Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Комп'ютерна система обробки зображень
з використанням машинного навчання

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-5
Олександр АВРУНІН
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: ст. викл. Владислав ДЯЧЕНКО
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Аврунінy Олександру Олеговичу
(прізвище, ім’я, по батькові)

1. Тема роботи Комп'ютерна система обробки зображень з використанням машинного навчання

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 16 червня 2025 р.

3. Вхідні дані до роботи
комп'ютерна система
мобільний застосунок
штучна нейронна мережа
машинне навчання
обробка зображень

4. Перелік питань, що потрібно опрацювати у роботі
Аналіз предметної області та постановка завдання
Аналіз існуючих мобільних застосунків
Вибір програмного забезпечення та реалізація

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій 11 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання та аналіз літератури	26.05.2025–31.05.2025	
2	Аналіз існуючих методів обробки зображень	01.06.2025–03.06.2025	
3	Аналіз методів машинного навчання	04.06.2025–06.06.2025	
4	Розробка комп'ютерної системи та мобільного	07.06.2025–08.06.2025	
	застосунку		
5	Вибір програмних засобів	09.06.2025–11.06.2025	
6	Реалізація, тестування та аналіз результатів	12.06.2025–13.06.2025	
7	Оформлення пояснювальної записки	14.06.2025–15.06.2025	

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Владислав ДЯЧЕНКО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 61 с., 36 рис., 2 дод., 8 джерел.

ОБРОБКА ЗОБРАЖЕНЬ, НЕЙРОМЕРЕЖА, КОМП'ЮТЕРНИЙ ЗІР, ANDROID, YOLO, TENSORFLOW, ВІДЕОСПОСТЕРЕЖЕННЯ.

Метою кваліфікаційної роботи є розробка комп’ютерної системи класифікації об’єктів на зображеннях.

У ході виконання кваліфікаційної роботи кваліфікаційної роботи було проведено аналіз існуючих рішень для цього завдання та методи обробки зображень. Для класифікації використовується модель сімейства YOLO – це потужна нейронна мережа, що використовується для пошуку об’єктів на зображенні. На цей момент доступна 11 версія цієї моделі. Використання моделі YOLO у мобільному застосунку непередбачено, тому після завершення навчання було прийнято рішення про зміну формату моделі до підтримуваного та оптимізованого для мобільних пристроїв – TensorFlow Lite.

ABSTRACT

Bachelor’s thesis: 61 pages, 36 figures, 2 appendices, 8 sources.

IMAGE CLASSIFICATION, NEURAL NETWORK, COMPUTER VISION, ANDROID, YOLO, TENSORFLOW, VIDEO SURVEILLANCE

The aim of this qualification thesis is to develop a computer system for object classification in images. During the implementation of the thesis, an analysis of existing solutions for this task and image processing methods was conducted.

In order to complete this task, a model from the YOLO family is used. It is a powerful neural network designed for object detection in images. Currently, the 11th version of this model is available. Since using the YOLO model directly in a mobile application is not supported, the model was converted to a optimized for mobile devices – TensorFlow Lite, which is developed to ensure efficient performance on the small and mostly low-power devices, such as smartphones, Arduino and STM32 plates.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ШНМ – штучна нейронна мережа

IOS – iPhone operating system

SIFT (Scale-Invariant Feature Transform)

SURF (Speeded-Up Robust Features)

NMS – Non-Maximum Suppression

YOLO – you look only once

ВСТУП

Все більше процесів потребують автоматизації та/або зменшення рівня відповідальності, що лежить на людині. Це дозволяє також знизити суб'єктивність при аналізі інформації, зокрема, зображень. Цьому сприяють великі промислові системи, однією зі складових яких є нейромережі.

Використання нейромереж не є новинкою, вже багато років на виробництвах вони допомагають людям слідкувати за якістю готової продукції або процесом її виготовлення. З розвитком технологій нейромережі стали доступними і для приватних користувачів. як найбільш розповсюджений приклад, застосунок «Камера» у смартфонах вже давно використовує штучний інтелект та «розумні» аналізатори для покращення якості фото – підвищення деталізації, розширення динамічного діапазону, вибір режиму зйомки відповідно до умов тощо.

Метою кваліфікаційної роботи є виявлення об’єктів, що бачить камера пристрою – чи то смартфон, чи камера відеоспостереження, чи дрона-розвідника. Для досягнення цієї мети було використано сучасну архітектуру для класифікації об’єктів на зображеннях YOLO, яка зарекомендувала себе як одна з найефективніших у вирішенні подібних завдань.

Застосування комп’ютерного зору охоплює багато аспектів нашого життя, такі як системи безпеки, контроль виготовлення об’єктів, системи візуального контролю та управління, контроль транспортних засобів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Методи розпізнавання образів

Розпізнавання образів – це процес виділення із вихідних даних необхідної інформації, для чого використовуються спеціальні алгоритми. Це потужні технології, які сьогодні знаходять широке застосування у багатьох галузях. Сфери, де вони вже впроваджуються:

- у медицині системи розпізнавання образів допомагають лікарям виявляти патології на рентгенівських знімках, сканах магнітно-резонансної та комп'ютерної томографії. Застосування технології розпізнавання образів збільшує точність діагнозів та допомагає лікарю не втратити деталі знімку[1];

- у автомобільній галузі технології розпізнавання образів використовуються для створення систем у допомозі керування транспортним засобом, виявлення пішоходів та автомобілів, а також для розробки автономних транспортних засобів;

- у сільскому господарстві розпізнавання образів застосовується для моніторингу стану посівів, виявлення хвороб рослин, визначення вологості ґрунту тощо;

- у робототехниці роботи, які обладнані системами розпізнавання образів, можуть аналізувати оточуюче середовище, переміщуватися, виконувати завдання в місцях, де потрібне взаємодія з людьми;

- у сфері безпеки та відеоспостереженні найбільше поширення знайшла ця технологія в сфері ідентифікації осіб та контролю доступу, виявлення аномальної поведінки людей та автоматичного аналізу відеопотоків для забезпечення безпеки в громадських місцях та на підприємствах;

- маркетинг та реклама. Розпізнавання образів дозволяє компаніям аналізувати дані про споживачів, надаючи персоналізовані рекомендації та

рекламні комунікації, поліпшуючи таким чином досвід користувачів;

- у промисловості та виробництві ці технології використовуються для контролю якості продукції, оптимізації процесів, виявлення дефектів на виробничій лінії тощо;
- у сфері розваг розпізнавання образів використовується для створення інтерактивних ігор та віртуальної реальності, допомагає збагачувати геймплей, наприклад генерацією ландшафту, та візуальний досвід гравців.

Важливо пам’ятати, що на даний момент не існує універсальних алгоритмів порівняння зображень, тому необхідно створити спеціальні алгоритми, параметри яких підбираються користувачем, виходячи з конкретних умов. Щоб розібратись у питанні, необхідно дізнатись яким чином можливо виділяти цікаві об’єкти на зображенні.

1.1.1 Сегментація зображень

Цифрове зображення складається з кінцевої кількості елементів, кожен з яких розташований в конкретному місці та набуває певного значення. Найчастіше для елементів цифрового зображення використовується термін «піксель».

Сегментація зображення – це процес виділення об’єкту від фону. Цей крок необхідний для того, щоб спростити зображення для його аналізу. Використовуючи різні методи сегментації зображення, можливо кожен піксель віднести до певного класу та групувати їх.

Розрізнюють попередню (грубу, або семантичну) та повну (з розміченням об’єктів та визначенням окремих екземплярів сегментацію екземплярів) стадії сегментації. Проте, основною умовою для успішної сегментації є відсутність «накладання» об’єктів, як на рисунку 1.1:

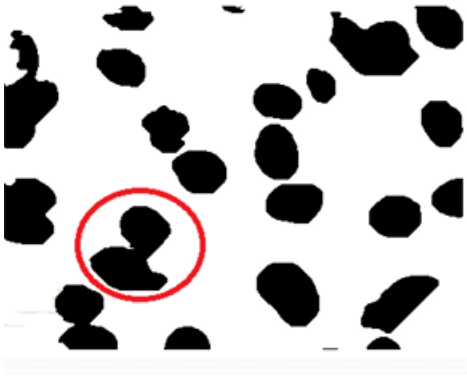

Рисунок 1.1 – Накладання нейронів мозку

Попередня (груба) сегментація виконується побудовою бінарної характеристичної функції зображення. У цьому випадку, необхідно визначення порогу «T» сегментації. Це може бути середнє значення між двома максимуми або поступова зміна шляхом виключення пікових значень. Цей підхід має дуже суттєвий недолік – гістограма залежить від умов освітленості. На рисунку 1.2 продемонстровано, що фактично одне й те саме зображення, але із різним налаштуванням яскравості має різну гістограму.

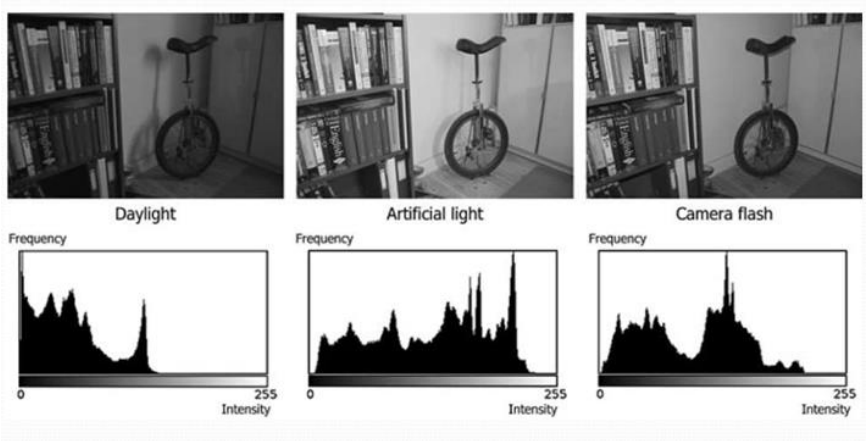

Рисунок 1.2 – Залежність гістограм від умов освітлення

Необхідно розуміти, який саме об’єкт нас цікавить. Іноді, цих об’єктів декілька та вони можуть відрізнятись один від одного за, наприклад, рівнем інтенсивності. У цьому випадку, необхідно визначити окрему бінарну характеристичну функцію для кожного класу об’єктів.

Якщо проводити аналіз гістограми за окремими ділянками, можна отримувати статистичні характеристики, такі як екстремальні та середні значення і стандартне відхилення (рисунок 1.3):

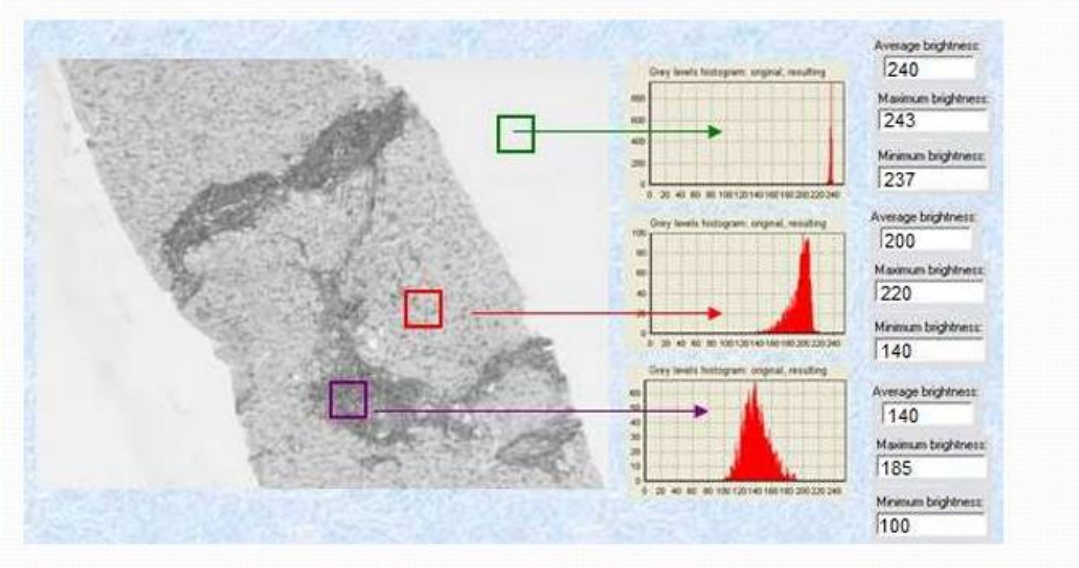

Рисунок 1.3 – Локальні екстремуми та гістограми

На зображенні фон може неоднорідний. Об’єкт та фон можуть зливатись та при різних порогах один чи другий об’єкти будуть неправильно виділятись або взагалі зникати. Відповідно доцільно використовувати динамічний поріг або вирівняти фон (рисунок 1.4).

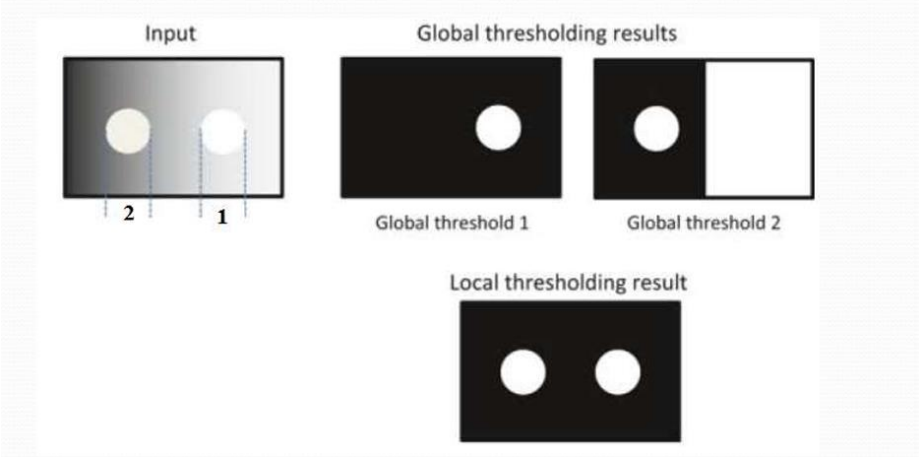

Рисунок 1.4 – Приклад сегментації за глобальним та локальними порогами інтенсивності

1.1.2 Використання нейромереж

Нейронні мережі, або нейромережі, є одним з найважливіших інструментів сучасного штучного інтелекту. Свою назву вони отримали завдяки схожості по своїй структурі до нейрону мозку.

Нервова тканина складається з нервових клітин – нейронів і міжклітинної речовини – нейроглії (рисунок 1.5). Нейрон – основна структурно-функціональна одиниця нервової системи. Він складається з тіла ті відростків: дендритів та аксонів. По дендритах проводиться збудження до тіла нейрона, по аксонах – від тіла до інших клітин.

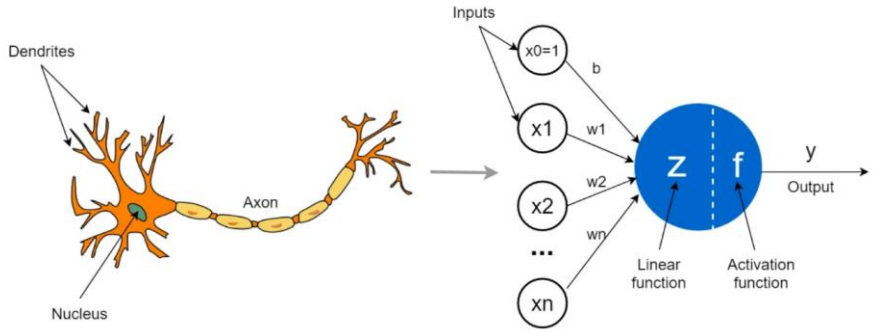

Рисунок 1.5 – Структура нейрона мозку та перцептрона

Перцептрон – це найпростіший тип штучної нейронної мережі, розроблений Френком Розенблаттом у 1957 році. Це базовий елемент нейронних мереж, який імітує функціонування біологічного нейрона. Перцептрон можна використовувати для задач класифікації, зокрема для розпізнавання образів і категоризації даних.

Перцептрон складається з наступних компонентів:

- входи (Inputs) – вхідні сигнали, які подаються на перцептрон. Кожен вхід має відповідну вагу;

- ваги (Weights) – кожен вхід множиться на вагу, яка визначає важливість цього входу;

- сума (Summation) – всі вхідні сигнали, помножені на свої ваги, підсумовуються;

- активаційна функція (Activation Function) – отримана сума передається через активаційну функцію, яка визначає вихідний сигнал перцептрона;

- вихід (Output) – результат, який видає перцептрон після обробки вхідних сигналів.

Перцептрон зазвичай використовується для задач бінарної класифікації. Він має два можливі виходи (класи), наприклад, 0 і 1. Проте його можна адаптувати для багатокласової класифікації, використовуючи декілька перцептронів в архітектурі з одним виходом для кожного класу або шляхом використання методів, таких як one-vs-all (один проти всіх).

Основним недоліком є відсутність здатності до узагальнення своїх характеристик на нові стимули або нові ситуації, а також нездатність аналізувати складні ситуації у зовнішньому середовищі шляхом розчленування їх на простіші.

Будь-яка нейромережа складається з 3 основних частин – вхідний шар, приховані шари та вихідний шар:

- вхідний шар це буквально шар, який вводить інформацію для нейронної мережі для обробки;

- приховані шари – це шари, які виконують всю обробку для нейронних мереж. Чим більше шарів існує, тим точніше буде нейронна мережа;

- вихідний шар це шар, який просто об'єднує інформацію з останнього прихованого шару мережі для виведення всієї необхідної інформації з програми.

Машинне навчання – це клас методів штучного інтелекту, що розв’язує задачу не прямим способом, а шляхом пошуку закономірностей у даних після навчання алгоритму на безлічі прикладів. Ці алгоритми можуть знаходити певні об’єкти на фотографіях, пішоходів перед автомобілями шукати спам.

У традиційному програмуванні під час виконання завдання комп’ютер слідує заздалегідь визначеним інструкціям. Однак у машинному навчанні системі дається набір зразків, за допомогою яких їй необхідно з’ясувати, яким чином розв’язати поставлену задачу. На базовому рівні машинне навчання можна поділити на три типи – навчання з учителем, навчання без вчителя та навчання з підкріпленням[1].

Навчання з учителем (supervised learning) – передбачає, що система тренується знаходити закономірності на власному прикладі. Під час навчання алгоритму надають масиви розмічених даних, наприклад, зображення рукописних цифр з анотаціями, що вказують на їхній відповідний номер. За наявності достатньої кількості прикладів система навчиться розпізнавати кластери пікселів і форм, пов’язаних із кожним об’єктом.

Навчання без вчителя спрямоване на тренування алгоритму виявляти закономірності у вхідній інформації. Система тренується виявляти подібності в даних і розділяти їх на категорії. Алгоритми навчання без вчителя не призначені для виділення конкретних типів даних. Вони призначені для знаходження схожості певних характеристик.

Навчання з підкріпленням передбачає, що ШІ-агенти тренуються ухвалювати рішення, взаємодіючи з певним середовищем. За кожні вжиті дії їх винагороджують або карають балами. Мета агента – максимізувати загальну кількість очок. Щоб зрозуміти суть навчання з підкріпленням, необхідно подумати про те, як людина вперше грає в комп’ютерну гру без знань правил і управління. Вона може виявитися абсолютним новачком, але дивлячись на взаємозв’язок між натисканнями кнопок, і тим, що відбувається на екрані, продуктивність гравця буде збільшуватися. Прикладом може слугувати машина, яка вчиться поступати місце іншим учаникам дорожнього руху, реагувати на пішоходів, спецтранспорт та навіть правопорушників.

Крім того, існують так звані методи навчання з підкріпленням (semi-supervised learning), що розташовані приблизно посередині між машинним навчанням з учителем та машинним навчанням без учителя.

Перехресне навчання (Cross-validation) – це метод, що використовується для оцінки точності моделі машинного навчання та уникнення перенавчання. Зазвичай дані розділяються на навчальний і тестовий набори. Модель навчається на навчальному наборі, а потім перевіряється на тестовому для оцінки її ефективності.

1.1.3 Метод SURF

SURF вирішує дві задачі – пошук особливих точок зображення і створення їх дескрипторів, інваріантних до масштабу і обертанню. Опис ключової точки буде однаково, навіть якщо зразок змінить розмір і буде обернутий. Крім того, сам пошук ключових точок володіє інваріантністю. Так, що повернений об'єкт сцени має той же набір ключових точок, що і зразок [3].

Таблиця інтегрального зображення або підсумкова область створена в 1984 році. Інтегральне зображення використовується як швидкий та ефективний спосіб обчислення суми значень у даному зображенні – або прямокутної підмножини сітки. Він також використовується для розрахунку середньої інтенсивності в межах даного зображення [4].

Цей алгоритм шукає ключові точки та будує опис знайдених ключових точок через дескриптори особливостей і є аналогом методу SIFT. Ключовою точкою є локальний екстремум детермінанта матриці Гессе. Гаусіан є інваріантним до зміщення яскравості та обертання, але не до масштабу, тому для досягнення інваріантності до масштабу використовуються різні масштаби та фільтри. Цю проблему вирішують через багаторазове застосування гаусових фільтрів на різних масштабах, що дає змогу знаходити ключові точки в межах різних рівнів деталізації.

Як опорні точки вибирають локальні максимуми гессіанів, які відповідають локальним максимумам зміни градієнта яскравості. Після знаходження точок локальних максимумів визначають точку істинного

максимуму гессіана. Ця методика гарантує, що в околі ключової точки розташовані ділянки з різними градієнтами.

Для обчислення детермінанта матриці Гессе спочатку необхідно застосувати згортку з гаусовим ядром, потім похідною другого порядку. Після успіху Лоу з наближеннями LoG (SIFT), SURF висуває апроксимацію (як згортку, так і похідну другого порядку) ще далі за допомогою віконних фільтрів. Ці приблизні гаусівські похідні другого порядку будуть за дуже низькою обчислювальною вартістю за допомогою інтегральних зображень і незалежно від розміру, і це є частиною причини швидкого SURF (рисунок 1.6).

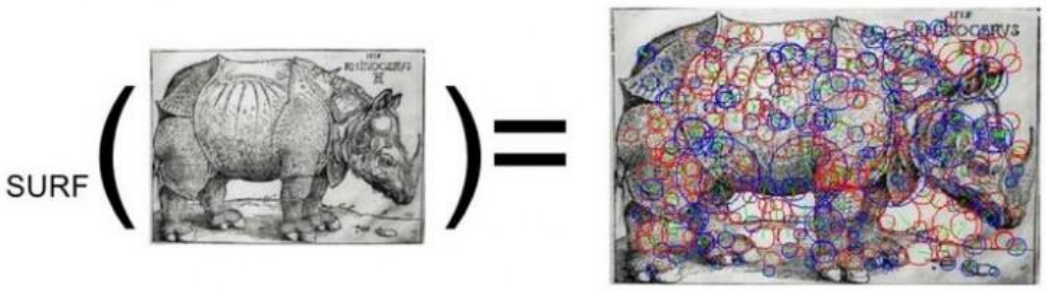

Рисунок 1.6 – Використання методу SURF для визначення маркеру додаткової реальності

Масштабні розміри здебільшого реалізуються через пірамідальні представлення зображень. У цьому підході зображення поступово згладжуються за допомогою гаусових фільтрів, а потім зменшуються для отримання підвибірок, що відповідають вищим рівням піраміди. Використання віконних фільтрів та інтегральних зображень дозволяє SURF уникати ітеративного застосування одного і того ж фільтра до виходу попередньо обробленого шару.

Натомість цей метод може застосовувати фільтри різних розмірів до вихідного зображення з однаковою швидкістю, що забезпечує паралельну обробку.

Щоб забезпечити інваріантність до обертання, SURF прагне визначити стабільну орієнтацію для кожної точки інтересу. Спершу створюється квадратна область, центрована навколо ключової точки та орієнтована відповідно до напрямку обертання. Потім ця область ділиться на 16 менших квадратів, як показано на рисунку 1.4. У кожному з цих кадратів розміщується регулярна сітка розміром 5x5. Для кожної точки сітки обчислюється градієнт, застосовуючи фільтр Хаара. Розмір цього фільтра визначається як 2s, де s – масштаб. Для першого рівня октави фільтр має розмір 4x4.

Сам алгоритм методу SURF (Speeded Up Robust Features) передбачає виконання таких етапів:

- масштабно-просторове представлення;
- розрахунок значень гессіана;
- пошук точок локальних максимумів. Визначення точки істинного максимуму;
- визначення орієнтації опорної точки;
- формування дескриптора опорної точки.

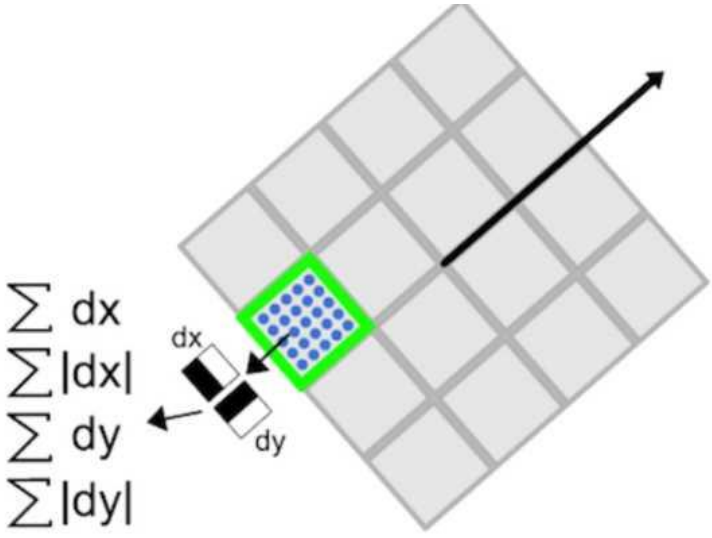

Рисунок 1.7 – Приклад роботи десткриптору

1.1.4 Метод BRIEF

SURF потребує щонайменше 256 байтів пам’яті для зберігання 64-вимірного дескриптора. Аналогічно, SIFT використовує 128-вимірний вектор дескрипторів, що складається з чисел з плаваючою комою, і займає приблизно 512 байтів. Створення таких векторів для тисяч ключових точок споживає значну кількість пам’яті, що є неприйнятним для систем із обмеженими ресурсами, особливо для вбудованих систем. Відповідно, чим більший об’єм пам’яті використовується — тим більше часу потрібно на обчислення та співставлення ознак.

BRIEF – дескриптор з низькою швидкістю передачі бітів, введений для зіставлення зображень з випадковими класифікаторами типу лісу і дерев. Він належить до сімейства двійкових дескрипторів, виконує простий двійковий тест порівняння з використанням відстані Хеммінга [4].

Важливо зазначити, що BRIEF – це лише дескриптор ознак. Він не виконує пошуку самих ознак. Потрібно використовувати інші детектори ознак, наприклад як SIFT чи SURF.

Після виявлення ключової точки продовжується обчислення дескриптора для кожної з них. Дескриптори функцій кодують інформацію в числовій формі, яка виступає свого роду цифровим "відбитком", що дозволяє відрізнити одну функцію від іншої. Визначена область навколо пікселя, відома як патч, має форму квадрата, розміри якого визначаються шириною і висотою пікселя.

Ефективна класифікація зразків зображень можлива завдяки порівняно невеликій кількості парних порівнянь інтенсивності. Коротке перетворення патчів зображень у бінарний вектор ознак здійснюється так, щоб він міг репрезентувати об'єкт. Бінарний вектор ознак, також відомий як дескриптор бінарних ознак, містить тільки 1 і 0. Кожна ключова точка представлена вектором ознак у вигляді рядка довжиною 128-512 біт (рисунок 1.8).

Метод BRIEF кодує зображення на рівні пікселів, але залишається чутливим до шуму. Для зменшення цієї чутливості використовується попереднє згладжування патча за допомогою Гаусівського ядра з дисперсією, рівною 2, і розміром 9x9 пікселів (рисунок 1.9).

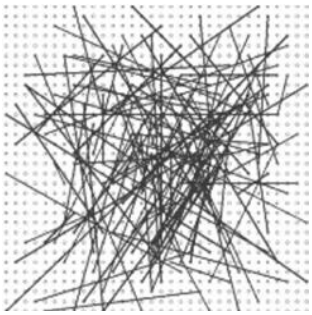

Рисунок 1.8 – Приклад роботи дескриптору BRIEF

Рисунок 1.9 – Згладжування зображення

Для створення вектора бінарних ознак із патча спочатку генерується бінарний вектор характеристик двійкових тестових відповідей. Вектор довжиною n біт формується з вибраних випадкових пар точок (x, y), які знаходяться всередині патча. Дескриптор BRIEF має два основні параметри налаштування: кількість двійкових пар пікселів і двійковий поріг.

Усього вибирається n таких тестів із п'яти можливих геометричних підходів:

Підходи для знаходження пар (рисунок 1.10):

- уніформа (G I): пікселі X та У випадкової парі витягуються з Unifrom distribution або поширення S / 2 навколо ключової точки. Пара (тест) може лежати поблизу кордону;

- гауссова (G II): Пікселі x та y витягуються у випадковій парі з розподілу Гаусса або поширення 0,04 * S2 навколо ключової точки;

- гауссова (G III): Перший піксель обирається з розподілу Гаусса навколо ключової точки, а другий — з розподілу навколо першого пікселя з меншою дисперсією (0,01 * S2);

- груба полярна сітка (G IV): Пікселі вибираються з дискретних позицій у межах полярної сітки;

- груба полярна сітка (G V): Перший піксель знаходиться у центрі (0, 0), другий вибирається із полярної сітки.

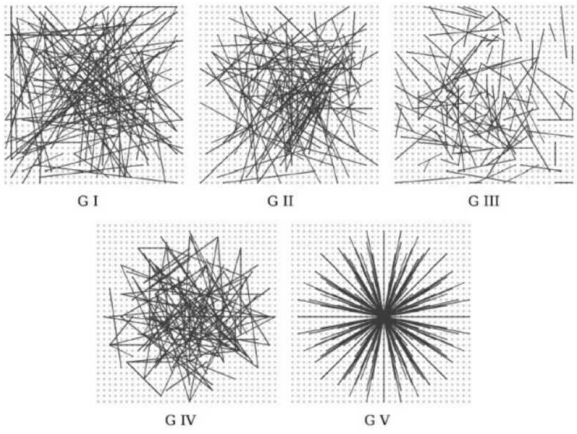

Рисунок 1.10 – Візуалізація для знаходення пар

До основних недоліків методу BRIEF можна віднести неефективний вибір пар пікселів для побудови дескриптора, а також відсутність врахування орієнтації ключової точки під час розпізнавання.

1.2 Постановка задачі

Кінцевим продуктом є натренована нейромережа, яка здатна розпізнавати військову техніку у режимі реального часу та застосунок для перевірки працездатності. Нейромережа повинна відрізняти важку броньовану техніку(танк) та легку броньовану техніку (БМП/БТР тощо). Нейромережа може продовжувати навчання на нових наборах даних.

2 АНАЛІЗ ІСНУЮЧИХ МОБІЛЬНИХ ЗАСТОСУНКІВ

2.1 Графічний інтерфейс

Під час розробки програмного забезпечення важливо аналізувати вже існуючі додатки. Це дає змогу виявити їхні переваги й недоліки, ознайомитися з підходами до реалізації подібних завдань і відмітити ідеї щодо побудови зручного та ефективного інтерфейсу користувача. Цей проект стосується розробки як ШНМ, так і мобільного додатку.

У всіх розглянутих програмах використовується модифікація застосунку «Камера» у будь-якому смартфоні (рисунки 2.1 та 2.2). Це робить інтерфейс дружнім до користувача та не перевантажує його:

Рисунок 2.1 та 2.2 – Інтерфейси застосунків «Камера»
у смартфонах Samsung та Apple

У даному розділі розглядатимуться наступні програми:

- ObjectDetectionApp;
- TensorFlow Lite Flutter Object Detectio;
- InstaLens;
- Google Lens;
- Tensorflow-demo.

2.2 ObjectDetectionApp

ObjectDetectionApp – це застосунок для Android, призначений для виявлення об’єктів за допомогою TensorFlow Lite на мобільному пристрої. У ньому використовується попередньо натренована квантизована модель MobileNet SSD. Вона навчена на датасеті ImageNet і здатна розпізнавати близько 90 класів об’єктів, включаючи банани, ножиці, ноутбуки, пульти, вази тощо.

«Основною проблемою при розпізнаванні об’єктів на мобільних пристроях раніше було те, що моделі були надто «важкими» для запуску. TensorFlow Lite вирішує цю проблему, надаючи легку платформу для запуску моделей машинного навчання на мобільних пристроях. Ви самі можете побачити, як добре працює застосунок і як швидко він виявляє кілька об’єктів одночасно» – так описує свою програму розробник.

Дійсно, обробка зображень і аналіз моделлю ШНМ дуже вимогливі завдання. На демонстраційному відео можна побачити, що картинка оновлюється значно швидше ніж малюються прямокутники над знайденими об’єктами класів (рисунок 2.3):

Рисунок 2.3 – Робота програми ObjectDetectionApp

Ця програма виконує свою функцію, проте в неї єдиним елементом дизайну є видошукач. З демонстраційного відео видно, що детекція

проводиться із значною затримкою. Хоча розробник не описав технічну реалізацію застосунку, це можу бути спричинено відсутністю апаратного прискорення.

2.3 TensorFlow Lite Flutter Object Detection

На жаль, розробник не надав опису роботи та ключових переваг свого застосунку. Однак варто звернути увагу на низку важливих аспектів, які все ж можна відзначити під час аналізу функціональності програми:

- збір базової статистики. Інтерфейс представлено у вигляді видошукача, але також можливо відстежувати час безпосередньої інтерпретації моделі, час, витрачений на попередню обробку зображення, загальний час обробки та роздільна здатність;
- додаток є кросплатформеним оскільки базується на Flutter. Це надає можливість його запуску як на пристроях з Android, так і на iOS. Це є значною перевагою, адже забезпечує ширше охоплення аудиторії без необхідності створення окремих додатків для кожної платформи.

На рисунку 2.4 представлено інтерфейс користувача.

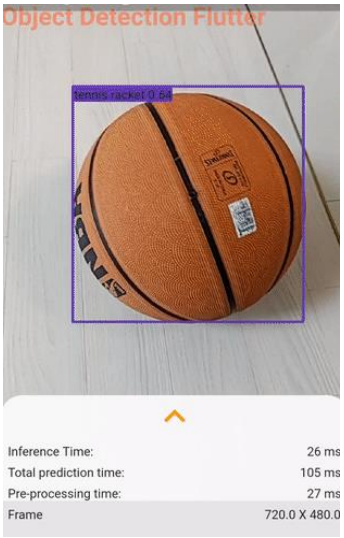

Рисунок 2.4 – Інтерфейс програми TensorFlow Lite Flutter Object Detection

2.4 InstaLens

InstaLens – це мобільний застосунок для розпізнавання об’єктів, який використовує можливості TensorFlow Lite для ідентифікації об'єктів у реальному часі за допомогою камери пристрою. Головними перевагами цього продукту є:

- застосунок забезпечує підтримку як основної (задньої), так і фронтальної камери, що надає користувачам гнучкість у використанні. Це особливо корисно в різних сценаріях – наприклад, при наведенні камери на навколишні об'єкти або під час використання додатку для селфі;

- інтерфейс користувача (рисунок 2.5) містить зручний повзунок для регулювання порогу впевненості в розпізнаванні. Це дозволяє користувачеві фільтрувати лише ті об’єкти, які система розпізнала з достатньо високим рівнем точності;

- додаток здатний обробляти відеопотік з камери та здійснювати розпізнавання об’єктів у реальному часі без помітних затримок чи «підвисань», забезпечуючи плавний досвід для користувача;

- для зручності користувачів InstaLens підтримує обидва режими інтерфейсу: світлий і темний. Це дозволяє адаптувати вигляд додатку до умов освітлення або особистих уподобань користувача.

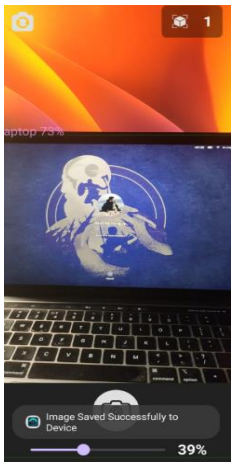

Рисунок 2.5 – Інтерфейс програми InstaLens

2.5 Google Lens

Цей застосунок дозволяє користувачам сканувати навколишнє середовище за допомогою камери смартфона (рисунок 2.5) і отримувати інформацію про побачене. Додаток інтегрований у багато сучасних пристроїв Android, а також доступний як окремий додаток у Google Play.

Головною відмінністю цього застосунку від інших є те, що:

- по-перше, він вимагає використання інтернету;
- по-друге всі обчислення та аналіз виконуються потужностями серверів Google.

Це суттєві відмінності, проте під час розгляду необхідно вивчити якомога більше варіантів реалізацій.

Взагалі, Google Lens це приклад того, як інновації у сфері штучного інтелекту та комп'ютерного зору інтегруються в повсякденне життя користувача. Це дійсно високофункціональний застосунок, здатний допомогти людям у багатьох справах, наприклад:

- пошук товарів, предметів людей;
- читання надписів та їх переклад.

Рисунок 2.6 – Пошук породи собаки за знімком

2.6 Tensorflow-demo

Застосунок створено на платформі Flutter з використанням пакету tflite_flutter для інтеграції з моделлю TensorFlow Lite. Його мета – забезпечити зручне та ефективне розпізнавання об’єктів як у реальному часі через камеру пристрою, так і на статичних зображеннях. Завдяки кросплатформеності Flutter, застосунок легко адаптується як для Android, так і для IOS, що робить його універсальним інструментом для демонстрації можливостей мобільного машинного навчання.

Можна сказати, що цей застосунок має великий потенціал – функції завантаження фото із галереї, можливість заміни моделі ШНМ а також кросплатформеність його дуже зручним. Інтерфейс та демонстрація можливостей наведено на рисунку 2.7:

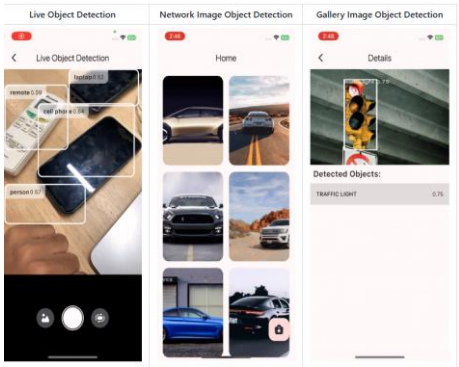

Рисунок 2.7 – Інтерфейс застосунку Tensorflow-demo

2.7 Розробка інтерфейсу програми

На основі аналізу програм найкращим варіантом є використання класичної схеми видошукача із кнопкою «catch», яка дозволяє зберегти останнє зображення із рамкою. Сучасні застосунки камери надають можливість вручну підлаштовувати параметри до умов зйомки, оскільки «розумний» та режим «авто» не завжди можуть правильно проаналізувати зображення і відповідно можуть погіршити кінцеву якість фото.

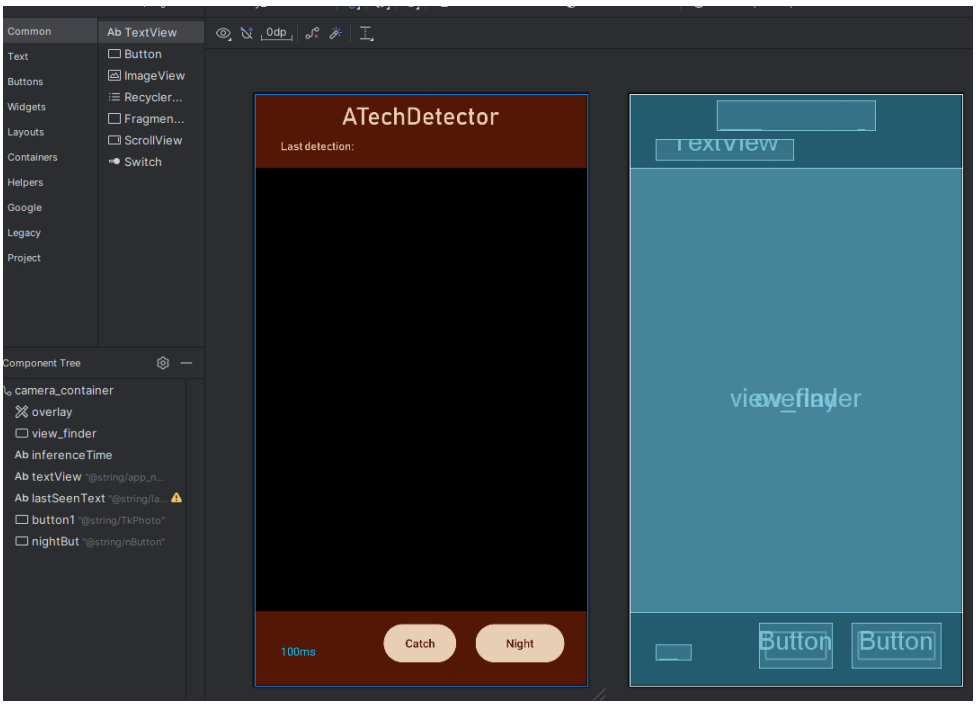

Рисунок 2.8 – інтерфейс програми ATechDetector

Як можна побачити, існує назва застосунку, видошукач, лічильний часу, що витрачено на детекцію об’єкту та 2 кнопки. Додатково є лічильник, що вказує на час який пройшов із останньої детекції. На самому початку роботи застосунку він не відображається.

Зрозуміло, що щоб мати можливість малювати прямокутники на зображенні необхідно визначити його параметри. Для цього створено клас BoundingBox (лістинг 2.1):

Лістинг 2.1 – Властивості класу BoundingBox

```
data class BoundingBox(
val x1: Float,
val x2: Float,
val y1: Float,
val y2: Float,
val cx: Float,
val cy: Float,
val w: Float,
val h: Float,
val cnf: Float,
val cls: Int,
val clsName: String)
```

- x1 – це координата лівого верхнього кута по осі X ;

- x2 – це координата правого нижнього кута по осі X;

- y1 – це координата лівого верхнього кута по осі У;

- y2 – це координата правого нижнього кута по осі Y;

- cx та cy – це координати центру рамки;

- w та h – це ширина та висота рамки відповідно;

- cnf: рівень впевненості моделі в тому;

- cls – це індекс класу. Це 0 для танку та 1 для легкої техніки;

- clsName – це назва класу.

Коли камера захоплює кадр, він передається у метод detect(frame: Bitmap) для обробки (лістинг 2.2). Цей метод підготовлює зображення до входу в нейронну мережу. Змінює розмір до роздільності 1024x1024 – саме на такій роздільній здатності зображень була натренована модель.

Лістинг 2.2 – Масштабування, перетворення у тензор і передача до моделі

```
val resizedBitmap = Bitmap.createScaledBitmap(frame,
tensorWidth, tensorHeight, false)
val tensorImage = TensorImage(DataType.FLOAT32)
tensorImage.load(resizedBitmap)
val processedImage = imageProcessor.process(tensorImage)
val imageBuffer = processedImage.buffer
interpreter?.run(imageBuffer, output.buffer)
```

Після того, як зображення передалось на модель, воно обробляється тензором. Нейронна мережа повертає великий масив чисел – вихідну тензорну карту (FloatArray), яка містить координати об'єктів, їх розміри, ймовірності (confidence), та класифікаційні індекси.

Метод знаходить клас з найвищою впевненістю, обчислює координати центра, ширину та висоту. Далі переводить їх у координати кутів та перевіряє, чи вони знаходяться в межах зображення (нормовані від 0 до 1).

Якщо рівень впевненості перевищує заданий поріг, то створюється об'єкт BoundingBox, який зберігається в списку (лістинг 2.3).

Лістинг 2.3 – Перевірка рівня впевненості та визначення границі прямокутника

```
if (maxConf > CONFIDENCE_THRESHOLD) {
val clsName = labels[maxIdx]
val cx = array[c] // 0
val cy = array[c + numElements] // 1
val w = array[c + numElements * 2]
val h = array[c + numElements * 3]
val x1 = cx - (w/2F)
val y1 = cy - (h/2F)
val x2 = cx + (w/2F)
val y2 = cy + (h/2F)
if (x1 < 0F || x1 > 1F) continue
if (y1 < 0F || y1 > 1F) continue
if (x2 < 0F || x2 > 1F) continue
if (y2 < 0F || y2 > 1F) continue
boundingBoxes.add(
BoundingBox(
x1 = x1, y1 = y1, x2 = x2, y2 = y2,
cx = cx, cy = cy, w = w, h = h,
cnf = maxConf, cls = maxIdx, clsName = clsName))}}
if (boundingBoxes.isEmpty()) return null
return applyNMS(boundingBoxes)
```

У самому кінці використовується функція Non-Maximum Suppression (NMS) Цей метод не слід плутати із самою архітектурою виявлення, тобто з детекторами, заснованими на якір та без нього. Ці архітектури визначають як пропонуються потенційні бокси, коли NMS уточнює ці пропозиції. Після того, як модель виявила об’єкт із необхідною впевненістю, викликається метод onDetect(). Він встановлює лічильник часу обробки та виключає текст, що вказував на минулі детекції (лістинг 2.4).

Лістинг 2.4 – Зникання тексту

```
hasDetectedOnce = true
lastDetectionTime = System.currentTimeMillis()
binding.inferenceTime.text = "${timeToProcess}ms"
binding.lastSeenText.visibility = View.GONE
```

Цей метод також накладає прямокутники на видошукач. На схемі інтерфейсу можна побачити view_finder та overlay. Саме після накладання оверлею користувач бачить обведені об’єкти (лістинг 2.5).

Лістинг 2.5 – Накладання оверлею використовуючи раніше визначені параметри прямокутників

```
val bitmap = detector.getLastProcessedBitmap()
if (bitmap != null) {
val bitmapWithBoxes = bitmap.copy(Bitmap.Config.ARGB_8888, true)
val canvas = Canvas(bitmapWithBoxes)
val boxPaint = Paint().apply {
color = Color.parseColor("#E7CEB5")
strokeWidth = 8f
style = Paint.Style.STROKE}
val textPaint = Paint().apply {
color = Color.WHITE
style = Paint.Style.FILL}

for (box in boundingBoxes) {
val left = (box.x1 * bitmap.width).coerceIn(0f,
bitmap.width.toFloat())
val top = (box.y1 * bitmap.height).coerceIn(0f,
bitmap.height.toFloat())
val right = (box.x2 * bitmap.width).coerceIn(0f,
bitmap.width.toFloat())
val bottom = (box.y2 * bitmap.height).coerceIn(0f,
bitmap.height.toFloat())
canvas.drawRect(left, top, right, bottom, boxPaint)
val boxWidth = right - left
val textSize = boxWidth / 10f
textPaint.textSize = textSize.coerceAtLeast(20f)
val labelText = box.clsName
canvas.drawText(labelText, left, top - 8, textPaint)}
lastBitmapWithBoxes = bitmapWithBoxes}
```

У даному застосунку існує додаткова кнопка – нічний режим. Після активації до моделі ШНМ буде надходити зображення із піднятою гамою до рівня 1.4 (лістинг 2.6). Це значення підвищіть яскравість та може бути корисним при праці вночі [5]. На рисунках 2.8 та 2.9 проведено порівняння знімку із використанням нічного та стандартого режимів.

Лістинг 2.6 – Підвищення яскравості зображення

```
val gamma = 1.4
val gammaCorrection = FloatArray(256) { i ->
 ((255.0 * Math.pow(i / 255.0, 1.0 /
gamma)).toInt()).coerceIn(0, 255).toFloat()}
val result = Bitmap.createBitmap(input.width, input.height,
Bitmap.Config.ARGB_8888)
val canvas = Canvas(result)
```

```
val paint = Paint()
val gammaCorrected = Bitmap.createBitmap(input.width,
input.height, Bitmap.Config.ARGB_8888)
for (x in 0 until input.width) {
for (y in 0 until input.height) {
val pixel = input.getPixel(x, y)
val r = gammaCorrection[Color.red(pixel)].toInt()
val g = gammaCorrection[Color.green(pixel)].toInt()
val b = gammaCorrection[Color.blue(pixel)].toInt()
val a = Color.alpha(pixel)
gammaCorrected.setPixel(x, y, Color.argb(a, r, g, b))}}
canvas.drawBitmap(gammaCorrected, 0f, 0f, paint)
return result}
```

Рисунок 2.8 – Фото у стандартному режимі

Рисунок 2.9 – Фото із підвищеною яскравістю

Важливо також зауважити, що використання такого методу, як підвищення яскравості, може призвести також до зворотного ефекту – коли раніше невизначений піксель асоціювався із певним класом. Програма надає можливість відстежувати час, затрачений на обробку та аналіз зображення.

Рисунок 2.10 – Аналіз фото без апаратного прискорення

Рисунок 2.11 – Аналіз фото із прискоренням

На початку можна помітити слабу продуктивність мобільного пристрою. Проблема полягає в тому, що зображення оброблюється процесором. Час обробки картинки із об’єктами на чотирьохядерному процесорі із використанням від 5 до 6,5 с. Водночас, із використанням прискорення графічним адаптером смартфону Samsung Galaxy S23 на одне зображення приділяється від 120 до 200 мс. Порівняння роботи із прискоренням та без наведено на рисунках 2.10 та 2.11.

3 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Побудова штучної нейронної мережі

Щоб створити штучну нейронну мережу, необхідно підготувати для цього систему. Це може буди локальний комп’ютер або віддалене, серверне рішення. У цій роботі майже вся робота виконувалась на власному комп’ютері.

Усі методи вимагають створення окремого, віртуального, простору. Це дає можливість налагодити сумісній версій, не втручаючись у роботу системи. Створити віртуальне середовище можливо завдяки вбудованій у системи командного терміналу (рисунок 3.1 та 3.2) , або завдяки спеціальним застосункам (рисунок 3.3 та 3.4).

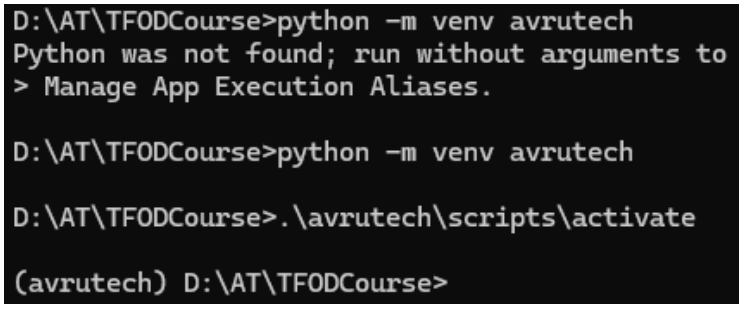

Рисунок 3.1 – Використання командного рядку

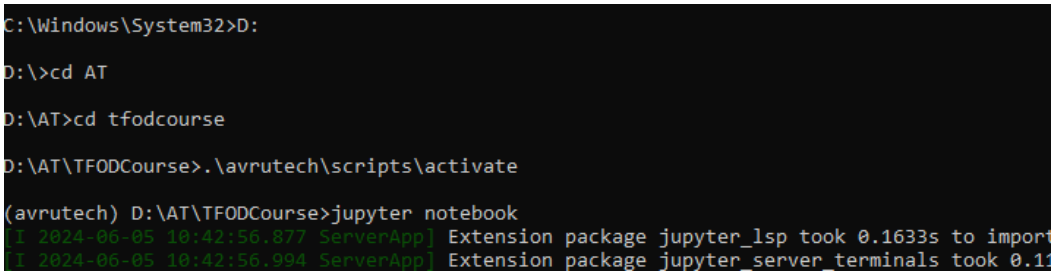

Рисунок 3.2 – Запуск віртуального середовища

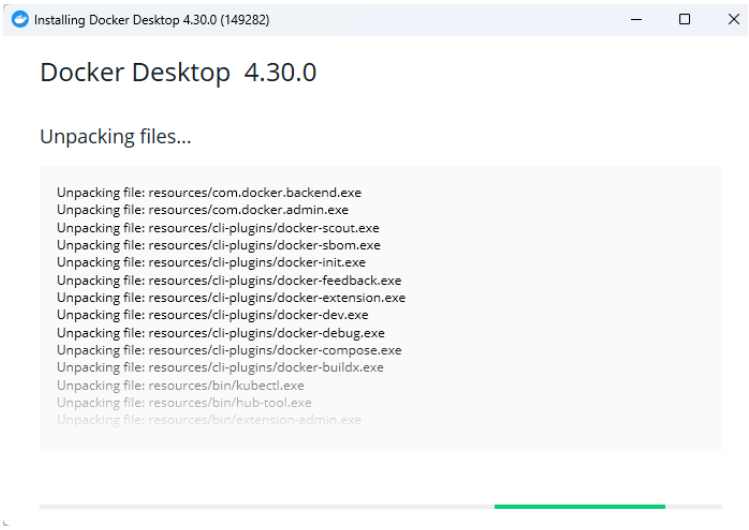

Рисунок 3.3 – Використання Docker для створення віртуального середовища

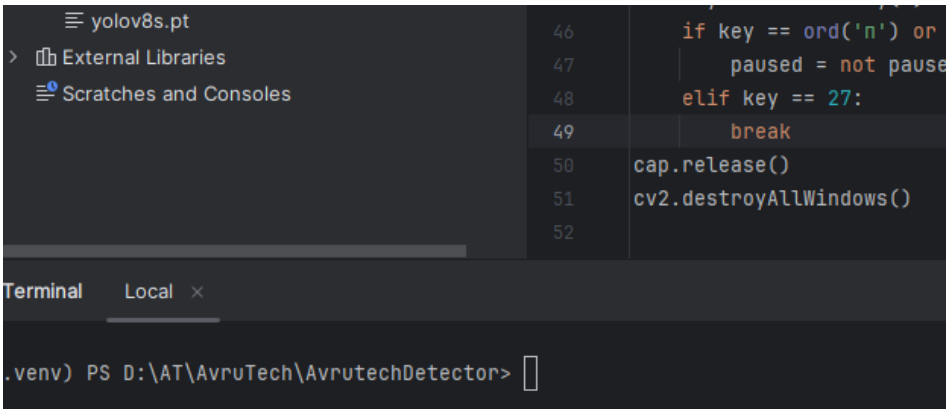

Рисунок 3.4 – Використання PyCharm для створення віртуального середовища

Docker – це потужний інструмент контейнеризації, який дозволяє створювати ізольовані середовища для розробки, тестування та розгортання програм. Він дозволяє створювати віртуальні машини, проте спираючись на ядро хостової системи.

До основних переваг Docker відносять:

- забезпечує запуск контейнерів без значного впливу на швидкодію системи порівняно з нативним виконанням;

- масштабованість та портативність, що дозволяє легко розгортати контейнери на різних платформах, зокрема на локальних серверах, у хмарних середовищах і кластерах Kubernetes;

- ізоляцію процесів і додатків, що забезпечує автономну роботу кожного контейнера без впливу на інші контейнери;

- гнучке управління ресурсами, що забезпечує можливість керувати дисковим простором, CPU та RAM, задаючи визначений обсяг для кожного контейнера.

У контексті створення систем штучного інтелекту та машинного навчання Docker забезпечує відтворюваність середовища незалежно від ОС, що критично важливо при роботі з різними версіями бібліотек, фреймворків (наприклад, TensorFlow, PyTorch). За допомогою Docker-образів можна швидко налаштувати середовище з потрібними інструментами, уникнувши проблем, пов’язаних із несумісністю пакетів.

PyCharm – це інтегроване середовище розробки (IDE) здебільшого для мови Python, яке широко використовується в розробці як звичайних програм, так і проектів в основі яких штучний інтелект.

Воно забезпечує зручний інтерфейс, інтелектуальне автодоповнення коду, інтеграцію з системами контролю версій (Git), а також підтримку віртуальних середовищ. PyCharm дозволяє ефективно організовувати структуру проекту, відлагуджувати код, запускати скрипти та аналізувати результати. Дуже зручним є контроль версій, також підказки під час встановлення компонентів та бібліотек.

Важливо також згадати про використання Google Colab. Це безплатне, хоча і з платними функціями, хмарне середовище для розробки. Воно дозволяє запускати Jupyter-ноутбуки без потреби в локальній установці програм чи бібліотек. Colab надає доступ до обчислювальних ресурсів Google. Також можливо використання локальних GPU, що особливо корисно при навчанні важких нейронних мереж. Також є широка підтримка бібліотек, контроль за версіями. Інтеграція із диском, що надається до акаунту дуже допомагає при перенесенні та резервуванні даних. Прикладом розробки ШНМ можливо побачити на рисунку 3.5

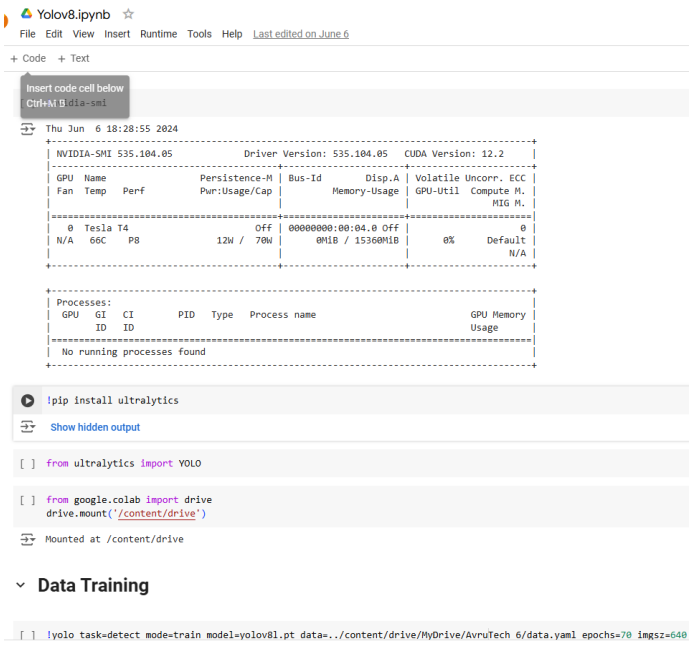

Рисунок 3.5 – Початок роботи на потужностях Google Colab

Тут є можливість навчання моделі та її експорту у необхідний формат (рисунок 3.6). На жаль, час роботи у безоплатному режимі приблизно 5 годин. Цього часу недостатньо для навчання моделей із великою кількістю даних.

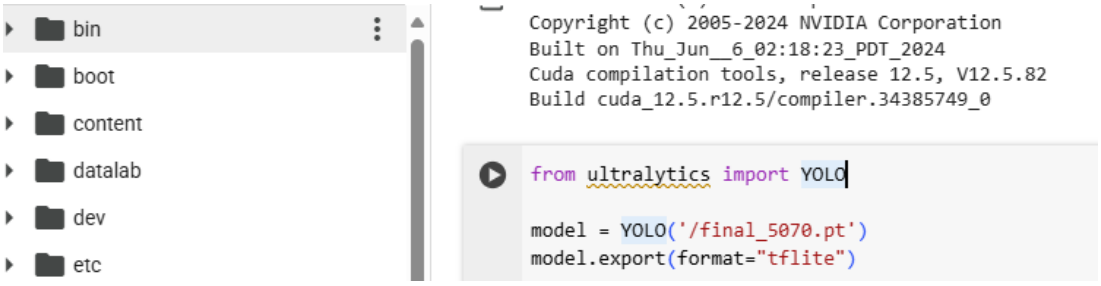

Рисунок 3.6 – Експорт моделі у необхідний формат у Google Colab

3.2 Вибір моделі нейронної мережі

В процесі розробки системи комп’ютерного зору для розпізнавання об’єктів розглянуто декілька сучасних підходів та фреймворків. Першою є TensorFlow, а другою YOLO. Кожна є дуже потужним інструментом при подачі гарно розміченого датасету, налагодженні на застосуванні.

На самому початку була спроба розробки саме на TensorFlow, навіть не через її ефективність, тобто точність та швидкодію, а через розповсюдженість та велику кількість матеріалів.

На жаль, під час роботи виникли критичні складнощі на моїй системі – сумісність версій. Спроба використати командний рядок та jupyter notebook була на початку не складною.

Процес створення віртуального середовища є простим, але під час початку роботи у jupyter notebook не працювало створене ядро. Перевстановлення та перебір різних версій ніяк не вирішували проблему (рисунок 3.7):

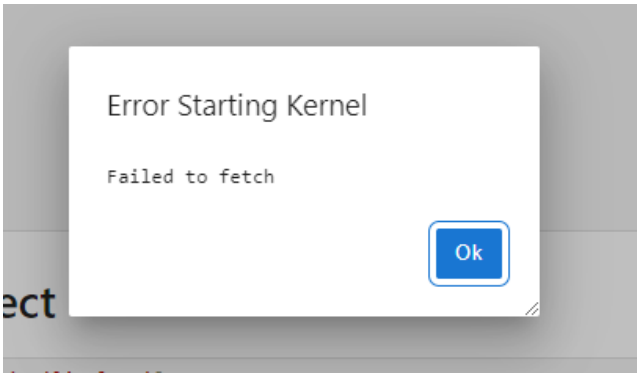

Рисунок 3.7 – Помилка запуску ядра у jupyter notebook

Таким чином реалізація поставленої мети шляхом тренування моделі TensorFlow стала неможливою. Насамперед, це тільки додало цікавості до виконання поставленого завдання.

Далі було встановлено Docker та програму для аннотацій зображень CVAT (рисунок 3.8). Завдяки цим інтрументам можливо одразу у середовищі підготувати модель до тренування та, за необхідністю, провести розмітку зображень.

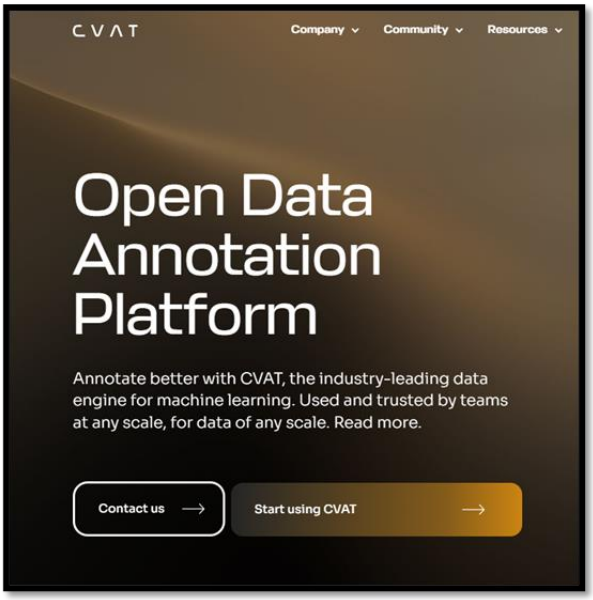

Рисунок 3.8 – CVAT для аннотацій відео та зображень

CVAT дозволяє ефективно розмічати великі набори зображень, підтримує такі типи анотацій як обрамлювальні прямокутники (bounding boxes), полігони, точки, лінії та траєкторії. Однією з ключових переваг є можливість експортувати розмічені дані у формати, сумісні з різними фреймворками глибинного навчання (YOLO, COCO, Pascal VOC тощо). CVAT також підтримує командну роботу, автоматичне попереднє розмічування за допомогою моделей, інтеграцію з Docker і зручне API для автоматизації процесів.

Після встановлення всіх необхідних плагинів, програма почала працювати нестабільно – це супроводжувалось неочікуваним завершенням роботи, зависаннями. Таким чином, обрано варіант розгортання у середі розробки PyCharm, із якою я вже був ознайомлений на минулих курсах.

3.3 Підготовка набору даних

Будь-яка нейронна мережа, штучна чи природна, вимагає навчання на певному наборі даних. Від цих даних буде залежати якість виконання поставлених задач та швидкість, із якою буде виконане завдання[6].

У моделях для обробки зображень важливо надати якомога більше різноманітних варіації об’єктів. Це надає більшу точність та вірогідність знайти більшу кількість об’єктів певного класу[7].

Для підготовки датасету була використана програма LabelImg. Хоча вона й проста та має обмежений інструментарій (рисунок 3.9), проте повністю підходить для цілей дослідження. Завдяки своїй простоті, розміткою датасету може займатися навіть людина без попереднього досвіду. Найголовніше – LabelImg підтримує збереження розмітки у форматі YOLO, що є важливим для підготовки датасету для подальшого навчання нейронної мережі в цьому дослідженні.

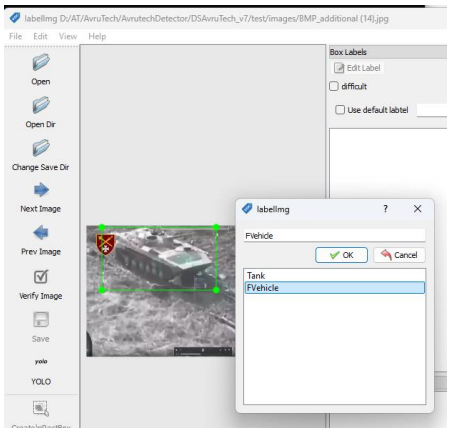

Рисунок 3.9 – Інтерфейс програми LableImg

Для дослідження було обрано два класи об’єктів: танки(тобто важка бронетехніка) та бронемашини (легка техніка). Невелика кількість класів дала можливість зробити більше посилань на об’єкти класів при тій же кількості зображень. Різниця між модифікаями та моделями танків чи автівок не є необхідною, оскільки не завжди можуть бути розпізнані навіть

фахівцями при низькій якості самого вхідного зображення. Головний інтерес – саме реєстрація техніки на екрані пристрою. Людина може не втратити одиницю техніки при зміні зображення або під дією зовнішніх чинників.

У ході реалізації підготовлено 1530 зображень для тренування із різною кількістю посилань на об’єкти, 273 зображення для валідації та 71 до тестування.

Навчання моделі це довготривалий процес. На рисунку 3.10 представлено час тренування моделі із загальною кількістю зображень у 1900 штук та 80 епохами:

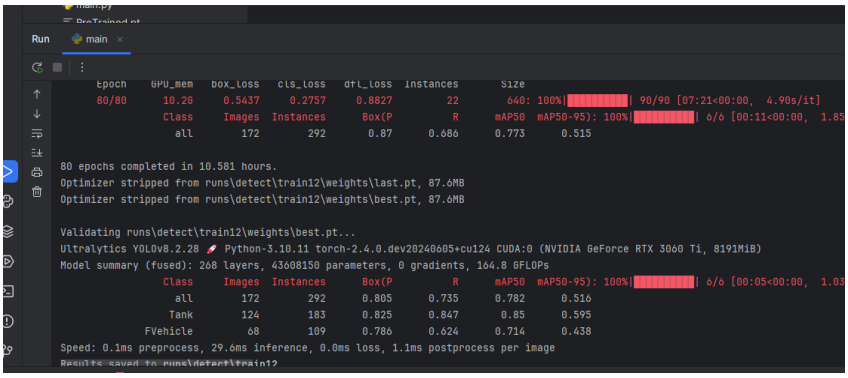

Рисунок 3.10 – Кінець навчання моделі

Як можна побачити, це займає майже 11 годин із використанням прискорення графічним адаптером.

Далі необхідно перевірити – чи модель дійсно працює та оцінити її спроможність виконання завдання детекції класів техніки. На рисунках 3.11 та 3.12 представлено приклади роботи:

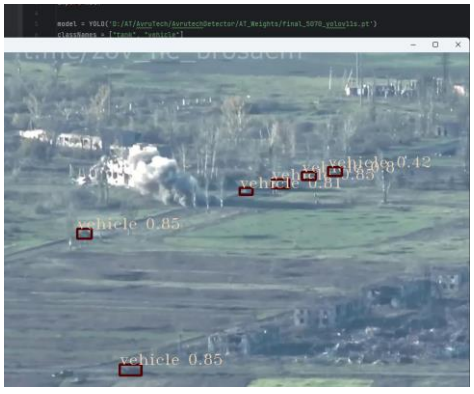

Рисунок 3.11 – Тестування ШНМ

Рисунок 3.12 – Тестування ШНМ

Таким чином доведено, що нейронна мережа успішно закінчила своє навчання. Вона вже зараз здатна відзначати об’єкти класів та зображенні. Далі необхідно підготувати її до переносу та реалізації на потужностях смартфону.

Для оцінки якості навчання існує спецальна метрика[8].

- Images – метрика, що показує кількість зображень у валідаційному наборі, які містять клас об'єктів;

- Instances – показує, скільки разів клас з'являється на всіх зображеннях у валідаційному наборі;

- Instances – показує, скільки разів клас з'являється на всіх зображеннях у валідаційному наборі.

Потім йде набір метрик Box, він дає уявлення про ефективність моделі у виявленні об'єктів, рогзлянемо детальніше кожний елемент:

- P (Precision) – це точність виявлених об'єктів, яка показує, скільки об'єктів було виявлено правильно;

- R (Recall) – це здатність моделі ідентифікувати всі екземпляри об'єктів на зображеннях;

- mAP50 – це середня точність, розрахована при порозі перетину над об'єднанням (IoU) 0,50. Це міра точності моделі, що враховує лише «легкі» виявлення;

- mAP50-95 – середнє значення mAP, розрахованої при різних порогах IoU, в діапазоні від 0,50 до 0,95. Що дає комплексне уявлення про роботу моделі на різних рівнях складності виявлення.

На цьому зображенні можна побачити результат навчання моделі:

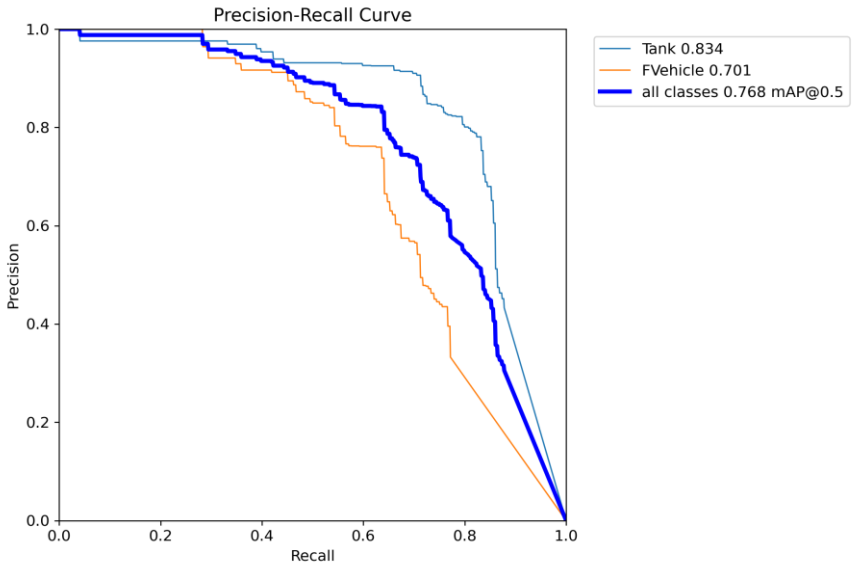

Рисунок 3.13 – Результат навчання моделі.

Precision (точність) та Recall (повнота) це дві важливі метрики для оцінки якості моделей машинного навчання, особливо в задачах класифікації та детекції об’єктів.

Точність це частка правильно передбачених позитивних випадків серед усіх передбачених як позитивні. Повнота – це частка правильно передбачених позитивних випадків серед усіх фактичних позитивних прикладів.

3.4 Моделі YOLO

YOLO дуже швидко розвивається. На початку роботи була доступна 10-та версія моделі, а зараз у відкритому доступі вже з’явилася 11-та. Незважаючи на наявність новіших версій, під час старту розробки було обрано саме восьму – вона не поступається новим моделям, а в умовах

недостатнього освітлення може навіть перевершувати їх. Проте головною причиною такого вибору стала сумісність: на жаль, на локальному комп'ютері виникали проблеми з останніми версіями. Тому найкращим рішенням було використання саме старішої, стабільної моделі.

Наразі вже доступна версія 11. Розробники усунули проблеми із сумісністю, тому прийнято рішення про перехід із YOLOv8 на YOLOv11. Це також дозволить об'єктивно порівняти обидві моделі, адже вони будуть навчені за однаковими параметрами:

- 120 епох;
- роздільна здатність зображень 1024;
- єдиний датасет.

На рисунку 3.14 зверху знаходиться модель 11 версії, а знизу – 8. Таким чином можна порівняти ці моделі на реальних прикладах:

Рисунок 3.14 – порівняння двох моделей

На лівому наборі можна побачити, що восьма модель втратила одну одиницю техніки при високім рівні впевненості для знайдених, коли 11 знайшла все що було на зображенні. На правому – модель 11 знайшла всі одиниці впоралась із завданням, коли минула версія зробила багато помилок.

3.5 Мобільний застосунок

Для реалізації мобільного застосунку, що використовує модель глибинного навчання, обрано середовище Android Studio як основний інструмент розробки. Це інтегроване середовище розробки (IDE) від Google. Воно було створене для створення застосунків під операційну систему Android. Забезпечує потужний функціонал для розробки, відлагодження та тестування. Підтримує такі сучасних мов програмування, таких як Java та Kotlin. Із цією програмою я був ознайомлений під час занять із дисципліни системні інтерфейси та інтерфейси користувача.

Також під час навчання я зіткнувся із Flutter. Це середовище розробки для створення додатків, які можна запускати на Android та IOS. Дуже гарний інструмент. Ключова ідея фреймворку Flutter полягає в тому, аби кросплатформовий додаток працював так само швидко, як і нативний (тобто, створений спеціально під окрему ОС). Проте, Android Studio, натомість, є «рідним» середовищем для Android, надає широкі можливості для оптимізації та підтримує всі сучасні інструменти для роботи з TensorFlow Lite.

Для написання самого застосунку була обрана мова Kotlin, яка розробляється компанією JetBrains. Останніми роками стала основною рекомендованою Google мовою для розробки під Android. Має не дуже складний синтаксис та багато вказівних матеріалів.

Одним з ключових етапів у реалізації проекту стало впровадження моделі глибинного навчання у мобільний застосунок. Оскільки повноцінні TensorFlow або PyTorch-моделі є «важкими» для мобільних пристроїв, використовується TensorFlow Lite (TFLite) – полегшений формат моделей, який оптимізовано для мобільного та вбудованого застосування. Дуже допомагає можливість прискорення обчислень шляхом використання прискорення графічним адаптером телефону.

У новітніх версіях Android, починаючи з Android 10, API level 29 та вище, політика доступу до ресурсів пристрою, зокрема камери, стала значно суворішою. Будь-яке звернення до камери вимагає попереднього отримання дозволу, який має бути явно запитаний у користувача під час виконання програми.

Крім того, при використанні сторонніх бібліотек або власноручного доступу до CameraX чи Camera2 API, розробник повинен правильно обробити сценарії, коли дозвіл не надано – наприклад, через відмову користувача або політику пристрою. У деяких випадках, особливо на Android 12 і вище, система також повідомляє користувача про використання камери через відповідну іконку у статус-барі, підвищуючи прозорість. На даному рисунку представлено фрагмент коду, що відповідає за запит у користувача на надання дозволу до камери:

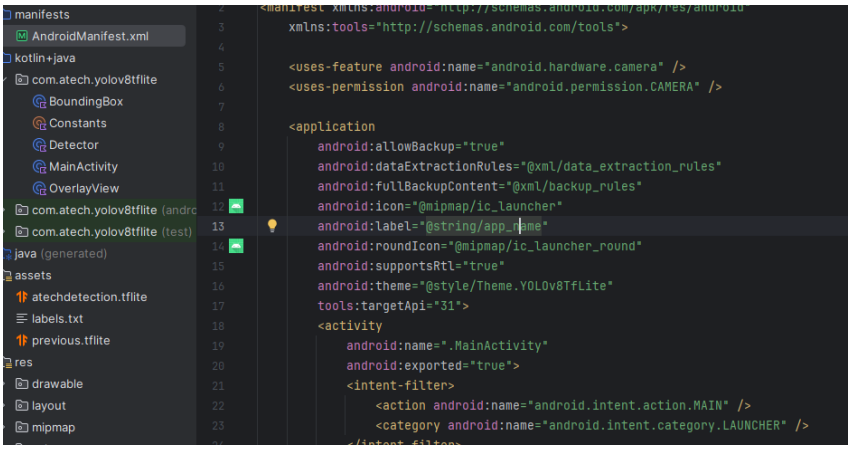

Рисунок 3.15 – Надання доступу до камери

На попередньому етапі було отримано модель YOLO для задачі детекції об’єктів. Однак для запуску YOLO на мобільному пристрої необхідно попередньо експортувати модель у формат TFLite. Це неможливо зробити напряму. Цей процес передбачає перетворення моделі з початкового формату із PyTorch до ONNX, потім у TensorFlow із подальшою конвертацію в TFLite. Дуже добре, що розробник моделі YOLO – Ultralytics, передбачає конвертацію моделей та надає вбудовану функцію для цього.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи створено мобільний застосунок, що дозволяє знаходити військову техніку та зберігати її зображення.

Після тренування моделей проведено дослідження, де модель 11 версії є більш точнішою та має потенціал для подальшого покращення. Після тренування моделей можливо отримати графіки із статистикою за кожним класом та рівнем впевненості при детекції.

Лише реальне користування на тестування на нових прикладах може показати кращу модель. На початку роботи існували моделі, де кількість класів була більшою – модифікації танків та тип легкої техніки. Це вимагає значно більших наборів даних, що водночас вимагатиме використання ресурсів більших, ніж існує у локальної машини при навчанні.

Прискорення графічним адаптером дозволило збільшити кількість проаналізованих зображень у більше, ніж 20 разів.

Застосунок має потенціал до покращення. Наступними кроками можуть бути:

- поява авторизації користувача;
- оптимізація та покращення детекцій нейромережі;
- можливість обрання зображення з пам’яті пристрою для обробки;
- відправка оброблених зображень на сервер;
- інтеграція із застосунками безпілотних літальних апаратів;
- захист програми та самої мережі від несанкціонованого втручання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Application of SOFA framework for physics-based simulation of deformable human anatomy of nasal cavity / Maksym Tymkovych et. al// IFMBE Proceedings (2021). DOI: 10.1007/978-3-030-64610-3_14

2. V. Diachenko, O. Mikhal. Perspectives of application of the advanced classical algorithm of Kohonen maps in distributed energy-critical sensor networks. Control, Navigation and Communication Systems, vol.4, 2023. P.75-79. https://doi.org/10.26906/SUNZ.2023.4.075

3. SURF[https://prometheus.org.ua/course/course v1:IRF+ML101+2016_T3SURF.

4. Novoseltsev IV Siamese Neural Networks / IV Novoseltsev., N.G. Aksak

5. Аврунін О.О., О. Ю. Барковська Оцінка впливу методів препроцесінгу на точність пошуку обєктів на зображеннях міського середовища. Т. 3 : секція 3,4 / Нац. ун-т оборони Азербайджанської Республіки [та ін.]. – Харків : Impress., 2025. – С. 23.

6. M: Information Processing Systems, 2007. - Vip. 3. - 64 c.

7. Flach P. A. Machine Learning: The Art and Science of Algoritms that Makes Sense of Data. Cambridge: Cambridge University Press, 2012. 291 p. https://doi.org/10.1017/CBO9780511973000

8. Pratt W. Neural Networks / W. Pratt - M .: Mir, 1982. - 480 c.