нейромережевих архітектур та динамічним підходом до обчислювального графу. Дана система забезпечує ефективні тензорні обчислення з можливістю прискорення на графічних процесорах, що суттєво скорочує час навчання складних нейронних мереж.

Для роботи з попередньо навченими моделями, зокрема ResNet50, використовувалася бібліотека torchvision, яка містить імплементації стандартних архітектур та забезпечує доступ до багатофункціонального інтерфейсу трансформацій зображень. У роботі ResNet-50 була навчена IMAGNETIC V2 – оновленій версії оригінального датасету ImageNet, яка краще відображає реальні дистрибуції даних. Це дозволяє моделі краще узагальнювати нові зображення. Моніторинг та візуалізація процесу навчання здійснювалися за допомогою інструментарію TensorBoard, що дозволило відслідковувати динаміку навчання та аналізувати ключові метрики продуктивності моделі у реальному часі.

3.2.2 Засоби обробки зображень

Обробка вхідних зображень архітектурних споруд реалізована з використанням бібліотеки OpenCV (cv2), яка надає широкий інструментарій для маніпуляцій із зображеннями, включаючи масштабування, нормалізацію та підвищення якості вхідних даних. Дана бібліотека відіграє ключову роль у підготовчому етапі обробки даних, що безпосередньо впливає на ефективність класифікації.

Для реалізації числових операцій та маніпуляцій з багатовимірними масивами даних застосовано бібліотеку NumPy, що забезпечує високопродуктивні обчислення, необхідні для попередньої та проміжної обробки зображень.

3.2.3 Зберігання та оптимізація моделі

З метою забезпечення кросплатформності розробленої моделі та оптимізації її продуктивності під час експлуатації було застосовано технологію ONNX (Open Neural Network Exchange). Дана технологія дозволяє конвертувати натреновану модель у стандартизований формат, що забезпечує її сумісність із різними програмними та апаратними середовищами.

Для забезпечення високоефективного виконання інференційних операцій було інтегровано ONNX Runtime, що надає оптимізоване середовище виконання для моделей у форматі ONNX, суттєво підвищуючи швидкодію системи класифікації. Використання ONNX було обумовлено потребою у забезпеченні кросплатформності та незалежності від фреймворків. Це дозволяє використовувати модель не лише в середовищі Python/PyTorch, але й у вебдодатках, мобільних пристроях або вбудованих системах з підтримкою ONNX Runtime. Крім того, ONNX дозволяє оптимізувати модель для інференції, що суттєво скорочує час обробки під час практичного застосування.

3.2.4 Інструменти машинного навчання

У процесі розробки та оцінки ефективності моделі активно використовувалась бібліотека scikit-learn, за допомогою якої було реалізовано методології крос-валідації, оцінки якості та підбору оптимальних гіперпараметрів моделі. Застосування даного інструментарію дозволило досягти значного підвищення точності класифікації та уникнути проблем перенавчання.

3.3 Проєктування моделі класифікації архітектурних стилів

3.3.1 Підготовка навчальних даних

У роботі використано спеціалізований набір зображень архітектурних споруд, систематизованих за їх стильовими характеристиками. Експериментальна база дослідження сформована на основі відкритого датасету, отриманого з репозитарію платформи Kaggle [32]. З представленого у вихідному наборі даних розмаїття архітектурних стилів було селективно обрано 10 репрезентативних категорій, частина з яких характеризується значною

візуальною та концептуальною спорідненістю, що створює додаткові виклики для алгоритмів машинного розпізнавання і підвищує науково-практичну цінність дослідження.

Детальну структурну композицію і кількісні характеристики датасету описано у таблиці 3.1 та візуалізовано на діаграмі (рис. 3.1) і на графіку (рис. 3.2).

На рисунку 3.2 зліва зображено графік розподілу ширини та висоти зображень у вибірці: кожна точка відповідає окремому зображенню, де по горизонталі відкладено ширину, а по вертикалі — висоту. Медіанні значення становлять 539 пікселів для ширини та 405 пікселів для висоти, що вказує на типовий розмір зображень у датасеті. Праворуч наведено приклади зображень архітектурних об’єктів, які відібрані як з тренувальної, так і з тестової вибірки. Це ілюструє різноманіття класів та візуальних особливостей, що враховуються при навчанні та перевірці моделі. Набір даних цілеспрямовано орієнтований на розв'язання задачі закритого типу (Closed set classification), метою якої є віднесення вхідного зображення до одного з попередньо визначених класів.

Таблиця 3.1

Опис класів з набору даних

Клас	Кількість зображень
American Craftsman	200
Art Deco	200
Art Nouveau	200
Baroque	126
Beaux-Arts	200
Georgian	130
Japanese Traditional	200
Queen Annes	155
Russian Revival	200
Tudor Revival	200
Загальна кількість	1811

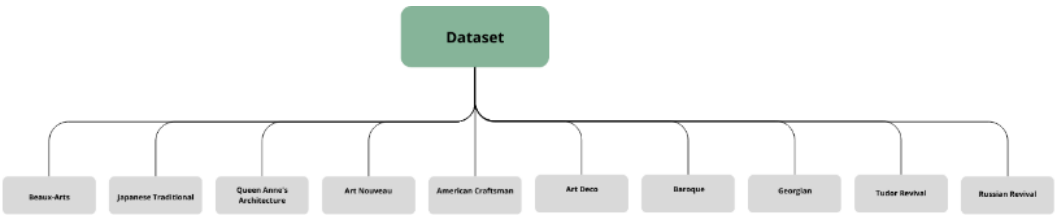


Рисунок 3.1 – Структура датасету

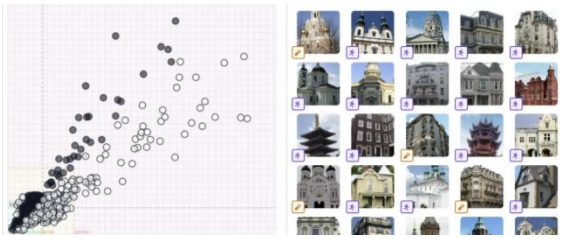


Рисунок 3.2 – Візуалізація класів зображень

Архітектура системи передбачає можливість подальшої модифікації шляхом включення додаткових стильових категорій або виключення існуючих без необхідності кардинальної реструктуризації моделі.

Необхідно зазначити, що розроблена модель має певні функціональні обмеження, зумовлені специфікою підходу до навчання, при якому система орієнтована на виявлення характерних візуальних ознак, що дозволяють ідентифікувати архітектурний стиль серед заданого переліку альтернатив. Модель здійснює класифікацію зображень, які не були представлені у навчальній вибірці, на основі узагальнених патернів, екстрагованих з тренувальних даних, що відповідає сучасній парадигмі машинного навчання та комп'ютерного зору.

3.3.2 Розподіл на навчальну, валідаційну та тестову вибірки

У процесі розробки системи класифікації архітектурних стилів критичним етапом є коректна організація вхідних даних. Вхідний набір зображень було попередньо структуровано за категоріями архітектурних стилів, де кожен стиль представлено відповідним підкаталогом. Для забезпечення належної оцінки ефективності моделі було застосовано принцип розподілу даних на незалежні вибірки:

– тренувальний набір використовується безпосередньо для навчання параметрів нейронної мережі;

– валідаційний набір застосовується для контролю процесу навчання та запобігання перенавчанню моделі;

– тестовий набір застосовується безпосередньо для тестування моделі, оцінюючи її здатність генералізувати вивчені риси на ще небачених зображеннях.

Варто зазначити, що валідаційний набір було створено програмно, розділивши тренувальний набір на 80% тренувальний і 20% валідаційний із застосуванням функції random_split бібліотеки PyTorch, що забезпечує випадковий характер розподілу з дотриманням фіксованого співвідношення між вибірками та гарантує, що модель не тренується на валідаційних даних.

3.3.3 Проблема незбалансованості даних

Навчальний набір зображень складається з 76-150 зображень для кожного стилю для тренувальних даних і 50 зображень для кожного стилю для тестувальних даних. Але, як вже було сказано, у майбутньому є вірогідність додавання нових класів для навчання моделі або просто заради розширення розмір датасету. Тобто існує суттєвий виклик при роботі з реальними наборами даних є потенційна незбалансованість класів, що може призвести до зміщення моделі у бік більш представлених категорій. Для мінімізації негативного впливу цього фактора у розробленій системі було впроваджено підхід зваженої вибірки.

1. На першому етапі здійснюється аналіз розподілу класів шляхом підрахунку кількості зразків у кожній категорії.

2. На основі отриманого розподілу обчислюються вагові коефіцієнти для кожного класу, причому класи з меншою представленістю отримують вищі коефіцієнти. Застосування квадратного кореня у формулі забезпечує більш збалансований підхід, уникаючи надмірної компенсації для слабо представлених класів.

3. Отримані вагові коефіцієнти застосовуються для формування спеціалізованого семплера WeightedRandomSampler, який забезпечує коректування частоти появи зразків різних класів під час тренування.

Наведений нижче код відображає описаний підхід зваженої вибірки.

```
        from torch.utils.data import WeightedRandomSampler

        class_counts = Counter(train_labels)
        class_weights = {cls: 1.0 / np.sqrt(count) for cls, count in
class_counts.items()}
        sample_weights = [class_weights[label] for label in train_labels]

        sampler = WeightedRandomSampler(weights=sample_weights,
num_samples=len(sample_weights), replacement=True)
```

Даний підхід дозволяє моделі приділяти належну увагу всім класам, незалежно від їх представленості у вихідному наборі даних.

3.3.4 Аугментація даних

Аугментація даних є методом штучного розширення набору даних шляхом створення варіацій існуючих зразків без залучення нових даних. Ця методика особливо корисна у випадках, коли обсяг доступної інформації обмежений або містить користувацький контент.

Основні методи аугментації включають обрізання (cropping), додавання полів (padding), віддзеркалення (flipping), зміни кольорової гами, а також внесення шуму та спотворень, що сприяє підвищенню стійкості моделі до варіативності вхідних даних.

Для розширення навчальної вибірки, підвищення узагальнювальної здатності моделі та подолання проблеми обмеженості даних було впроваджено комплексну систему аугментації зображень. Застосовано різноманітні методи трансформації, зокрема RandomResizedCrop, RandomHorizontalFlip, RandomAffine, ColorJitter, GaussianBlur, EdgeSharpen, RandomPerspective та RandomAutocontrast. Основний принцип полягає в тому, що такі випадкові модифікації запобігають надмірній залежності моделі від окремих візуальних характеристик (наприклад, точного розташування об'єкта чи кольорової гами), що сприяє кращому узагальненню. Внаслідок цього модель демонструє підвищену стійкість до варіацій вхідних даних і меншу схильність до перенавчання [33].

Важливою особливістю запропонованого підходу є застосування диференційованої стратегії аугментації. Під час аналізу результатів початкового тренування за допомогою матриці помилок (confusion matrix) було виявлено, що модель систематично неправильно класифікувала об'єкти певних класів через їхню візуальну подібність. Наприклад, стилі Georgian та Tudor Revival характеризуються подібною симетрією фасадів, а Baroque та Beaux-Arts мають схожі орнаментальні елементи.

Зважаючи на це, було розроблено двоетапну систему аугментації:

1. Стандартна аугментація – базовий набір трансформацій, що застосовується до всіх класів

```
basic_transforms = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean, std)
])
```

2. Посилена аугментація – розширений комплекс трансформацій, спрямований на класи, які модель найчастіше плутала між собою

```
strong_transforms = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(),
    transforms.RandomAffine(degrees=20),
    transforms.RandomPerspective(),
    transforms.GaussianBlur(3),
    transforms.ToTensor(),
    transforms.Normalize(mean, std)
])
```

3.3.5 Налаштування параметрів навчання

Для ефективного навчання нейронної мережі критичним є вибір оптимізаційного алгоритму, який керує процесом оновлення вагових

коефіцієнтів під час мінімізації функції втрат. Серед сучасних методів оптимізації для глибоких нейронних мереж особливо виділяється Adam (Adaptive Moment Estimation) — адаптивний оптимізатор, що поєднує переваги алгоритмів AdaGrad та RMSProp. Його ефективність зумовлена здатністю встановлювати індивідуальні темпи навчання для кожного параметра на основі оцінки першого та другого моментів градієнтів, що забезпечує швидку та стабільну збіжність.

У роботі застосовано вдосконалену модифікацію — AdamW, яка відрізняється від класичного Adam реалізацією відокремленої регуляризації вагових коефіцієнтів (decoupled weight decay). Це суттєво покращує генералізаційні властивості моделі та підвищує її стійкість до перенавчання. Математично це можна представити наступним чином:

$$\theta = \theta - \eta \cdot AdamUpdate(\theta) - \eta \cdot \lambda\theta. \quad (3.1)$$

Принципова відмінність від традиційного підходу полягає в тому, що регуляризація вагових коефіцієнтів відокремлена від процесу оновлення градієнтів. Спочатку обчислюється оновлення параметрів за допомогою алгоритму Adam на основі чистого градієнта, а потім окремо застосовується weight decay – зменшення вагових коефіцієнтів пропорційно до їхніх поточних значень [34].

Для вирішення задачі багатокласової класифікації обрано функцію втрат Cross-Entropy Loss, яка є оптимальним вибором для таких завдань. Ця функція кількісно визначає розбіжність між розподілом імовірностей, прогнозованим моделлю, та істинним розподілом, представленим у форматі one-hot кодування. В окремих експериментальних конфігураціях було також імплементовано техніку пом'якшення міток (label smoothing), що запобігає надмірній упевненості моделі у власних передбаченнях і, як наслідок, сприяє підвищенню її узагальнювальної здатності.

3.4 Тренування моделі класифікації

Для тренування моделі було використано підхід трансферного навчання (transfer learning) із поступовим розморожуванням шарів попередньо натренованої архітектури ResNet50. Такий підхід дозволяє адаптувати знання, отримані на великому датасеті ImageNet, до конкретного завдання класифікації архітектурних стилів.

Навчання проводилось у стабільному обчислювальному середовищі з однаковими апаратними та програмними умовами для всіх експериментів. Щоб уникнути зовнішніх впливів на результат, фонові процеси були вимкнені, що дозволило точно порівнювати продуктивність моделей. Основними критеріями ефективності навчання були:

- точність класифікації на навчальній та валідаційній вибірках;
- функція втрат (loss function) на цих же вибірках.

На рисунках 3.3, 3.5, 3.7 і 3.9 показані графіки зміни точності та значення функції втрат протягом епох тренування як для тренувального, так і для валідаційного набору. Додатково представлені матриці помилок (confusion matrices), що відображають результати класифікації кожного класу та дозволяють виявити слабкі місця моделі (рис. 3.4, 3.6, 3.8).

Кожна конфігурація тренувалася протягом 30 епох. Для запобігання перенавчанню (overfitting) застосовано регуляризацію Dropout з імовірністю 0.5. Середнє завантаження графічного процесора (GPU) під час тренування складало приблизно 85%, що свідчить про раціональне використання обчислювальних ресурсів.

Нижче наведено приклад програмного коду для розморожування одного з блоків моделі в середовищі PyTorch:

```
for param in model.layer2.parameters():
        param.requires_grad = True
```

На початковому етапі всі шари були заморожені (requires_grad = False), окрім класифікатора. Це дало змогу швидко адаптувати модель до нових класів без необхідності перенавчання всієї архітектури.

Спочатку було проведено тренування моделі з *розмороженим лише повнозв'язним шаром* (fully connected layer). У процесі навчання модель продемонструвала значне підвищення точності на валідаційному наборі – з 54.61% до 71.21%, а на тренувальному – з 55.35% до 81.27%. Кінцеві значення функції втрат склали 0.5874 для тренувального набору та 0.8632 для валідаційного.

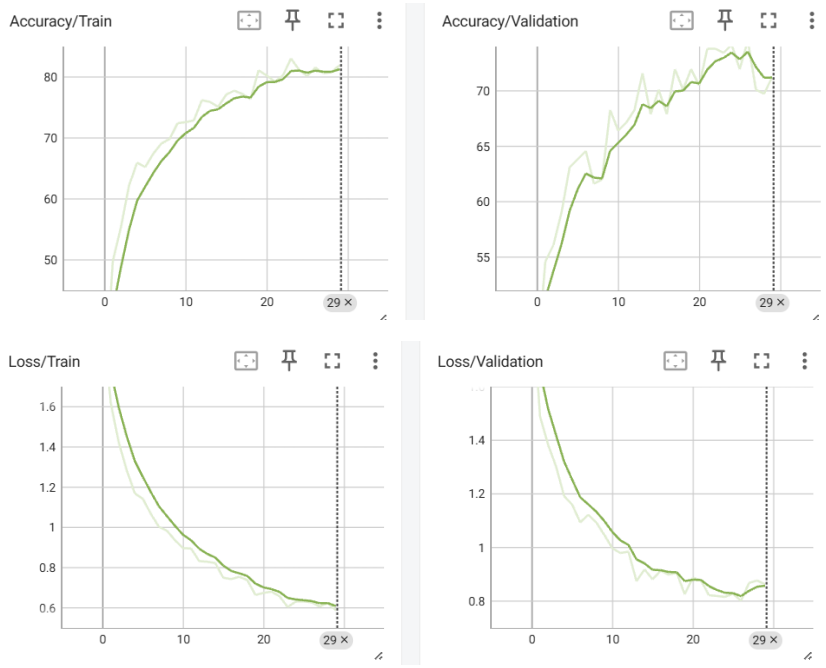


Рисунок 3.3 – Результати тренування з розмороженим повнозв'язним шаром

Confusion matrix на рис. 3.4 свідчить про переважання правильних передбачень на діагоналі, що вказує на загалом високу точність. Це підтверджує правильність обраної стратегії навчання. Найбільше помилок спостерігалося при класифікації класу Queen Anne’s, що може свідчити про складність цього стилю для моделі.

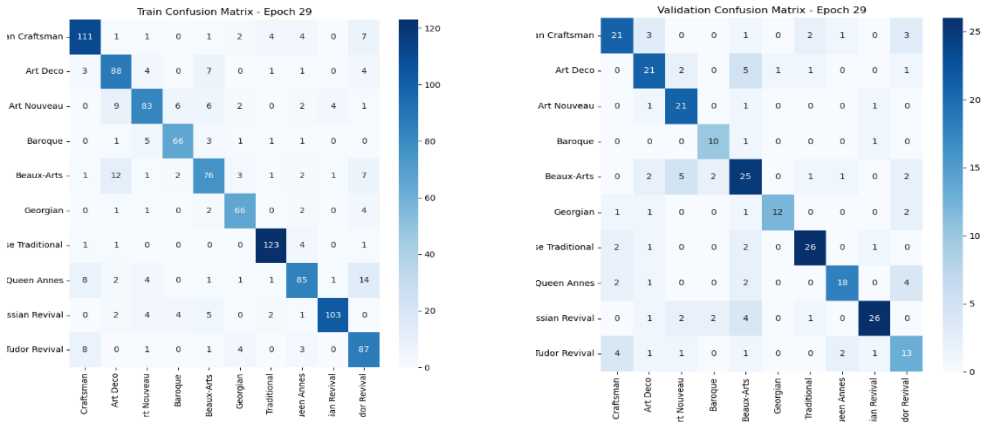


Рисунок 3.4 – Матриці помилок в результаті тренування з розмороженим повнозв'язним шаром

Наступним етапом було *розморожування найвищого блоку мережі* (*layer4*). Результати тренування продемонстрували значне покращення: точність на тренувальному наборі досягла 94.98%, а на валідаційному – 85.43%. Відповідні значення функції втрат становили 0.1699 для тренувального та 0.4325 для валідаційного наборів. Такі показники свідчать про суттєве підвищення ефективності моделі порівняно з попередньою конфігурацією.

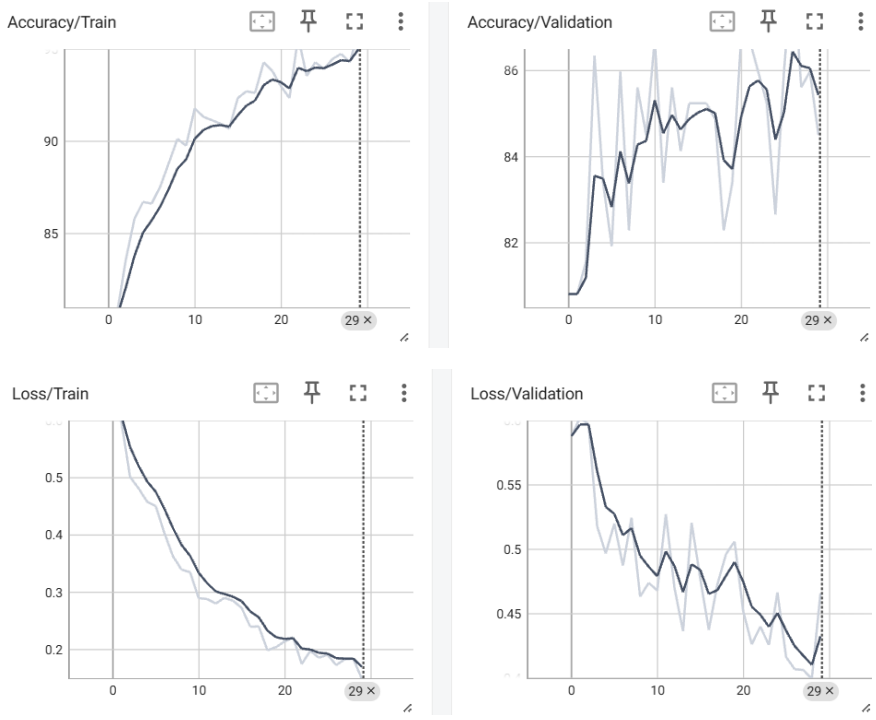


Рисунок 3.5 – Результати тренування з розмороженим шаром 4

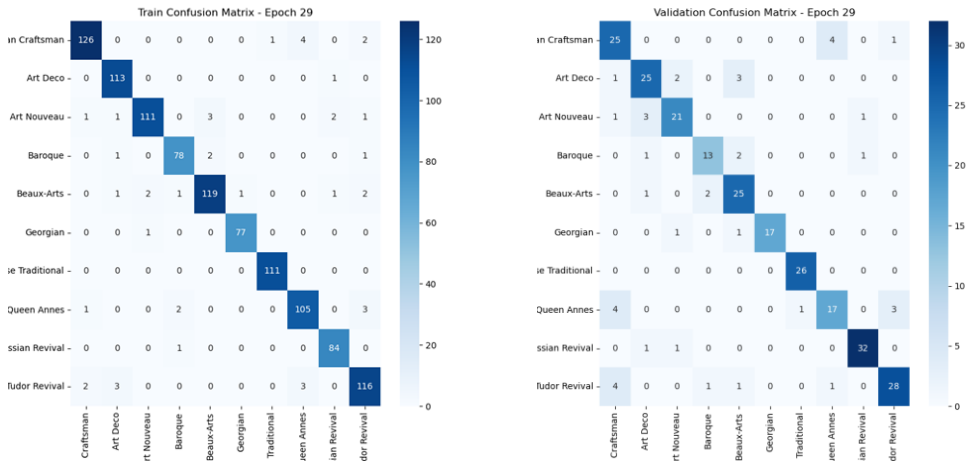


Рисунок 3.6 – Матриці помилок в результаті тренування з розмороженим шаром 4

При *розморожуванні layer3* спостерігалося подальше підвищення продуктивності моделі. Точність на тренувальному наборі склала 93.97%, тоді як на валідаційному досягла вражаючих 97.01%. Значення функції втрат становили 0.1899 для тренувального та 0.805 для валідаційного наборів. Особливо важливо відзначити, що валідаційна точність перевищила тренувальну, що є нетиповим явищем і може свідчити про виняткову здатність моделі до узагальнення на нових даних.

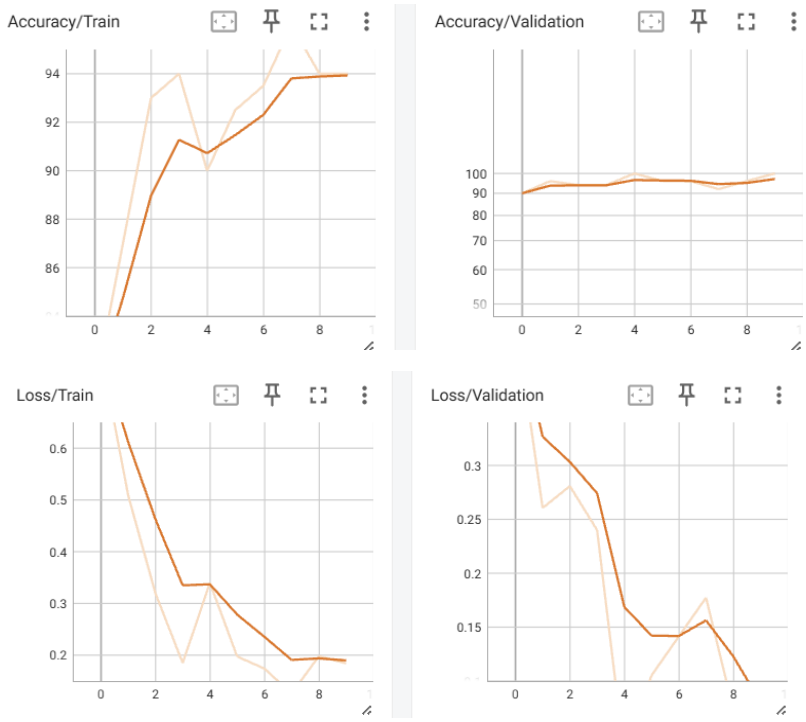


Рисунок 3.7 – Результати тренування з розмороженим шаром 3

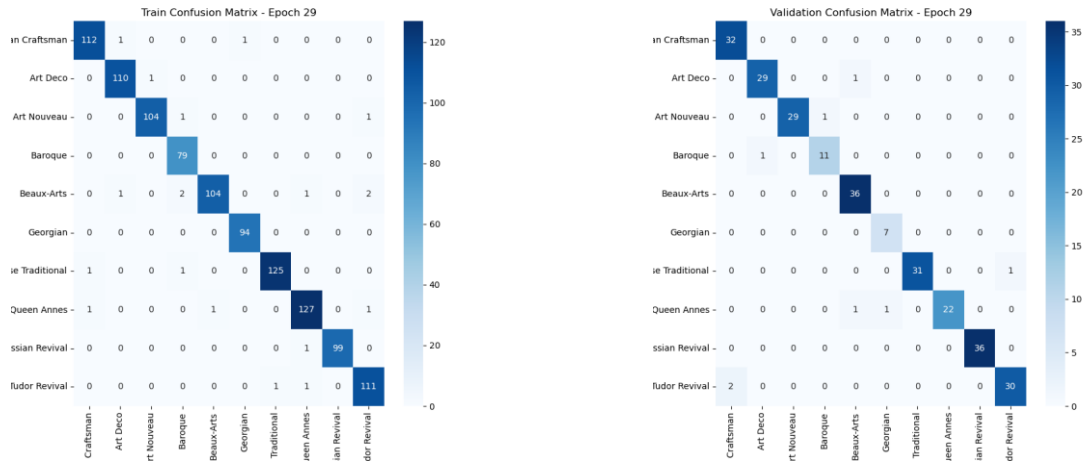


Рисунок 3.8 – Матриці помилок в результаті тренування з розмороженим шаром 3

Після *розморожування layer2* точність залишалася стабільно високою на обох наборах даних, досягаючи показника 98%. Втрати були мінімальними, а навчання було завершене вже після двох епох, оскільки показники як на тренувальному, так і на валідаційному наборах майже не змінювалися. Це свідчить про те, що модель досягла стелі своїх можливостей у рамках цієї архітектури та набору даних.

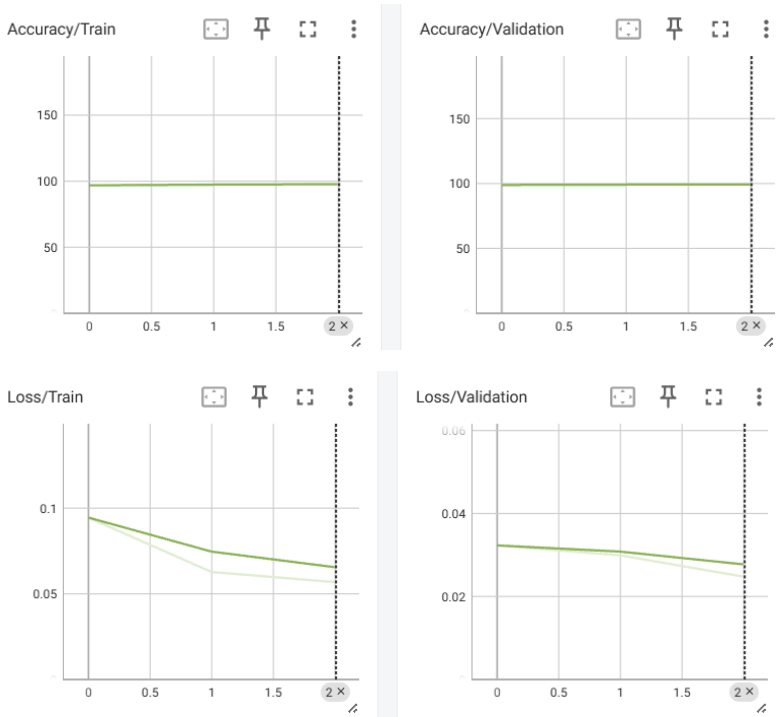


Рисунок 3.9 – Результати тренування з розмороженим шаром 2

Модель тренувалася по 30 епох для кожної конфігурації, де одна епоха відповідає повному проходженню мережі через весь набір навчальних даних. Це дозволяє моделі ідентифікувати та вивчити характерні ознаки зображень різних архітектурних стилів.

Перше повне тренування тривало близько двох годин, тоді як наступні ітерації відбувалися значно швидше. Це відносно ефективний результат, враховуючи кількість шарів та обсяг зображень, які необхідно було обробити.

На початкових етапах тренування мережа демонструвала відносно високі значення функції втрат та низькі значення точності (рис. 3.10). Проте в процесі навчання ці показники стабільно покращувалися, що свідчить про поступове вдосконалення параметрів моделі (рис. 3.11).

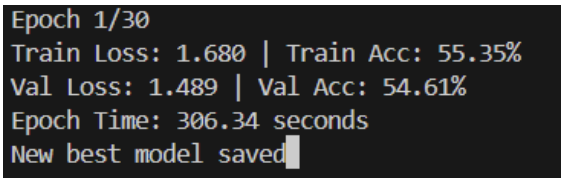


Рисунок 3.10 – перші кроки тренування

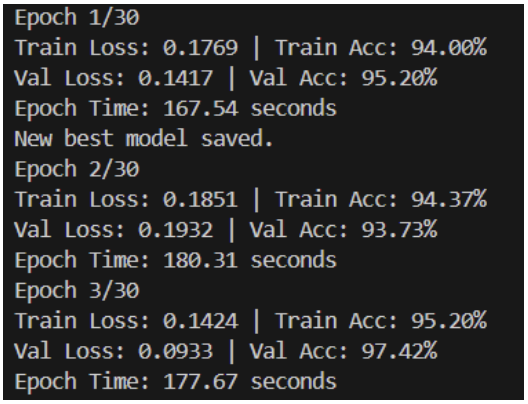


Рисунок 3.11 – Фінальні кроки тренування

На завершальних етапах тренування модель демонструвала винятково високі результати як на тренувальному, так і на валідаційному наборах даних, у деяких випадках наближаючись до показника точності 100%.

Архітектура нейронної мережі ResNet50, адаптована для класифікації архітектурних стилів, представлена детальним звітом про параметри (рис. 3.13).

Загальна кількість параметрів моделі (рис. 3.12) перевищила 24 мільйони, з яких 24,337,930 були тренованими, а 225,344 залишилися незмінними. Параметри, які не тренуються, відповідають вагам, що не були систематично оновлені під час навчання і, відповідно, не впливали на загальну продуктивність моделі.

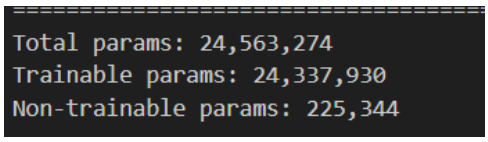
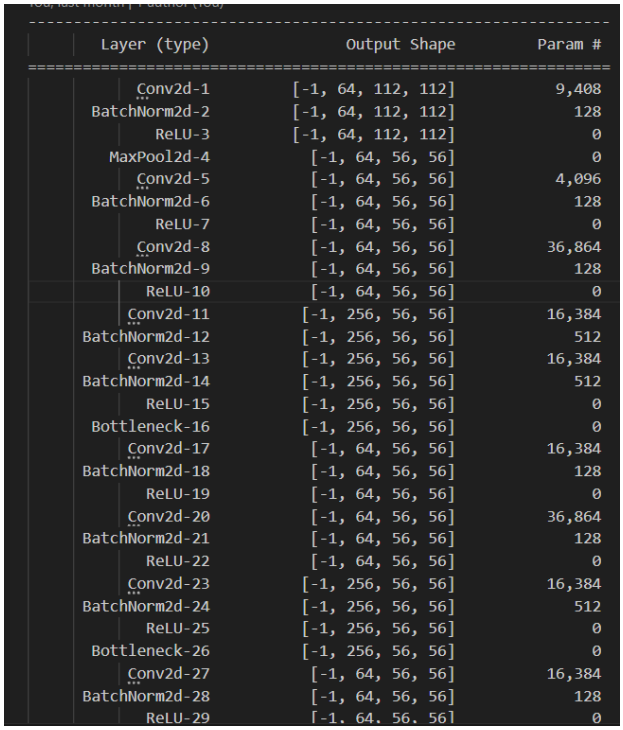


Рисунок 3.12 – Кількість параметрів моделі



Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 112, 112]	9,408
BatchNorm2d-2	[-1, 64, 112, 112]	128
ReLU-3	[-1, 64, 112, 112]	0
MaxPool2d-4	[-1, 64, 56, 56]	0
Conv2d-5	[-1, 64, 56, 56]	4,096
BatchNorm2d-6	[-1, 64, 56, 56]	128
ReLU-7	[-1, 64, 56, 56]	0
Conv2d-8	[-1, 64, 56, 56]	36,864
BatchNorm2d-9	[-1, 64, 56, 56]	128
ReLU-10	[-1, 64, 56, 56]	0
Conv2d-11	[-1, 256, 56, 56]	16,384
BatchNorm2d-12	[-1, 256, 56, 56]	512
Conv2d-13	[-1, 256, 56, 56]	16,384
BatchNorm2d-14	[-1, 256, 56, 56]	512
ReLU-15	[-1, 256, 56, 56]	0
Bottleneck-16	[-1, 256, 56, 56]	0
Conv2d-17	[-1, 64, 56, 56]	16,384
BatchNorm2d-18	[-1, 64, 56, 56]	128
ReLU-19	[-1, 64, 56, 56]	0
Conv2d-20	[-1, 64, 56, 56]	36,864
BatchNorm2d-21	[-1, 64, 56, 56]	128
ReLU-22	[-1, 64, 56, 56]	0
Conv2d-23	[-1, 256, 56, 56]	16,384
BatchNorm2d-24	[-1, 256, 56, 56]	512
ReLU-25	[-1, 256, 56, 56]	0
Bottleneck-26	[-1, 256, 56, 56]	0
Conv2d-27	[-1, 64, 56, 56]	16,384
BatchNorm2d-28	[-1, 64, 56, 56]	128
ReLU-29	[-1, 64, 56, 56]	0

Рисунок 3.13 – Звіт про параметри моделі

3.5 Тестування моделі класифікації зображень споруд

У результаті роботи програми на тестувальному наборі зображень фінальна модель продемонструвала наступні результати.

Після завершення навчання нейронної мережі було проведено тестування на 500 зображеннях (50 зображень на кожен клас), які не входили до складу ні навчального, ні валідаційного наборів. Це дозволило отримати об'єктивну оцінку здатності моделі до генералізації на нових даних.

Результати класифікації агрегуються по кожному з 10 класів: для кожного класу обчислюється відсоток правильних передбачень. Загальна точність визначається як середнє значення цих відсотків по всіх класах, що дозволяє коректно оцінити збалансованість моделі по всіх архітектурних стилях незалежно від розподілу класів у тестовому наборі.

Тестовий набір включав зображення з реалістичними умовами зйомки – з різною якістю та за несприятливих погодних умов (різне освітлення, часткове затемнення, дощ, туман тощо). Це дозволило оцінити, наскільки модель є стійкою до шуму та варіативності у вхідних даних, що є важливим фактором для реальних застосувань.

Модель досягла високих результатів (рис. 3.14): загальна тестова точність становить 86.40%, а F1-міра – 0.8652, що свідчить про збалансовану здатність правильно класифікувати зображення десяти архітектурних стилів. Високі precision та recall у більшості класів, зокрема Japanese Traditional (0.96) і Baroque (0.93), підтверджують ефективність моделі.

```
Test Accuracy: 86.40%
Test Loss: 0.5388
Precision: 0.8773
Recall: 0.8640
F1 Score: 0.8652

Classification Report:
                     precision    recall  f1-score   support

  American Craftsman       0.87      0.82      0.85        50
           Art Deco        0.90      0.86      0.88        50
        Art Nouveau        0.87      0.78      0.82        50
            Baroque        0.98      0.88      0.93        50
          Beaux-Arts       0.64      0.94      0.76        50
            Georgian       0.92      0.68      0.78        50
Japanese Traditional       0.93      1.00      0.96        50
        Queen Annes        0.86      0.88      0.87        50
     Russian Revival       0.92      0.92      0.92        50
       Tudor Revival       0.90      0.88      0.89        50

            accuracy                           0.86       500
           macro avg       0.88      0.86      0.87       500
        weighted avg       0.88      0.86      0.87       500
```

Рисунок 3.14 – Класифікаційний звіт тестування моделі

На рисунках нижче можемо побачити приклади того, як класифікує модель. Наприклад, на рис. 3.15 та рис. 3.16 бачимо зразки з аугментованими зображеннями різних стилів. Можемо помітити, що деякі фото репрезентують

одну й ту саму споруду, але є самі по собі різними фотографіями, знятими під різними кутами або в різний час. Бачимо на підписах над фото, що з високою точністю спрогнозовані стилі відповідають дійсним стилям архітектури.

Рисунок 3.15 – Приклад №1 класифікації будівель

Рисунок 3.16 – Приклад №2 класифікації будівель

На рис. 3.17 можна побачити інший зразок із зображеннями. Тут зображення не є аугментованими, але точність також дуже висока і є насправді для моделі однаковою, що свідчить про стабільність класифікації незалежно від попередньої обробки зображень.

Рисунок 3.17 – Приклад класифікації зображень без аугментації

Висновки до розділу 3

Побудована модель продемонструвала здатність розрізняти архітектурні стилі, які навіть для людини без спеціальної підготовки є важкими для ідентифікації. Це свідчить про глибокий рівень навчання і узагальнення, якого вдалося досягти в результаті тренування моделі. Хоча результати вже є дуже переконливими, можливості моделі можуть бути покращені в подальшій роботі. Наприклад, можна розширити обсяг і різноманітність навчального набору, включити нові стилі, покращити обробку складних випадків класифікації, а також інтегрувати модель у реальні додатки, такі як мобільні гіди, інструменти для реставраторів або освітні платформи.

ВИСНОВКИ

У кваліфікаційній роботі виконано комплексний аналіз проблем, пов’язаних із застосуванням комп’ютерного зору для класифікації архітектурних стилів за зображеннями фасадів будівель. Проведено огляд існуючих методів та технологій, зокрема згорткових нейронних мереж, що є одними з найефективніших для вирішення задач розпізнавання образів.

Розроблена програмна модель для класифікації архітектурних стилів на основі глибокої згорткової нейронної мережі ResNet50. Запропоновано архітектурне рішення, що включає застосування підходу transfer learning для адаптації моделі до вузькоспеціалізованої задачі, а також механізми донавчання на обмежених вибірках даних. Основним результатом є побудова програмної моделі, яка забезпечує високу точність класифікації навіть за умов обмеженої кількості навчальних прикладів та значної міжкласової схожості об’єктів.

У процесі побудови моделі класифікації було передбачено:

– використання глибокої згорткової нейронної мережі для виділення як низькорівневих, так і високорівневих ознак фасадів;

– адаптацію моделі до нових класів за допомогою донавчання;

– досягнення балансу між точністю, швидкодією та можливістю подальшого розширення архітектури.

Окрім того, виконано візуалізацію ключових етапів роботи системи, включаючи архітектуру отриманої моделі, приклади обробки зображень, розподіл класів у датасеті, а також матриці плутанини, що дозволяє наочно оцінити якість класифікації та характер помилок.

Таким чином, розроблена модель демонструє ефективність сучасних методів глибинного навчання для задач комп’ютерного зору та може слугувати основою прикладних сервісів у сфері архітектурної ідентифікації, освітніх платформ та мобільних додатків. Перспективи подальшого розвитку включають розширення навчального набору, інтеграцію нових стилів, оптимізацію архітектури та впровадження системи в реальні додатки.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mitchell T. Machine Learning (Mcgraw-Hill International Edit). McGraw-Hill Education (ISE Editions), 1997.

2. Ben-David S., Shalev-Shwartz S. Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, 2014.

3. Новіченко Н. В. Магістерська дисертація з машинного навчання. Київ : НТУУ «КПІ імені Ігоря Сікорського», 2021. 90 с.

4. Boiman O., Shechtman E., Irani M. In defense of Nearest-Neighbor based image classification. *2008 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Anchorage, AK, USA, 23–28 June 2008. 2008. [Електронний ресурс]. — режим доступу: DOI: https://doi.org/10.1109/cvpr.2008.4587598 (дата звернення: 10.02.2024).

5. Haddad Z., Chen Y., Krahe J. L. Image Processing and Pattern Recognition Tools for the Automatic Image Transcription. *Lecture Notes in Computer Science*. Cham : Springer, 2016. Vol. 9729.P. 197–203. . [Електронний ресурс].— режим доступу: DOI: https://doi.org/10.1007/978-3-319-41264-1_26 (дата звернення: 11.02.2024).

6. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems; MIT Press: Cambridge, MA, USA, 2012.P.1097–1105.

7. Chu W.-T., Tsai M. Visual pattern discovery for architecture image classification and product image search. *ICMR ’12: Proceedings of the 2nd ACM International Conference on Multimedia Retrieval*. 2012. ISBN 9781450313292.

8. Goel A., Juneja M., Jawahar C. Are buildings only instances?: exploration in architectural style categorie. *Proceedings of the Eighth Indian Conference on Computer Vision, Graphics and Image Processing*. 2012. [Електронний ресурс]. — режим доступу: DOI: https://doi.org/10.1145/2425333 (дата звернення: 18.03.2025).

9. Xu Z., Tao D., Zhang Y., Wu J., Tsoi A. C. Architectural Style Classification Using Multinomial Latent Logistic Regression. Computer Vision – ECCV 2014. Cham : Springer, 2014. P. 600–615. [Електронний ресурс].— режим доступу: DOI: https://doi.org/10.1007/978-3-319-10590-1_39 (дата звернення: 18.03.2025).

10. Xu Z., Tao D., Zhang Y., Wu J., Tsoi A. C. Multinomial Latent Logistic Regression for Image Understanding. IEEE Transactions on Image Processing. 2016. Vol. 25, No. 2. P. 973–987. [Електронний ресурс].— режим доступу: DOI: https://doi.org/10.1109/tip.2015.2509422 (дата звернення: 19.03.2025).

11. Lazebnik S., Schmid C., Ponce J. Spatial Pyramid Matching. *Object Categorization* / ed. by S. J. Dickinson et al. Cambridge. P. 401–415. [Електронний ресурс]. — режим доступу: DOI: https://doi.org/10.1017/cbo9780511635465.022 (дата звернення 06.06.2025).

12. Mueller J. P., Massaron L. Machine Learning for Dummies. Wiley & Sons, Incorporated, John, 2016. 432 p.

13. Nielsen M. A. Neural Networks and Deep Learning. Determination Press, 2015. [Електронний ресурс].— режим доступу: URL: http://neuralnetworksanddeeplearning.com/ (дата звернення: 01.02.2025).

14. The Mathematics of Neural Networks. *Medium*.2020. [Електронний ресурс].— режим доступу: URL: https://medium.com/coinmonks/the-mathematics-of-neural-network-60a112dd3e05 (дата звернення: 15.03.2025).

15. Vasudeva, L. *The Crucial Mathematical Fundamentals Behind Machine Learning*. Medium. [Електронний ресурс].— режим доступу: URL: https://medium.com/@LakshyaV/the-crucial-mathematical-fundamentals-behind-machine-learning-2041dea8c47a (дата звернення: 15.03.2025).

16. 1. McCaffrey J. Use Python with Your Neural Networks. *Visual Studio Magazine*. 2014.[Електронний ресурс].— режим доступу: URL: https://visualstudiomagazine.com/articles/2014/11/01/use-python-with-your-neural-networks.aspx (дата звернення: 20.03.2025).

17. Sharma A. *Understanding Activation Functions in Neural Networks*. Medium.[Електронний ресурс].— режим доступу: URL: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0 (дата звернення: 22.03.2025).

18. Activation Functions in Neural Networks [12 Types & Use Cases].V7Labs.[Електронний ресурс]. – режим доступу: URL: https://www.v7labs.com/blog/neural-networks-activation-functions (дата звернення: 22.03.2025).

19. Bergmann, D., Stryker, C. *What is Loss Function? | IBM*. IBM - United States.[Електронний ресурс].— режим доступу: URL: https://www.ibm.com/think/topics/loss-function (дата звернення: 22.03.2025).

20. Mastering Backpropagation: A Comprehensive Guide for Neural Networks. *DataCamp*.[Електронний ресурс].— режим доступу: URL: https://www.datacamp.com/tutorial/mastering-backpropagation (дата звернення: 20.03.2025).

21. iManassa. *Mathematics Behind Gradient Descent*. Medium.[Електронний ресурс].— режим доступу: URL: https://medium.com/geekculture/mathematics-behind-gradient-descent-f2a49a0b714f (дата звернення: 23.03.2025).

22. Autoren der Wikimedia-Projekte.*Neocognitron – Wikipedia*. Wikipedia – Die freie Enzyklopädie.[Електронний ресурс].— режим доступу: URL: https://de.wikipedia.org/wiki/Neocognitron (дата звернення: 24.03.2025).

23. Pooja, D. *Convolutional Neural Network*. Medium.[Електронний ресурс].— режим доступу: URL: https://medium.com/@drpa/convolutional-neural-network-73a270dd106d (дата звернення: 24.03.2025).

24. Nielsen, D. *Deep Learning Cage Match: Max Pooling vs Convolutions*. Medium.[Електронний ресурс].— режим доступу: URL: https://medium.com/@duanenielsen/deep-learning-cage-match-max-pooling-vs-convolutions-e42581387cb9 (дата звернення: 22.03.2025).

25. Dharmaraj. (2022, 1 червня). *Convolutional Neural Networks (CNN) — Architectures Explained*. Medium.[Електронний ресурс].— режим доступу:

URL: https://medium.com/@draj0718/convolutional-neural-networks-cnn-architectures-explained-716fb197b243 (дата звернення: 24.03.2025).

26. Ioffe, S., & Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. У *ICML'15: Proceedings of the 32nd International Conference on International Conference on Machine Learning*.2015.

27. *Transfer Learning: Anwendungsfälle und Beispiele*. Datasolut GmbH.[Електронний ресурс].— режим доступу: URL: https://datasolut.com/was-ist-transfer-learning/ (дата звернення: 27.03.2025).

28. He, K., Zhang, X., Ren, S., & Sun, J. Deep Residual Learning for Image Recognition. У *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.2016. [Електронний ресурс].— режим доступу: DOI: https://doi.org/10.1109/cvpr.2016.90

29. *Understanding ResNet Architecture: A Deep Dive into Residual Neural Network*. Medium.2023.[Електронний ресурс].— режим доступу: URL: https://medium.com/@ibtedaazeem/understanding-resnet-architecture-a-deep-dive-into-residual-neural-network-2c792e6537a9

30. Bendjillali, R. I., Beladgham, M., Merit, K., Taleb-Ahmed, A. Illumination-robust face recognition based on deep convolutional neural networks architectures. *Indonesian Journal of Electrical Engineering and Computer Science*. 2020. Vol. 18, No. 2. P. 1015-1027. [Електронний ресурс].— режим доступу: DOI: https://doi.org/10.11591/ijeecs.v18.i2.pp1015-1027 (дата звернення: 24.03.2025).

31. Zhang A., Lipton Z. C., Li M., Smola A. J. *Modern Convolutional Neural Networks.* Dive into Deep Learning. Cambridge : Cambridge University Press, 2023.[Електронний ресурс].— режим доступу: URL: https://www.d2l.ai/chapter_convolutional-modern/index.html (дата звернення: 24.02.2025).

32. Architecture Dataset. Kaggle. [Електронний ресурс].— режим доступу: URL: https://www.kaggle.com/datasets/wwymak/architecture-dataset (дата звернення: 03.02.2025).

33. Zhang A., Lipton Z. C., Li M., Smola A. J. *Computer Vision*. Dive into Deep Learning. Cambridge : Cambridge University Press, 2023.[Електронний ресурс].— режим доступу: URL: https://www.d2l.ai/chapter_computer-vision/index.html (дата звернення: 24.02.2025).

34. Loshchilov I., Hutter F. Decoupled Weight Decay Regularization. *Published as a conference paper at ICLR 2019*. Freiburg, Germany: University of Freiburg, 2019.

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) **Бакалавр**
Галузь знань: 12 – Інформаційні технології
Спеціальність: 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних
систем та робототехнікики
к. ф.-м. н., доц. ХРУСЛОВ М. М.
«02» жовтня 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Єрьоміної Надії Володимирівни
(прізвище, ім'я, по батькові студента)

1. Тема роботи «**МОДЕЛЬ КЛАСИФІКАЦІЇ СТИЛІВ АРХІТЕКТУРНИХ СПОРУД**»
керівник роботи Стрілець Вікторія Євгенівна, кандидат технічних наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від ***16 квітня 2025 року № 4101-5/962***

2. Строк подання студентом роботи ***30 травня 2025 року***

3. Перелік питань, які потрібно розробити:
1. Аналіз існуючих технологій, підходів, методів і моделей до розпізнавання стилів архітектурних споруд на зображеннях.
2. Створення моделі класифікації стилів архітектурних споруд на зображеннях на основі глибинних нейронних мереж.
3. Тестування та оцінка ефективності розробленої моделі класифікації стилів архітектурних споруд.
4. Аналіз подальших можливостей розвитку запропонованої моделі.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Пошук наукової літератури за темою роботи	02.10.2024-24.10.2024
2	Аналіз технологій і підходів класифікації стилів споруд за зображенням.	02.10.2024-24.10.2024
3	Огляд методів і моделей класифікації і аналізу зображень.	25.10.2024-09.11.2024
4	Розробка моделі класифікації стилів архітектурних споруд на зображеннях на основі нейронних мереж.	10.11.2024-24.11.2024
5	Обґрунтування технологій машинного навчання для розробки моделі.	25.10.2024-08.11.2024
6	Збір даних для навчання моделі.	09.11.2024-29.11.2024
7	Створення моделі класифікації на основі техніки передавального навчання (Transfer Learning).	30.11.2024-31.12.2024
8	Тестування та оцінка ефективності моделі класифікації стилів архітектурних споруд.	01.01.2025-01.02.2025
9	Аналіз можливостей подальшого вдосконалення запропонованої моделі.	01.03.2025-30.04.2025
10	Підготовка і оформлення звітних матеріалів та додатків кваліфікаційної роботи. Оформлення списку літератури	01.04.2025-30.04.2025
11	Оформлення пояснювальної записки кваліфікаційної роботи відповідно вимогам до звітів про НДР.	01.05.2025-30.05.2025
12	Оформлення звіту про переддипломну практику	01.05.2025-30.05.2025
13	Представлення кваліфікаційної роботи керівнику та рецензенту	30.05.2025

5. Дата видачі завдання *02 жовтня 2024 року*.

Студент Єрьоміна Н.В. _____

підпис

Керівник роботи Стрілець В.Є. _____

підпис

Додаток Б

Затверджую

_____

«_____»_____2025 р.

Технічне завдання

на розробку програмного виробу «МОДЕЛЬ КЛАСИФІКАЦІЇ СТИЛІВ АРХІТЕКТУРНИХ СПОРУД»

Назва розділу	Назва та зміст підрозділу
1. Введення	1.1. Назва програмного виробу – МОДЕЛЬ КЛАСИФІКАЦІЇ СТИЛІВ АРХІТЕКТУРНИХ СПОРУД 1.2. Галузь застосування – системи візуальної аналітики простору.
2. Підстава для розробки	2.1. Навчальний план за спеціальністю 123 – Комп'ютерна інженерія. 2.2. Завдання на кваліфікаційну роботу бакалавра: НАКАЗ № 4101-5/962 від 16.04.2025 (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи)._____
3. Призначення розробки	3.1. Мета розробки програмного виробу: підвищення якості автоматизованого розпізнавання стилів архітектурних споруд на зображеннях за допомогою моделей і методів штучного інтелекту. Для реалізації поставленої мети необхідно вирішити такі *завдання*: 1. Провести аналітичний огляд сучасних методів класифікації зображень, зокрема архітектурних об'єктів. 2. Підготувати набір зображень фасадів будівель із відповідною розміткою архітектурних стилів. 3. Виконати попередню обробку набору даних і провести аугментацію для підвищення узагальнюючої здатності моделі. 4. Адаптувати архітектуру моделі ResNet50 до задачі багатокласового розпізнавання зображень та навчити модель за допомогою фреймворку PyTorch. 5. Провести оцінку якості класифікації на основі метрик ефективності. 6. Проаналізувати результати, визначити переваги й обмеження створеної моделі та її програмної реалізації. 3.2. Призначення програмного виробу:

	Цей програмний виріб призначений для автоматизації процесу класифікації архітектурних стилів на основі зображень, зокрема для навчальних та дослідницьких цілей у галузі штучного інтелекту. 3.3. Вихідні дані для розробки : • Використання трансферного навчання на попередньо навченій моделі ResNet50 для покращення результатів класифікації при обмеженій кількості даних. • Набір зображень, що містить 10 архітектурних стилів: American Craftsman, Art Deco, Art Nouveau, Baroque, Beaux-Arts, Georgian, Japanese Traditional, Queen Anne's, Russian Revival, Tudor Revival.
4. Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: а) програмний виріб повинен автоматично виконувати класифікацію архітектурних стилів на основі вхідного зображення; б) програмний виріб має підтримувати обробку зображень у форматах .jpg, .jpeg та .png; в) користувач повинен мати можливість завантажити зображення, вказавши шлях до файлу або надавши шлях до папки з зображеннями, якщо йдеться про обробку великої кількості зображень; г) система має забезпечувати класифікацію зображення за одним із 10 архітектурних стилів: American Craftsman, Art Deco, Art Nouveau, Baroque, Beaux-Arts, Georgian, Japanese Traditional, Queen Anne's, Russian Revival, Tudor Revival; д) після класифікації має виводитись назва визначеного стилю; е) максимальний обсяг оперативної пам'яті, що використовується під час класифікації одного зображення, не повинен перевищувати 500 МБ; є) час inference (обробки) одного зображення розміром 224×224 пікселів не перевищує 3 с на ПК із GPU NVIDIA GTX 1050 та відповідними оптимізаціями. 4.2. Вимоги до надійності: а) модель має бути стійкою до невеликих змін у зображенні (варіації освітлення, шум); б) обробка винятків у випадку пошкоджених файлів або некоректного формату вхідних даних; в) система не повинна аварійно завершуватись у випадку проблеми із зображенням; г) програмний виріб має зберігати цілісність моделей та вихідних даних навіть у разі збоїв у роботі. 4.3. Вимоги до умов експлуатації:

	а) Кліматичні умови: температура навколишнього середовища: від +10°C до +35°C. Відносна вологість: від 20% до 80%, без конденсації вологи. б) Програма та модель можуть бути завантажені з репозиторію. Для запуску програми достатньо завантажити архів з програмним забезпеченням та розпакувати його на локальний диск. в) Запуск програми здійснюється безпосередньо з локального комп'ютера без потреби в додаткових налаштуваннях або підключеннях до серверів. г) Після завантаження програми користувач може одразу почати використовувати модель для класифікації архітектурних стилів, вказавши шлях до зображення у програмі. д) Програма працює в офлайн-режимі і не потребує постійного підключення до Інтернету після початкового завантаження. 4.4. Вимоги до складу параметрів технічних засобів: а) програма повинна підтримувати виконання на персональних комп’ютерах під операційними системами Windows (7 і вище), Linux (будь-які поширені дистрибутиви) та macOS (10.13 і вище). Для запуску необхідний інтерпретатор Python версії 3.6 або вище; б) для користування програмою мінімальними вимогами є підключення комп'ютера до електромережі, мінімальна оперативна пам'ять 8 ГБ, дисковий простір 1 ГБ, процесор 3.0-3.5 ГГц; в) для прискорення обчислень рекомендується використовувати дискретний графічний процесор (GPU) з підтримкою CUDA (версія 10.2 або новіша), наприклад NVIDIA GTX 1050 або вище. 4.5. Вимоги до інформаційної та програмної сумісності Програма працює автономно під керуванням сучасних ОС: Windows 10/11, Linux (Ubuntu 20.04+), macOS 10.15+. Потрібен Python 3.12+ (або сумісних) та бібліотеки PyTorch 2.7.0, NumPy 2.0.1, ONNX 1.17.0 тощо. Вхідні дані — зображення у форматах .jpg/.png (розмір 224×224), вихід — клас і ймовірність. Програма реалізує згорткову нейронну мережу, може бути експортована у формат ONNX для сумісності. Код написаний на Python, легко вбудовується до більших систем. Захист даних не реалізовано, оскільки продукт призначено для локального навчального та дослідницького використання. 4.6. Вимоги до маркування та упаковки

	Вимоги до маркування та упаковки не пред'являються, оскільки програмний продукт не потребує спеціальної упаковки для навчального чи дослідницького використання. 4.7. Вимоги до транспортування та зберігання Вимоги до транспортування та зберігання не пред'являються, оскільки програмний продукт є цифровим і не потребує фізичного транспортування чи спеціальних умов зберігання. 4.8. Спеціальні вимоги Спеціальних вимог до програмного виробу не пред'являються. Програмний продукт є стандартним додатком для класифікації архітектурних стилів та не потребує додаткових специфікацій або унікальних умов для використання.
5. Вимоги до програмної документації.	Програмною документацією до виробу «Модель класифікації стилів архітектурних споруд на зображеннях» вважати: 1) Це Технічне завдання на розробку програмного виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Програму та методику випробувань розробленого програмного виробу(представити у вигляді Додатку В до пояснювальної записки до кваліфікаційної роботи). 3) Опис програмного виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи). 4) Текст програми (представити в Додатку Г до пояснювальної записки до кваліфікаційної роботи).
6. Техніко-економічні показники	1) Планується досягти точності класифікації архітектурних стилів на рівні близько 85%, що є конкурентним показником з огляду на складність розмежування стилів. Для порівняння, аналогічні рішення демонструють точність у межах 80–90%. 2) Розробка моделі триватиме орієнтовно 3 місяці. Планується використовувати безкоштовне програмне забезпечення та доступне апаратне забезпечення. У разі масштабування — можливі витрати на продуктивніші графічні процесори. 3) Очікується, що модель матиме подібний або вищий рівень ефективності, ніж наявні рішення. Вона буде доступнішою для інтеграції в освітні або дослідницькі застосунки завдяки спрощеній реалізації.

7. Стадії та етапи розробки	Розробка моделі класифікації архітектурних споруд проходить у три основні стадії відповідно до ДЕСТ 19.102-77: Стадії: 1) Розробка технічного завдання— формування вимог, постановка задачі, погодження з керівником. 2) Робоче проектування— включає програмування, налагодження, розробку документації та випробування. 3) Впровадження розробленого виробу — підготовка, передача програми та документації до експлуатації. Етапи: 1) Розробка програми (підбір і підготовка датасету, створення моделі на основі підходу Transfer learning, тренування та валіадатація моделі, коригування за потрібності, тестування, створення звітів з аналітичними результатами роботи моделі). 2) Розробка програмної документації.(створення пояснювальної записки, візуалізацій результатів). 3) Впровадження (випробування) програми (передача готової моделі для практичного або навчального використання).
8. Порядок контролю та приймання	1) Перевірку ходу розроблення заданих проектних документів керівнику курсової роботи здійснювати 1 раз на 3 тижні. 2) Випробування програмного виробу. 3) Захист розробленого програмного виробу провести на засіданні ДЕК. 4) Розроблені проектні документи подати в електронному вигляді та на паперових носіях в одному примірнику.

Виконавець

Студент групи КІ-41

Єрьоміна Надія Володимирівна

Замовник

доцент кафедри комп'ютерних систем та робототехніки, к.т.н

Стрілець Вікторія Євгенівна

Додаток В

Програма та методика випробувань програмного виробу «Модель класифікації стилів архітектурних споруд на зображеннях»

1. Об'єкт випробувань

1.1 Найменування програмного виробу:

«МОДЕЛЬ КЛАСИФІКАЦІЇ СТИЛІВ АРХІТЕКТУРНИХ СПОРУД».

1.2 Галузь застосування застосування:

Область застосування розробленої моделі охоплює автоматизований візуальний аналіз міського середовища, цифрову архівацію й класифікацію об’єктів культурної спадщини, підтримку містобудівних і реставраційних проєктів, а також використання у навчальних і спеціалізованих додатках для архітекторів, урбаністів і музейних працівників.

1.3 Умовне призначення розробки:

Цей програмний виріб призначений для автоматизації процесу класифікації архітектурних стилів на основі зображень, зокрема для навчальних та дослідницьких цілей у галузі штучного інтелекту.

2. Мета випробувань

Метою випробувань є перевірка відповідності функціональних характеристик розробленого програмного виробу вимогам, зазначеним у Технічному завданні(Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Загальні положення

3.1 Підстави щодо випробувань

Підставою для проведення випробувань є: 1) НАКАЗ № 4101-5/962 від 16.04.2025 про затвердження тем кваліфікаційних робіт бакалаврів; 2) Навчальний план дисципліни.

3.2 Місце та тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі комп'ютерного класу кафедри в період переддипломної практики.

Підрозділ 3.3. «Обсяг випробувань»

Приймальні випробування програмного виробу проводяться в обсязі відповідно до цієї програми та методики випробувань.

3.4 Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програмного виробу

а) програмний виріб повинен автоматично виконувати класифікацію архітектурних стилів на основі вхідного зображення;

б) програмний виріб має підтримувати обробку зображень у форматах .jpg, .jpeg та .png;

в) користувач повинен мати можливість завантажити зображення, вказавши шлях до файлу або надавши шлях до папки з зображеннями, якщо йдеться про обробку великої кількості зображень;

г) система має забезпечувати класифікацію зображення за одним із 10 архітектурних стилів: American Craftsman, Art Deco, Art Nouveau, Baroque, Beaux-Arts, Georgian, Japanese Traditional, Queen Anne's, Russian Revival, Tudor Revival;

д) після класифікації має виводитись назва визначеного стилю;

е) максимальний обсяг оперативної пам'яті, що використовується під час класифікації одного зображення, не повинен перевищувати 500 МБ;

є) час inference (обробки) одного зображення розміром 224×224 пікселів не перевищує 3 с на ПК із GPU NVIDIA GTX 1050 та відповідними оптимізаціями.

5. Вимоги до програмної документації

Програмною документацією до виробу «Модель класифікації стилів архітектурних споруд на зображеннях» вважати:

1) це Технічне завдання на розробку програмного виробу (представити у вигляді Додатка Б до пояснювальної записки до кваліфікаційної роботи).

2) програму та методику випробувань розробленого програмного виробу (представити у вигляді Додатка В до пояснювальної записки до кваліфікаційної роботи).

3) опис програмного виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи).

4) текст програми (представити в Додатку Г до пояснювальної записки до кваліфікаційної роботи).

6. Засоби та порядок випробувань

6.1 Засоби випробувань

Для проведення випробувань буде використовуватись наступне технічне та програмне забезпечення:

Технічні засоби:

- Сервери або робочі станції з достатньою кількістю оперативної пам'яті та потужними графічними процесорами (GPU) для прискорення обчислень.
- Вихідні зображення архітектурних об’єктів для тестування та класифікації.

Програмні засоби:

- Python — основна мова програмування для реалізації моделі глибокого навчання.
- PyTorch — фреймворк для створення та навчання нейронної мережі, використовуючи архітектуру ResNet50 для класифікації архітектурних стилів.
- Visual Studio Code (VS Code) — середовище для розробки, в якому буде здійснюватися написання коду та його виконання.
- TensorBoard — для візуалізації процесу навчання моделі, включаючи побудову графіків точності та втрат, а також аналіз показників моделі під час навчання. Використовуватиметься через командний рядок або у VS Code для аналізу навчання моделі.
- NumPy, Pandas — для обробки даних та аналізу результатів.

- Matplotlib — для візуалізації результатів тестування та графічного представлення ефективності моделі.
- OpenCV — для обробки зображень.

Та інші.

Програма тестування буде надаватися у вигляді інсталяційної версії розробленого програмного продукту з документацією, яка містить інструкції щодо налаштування середовища для запуску моделі.

6.2 Порядок проведення випробувань

1-й етап. Перевірка комплектності програмної документації

Перевіряється наявність усіх необхідних документів згідно з Технічним завданням. Перевірка комплектності технічних та програмних засобів здійснюється за критерієм відповідності їх складу вимогам середовища виконання (Visual Studio Code, PyTorch, TensorBoard, Python 3.10, ОС Windows 10/11).

2-й етап. **Перевірка якості програмної документації**

Якість програмної документації відповідає вимогам ДЕСТ 19.301–79 ЄСПД.

3-й етап. Функціональне випробування програмного виробу

На цьому етапі перевіряється працездатність моделі та ступінь виконання функціональних вимог.

1. Перевірка працездатності програмного виробу

Методика проведення Тесту 1:

PyTorch використовується на етапі розробки та навчання моделі.

- Запуск тестування моделі здійснюється у середовищі Visual Studio Code командою python test_model.py.
- Вибирається тестове одне зображення з директорії test_data/.

Наприклад для тесту оберемо зображення:

Рисунок В. 1 – приклад зображення, яке містить будівлю у архітектурному стилі Beaux-Arts

- Шлях до зображення може буде вказаний просто скопійованим шляхом або через файл з розширенням .yaml .
- Програма виконує:

- завантаження моделі PyTorch;
- попередню обробку зображення;
- виконання передбачення;
- вимір часу виконання та обсягу використаної пам’яті;
- вивід назви передбаченого класу у консоль.

Очікуваний результат:

- У консолі виводиться назва класу (Beaux-Arts)
- Вивід часу виконання та спожитої пам’яті

Фактичний результат:

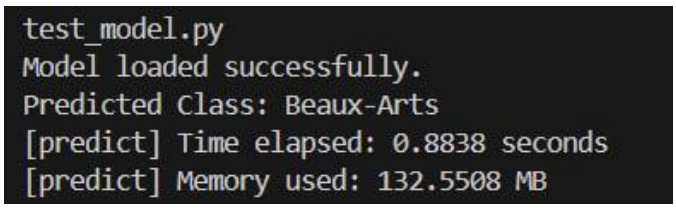


Рисунок В. 2 – Результат перевірки працездатності за тестом 1

Висновок: у результаті виконання Тесту 1 функціональна вимога щодо якості класифікації виконана. А також було виміряно час виконання обробки одного зображення та використання пам’яті, що не перевищує вимоги.

Методика проведення Тесту 2:

ONNX (Open Neural Network Exchange) — це формат для перенесення моделі з PyTorch у продакшн, наприклад, у веб або мобільний застосунок.

- Запуск тестування моделі здійснюється у середовищі Visual Studio Code командою python test_opt_model.py /тут вказати шлях до зображення/.
- Вибирається тестове одне зображення з директорії test_data/.(оберемо таке саме зображення як і у тесті 1 (рис.В. 1)
- Шлях до зображення може буде вказаний просто скопійованим шляхом або через файл з розширенням .yaml .
- Програма виконує:
  - завантаження моделі ONNX;
  - попередню обробку зображення;
  - виконання передбачення;
  - вимір часу виконання та обсягу використаної пам’яті;
  - вивід назви передбаченого класу у консоль.

Очікуваний результат:

- У консолі виводиться назва класу (Beaux-Arts).
- Вивід часу виконання та спожитої пам’яті.

Фактичний результат:

Рисунок В. 3 – Результат перевірки працездатності за тестом 2

Висновок: у результаті виконання Тесту 2 програмний виріб вважається працездатним, оскільки коректно класифікував вхідне зображення, без помилок завершив роботу та вивів очікуваний результат. Даний формат збереженої моделі також швидще процює за попередній, але і займає більше пам’яті.

Методика проведення Тесту 3:

- Запуск програми здійснюється з командного рядка командою: python test_evaluation.py
- Програма завантажує навчений класифікаційний модель final_model.pth з каталогу ../architecture_classifier/models.
- Автоматично відбувається завантаження тестових даних за допомогою функції load_data().

На основі даних виконується оцінка якості моделі:

- обчислюються метрики: точність (Accuracy), precision, recall, F1-score;
- виводиться повний звіт класифікації (classification_report);
- будується матриця плутанини (confusion matrix).

Результати логуються у TensorBoard та зберігаються у файл test_evaluation_results.txt у робочій директорії.

Для перегляду результатів оцінки в TensorBoard виконується команда: tensorboard --logdir=runs

Очікуваний результат:

- Виведення статистики тестування у консоль;
- Збереження файлу з результатами;
- Відображення логів з результатами у TensorBoard.

Фактичний результат:

```
Test Accuracy: 86.40%
Test Loss: 0.5388
Precision: 0.8773
Recall: 0.8640
F1 Score: 0.8652

Classification Report:
                     precision    recall  f1-score   support

  American Craftsman       0.87      0.82      0.85        50
           Art Deco       0.90      0.86      0.88        50
       Art Nouveau       0.87      0.78      0.82        50
           Baroque       0.98      0.88      0.93        50
       Beaux-Arts       0.64      0.94      0.76        50
         Georgian       0.92      0.68      0.78        50
Japanese Traditional       0.93      1.00      0.96        50
       Queen Annes       0.86      0.88      0.87        50
```

Рисунок В. 4 – Результат формування звіту класифікації

Рисунок В. 5 – Приклад з TensorBoard на якому відображено залоговані використані зображенння з тестового сету з їх справжніми та передбаченими зображеннями

Висновок:

Програма успішно виконала всі етапи тестування моделі. В результаті були отримані точні метрики, такі як точність (accuracy), precision, recall та F1-score, а також був згенерований повний звіт класифікації, включаючи матрицю плутанини. Усі результати було успішно залоговано в TensorBoard та збережено у файл test_evaluation_results.txt.

Очікувані результати були досягнуті: статистика тестування була виведена у консоль, файли з результатами збережено, а також візуалізовано зображення тестових даних у TensorBoard. Програма працює стабільно, ефективно оцінюючи модель і забезпечуючи зручний доступ до результатів тестування.

Фактичний результат відповідає вимогам, що підтверджує коректність роботи програми та ефективність виконання тесту.

Виконавець: студент групи КІ-41, Єрьоміна Н.В. _____

Додаток Г

Лістинг коду «Модель для класифікації стилів архітектурних споруд»

```
import os
import torch
from collections import Counter
from torch import nn
from torch.utils.data import WeightedRandomSampler
from torchvision import models

def build_model(num_classes, pretrained=True):
    """
    Parameters:
    num_classes (int): The number of classes in the dataset.
    pretrained (bool): Whether to load pretrained weights.

    Returns:
     model (torch.nn.Module): The constructed ResNet-50 model with
custom adjustments.
    """
    # Load ResNet-50
    model = models.resnet50(
        weights=models.ResNet50_Weights.IMAGENET1K_V2 if pretrained
else None
    )

    # Freeze initial layers to retain pretrained features
    for param in model.parameters():
        param.requires_grad = False

    # Unfreeze deeper layers for more detailed feature learning
    for param in model.layer2.parameters():
        param.requires_grad = True
    for param in model.layer3.parameters():
        param.requires_grad = True
    for param in model.layer4.parameters():
        param.requires_grad = True

     # Replace final fully connected layer to fit the number of
classes
    model.fc = nn.Sequential(
        nn.Linear(model.fc.in_features, 512),
        nn.BatchNorm1d(512),
        nn.ReLU(),
        nn.Dropout(0.5),
        nn.Linear(512, num_classes),
    )

    return model

def get_class_weights(dataset):
    """
    Computes the class weights based on the frequency of each class
in the dataset.
```

```
    Parameters:
     dataset (torch.utils.data.Dataset): The dataset from which to
compute class weights.

    Returns:
    sample_weights (list): A list of weights for each sample in the
dataset.
    """
    # Extract target labels based on dataset structure
    if hasattr(dataset, "targets"):
        targets = dataset.targets
    else:
              targets   =   [dataset.dataset.targets[i]   for   i   in
dataset.indices]

    # Compute class counts and total samples
    class_counts = Counter(targets)
    total_samples = sum(class_counts.values())

    # Compute class weights using the inverse square root of class
frequencies
    weights = {
        cls: (total_samples / (count + 1e-6)) ** 0.5
        for cls, count in class_counts.items()
    }

    # Assign the appropriate weight to each sample
    sample_weights = [weights[label] for label in targets]

    return sample_weights

def
load_model(model_path="../architecture_classifier/models/best_mode
l.pth"):
    """
    Loads a trained model from a checkpoint or returns an untrained
model.

    Parameters:
    model_path (str): Path to the model checkpoint file.

    Returns:
    model (torch.nn.Module): The loaded or newly constructed model.
    class_names (list): List of class names in the dataset.
    """
          dataset_root    =    "../architecture_classifier/ADrepo-
main/150_set/Training_Set"

    # Get class names by listing directories in the dataset
    class_names = sorted(
        [
            d
            for d in os.listdir(dataset_root)
            if os.path.isdir(os.path.join(dataset_root, d))
        ]
    )
```

```
    # Build the model
    model = build_model(len(class_names))

    # Load the model checkpoint if it exists
    if os.path.exists(model_path):
        model.load_state_dict(torch.load(model_path))
        print("Model loaded successfully.")
    else:
        print(
            f"Warning: Model checkpoint not found at {model_path}.
Returning untrained model."
        )

    # Set the model to training mode
    model.train()

    return model, class_names

def get_loss_function():
    """
     Returns a CrossEntropyLoss function with label smoothing for
better generalization.

    Returns:
     loss_function (torch.nn.CrossEntropyLoss): The loss function
with label smoothing.
    """
    return nn.CrossEntropyLoss(label_smoothing=0.1)

def get_optimizer(model):
    """
    Returns an AdamW optimizer configured to update only trainable
parameters.

    Parameters:
    model (torch.nn.Module): The model to optimize.

    Returns:
    optimizer (torch.optim.AdamW): The AdamW optimizer for training.
    """
    return torch.optim.AdamW(
        filter(lambda p: p.requires_grad, model.parameters()),
        lr=3e-4,
        weight_decay=1e-4,
    )
```

Додаток Д

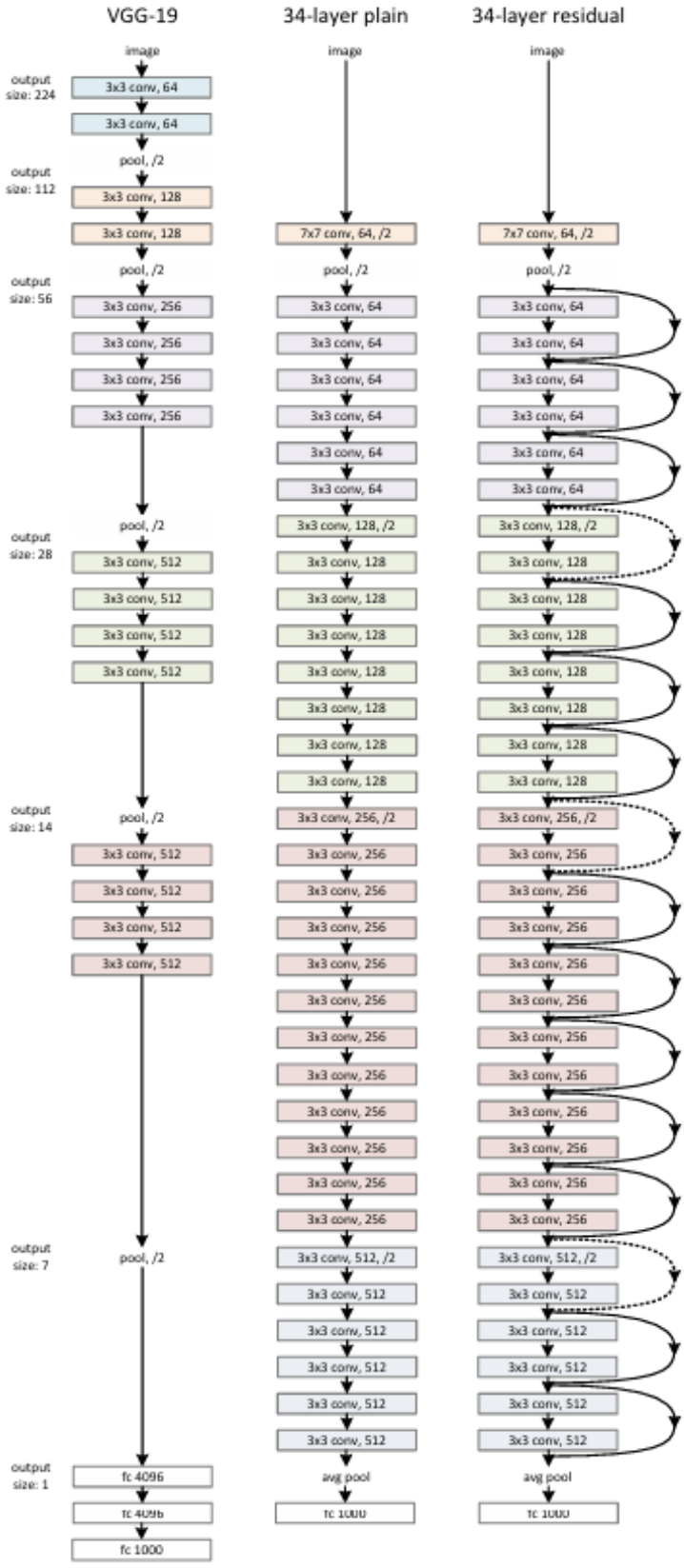


Рисунок Д.1 – Порівняння мереж VGG-19, 34-шарової нейронної мережі та залишкової мережі з 34 шарами