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 ШТУЧНОГО ЗОРУ РОБОТА ДЛЯ РОЗПІЗНАВАННЯ РАДІОДЕТАЛЕЙ**»

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.

Виконав:
Студент групи КІ–41
БОЖКО Тарас Вячеславович
_____

Керівник: к.т.н., доцент, доцент ЗВО кафедри комп'ютерних систем та робототехніки
БИКОВА Тетяна Володимирівна
_____

Рецензент: к.т.н., доцент, доцент ЗВО кафедри математичного моделювання та аналізу даних
КОРОБЧИНСЬКИЙ Кирил Петрович
_____

Харків – 2025

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра включає вступ, три розділи, висновки, список використаних джерел і додатки. Загальний обсяг роботи становить 80 сторінок, із них 47 сторінок основного тексту з 37 рисунками, 2 таблицями, 17 найменуваннями у списку літератури та чотирьома додатками.

Об'єкт дослідження – процес розпізнавання радіодеталей за допомогою комп'ютерного зору.

Предмет дослідження – модель штучного зору на базі YOLOv8x та програмні модулі для її практичного застосування в режимах пакетної обробки й аналізу в реальному часі.

Мета роботи – створення моделі штучного зору для автоматичного розпізнавання радіодеталей і розробка програмного забезпечення для її практичного застосування в режимах пакетної обробки та реального часу.

Проблема, яку вирішує робота – створення та вдосконалення системи розпізнавання радіодеталей із високою точністю й швидкістю в умовах шумів, змінного освітлення та динамічних сценаріїв. Існуючі рішення часто мають обмеження в адаптивності до реальних виробничих умов і потребують оптимізації для інтеграції в роботизовані системи.

Область застосування – автоматизація процесів сортування, складання та контролю якості в електроніці й робототехніці. Система може бути використана у виробництві, навчальних проектах і розробці автономних технологій.

Ключові слова: штучний зір, розпізнавання об'єктів, YOLOv8x, радіодеталі, автоматизація, робототехніка, обробка даних, машинне навчання, пакетний аналіз, реальний час.

ABSTRACT

The explanatory note to the bachelor's qualification work includes an introduction, three sections, conclusions, a list of references, and appendices. The total volume of the work is 80 pages, with 47 pages of main text containing 37 figures, 2 tables, 17 reference items, and four appendices.

The object of study is the process of recognizing electronic components using computer vision.

The subject of study is an artificial vision model based on YOLOv8x and software modules for its practical application in batch processing and real-time analysis modes.

The aim of the work is to develop an artificial vision model for automated recognition of electronic components and to create software for its practical application in batch processing and real-time modes.

The problem addressed by the work is the development and improvement of a system for recognizing electronic components with high accuracy and speed under conditions of noise, varying lighting, and dynamic scenarios. Existing solutions often lack adaptability to real production environments and require optimization for integration into robotic systems.

The field of application includes automation of sorting, assembly, and quality control processes in electronics and robotics. The system can be utilized in manufacturing, educational projects, and the development of autonomous technologies.

Keywords: artificial vision, object recognition, YOLOv8x, electronic components, automation, robotics, data processing, machine learning, batch analysis, real-time.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

COCO JSON – Формат даних для зберігання анотацій об’єктів, який використовує координати та класи.

Edge Impulse – Платформа для розробки моделей комп’ютерного зору для вбудованих систем.

FPS – Frames Per Second (Кадри за секунду) – показник частоти обробки кадрів у реальному часі.

IoU – Intersection over Union (Перетин над об’єднанням) – метрика для оцінки точності детекції об’єктів.

JSON – JavaScript Object Notation – формат зберігання структурованих даних.

mAP50 – Mean Average Precision при IoU=0.5 – середня точність детекції при порозі збігу 50%.

mAP50-95 – Mean Average Precision для IoU від 0.5 до 0.95 – середня точність із урахуванням точності меж.

OpenCV – Open Source Computer Vision Library – бібліотека для обробки зображень і відео.

Python – Мова програмування, використана для розробки скриптів і модулів.

Ultralytics – Розробник бібліотеки YOLOv8, яка використовується в роботі.

YOLO – You Only Look Once – сімейство одноетапних алгоритмів детекції об’єктів.

YOLOv8n – Найлегша версія YOLOv8 (nano) для обмежених ресурсів.

YOLOv8x – Найглибша версія YOLOv8 (extra large) із високою точністю.

ВСТУП

Актуальність теми. Сучасний розвиток робототехніки та автоматизованих систем значною мірою залежить від технологій комп'ютерного зору, які забезпечують обробку візуально-просторових даних для ідентифікації об'єктів. Одним із ключових напрямів у промисловій автоматизації є розпізнавання електронних компонентів, зокрема радіодеталей (конденсаторів, діодів, світлодіодів тощо), що є важливими складовими електронних пристроїв. Автоматизація їхньої ідентифікації здатна підвищити продуктивність виробничих процесів, зменшивши час і витрати на сортування та контроль якості. У цьому контексті розробка моделі штучного зору для розпізнавання радіодеталей має високу актуальність, поєднуючи теоретичні основи машинного навчання з практичними потребами промисловості.

Метою роботи є створення моделі штучного зору для автоматичного розпізнавання радіодеталей і розробка програмного забезпечення для її практичного застосування в режимах пакетної обробки та реального часу.

Для реалізації цієї мети визначено такі задачі:

1. Проаналізувати сучасні методи комп'ютерного зору для розпізнавання об'єктів.
2. Підготувати набір даних із зображень радіодеталей.
3. Розробити та навчити модель штучного зору.
4. Створити програмні модулі для аналізу зображень із папки та в реальному часі.
5. Оцінити точність і ефективність розроблених рішень.

Об'єкт дослідження – процес розпізнавання радіодеталей за допомогою комп'ютерного зору.

Предмет дослідження – модель штучного зору на базі YOLOv8x та програмні модулі для її практичного застосування в режимах пакетної обробки й аналізу в реальному часі.

РОЗДІЛ 1.
ТЕОРЕТИЧНИЙ АНАЛІЗ І ПОСТАНОВКА ЗАДАЧІ

1.1. Огляд сучасних методів комп'ютерного зору для розпізнавання об'єктів

Розпізнавання об'єктів як напрям комп'ютерного зору поєднує методи обробки зображень і машинного навчання, що дає змогу ідентифікувати та класифікувати елементи на зображеннях із високою точністю [4]. За останні роки цей напрям активно розвивався завдяки появі глибоких нейронних мереж, які значно перевершили традиційні підходи, такі як використання гістограм орієнтованих градієнтів (HOG) чи методів на основі ключових точок (SIFT, SURF).

Серед сучасних методів розпізнавання об'єктів особливе місце займають згорткові нейронні мережі (CNN), які стали основою для багатьох алгоритмів [1], [14].

Одним із перших успішних прикладів є модель R-CNN (Region-based Convolutional Neural Network), запропонована у 2014 році [5]. Вона використовувала підхід із виділенням регіонів пропозицій, які потім класифікувались за допомогою CNN. Проте цей метод був повільним через багатоступеневий процес обробки. Подальший розвиток призвів до появи швидших алгоритмів, таких як Fast R-CNN і Faster R-CNN, які оптимізували обчислення завдяки інтеграції регіональних пропозицій у єдину мережу [5]. Ці моделі широко застосовуються там, де потрібна висока точність, наприклад у медичній діагностиці чи автономному водінні.

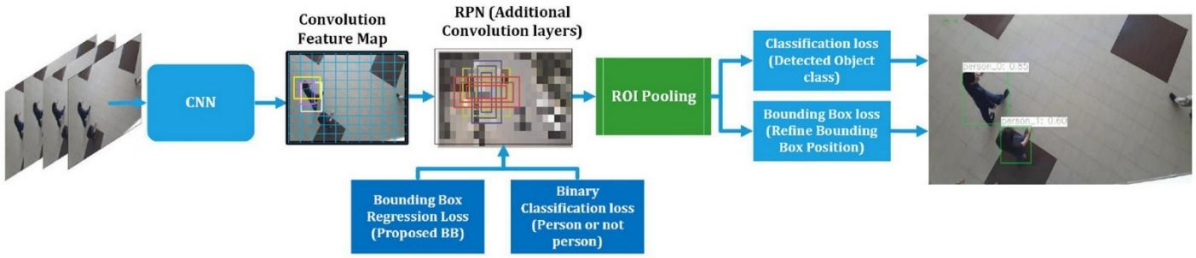


Рисунок 1.1 – Схема виявлення людей зверху за допомогою Faster-RCNN [1].

Паралельно розвивалися одноетапні детектори, які поєднують виявлення та класифікацію об’єктів в одному проході мережі, що значно підвищує швидкість обробки. Найвідомішими представниками цього класу є сімейство алгоритмів YOLO (You Only Look Once). Перша версія YOLO, представлена у 2016 році, розглядала зображення як сітку, прогнозуючи межі об’єктів і їхні класи для кожної клітинки [11].

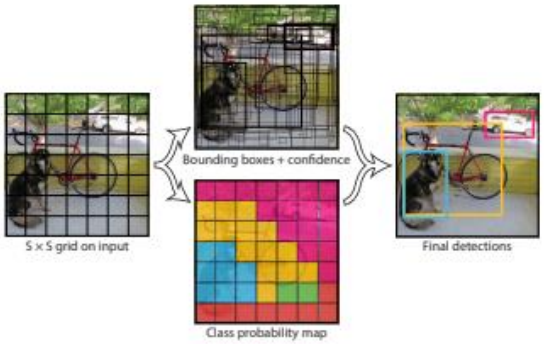


Рисунок 1.2 – Принцип роботи алгоритму YOLO: поділ зображення на сітку для одноетапної детекції [11].

Подальші версії, зокрема YOLOv3, YOLOv4 і YOLOv8, покращили точність і продуктивність завдяки використанню складніших архітектур, таких як Darknet та CSPNet [16], а також оптимізаціям, наприклад, багаторівневому передбаченню для об’єктів різного розміру. YOLOv8, розроблений командою Ultralytics, є однією з найсучасніших моделей, яка забезпечує баланс між швидкістю та точністю, що робить її придатною для задач реального часу [12].

Іншим популярним одноетапним методом є SSD (Single Shot MultiBox Detector), який використовує кілька згорткових шарів для прогнозування об’єктів різного масштабу [9]. SSD є швидшим за R-CNN, але поступається YOLO у швидкості обробки великих наборів даних. Окрім того, у задачах із обмеженими обчислювальними ресурсами застосовуються легші моделі, такі як MobileNet, які оптимізовано для роботи на вбудованих системах [13]. Ці моделі часто використовуються в платформах типу Edge Impulse, де акцент робиться на простоті розгортання.

У контексті розпізнавання радіодеталей важливим є вибір методу, який враховує специфіку об’єктів: невеликий розмір, схожість форм і текстур, а також можливі артефакти на зображеннях [8]. Одноетапні детектори, такі як YOLO, є перспективними для таких задач завдяки їхній здатності швидко обробляти зображення та адаптуватися до складних умов. Саме тому в даній роботі акцент зроблено на використанні YOLOv8, що дозволяє ефективно вирішувати задачу ідентифікації радіодеталей як у статичних зображеннях, так і в реальному часі.

1.2. Аналіз вимог до автоматизованого розпізнавання радіодеталей

Автоматизоване розпізнавання радіодеталей є важливою задачею в контексті інтеграції комп’ютерного зору в роботизовані системи, що застосовуються у виробництві електроніки. Радіодеталі, такі як конденсатори, діоди, світлодіоди, резистори та фон як окремий клас, мають специфічні характеристики, які необхідно враховувати при розробці моделі штучного зору. Аналіз вимог до таких систем дозволяє визначити ключові аспекти, які впливають на їхню ефективність і практичну застосовність.

Першою вимогою є висока точність класифікації об’єктів. Радіодеталі часто мають схожі форми, розміри та кольори, що ускладнює їхнє розрізнення [17]. Наприклад, конденсатори та резистори можуть бути подібними за контурами. У зв’язку з цим модель має забезпечувати точність не нижче 95%, щоб мінімізувати помилки в ідентифікації, які можуть призвести до збоїв у подальших етапах автоматизованого процесу, наприклад, при сортуванні чи складанні. Крім того, необхідно враховувати ймовірність правильного визначення кожного класу, що вимірюється через показник упевненості (confidence score), середнє значення якого для всіх класів має бути не нижче 90% та не нижче 80% для окремого об'єкту.

Другою важливою вимогою є здатність системи працювати зі зображеннями різної якості. У реальних умовах радіодеталі можуть бути зняті з різним освітленням, кутами огляду, а також із шумом чи артефактами, такими як розмиття чи відблиски. Це вимагає від моделі стійкості до таких змін і

можливості узагальнення на основі різноманітних даних. У даній роботі це було враховано шляхом завдання відповідних параметрів при навчанні моделі та використання аугментованого набору даних, який включає зображення з різними умовами, що імітують реальні виробничі сценарії.

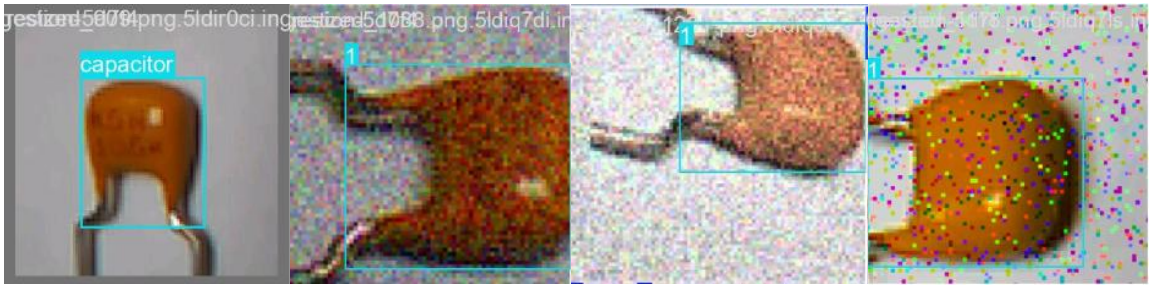


Рисунок 1.3 – Приклади зображень конденсатора із різними умовами зйомки.

Третьою вимогою є швидкість обробки зображень. У задачах автоматизації, особливо якщо система інтегрується з роботами, час реакції має бути мінімальним. Для пакетної обробки великої кількості зображень (3250 фото) бажано досягти часу обробки в межах 30 секунд, а для аналізу в реальному часі забезпечити стабільну роботу з частотою 10-15 FPS (кадрів за секунду). Це ставить вимоги до оптимізації моделі, через використання батчової обробки та ефективних алгоритмів, таких як YOLOv8, який відомий своєю швидкодією. Батчова обробка – метод обробки даних, при якому велика кількість даних (наприклад, зображень) обробляється одночасно як єдиний набір, а не по одному елементу.

Четверта вимога стосується адаптивності моделі до різних джерел даних. У виробничих умовах зображення можуть надходити як із статичних камер (наприклад, для аналізу з папки), так і з динамічних джерел наприклад роботизованих систем. У роботі це реалізовано через розробку двох програмних модулів: один для обробки статичних зображень, інший – для потокового відео.

Нарешті, п’ята вимога пов’язана з практичною реалізацією. Система має бути простою у розгортанні та не потребувати надмірних обчислювальних ресурсів, щоб її можна було використовувати на стандартному обладнанні [6],

наприклад, звичайному комп’ютері. Водночас результати роботи моделі повинні бути зрозумілими та придатними для подальшого використання, наприклад у вигляді зображень із мітками, лог-файлів чи графіків, які демонструють точність і розподіл класів.

Автоматизоване розпізнавання радіодеталей потребує моделі, яка поєднує високу точність, стійкість до змін умов, швидкість обробки, адаптивність до різних джерел даних і простоту реалізації. Ці вимоги лягли в основу вибору інструментів і підходів у даній роботі, зокрема використання YOLOv8 як основи моделі та розробки програмного забезпечення для її практичного застосування.

1.3. Формулювання задачі розробки моделі штучного зору

На основі проведеного огляду сучасних методів комп’ютерного зору та аналізу вимог до автоматизованого розпізнавання радіодеталей можна чітко сформулювати задачу роботи. Основною метою є створення моделі штучного зору, яка здатна ідентифікувати та класифікувати радіодеталі (конденсатори, діоди, світлодіоди, резистори та фон як окремий клас) на зображеннях із різними умовами зйомки, а також розробка програмного забезпечення для її практичного використання в статичному та динамічному режимах. Ця задача потребує комплексного підходу, що охоплює підготовку даних, вибір алгоритму, програмну реалізацію та оцінку результатів.

Першим етапом задачі є підготовка даних для навчання моделі. Вихідним матеріалом слугує базовий набор даних із 250 зображень. Цей набір включає прості фотографії радіодеталей із чіткими контурами та стабільним освітленням, що дозволяє швидко перевірити працездатність базових моделей. Однак для реальних умов цього недостатньо, тому для фінальної моделі було використано аугментований набор даних із 3250 зображень. Аугментація включає додавання шумів, зміну кутів огляду, а також імітацію руху, що відображає складніші сценарії, з якими може зіткнутися система в промислових умовах [2]. Усі зображення приведено до однакової роздільної здатності 320x320 пікселів, що є компромісом між якістю деталізації та обчислювальною ефективністю.

Другим етапом є вибір і адаптація алгоритму машинного навчання. З огляду на вимоги до швидкості та точності, а також специфіку одноетапних детекторів, було обрано бібліотеку Ultralytics, яка включає моделі YOLOv8 та YOLOv8x.

Цей вибір зумовлений кількома факторами:

- YOLOv8/x забезпечує високу продуктивність завдяки оптимізованій архітектурі, яка дозволяє обробляти зображення за один прохід мережі;
- Модель підтримує гнучке налаштування параметрів, таких як поріг упевненості (confidence threshold) та IoU (Intersection over Union), що важливо для точного виділення об’єктів;
- YOLOv8/x добре адаптується до аугментованих даних, що критично для роботи з 3250 зображеннями.

У процесі розробки планується використати передтреновану модель YOLOv8x (збільшеної глибини), яку буде донавчено на підготовленому набору даних для досягнення середньої впевненості не нижче 90%.

Третім етапом задачі є створення програмного забезпечення для практичного застосування моделі. Перший програмний модуль призначений для пакетної обробки зображень із папки.

Він має виконувати такі функції:

- Завантаження зображень.
- Аналіз зображень із використанням навченої моделі.
- Додавання міток із назвами класів і ймовірностями правильності.
- Збереження результатів у вихідну папку.
- Ведення статистики точності аналізу та кількості виявлених радіодеталей.
- Підрахунок часу виконання програми.
- Генерація та ведення лог-файлу з інформацією про виявлені об’єкти та статистикою.
- Побудова аналітичних графіків.

Для оптимізації цього модуля передбачено використання батчової обробки [2].

Другий модуль орієнтований на аналіз у реальному часі з камери телефону через EpocCam, камера телефону використовується для імітації зору робота. Він адаптує модель до потокового відео, забезпечуючи стабільне розпізнавання при зміні умов, таких як рух чи освітлення, і зберігає результати у вигляді анотованих кадрів [10]. Другий модуль має виконувати аналогічні функції, що модуль пакетної обробки, але у реальному часі.

Останнім етапом є оцінка ефективності розробленої системи. Основними критеріями виступають точність класифікації (не нижче 95%), швидкість обробки (30 секунд для пакетного аналізу та 10-15 FPS у реальному часі) і стійкість до змін умов зйомки. Результати мають бути представлені у вигляді зображень із мітками, графіків точності та кількості виявлених об’єктів, а також лог-файлів для аналізу.

Задачу можна сформулювати наступним чином:

1. Розробити модель штучного зору на базі YOLOv8x для розпізнавання п’яти класів радіодеталей із середньою точністю не нижче 90%.
2. Створити два програмні модулі для пакетної обробки зображень і аналізу в реальному часі, оптимізувати їх для швидкості та стабільності, а також додати ведення аналітичних даних.
3. Оцінити ефективність системи за ключовими показниками.

Ці задачі є основою для практичної реалізації, яка буде детально описана в наступних розділах.

Висновки за розділом 1.

У першому розділі було визначено актуальність розробки системи розпізнавання радіодеталей для промислової автоматизації. Мета роботи – створити модель штучного зору для автоматичного розпізнавання радіодеталей та програмне забезпечення для її застосування.

Були поставлені задачі, що включають аналіз методів комп'ютерного зору, підготовку даних, розробку моделі, створення програмних модулів та оцінку ефективності. Об'єктом дослідження є процес розпізнавання радіодеталей за допомогою комп’ютерного зору, предметом – модель штучного зору на базі YOLOv8x та програмні модулі для її практичного застосування.

Проаналізовано сучасні методи комп'ютерного зору, зокрема глибокі нейронні мережі та одноетапні детектори (YOLO, SSD), визначивши YOLOv8x як оптимальний вибір для швидкого та точного розпізнавання.

Сформульовано ключові вимоги до системи: висока точність (не нижче 95%, confidence >80%), стійкість до змін умов зйомки (шум, освітлення, ракурси), швидкість обробки (пакетна <30с для 3250 зображень, реальний час 10-15 FPS), адаптивність до джерел даних та простота реалізації. Задача полягає у розробці моделі YOLOv8x для п'яти класів радіодеталей з точністю $\geq 95\%$ та двох програмних модулів для пакетної та реального часу обробки.

РОЗДІЛ 2.
РОЗРОБКА ТА ПІДГОТОВКА ДАНИХ ДЛЯ МОДЕЛІ

2.1. Характеристика наборів даних: базовий та аугментований

Підготовка даних є важливим етапом у розробці моделі комп'ютерного зору, адже від якості, кількості та різноманітності навчального набору залежить здатність моделі точно розпізнавати об'єкти та узагальнювати знання на нові зображення [4]. Було використано два набори даних: базовий набір із 250 зображень і аугментований набір із 3250 зображень. Обидва набори даних складаються з п'яти класів: конденсатори (capacitor), діоди (diode), світлодіоди (led), резистори (resistor) і фон (background). Останній клас включено як окремий для підвищення коректності моделі, щоб вона могла відрізняти радіодеталі від сторонніх об'єктів чи поверхонь, що є важливим у реальних умовах.

Базовий набір даних був отриманий з курсу Computer Vision with Embedded Machine Learning, який є відкритим навчальним ресурсом для вивчення комп'ютерного зору на вбудованих системах [7]. Цей набір складається з 250 фотографій радіодеталей, виконаних у контрольованих умовах із використанням стандартного обладнання. Кожне зображення має початкову роздільну здатність 96x96 пікселів, що є досить низьким показником для детального аналізу, але достатнім для попередніх експериментів. Фотографії характеризуються чіткими контурами об'єктів, стабільним і рівномірним освітленням, а також відсутністю значних шумів, розмиття чи інших артефактів (див. рис. 2.1).

Розподіл класів у базовому наборі є рівномірним: кожен із п'яти класів представлений 50 зображеннями. Перевагою цього набору даних є його простота і компактність, що зробило його ідеальним для швидкого тестування на платформі Edge Impulse. Однак його обмеження очевидні: мала роздільна здатність (96x96 пікселів), а одноманітність умов зйомки не дає змоги моделі адаптуватися до реальних сценаріїв, де можуть бути присутні шум, перекриття об'єктів чи нестандартні ракурси.

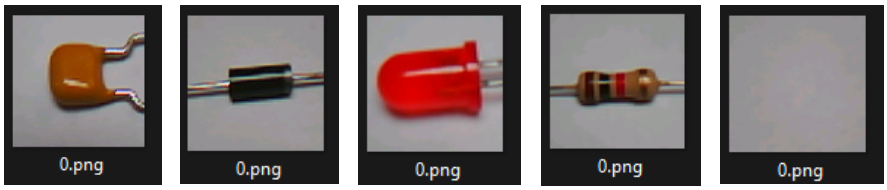


Рисунок 2.1 – Вигляд базового набору даних з роздільною здатністю 96x96.

Щоб подолати недоліки базового набору, треба використовувати аугментований набор даних із 3250 зображень з того ж курсу, він також має роздільну здатність 96x96. Цей набір піддано численним трансформаціям для створення різноманітних умов щоб імітувати складніші сценарії, які можуть виникати у виробничому середовищі.

Зокрема, присутні наступні трансформації:

- Шум на зображенні, що імітує низькоякісні камери чи забруднення об’єктива або технічні неполадки.
- Артефакти на зображенні, що представляє собою точки різних кольорів.
- Повороти зображень на кути від -45 до +45 градусів для врахування різних орієнтацій об’єктів.
- Масштабування об’єктів (збільшення чи зменшення до 50% від початкового розміру), щоб модель могла розпізнавати деталі різного масштабу.
- Показ радіодеталі частково.
- Розмиття у випадкових напрямках для імітації руху камери чи об’єкта під час зйомки.

Кожен із цих методів застосовувався у різних комбінаціях, що дозволило значно розширити початковий набір даних.

Зображення аугментованого набору даних також мають роздільну здатність 96x96 пікселів, для збільшення якості навчання моделі усі зображення було приведено до роздільної здатності 320x320 пікселів за допомогою розробленого скрипта stetch.py (див. Додаток Г, Лістинг Г.3) [3], який використовує бібліотеку OpenCV із методом інтерполяції INTER_AREA для

збереження якості. Роздільна здатність 320x320 була обрана як оптимальна, вона забезпечує кращу деталізацію об’єктів порівняно з 96x96 пікселів, але залишається прийнятною для обчислювальних ресурсів.

Розподіл аугментованого набору даних організовано за стандартним підходом у машинному навчанні: 2600 зображень (80%) відведено для тренування моделі, а 650 зображень (20%) для тестування.

Таблиця 2.1.

Розподіл аугментованого датасету

Клас	Тренувальний набір	Тестовий набір	Загалом
Background	520	130	650
Capacitor	520	130	650
Diode	520	130	650
Led	520	130	650
Resistor	520	130	650
Всього	2600	650	3250

Такий розподіл дозволяє забезпечити достатню кількість даних для навчання, зберігаючи при цьому незалежний тестовий набір для оцінки результатів. Завдяки аугментації кожен клас отримав значно більше варіацій. Наприклад, якщо в базовому наборі діод був представлений лише в одному ракурсі з чітким фоном, то в аугментованому він з’являється в десятках варіантів: із шумом, поворотами, затемненням чи розмиттям.

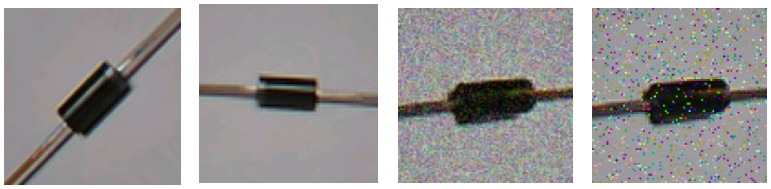


Рисунок 2.2 – Приклад аугментованих зображень діода.

Фон також аугментуваний, щоб модель могла краще відокремлювати радіодеталі від навколишнього середовища.

Базовий набор даних із 250 зображень став відправною точкою для початкових експериментів завдяки своїй простоті, тоді як аугментований набір із 3250 зображень забезпечив необхідну різноманітність і обсяг для створення остаточної моделі.

2.2. Перетворення зображень і міток

Ефективна підготовка даних для моделі комп'ютерного зору передбачає не лише створення різноманітного набору даних, а й його адаптацію до вимог обраних алгоритмів і інструментів. Ключовими етапами підготовки стали зміна роздільної здатності зображень із початкових 96x96 до 320x320 пікселів та конвертація міток із формату Edge Impulse у формат, сумісний із бібліотекою YOLO. Ці завдання було реалізовано за допомогою двох спеціально розроблених скриптів: stetch.py (див. Додаток Г, Лістинг Г.3) для обробки зображень і convert_labels.py (див. Додаток Г, Лістинг Г.4) для трансформації міток. Обидва скрипти написані мовою Python із використанням бібліотек OpenCV і стандартних модулів для роботи з файлами та JSON-структурами.

Першим кроком стало перетворення роздільної здатності зображень. Базовий набор даних із 250 фотографій, отриманий із курсу Coursera, мав роздільну здатність 96x96 пікселів, що було достатньо для початкових експериментів на платформі Edge Impulse, але недостатньо для детального розпізнавання об'єктів у складніших умовах. Низька роздільна здатність обмежувала якість детекції дрібних радіодеталей та не сумісна з більшістю алгоритмів навчання моделі. Для вирішення цієї проблеми було прийнято рішення збільшити розмір зображень до 320x320 пікселів, що є стандартним значенням для багатьох сучасних моделей, зокрема YOLOv8, і забезпечує баланс між деталізацією та обчислювальною складністю.

Для реалізації цього завдання розроблено скрипт stetch.py (див. Додаток Г, Лістинг Г.3), який базується на бібліотеці OpenCV.

Скрипт виконує такі дії:

- Визначається вихідний каталог із зображеннями, де зберігаються фото.
- Створюється цільовий каталог, куди записуються оброблені зображення.
- Скрипт підтримує обробку файлів із розширеннями .png, .jpg і .jpeg, що охоплює всі формати, використані в наборі даних.
- Для кожного зображення виконується завантаження за допомогою функції cv2.imread().
- Зміна розміру до 320x320 пікселів із використанням методу інтерполяції cv2.INTER_AREA. Цей метод був обраний через його здатність зберігати якість при збільшенні зображень, уникаючи надмірної пікселізації чи розмиття.
- Змінені зображення зберігаються з новими назвами (наприклад, resized_1.png), що дозволяє відстежувати їхній порядок і уникати перезапису оригіналів.
- У разі помилки завантаження (наприклад, якщо файл пошкоджено) скрипт виводить повідомлення і пропускає таке зображення, забезпечуючи стабільність процесу.

Перетворення було застосовано як до базового набору з 250 зображень, так і до аугментованого набору з 3250 зображень. У результаті всі фотографії отримали однакову роздільну здатність 320x320 пікселів, що забезпечило одноманітність даних для подальшого навчання. Збільшення розміру дозволило моделям краще визначати межі об’єктів.

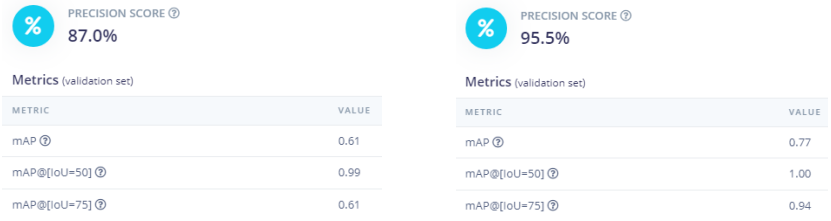


Рисунок 2.3 – Порівняння точності базових моделей з розміром зображень 96x96(87%) vs 320x320(95,5%).

Другим етапом підготовки стала конвертація попередньо експортованих міток класифікації зображень із формату Edge Impulse у формат YOLO, необхідний для локального навчання моделі з бібліотекою Ultralytics. Експортований файл зберігає розмітку у вигляді файлу bounding_boxes.labels – JSON-структури, де для кожного зображення вказувалися координати меж (x, y, width, height) і назви класів (background, capacitor тощо). Цей формат використовується на сайті Edge Impulse, але є несумісним із YOLO, який вимагає текстових файлів (.txt) із нормалізованими координатами (x_center, y_center, width, height) для кожного зображення. Для виконання цього завдання розроблено скрипт convert_labels.py.(див. Додаток Г, Лістинг Г.4).

Скрипт convert_labels.py (див. Додаток Г, Лістинг Г.4) працює наступним чином:

- Визначаються каталоги: вихідний із зображеннями та мітками і цільовий для збереження конвертованих міток.
- Скрипт зчитує файл bounding_boxes.labels, витягує інформацію про межі та класи, а потім перевіряє наявність відповідних зображень у каталозі.
- Для кожного зображення створюється окремий .txt-файл із назвою, що відповідає імені зображення.
- Класи переводяться у числові ідентифікатори за словником CLASS_MAP (background – 0, capacitor – 1 тощо).
- Координати перераховуються з абсолютних значень у нормалізовані.
- Значення записуються у файл у форматі YOLO: "class_id x_center y_center norm_width norm_height" [11].

Рисунок 2.4 – Приклад мітки у форматі YOLO TXT.

Конвертація виконана як для тренувального, так і тестового набору зображень. Мітки базового та аугментованого наборів даних були успішно переведені у формат YOLO. Скрипт також включає перевірку на помилки: якщо клас невідомий чи зображення відсутнє, видається попередження, що дозволяє уникнути некоректних даних. У результаті було отримано повний набір міток для 3250 зображень, розподілених на 2600 тренувальних і 650 тестових, що стало основою для навчання YOLOv8.

Скрипти для зміни роздільної здатності зображення і конвертер з JSON до YOLO забезпечили перетворення зображень і міток відповідно до потреб проекту. Зміна роздільної здатності до 320x320 пікселів підвищила якість детекції, а конвертація міток у формат YOLO дозволила перейти від експериментів на Edge Impulse до локального навчання з Ultralytics. Ці етапи підготували дані для створення високоточної моделі розпізнавання радіодеталей, що буде детально описано далі.

2.3. Попереднє тестування на Edge Impulse: результати та обмеження

Платформа Edge Impulse стала початковим інструментом для тестування можливостей розпізнавання радіодеталей, оскільки сайт пропонує зручний інтерфейс і набір готових рішень для створення моделей комп’ютерного зору, орієнтованих на вбудовані системи [3]. Цей етап дозволив отримати перші результати, виявити сильні сторони та обмеження Edge Impulse, а також визначити подальший напрям розробки.

Спочатку було завантаження базовий набор даних з 250 зображень розміром 320x320 пікселів до сайту. Потім було виконано розмітку кожного зображення для подальшого навчання моделі на основі ручної класифікації. Платформа автоматично розподілила дані на тренувальний (80%, 200 зображень) і тестовий (20%, 50 зображень). Після класифікації на основі 200 тренувальних зображень було створено графік розподілення класів.

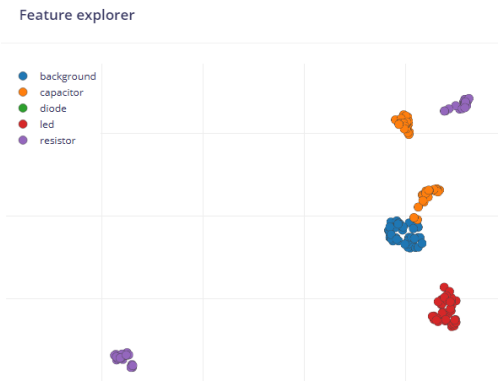


Рисунок 2.5 – Графік розподілення класів в залежності від подібності.

На графіку можна спостерігати що кожен клас радіодеталей згруповано по подібності. Простір між класами говорить про унікальні відмінні властивості для кожного класу. Чим ближче точки одна до одної – тим більше їх подібність. Можна спостерігати що resistor і capacitor розбилися на 2 зони, що обумовлено віддзеркаленими фотографіями resistor і іншою стороною для capacitor(у конденсатора одна сторона з надписом, а інша без).

У розділі Object Detection було обрано найкращу доступну модель на сайті: MobileNetV2 SSD FPN-Lite (320x320). Модель є легкою згортковою нейронною мережею, оптимізованою для швидкої роботи на пристроях із обмеженими ресурсами.

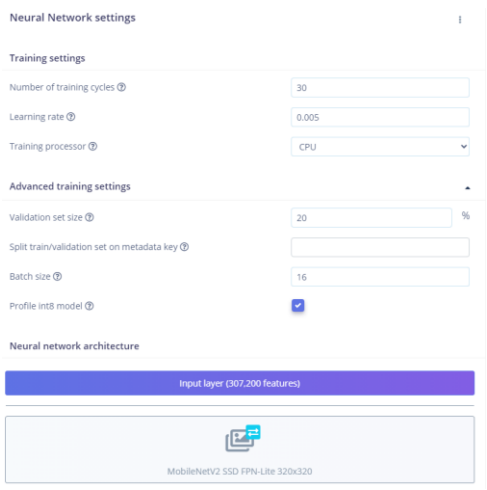


Рисунок 2.6 – Параметри навчання моделі на сайті Edge Impulse.

Тестування отриманої моделі на платформі демонструє точність 92% на тестовому наборі з 50 зображень.

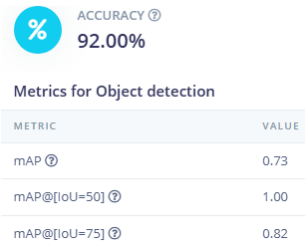


Рисунок 2.7 – Точність моделі на основі 50 тестових зображень.

Тестування базової моделі на аугментованому набору даних. Платформа надає великий вибір форматів експортування розробленої моделі, для локальної пакетної обробки найбільш підходить метод Linux x86 з форматом моделі .eim. Модель було експортовано та використано базовий скрипт на python з курсу Computer Vision with Embedded Machine Learning для аналізу аугментованого набору даних з 3250 фото [7]. Базова модель показала точність близьку до 70% коректного виявлення радіодеталей. Так як модель навчалась на невеликому і простому набору даних результат у 70% точності є задовільним.

Рисунок 2.8 – Вибірка результату аналізу базової моделі на аугментованих даних, можна спостерігати помилковий аналіз та не чіткі мітки.

Розробка більш складної моделі на основі аугментованого набору даних. Щоб класифікувати вручну 3250 фотографій потрібно приблизно 10 годин часу, через це прийнято рішення автоматизації цього процесу. Для автоматизації використано базову модель, а саме її можливість до розпізнавання радіодеталей.

Задумка полягає в тому, щоб за допомогою базової моделі створити мітки класифікації аугментованого набору даних як при звичайному аналізі і запис координат цих міток, але модель у форматі .eim може лише малювати мітки і не

має можливості створити файл JSON або інший формат який би зберігав координати меж x, y, width, height.

Тому було створено новий скрипт classify_3250_photos.py (див. Додаток Г, Лістинг Г.5). Скрипт реалізує автоматизований процес розпізнавання радіодеталей на 3250 зображеннях, використовуючи легку модель YOLOv8n, яка одразу тренується на 250 зображеннях базової моделі які зберігаються у YOLO форматі з мітками. Після аналізу скрипт генерує розмітку у форматі COCO JSON для аугментованого набору.

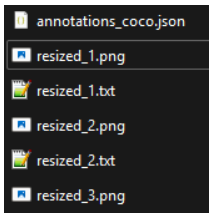


Рисунок 2.9 – Вигляд файлу з мітками створеного за допомогою скрипту.

Було отримано готові мітки у форматі COCO JSON для 2300 зображень(70%), а решта 950 мали помилки, або не розпізнанні і потребували доопрацювання вручну. Отримані файли розмітки були імпортовані знов на платформу Edge Impulse для створення нової моделі. Через інтерфейс сайту було дороблено і виправлено 950 зображень що містили полковий аналіз, чи зовсім його не мали.

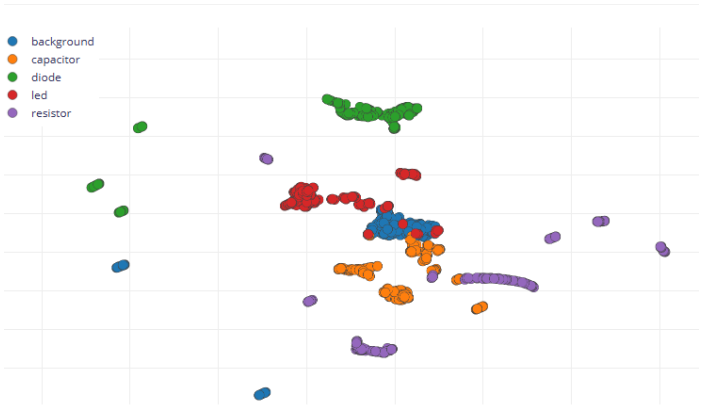


Рисунок 2.10 – Графік розподілу радіодеталей по відмінності для аугментованого набору даних після ручного виправлення помилок.

На графіку можна спостерігати розбиття кожного класу радіодеталей на кілька підкласів, що зумовлено присутністю зображень із складними умовами зйомки.

Наступним кроком після класифікації 3250 фото було навчання моделі, проте тут виявилися суттєві обмеження платформи. Edge Impulse має ліміт у 20 хвилин на генерацію моделі в безкоштовній версії, що виявилося недостатнім для обробки 3250 зображень. Навіть при низькій кількості епох навчання (10) процесу навчання було потрібно більше однієї години. Це обумовлюється тим що платформа орієнтована на дуже легкі моделі для вбудованих систем і також використовує свої сервери для навчання моделі, де використовуються лише CPU, що значно збільшує час навчання. Через це прийнято рішення навчити модель на локальному обладнанні, де немає лімітів.

Результати тестування на Edge Impulse мали як позитивні, так і негативні сторони.

З позитивного:

- Базова модель MobileNetV2 на 250 зображеннях показала високу точність (92%), що підтвердило працездатність підходу до розпізнавання радіодеталей.
- Імпорт та експорт даних у платформу пройшов без значних труднощів.

Однак обмеження стали вирішальними:

- Низька точність (70%) на аугментованому наборі.
- Часові ліміт на 20 хвилин навчання моделі.
- Недостатня гнучкість і неефективність для великих наборів.

Саме тому було експортовано виправлений набір аугментованих даних з Edge Impulse у форматі .label, конвертовано через скрипт convert_labels.py (див. Додаток Г, Лістинг Г.4) розроблений раніше до формату міток YOLO і продовження розробки на локальному обладнанні з використанням бібліотеки Ultralytics YOLOv8.

Попереднє тестування на Edge Impulse стало важливим етапом для оцінки базових можливостей і підготовки даних, але виявлені обмеження платформи визначили перехід до більш гнучкого і продуктивного рішення. Отримані результати (92% на базовому наборі та 70% на аугментованому) слугували орієнтиром для подальшого вдосконалення моделі, яке буде описано в наступних розділах.

Висновки за розділом 2.

У другому розділі висвітлено важливий етап підготовки даних. Використано два набори даних: базовий (250 зображень) для початкових експериментів та аугментований (3250 зображень) для навчання фінальної моделі.

Набори включають п'ять класів: конденсатори, діоди, світлодіоди, резистори та фон. Аугментований набір отримано шляхом застосування численних трансформацій (шум, артефакти, повороти, масштабування, розмиття, частковий показ) для імітації реальних виробничих умов та підвищення адаптивності моделі. Усі зображення приведено до однакової роздільної здатності 320x320 пікселів.

Аугментований набір розподіллено на 80% для тренування (2600 зображень) та 20% для тестування (650 зображень). Виконано перетворення міток зображень у формат YOLO TXT. Попереднє тестування на платформі Edge Impulse на базовому наборі показало високу точність (92%), але на аугментованому лише 70%. Обмеження платформи стали підставою для переходу до локального навчання.

РОЗДІЛ 3.
СТВОРЕННЯ МОДЕЛІ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА БАЗІ YOLOV8.

3.1. Тренування моделі YOLOv8x: параметри та результати.

Після попереднього тестування на платформі Edge Impulse і виявлення її обмежень для обробки великих аугментованих даних було прийнято рішення перейти до локального навчання з використанням бібліотеки Ultralytics YOLOv8. Цей вибір зумовлений здатністю YOLOv8 ефективно працювати з великими наборами даних, високою швидкістю обробки та можливістю гнучкого налаштування параметрів, що ідеально відповідає вимогам задачі розпізнавання радіодеталей [16]. Обрано модель YOLOv8x – одну з найглибших і найточніших версій у сімействі YOLOv8, яка була навчена на підготовленому наборі даних з 3250 зображень. Процес тренування реалізовано за допомогою скрипта yolox_model.py (див. Додаток Г, Лістинг Г.6).

Першим етапом було підготовче налаштування середовища та даних. Для роботи використано локальну систему з графічним процесором RTX 4070, що забезпечило достатню обчислювальну потужність для швидкого навчання. Набор даних із 3250 зображень розміром 320x320 пікселів з мітками, підготовлений на попередніх етапах (2600 тренувальних і 650 тестових, мітки у форматі YOLO TXT).

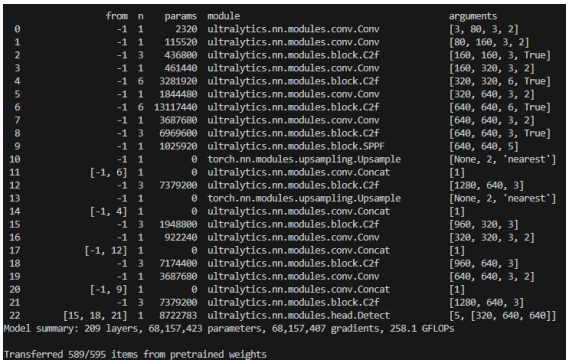
Для конфігурації тренування створено файл data.yaml, який містив шляхи до тренувальних і тестових даних, кількість класів (nc: 5) і їхні назви (background, capacitor, diode, led, resistor). Цей файл генерувався автоматично функцією create_data_yaml у скрипті, що спростило процес підготовки.

data.yaml

```
1  train: /mnt/e/univer/4k2s/dyplom/WSL/yolo/3250model/training
2  val: /mnt/e/univer/4k2s/dyplom/WSL/yolo/3250model/testing
3  nc: 5
4  names: ['background', 'capacitor', 'diode', 'led', 'resistor']
5  path: /mnt/e/univer/4k2s/dyplom/WSL/yolo/3250model
```

Рисунок 3.1 – Конфігурація шляхів та класів у файлі data.yaml.

Тренування розпочалося з ініціалізації передтренованої моделі YOLOv8x, завантаженої з бібліотеки Ultralytics. Використання передтренованої моделі дозволило прискорити навчання завдяки перенесенню знань (transfer learning) із загального набору даних, такого як COCO, на специфічний набор радіодеталей.



	from	n	params	module	arguments
0	-1	1	2320	ultralytics.nn.modules.conv.Conv	[3, 80, 3, 2]
1	-1	1	115520	ultralytics.nn.modules.conv.Conv	[80, 160, 3, 2]
2	-1	3	436800	ultralytics.nn.modules.block.C2f	[160, 160, 3, True]
3	-1	1	461440	ultralytics.nn.modules.conv.Conv	[160, 320, 3, 2]
4	-1	6	3281920	ultralytics.nn.modules.block.C2f	[320, 320, 6, True]
5	-1	1	1844480	ultralytics.nn.modules.conv.Conv	[320, 640, 3, 2]
6	-1	6	13117440	ultralytics.nn.modules.block.C2f	[640, 640, 6, True]
7	-1	1	3687680	ultralytics.nn.modules.conv.Conv	[640, 640, 3, 2]
8	-1	3	6969600	ultralytics.nn.modules.block.C2f	[640, 640, 3, True]
9	-1	1	1025920	ultralytics.nn.modules.block.SPPF	[640, 640, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
12	-1	3	7379200	ultralytics.nn.modules.block.C2f	[1280, 640, 3]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
15	-1	3	1948800	ultralytics.nn.modules.block.C2f	[960, 320, 3]
16	-1	1	922240	ultralytics.nn.modules.conv.Conv	[320, 320, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
18	-1	3	7174400	ultralytics.nn.modules.block.C2f	[960, 640, 3]
19	-1	1	3687680	ultralytics.nn.modules.conv.Conv	[640, 640, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]
21	-1	3	7379200	ultralytics.nn.modules.block.C2f	[1280, 640, 3]
22	[15, 18, 21]	1	8722783	ultralytics.nn.modules.head.Detect	[5, [320, 640, 640]]

Model summary: 209 layers, 68,157,423 parameters, 68,157,407 gradients, 258.1 GFLOPs

Transferred 589/595 items from pretrained weights

Рисунок 3.2 – Загальний набор предтренерованих даних який використовувався.

У скрипті yolox_model.py (див. Додаток Г, Лістинг Г.6) було задано основні параметри тренування:

- Кількість епох(циклів навчання) – 50.
- розмір зображень (imgsz) – 320.
- розмір батчу(кількість паралельних процесів) – 16.

Для підвищення узагальнюючої здатності моделі застосовано аугментацію даних із такими параметрами:

- hsv_h=0.015 (зміна відтінку),
- hsv_s=0.7 (насиченість),
- hsv_v=0.4 (яскравість),
- translate=0.1 (зсув),
- scale=0.5 (масштабування),
- fliplr=0.5 (50% ймовірність горизонтального відображення).

Обертання (degrees), нахил (shear) і перспективні викривлення (perspective) відключено, щоб зберегти стабільність для радіодеталей, які зазвичай мають фіксовану форму.

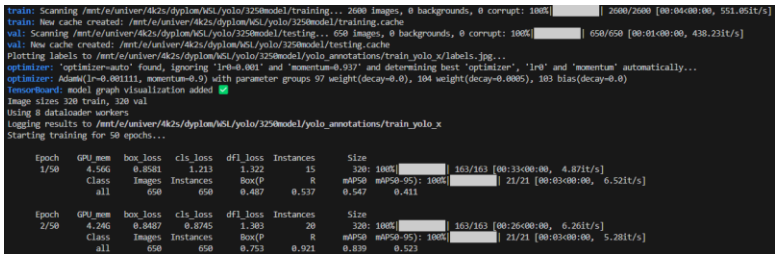


Рисунок 3.3 – Імпорт набору даних та початок тренування моделі.

Оптимізація моделі здійснювалася за допомогою адаптивного алгоритму навчання з початковою швидкістю (lr0) 0.001 і кінцевою (lrf) 0.01, що забезпечило поступове зниження швидкості для уникнення перетренування. Параметр patience=20 увімкнув ранню зупинку, якщо показник mAP50 (mean Average Precision при IoU=0.5) не покращувався протягом 20 епох. Тренування проводилося на GPU, а результати зберігалися в каталозі. Процес тривав приблизно 25 хвилин, що швидко для набору з 3250 зображень і 50 епох. Логування показників (loss, mAP тощо) відбувалося автоматично, а найкраща модель зберігалася у файлі best.pt.

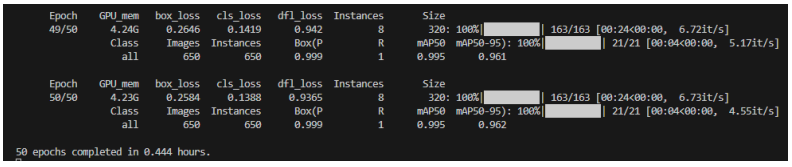


Рисунок 3.4 – Закінчення навчання моделі.

По завершенню тренування модель була оцінена на тестовому наборі з 650 зображень за допомогою методу model.val(). Основними показниками якості обрано mAP50, який відображає середню точність знаходження об’єкту при порозі IoU=0.5, та mAP50-95, який відображає наскільки модель добре здатна вказувати границі об’єкту та його місцеположення.

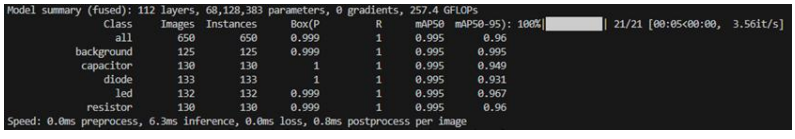


Рисунок 3.5 – Оцінка моделі на тестових даних.

Середній результат maP50 = 0.995 (із 1 можливих), та maP50-95 = 0.96, що вказує на високу точність знаходження об’єкту(99.5%) та на високу точність визначення границь об’єктів(96%). Детальний аналіз показав, що моделі складніше усього виявити границі діода та конденсатора – 93,1% для діода та 94,9% для конденсатора відповідно. Це пов’язано через більшу складність форми цих радіодеталей та з неідеальною класифікацією, що є людським фактором. Інші радіодеталі мають точність визначення границі більше 95% що майже ідеально.

Тренування моделі YOLOv8x пройшло успішно, досягнуто високої точності (96%) на тестових даних і модель готова до інтеграції з програмами для аналізу радіодеталей.

3.2. Розробка програми для аналізу зображень із папки

Після успішного тренування моделі наступним етапом стала розробка програмного забезпечення для її практичного використання. Одним із ключових компонентів роботи є програма для пакетної обробки зображень із папки, яка реалізує аналіз статичних даних, додає мітки до виявлених радіодеталей, зберігає результати та генерує аналітичні звіти. Ця функціональність була реалізована за допомогою скрипта main_yolo.py (див. Додаток Г, Лістинг Г.1), який став основою для першого програмного модуля. У процесі розробки програма була вдосконалена для підвищення швидкості, точності та інформативності результатів, що робить її придатною для автоматизованих систем у виробничих умовах.

Розробка розпочалася з визначення структури програми та її основних функцій. Скрипт написано мовою Python із використанням бібліотеки Ultralytics YOLOv8 для роботи з моделлю, OpenCV для обробки зображень, а також модулів os, logging, matplotlib і concurrent.futures для роботи з файлами, логуванням, генерацією графіків і паралельного завантаження зображень.

Основне завдання програми – обробка всіх зображень із вхідного каталогу, детекція радіодеталей із п’яти класів (background, capacitor, diode, led, resistor),

нанесення міток із назвами та ймовірностями правильності, збереження анотованих зображень у вихідний каталог, створення лог-файлу та побудова графіків для аналізу результатів.

Програма розпочинає роботу з ініціалізації моделі YOLOv8x, завантажуючи її з файлу best.pt. Модель налаштовується на використання GPU, що забезпечує високу швидкість обробки. Для оптимізації задано поріг упевненості CONFIDENCE_THRESHOLD=0.5, що дозволяє відфільтрувати слабкі передбачення та уникнути дублювання міток на одному об’єкті. Логування активується через модуль logging із рівнем INFO, а лог-файл створюється у вихідному каталозі для фіксації часу початку обробки, кількості оброблених зображень і виявлених об’єктів.

Основний цикл обробки реалізовано через функцію process_batch, яка приймає список зображень і відповідні шляхи до файлів. Скрипт сканує каталог за допомогою os.listdir(), відбираючи файли з розширеннями .jpg, .png і .jpeg.

Для підвищення продуктивності використано батчову обробку з розміром батчу batch_size=16. Зображення групуються по 16 одиниць і передаються моделі через метод model.predict().

Для прискорення завантаження зображень додано паралельну обробку за допомогою concurrent.futures.ThreadPoolExecutor, що значно скоротило час підготовки даних.

Час обробки одного батчу становить приблизно 0.1 секунду, що дозволило обробити весь набір із 3250 зображень за 20 секунди на локальному обладнанні. У разі помилки завантаження зображення (наприклад, пошкоджений файл) скрипт записує попередження в лог і пропускає файл, забезпечуючи стабільність роботи, але подібних аномалій під час тестування не виявлено.

Для кожного зображення модель повертає список виявлених об’єктів із класами та ймовірностями (confidence scores). Ці дані обробляються функцією process_batch, яка використовує метод result.plot() для нанесення міток і прямокутників на зображення. Мітки для кожного класу мають свій колір прямокутника та тексту, що полегшує візуальну інтерпретацію.

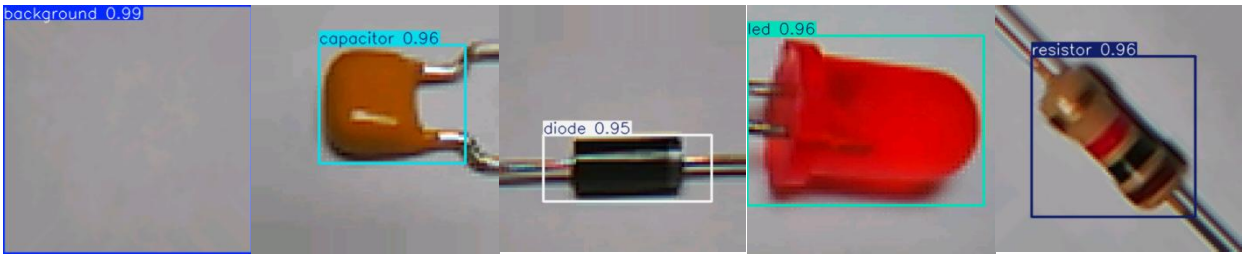


Рисунок 3.6 – Вигляд міток для кожного класу радіодеталей.

У лог-файл записується інформація про кожен об’єкт: назва файлу, клас, ймовірність і координати, що дозволяє проводити подальший аналіз чи пошук помилок.

```
2025-03-11 14:38:52,957 - Логування ініціалізовано в: C:\dyplom\3250all\output91\predictions.log
2025-03-11 14:38:52,957 - GPU доступний: NVIDIA GeForce RTX 4070 SUPER
2025-03-11 14:38:53,112 - Модель завантажено успішно. Класи: {0: 'background', 1: 'capacitor', 2: 'diode', 3: 'led', 4: 'resistor'}
2025-03-11 14:38:53,392 - Обробка батчу з 16 зображень...
2025-03-11 14:38:54,225 - Об’єкти виявлено для resized_1.png:
2025-03-11 14:38:54,225 - Об’єкт: background, Упевненість: 0.9940
2025-03-11 14:38:54,226 - Об’єкти виявлено для resized_10.png:
2025-03-11 14:38:54,227 - Об’єкт: background, Упевненість: 0.9941
2025-03-11 14:38:54,228 - Об’єкти виявлено для resized_100.png:
2025-03-11 14:38:54,228 - Об’єкт: background, Упевненість: 0.9945

2025-03-11 14:39:14,043 - Об’єкти виявлено для resized_999.png:
2025-03-11 14:39:14,043 - Об’єкт: capacitor, Упевненість: 0.9371
2025-03-11 14:39:14,045 - Результати обробки збережено в: C:\dyplom\3250all\output91
2025-03-11 14:39:14,045 -
Статистика об’єктів:
2025-03-11 14:39:14,045 - background: 650 раз(ів), Середня упевненість: 0.9940
2025-03-11 14:39:14,045 - capacitor: 650 раз(ів), Середня упевненість: 0.9350
2025-03-11 14:39:14,045 - diode: 650 раз(ів), Середня упевненість: 0.9265
2025-03-11 14:39:14,045 - led: 650 раз(ів), Середня упевненість: 0.9453
2025-03-11 14:39:14,046 - resistor: 650 раз(ів), Середня упевненість: 0.9356
2025-03-11 14:39:14,046 - Загальна середня упевненість: 0.9473
2025-03-11 14:39:14,046 - Час обробки при batch=16: 20.65 секунд
2025-03-11 14:39:14,046 - Кількість оброблених зображень: 3250
2025-03-11 14:39:14,356 - Графіки збережено в: C:\dyplom\3250all\output91
```

Рисунок 3.7 – Приклад вигляду лог-файлу

Додатковою функцією програми є генерація аналітичних звітів. Скрипт підраховує загальну кількість виявлених об’єктів для кожного класу і зберігає ці дані в словнику object_counts. Також фіксується сума ймовірностей (confidence_sums) для обчислення середньої точності кожного класу.

Після обробки всіх зображень програма будує три графіки за допомогою бібліотеки matplotlib:

- Стовпчикова діаграма кількості об’єктів по класах.
- Стовпчикова діаграма середньої ймовірності правильності для кожного класу із позначенням загальної середньої точності.

- Графік часу обробки зображень, що показує розподіл часу для кожного зображення та середній час обробки.

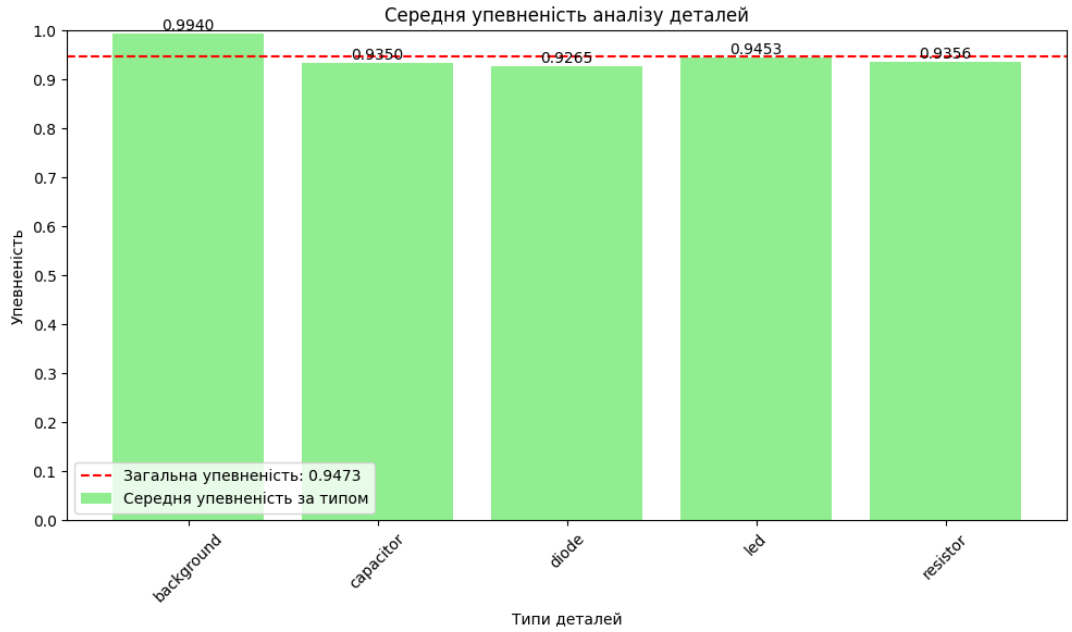


Рисунок 3.8 – Стовпчикова діаграма середньої упевненості аналізу деталей для кожного класу та середня ймовірність для усіх класів.

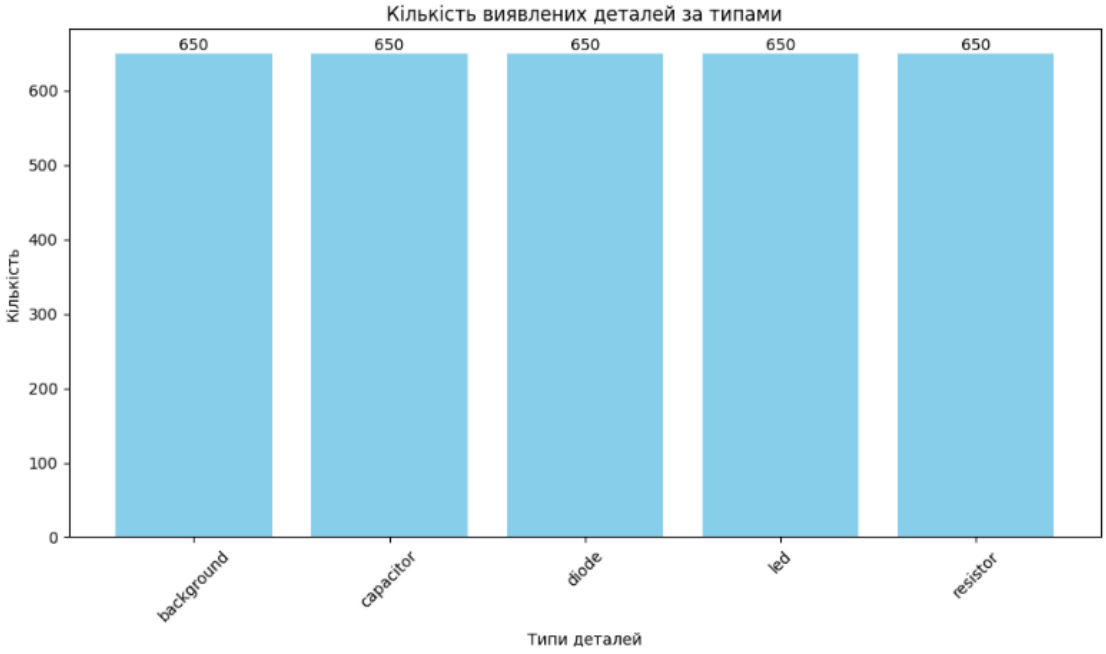


Рисунок 3.9 – Стовпчикова діаграма кількості деталей по класах.

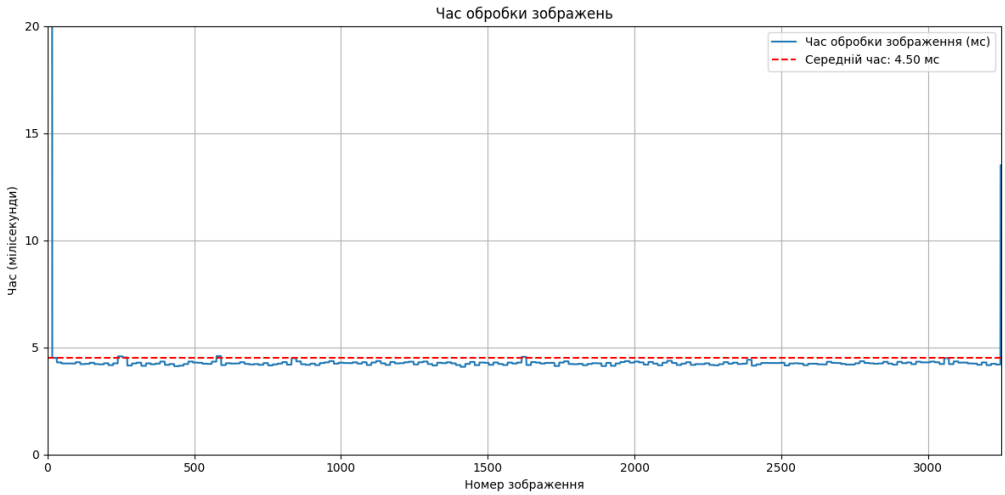


Рисунок 3.10 – Графік часу обробки зображень.

Для 3250 зображень результати показали 650 одиниць для кожного класу із середньою ймовірністю коректності аналізу 94.7%. Графіки зберігаються у форматі .png у вихідному каталозі разом із лог-файлом. Графік часу обробки дозволяє оцінити стабільність роботи програми: середній час обробки одного зображення склав 4.5 мс, що підтверджує високу ефективність оптимізації.

Тестування програми проводилося на повному наборі з 3250 зображень.

Результати показали високу стабільність:

- 94% об’єктів мають упевненість коректності класифікації (confidence score) понад 0.9.
- 6% об’єктів мають упевненість у діапазоні 0.80-0.89.
- Об’єктів нижче впевненості 0.80 не виявлено.

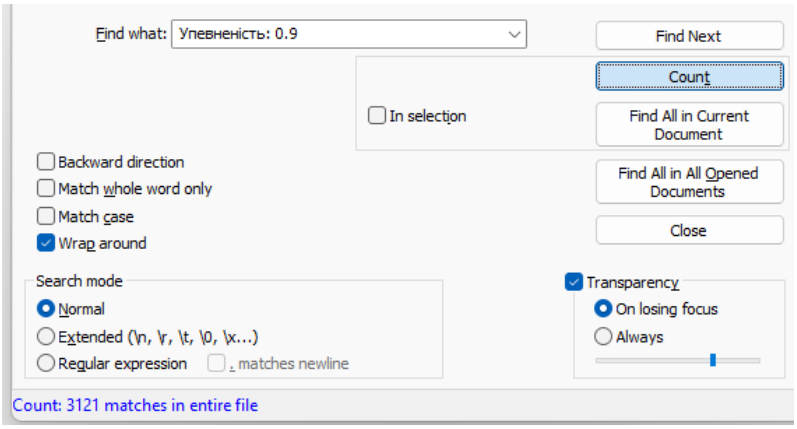


Рисунок 3.11 – 3121 зображень із 3250 мають рівень упевненості вище 0.9.

Час обробки для 3250 зображень склав 20 секунд і виконує вимоги до швидкості, поставлені в задачі, що підтверджує ефективність батчової обробки та використання GPU.

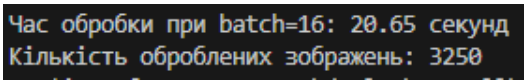


Рисунок 3.12 – Час обробки який підраховується у програмі.

Програма також передбачає можливість масштабування: зміна параметрів conf, IoU чи batch_size у коді дозволяє налаштувати її під інше апаратне забезпечення. Наприклад, зменшення розміру батчу подовжує час обробки зменшив навантаження на систему, але ніяк не впливає на якість результатів. Лог-файл і графіки виявилися зручними для аналізу: стовпчикова діаграма чітко показала розподіл класів, графік ймовірностей допоміг виявити слабкі місця, такі як нижча точність для діодів (0.92) через їхній малий розмір, а графік часу обробки підтвердив стабільність роботи програми.

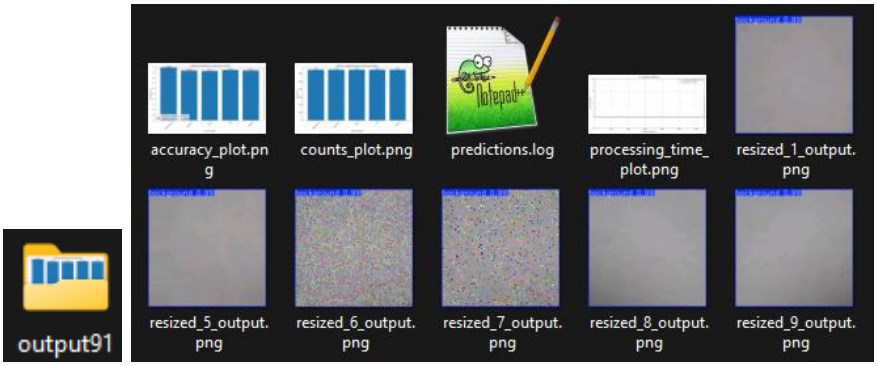


Рисунок 3.13 – Вигляд папки Output після виконання програми: 3 графіки, 1 лог-файл, 3250 зображень з мітками на них.

Програма забезпечує швидку, точну і зручну обробку зображень із папки. Вона генерує анотовані зображення, лог-файл і графіки, які можуть бути використані для контролю якості чи інтеграції з роботизованими системами.

3.3. Адаптація для аналізу в реальному часі з камери телефону

Для розширення можливостей застосування моделі до роботи в динамічних умовах було розроблено другий програмний модуль, який забезпечує аналіз відеопотоку в реальному часі з камери телефону. Цей модуль, реалізований у програмі main_phone.py (див. Додаток Г, Лістинг Г.2). Використовується додаток EpocCam для передачі зображень через Wi-Fi, дозволяючи комп'ютеру обробляти потік як із звичайної камери з індексом 0. Такий підхід імітує реальні сценарії в роботизованих системах, де потрібне безперервне розпізнавання радіодеталей із збереженням результатів і логуванням. Програма поєднує високу точність із продуктивністю, адаптуючи модель до потокового відео.

Розробка модуля почалася з налаштування зв'язку між телефоном і комп'ютером. Додаток EpocCam, установлений на смартфоні, перетворює його камеру на веб-камеру, яка передає відеопотік через Wi-Fi до комп'ютера. Завдяки драйверам EpocCam потік розпізнається як стандартний пристрій відеозахоплення з індексом 0. Телефон і комп'ютер підключалися до однієї Wi-Fi-мережі, а роздільна здатність потоку EpocCam становила 720p із частотою 30 FPS(мінімум).

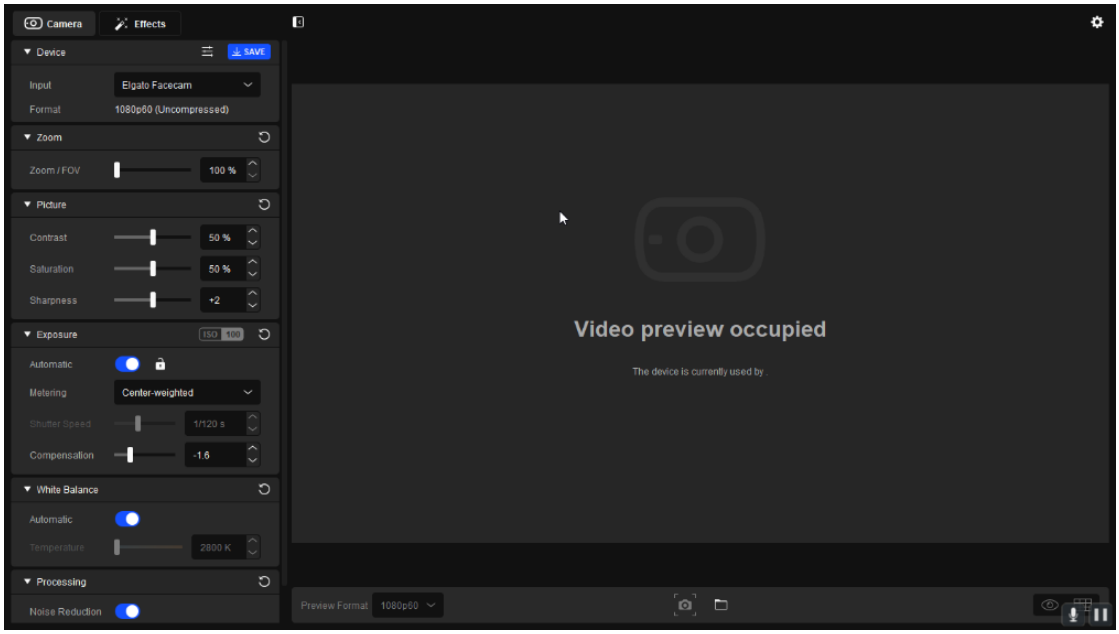


Рисунок 3.14 – Інтерфейс програми Camera Hub для з'єднання с телефоном.

Програму написано на Python із використанням бібліотек Ultralytics YOLOv8 для роботи з моделлю, OpenCV для обробки кадрів, а також os, numpy, logging і matplotlib для управління файлами, обчислень, логування та побудови графіків. Модель завантажується з файлу best.pt із порогом упевненості 0.8 (збільшено з 0.5 для зменшення хибнопозитивних детекцій у реальному часі). Програма створює вихідну папку для результатів за базовим шляхом, додаючи номер (output1/2.../n), якщо папка вже існує, що реалізовано через цикл із os.path.exists(). Логування налаштовано через logging.basicConfig із файлом predictions.log у вихідній папці, де фіксуються час, виявлені об’єкти та повідомлення.

Ключовою функцією є crop_to_320x320_clean, яка готує кадри для моделі. Початковий кадр обрізається до квадратної області, а потім із центру вирізається область 320x320 пікселів – розмір, на якому тренувалася модель. Далі кадр аналізується для видалення чорних рамок: перетворюється в градації сірого (cv2.cvtColor), застосовується поріг (cv2.threshold) для виділення ненульових пікселів, і за допомогою cv2.boundingRect визначається область із вмістом. Таким чином отримуємо потік 320x320, що імітує камеру робота.

Функція process_frame обробляє кожен кадр. Спочатку викликається crop_to_320x320_clean для підготовки зображення, потім модель виконує передбачення через results = model(cleaned_frame, conf=0.8, iou=0.5). Анотований кадр генерується методом results[0].plot(), який додає прямокутники та мітки. Виявлені об’єкти (results[0].boxes) аналізуються: для кожного об’єкта формується рядок label: score (наприклад, capacitor: 0.95), який виводиться в консоль і лог як підсумок. Кількість (object_counts) і сума ймовірностей (confidence_sums) оновлюються для статистики. Якщо об’єктів немає, це фіксується повідомленням. Кадр зберігається з унікальною назвою за допомогою cv2.imwrite, а шлях і анотований кадр повертаються.

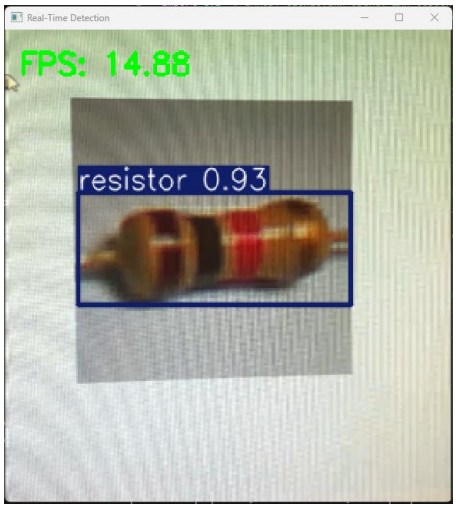


Рисунок 3.15 – Вигляд прогами аналізу радіодеталей у реальному часі.

Для стабільної роботи в реальному часі додано обмеження частоти кадрів (TARGET_FPS=15), що дозволяє уникнути перевантаження системи та забезпечує плавність обробки. Хоча локальна система дозволяє підтримувати стабільні 58-60 FPS під час аналізу у реальному часі, але це надлишково для аналізу радіодеталей через більш високе навантаження на систему та значного збільшення збережених кадрів для одного об’єкту. Ліміт у 15 FPS є оптимальним.

Час обробки кожного кадру фіксується, а FPS відображається на екрані за допомогою cv2.putText. Головний цикл захоплює кадри через cap.read(), перевіряє їхню наявність (ret) і передає в process_frame. Отриманий анотований кадр відображається у вікні Real-Time Detection (cv2.imshow), а цикл завершується натисканням q (cv2.waitKey(delay) == ord('q')).

Якщо камера недоступна (cap.isOpened() == False) або кадр не отримано, видаються повідомлення про помилку. Після завершення виводиться шлях до вихідної папки, якщо є збережені файли.

Також програма включає генерацію аналітичних графіків.

Аналогічно до програми пакетної обробки зображень будуються три графіки:

- Стовпчикову діаграму кількості виявлених об’єктів по класах.
- Стовпчикову діаграму середньої ймовірності правильності для кожного класу із позначенням загальної середньої точності.

- Графік часу обробки кадрів, що показує розподіл часу обробки та середній час на кадр.

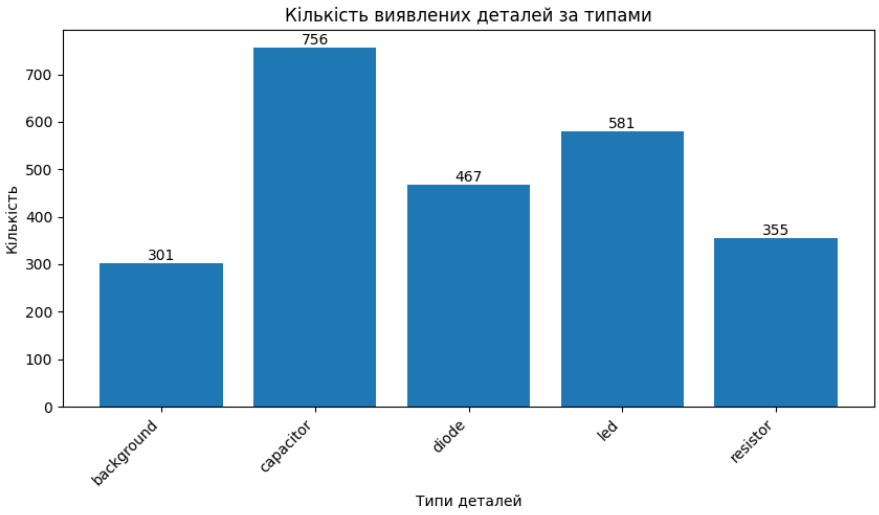


Рисунок 3.16 – Вигляд стовпчикової діаграми кількості виявлених радіодеталей.

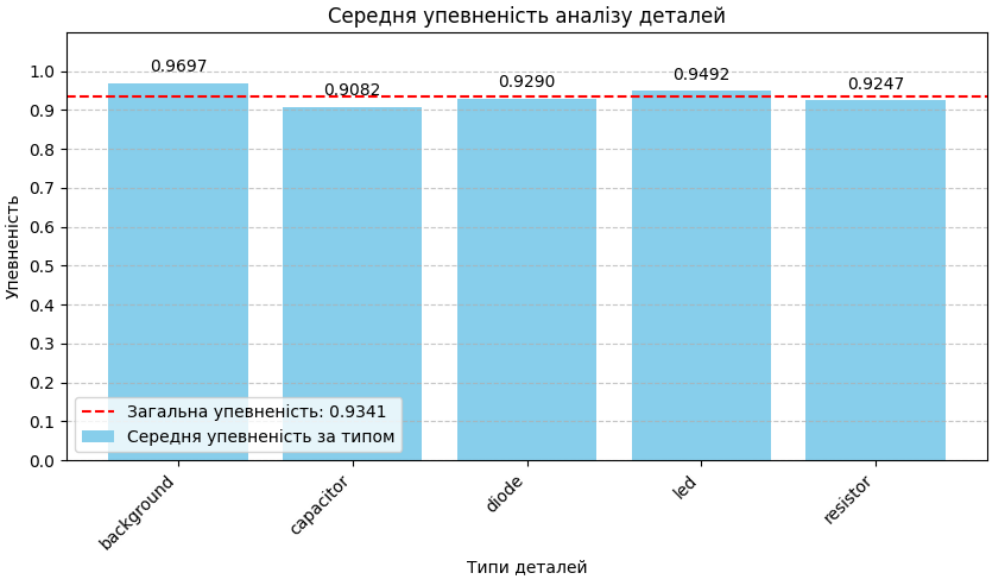


Рисунок 3.17 – Вигляд стовпчикової діаграми середньої упевненості аналізу деталей.

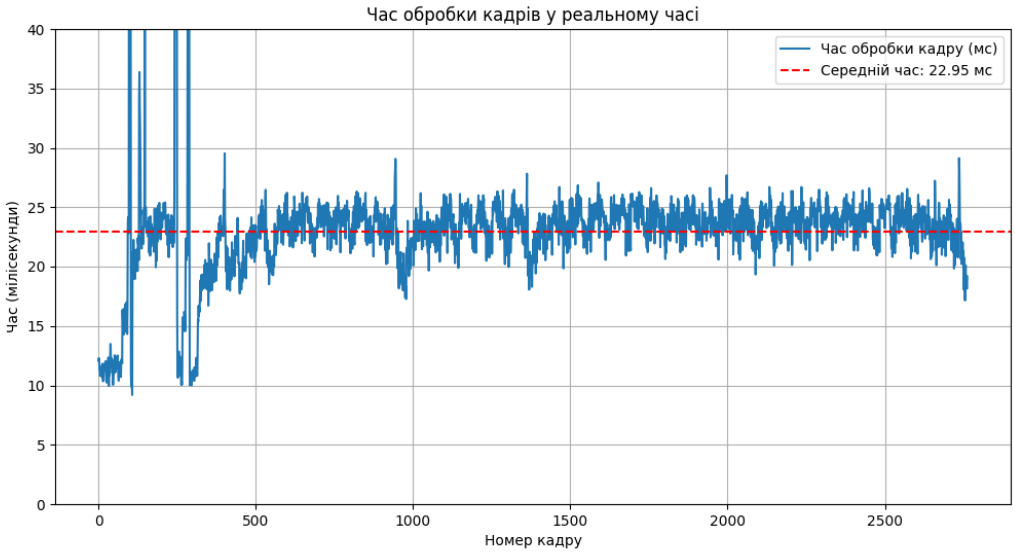


Рисунок 3.18 – Вигляд графіку часу обробки кадрів.

Тестування програми показало стабільну частоту обробки ~14.9 fps. Середня точність склала 93%, що нижче на ~1% у порівнянні з пакетню обробку (94%) через динамічні зміни та не фіксованість камери, яка знаходилась у руках.

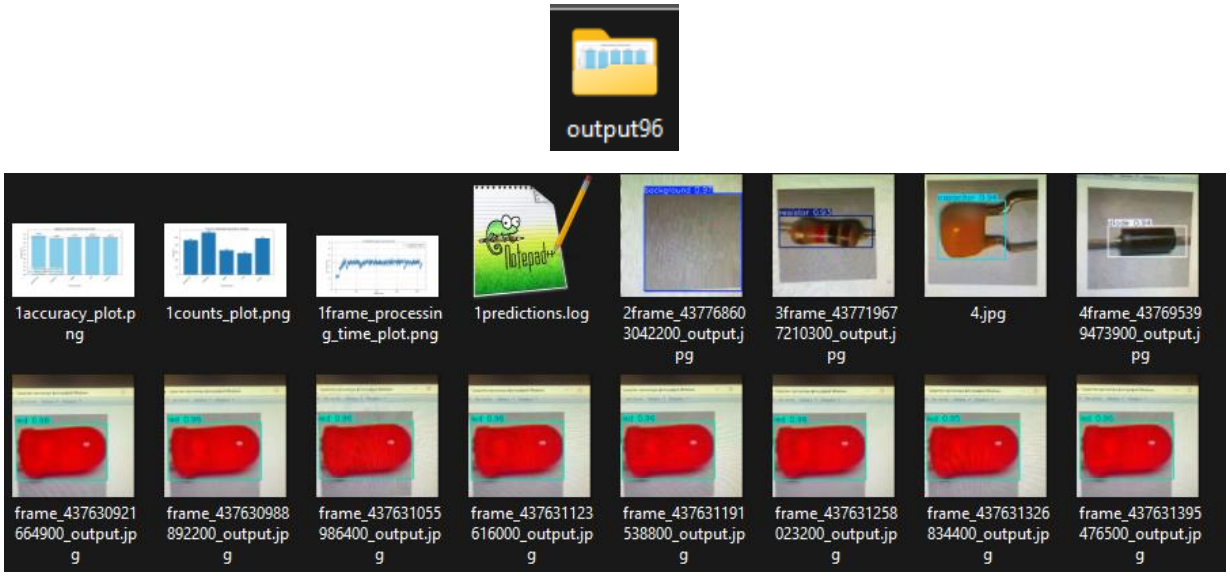


Рисунок 3.19 – Вигляд Output папки після виконання програми.

Модель було успішно адаптовано для реального часу, реалізуючи обрізку кадрів, детекцію, збереження та логування з частотою 15 fps і середньою точністю 93%.

3.4. Оцінка точності моделі та ефективності програм

Розроблена модель для розпізнавання радіодеталей та пов’язане з нею програмне забезпечення пройшли комплексне тестування, яке дозволило оцінити їхню ефективність за ключовими показниками: точність, швидкість обробки та стійкість до змін умов. У цьому підрозділі представлено детальний аналіз результатів, отриманих у ході експериментів, а також їхньої відповідності поставленим завданням.

Точність моделі. Тренування YOLOv8x на аугментованому наборі з 3250 зображень забезпечило високу якість класифікації, що підтверджується показником mAP50=0.995 та mAP50-95=0.96 на тестовому наборі з 650 зображень [15]. Це свідчить, що модель здатна точно визначати радіодеталі (background, capacitor, diode, led, resistor) із ймовірністю понад 99% при порозі IoU=0.5. Детальний аналіз точності для кожного класу наведено в таблиці нижче:

Таблиця 4.1.

Точність моделі YOLOv8x для кожного класу

Class	mAP50	mAP50-95
Background	0.995	0.995
Capacitor	0.995	0.949
Diode	0.995	0.931
Led	0.995	0.967
Resistor	0.995	0.96
All	0.995	0.96

Результати показали, що середня точність варіюється залежно від складності об’єктів: діоди та конденсатори мають трохи нижчі значення (93.1% та 94.9% відповідно), що пов’язано з їхньою формою та меншою розбірливістю у порівнянні з іншими класами. Світлодіоди, резистори та фон досягли точності понад 95%.

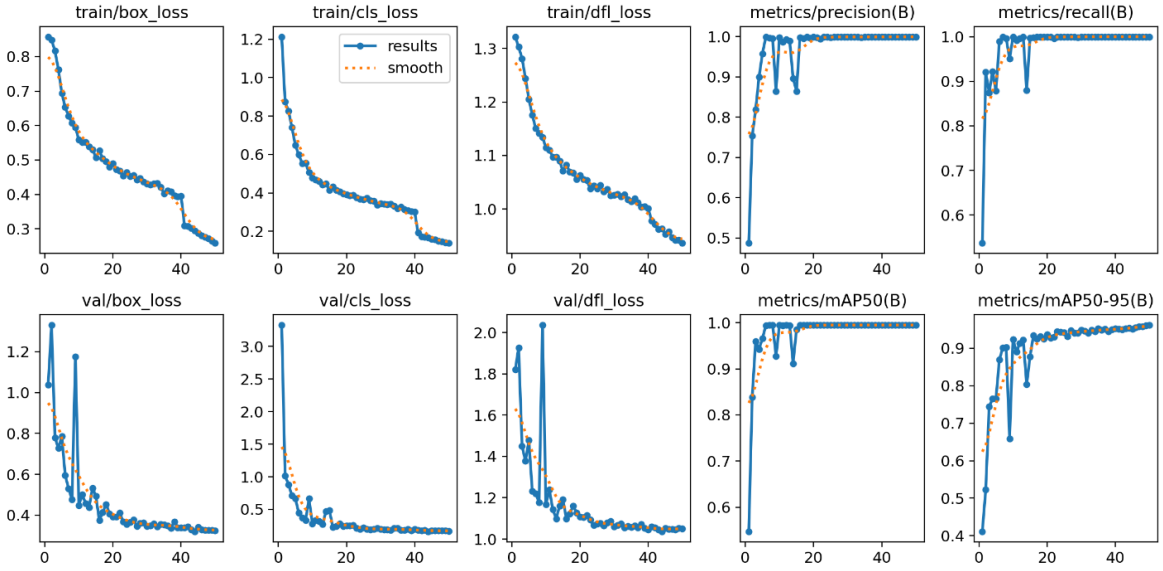


Рисунок 3.20 – Набір графіків метрики, тренування та валідації про навчання моделі.

Усі три графіки типу train показують, що втрати зменшуються з часом тренування. Це свідчить про те, що модель навчається і її передбачення стають точнішими.

Втрати на валідаційному(val) наборі також зменшуються, але є певна нестабільність (val/box_loss має стрибки на початку). Це свідчить про неоднорідність даних, що в нашому випадку правда через присутню аугментовацію. На середині навчання(після 15 епохи) та далі стрибків не виявлено.

Значення на графіках метрик(metrics) близько до 1 свідчить про високу точність аналізу.

Швидкість обробки. Для пакетної обробки 3250 зображень час виконання склав 20.65 секунд, що відповідає середній швидкості 6.35 мс на зображення. Це покращення досягнуто завдяки паралельному завантаженню зображень за допомогою concurrent.futures та оптимізації батчової обробки (batch_size=16).

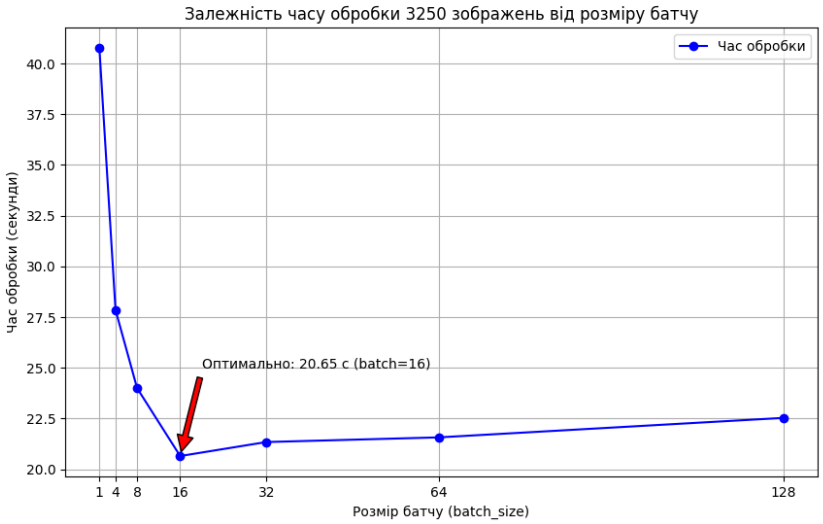


Рисунок 3.21 – Графік залежності розміру батчу на швидкість часу пакетної обробки.

Розмір батчу не є універсальним показником ефективності, і значення, відмінне від оптимального, може призводити до зниження продуктивності. Наприклад, при малому розмірі батчу (batch=1) час обробки зростає до 40.77 секунд через часте перемикання між окремими зображеннями та недостатнє використання паралельних обчислень GPU. Натомість при великих розмірах батчу (batch=128) час обробки збільшується до 22.53 секунд через обмеження оперативної пам’яті та потенційне перевантаження системи, що знижує ефективність обробки. Оптимальний розмір батчу 16 для локальної системи досягнуто завдяки балансу між паралельною обробкою зображень і доступними обчислювальними ресурсами, що забезпечило мінімальний час 20.65 секунд. Однак для інших систем із різною конфігурацією апаратного забезпечення оптимальне значення batch_size може відрізнятися, вимагаючи індивідуального тестування для максимізації продуктивності.

Модуль аналізу у реальному часу здатен працювати на стабільних 58-60 FPS на локальній системі, що є надмірним, через це було програмно обмежено частоту до 15 FPS (66 мс на кадр). Задача досягти 10-15 FPS у реальному часу досягнута. Показники підтверджують, що модель оптимізована для роботи як із статичними даними, так і з потоковим відео.

Рисунок 3.22 – Швидкість обробки кадрів у реальному часі.

На початку графіку можна бачити високу швидкість обробки кадру приблизно 13 мс через відсутність об’єкта для аналізу(Час наводження камери на радіодеталь). Потім почався неперервний аналіз радіодеталей з постійною зміною об’єкта, при таких умовах швидкість у діапазоні 18-28 мс, що задовольняє потребу до швидкості та стабільності. Стійкість моделі до змін умов зйомки була перевірена на аугментованих даних із шумом, поворотами та розмиттям.

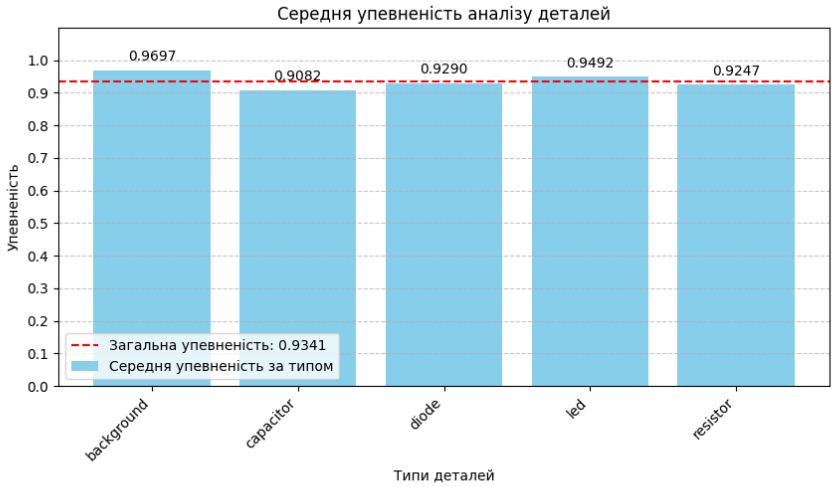


Рисунок 3.23 – Стовпчикова діаграма упевненості аналізу 2100 кадрів у реальному часі.

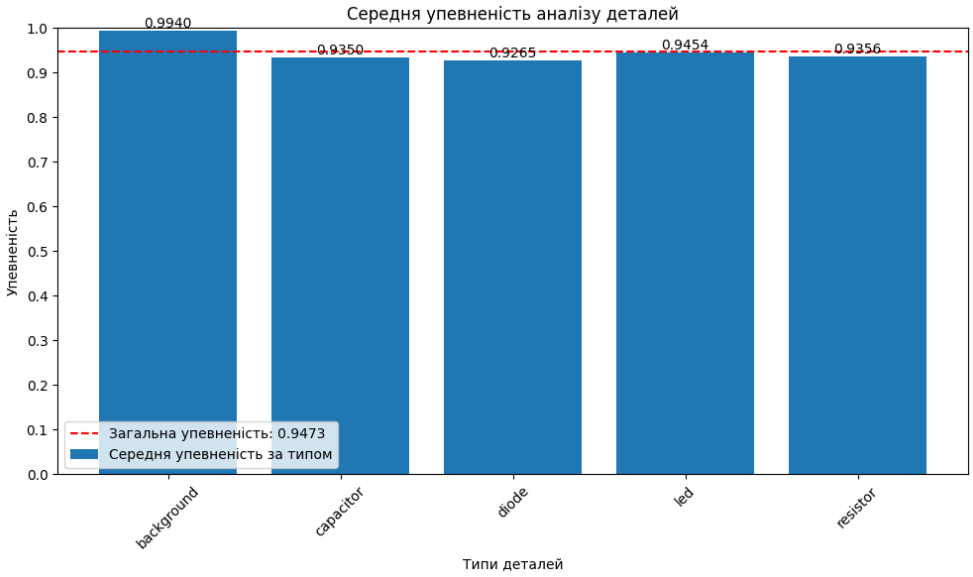


Рисунок 3.24 – Стовпчикова діаграма упевненості аналізу 3250 зображень при пакетній обробці.

Різниця точності між пакетною обробку та реальним часом: 0.9473-0.9341=0.132, що 1,3%. Модель реального часу продемонструвала стабільну роботу, зберігаючи середню точність 93-94% у реальному часі, що свідчить про її адаптивність до складних сценаріїв. Результати відповідають поставленим вимогам (середня точність не нижче 90%, 10-15 fps), підтверджуючи ефективність розробленого рішення для автоматизованого розпізнавання радіодеталей.

Аналіз ефективності показав, що розроблена модель YOLOv8x повністю відповідає поставленим цілям. Висока точність (94% у середньому), оптимальна швидкість (20 секунд для 3250 зображень і 15 fps у реальному часі) та стійкість до змін умов.

Висновки за розділом 3.

У третьому розділі кваліфікаційної роботи було проведено дослідження та реалізовано модель штучного зору та програмне забезпечення для розпізнавання радіодеталей на базі YOLOv8. Після попереднього тестування, яке виявило суттєві обмеження платформи Edge Impulse для ефективної обробки великих

аугментованих наборів даних, було прийнято рішення перейти до локального навчання з використанням бібліотеки Ultralytics YOLOv8.

Для вирішення задачі обрано модель YOLOv8x, яку навчено на підготовленому аугментованому наборі даних з 3250 зображень. Результати тренування показали високі метрики: mAP50-95 склав 94%, а mAP50 99.5%. Хоча точність визначення границь для діодів (93.1%) та конденсаторів (94.9%) була трохи нижчою, інші класи досягли точності понад 95%.

На основі навченої моделі розроблено два програмні модулі: для пакетної обробки статичних зображень та аналізу відеопотоку в реальному часі. Модуль пакетної обробки обробив 3250 зображень за 20.65 секунд, забезпечивши середню швидкість 6.35 мс/зображення, що перевищує вимогу не більше 9 мс. Середня впевненість аналізу склала 94.7%. Модуль аналізу в реальному часі показав стабільну частоту ~14.9 FPS, відповідаючи вимозі 10-15 FPS, із середнім часом обробки кадру 18-28 мс. Середня точність у цьому режимі становила 93%.

Таким чином, у третьому розділі було успішно розроблено та реалізовано систему розпізнавання радіодеталей, яка демонструє високу точність (в середньому 94%), оптимальну швидкість обробки та стійкість до змін умов зйомки.

ВИСНОВКИ

У результаті виконаної кваліфікаційної роботи було успішно реалізовано поставлену мету – створення моделі штучного зору для автоматичного розпізнавання радіодеталей та розробку програмного забезпечення для її використання в режимах пакетної обробки й аналізу в реальному часі. Проведене дослідження дозволило розв'язати всі визначені задачі, що забезпечило досягнення високої ефективності системи та її відповідність потребам промислової автоматизації.

Перша задача – аналіз сучасних методів комп'ютерного зору показав, що одноетапні детектори, зокрема YOLOv8x, є оптимальними для швидкого й точного розпізнавання об'єктів завдяки їхній здатності обробляти зображення за один прохід мережі.

Друга задача – підготовка набору даних була виконана шляхом використанням аугментованого набору з 3250 зображень, який враховує різноманітні умови зйомки (шум, повороти, розмиття), що підвищило адаптивність моделі до реальних сценаріїв.

Третя задача – розробка та навчання моделі завершилася тренуванням YOLOv8x із середньою точністю 94% (mAP50=0.995, mAP50-95=0.96) на тестовому наборі, що перевищує встановлений поріг у 90%.

Четверта задача – створення програмних модулів реалізувалася через розробку двох програм: для пакетної обробки (20.65 секунд на 3250 зображень, середня швидкість 6.35 мс на зображення) та аналізу в реальному часі (15 кадрів за секунду з точністю 93-94%). Обидва модулі, написані на Python із використанням бібліотек Ultralytics YOLOv8 і OpenCV, забезпечують стабільну роботу, генерацію анотованих зображень, лог-файлів і аналітичних графіків.

Остання задача – оцінка точності й ефективності підтвердила, що система відповідає вимогам: середня точність класифікації становить 94% у пакетному режимі та 93-94% у реальному часі, швидкість обробки 20 секунд для партії із 3250 зображень і 15 FPS для відеопотоку, а стійкість до змін умов зйомки забезпечена аугментацією даних.

Практична цінність роботи полягає у створенні системи, яка може бути інтегрована в роботизовані комплекси для автоматизації сортування, складання та контролю якості радіодеталей у виробництві електроніки. Досягнута швидкість і точність перевищують типові промислові стандарти, а гнучкість модулів дозволяє адаптувати їх до різних апаратних платформ. Розробка також має теоретичне значення, розширюючи знання про застосування одноетапних детекторів у задачах із малими об’єктами та складними умовами.

Усі поставлені задачі виконано, а мета роботи досягнута. Розроблена модель і програмне забезпечення демонструють високу продуктивність і відкривають перспективи для подальшого вдосконалення наприклад, розширення набору класів і створення графічного інтерфейсу для зручності використання в промислових умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ahmad M. et al. Convolutional neural network–based person tracking using overhead views. International Journal of Distributed Sensor Networks. 2020. Vol. 16, no. 6. P. 155014772093473. URL: https://doi.org/10.1177/1550147720934738 (дата звернення: 06.06.2025).
2. Chilamkurthy S. PyTorch: Transfer Learning for Computer Vision Tutorial. 2024. URL: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html (дата звернення: 05.06.2025).
3. Edge Impulse. Документація Edge Impulse. 2023. URL: https://docs.edgeimpulse.com/docs (дата звернення: 05.06.2025).
4. Goodfellow I., Bengio Y., Courville A. Deep Learning Book: Конволюційні мережі. 2016. URL: https://www.deeplearningbook.org/contents/convnets.html (дата звернення: 05.06.2025).
5. He K., Gkioxari G., Dollár P., Girshick R. Mask R-CNN. Proceedings of the IEEE International Conference on Computer Vision (ICCV). 2017. P. 2961–2969. URL: https://openaccess.thecvf.com/content_ICCV_2017/papers/He_Mask_R-CNN_ICCV_2017_paper.pdf (дата звернення: 05.06.2025).
6. Hussain M. Sustainable Machine Vision for Industry 4.0: A Comprehensive Review of Convolutional Neural Networks and Hardware Accelerators in Computer Vision. AI. 2024. Vol. 5, no. 3. P. 1324–1356. URL: https://doi.org/10.3390/ai5030064 (дата звернення: 06.06.2025).
7. Hymel S. Комп’ютерне бачення з вбудованим машинним навчанням. Coursera. 2020. URL: https://www.coursera.org/learn/computer-vision-with-embedded-machine-learning (дата звернення: 05.06.2025).
8. Kustija J. et al. Object detection in printed circuit board quality control: comparing algorithms faster region-based convolutional neural networks and YOLOv8. International Journal of Electrical and Computer Engineering

(IJECE). 2025. Vol. 15, no. 3. P. 2796. URL: https://doi.org/10.11591/ijece.v15i3.pp2796-2808 (дата звернення: 06.06.2025).

9. Liu W., Anguelov D., Erhan D., Szegedy C., Reed S., Fu C.-Y., Berg A. C. SSD: Single Shot MultiBox Detector. European Conference on Computer Vision (ECCV). 2016. P. 21–37. URL: https://link.springer.com/chapter/10.1007/978-3-319-46448-0_2 (дата звернення: 05.06.2025).
10. Quinn E., Corcoran N. Automation of Computer Vision Applications for Real-time Combat Sports Video Analysis. International Conference on AI Research. 2022. Vol. 4, no. 1. P. 162–171. URL: https://doi.org/10.34190/icair.4.1.930 (дата звернення: 06.06.2025).
11. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. P. 779–788. URL: https://arxiv.org/pdf/1506.02640 (дата звернення: 05.06.2025).
12. Şahin D. et al. Real-Time Classification of Chicken Parts in the Packaging Process Using Object Detection Models Based on Deep Learning. Processes. 2025. Vol. 13, no. 4. P. 1005. URL: https://doi.org/10.3390/pr13041005 (дата звернення: 06.06.2025).
13. Sandler M., Howard A., Zhu M., Zhmoginov A., Chen L.-C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv:1801.04381. 2018. URL: https://arxiv.org/pdf/1801.04381 (дата звернення: 05.06.2025).
14. Simonyan K., Zisserman A. VGGNet: Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556. 2014. URL: https://arxiv.org/pdf/1409.1556 (дата звернення: 05.06.2025).
15. Solawetz J. Mean Average Precision (mAP) for Object Detection. Roboflow. 2023. URL: https://blog.roboflow.com/mean-average-precision/ (дата звернення: 05.06.2025).
16. Wang C.-Y., Bochkovskiy A., Liao H.-Y. M. YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors.

arXiv:2207.02696. 2023. URL: https://arxiv.org/pdf/2207.02696 (дата звернення: 05.06.2025).

17. Шматко О. В. et al. Інформаційна система розпізнавання зображень. Системи озброєння і військова техніка. 2021. No. 4 (68). P. 130–137. URL: https://doi.org/10.30748/soivt.2021.68.17 (дата звернення: 06.06.2025).

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'ютерна інженерія

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

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

БОЖКА Тараса Вячеславовича
(прізвище, ім'я, по батькові студента)

1. Тема роботи **«МОДЕЛЬ ШТУЧНОГО ЗОРУ РОБОТА ДЛЯ РОЗПІЗНАВАННЯ РАДІОДЕТАЛЕЙ»**

керівник роботи **Бикова Тетяна Володимирівна, к.т.н., доцент кафедри комп'ютерних систем та робототехніки.**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

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

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

3. Перелік питань, які потрібно розробити
1. Аналіз сучасних методів комп'ютерного зору для розпізнавання об'єктів.
2. Вивчення особливостей розпізнавання радіодеталей на зображеннях.
3. Підготовка датасету для тренування моделі.
4. Розробка та тренування моделі на основі бібліотеки YOLOv8.
5. Створення програмного модуля для пакетної обробки зображень.
6. Розробка програмного модуля для аналізу в реальному часі з камери.
7. Оцінка точності та ефективності моделі та програмних модулів.
8. Аналіз можливостей інтеграції системи з роботизованими пристроями.
9. Дослідження перспектив удосконалення моделі та програмного забезпечення.
10. Оформлення результатів роботи.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Затвердження теми роботи з керівником	01.10.2024 - 31.10.2024
2	Аналіз та пошук методичної літератури щодо методів комп'ютерного зору	01.11.2024 - 30.11.2024
3	Вивчення особливостей розпізнавання радіодеталей та підготовка набіру даних із зображень	01.12.2024 - 15.12.2024
4	Розробка та тренування моделі YOLOv8	16.12.2024 - 31.12.2024
5	Створення програмного модуля для пакетної обробки зображень	01.01.2025 - 15.01.2025
6	Розробка програмного модуля для аналізу в реальному часі	16.01.2025 - 31.01.2025
7	Тестування та оцінка ефективності моделі та програмних модулів	01.02.2025 - 15.02.2025
8	Дослідження інтеграції з роботизованими пристроями	16.02.2025 - 28.02.2025
9	Аналіз перспектив удосконалення моделі та програмного забезпечення	01.03.2025 - 15.03.2025
10	Підготовка та оформлення звітних матеріалів	16.03.2025 - 31.03.2025
11	Оформлення пояснювальної записки та додатків	01.04.2025 - 15.04.2025
12	Представлення кваліфікаційної роботи керівнику та рецензенту	15.04.2025- 31.05.2025

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

Студент **Божко Т.В**. _____
ініціали, прізвище підпис

Керівник роботи **Бикова Т.В.** _____
ініціали, прізвище підпис

Додаток Б

Затверджую

_____

«_____»_____2025 р.

Технічне завдання
на розробку системи
«Система штучного зору робота для розпізнавання радіодеталей».

1.	Введення	1.1 Назва роботи – Система штучного зору робота для розпізнавання радіодеталей. 1.2. Галузь застосування: Інформаційні технології
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 123 – Комп'ютерна інженерія 2.2. Завдання на кваліфікаційну роботу бакалавра №4101-5/962 від «16» квітня 2025 року (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи).
3.	Призначення розробки	3.1. Мета розробки: створення моделі штучного зору для автоматичного розпізнавання радіодеталей та розробка програмного забезпечення для її практичного застосування в режимах пакетної обробки та аналізу в реальному часі. 3.2. Призначення розробки: для використання в автоматизації процесів сортування, складання та контролю якості в електроніці та робототехніці. 3.3. Вхідні дані: зображення радіодеталей, вони можуть надходити із статичних камер (аналіз з папки) або з динамічних джерел (відеопотік з камери). 3.4. Вихідні дані: Анотовані зображення/кадри з позначеними виявленими радіодеталями, вказанням назви класу та ймовірності правильності (впевненості). Лог-файли, що містять інформацію про виявлені об'єкти, статистику та час виконання. Аналітичні графіки, що візуалізують статистику, зокрема: кількість об'єктів за класами, середня впевненість для кожного класу та час обробки кадрів/зображень.

4.	Технічні вимоги до програмного виробу	4.1. Функціональні вимоги: - Система має розпізнавати та класифікувати об'єкти за п’ятьма класами: конденсатори, діоди, світлодіоди, резистори та фон. - Надавати можливість обробки набору статичних зображень, що зберігаються в окремій папці. - Забезпечувати обробку зображень, що надходять з відеопотоку, наприклад, з камери телефону через мережу Wi-Fi. - Створювати вихідні зображення або кадри відеопотоку з нанесеними мітками (bounding boxes) для виявлених об’єктів, із зазначенням назви класу та ймовірності правильності (впевненості). - Фіксувати інформацію про виявлені об’єкти, статистику аналізу та час виконання програми у лог-файлі. - Автоматично генерувати графіки, що візуалізують результати аналізу, зокрема: стовпчикову діаграму кількості об’єктів за класами, стовпчикову діаграму середньої ймовірності правильності для кожного класу та графік часу обробки кадрів/зображень. 4.2. Нефункціональні вимоги: - Система повинна бути стійкою до змін умов зйомки, таких як шум, артефакти, різне освітлення, кути огляду (повороти), масштабування, частковий показ об'єктів або розмиття, що імітує реальні виробничі сценарії. - Система повинна точною, точність розпізнавання п’яти класів повинна бути не менше 95%. Середня впевненість розпізнавання повинна бути не нижче 0.90. Бажано підтримувати показник упевненості не нижче 80% для кожного об'єкта. - Система повинна працювати швидко, час обробки одного зображення в пакетному режимі не більше 9 мс. Аналіз у реальному часі повинен забезпечувати частоту обробки 10-15 FPS. Час обробки кадру в реальному часі не більше 66 мс

		- Система повинна ефективною, використовуючи графічний процесор (GPU) для оптимізації обчислень та забезпечення високої швидкості обробки.. - Система повинна бути простою у розгортанні на стандартному персональному комп'ютері без потреби у додатковому спеціалізованому обладнанні. - Система повинна бути масштабованою, повинна існувати можливість адаптації до інших типів об'єктів шляхом донавчання моделі, а також можливість адаптації до менш продуктивних апаратних платформ шляхом використання легших версій моделі. 4.3. Вимоги до інтеграції: - Система повинна бути інтегрована з джерелом відеопотоку для забезпечення аналізу в реальному часі. Як приклад такого джерела, в процесі розробки та тестування використовувався телефон на IOS з додатком EpocCam, який передавав зображення через Wi-Fi мережу на комп'ютер для обробки. - Система повинна бути інтегрована з папкою з зображеннями для виконання пакетної обробки. 4.4. Вимоги до безпеки: Система передбачає регулярне оновлення бібліотек для усунення вразливостей.
5.	Вимоги до програмної документації	Документацією до виробу «Система штучного зору робота для розпізнавання радіодеталей» вважати: 1) Опис основних вимог та функціональності системи (представити у вигляді Додатку Б пояснювальної записки до кваліфікаційної роботи). 2) Програму і методику випробувань розробленої програми (представити як додаток В до пояснювальної записки до кваліфікаційної роботи). 3) Опис розробленої моделі (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи).

6.	Вимоги до техніко-економічних показників	Документацією до виробу «Система штучного зору робота для розпізнавання радіодеталей» вважати: 1) Справжнє Технічне завдання на розробку системи (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Опис розробленої системи (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи). 3) Джерела базової інформації.	
7.	Стадії і етапи розробки	Дата	Назва етапу
		01.11.2024 – 30.11.2024	Аналіз та пошук методичної літератури щодо методів комп'ютерного зору
		01.12.2024 – 15.12.2024	Вивчення особливостей розпізнавання радіодеталей та підготовка набіру даних із зображень
		16.12.2024 – 31.12.2024	Розробка та тренування моделі YOLOv8
		01.01.2025 – 15.01.2025	Створення програмного модуля для пакетної обробки зображень
		16.01.2025 – 31.01.2025	Розробка програмного модуля для аналізу в реальному часі
		01.02.2025 – 15.02.2025	Тестування та оцінка ефективності моделі та програмних модулів
		16.02.2025 – 28.02.2025	Дослідження інтеграції з роботизованими пристроями
		01.03.2025 – 15.03.2025	Аналіз перспектив удосконалення моделі та програмного забезпечення
		16.03.2025 – 31.03.2025	Підготовка та оформлення звітних матеріалів
		01.04.2025 – 15.04.2025	Оформлення пояснювальної записки та додатків

		15.04.2025 – 31.05.2025	Представлення кваліфікаційної роботи керівнику та рецензенту
8.	Порядок контролю і приймання програмного продукту (моделі)	1) Перевірку ходу розробки програмного виробу керівнику робіт виконувати раз в 3 тижні. 2) Випробування програмного продукту провести відповідно до програми та методики випробувань на базі комп'ютерного класу. 3) Захист розробленої моделі провести на засіданні Атестаційної комісії. 4) Пояснювальну записку подати в електронному вигляді в 1 примірнику.	

Виконавець

Студент групи КІ-41

Божко Т. В.

Замовник

к. техн. наук, доц.

Бикова Т. В.

Додаток В

Програма і методика випробувань програмного виробу

«Система штучного зору робота для розпізнавання радіодеталей».

1. **Об’єкт випробувань**

1.1. Назва: Система штучного зору робота для розпізнавання радіодеталей.

1.2. Область застосування: Інформаційні технології.

1.3. Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. **Мета випробувань**

Перевірка відповідності функціональні можливості системи заявленим функціональним можливостям в технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. **Загальні положення**

3.1. Підстави для випробувань: Наказ про призначення атестаційної комісії.

3.2. Приймальні (приймально-здавальні) випробування проводяться на базі комп'ютерного класу кафедри в період роботи атестаційної комісії.

3.3. Обсяг випробувань: Відповідність програмі та методиці випробувань.

3.4. Організації:

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

4. **Вимоги до програмного забезпечення**

4.1. Функціональні вимоги:

- Система має розпізнавати та класифікувати об'єкти за п’ятьма класами: конденсатори, діоди, світлодіоди, резистори та фон.
- Надавати можливість обробки набору статичних зображень, що зберігаються в окремій папці.
- Забезпечувати обробку зображень, що надходять з відеопотоку, наприклад, з камери телефону через мережу Wi-Fi.
- Створювати вихідні зображення або кадри відеопотоку з нанесеними мітками (bounding boxes) для виявлених об’єктів, із зазначенням назви класу та ймовірності правильності (впевненості).
- Фіксувати інформацію про виявлені об’єкти, статистику аналізу та час виконання програми у лог-файлі.
- Автоматично генерувати графіки, що візуалізують результати аналізу, зокрема: стовпчикову діаграму кількості об’єктів за класами, стовпчикову діаграму середньої ймовірності правильності для кожного класу та графік часу обробки кадрів/зображень.

4.2. Нефункціональні вимоги:

- Система повинна бути стійкою до змін умов зйомки, таких як шум, артефакти, різне освітлення, кути огляду (повороти), масштабування, частковий показ об'єктів або розмиття, що імітує реальні виробничі сценарії.
- Система повинна точною, точність розпізнавання п’яти класів повинна бути не менше 95%. Середня впевненість розпізнавання повинна бути не нижче 0.90. Бажано підтримувати показник упевненості не нижче 80% для кожного об'єкта.
- Система повинна працювати швидко, час обробки одного зображення в пакетному режимі не більше 9 мс. Аналіз у реальному часі повинен забезпечувати частоту обробки 10-15 FPS. Час обробки кадру в реальному часі не більше 66 мс
- Система повинна ефективною, використовуючи графічний процесор (GPU) для оптимізації обчислень та забезпечення високої швидкості обробки..
- Система повинна бути простою у розгортанні на стандартному персональному комп’ютері без потреби у додатковому спеціалізованому обладнанні.
- Система повинна бути масштабованою, повинна існувати можливість адаптації до інших типів об’єктів шляхом донавчання моделі, а також можливість адаптації до менш продуктивних апаратних платформ шляхом використання легших версій моделі.

4.3. Вимоги до інтеграції:

- Система повинна бути інтегрована з джерелом відеопотоку для забезпечення аналізу в реальному часі. Як приклад такого джерела, в процесі розробки та тестування використовувався телефон на IOS з додатком EpocCam, який передавав зображення через Wi-Fi мережу на комп'ютер для обробки.
- Система повинна бути інтегрована з папкою з зображеннями для виконання пакетної обробки.

4.4. Вимоги до безпеки:

- Система передбачає регулярне оновлення бібліотек для усунення вразливостей.

5. Вимоги до програмної документації

Документацією до виробу «Система штучного зору робота для розпізнавання радіодеталей» вважати:

1) Опис основних вимог та функціональності системи (представити у вигляді Додатку Б пояснювальної записки до кваліфікаційної роботи).
2) Програму і методику випробувань розробленої програми (представити як додаток В до пояснювальної записки до кваліфікаційної роботи).

3) Опис розробленої моделі (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи).

6. Засоби та порядок випробувань

6.1 Засоби випробувань:

ПК: CPU R7 9800x3d, GPU RTX 4070, 32 ГБ RAM, SSD, Windows 11.

Телефон: на IOS, камера 720p EpocCam.

Програмне забезпечення: Python 3.8+, OpenCV, Ultralytics YOLOv8, PyTorch, Matplotlib.

Набір даних: 3250 зображень (320x320 пікселів).

6.2Порядок випробувань:

Випробування проводяться у два етапи:

- Ознайомлювальний етап: Перевірка документації (ТЗ, ПМ, опис моделі) та демонстрація роботи системи (пакетна обробка, реальний час).
- Основний етап: Виконання тестів для перевірки відповідності вимогам ТЗ.

Перевірка функціональності:

- Система вірно розпізнає та класифікує об'єкти за п’ятьма визначеними класами.
- Система надає можливість обробки набору статичних зображень, що зберігаються у вказаній папці.
- Система забезпечує обробку зображень, що надходять з відеопотоку, наприклад, з камери телефону через мережу Wi-Fi.
- Система створює вихідні зображення або кадри відеопотоку з нанесеними мітками (bounding boxes) для виявлених об’єктів, із зазначенням назви класу та ймовірності правильності (впевненості).
- Система фіксує інформацію про виявлені об’єкти, статистику аналізу та час виконання програми у лог-файлі для подальшого аналізу.
- Система автоматично генерує графіки, що візуалізують результати аналізу, зокрема: стовпчикову діаграму кількості об’єктів за класами, стовпчикову діаграму середньої ймовірності правильності для кожного класу та графік часу обробки кадрів/зображень.

Перевірка інтеграції:

- Система може успішно інтегруватися з джерелом відеопотоку для забезпечення аналізу в реальному часі, наприклад, з камерою телефону через мережу Wi-Fi.
- Система може успішно інтегруватися з папкою, що містить зображення, для виконання пакетної обробки.
- Для аналізу в реальному часі додатково потрібне джерело відеопотоку (телефон, камера або інший пристрій) з можливістю підключення по Wi- Fi або проводу для передачі даних на комп'ютер.

Тест 1. Перевірка точності розпізнавання

Мета: Переконатися, що система розпізнає п’ять класів з середньою впевненістю ≥0.9 та точністю ≥95%.

Порядок:

- Завантажити вибірку (3250 зображень, по 650 для кожного класу).
- Виконати пакетну обробку за допомогою скрипта main_yolo.py (див. Додаток Г, Лістинг Г.1).
- Переглянути лог-файл і графіки (counts_plot.png, accuracy_plot.png).

Очікуваний результат: Середня впевненість ≥0.9, точність ≥95%.

Критерій успіху: Середня впевненість ≥0.9, точність ≥95%.

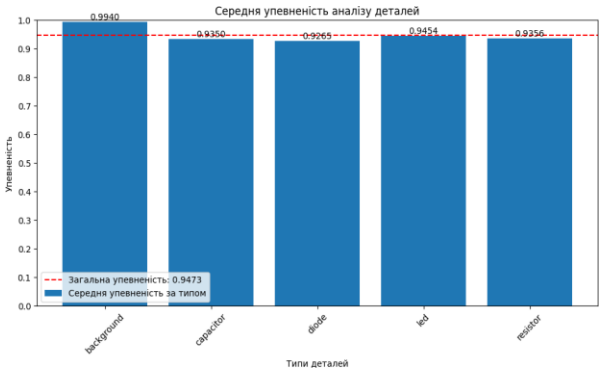


Рисунок В.1 – Графік середньої впевненості розпізнавання.

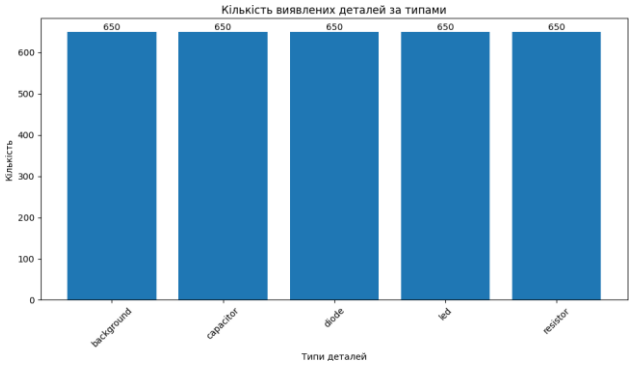


Рисунок В.2 – Графік кількісті виявлених деталей за типом.

Тест 2. Перевірка швидкості пакетної обробки

Мета: Переконатися, що 3250 зображень обробляються за ≤30 секунд. Порядок:

- Запустити скрипт main_yolo.py (див. Додаток Г, Лістинг Г.1) з batch=16.
- Зафіксувати час обробки в лог-файлі.
- Переглянути графік часу обробки зображень.

Очікуваний результат: Час обробки ≤30 секунд, середній час на зображення ≤9 мс.

Критерій успіху: Загальний час обробки не перевищує 30 секунд.

```
Час обробки при batch=16: 20.91 секунд
Кількість оброблених зображень: 3250
```

Рисунок В.3 – Час обробки із лог-файлу.

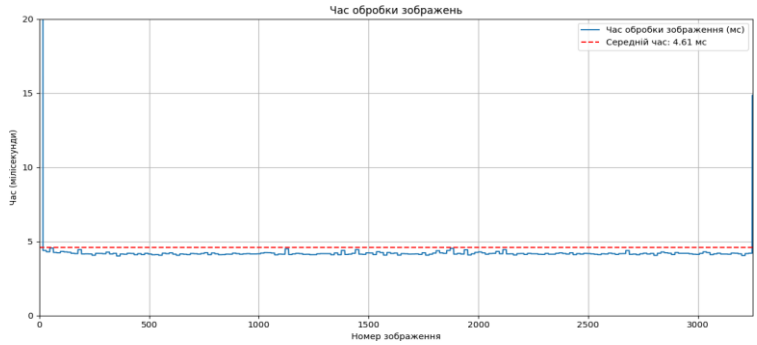


Рисунок В.4 – Графік часу обробки зображень.

Тест 3. Перевірка роботи в реальному часі

Мета: Переконатися, що система забезпечує 10-15 FPS у реальному часі.
Порядок:

- Підключити телефон через EpocCam.
- Запустити скрипт main_phone.py (див. Додаток Г, Лістинг Г.2).
- Показати радіодеталі перед камерою.
- Переглянути графік часу обробки зображень.

Очікуваний результат: Частота кадрів ≥10-15 FPS, час обробки кадру ≤66 мс.
Критерій успіху: Система стабільно розпізнає об’єкти при зміні умов.

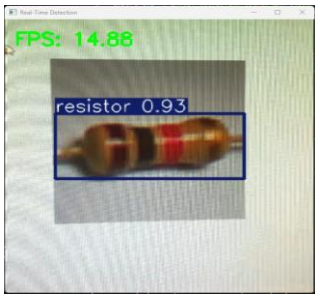


Рисунок В.5 – Приклад розпізнавання в реальному часі.

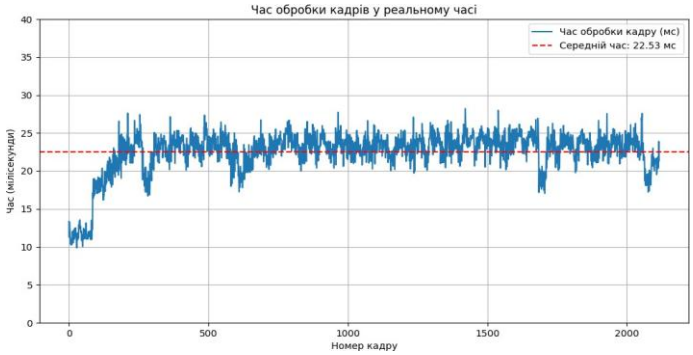


Рисунок В.6 – Графік часу обробки зображень у реальному часі.

Висновки: У тестах було успішно досягнуто поставлених швидкостей та точності. Випробовування пройдено успішно.

Виконавець: студент групи КІ41, БОЖКО Т.В. _____

Додаток Г

Лістинг Г.1 – Скрипт main_yolo.py для пакетної обробки зображень.

```
import cv2
import os
import numpy as np
from ultralytics import YOLO
from collections import defaultdict
import matplotlib.pyplot as plt
import logging
import time
import torch
import concurrent.futures
from typing import List

# Перевірка доступності GPU для прискорення обчислень
if torch.cuda.is_available():
    device = torch.device("cuda:0")  # Встановлення GPU як основного пристрою
    print(f"GPU доступний: {torch.cuda.get_device_name(0)}")
else:
    device = torch.device("cpu")  # Встановлення CPU, якщо GPU недоступний
    print("GPU недоступний, використовується CPU.")

# Налаштування шляхів і параметрів для обробки зображень
MODEL_PATH = "C:\\dyplom\\bestx.pt"  # Шлях до файлу навченої моделі YOLOv8
IMAGE_DIR = "C:\\dyplom\\3250all"  # Шлях до каталогу з вхідними зображеннями
(роздільна здатність 320x320)
CONFIDENCE_THRESHOLD = 0.5  # Поріг упевненості для виявлення об’єктів (0-1)
BATCH_SIZE = 16  # Розмір батчу для пакетної обробки зображень

# Створення унікальної вихідної папки для збереження результатів
base_output_dir = os.path.join(IMAGE_DIR, "output")  # Визначення базового шляху
для вихідної папки
output_dir = base_output_dir
counter = 1
while os.path.exists(output_dir):
    output_dir = f"{base_output_dir}{counter}"  # Додавання номера до назви
папки, якщо вона вже існує
    counter += 1
os.makedirs(output_dir, exist_ok=True)  # Створення папки для збереження
результатів

# Налаштування логування для запису результатів обробки
log_file_path = os.path.join(output_dir, 'predictions.log')  # Шлях до файлу
логування
logging.basicConfig(filename=log_file_path, level=logging.INFO,
                    format='%(asctime)s - %(message)s', encoding='utf-8')
logging.info(f"Логування ініціалізовано в: {log_file_path}")

# Логування інформації про пристрій, який використовується
if torch.cuda.is_available():
    logging.info(f"GPU доступний: {torch.cuda.get_device_name(0)}")
else:
    logging.info("GPU недоступний, використовується CPU.")

# Завантаження моделі YOLOv8
model = YOLO(MODEL_PATH)  # Ініціалізація моделі
model.model.to(device)  # Переміщення моделі на вибраний пристрій (GPU/CPU)
print(f"Модель завантажено успішно. Класи: {model.model.names}")
logging.info(f"Модель завантажено успішно. Класи: {model.model.names}")

# Ініціалізація змінних для статистики
object_counts = defaultdict(int)  # Лічильник об’єктів за класами
```

```
confidence_sums = defaultdict(float)  # Сума значень упевненості для обчислення середньої точності
processing_times = []  # Список для збереження часу обробки кожного зображення

# Функція для завантаження зображення
def load_image(image_path: str) -> np.ndarray:
    """
    Завантажує зображення з диска та повертає його як масив NumPy.

    Args:
        image_path (str): Шлях до файлу зображення.

    Returns:
        np.ndarray: Завантажене зображення у форматі масиву NumPy або None у разі помилки.
    """
    image = cv2.imread(image_path)  # Завантаження зображення за допомогою OpenCV
    if image is None:
        logging.error(f"Помилка: не вдалося завантажити зображення {image_path}")
        return None
    return image

# Функція для пакетної обробки зображень
def process_batch(images: List[np.ndarray], image_paths: List[str]) -> None:
    """
    Обробляє батч зображень моделлю YOLO, анотує результати та зберігає їх.

    Args:
        images (List[np.ndarray]): Список зображень для обробки.
        image_paths (List[str]): Список шляхів до файлів зображень.
    """
    global processing_times
    start_time = time.time()  # Початок вимірювання часу обробки батчу
    with torch.inference_mode():  # Відключення обчислення градієнтів для прискорення
        results = model(images, conf=CONFIDENCE_THRESHOLD, iou=0.5, device=device)
    batch_processing_time = (time.time() - start_time) * 1000  # Час обробки батчу в мілісекундах
    time_per_image = batch_processing_time / len(images) if len(images) > 0 else 0  # Середній час на зображення
    processing_times.extend([time_per_image] * len(images))  # Збереження часу для кожного зображення

    for i, result in enumerate(results):
        annotated_image = result.plot()  # Анотація зображення з виявленими об’єктами
        detected_objects = result.boxes  # Отримання інформації про виявлені об’єкти
        image_name = os.path.basename(image_paths[i])  # Отримання імені файлу зображення

        if len(detected_objects) > 0:
            logging.info(f"Об’єкти виявлено для {image_name}:")
            print(f"Об’єкти виявлено для {image_name}:")
            for box in detected_objects:
                label_id = int(box.cls.item())  # Ідентифікатор класу об’єкта
                score = box.conf.item()  # Упевненість у виявленні
                label = model.model.names[label_id]  # Назва класу
                message = f"Об’єкт: {label}, Упевненість: {score:.4f}"
                print(message)
```

```
                logging.info(message)
                object_counts[label] += 1
                confidence_sums[label] += score
        else:
            message = f"Об’єкти не виявлено для {image_name}."
            print(message)
            logging.info(message)

        output_path = os.path.join(output_dir, image_name.replace('.',
'_output.'))  # Шлях для збереження анотованого зображення
        cv2.imwrite(output_path, annotated_image)  # Збереження зображення

# Завантаження та обробка зображень
image_files = sorted([f for f in os.listdir(IMAGE_DIR) if
f.lower().endswith(('.png', '.jpg', '.jpeg'))])  # Список файлів зображень
image_paths = [os.path.join(IMAGE_DIR, f) for f in image_files]  # Повні шляхи
до зображень

images = []  # Список для завантажених зображень
with concurrent.futures.ThreadPoolExecutor() as executor:  # Паралельне
завантаження зображень
    images = list(executor.map(load_image, image_paths))
images = [img for img in images if img is not None]  # Фільтрація зображень, які
не вдалося завантажити
image_paths = [image_paths[i] for i, img in enumerate(images) if img is not
None]  # Фільтрація відповідних шляхів

total_images_processed = 0  # Лічильник оброблених зображень
start_time = None  # Час початку обробки
for i in range(0, len(images), BATCH_SIZE):
    batch_images = images[i:i + BATCH_SIZE]  # Вибір батчу зображень
    batch_paths = image_paths[i:i + BATCH_SIZE]  # Вибір відповідних шляхів
    total_images_processed += len(batch_images)
    if start_time is None:
        start_time = time.time()  # Початок вимірювання часу
    if batch_images:
        print(f"\nОбробка батчу з {len(batch_images)} зображень...")
        logging.info(f"Обробка батчу з {len(batch_images)} зображень...")
        process_batch(batch_images, batch_paths)
    else:
        print("\nПропущено порожній батч.")
        logging.info("Пропущено порожній батч.")

# Виведення місця збереження результатів
print(f"\nРезультати обробки збережено в: {output_dir}")
logging.info(f"Результати обробки збережено в: {output_dir}")

# Виведення статистики виявлених об’єктів
print("\nСтатистика об’єктів:")
logging.info("\nСтатистика об’єктів:")
total_objects = sum(object_counts.values())  # Загальна кількість виявлених
об’єктів
total_confidence = sum(confidence_sums.values())  # Загальна сума упевненості
for label, count in sorted(object_counts.items()):
    avg_confidence = confidence_sums[label] / count if count > 0 else 0  #
Середня упевненість для класу
    print(f"{label}: {count} раз(ів), Середня упевненість:
{avg_confidence:.4f}")
    logging.info(f"{label}: {count} раз(ів), Середня упевненість:
{avg_confidence:.4f}")
if total_objects > 0:
    overall_accuracy = total_confidence / total_objects  # Загальна середня
упевненість
```

```
    print(f"Загальна середня упевненість: {overall_accuracy:.4f}")
    logging.info(f"Загальна середня упевненість: {overall_accuracy:.4f}")
else:
    print("Об’єкти не виявлено, загальна упевненість не обчислюється.")
    logging.info("Об’єкти не виявлено, загальна упевненість не обчислюється.")

# Вимірювання та виведення часу обробки
if start_time is not None:
    end_time = time.time()  # Час завершення обробки
    processing_time = end_time - start_time  # Загальний час обробки
    print(f"\nЧас обробки при batch={BATCH_SIZE}: {processing_time:.2f} секунд")
    print(f"Кількість оброблених зображень: {total_images_processed}")
    logging.info(f"Час обробки при batch={BATCH_SIZE}: {processing_time:.2f}
секунд")
    logging.info(f"Кількість оброблених зображень: {total_images_processed}")
else:
    print("\nЧас обробки не вимірювався (об’єктів не оброблено).")
    logging.info("Час обробки не вимірювався (об’єктів не оброблено).")

# Візуалізація статистики за допомогою графіків
labels, counts = zip(*sorted(object_counts.items()))  # Підготовка даних для
графіків
avg_confidences = [confidence_sums[label] / count if count > 0 else 0 for label
in labels]  # Середня упевненість для кожного класу

# Графік 1: Кількість виявлених об’єктів за типами
plt.figure(figsize=(10, 6))
bars = plt.bar(labels, counts)
plt.title('Кількість виявлених деталей за типами')
plt.xlabel('Типи деталей')
plt.ylabel('Кількість')
plt.xticks(rotation=45)
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2., height, f'{int(height)}',
ha='center', va='bottom')
plt.tight_layout()
plt.savefig(os.path.join(output_dir, 'counts_plot.png'))
plt.close()

# Графік 2: Середня упевненість для кожного типу
plt.figure(figsize=(10, 6))
bars = plt.bar(labels, avg_confidences, label='Середня упевненість за типом')
plt.axhline(y=overall_accuracy, color='red', linestyle='--', label=f'Загальна
упевненість: {overall_accuracy:.4f}')
plt.title('Середня упевненість аналізу деталей')
plt.xlabel('Типи деталей')
plt.ylabel('Упевненість')
plt.xticks(rotation=45)
plt.ylim(0, 1)
plt.yticks(np.arange(0, 1.1, 0.1))
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2., height, f'{height:.4f}',
ha='center', va='bottom')
plt.legend()
plt.tight_layout()
plt.savefig(os.path.join(output_dir, 'accuracy_plot.png'))
plt.close()

# Графік 3: Час обробки зображень
plt.figure(figsize=(12, 6))
```

```
plt.plot(range(len(processing_times)), processing_times, label='Час обробки
зображення (мс)')
if processing_times:
    mean_time = np.mean(processing_times)  # Середній час обробки
    plt.axhline(y=mean_time, color='red', linestyle='--', label=f'Середній час:
{mean_time:.2f} мс')
plt.title('Час обробки зображень')
plt.xlabel('Номер зображення')
plt.ylabel('Час (мілісекунди)')
plt.ylim(0, 20)
plt.yticks(np.arange(0, 21, 5))
plt.xlim(0, total_images_processed)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig(os.path.join(output_dir, 'processing_time_plot.png'))
plt.close()

# Виведення місця збереження графіків
print(f"Графіки збережено в: {output_dir}")
logging.info(f"Графіки збережено в: {output_dir}")
```

Лістинг Г.2 – Скрипт main_phone.py для аналізу в реальному часі.

```
import cv2
import os
import numpy as np
from ultralytics import YOLO
from collections import defaultdict
import matplotlib.pyplot as plt
import logging
import time
import torch
from typing import List, Tuple
# Перевірка доступності GPU для прискорення обчислень
if torch.cuda.is_available():
    device = torch.device("cuda:0")  # Встановлення GPU як основного пристрою
    print(f"GPU доступний: {torch.cuda.get_device_name(0)}")
else:
    device = torch.device("cpu")  # Встановлення CPU, якщо GPU недоступний
    print("GPU недоступний, використовується CPU.")

# Налаштування шляхів і параметрів
MODEL_PATH = "C:\\dyplom\\bestx.pt"  # Шлях до файлу навченої моделі YOLOv8
CONFIDENCE_THRESHOLD = 0.8  # Поріг упевненості для виявлення об’єктів (0-1)
BATCH_SIZE = 1  # Розмір батчу для обробки (1 для роботи з камерою)
TARGET_FPS = 15  # Обмеження частоти кадрів для стабільної роботи (15 FPS)

# Створення папки для збереження результатів
base_output_dir = os.path.join("c:\\dyplom\\3250all\\", "output")  # Базовий
шлях для вихідної папки
output_dir = base_output_dir
counter = 1
while os.path.exists(output_dir):
    output_dir = f"{base_output_dir}{counter}"  # Додавання номера до назви
папки, якщо вона вже існує
    counter += 1
os.makedirs(output_dir, exist_ok=True)  # Створення папки для збереження
результатів

# Налаштування логування для запису результатів обробки
logging.basicConfig(filename=os.path.join(output_dir, 'predictions.log'),
level=logging.INFO,
                    format='%(asctime)s - %(message)s', encoding='utf-8')
logging.info(f"Логування ініціалізовано в: {os.path.join(output_dir,
'predictions.log')}")

# Логування інформації про пристрій
if torch.cuda.is_available():
    logging.info(f"GPU доступний: {torch.cuda.get_device_name(0)}")
else:
    logging.info("GPU недоступний, використовується CPU.")

# Завантаження моделі YOLOv8
model = YOLO(MODEL_PATH)  # Ініціалізація моделі
model.model.to(device)  # Переміщення моделі на вибраний пристрій (GPU/CPU)
print(f"Модель завантажена. Класи: {model.model.names}")
logging.info(f"Модель завантажена. Класи: {model.model.names}")

# Ініціалізація змінних для статистики
object_counts = defaultdict(int)  # Лічильник об’єктів за класами
confidence_sums = defaultdict(float)# Сума значень упевненості
frame_timestamps = []  # Список для збереження часу обробки кадрів

# Функція для обрізки кадру до 320x320 та видалення чорних рамок
def crop_to_320x320_clean(frame: np.ndarray) -> np.ndarray:
```

```
    height, width = frame.shape[:2]
    min_dim = min(width, height)  # Визначення мінімального розміру для
квадратного кадру
    start_x = (width - min_dim) // 2
    start_y = (height - min_dim) // 2
    square_frame = frame[start_y:start_y + min_dim, start_x:start_x +
min_dim]  # Обрізка до квадрата

    start_x = (min_dim - 320) // 2
    start_y = (min_dim - 320) // 2
    cropped_frame = square_frame[start_y:start_y + 320, start_x:start_x +
320]  # Обрізка до 320x320

    gray = cv2.cvtColor(cropped_frame, cv2.COLOR_BGR2GRAY)  # Переведення в
градації сірого
    _, thresh = cv2.threshold(gray, 10, 255, cv2.THRESH_BINARY)  # Бінаризація
для видалення чорних рамок
    coords = cv2.findNonZero(thresh)  # Пошук ненульових пікселів
    if coords is not None:
        x, y, w, h = cv2.boundingRect(coords)  # Визначення обмежуючого
прямокутника
        cropped_frame = cropped_frame[y:y + h, x:x + w]  # Обрізка до ненульової
області
        if cropped_frame.shape[0] < 320 or cropped_frame.shape[1] < 320:
            cropped_frame = cv2.resize(cropped_frame, (320, 320),
interpolation=cv2.INTER_AREA)  # Зміна розміру до 320x320
    return cropped_frame

# Функція обробки кадру
def process_frame(frame: np.ndarray) -> Tuple[str, np.ndarray]:
    """
    Обробляє кадр камери моделлю YOLO, анотує виявлені об’єкти та зберігає
результат.

    Args:
        frame: Вхідний кадр для обробки.

    Returns:
        tuple: Шлях до збереженого кадру та анотований кадр.
    """
    global object_counts, confidence_sums
    cleaned_frame = crop_to_320x320_clean(frame)  # Підготовка кадру
    start_time = time.time()  # Початок вимірювання часу обробки
    with torch.inference_mode():  # Відключення обчислення градієнтів для
прискорення
        results = model(cleaned_frame, conf=CONFIDENCE_THRESHOLD, iou=0.5,
device=device)
    annotated_frame = results[0].plot()  # Анотація кадру з виявленими об’єктами
    detected_objects = results[0].boxes  # Отримання інформації про виявлені
об’єкти

    if len(detected_objects) > 0:
        objects_info = []
        for box in detected_objects:
            conf = box.conf.item()
            # Перевірка, що значення впевненості в межах [0, 1]
            if conf < 0 or conf > 1:
                logging.warning(f"Некоректне значення впевненості: {conf},
пропускаємо.")
                continue
            if conf >= 0.3:  # Фільтрація за мінімальним порогом для виведення
                label = model.model.names[int(box.cls.item())]
                objects_info.append(f"{label}: {conf:.4f}")
```

```
                # Оновлення статистики
                object_counts[label] += 1
                confidence_sums[label] += conf

        if objects_info:
            summary = f"Виявлено об’єкти: {', '.join(objects_info)}"
            print(summary)
            logging.info(summary)
    else:
        print("Об’єкти не виявлено.")
        logging.info("Об’єкти не виявлено.")

    timestamp = cv2.getTickCount()  # Отримання унікальної мітки часу для імені файлу
    output_path = os.path.join(output_dir, f"frame_{timestamp}_output.jpg")  # Шлях для збереження кадру
    cv2.imwrite(output_path, annotated_frame)  # Збереження анотованого кадру
    frame_timestamps.append((time.time() - start_time) * 1000)  # Збереження часу обробки в мілісекундах
    return output_path, annotated_frame

# Підключення до камери
camera_index = 0  # Індекс камери (зазвичай 0 для вбудованої камери)
cap = cv2.VideoCapture(camera_index, cv2.CAP_DSHOW)  # Ініціалізація камери з використанням DirectShow

if not cap.isOpened():
    print(f"Помилка: камера {camera_index} недоступна!")
    logging.error(f"Помилка: камера {camera_index} недоступна!")
    exit()

# Отримання параметрів камери
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))
print(f"Роздільна здатність камери: {width}x{height}, {fps} FPS")

# Ініціалізація змінних для підрахунку FPS і статистики
prev_time = time.time()  # Час для підрахунку FPS
frame_count = 0  # Лічильник кадрів для FPS
fps_display = 0  # Поточне значення FPS для відображення
total_frames_processed = 0  # Загальна кількість оброблених кадрів
start_time = time.time()  # Час початку обробки

# Основний цикл обробки кадрів
window_name = 'Real-Time Detection'  # Назва вікна для відображення
cv2.namedWindow(window_name, cv2.WINDOW_NORMAL)  # Створення вікна для відображення кадрів
while True:
    frame_start_time = time.time()  # Початок обробки кадру

    ret, frame = cap.read()  # Зчитування кадру з камери
    if not ret:
        print("Помилка: не вдалося отримати кадр!")
        logging.error("Помилка: не вдалося отримати кадр!")
        break
    output_path, annotated_frame = process_frame(frame)  # Обробка кадру
    total_frames_processed += 1

    # Підрахунок FPS
    frame_count += 1
    current_time = time.time()
    elapsed_time = current_time - prev_time
```

```
    if elapsed_time >= 1.0:  # Оновлення FPS кожну секунду
        fps_display = frame_count / elapsed_time
        frame_count = 0
        prev_time = current_time

    # Відображення FPS на кадрі
    text = f"FPS: {fps_display:.2f}"
    cv2.putText(annotated_frame, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0, 255, 0), 2)

    cv2.imshow(window_name, annotated_frame)  # Відображення кадру у вікні
    # Обмеження частоти кадрів до TARGET_FPS
    frame_processing_time = time.time() - frame_start_time
    frame_time_target = 1.0 / TARGET_FPS  # Час на кадр для TARGET_FPS
    delay = max(1, int((frame_time_target - frame_processing_time) * 1000))  #
Затримка в мілісекундах
    if cv2.waitKey(delay) == ord('q'):  # Вихід за натисканням 'q'
        break

# Завершення роботи
cap.release()  # Звільнення камери
cv2.destroyAllWindows()  # Закриття всіх вікон OpenCV

# Виведення місця збереження файлів
saved_files = [f for f in os.listdir(output_dir) if f.endswith('_output.jpg')]
if saved_files:
    print(f"Збережені файли знаходяться в: {output_dir}")
    logging.info(f"Збережені файли знаходяться в: {output_dir}")

# Виведення статистики виявлених об’єктів
print("\nСтатистика об’єктів:")
logging.info("\nСтатистика об’єктів:")
total_objects = sum(object_counts.values())
total_confidence = sum(confidence_sums.values())
avg_confidences_dict = {}  # Словник для зберігання середньої впевненості
for label, count in sorted(object_counts.items()):
    avg_confidence = confidence_sums[label] / count if count > 0 else 0
    avg_confidences_dict[label] = avg_confidence
    print(f"{label}: {count} раз(ів), Середня упевненість:
{avg_confidence:.4f}")
    logging.info(f"{label}: {count} раз(ів), Середня упевненість:
{avg_confidence:.4f}")
if total_objects > 0:
    overall_accuracy = total_confidence / total_objects
    print(f"Загальна середня упевненість: {overall_accuracy:.4f}")
    logging.info(f"Загальна середня упевненість: {overall_accuracy:.4f}")
else:
    print("Об’єкти не виявлено, загальна упевненість не обчислюється.")
    logging.info("Об’єкти не виявлено, загальна упевненість не обчислюється.")

# Виведення часу обробки
end_time = time.time()
processing_time = end_time - start_time
print(f"\nЧас обробки: {processing_time:.2f} секунд")
print(f"Кількість оброблених кадрів: {total_frames_processed}")
logging.info(f"Час обробки: {processing_time:.2f} секунд")
logging.info(f"Кількість оброблених кадрів: {total_frames_processed}")

# Візуалізація статистики за допомогою графіків
if object_counts:
    labels = sorted(object_counts.keys())
    counts = [object_counts[label] for label in labels]
    # Використовуємо значення з avg_confidences_dict для узгодженості
```

```
    avg_confidences = [avg_confidences_dict[label] for label in labels]

    # Графік 1: Кількість виявлених об’єктів за типами
    plt.figure(figsize=(max(10, len(labels) * 0.5), 6))
    bars = plt.bar(labels, counts)
    plt.title('Кількість виявлених деталей за типами')
    plt.xlabel('Типи деталей')
    plt.ylabel('Кількість')
    plt.xticks(rotation=45, ha='right')
    for bar in bars:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2., height, f'{int(height)}',
ha='center', va='bottom')
    plt.subplots_adjust(bottom=0.3, top=0.9)
    plt.savefig(os.path.join(output_dir, 'counts_plot.png'))
    plt.close()

    # Графік 2: Середня упевненість для кожного типу (виправлений)
    plt.figure(figsize=(max(10, len(labels) * 0.5), 6))
    # Фільтрація значень, щоб уникнути виходу за межі [0, 1]
    avg_confidences = [min(max(conf, 0), 1) for conf in avg_confidences]
    bars = plt.bar(labels, avg_confidences, label='Середня упевненість за
типом', color='skyblue')
    plt.axhline(y=overall_accuracy, color='red', linestyle='--',
label=f'Загальна упевненість: {overall_accuracy:.4f}')
    plt.title('Середня упевненість аналізу деталей')
    plt.xlabel('Типи деталей')
    plt.ylabel('Упевненість')
    plt.xticks(rotation=45, ha='right')
    plt.ylim(0, 1.1)  # Збільшуємо верхню межу, щоб текст не обрізався
    plt.yticks(np.arange(0, 1.1, 0.1))
    for bar in bars:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2., height + 0.02,
f'{height:.4f}', ha='center', va='bottom')
    plt.legend()
    plt.grid(True, axis='y', linestyle='--', alpha=0.7)
    plt.subplots_adjust(bottom=0.3, top=0.9)
    plt.savefig(os.path.join(output_dir, 'accuracy_plot.png'))
    plt.close()
    # Графік 3: Час обробки кадрів
    plt.figure(figsize=(12, 6))
    filtered_timestamps = [t for t in frame_timestamps if t <= 100]  #
Фільтрація викидів
    plt.plot(filtered_timestamps, label='Час обробки кадру (мс)')
    if filtered_timestamps:
        mean_time = np.mean(filtered_timestamps)  # Середній час обробки
        plt.axhline(y=mean_time, color='red', linestyle='--', label=f'Середній
час: {mean_time:.2f} мс')
    plt.title('Час обробки кадрів у реальному часі')
    plt.xlabel('Номер кадру')
    plt.ylabel('Час (мілісекунди)')
    plt.ylim(0, 40)
    plt.yticks(np.arange(0, 41, 5))
    plt.legend()
    plt.grid(True)
    plt.savefig(os.path.join(output_dir, 'frame_processing_time_plot.png'))
    plt.close()
    print(f"Графіки збережено в: {output_dir}")
    logging.info(f"Графіки збережено в: {output_dir}")
else:
    print("Графіки не створено, оскільки об’єкти не виявлено.")
    logging.info("Графіки не створено, оскільки об’єкти не виявлено.")
```

Лістинг Г.3 – Скрипт stetch.py для зміни роздільної здатності зображень.

```
import cv2
import os

# Вихідний каталог із зображеннями 96x96
SOURCE_DIR = "C:\\dyplom"  # Шлях до каталогу із зображеннями
TARGET_DIR = os.path.join(SOURCE_DIR, "320x320")  # Цільовий каталог для
зображень 320x320

# Створення цільового каталогу, якщо його не існує
os.makedirs(TARGET_DIR, exist_ok=True)  # Забезпечення наявності цільового
каталогу

# Підтримувані розширення файлів зображень
IMAGE_EXTENSIONS = ('.png', '.jpg', '.jpeg')  # Формати зображень для обробки

# Отримання списку файлів зображень у вихідному каталозі
image_files = [f for f in os.listdir(SOURCE_DIR) if
f.lower().endswith(IMAGE_EXTENSIONS)]

for idx, image_file in enumerate(image_files, 1):  # Цикл по зображеннях із
нумерацією від 1
    SOURCE_PATH = os.path.join(SOURCE_DIR, image_file)  # Повний шлях до
вихідного зображення

    # Завантаження зображення через OpenCV
    image = cv2.imread(SOURCE_PATH)
    if image is None:
        print(f"Помилка завантаження зображення {image_file}. Пропущено.")  #
Повідомлення про помилку
        continue

    # Зміна розміру до 320x320 з інтерполяцією INTER_AREA
    image_resized = cv2.resize(image, (320, 320), interpolation=cv2.INTER_AREA)

    # Збереження зміненого зображення з новою назвою
    TARGET_PATH = os.path.join(TARGET_DIR,
f"resized_{idx}{os.path.splitext(image_file)[1]}")
    cv2.imwrite(TARGET_PATH, image_resized)  # Збереження результату
    print(f"Оброблено: {image_file} → {TARGET_PATH}")

print(f"Обробка завершена. Результати збережено у {TARGET_DIR}")
```

Лістинг Г.4 – Скрипт convert_labels.py для конвертації міток.

```
import os
import json

# Шляхи до каталогів для роботи з даними
DATA_DIR = "C:\\dyplom\\yolo"  # Кореневий каталог для YOLO даних
TRAINING_DIR = os.path.join(DATA_DIR, "training")  # Каталог тренувальних
зображень
TESTING_DIR = os.path.join(DATA_DIR, "testing")  # Каталог тестових зображень
LABELS_DIR = os.path.join(DATA_DIR, "labels")  # Каталог для міток

# Створення підкаталогів для міток, якщо їх немає
TRAIN_LABELS_DIR = os.path.join(LABELS_DIR, "train")  # Мітки тренувальних даних
VAL_LABELS_DIR = os.path.join(LABELS_DIR, "val")  # Мітки валідаційних даних
os.makedirs(TRAIN_LABELS_DIR, exist_ok=True)  # Забезпечення наявності каталогу
train
os.makedirs(VAL_LABELS_DIR, exist_ok=True)  # Забезпечення наявності каталогу
val

# Відповідність назв класів числовим ідентифікаторам
CLASS_MAP = {
    "background": 0,
    "capacitor": 1,
    "diode": 2,
    "led": 3,
    "resistor": 4
}

def convert_labels_to_yolo(source_dir, target_dir, source_type="train"):
    """
    Перетворює мітки з JSON (Edge Impulse) у TXT (YOLO).
    Аргументи:
        source_dir: Каталог із зображеннями та JSON-файлом міток
        target_dir: Каталог для TXT-файлів YOLO
        source_type: Тип даних ('train' або 'val')
    """
    print(f"Сканування {source_dir} для пошуку міток...")

    # Пошук файлу міток
    label_file = os.path.join(source_dir, "bounding_boxes.labels")
    if not os.path.exists(label_file):
        print(f"Файл міток {label_file} не знайдено.")  # Повідомлення про
відсутність файлу
        return

    print(f"Знайдено файл міток: {label_file}")

    # Завантаження JSON із мітками
    with open(label_file, "r") as f:
        labels_data = json.load(f)

    bounding_boxes = labels_data.get("boundingBoxes", {})
    if not bounding_boxes:
        print(f"Мітки у {label_file} відсутні.")  # Повідомлення про порожній
JSON
        return

    # Список зображень у каталозі
    image_files = {f: f for f in os.listdir(source_dir) if
f.lower().endswith(('.png', '.jpg', '.jpeg'))}
    processed_images = set()  # Множина для уникнення дублів

    # Перетворення міток для кожного зображення
```

```
    for full_image_name in bounding_boxes.keys():
        if full_image_name not in image_files:
            print(f"Зображення {full_image_name} не знайдено. Пропущено.")  #
Повідомлення про відсутність зображення
            continue

        txt_base_name = os.path.splitext(full_image_name)[0]  # Базова назва для
TXT-файлу
        txt_path = os.path.join(target_dir, f"{txt_base_name}.txt")

        if txt_base_name in processed_images:
            print(f"Дубль {txt_base_name}. Пропущено.")  # Повідомлення про
дубль
            continue
        processed_images.add(txt_base_name)

        # Запис міток у форматі YOLO
        with open(txt_path, "w") as f_out:
            for bbox in bounding_boxes[full_image_name]:
                label = bbox["label"]
                if label not in CLASS_MAP:
                    print(f"Невідомий клас {label} у {full_image_name}.
Пропущено.")  # Повідомлення про невідомий клас
                    continue

                class_id = CLASS_MAP[label]  # Ідентифікатор класу
                x = float(bbox["x"])
                y = float(bbox["y"])
                width = float(bbox["width"])
                height = float(bbox["height"])

                # Нормалізація координат для YOLO (зображення 320x320)
                x_center = (x + width / 2) / 320
                y_center = (y + height / 2) / 320
                norm_width = width / 320
                norm_height = height / 320

                f_out.write(f"{class_id} {x_center} {y_center} {norm_width}
{norm_height}\n")
        print(f"Створено: {txt_path}")

# Перетворення міток для тренувальних і тестових даних
convert_labels_to_yolo(TRAINING_DIR, TRAIN_LABELS_DIR, "train")
convert_labels_to_yolo(TESTING_DIR, VAL_LABELS_DIR, "val")

print(f"Мітки збережено у {TRAIN_LABELS_DIR} та {VAL_LABELS_DIR}")
```

Лістинг Г.5 – Скрипт classify_3250_photos.py для класифікації аугментованого набору даних на основі базового набору.

```
from ultralytics import YOLO
import os
import json
import cv2
# Шляхи до каталогів із даними
DATA_DIR = "C:\\dyplom\\yolo"  # Кореневий каталог проєкту
TRAIN_IMAGES_DIR = os.path.join(DATA_DIR, "training")  # Тренувальні зображення
VAL_IMAGES_DIR = os.path.join(DATA_DIR, "testing")  # Тестові зображення
NEW_IMAGES_DIR = os.path.join(DATA_DIR, "3250")  # Нові зображення для
передбачення
OUTPUT_DIR = os.path.join(DATA_DIR, "yolo_annotations")  # Каталог для
результатів

# Створення каталогу для результатів
os.makedirs(OUTPUT_DIR, exist_ok=True)#Забезпечення наявності вихідного каталогу

# Конфігурація для YOLO у форматі YAML
data_yaml = """
train: {TRAIN_IMAGES_DIR}
val: {VAL_IMAGES_DIR}
nc: 5
names: ['background', 'capacitor', 'diode', 'led', 'resistor']
"""

with open(os.path.join(DATA_DIR, "data.yaml"), "w") as f:
    f.write(data_yaml.format(TRAIN_IMAGES_DIR=TRAIN_IMAGES_DIR,
VAL_IMAGES_DIR=VAL_IMAGES_DIR))

def train_and_predict():
    """Тренує модель YOLOv8n і виконує передбачення для нових зображень."""
    # Завантаження моделі YOLOv8n
    model = YOLO("yolov8n.pt")  # Легка модель YOLOv8 Nano
    # Тренування моделі на тренувальних даних
    model.train(
        data=os.path.join(DATA_DIR, "data.yaml"),  # Шлях до конфігурації
        epochs=5,  # Кількість епох
        imgsz=320,  # Розмір зображень 320x320
        batch=16,   # Розмір батчу
        project=DATA_DIR,  # Каталог для результатів тренування
        name="train_yolo",  # Назва експерименту
        exist_ok=True  # Дозвіл на перезапис
    )
    # Передбачення для нових зображень
    results = model.predict(
        source=NEW_IMAGES_DIR,  # Каталог із новими зображеннями
        save=True,  # Збереження анотованих зображень
        conf=0.3,   # Поріг упевненості
        iou=0.5,    # Поріг IoU для NMS
        project=OUTPUT_DIR,  # Каталог для результатів передбачення
        name="predictions",  # Назва експерименту
        exist_ok=True
    )
    # Формування анотацій у форматі COCO JSON
    coco_annotations = {
        "images": [],
        "annotations": [],
        "categories": [{"id": i, "name": name} for i, name in
enumerate(['background', 'capacitor', 'diode', 'led', 'resistor'])],
        "info": {"description": "Анотації для Edge Impulse"}
    }
```

```
    image_id = 0
    annotation_id = 0

    for result in results:
        image_path = result.path
        image = cv2.imread(image_path)
        h, w = image.shape[:2]

        coco_annotations["images"].append({
            "id": image_id,
            "file_name": os.path.basename(image_path),
            "width": w,
            "height": h
        })

        for box in result.boxes:
            x1, y1, x2, y2 = box.xyxy[0].tolist()  # Координати рамки
            conf = box.conf.item()  # Упевненість
            cls = int(box.cls.item())  # Ідентифікатор класу

            # Перетворення у формат COCO
            x = x1
            y = y1
            width = x2 - x1
            height = y2 - y1
            coco_annotations["annotations"].append({
                "id": annotation_id,
                "image_id": image_id,
                "category_id": cls,
                "bbox": [x, y, width, height],
                "area": width * height,
                "iscrowd": 0,
                "score": conf
            })
            annotation_id += 1
        image_id += 1

    # Збереження анотацій у COCO JSON
    coco_path = os.path.join(OUTPUT_DIR, "annotations_coco.json")
    with open(coco_path, "w") as f:
        json.dump(coco_annotations, f, indent=4)
    print(f"Анотації COCO збережено: {coco_path}")

    # Збереження анотацій у форматі YOLO TXT
    for result in results:
        image_path = result.path
        base_name = os.path.splitext(os.path.basename(image_path))[0]
        txt_path = os.path.join(OUTPUT_DIR, f"{base_name}.txt")

        with open(txt_path, "w") as f:
            for box in result.boxes:
                cls = int(box.cls.item())  # Ідентифікатор класу
                x_center = (box.xyxy[0][0] + box.xyxy[0][2]) / 2 / 320  #
Нормалізовані координати
                y_center = (box.xyxy[0][1] + box.xyxy[0][3]) / 2 / 320
                width = (box.xyxy[0][2] - box.xyxy[0][0]) / 320
                height = (box.xyxy[0][3] - box.xyxy[0][1]) / 320
                f.write(f"{cls} {x_center} {y_center} {width} {height}\n")
    print(f"Анотації YOLO TXT збережено у {OUTPUT_DIR}")
if __name__ == "__main__":
    train_and_predict()
```

Лістинг Г.6 – Скрипт yolox_model.py для тренування моделі YOLOv8x.

```
import os
from ultralytics import YOLO

# Шляхи до каталогів із даними
DATA_DIR = "C:\\dyplom\\yolo"  # Кореневий каталог даних
TRAIN_IMAGES_DIR = os.path.join(DATA_DIR, "training")  # Тренувальні зображення
VAL_IMAGES_DIR = os.path.join(DATA_DIR, "testing")  # Тестові зображення
NEW_IMAGES_DIR = os.path.join(DATA_DIR, "3250")  # Нові зображення для
передбачення
OUTPUT_DIR = os.path.join(DATA_DIR, "yolo_annotations")  # Каталог для
результатів

# Створення вихідного каталогу
os.makedirs(OUTPUT_DIR, exist_ok=True)  # Забезпечення наявності каталогу

def create_data_yaml(train_images, val_images, labels_dir, output_path):
    """
    Створює файл data.yaml для тренування YOLO.
    Аргументи:
        train_images: Шлях до тренувальних зображень
        val_images: Шлях до валідаційних зображень
        labels_dir: Кореневий каталог міток
        output_path: Шлях для збереження data.yaml
    """
    data_yaml_content = f"""
train: {train_images}
val: {val_images}
nc: 5
names: ['background', 'capacitor', 'diode', 'led', 'resistor']
path: {labels_dir}
    """
    with open(output_path, "w") as f:
        f.write(data_yaml_content.strip())  # Збереження конфігурації
    print(f"Створено: {output_path}")

def main():
    # Створення конфігураційного файлу
    data_yaml_path = os.path.join(DATA_DIR, "data.yaml")
    create_data_yaml(
        train_images=TRAIN_IMAGES_DIR,
        val_images=VAL_IMAGES_DIR,
        labels_dir=DATA_DIR,
        output_path=data_yaml_path
    )

    # Завантаження моделі YOLOv8x
    print("Тренування моделі YOLOv8x...")
    model = YOLO("yolov8x.pt")  # Високоточна модель YOLOv8x

    # Тренування моделі
    model.train(
        data=data_yaml_path,  # Шлях до конфігурації
        epochs=50,  # Кількість епох
        imgsz=320,  # Розмір зображень 320x320
        batch=16,   # Розмір батчу
        project=OUTPUT_DIR,  # Каталог результатів
        name="train_yolo_x",  # Назва експерименту
        exist_ok=True,  # Дозвіл на перезапис
        device=0,  # Використання GPU
        patience=20,  # Рання зупинка після 20 епох без покращення
        val=True,  # Валідація на тестових даних
        augment=True,  # Аугментація даних
```

```
            hsv_h=0.015,  # Варіація відтінку
            hsv_s=0.7,  # Варіація насиченості
            hsv_v=0.4,  # Варіація яскравості
            translate=0.1,  # Зсув зображення
            scale=0.5,  # Масштабування
            fliplr=0.5,  # Горизонтальне відображення
            lr0=0.001,  # Початковий learning rate
            lrf=0.01  # Кінцевий learning rate
        )

        # Оцінка моделі на тестових даних
        print("Оцінка моделі на тестовому наборі...")
        model.val(data=data_yaml_path)

        # Передбачення для нових зображень, якщо каталог існує
        if os.path.exists(NEW_IMAGES_DIR):
            print("Передбачення для нових зображень...")
            model.predict(
                source=NEW_IMAGES_DIR,  # Каталог із новими зображеннями
                save=True,  # Збереження результатів
                conf=0.3,  # Поріг упевненості
                iou=0.5,  # Поріг IoU
                project=OUTPUT_DIR,  # Каталог результатів
                name="predictions",  # Назва експерименту
                exist_ok=True
            )

        # Збереження моделі
        final_model_path = os.path.join(OUTPUT_DIR, "train_yolo_x", "weights",
"best_final.pt")
        model.save(final_model_path)  # Збереження у форматі .pt
        print(f"Модель збережено: {final_model_path}")

if __name__ == "__main__":
    main()  # Запуск основної функції
```