МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____»_____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

ІНТЕЛЕКТУАЛЬНА СИСТЕМА КОМП'ЮТЕРНОГО ЗОРУ ДЛЯ РОЗПІЗНАВАННЯ ТА ВИЯВЛЕННЯ НАЗЕМНИХ МІН ПФМ-1

Спеціальність 122 «Комп'ютерні науки»

122 – КРМ – 601.21810121

Виконав студент 6-го курсу, групи 601
_____М. О. Салютін
«20» __лютого__ 2024 р.

Керівник: канд. техн. наук, доцент
_____Є. В. Сіденко
«20»__лютого__ 2024 р.

Миколаїв – 2024

Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Освітньо-кваліфікаційний рівень **магістр**
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____20__р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

Салютіну Максиму Олександровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи магістра
«Інтелектуальна система комп'ютерного зору для розпізнавання та виявлення наземних мін ПФМ-1».
Керівник роботи Сіденко Євген Вікторович, канд. техн. наук, доцент.
Затв. наказом Ректора ЧНУ ім. Петра Могили від «20» жовтня 2023 р. № 201
2. Строк подання студентом роботи «20» лютого 2024 р.
3. Вхідні (початкові) дані до роботи: набір даних у вигляді зображень мін ПФМ-1, архітектури нейронних мереж CNN та YOLOv8 для вирішення задачі розпізнавання та виявлення. Очікуваний результат: створена інтелектуальна система комп'ютерного зору для розпізнавання та виявлення наземних мін ПФМ-1.
4. Зміст пояснювальної записки (перелік питань, які потрібно розглянути):

- аналіз сучасного стану задачі розпізнавання об'єктів, зокрема наземних мін ПФМ-1;
- дослідження та аналіз готових рішень, існуючих технологій;

– огляд архітектур нейронних мереж CNN, YOLOv8: історія виникнення, основні принципи роботи, опис архітектур;

– навчання та створення моделей нейронної мережі, імпорт моделі в систему ОС Android.

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини: «Забезпечення вимог охорони праці у офісному приміщенні, роботи за комп'ютером та у надзвичайних ситуаціях»

7. Консультанти:

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	**Доктор біологічних наук, професор Григор'єва Л.І.**	
Методична частина	**Канд техн. наук, доцент Сіденко Є. В.**	

Керівник роботи __канд техн. наук, доцент Сіденко Є. В._____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ___Салютін М. О._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «31» ____жовтня____ 2023 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи магістра

Тема : Інтелектуальна система комп'ютерного зору для розпізнавання та виявлення наземних мін ПФМ-1

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників КРМ	15.10.2023	15.10.2023	Виконано
2	Отримання завдання на виконання КРМ	11.10.2023	01.11.2023	Виконано
3	Складання календарного плану роботи на період виконання КРМ	08.11.2023	09.11.2023	Виконано
4	Отримання завдання на передатестаційну практику, огляд матеріалу для КРМ	24.11.2023	24.11.2023	Виконано
5	Проходження передатестаційної практики, збір та аналіз матеріалів до КРМ	27.11.2023	24.12.2023	Виконано
6	Опис фахової частини КРМ, аналіз сучасного стану задачі розпізнавання об’єктів, огляд архітектур нейронних мереж для поставленої задачі, реалізація системи	28.10.2023	05.01.2024	Виконано
7	Розробка спеціальної частини з охорони праці та надзвичайних ситуацій, методичної частини	30.10.2023	12.12.2023	Виконано
8	Перший попередній захист КРМ на засіданні комісії кафедри	29.01.2024	29.01.2024	Виконано
9	Другий попередній захист КРМ на засіданні комісії кафедри	12.02.2024	12.02.2024	
10	Коригування роботи за результатами попереднього захисту	12.02.2024	13.02.2024	Виконано
11	Коригування та остаточне оформлення КРМ	13.02.2024	14.02.2024	Виконано
12	Подання КРМ рецензенту	14.02.2024	14.02.2024	Виконано
13	Рецензування КРМ	14.02.2024	15.02.2024	Виконано
14	Подання БКР, її електронної копії та інших документів (відгуку, рецензії) до захисту	18.02.2024	18.02.2024	Виконано
15	Захист КРМ перед екзаменаційною комісією (ЕК)	26.02.2024	26.02.2024	Виконано

Розробив студент _____Салютін М. О_____ _____
(прізвище та ініціали) (*підпис*)

Керівник роботи _канд техн. наук, доцент Сіденко Є. В._ _____
(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

«09» листопада 2023 р.

АНОТАЦІЯ

до кваліфікаційної роботи магістра
студента 601 групи ЧНУ ім. Петра Могили

Салютіна Максима Олександровича

на тему**: «ІНТЕЛЕКТУАЛЬНА СИСТЕМА КОМП'ЮТЕРНОГО ЗОРУ ДЛЯ РОЗПІЗНАВАННЯ ТА ВИЯВЛЕННЯ НАЗЕМНИХ МІН ПФМ-1»**

Актуальність дослідження полягає в створенні системи для розпізнавання та виявлення наземних мін ПФМ-1 для безпеки працівників ДСНС, військових та цивільних. Подібні програми вимагають високої точності розпізнавання, щоб відрізнити міну ПФМ-1 від звичайного листя або каміння, помилка повинна бути мінімізована.

Об’єктом дослідження є процеси розпізнавання та виявлення наземних мін з використанням штучного інтелекту.

Предметом дослідження є архітектури нейронних мереж для розпізнавання та виявлення наземних мін.

Метою дослідження є розпізнавання і виявлення наземних мін за допомогою мобільних застосунків та з використанням різних архітектур нейронних мереж.

Робота складається з фахового розділу, спеціальної частини з охорони праці та надзвичайних ситуацій, та методичної частини. Пояснювальна записка складається зі вступу, чотирьох розділів та висновків. Кожен розділ присвячено: аналіз всесвітньої проблеми, викликану через воєнні конфлікти, аналіз наявних аналогів систем розпізнавання, постановка задачі, вибір сучасних архітектур нейронних мереж для вирішення задачі розпізнавання наземних мін ПФМ-1, їх порівняння, моделюванню системи та імпорт моделі на ОС Android.

Кваліфікаційна робота магістра містить 142 сторінки, 92 рисунки, 2 таблиці, 55 літературних джерел.

Ключові слова: розпізнавання, комп’ютерний зір, нейронні мережі, набір даних, CNN, YOLOv8, Python.

ABSTRACT

to the master's qualification work by the student of the group 601 of Petro Mohyla Black Sea National University

Saliutin Maksym

«INTELLIGENT COMPUTER VISION SYSTEM FOR RECOGNITION AND DETECTION OF LANDMINES PFM-1»

The relevance of the study is to create a system for recognizing and detecting PFM-1 landmines for the safety of rescue workers, military and civilians. Such applications require high recognition accuracy to distinguish a PFM-1 mine from ordinary leaves or stones, and the error must be minimized.

The object of research is the processes of recognition and detection of PFM-1 landmines.

The subject of the study is the methods of training a neural network that will allow recognizing and detecting PFM-1 mines.

The aim of the work is to create a system for recognizing PFM-1 mines from a smartphone camera using different neural network architectures.

The work consists of a professional section, a special part on labor protection and emergencies, and a methodological part. The explanatory note consists of an introduction, five chapters, and conclusions. Each chapter is devoted to: analysis of the global problem caused by military conflicts, analysis of existing analogues of recognition systems, problem statement, selection of modern neural network architectures for solving the problem of PFM-1 landmine recognition, their comparison, system modeling and importing the model to Android.

The master's qualification work contains 142 pages, 92 figures, 3 tables, 55 references.

Keywords: recognition, computer vision, neural networks, data set, CNN, YOLOv8, Python.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

НМ – нейронна мережа
ОС – операційна система
ПЗ – програмне забезпечення
ПФМ-1 – протипіхотна фугасна міна
ШІ – штучний інтелект
ШНМ – штучна нейронна мережа

CNN – convolutional neural network
NNAPI – neural networks application-programming interface
R-CNN – region based convolutional neural network
YOLO – you only look once

ВСТУП

Розпізнавання та виявлення наземних мін типу ПФМ-1 життєво важливі у сучасному світі. Історія показала, що військові конфлікти залишають після себе небезпечні пристрої в вигляді наземних мін будь-якого типу (протипіхотні, протитанкові та «пелюстки»). Міни як знаряддя для зруйнування завдає значних збитків людям та інфраструктурі, тому усунення цієї загрози для цивільних та військових підрозділів стало однією із найбільших глобальних проблем. Даний тип міни може висіти на деревах, бути у кущах та просто у землі. Рішення цієї проблеми вимагає використання сучасних методів та технологій, які можуть допомогти безпечно та ефективно розпізнавати та виявляти міни типу ПФМ-1. Таким чином дана тема є актуальною.

Об’єктом дослідження є процеси розпізнавання та виявлення наземних мін з використанням штучного інтелекту.

Предметом дослідження є архітектури нейронних мереж для розпізнавання та виявлення наземних мін.

Метою роботи є розпізнавання і виявлення наземних мін за допомогою мобільних застосунків та з використанням різних архітектур нейронних мереж.

Для досягнення цієї мети необхідно створити власний набір даних, на якому буде навчатися декілька моделей нейронних мереж, після чого буде обрано найкращу серед усіх.

Для вирішення проблеми потрібно виконати наступні задачі:

1) проаналізувати сучасні технології для вирішення задачі розпізнавання об’єктів у складних умовах, зокрема наземних мін ПФМ-1;

2) дослідити існуючі матеріали, методи та технології для розпізнавання наземних мін ПФМ-1;

3) створити власний набір даних, навчити на них нейронні мережі та імпортувати їх у Android-застосунок;

4) за допомогою тестування порівняти отримані моделі нейронних мереж для розпізнавання мін ПФМ-1.

У першому розділі розглядується сучасний стан задачі розпізнавання та виявлення наземних мін ПФМ-1, огляд наявних рішень даної проблеми, публікацій та визначено задачі, які необхідно виконати.

Другий розділ містить у собі огляд існуючих технологій та алгоритмів, проведено аналіз моделей для поставленої задачі.

У третьому розділі описується підхід до створення власного набору даних, їх анотація, навчання нейронних мереж, їх порівняння між собою та імпорт отриманої моделі у Android-застосунок.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ВИЗНАЧЕННЯ АКТУАЛЬНОСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної області та актуальність

Розпізнавання та виявлення наземних мін типу ПФМ-1 життєво важливі у сучасному світі. Історія показала, що військові конфлікти залишають після себе небезпечні пристрої в вигляді наземних мін будь-якого типу (протипіхотні, протитанкові та «пелюстки»). Міни як знаряддя для зруйнування завдає значних збитків людям та інфраструктурі, тому усунення цієї загрози для цивільних та військових підрозділів стало однією із найбільших глобальних проблем. Даний тип міни може висіти на деревах, бути у кущах та просто у землі, чи під нею. Рішення цієї проблеми вимагає використання сучасних методів та технологій, які можуть допомогти безпечно та ефективно розпізнавати та виявляти міни типу ПФМ-1 як цивільним, робітникам ДСНС та військовим [1].

Дана тема стала дуже важлива в сучасних конфліктах та питаннях безпеки населення, тому дослідження використання навченої моделі нейронної мережі та технологій комп’ютерного зору для точної ідентифікації наземних мін ПФМ-1, які можуть бути знайдені у будь-якої місцевості є актуальною. Створення та покращення подібних систем дозволить врятувати багато життів та допомогти розчистити території від небезпечної та ледве помітної для ока людини міни.

Виявлення наземних мін ПФМ-1 за допомогою комп’ютерного зору привернуло увагу розробників та вчених, тому що сучасні методи боротьби із подобною зброєю дуже складна – ці пристрої становлять особливо складну проблему для звичайних методів виявлення наземних мін, таких як металошукач, оскільки міни в основному складаються з пластику та важать у середньому лише 75 грам [1].

Існує декілька методів та технологій розпізнавання вибухонебезпечних пристроїв, які поєднані у комбінації апаратного та програмного забезпечення.

Наприклад магнітні резонансні технології, які базуються на здатності деяких мін до взаємодії з магнітним полем, які можуть допомогти виявити феромагнітні властивості пристроїв. Також існує технологія інфрачервоний спектр, який дозволяє виявити хімічні речовини, які також можуть бути частиною вибухонебезпечного пристрою. Самий поширений спосіб – це сучасні металошукачі, які можуть дуже швидко реагувати на металеві предмети, які можуть бути закопані у землі або піску.

Але усі ці підходи не такі функціональні, або у деяких випадках можуть не спрацювати, як наприклад у випадку із міною ПФМ-1, так як її матеріал складається із пластику, тому металошукач не спрацює на дану міну. Тому ознайомившись із сучасними технологіями у допомогу до таких пристроїв використовують нейронні мережі, які можуть класифікувати та виявити об’єкт у реальному часі використовуючи відеопотік або вхідне зображення [1].

Розпізнавання зображень – одне із завдань, у якому глибокі нейронні мережі (ГНМ) досягають великих успіхів. При розробці архітектури мережі автори знайшли натхнення у будові людського мозку, звідси й назва. Вони складаються з трьох типів шарів: вхідного, прихованих шарів і вихідного (результативний). На вхідний шар надходить сигнал, прихований шар обробляє його, а вихідний шар ухвалює рішення або робить прогноз щодо вхідних даних у форматі самих даних, процентної точності або зображення. Кожен шар мережі складається із взаємопов'язаних вузлів (штучних нейронів), які виконують обчислення вхідних даних, створюють ваги та завдяки цьому аналізують інформацію.

Популярний варіант для вирішення задачі класифікації та розпізнавання є архітектура нейронної мережі CNN (Convolutional Neural Network) [2].

CNN належить до класу нейронних мереж глибокого навчання. CNN являють собою величезний прорив у галузі розпізнавання образів. Вони найчастіше використовуються для аналізу візуальних образів і часто працюють за лаштунками класифікації зображень. Вони лежать в основі найрізноманітніших систем – від

маркування фотографій у соціальних мережах до автомобілів на автоматичному керуванні. Їх активно використовують у всіх сферах – від охорони здоров'я до безпеки громадян. На даний момент розробки подібних моделей існує декілька інших видів згорткових нейронних мереж – R-CNN та Faster/Fast-CNN. R-CNN (Regions with CNN) використовує алгоритм вибіркового пошуку, щоб запропонувати області інтересу на зображенні, а потім використовує CNN для класифікації об’єкту на вхідному зображенні.

Fast R-CNN покращує R-CNN за рахунок використання спільного шару згортки для обробки всього зображення, замість того, щоб обробляти кожну область інтересу на зображенні незалежно. Це значно скорочує обчислювальний час, необхідний для виявлення об'єктів на вхідному зображенні та знижує навантаження на систему під час обробки.

Faster R-CNN це покращена версія архітектури Fast R-CNN, дана архітектура використовує для генерації області інтересу на зображенні мережу регіональних пропозицій яка набагато швидша за алгоритм вибіркового пошуку, що використовується в R-CNN і Fast R-CNN, але модель Faster R-CNN справляється трохи гірше з локалізацією, тому вона підходить до менших наборів даних. Мережа використовує карти особливостей CNN для прогнозування меж об'єктів. Серед особливостей даної мережі – вона може розпізнавати об’єкт в кожній позиції, що поширює функціональність нейронної мережі в декілька разів.

Також існує архітектура SSD, яка дозволяє створити модель для виявлення об’єктів за допомогою єдиної глибокої нейронної мережі, яка поєднує регіональні пропозиції та вилучення ознак. Зазвичай використовується мережа ResNet, яка була навчена на ImageNet і позбавлена останнього повністю підключеного шару класифікації.

Це створює глибоку нейронну мережу, здатну виділяти семантичне значення з вхідного зображення, зберігаючи просторову структуру зображення при меншій роздільній здатності. Якщо дивитися на її оптимізацію, то для зображень розміром

300 на 300 пікселів вона має дуже високу швидкість обчислень та на сучасній машині видає 59 FPS.

На даний момент нейронні мережі тісно пов’язані із вирішенням сучасних проблем та задач. Використання новітніх архітектур дозволяє розробникам оптимізувати та автоматизувати будь-які процеси та знайти рішення проблем, які неможливо вирішити іншим чином.

1.2 Аналіз аналогічних систем та публікацій

При виконанні кваліфікаційної роботи було проаналізовано схожі системи та публікації які вирішують питання розпізнавання вибухонебезпечних об’єктів використовуючи вхідне зображення або відеопотік.

Перша публікація під назвою «Застосування глибокого навчання для автоматизації виявлення розкиданих наземних мін на основі БПЛА» розглядається оптимізація використання БПЛА з легкими багатоспектральними і тепловими інфрачервоними датчиками, які дозволяють швидко виявляти забруднення наземними мінами на великій території та проводити дослідження на відкритих площах. Головна ідея це запускати БПЛА і на аналізі мультиспектральних і теплових наборів даних, зібраних автоматизованою системою зйомки, що містить розсіяні наземні міни типу PFM-1 [1].

Серед матеріалів для навчання було створено набори даних у полі з рідкою рослинністю в парку штату Ченанго Веллі, щоб представити середовище пустелі та рідкої рослинності у трьох варіаціях – без рослин, з рослинами та поле покрите снігом.

У саму систему БПЛА завантажено модель нейронної мережі Faster R-CNN, яка показала точність тестування 99,3% для частково забороненого набору тестування та 71,5% точності тестування для повністю вилученого набору тестування.

Перший раз навчальні дані складалися з 165 RGB-зображень, отриманих із різних культур шести фото. Фото складалися з трьох прольотів над тим самим щебінковим середовищем розміром 10x20 метрів і трьох прольотів над тим самим трав'яним середовищем розміром 10x20 метрів. Дані про траву та щебінь були зібрані восени 2019 року та містять 28 мін PFM-1, чотири гільзи KSF і два ковпаки KSF, розкидані по всьому полю. Усе навчання та тестування проводилися на процесорі Intel Xeon Silver 4114 на 2,20 ГГц із 128 ГБ оперативної пам'яті та графічним процесором Titan із 12 ГБ оперативної пам'яті, що дозволило розробникам створити систему, яка може працювати та обробляти вхідну інформацію у режимі реального часу.

Друга публікація під назвою «Впровадження підходу штучного інтелекту до систем георадара для виявлення наземних мін» також розглядається виявлення мін, але із використанням вже готової системи георадару автори використовували моделі нейронних мереж для покращення створеного георадару, а саме Fast CNN та RNN. Найкращою моделлю при пошуку вибухонебезпечних пристроїв себе показав Fast CNN [2].

Третя публікація «Система виявлення морських мін з використанням моделей глибокого навчання YOLO, SSD та EfficientDet» порівнює між собою наступні нейронні мережі та їх модифікації для розпізнавання морських мін:

1) Fast R-CNN з точністю 70%;
2) Faster R-CNN з точністю 78.80%;
3) R-CNN з точністю 82.00%;
4) SSD з точністю 78.60%;
5) YOLOv5 з точністю 78.85%.

Для виявлення плаваючих мін використовувалися дві мережі, починаючи з моделей YOLOv5 і SSD, відомих своєю ефективністю і точністю. З огляду на поточну ситуацію, очевидно, що алгоритм YOLOv5 показав кращі результати, досягнувши кращої точності за набагато коротший час навчання. Модель SSD

відома високими обчислювальними витратами, але за допомогою можливих оптимізацій результати можуть покращитися. З іншого боку, для мін, як YOLOv5, так і EfficientDet досягли дуже хороших результатів за надзвичайно короткий час навчання [3].

У четвертій публікації під назвою «Виявлення та класифікація протипіхотних мін за даними георадару за допомогою швидшого R-CNN» автори представляють рішення для автоматичного виявлення та класифікації похованих об’єктів шляхом впровадження згорткової нейронної мережі Faster Region (Faster R-CNN) із системою георадара (GPR). Зокрема, було обрано Faster R-CNN Inception-v2 як компроміс між обчислювальним навантаженням і точністю порівняно з іншими Faster R-CNN. Набір даних зображень, який використовується для навчання та тестування мережі R-CNN, складається з B-сканів георадара, отриманих як за допомогою моделювання на основі gprMax, так і з реальних виміряних даних георадара. Ефективність методу оцінюється за допомогою матриць плутанини та кривих ROC. Підхід до обробки, заснований на розмірі об’єкта та глибині під поверхнею землі, дозволяє розрізняти протибійні міни [4].

У п’ятій публікації «Виявлення об'єктів на основі алгоритму Faster R-CNN з об'єднанням пропусків та злиттям контекстної інформації» автори досліджують питання поліпшення нейронної мережі Faster R-CNN, з об'єднанням пропусків і злиттям контекстної інформації. Цей алгоритм може покращити ефективність виявлення в особливих умовах на основі Faster R-CNN за допомогою трьох етапів, а саме вилучення контекстної інформації після згорткового шару, об’єднання пропусків для того, щоб перша частина могла отримати повну інформацію про наданий об’єкт на зображенні, та третя частина замінює змінює мережу регіональних пропозицій на більш ефективну GA-RPN, яка підвищує якість та ефективність виявлення об’єктів на вхідному зображенні. Середнє покращення даної моделі складає 6,857% за індексом середньої точності (у порівнянні із YOLO v3 та SSD512). Даний підхід може покращити роботу нейронної мережі у

реальному часі та спростити імпорт моделі на інші більш слабкі платформи, такі як мобільні пристрої на базі Android [5].

Шоста публікація «Алгоритми виявлення об'єктів на основі глибокого навчання на зображеннях і відео» авторами якої є Прасанта Дас та Ангшуман Чакраборті провели дослідження, яке призвело до зміни напрямку або підходу до автоматизованого виявлення об'єктів на зображеннях і відео. Деякі проблеми у виявленні відео роблять їх складними. Рухливість об'єктів і складність умов низької освітленості роблять виявлення об'єктів дуже важливою сферою досліджень. Багато дослідників запропонували кілька підходів для виявлення об'єктів на зображеннях і відео. Для цього експерименту були використані алгоритми YOLOv7 та YOLOv8 з набором даних яблук, які досягли середньої точності 94.2% та 91.2% відповідно. При цьому точність впізнавання становить 98%, 93.8% для YOLOV7 та 98%, 93.9% для YOLOv8 [6].

Головною особливістю архітектури та роботи YOLOv8 було розглянуто у сьомій публікації під назвою «Ефективність виявлення об'єктів у реальному часі моделей YOLOv8 для безпілотних автомобілів у змішаному дорожньому середовищі», авторами якої є Афдхал Афдхал та Хайрун Саддамі. Головна мета дослідження полягала у вирішенні проблеми у розробці безпечних та ефективних самокерованих автомобілів є точне виявлення об'єктів у реальному часі в різноманітних і складних дорожніх умовах. У цій статті представлено оцінку ефективності виявлення об'єктів у реальному часі моделей YOLOv8, сучасної системи глибокого навчання для самокерованих автомобілів у змішаному дорожньому середовищі. Мета полягає в тому, щоб оцінити, наскільки можливості виявлення об'єктів YOLOv8 можуть бути використані для самокерованих автомобілів у складних реальних дорожніх сценаріях. Результати експерименту показують, що точність YOLOv8 за нормального денного освітлення коливається в межах 0,60~0,80 [7].

Восьма публікація «Виявлення об'єктів зброї за допомогою квантованого YOLOv8» авторами якої є Муралідхар Пуллакандам, Кешав Лойя та Пранав Салота. Головна мета дослідження це створити автоматизовану систему відеоспостереження для безпеки громадян, а саме відслідковування зброї у людини. Виявити злочинну поведінку в громадських місцях важко через складність реальних сценаріїв. Камери відеоспостереження можуть фіксувати підозрілі інциденти в громадських місцях, наприклад, носіння зброї, що допомагає владі вживати превентивних заходів для захисту громадян. Запропонована система використовує найсучаснішу модель YOLOv8 для виявлення зброї в реальному часі. Для забезпечення швидкої роботи вагові коефіцієнти YOLOv8 були квантовані. У експериментах оцінювалась ефективність моделей YOLOv8 і YOLOv5 у виявленні зброї. Середнє значення середньої точності (mAP), досягнуте за допомогою YOLOv8, склало 90,1%, що перевершило значення mAP 89,1%, отримане за допомогою YOLOv5. Крім того, застосувавши вагове квантування до моделі YOLOv8, було скорочено час висновку на 15% порівняно з оригінальною конфігурацією YOLOv8 [8].

Дев’ята публікація «Візуальне виявлення відходів за допомогою YOLOv8» авторами якої є Рам Бавankуле, Вайшнаві Гайквад та Індраяні Кулкарні. Автори використовують таку ж саму архітектуру для вирішення проблеми відходів, а саме оцінка ефективності YOLOv8, останньої версії серії моделей виявлення об'єктів YOLO, для автоматизованого сортування відходів. Метою є підвищення ефективності та безпеки процесів переробки відходів шляхом використання YOLOv8 для автоматизованого сортування відходів. Результати показують, що YOLOv8 перевершує свої найсучасніші алгоритми у виявленні та класифікації відходів, що робить його цінним інструментом для покращення практики поводження з відходами [9].

Також дуже інформативною була десята публікація «Виявлення людей на тепловізійних зображеннях за допомогою YOLOv8 для пошуково-рятувальних

місій» М. Різка та І. Баяда. Ця стаття присвячена вивченню YOLOv8, останньої версії моделі виявлення об'єктів, для виявлення людей у різних сценаріях. Новий набір даних з 17 148 теплових зображень у відтінках сірого з 90 882 анотаціями людей ретельно сконструйований для представлення різних умов і сценаріїв. Всі доступні варіанти YOLOv8 різних розмірів і архітектур (від Nano до Extra Large) були навчені і оцінені з використанням цього набору даних. Оцінка навчених моделей демонструє ефективність моделей YOLOv8 у виявленні людей з високою середньою точністю 95%. Крім того, у публікації підкреслено вплив розміру моделі на точність і достовірність [10].

Усі ці системи та програми розроблені для більш вузькоспрямованих задач, а саме зачищення великих площ, або поліпшення вже існуючих систем, алгоритмів. Тому є актуальною задача створити оптимальну та точну модель для розпізнавання наземних мін ПФМ-1 і зробити її доступною для всіх імпортувавши та оптимізувавши її для мобільних пристроїв на базі ОС Android.

1.3 Вимоги до програмного забезпечення, постановка задачі

Метою роботи є дослідження та порівняння різних специфікацій та моделей нейронних мереж для розпізнавання мін типу ПФМ-1 зі смартфону. Для реалізації ПЗ необхідно ознайомитися із покадровою обробкою вхідного відеопотоку з камери телефону та визначитись із процесом навчання нейронної мережі.

Для реалізації відповідного ПЗ необхідно вирішити ряд задач:

– створити власний набір даних для навчання майбутніх моделей нейронних мереж;

– обрати необхідні архітектури нейронних мереж для навчання на наборі даних;

– навчити кожну модель на наборі даних та провести тестування для визначення кращої моделі для поставленої задачі;

– розглянути алгоритми та API для покращення роботи камери в реальному часі з моделлю нейронної мережі;

– імпортувати дану модель у вигляді застосунку на мобільний пристрій базований на ОС Android.

Система повинна відповідати наступним вимогам:

– під час роботи на смартфоні повинно забезпечуватись допустима кількість кадрів (норма від 30 до 60 FPS включно) та не перенавантажувати пристрій;

– сповіщати людину про небезпеку звуковим та візуальним сигналом.

Для створення моделей і їх навчання було обрано мову програмування Python, яка має великий спектр бібліотек з відкритим кодом, що дозволить легко підібрати параметри для навчання та автоматизувати обробку зображень і на виході отримати необхідний результат. Також це ядро мультиплатформне, та має багато реалізацій для будь-яких систем. CPython для реалізації швидкого коду та легкого у збірці, Jython для використання коду Python у Java-системах. Також ця мова дозволить швидко опанувати сучасні технології, тому що вона відноситься до класу строго/динамічно-типізованих, що дозволяє розробнику модифікувати свій код та створювати програми у рази швидше.

Для реалізації системи розпізнавання мін типу ПФМ-1 зі смартфону була обрана мова програмування Java. Дана мова програмування може допомогти створити Android застосунок, серед переваг даної мови програмування можна виділити те, що API Android дуже схоже на API мови Java, і Android підтримує якщо не всі доступні в J2SE SDK класи, то принаймні найважливіші. Ще одна перевага: можна використовувати для розробки під Android ті ж інструменти, що і Java. Основною перевагою Java є підтримка концепції об'єктно-орієнтованого програмування (ООП). Це дає змогу писати розділені та повторно використовувані програмні компоненти, будуючи сувору ієрархію додатків.

З 1990 року Java використовується як платформа для розроблення мобільних застосунків і досі залишається однією з розповсюджених мов програмування в цій сфері. Java підтримують багато популярних середовищ розроблення (IDE), включно з Eclipse, Netbeans, Tomcat, JeBrains.

Такі інструменти, як Eclipse і Netbeans, відіграють вирішальну роль у перетворенні Java на одну з найкращих мов програмування для мобільної розробки. Java вважається фундаментальною мовою розробки додатків для Android. Вона також дає змогу розробникам писати код, який без проблем працює на кількох мобільних платформах.

Розробники можуть писати код у Windows і запускати його в інших ОС, а саме macOS і Linux. Тут діє принцип "написав один раз, запускай скрізь". Це спрощує роботу для розробників, які працюють на різних системах, і спрощує процес тестування на різних машинах. Наприклад, розробники можуть перевірити, чи буде програма правильно працювати на різних розмірах екрана та операційних системах. У реальному світі існують додатки Java в різних галузях: ігри, обмін миттєвими повідомленнями, потокове передавання музики та запуск різноманітних застосунків із нейронною мережею.

PyCharm IDE – це інтегроване середовище розробки для програмування мовою Python. Воно надає аналіз коду та графічний відладчик. Дана IDE також має функціонал у вигляді модульних тестів, інтегрує програмне забезпечення для управління версіями коду (GitHub, GitLab) та підтримує веб-розробку за допомогою фреймворку Django.

Дане IDE розроблене компанією JetBrains, це кросплатформне програмне забезпечення працює на Windows, MacOS та GNU/Linux. Завантажити можна дві офіційні версії – перша це професійна версія для веб розробників, що розповсюджується під власною ліцензією, та друга версія для спільноти, що розповсюджується під ліцензією Apache.

Для розробки Android-застосунків на мові програмування Java було використано IDE Android Studio – це середовище для розробки мобільних додатків для Android. Воно засноване на IntelliJ IDEA і використовує технологію Gradle. Його можна завантажити для відомих операційних систем, а саме: Windows, macOS, Chrome OS та Linux. Android Studio в основному дозволяє редагувати файли на Java/Kotlin коді і файли конфігурації XML програми Android для налаштування графічного інтерфейсу.

Серед іншого, він пропонує інструменти для керування розробкою багатомовних додатків і дозволяє швидко візуалізувати компонування екранів на екранах з різною роздільною здатністю одночасно. Він також містить емулятор, який дозволяє запускати віртуальну систему Android на комп’ютері та допомагає відлагодити код у реальному часі без потреби у смартфоні який базується на ОС Android.

Конфігурація для навчання моделей нейронних мереж, використовувався ноутбук Dell G5 5587:

– процесор Intel Core i5-8400H, 2500-4200 MHz (вбудований граф. процесор Intel UHD Graphics 630 350-1100 MHz);
– 20 Гб ОЗУ, DDR4, 3200 MHz;
– відеокарта NVIDIA GeForce GTX 1060, 6 Гб DDR5 відеопам’яті.

З такою конфігурацією можна навчити декілька моделей нейронних мереж для вирішення поставленої задачі.

1.4 Об’єкт дослідження

ПФМ-1 (протипіхотна осколково-фугасна міна) – вибухонебезпечний пристрій, який може бути розгорнутий з мінометів, вертольотів і літаків у великій кількості; вони ковзають на землю, не вибухаючи, і вибухнуть пізніше при контакті або за певним таймером (другий вид подібної міни). Міна складається з

поліетиленового пластикового контейнера, що містить 37г рідкої вибухової речовини VS-6D, сама міна разом із рідиною важить 75г.

Рисунок 1.1 – Навчальна міна ПФМ-1 з літерою «У»

Ці пристрої становлять особливо складну проблему для звичайних методів виявлення наземних мін, таких як металошукач та інші георадари, оскільки міни складаються виключно з пластику, що робить задачу розпізнавання дуже складною [11].

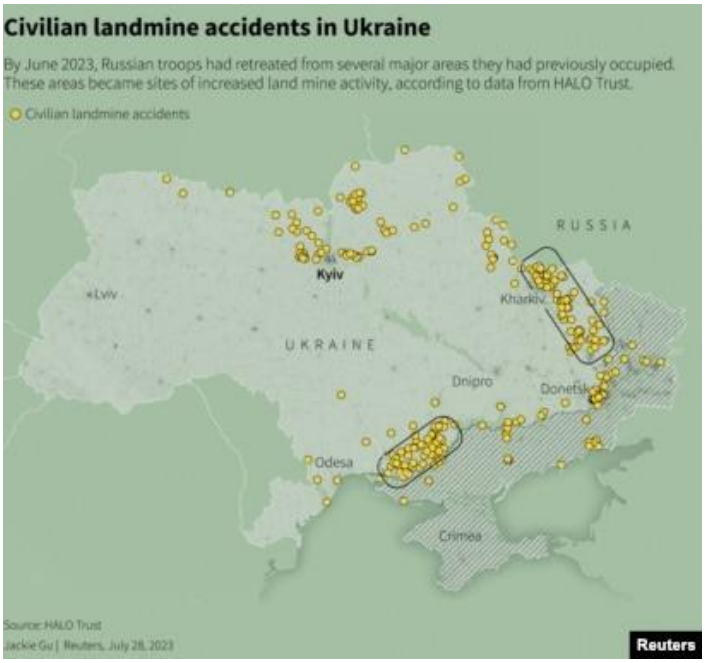

Рисунок 1.2 – Масштаби трагедії з мінами в Україні, серед яких розповсюджені міни типу ПФМ-1

На даний момент більша частина лісів та полів покрита даними мінами, що може призвести до небезпеки цивільне населення під час відпочинку та ДСНС під час зачищення територій із використанням металошукачів. Для активації даної міни потрібно тиск понад 5 кг, що свідчить про те, що будь який контакт із даною міною є небезпечним.

Тому для розробки системи було зібрано з відкритих джерел зображення даної міни для майбутньої розробки моделі нейронної мережі, щоб допомогти у розпізнаванні подібних вибухонебезпечних пристроїв, в майбутньому планується розширення набору даних та додавання інших видів вибухонебезпечних пристроїв.

Для навчання використовувався ноутбук із вбудованим графічним процесором, що дозволить відслідковувати процес навчання та конспектувати процес навчання та тестування розроблюваної моделі.

Висновки до розділу 1

При написанні першого розділу було ознайомлено із поширеною проблемою світу – розмінування великих площ та безпека цивільного населення. Інтелектуальна система комп'ютерного зору для розпізнавання та виявлення наземних мін ПФМ-1 – це той самий інструмент, який може допомогти людям вберегти себе від небезпеки маючи застосунок у себе в смартфоні, який вже став невід’ємною частиною нашого життя. Також було ознайомлено, проаналізовано та досліджено публікації для розпізнавання об’єктів в реальному часі з використанням комп’ютерного зору та нейронних мереж, серед цих усіх досліджень особливу увагу було приділено статті під назвою «Виявлення та класифікація протипіхотних мін за даними георадару за допомогою швидшого R-CNN», автором якої є Ден Мунтян, Діана Моіна та Крістіна Замфір.

Також було проаналізовано об’єкт дослідження, обґрунтовано його небезпеку та вирішено зібрати власний набір даних для навчання з використанням відкритих джерел.

2 АРХІТЕКТУРИ НЕЙРОННИХ МЕРЕЖ, КОМП’ЮТЕРНИЙ ЗІР ДЛЯ РОЗПІЗНАВАННЯ ТА ВИЯВЛЕННЯ НАЗЕМНИХ МІН ПФМ-1

Розвиток архітектури та технологій нейронних мереж відкриває нові горизонти у забезпеченні громадської та військової безпеки завдяки їх здатності до швидкої обробки інформації, яка може бути недоступною або складною для людського сприйняття. Інтеграція комп'ютерного зору та нейронних мереж у системи дозволяє автоматизувати процес ідентифікації та класифікації мінних загроз, знижує ризики для саперних груп, покращує ефективність демінування та надає можливість забезпечити людину застосунком, який він може носити із собою у своєму смартфоні.

2.1 Комп’ютерний зір

Комп'ютерний зір, або технічний зір – це наукова дисципліна, яка вивчає методи отримання, обробки, аналізу та розуміння вхідних зображень з пристроїв з метою отримання числової та символьної інформації, щоб її можна було обробити за допомогою комп'ютера. Подібно до того, як люди використовують свої очі та мозок для розуміння навколишнього світу, даний алгоритм намагається досягти подібного ефекту, щоб обчислювальні системи могли сприймати і розуміти зображення або послідовність зображень з вхідного відеопотоку та діяти відповідно до конкретної ситуації, для рішення якої вони запрограмовані.

Це розуміння досягається за допомогою різних областей, таких як статистика, фізика, математика та програмування. Збір даних для навчання та тестування формується різними способами, такими як збір послідовностей зображень, зображення з відеокамер, багатовимірні дані з медичних сканерів або власні набори даних тощо.

Існує багато технологій, які використовують комп'ютерний зір, включаючи розпізнавання об'єктів, виявлення подій, реконструкцію сцени (маппінг) і відновлення зображень.

Кінцевою метою комп'ютерного зору є розробка автоматичних стратегій для розпізнавання складних образів на багатодоменних зображеннях. Сьогодні багато галузей отримали вигоду від цього набору методів. Однією з найвідоміших є робототехніка, оскільки роботи з певним ступенем автономності повинні точно розпізнавати розташування об'єктів у своєму середовищі, щоб, наприклад, не зіткнутися з ними. Це часто досягається за допомогою датчиків або камер, причому останні є ідеальними пристроями для застосування стратегій комп'ютерного зору [12-13].

Однак робототехніка – це не єдина сфера, яка виграла від цього набору методів. Ми можемо виділити сферу медичної візуалізації, де системи здатні розпізнавати, наприклад, патологічні патерни в певній модальності зображення і діагностувати захворювання в автоматизованому режимі. Вони також використовуються в інших сферах, таких як системи безпеки, відстеження об'єктів (наприклад, відео-відстеження футболіста під час футбольного матчу) або виявлення аномалій у виготовлених деталях на виробничій лінії, останнє - як метод контролю якості [14-15].

Технічні втручання та оптимізація. Застосовуючи теоретичні концепції комп'ютерного зору, ми завжди будемо стикатися з перешкодами і проблемами, пов'язаними з навколишнім світом. Це пов'язано з тим, що світ не є досконалим, так само як і відео-пристрої, які використовуються під час тестування та роботи системи. Передивимося декілька видів технічного шуму, який може заважати працювати системі комп’ютерного зору.

1) Salt And Pepper – це імпульсивний шум який призводить до того, що уражені пікселі набувають екстремального значення, тобто максимального (білий) або мінімального (чорний). Вплив цього шуму на чорно-біле зображення, або зображення у відтінках сірого, полягає в тому, що різні чорні та білі точки хаотично розкидані по зображенню. Звідси й така креативна назва, оскільки подібна перешкода на зображенні виглядає так, ніби його обсипали цими речовинами;

2) Ґауссівський шум – це шум, отриманий від обладнання для зйомки Вплив на зображення буде подібним до рівномірного, тільки значення шуму не такі різкі, більше тяжіють до сірого, ніж до чорно-білого. Для вирішення проблеми можна використати фільтр просторового усереднення з ґауссовими коефіцієнтами. Ймовірність випадкової величини z дорівнює:

$$p_G(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \quad (2.1)$$

де z – сірий рівень;

μ – середнє значення;

σ – стандартне відхилення.

3) рівномірний шум – це шум, коли початкове значення спотвореного пікселя замінюється на інше, що має рівномірний розподіл у діапазоні можливих значень, тобто від білого до чорного. Людському оку такий вид шуму помітити майже неможливо, але при його виявленні здається ніби зображення було зашифроване. Цей шум може з'явитися під час квантування вхідного зображення. Один із відомих способів усунення рівномірного шуму – це використати фільтр просторового усереднення, це зменшить чіткість зображення, але дозволить позбутися проблеми.

Щоб вирішити проблему з будь-яким типом шуму, розробнику необхідно застосувати фільтр, сумісний з шумом, який йому заважає обробити вхідне зображення.

Дуже поширене використання комп’ютерного зорі на практиці – це підключення подібних алгоритмів у камери авто та камер відеоспостереження. Використання подібних технологій призводять до створення більш ефективних і дієвих систем безпеки. Комп’ютерний зір має фундаментальне значення в секторі відеоспостереження та безпеки [16].

Рисунок 2.1 – Приклад роботи комп’ютерного зору з виконанням задачі класифікації об’єктів на вхідному зображенні

У традиційній системі відеоспостереження (замкнутого контуру) зазвичай можна переглядати вміст до 16 або більше камер одночасно. Це складне завдання для охоронця об’єкту, оскільки дослідження показують, що після 22 хвилин моніторингу він втрачає до 95 відсотків активності на місці події. Система з комп’ютерним зором попереджає охоронця про рух або вказує, на якій камері найімовірніше відбувається підозріла або небезпечна активність завдяки сучасним алгоритмам та гарно підібраному набору даних.

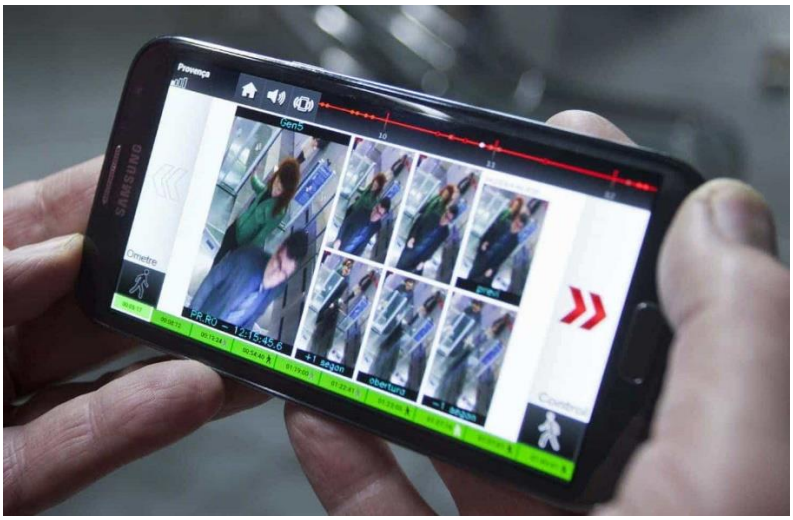

Рисунок 2.2 – Застосування технологій комп’ютерного зору разом із системою відеоспостереження у торгівельному пункті

Такі системи також використовуються в радарах та камерах на трасах або дорогах для фіксації швидкості руху автомобіля та його номеру, щоб знайти та ідентифікувати порушника який перевищує обмежену швидкість.

Деякі застосування включають виявлення покинутих об’єктів у місцях скупчення людей, моніторинг творів мистецтва в музеях і виявлення несанкціонованого в’їзду транспортних засобів на заборонені або захищені території.

Новітні моделі автомобілів мають вбудовані системи які працюють на алгоритмах комп’ютерного зору, наприклад найпростіша система парктронік, який має в основі камеру для аналізу 3D простору та вимірювання відстані до найближчого об’єкту, щоб при паркуванні водій не зіткнувся із іншою машиною або бордюром. На даний момент є версії які дозволяють точно розрахувати відстань до найближчої перешкоди та попередити звуковим сигналом про досягнення критичної відстані.

Рисунок 2.3 – Приклад роботи системи парктронік у сучасних автомобілях

Сучасні алгоритми комп’ютерного зору також можуть допомогти у створенні 3D моделей з 2D зображень, так наприклад у статті Айшварія Сінг «Алгоритм комп’ютерного бачення DeepMind використовує силу уяви для створення 3D-сцен із 2D-зображень» створила разом із командою систему GQN, яка може мислити й

уявляти як люди. Дана система може відтворювати 3D-об’єкти без необхідності навчатися тому, як виглядають об’єкти під різними кутами [17].

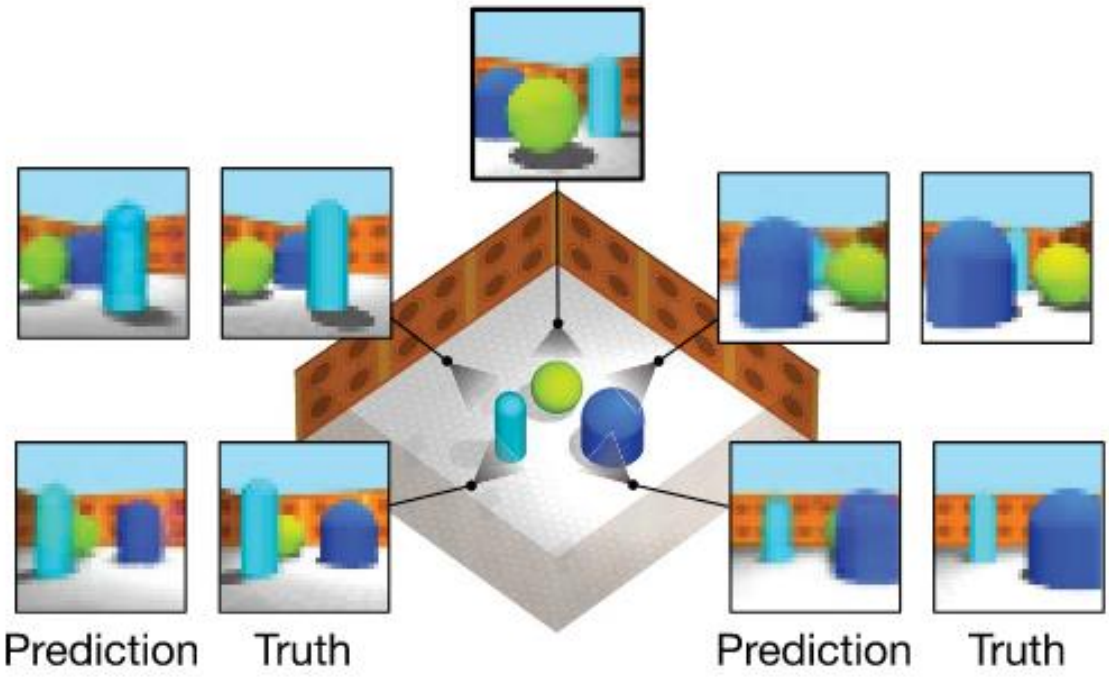

Рисунок 2.4 – Процес створення 3D-сцени використовуючи алгоритми комп’ютерного зору

Система GQN має можливість самостійно дізнаватися про форму, розмір і колір об’єкта, а потім може комбінувати всі ці характеристики для створення точної 3D-моделі. Більше того, дослідники змогли використати цей алгоритм для розробки нових сцен без необхідності явного навчання системи щодо того, який об’єкт куди має йти. Випробування проводилися у віртуальній кімнаті, а результати були опубліковані командою у згаданій вище дослідницькій роботі. Алгоритм наразі перевірявся лише на об’єктах і ще недостатньо розроблений для роботи з людськими обличчями.

Також автомобілі Tesla (моделі 3, S, Y) мають вбудований автопілот Tesla Vision, нову систему, оснащену лише камерами, що робить її однією з небагатьох компаній у світі, яка не використовує радари для розрахунків відстаней до об’єктів в реальному часі [18]. Команда інженерів навчила алгоритми комп’ютерного зору

вирішувати деякі з найскладніших проблем, таких як дотримання смуги руху, зміна смуги руху та круїз-контроль. У них також є додаткові завдання, такі як водіння на зоні паркування і водіння містом.

Рисунок 2.5 – Вісім камер автомобілю Tesla Model S

Дані системи потребують постійного покращення та повторно навчання, оскільки такі технології, як розпізнавання обличчя і виявлення підозрілого руху на територіях, все ще ненадійні і часто спричиняють хибні тривоги, тому досі залишається потреба у наглядачі, який буде керувати даними системами та слідкувати за подіями на екрані.

2.2 Convolutional Neural Network

Нейронні мережі є підмножиною машинного навчання та складають основу алгоритмів глибокого навчання. Вони складаються з вузлів, що містять вхідний рівень, один або кілька прихованих шарів і вихідний рівень. Кожен вузол з’єднується з іншим і має вагу та поріг активації. Якщо вихід створеного окремого вузла перевищує вказаний поріг, то даний вузол активується, надсилаючи дані на

наступний рівень НМ. В будь якому іншому випадку дані не передаються на наступний рівень створеної згорткової мережі.

Згорткові нейронні мережі відрізняються від інших нейронних мереж своєю чудовою продуктивністю в обробці зображень і природної мови. Вони мають три основні шари, а саме:

1) згортковий;
2) об'єднувальний;
3) повнозв'язний.

Згортковий рівень є першим шаром згорткової мережі. За згортковими шарами можуть слідувати додаткові згорткові шари або шари об’єднання, повністю пов’язаний рівень є останнім за будь-яких умов, оскільки його завдання полягає в класифікації обробленої інформації. З кожним наступним шаром складність нейронної мережі зростає, ідентифікуючи більші частини зображення. Попередні шари зосереджені на простих елементах, таких як кольори, краї зображення та положення зображення. Коли дані зображення просуваються між шарами CNN, вони починають розпізнавати більші особливості або форми об’єкта, поки не ідентифікують потрібний об’єкт [19].

Згортковий шар – це основний функціональний блок CNN, в ньому відбувається більшість обчислень при обробці вхідних даних. Він складається з декількох компонентів, а саме: вхідні дані, фільтр і карта ознак.

Якщо вхідними даними буде кольорове зображення, яке складається з матриці пікселів у 3D, то це означає, що вхідні дані матимуть три виміри - висоту, глибину та ширину, які повністю відповідають формату RGB на зображенні. В даному шарі також є детектор ознак, також відомий як фільтр(ядро), який буде рухатися по сприйнятливих полях зображення, перевіряючи, чи присутня ознака. Цей процес називається згортка.

Детектор ознак – це двовимірний масив вагових коефіцієнтів, який представляє частину вхідних даних (зображення). Даний фільтр може мати різний

розмір, фільтр/ядро зазвичай виставляється розміром 3x3, але розробник може сам вирішити який для нього є оптимальний розмір ядра; це також визначає розмір сприйнятливого поля. Потім фільтр накладається на ділянку зображення, і обчислюється точковий добуток між вхідними пікселями і фільтром. Цей точковий добуток подається у вихідний масив. Після цього фільтр зсувається на крок, повторюючи процес доти, доки ядро не пройде по всьому зображенню. Кінцевий результат серії точкових добутків вхідних даних і фільтра називається картою ознак або картою активації [20].

Ваги в детекторі ознак залишаються фіксованими при обробці зображення, що також називається розподілом параметрів. Параметри значення ваг змінюються під час навчання за допомогою зворотного розповсюдження та градієнтного спуску. Однак є три гіпер-параметри, які впливають на об'єм вихідних даних, які необхідно встановити до початку навчання нейронної мережі:

1) кількість фільтрів впливає на глибину вихідних даних. Наприклад, чотири різні фільтри дадуть чотири різні карти ознак, створюючи глибину, що дорівнює чотирьом;

2) крок, тобто відстань або кількість пікселів, на яку ядро переміщується по вхідній матриці. Хоча значення кроку в два і більше пікселів зустрічається нечасто, більший крок дає менший результат точності розпізнавання;

3) нульове заповнення зазвичай використовується, коли фільтри не підходять до вхідного зображення. В даному випадку всі елементи, що виходять за межі вхідної матриці, прирівнюються до нуля, що дає на виході більший або однаковий за розміром результат. Існує три типи заповнення:

– дійсне заповнення – даний тип також відомий як відсутність проміжків. В даному випадку остання згортка прибирається (видаляється(, якщо розміри матриці не вирівнюються згідно алгоритму;

– однакове заповнення – даний тип заповнення гарантує, що вихідний шар має той самий розмір, що і вхідний;

– повне заповнення – даний тип заповнення збільшує розмір вихідного шару шляхом додавання нулів до межі вхідного шару.

Після кожної операції згортки CNN застосовує перетворення лінійної одиниці (ReLU) до карти ознак, вносячи нелінійний тип даних у модель.

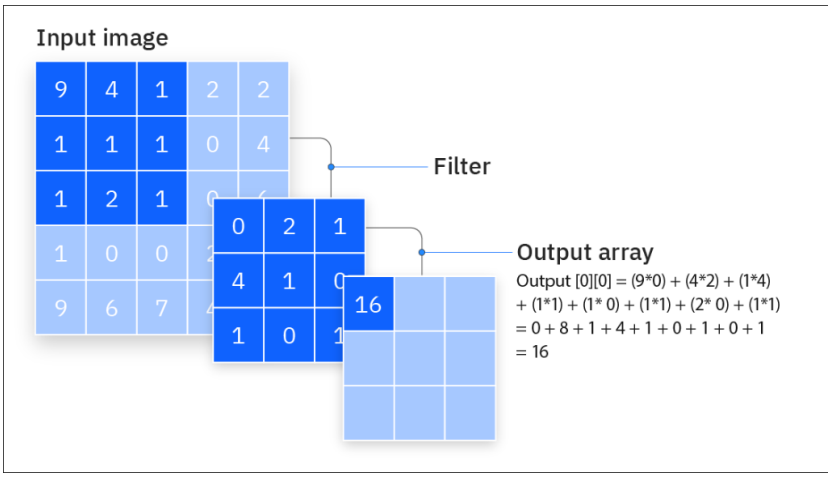

Рисунок 2.6 – Процес згортки зображення за допомогою ядра розміром 3×3

Додатковий шар згортки. Даний шар згортки може слідувати за початковим згортковим шаром. Коли це відбувається, структура CNN може стати ієрархічною, оскільки пізніші шари нейронної мережі можуть бачити пікселі в межах рецептивних полів попередніх шарів. Для прикладу надано наступне зображення (рис. 2.2), на ньому зображено спробу визначити, чи є на зображенні велосипед. Можна уявити велосипед як суму частин. Він складається з наступних частин: рами, керма, коліс, педалей тощо. Кожна окрема частина велосипеда становить шаблон нижчого рівня в нейронній мережі, а комбінація його частин представляє шаблон вищого рівня, створюючи ієрархію ознак у згортковій нейронній мережі. Згортковий шар перетворює зображення в числові значення, що дозволяє нейронній мережі інтерпретувати та виокремлювати необхідні деталі на вхідному зображенні.

Рисунок 2.7 – Принцип роботи додаткового шару згортки

Об'єднання шарів, зменшує розмірність і завдяки цьому зменшує кількість параметрів у вхідних даних. Подібно до шару згортки, операція об'єднання накладає фільтр(матрицю) на всі вхідні дані, але різниця полягає в тому, що цей фільтр не має вагових коефіцієнтів. Замість цього ядро застосовує функцію агрегування до значень в межах сприйнятливого поля, заповнюючи вихідний масив. Існує два основних типи об'єднання шарів, а саме:

– максимальне об'єднання – це коли фільтр рухається по вхідному сигналу, обираючи піксель з максимальним значенням, щоб відправити його до вихідного масиву. Цей підхід використовується частіше порівняно з середнім об'єднанням шарів;

– усереднене об'єднання – це коли фільтр рухається по вхідному сигналу та обчислює середнє значення в межах сприйнятливого поля, щоб відправити його у вихідний масив.

Хоча в шарі об'єднання втрачається багато інформації, він також має ряд переваг для нейронної мережі. Вони допомагають зменшити складність, підвищити ефективність і обмежити ризик надмірного пристосування та облегшити процес навчання для обчислювальних потужностей [21].

Повнозв'язний шар. У частково зв'язаних шарах пікселі вхідного зображення подані на вхід не пов'язані з вихідним шаром безпосередньо. Однак у повністю зв'язному шарі кожна вершина вихідного шару з'єднується з вершиною попереднього шару, тому задача даного зв’язного шару це класифікація на основі ознак, отриманих за допомогою попередніх шарів та їх різних фільтрів. В той самий момент згорткові та об'єднувальні шари за звичайним налаштуванням, використовують функції активації ReLu або подібні, повнозв’язні шари зазвичай використовують функцію активації SoftMax для належної класифікації вхідних даних, отримуючи ймовірність в нормалізованому вигляді від 0 до 1 для точної класифікації об’єкту/-ів при виконанні поставленої задачі [22].

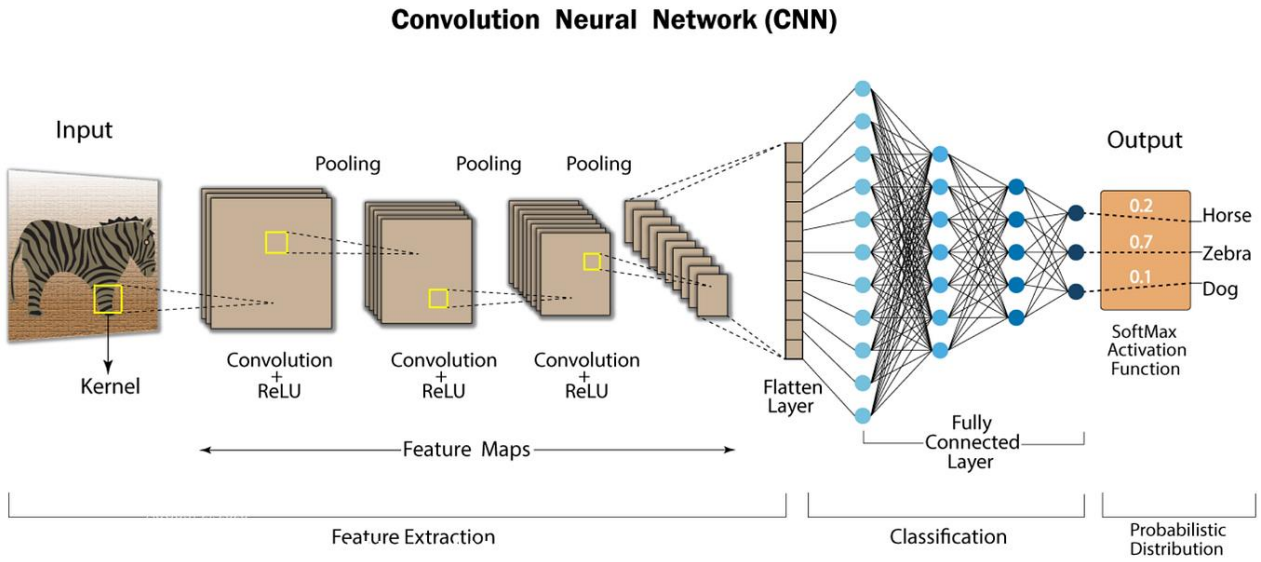

Рисунок 2.8 – Процеси розпізнавання об’єкта згортковою нейронною мережею при наданні вхідного зображення

2.3 Архітектури R-CNN, Fast R-CNN та Faster R-CNN

Така мережа як Convolutional Neural Network вперше була розроблена Яном Лекуном та його командою (стаття). Вона була в основному розроблена між 1989 і 1998 роками для завдання розпізнавання рукописних цифр. З того самого часу дана архітектура розвивалась та поєднувалась з іншими типами нейронних мереж.

Архітектура R-CNN (Region-Based CNN) була розроблена у 2014 році Россом Гірщік та його командою. Основна ідея алгоритму полягала у знаходженні потенційних об'єктів на зображенні та їхнього розбиття на регіони за допомогою методу selective_search – алгоритм пропозиції регіону, який використовується для виявлення об’єктів. Він створений для швидкої роботи з дуже високим рівнем запам’ятовування. Він заснований на обчисленні ієрархічного групування подібних регіонів на основі кольору, текстури, сумісності розміру та форми [23].

Рисунок 2.9 – Сегментоване зображення для аналізу методом selective seach

Зазвичай згорткові нейронні мережі використовують для вилучення характеристик з кожного отриманого регіону. Використання методу опорних векторів (SVM) для класифікації оброблених ознак та коригування меж регіонів за допомогою лінійної регресії.

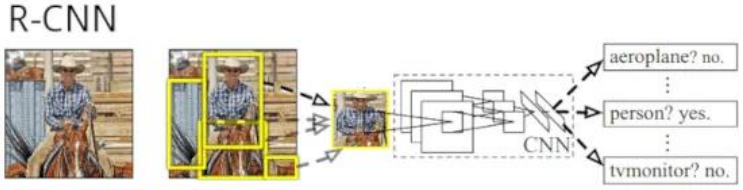

Рисунок 2.10 – Процес розпізнавання мережею R-CNN

При обробці зображення блок Region-Based передає мережі окремі регіони з об'єктами та їхніми класами. У центрі стоїть CNN, які показують високу точність на зображеннях. Маючи такий багатий функціонал дана модель має деякі недоліки під час роботи:

1) енерговитратний – потребує великої кількості часу на навчання. Метод selective_search сегментує зображення на багату кількість регіонів (або скільки задає розробник), які потім під час проходу по ним за допомогою важкого алгоритму об'єднуються в регіони більшого розміру. CNN також потребують обчислювальних операцій, які займають більшу кількість часу;

2) не може бути використаний для відеопотоку – ця проблема виникає через проблему в енерговитратності, тому кадри просто не встигатимуть оброблятися, і користувач у процесі отримає велику втрату при обробці зображень в реальному часі (близько 90% маючи потужні обчислювальні сили);

3) Selective_search – цей алгоритм не належить до алгоритмів МН (машинне навчання), тому можуть виникнути проблеми з виявленням необхідних об’єктів на вхідних зображеннях.

На даний момент R-CNN вважається застарілою архітектурою та не використовується на практиці.

Автори R-CNN узяли усі недоліки попередньою архітектури, проаналізували їх та у 2015 році випустили нову архітектуру під назвою Fast R-CNN. Вона працює за наступним алгоритмом.

Зображення подається на вхід CNN і обробляється методом selective search. У результаті виходить карта ознак на вхідному зображенні. Координати регіонів об’єктів, які розпізнаються у результаті перетворюються на координати на карті ознак, що дає змогу ефективніше знайти потрібні об'єкти під час обробки зображення. Карта ознак із регіонами передається шару Region of Interest (RoI). Тут на кожен регіон накладається сітка розміром, який задає розробник (що менша сітка, то більша деталізація і краще навчання, але повільніше навчання). Після чого

використовується шар MaxPolling задача якого це зменшення розмірності вхідних даних. Усі регіони об'єктів мають однакову фіксовану розмірність, це одна з особливостей CNN. У результаті знайдені усі сформовані ознаки подаються на вхід фінального (повнозв’язного) шару. Перший із функцією активації SoftMax визначає ймовірність приналежності до класифікації об’єкта, другий шар – це regressor межі (зміщення) регіону об’єкта для розпізнавання і його подальшого виділення [24].

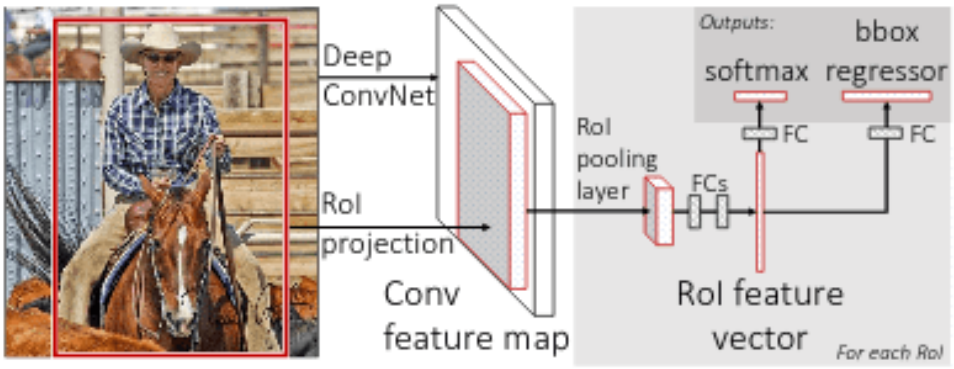

Рисунок 2.11 – Архітектура Fast R-CNN

Архітектура Fast R-CNN показує високу точність і більший приріст часу обробки на відміну від R-CNN, оскільки алгоритму не потрібно подавати всі знайдені регіони на шар згортки. Але тим не менш, цей метод використовує важкий та затратний selective_search алгоритм. Тому автори прийшли до більш нової CNN [25-26].

В 2016 році ті ж самі автори продовжили вдосконалення нейронної мережі та запропонували нову архітектуру під назвою Faster R-CNN. Вони розробили метод локалізації об'єкта замість Selective Search – Region Proporsal Networks (RPN). В основі RPN лежить система якорів. Архітектура Faster R-CNN працює за наступним алгоритмом – вхідне зображення або дані подаються на вхід CNN, після чого формується карта ознак. Карта ознак обробляється за допомогою Region Proporsal Networks шари. Матриця проходиться по карті ознак за допомогою фільтра певного розміру (залежить від потужності системи). Центр матриці пов'язаний із центром

якорів. Якорі умовно це області, які мають різні співвідношення та розміри, тобто матриця з різними n×m. Автори архітектури за замовчуванням використовують 3 співвідношення сторін для якоря і 3 розміри для ядра [27].

На основі метрики intersection-over-union (IoF), ступеня перетину областів якорів і істинних розмічених прямокутників – чи є необхідний об'єкт на зображенні або в даних чи ні. Далі використовується алгоритм з минулої архітектури Fast R-CNN – карта ознак з отриманими об'єктами передаються шару RoI з подальшим опрацюванням і фінальною класифікацією об’єкта, а також з визначенням зсуву об’єктів, які необхідно розпізнати на тому ж зображенні [28].

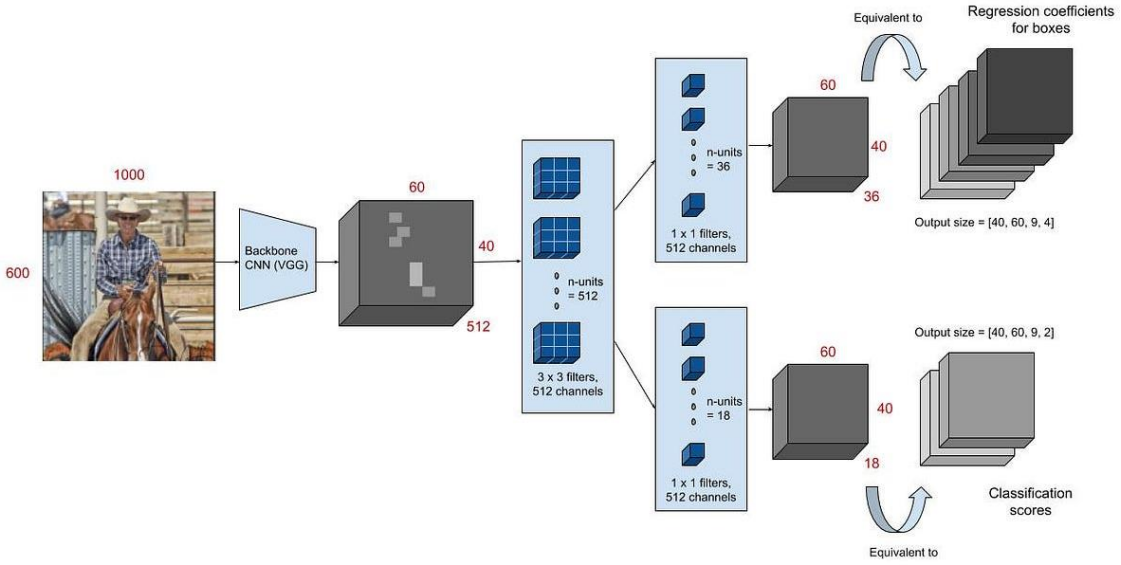

Рисунок 2.12 – Архітектура Faster R-CNN

Архітектура Faster R-CNN справляється трохи гірше з локалізацією, але працює швидше за Fast R-CNN, що є перевагою над попередніми двома моделями.

Можна зробити єдиний висновок для даних трьох архітектур:

1) R-CNN:

– використання selective search як генератор границь на вхідному зображенні;

– використання SVM + Ridge для класифікації та регресії об’єктів на зображенні (причому їхня паралельна робота не можлива);

– запуск нейронної мережі для обробки кожного об’єкту на зображенні окремо;

– низька швидкість роботи нейронної мережі;

2) Fast R-CNN:

– нейронна мережа запускається тільки один раз на зображення – всі знайдені об’єкти перевіряються на основі єдиної карти ознак;

– швидке та розумне опрацювання об’єктів різного розміру завдяки RoI шару;

– зміна SVN на SoftMax шар;

– можливість паралельної роботи класифікації та регресії, що значно пришвидшує роботу архітектури.

3) Faster R-CNN:

– генерація границь об’єкта за допомогою спеціального диференційованого модуля;

– зміни в процесі обробки зображення, пов'язані з появою RPN модуля;

– найшвидша, найточніша та нова модель, яка на цей час є найактуальнішою.

Розвиток R-CNN рухався від розрізнених алгоритмів, що розв'язують одне завдання, у бік єдиного рішення. Таке об'єднання дає змогу зробити майже будь-який підхід точнішим і найпродуктивнішим, Object Detection не став винятком [29].

2.4 YOLO як інструмент для object tracking. YOLOv8.

Для виконання задачі розпізнавання мін ПФМ-1 необхідні алгоритми, які підтримують не тільки object detection формат, а й object tracking. Object tracking – це дуже цікавий напрямок, який вивчається й еволюціонує не перший десяток років. Зараз багато розробок у цій царині побудовано на глибокому навчанні, яке має перевагу над стандартними алгоритмами, оскільки нейронні мережі можуть апроксимувати функції набагато краще.

Серед усіх відомих алгоритмів YOLO – один із найкращих у даній області. Так як основна ідея застосунку – це імпорт навченої моделі нейронної мережі для розпізнавання мін типу ПФМ-1 у реальному часі, то використання просто алгоритму object detection не є правильним, тому потрібно зрозуміти різницю між цими двома поняттями [30].

Object Detection – це просто визначення об'єктів на зображенні/кадрі. Тобто алгоритм або нейронна мережа визначають об'єкт і записують його позицію та bounding boxes (параметри прямокутників навколо об'єктів) [31]. Поки що мови про інші кадри не йде, і алгоритм працює тільки з одним. Приклад приведено на рисунку 2.13:

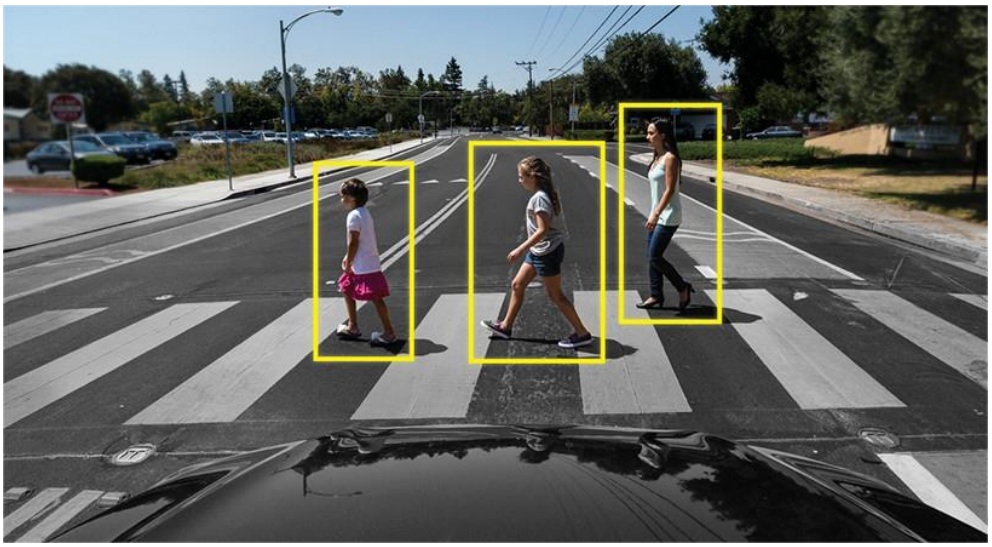

Рисунок 2.13 – Приклад роботи Object Detection на 1 кадрі зображення

Object Tracking має інший підхід до свого завдання. Головна мета не просто визначити об'єкти на вхідному кадрі, а ще й пов'язати інформацію з попередніх кадрів таким чином, щоб не втрачати об'єкт, або зробити його унікальним [32].

З наступного відеоряду було вирізано 1 кадр розпізнавання автомобілів, наступна задача Object Tracking це розпізнати усі автомобілі та порахувати їх, так можна збирати статистику про завантаженість траси у будь який час (рис. 2.14).

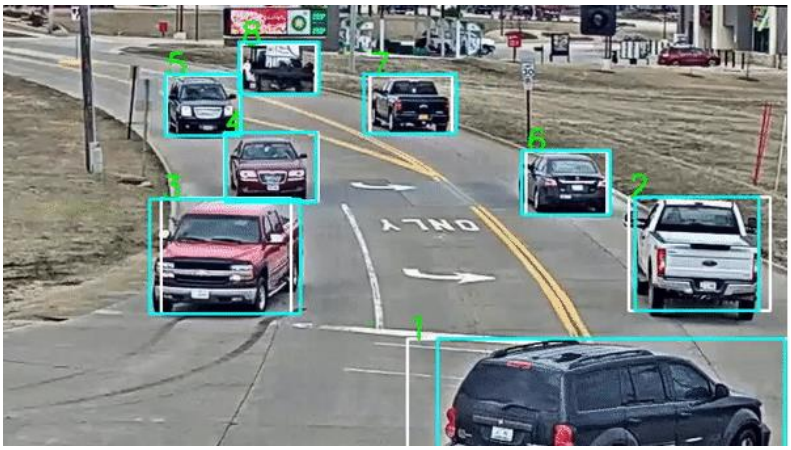

Рисунок 2.14 – Приклад роботи Object Tracking з нумерацією кількості автомобілів

Тобто Object Tracker включає в себе Object Detection для визначення об'єктів, та інші алгоритми для розуміння, який об'єкт на новому кадрі належить якому з попереднього кадру.

YOLO вважається ефективнішим за багато інших алгоритмів для визначення об'єктів. Ось невеликий графік для порівняння від авторів YOLO:

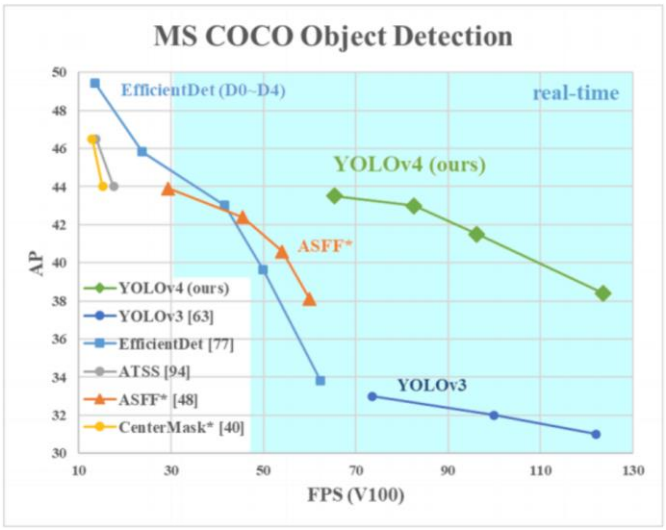

Рисунок 2.15 – Порівняння моделей YOLO та інших алгоритмів в реальному часі

Архітектури різних моделей Object Detectors. Існує кілька видів архітектур нейронних мереж, створених для визначення об'єктів. Вони в основному

поділяються на "дворівневі", такі як R-CNN, fast R-CNN і faster R-CNN, і "одноровневі", такі як YOLO.

"Дворівневі" нейронні мережі, перераховані вище, використовують так звані регіони на зображенні, щоб визначити, чи знаходиться в цьому регіоні певний об'єкт [33].

Зазвичай це виглядає так (для Faster R-CNN, яка є найшвидшою з перерахованих дворівневих систем):

1) подається картинка/кадр на вхід;
2) кадр проходить через CNN для формування feature maps;
3) окремою нейронною мережею визначаються регіони з високою ймовірністю знаходження в них об'єктів;
4) далі ці регіони за допомогою RoI pooling стискаються і подаються в нейронну мережу, що визначає клас об'єкта в регіонах.

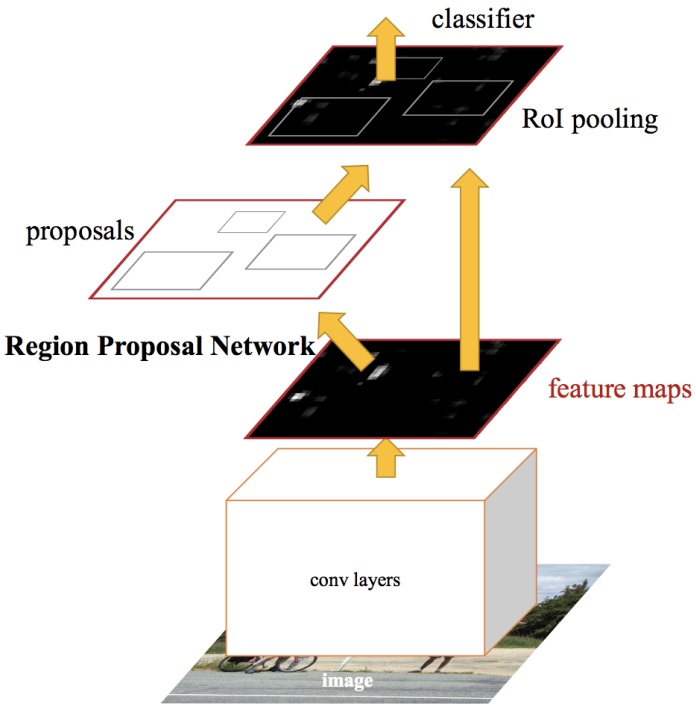

Рисунок 2.16 – Архітектура Faster R-CNN

Але в алгоритмі Faster R-CNN та подібних є одна проблема – вони не дивляться на картинку "повністю", а тільки на окремі регіони, і вони відносно повільні [34].

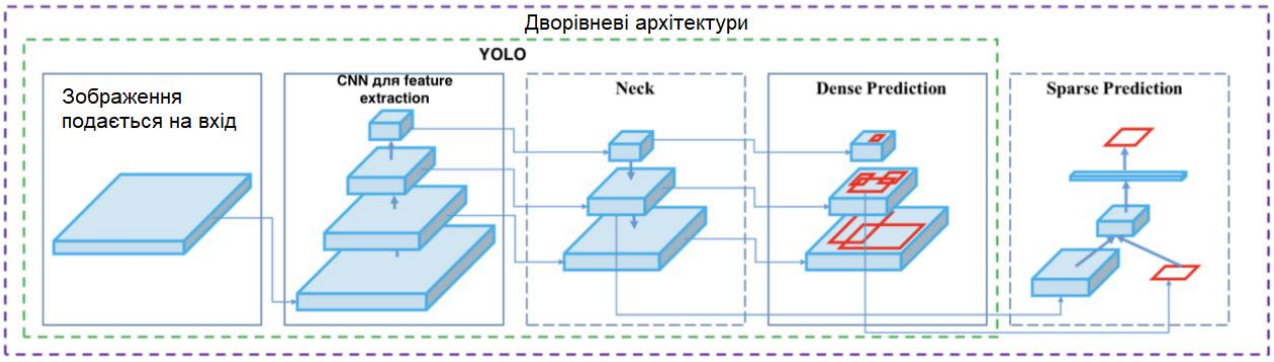

Рисунок 2.17 – Архітектура YOLO

Архітектура YOLO в перших блоках не надто відрізняється за "логікою блоків" від інших детекторів, тобто на вхід подають зображення, далі створюють feature maps за допомогою CNN (щоправда, в YOLO використовується своя CNN під назвою Darknet-53), потім ці feature maps певним чином аналізують (про це трохи згодом), видаючи на виході позиції та розміри bounding boxes і класи, яким вони належать [35].

На рисунку вище є такі блоки як Neck, Dense Prediction и Sparse Prediction.

Sparse Prediction – це просто повторення того, як дворівневі алгоритми працюють, визначають окремо регіони і потім класифікують ці регіони.

Neck – це окремий блок, який створений для того, щоб агрегувати інформацію від окремих шарів з попередніх блоків (як показано на малюнку вище) для збільшення акуратності передбачення. Схожі терміни, що відносяться до цього блоку, можуть називатися інакше, наприклад: "Path Aggregation Network", "Spatial Attention Module" і "Spatial Pyramid Pooling".

І те, що відрізняє YOLO від усіх інших архітектур – блок під назвою Dense Prediction. Цей блок допоміг YOLO вирватися в лідери за ефективністю визначення об'єктів [36].

YOLO (You Only Look Once) несе в собі філософію дивитися на картинку один раз, і за цей один перегляд (тобто один прогін картинки через одну нейронну мережу) робити всі необхідні визначення об'єктів.

Як YOLO навчається на даних:

1) зображення які подаються на вхід зменшуються під розмір 416×416 перед початком навчання нейронної мережі, щоб можна було їх подавати батчами на вхід нейронної мережі (для прискорення навчання);

2) зображення ділиться на клітинки розміром n×n. В YOLO прийнято ділити на клітини розміром 13x13. Такі клітинки, які називаються grid cells, лежать в основі ідеї YOLO. Кожна клітина є якорем, до якого прикріплюються bounding boxes. Тобто навколо клітинки малюють кілька прямокутників для визначення об'єкта (оскільки незрозуміло, якої форми прямокутник буде найвідповіднішим, їх малюють одразу кілька та різних форм), і їхні позиції, ширину та висоту обчислюють відносно центру цієї клітинки [37];

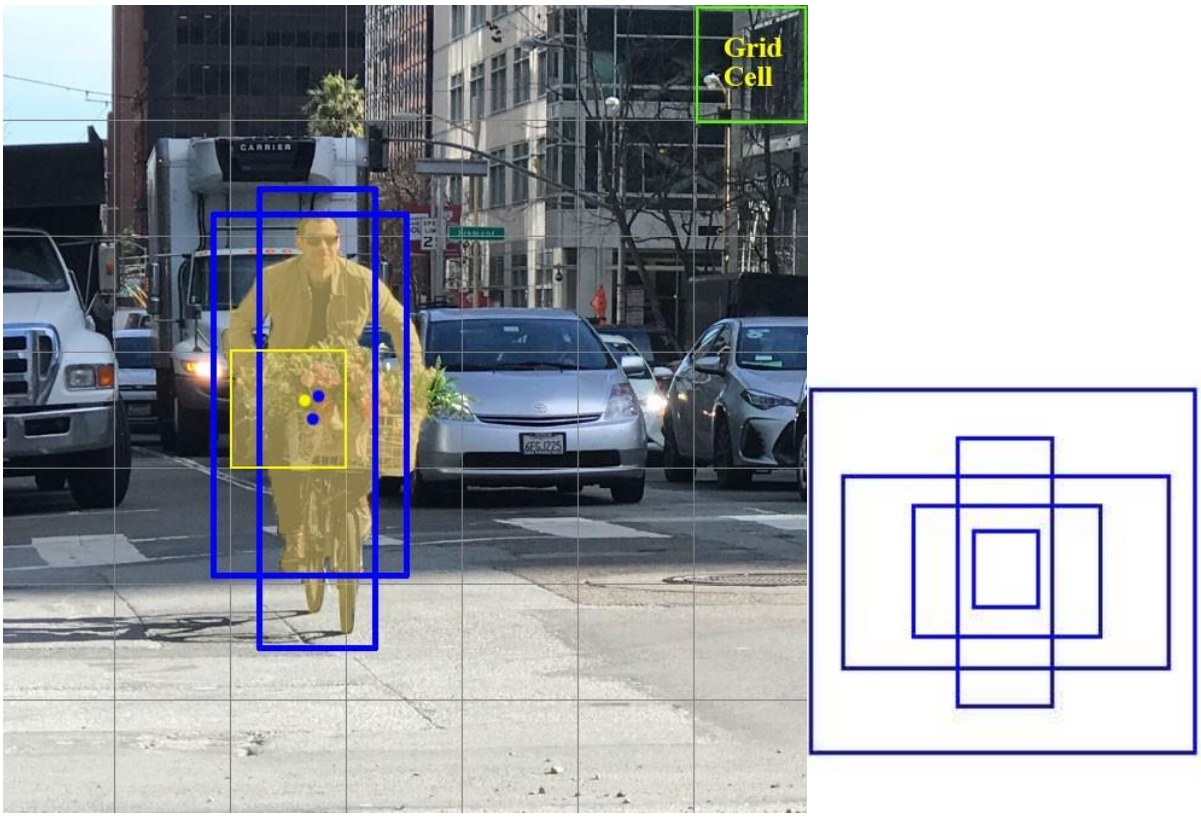

Рисунок 2.18 – Процес ділення на клітинки

3) для створення подібних прямокутників блакитного кольору використовується техніка anchor boxes (у перекладі - якірні коробки, або "якірні прямокутники"). Їх задають на самому початку або сам користувач, або їхні розміри визначають, виходячи з розмірів bounding boxes, які є в датасеті, на якому тренуватимуть YOLO (використовують K-means clustering і IoU для визначення найбільш підходящих розмірів). Зазвичай задають близько 3 різних anchor boxes, які будуть намальовані навколо (або всередині) однієї клітини [38];

4) зображення з набору даних проходить через нейронну мережу (зауважимо, що окрім картинки в тренувальному наборі даних мають бути визначені позиції та розміри справжніх bounding boxes для об'єктів, які є на ній. Це називається анотація даних і робиться це здебільшого вручну).

Далі на виході YOLO видає наступне. Для кожної клітини нам потрібно зрозуміти дві принципові речі:

1) котрий з anchor boxes, з трьох намальованих навколо клітини, найкраще підходить, і як його можна трохи налаштувати, щоб краще адаптуватися до об'єкта?

2) що саме знаходиться всередині цього anchor box, і чи присутній об'єкт у ньому взагалі?

Щоб максимально точно передбачити ці параметри і межі, для максимально точного визначення об'єкта на картинці. А confidence score, який визначається для кожного передбаченого bounding box, є таким собі фільтром для того, щоб відсіяти зовсім неточні передбачення. Для кожного передбаченого bounding box можна оцінити його IoU на ймовірність того, що це певний об'єкт (імовірнісний розподіл розраховується під час навчання нейронної мережі), береться найкраща ймовірність з усіх можливих, і якщо число після множення перевищує певний поріг, то можна залишити цей передбачений bounding box на картинці. Далі, коли залишилися тільки передбачені bounding boxes з високим confidence score (якщо їх візуалізувати) [39].

Тепер можна використовувати техніку NMS (non-max suppression), щоб відфільтрувати bounding boxes таким чином, щоб для одного об'єкта був тільки один передбачений bounding box [40].

Рисунок 2.18 – Процес роботи NMS

При розробці було використано останню версію YOLOv8, можна передивитися нові особливості:

– зручний для користувача API (командний рядок + Python);

– YOLOv8 має високу ефективність і гнучкість, підтримує безліч форматів експорту, і модель може працювати на CPU і GPU;

– у кожній категорії моделей YOLOv8 є п'ять моделей для виявлення, сегментації та класифікації. YOLOv8 Nano - найшвидший і найменший, тоді як YOLOv8 Extra Large (YOLOv8x) - найточніший, але найповільніший серед них.

Можна зробити висновок, що YOLOv8 – це новітнє сімейство моделей виявлення об'єктів на базі YOLO від Ultralytics, що забезпечують найсучасніші характеристики.

2.5 CNN-NNAPI як апаратна підтримка для Android-застосунків

В Android-пристроях є багатий спектр додатків, які використовують нейронні мережі. Маски в Snapchat та Instagram, категоризація зображень у галереї смартфону або cloud-мережі, поліпшення якості фотографій у Google Camera – усі ці функції в додатках використовують нейронні мережі. Майбутнє мобільних пристроїв тісно пов'язане з розвитком нейронних мереж та їхньою інтеграцією в різноманітні додатки: нейронні мережі допомагають удосконалити користувацький досвід і підвищити ефективність роботи додатків [41]. Для створення та впровадження нейронних мереж в Android додаток можна застосовувати різні інструменти. Розглянемо найбільш популярні:

1) TensorFlow Lite. Бібліотека спеціально розроблена для роботи з мобільними пристроями на базі Android. Забезпечує високу продуктивність і компактний розмір моделей, щоб не займати багато вільного простору. Підтримує інтеграцію з NNAPI-моделями – допомагає прискорити виконання на апаратному рівні;

2) Keras, PyTorch. Ці бібліотеки в парі мають багатий набір інструментів для навчання нейронних мереж і підтримують експорт моделей у формат TensorFlow Lite. Якщо перед розробником стоїть завдання створити складну нейронну мережу з нестандартною архітектурою, можна використовувати ці бібліотеки;

3) ML Kit. Сервіс надає готові рішення для низки завдань: наприклад, розпізнавання зображень і звуку, аналіз тексту тощо. Але при використанні ML Kit обмежені можливості налаштування створюваних моделей;

4) NNAPI (Neural Networks API) - це API, розроблене компанією Google. Дозволяє використовувати апаратне прискорення для виконання обчислень нейронних мереж на пристроях з операційною системою Android.

Користь від використання NNAPI для роботи з нейронними мережами на Android:

1) може збільшити швидкість роботи додатків і зменшити навантаження на процесор. Використовує апаратну прискорену обробку: це дає змогу підвищити продуктивність виконання операцій нейронних мереж [42];

2) забезпечує високу оптимізацію та ефективність виконання операцій для нейронної мережі. Смартфони від різних виробників можуть мати різні апаратні можливості та специфікації, включно з процесорами та графічними прискорювачами. Використання NNAPI дає змогу розробникам мобільних застосунків використовувати апаратні ресурси пристрою найкращим чином [43];

3) NNAPI підтримують різні бібліотеки, включно з TensorFlow Lite;

4) додаток стає енергоефективним: підвищується швидкість, зменшується споживання.

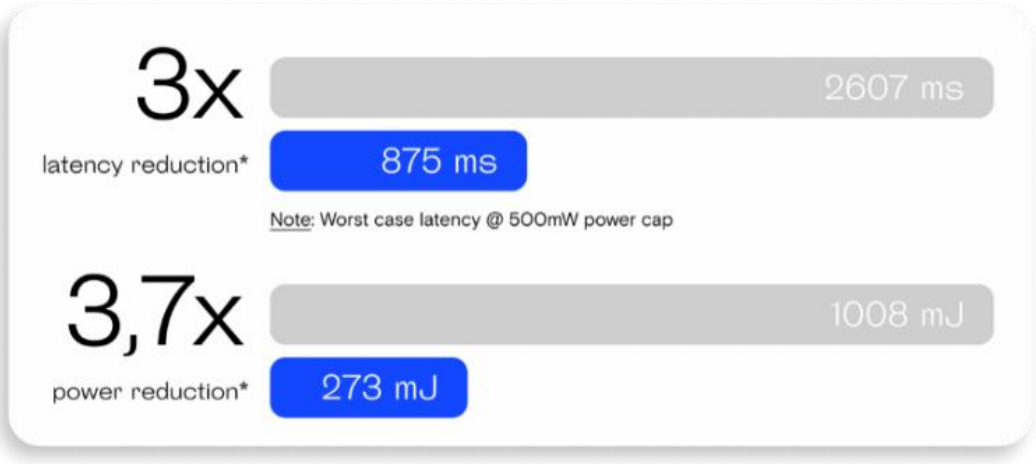

Рисунок 2.19 – Офіційна інформація від Google: NNAPI в 3 рази прискорює час очікування і в 3,7 рази знижує споживання заряду

Роздивимось структуру NNAPI та його технічні вимоги для пристрою на рис. 2.20. NNAPI доступний на всіх пристроях з Android 8.1 (API level 27) або вище [44].

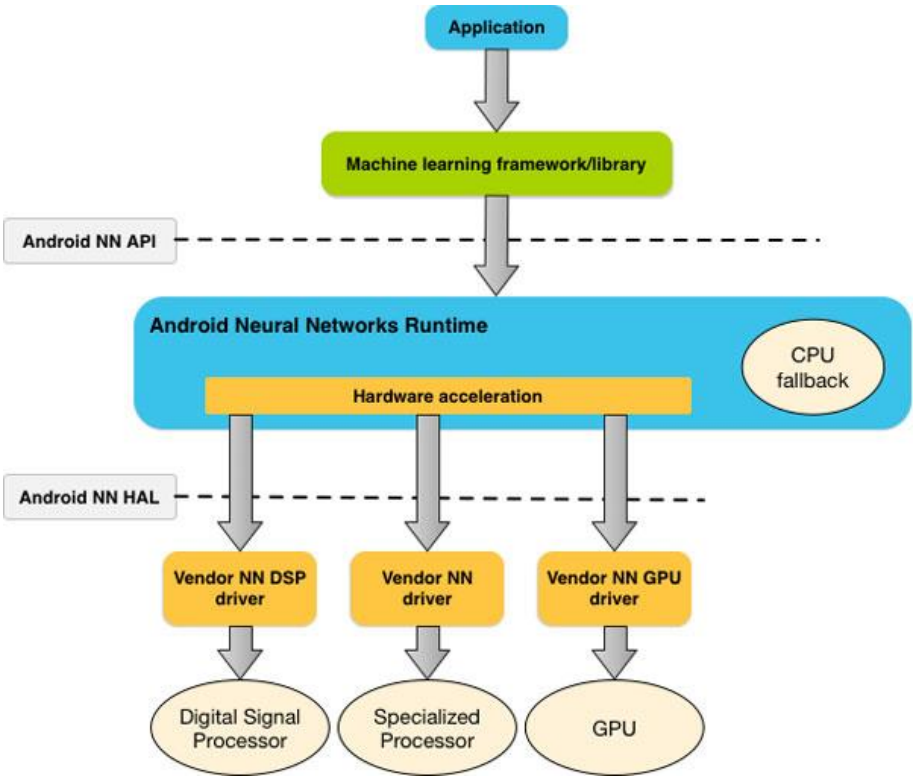

Рисунок 2.20 – Архітектура системи для Android Neural Networks API

NNAPI використовує чотири основні абстракції:

1) модель – це обчислювальний графік математичних операцій і постійних значень, отриманих у процесі навчання. Ці операції є специфічними для нейронних мереж. Вони включають двовимірну (2D) згортку, логістичну (сигмоподібну) активацію, випрямлену лінійну (ReLU) активацію тощо. Створення моделі є синхронною операцією. Після успішного створення його можна повторно використовувати в потоках і компіляціях. У NNAPI модель представлена як ANeuralNetworksModel екземпляр;

2) компіляція – це представлення конфігурації для компіляції моделі NNAPI в код нижчого рівня. Створення компіляції є синхронною операцією. Після успішного створення його можна повторно використовувати в потоках. У NNAPI кожна компіляція представлена як ANeuralNetworksCompilation екземпляр;

3) пам'ять представляє собою спільну пам'ять, файли, відображені в пам'яті, і подібні буфери пам'яті. Використання буфера пам’яті дозволяє середовищу

виконання NNAPI ефективніше передавати дані драйверам. Програма зазвичай створює один спільний буфер пам’яті, який містить усі тензори, необхідні для визначення моделі. Ви також можете використовувати буфери пам’яті для зберігання вхідних і вихідних даних екземпляра виконання. У NNAPI кожен буфер пам'яті представлений як ANeuralNetworksMemory екземпляр;

4) виконання: інтерфейс для застосування моделі NNAPI до набору вхідних даних і збору результатів. Виконання може виконуватися синхронно або асинхронно.

Для асинхронного виконання кілька потоків можуть очікувати на те саме виконання. Після завершення цього виконання всі потоки звільняються.

Висновки до розділу 2

Перед розробкою застосунку для розпізнавання мін типу ПФМ-1 було проведено аналіз технології комп’ютерного зору, використання у простих та складних прикладних задачах, також розглянуто статтю Айшварії Сінг як комп’ютерний зір «думає» та «доповнює» картину завдяки вхідним зображенням.

Розглянуто архітектури згорткових нейронних мереж, а саме:

– Convolutional Neural Network (CNN);
– R-CNN;
– Fast R-CNN та Faster R-CNN.

Також описано процеси навчання та наведено матеріали які зображують графічно навчання мереж.

Було розглянуто технологію NNAPI як апаратну підтримку для Android-застосунків. Використання такої технології як NNAPI допоможе застосунку для розпізнавання мін типу ПФМ-1 економити заряд батареї смартфону, що дозволить довше використовувати пристрій під час роботи застосунку в реальному часі.

3 СТВОРЕННЯ НАБОРУ ДАНИХ. НАВЧАННЯ НЕЙРОННИХ МЕРЕЖ

3.1 Створення набору даних для навчання нейронної мережі

Набір даних є ключовим елементом при навчанні нейронних мереж в вирішенні задачі розпізнавання об’єктів, оскільки це база, на якій ґрунтується робота системи. Для створення надійної моделі потрібно велика кількість даних, яка відповідає темі набору. Набір даних дозволяє моделі вивчати широкий спектр варіацій об'єктів, що підвищує її здатність до розпізнавання.

Перед навчанням нейронної мережі насамперед потрібно мати власний набір даних, або завантажити з відкритих ресурсів який підходить для обраної задачі. Головний об’єкт для розпізнавання – наземна міна типу ПФМ-1, тому було звернено до ресурсу Kaggle з метою знайти подібний набір даних та оцінити його якість.

У результаті при пошуку набору даних не було знайдено заздалегідь готового набору, на сайті Kaggle при пошуку видавало наступну інформацію:

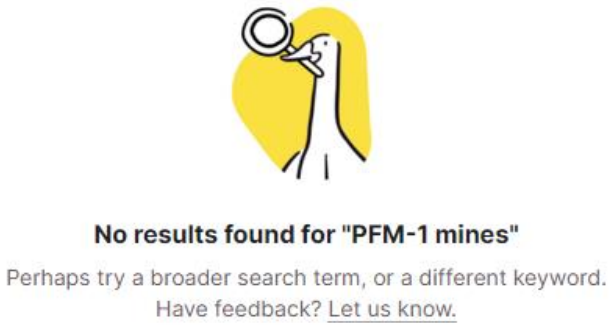

Рисунок 3.1 – Спроба знайти готовий набір даних для навчання нейронної мережі

Виходячи із даної відповіді можна зробити висновок о створенні власного набору даних, який буде повністю відповідати поставленій задачі. Для цього треба вирішити наступне питання – які зображення місцевості необхідні для розміщення на них мін ПФМ-1? У результаті було завантажено у відповідну директорію

“LAND_PICTURES” зображення полів, пустелі та лісу (де найчастіше всього розміщуються міни) та трохи зображень міста (але тут вони зустрічаються рідше усього). Приклад зібраних зображень зображено на рисунку 3.2:

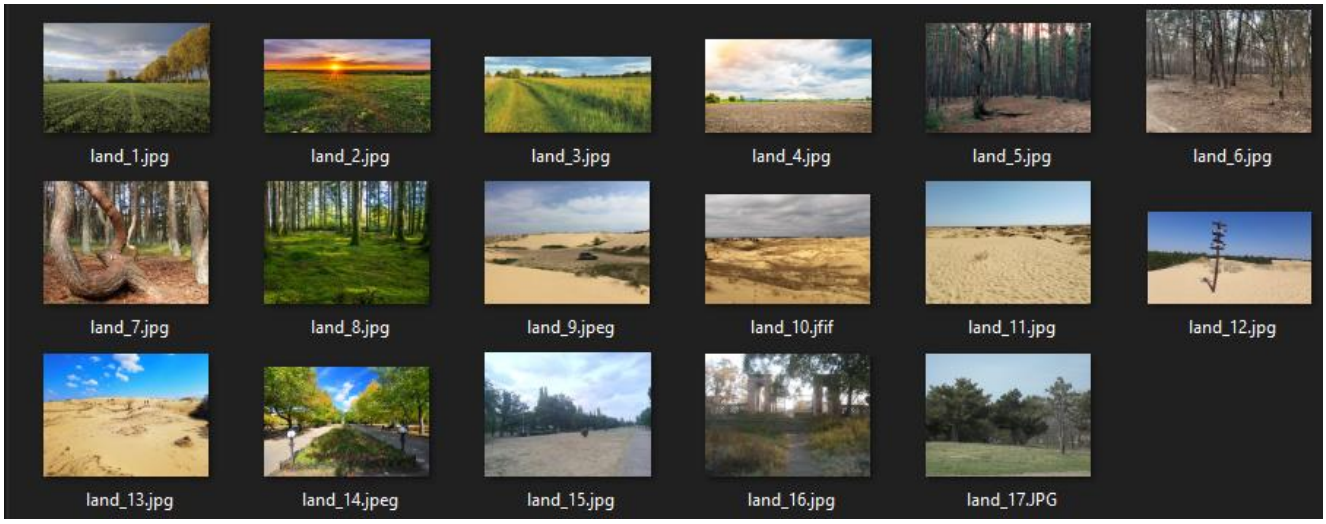

Рисунок 3.2 – Створений набір даних із зображенням місцевості

Після чого потрібно вирішити наступне завдання – зібрати готові зображення мін ПФМ-1 у “.png” форматі без фону для майбутнього розміщення їх на зображеннях. З відкритих ресурсів було знайдено зображення міни ПФМ-1 та завантажено в директорію під назвою “PNG_MINES”. Так як усі зображення формату jpeg, то вони не підходять для розміщення на заготовлених заздалегідь зображеннях місцевості. Для вирішення проблеми створюємо наступний скрипт, який приймає директорію, де розміщено зображення мін ПФМ-1, та обрізає їх перетворюючи у формат png, це досить швидко прискорює та автоматизує процес обробки зображень без втрати якості та довгої роботи у Photoshop:

```
import os
import rembg
from types import List
from PIL import Image
pfm1_images_path: str = "D:\\UNIVERSITY_WORKS\\2_KRM\\dataset\\+
NO_PNG_MINES\\"
```

```
output_path: str =
"D:\\UNIVERSITY_WORKS\\2_KRM\\dataset\\PNG_MINES\\mine-"
list_of_images: List[str] = os.listdir(pfm1_images_path)
i: int = 0
for input_image in list_of_images:
    input = Image.open(input_image)
    output = remove(input)
    output.save(output_path+f"{i}")
    i+=1
```

У результаті було отримано наступні зображення мін ПФМ-1:

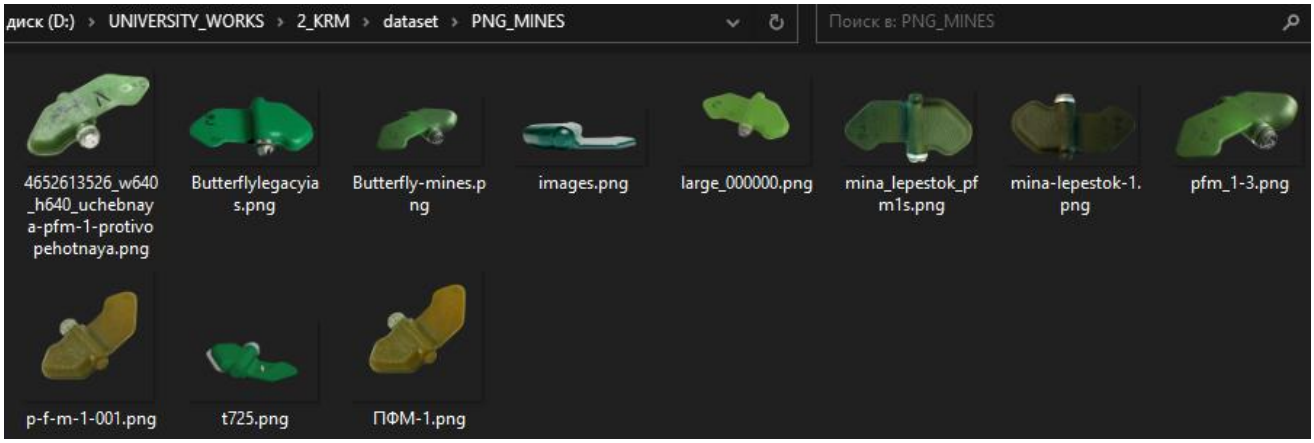

Рисунок 3.3 – Створений набір зображень мін ПФМ-1

Після створення двох відповідних наборів можна переходити до створення фінального набору даних – зображень місцевості із розміщеними мінами:

Рисунок 3.4 – Приклад одного зображення з набору даних

У результаті було отримано 9747 зображень за допомогою нейронної мережі DALL-E 3. На вхід подані зображення місцевості та мін ПФМ, на основі яких було згенеровано зображення.

Але одних зображень для нейронної мережі замало, для вирішення поставленої задачі було обрано метод навчання з вчителем. Навчання з вчителем використовує заздалегідь підготовлений набір, щоб навчити моделі отримувати бажаний результат. Цей навчальний набір даних включає вхідні та правильні вихідні дані, які дозволяють моделі навчатися з часом. Алгоритм вимірює свою точність за допомогою функції втрат, коригуючи, доки помилка не буде достатньо мінімізована.

Тому для реалізації даного навчання було додатково опрацьовано набір даних анотацією даних, для цього використовувався ресурс Computer Vision Annotation Tool [45]:

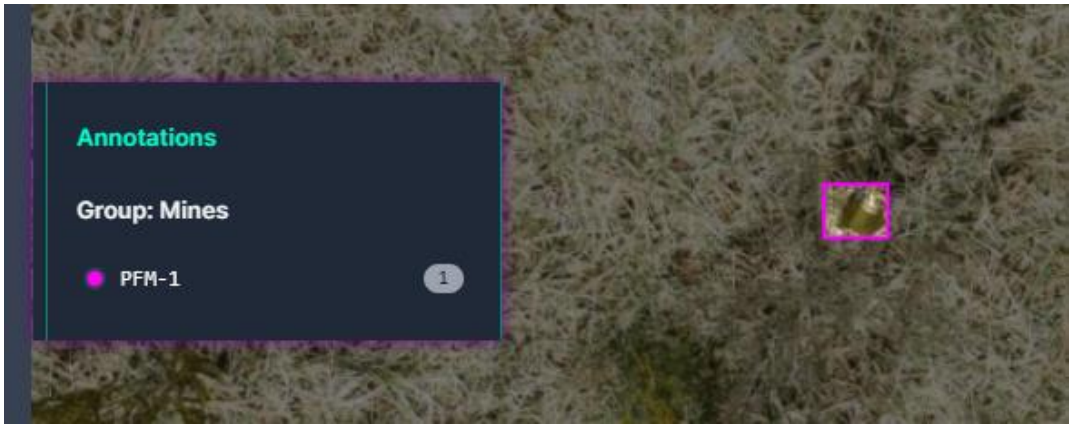

Рисунок 3.5 – Автоматизовний лейблінг мін ПФМ-1 на вхідних зображеннях з набору даних

У результаті на виході отримуємо наступні текстові файли:

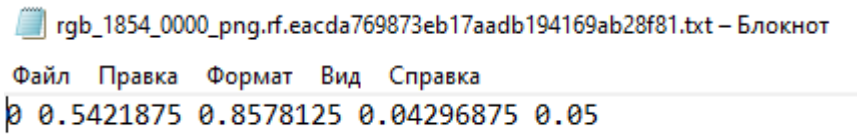

Рисунок 3.6 – Вхідні дані для навчання нейронної мережі одного із зображень

Так як розпізнавання мін – складна задача, тому що розміщення міни може бути в будь-якому положенні, тому при створенні набору даних було проведено аугментацію даних. Аугментація даних – це техніка штучного збільшення навчального набору шляхом створення модифікованих копій набору даних з використанням існуючих даних. Це включає внесення незначних змін до набору даних або використання глибокого навчання для створення нових точок даних [46]. Серед відомих методів аугментації було використано наступні:

– геометричні трансформації, тобто довільне перевертання, обрізання, обертання, розтягування та масштабування зображень;

– випадкова зміна каналів кольору RGB, контрастності та яскравості;

– видалення частини початкового зображення;

– змінено яскравість від -25% та +25%;

– шум збільшено до 5% пікселів.

Перед самим навчанням було поділено набір даних на наступні виборки (усього 9747 зображень):

1) набір для навчання мережі – 8597 зображень (92%);
2) набір для валідації – 397 зображень (4%);
3) набір для тестування – 393 зображення (4%).

Такий розподіл дозволить скорегувати параметри навчання нейронної мережі; дає нам змогу оцінити продуктивність моделі на незалежній вибірці та запобігти перенавчанню.

3.2 Навчання моделі CNN

Для навчання було створено першу модель. Це згорткова нейронна мережа з наступною архітектурою:

```
model = models.Sequential([
    layers.Conv2D(32,       (3,       3),       activation='relu',
input_shape=(image_size[0], image_size[1], 3)),
```

```
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
```

Дана мережа має 9 шарів:

1) вхідний шар «input_shape=(image_size[0], image_size[1], 3)» який визначає форму вхідних даних. В даному випадку використовується input_shape, який вказує розмір зображення, яке підходить до моделі. image_size[0] та image_size[1] відповідають ширині та висоті зображення, а 3 вказує на три канали (часто використовуються для зображень RGB);

2) згорткові шари (Conv2D) з функцією активації RELU. Послідовність Conv2D шарів із вказаними параметрами (кількість фільтрів, розмір ядра згортки) виконує згортку зображення, щоб вилучити особливості. Після цього застосовується функція активації ReLU, яка вводить нелінійність у мережу;

3) Pooling шари (MaxPooling2D). Ці шари здійснюють пулінг, зменшуючи розмірність отриманих карт ознак, зберігаючи при цьому найбільш важливі ознаки.

4) Flatten шар зменшує карту ознак до одновимірного масиву за допомогою цього шару для подальшого використання у повнозв’язному шарі;

5) повнозв’язний шар (Dense). Останній шар в архітектурі має один нейрон з активацією Sigmoid, яка використовується для бінарної класифікації (вихідне значення близьке до 0 або 1).

Для навчання було обрано 200 епох:

```
D:\UNIVERSITY_WORKS\2_KRM\venv\Scripts\python.exe D:\UNIVERSITY_WORKS\2_KRM\CNN_model\test.py
[INFO] loading images...
[INFO] compiling model...
[INFO] training head...
Train for 94 steps, validate on 752 samples
Train for 94 steps, validate on 752 samples
Epoch 1/200
94/94 [==============================] - 122s 817ms/step - loss: 0.3072 - accuracy: 0.8647 - val_loss: 0.1015 - val_accuracy: 0.9728
Epoch 2/200
94/94 [==============================] - 121s 789ms/step - loss: 0.1083 - accuracy: 0.9641 - val_loss: 0.0534 - val_accuracy: 0.9837
Epoch 3/200
94/94 [==============================] - 124s 756ms/step - loss: 0.0774 - accuracy: 0.9784 - val_loss: 0.0433 - val_accuracy: 0.9864
Epoch 4/200
94/94 [==============================] - 122s 784ms/step - loss: 0.0624 - accuracy: 0.9781 - val_loss: 0.0367 - val_accuracy: 0.9878
Epoch 5/200
94/94 [==============================] - 124s 791ms/step - loss: 0.0590 - accuracy: 0.9801 - val_loss: 0.0340 - val_accuracy: 0.9891
```

Рисунок 3.6 – Процес навчання згорткової нейронної мережі

На вхід було подано набір даних із зображеннями полів із мінами та без них, тому на виході після навчання було отримано наступні дані:

```
[INFO] evaluating network...
              precision    recall  f1-score   support

     no_mine       0.88      0.70      0.83       440
        mine       0.85      0.60      0.82       312

    accuracy                           0.80       752
   macro avg       0.72      0.81      0.75       752
weighted avg       0.73      0.79      0.78       752

[INFO] saving mine detector model...
[INFO] saving label encoder...
real    45m37.851s
user    396m43.701s
sys     406m53.058s
```

Рисунок 3.7 – Результат навчання згорткової нейронної мережі

Як можна побачити нейронна мережа навчалась 406 хвилин, дані, які було отримано у результаті показують, що точність класифікації для обох класів ("no_mine" та "mine") складає відповідно 88% та 85%. При цьому, модель визначила лише 70% зображень без мін і 60% зображень з мінами. Це означає, що вона помилково класифікувала частину зображень і не виявила всі міни.

Загальна точність моделі дорівнює 80%, що є гарним результатом через складність розпізнати міну на схожій за текстурою та кольором землі, на якої

розміщена дана міна. Показники f1-score, що враховують баланс між точністю та повнотою дорівнюють 83% для класу «no_mine» та 82% для класу «mine».

Перевіримо вручну узявши зображення із директорії «test»:

Рисунок 3.8 – Результат роботи розпізнавання та ідентифікації мін ПФМ-1

Виходячи із візуалізованих результатів – нейронна мережа змогла розпізнати тільки 3 міни із семи, звернемо увагу, що третя міна розпізнана із малим процентом (усього 57%, що дуже мало, коли мінімальний поріг це 70% та більше).

Рисунок 3.9 – Результат розпізнавання на зображенні із тестового набору даних

Рисунок 3.10 – Результат розпізнавання на зображенні із тестового набору даних

Тому із отриманих візуалізованих та текстових даних можна зробити висновок, що модель може потребувати додаткових налаштувань чи оптимізації для забезпечення кращої точності та здатності правильно розпізнавати міни на вхідному зображенні.

3.3 Навчання моделі YOLOv8, 50 епох

Для створення/навчання моделі на архітектурі YOLOv8 необхідно підготувати середовище розробки. Тому для початку було створено проект з наступною архітектурою:

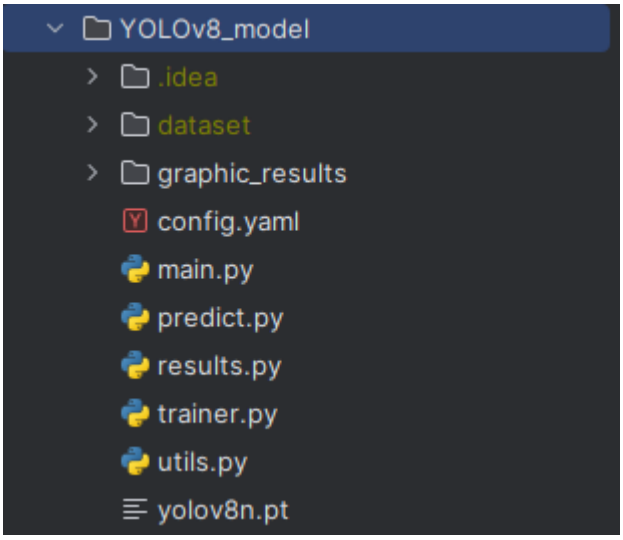

Рисунок 3.11 – Структура проекту для навчання моделі

Встановлюємо у віртуальне оточення бібліотеку ultralytics, яка має усі необхідні інструменти для налаштування нейронної мережі:

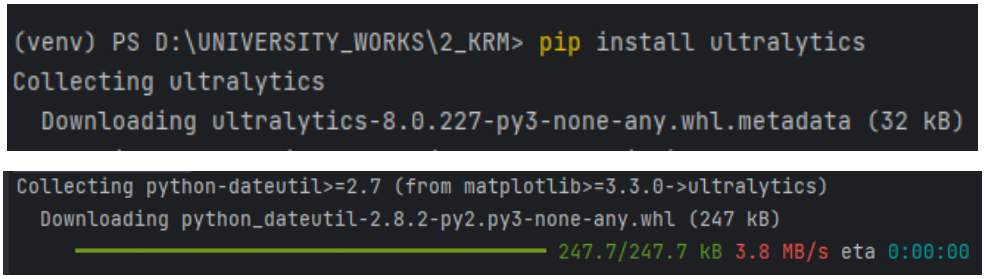

Рисунок 3.12 – Завантаження необхідних залежних бібліотек для роботи ultralytics

Далі було завантажено модель «yolov8n», де n – позначка що означає nano-версію, більш облегшена, далі буде наведено таблицю з порівнянням версій 8 покоління YOLO щоб зрозуміти різницю між nano, small, medium, large, extra-large.

Таблиця 3.1 – Порівняння моделей YOLOv8 за розміром

Модель	Розмір	mAP	Швидкість на відеокартах	Кіл-сть параметрів	FLOPs
YOLOv8n	640	37.3	80.4	3.2	8.7
YOLOv8s	640	44.9	128.4	11.2	28.6
YOLOv8m	640	50.2	234.7	25.9	78.9
YOLOv8l	640	52.9	375.2	43.7	165.2
YOLOv8x	640	53.9	479.1	68.2	257.8

Де FLOPs – це кількість операцій з плаваючою комою, які може виконати нейронна мережа за секунду. А літери після назви означають розмір моделі (n – nano, s – small, m – medium, l – large, x – eXtra large).

Наступним кроком йде налаштування файлу конфігурації навчання моделі, для цього створюється файл із назвою «config.yaml», де описується наступні дані:

1) шлях до директорії із набором даних (path);

2) шлях до тренувальних даних (train path, в даному випадку це 8957 зображень);

3) шлях до валідаційних даних (val path, в даному випадку це 397 зображень);

4) шлях до перевірочних(тестових) даних (train path, в даному випадку це 397 зображень);

5) назва класів, які нейронна мережа повинна розпізнавати, в полі names є тільки одна назва – pfm-1, які нейронна мережа буде вчитися ідентифікувати.

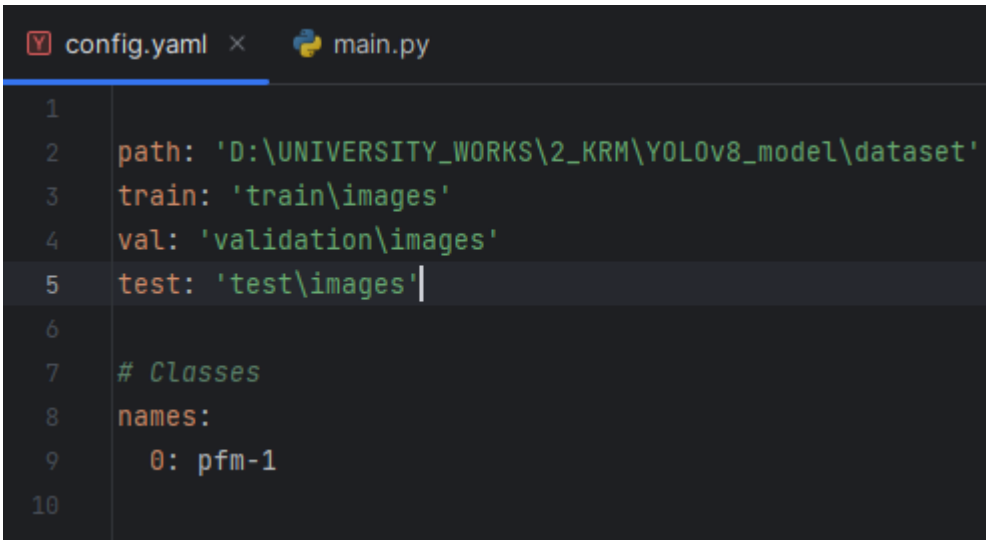

Рисунок 3.13 – YAML-конфігураційний файл для налаштування моделі

Далі переходимо до файлу «main.py», де задаємо наступний код для навчання нейронної мережі:

```python
from ultralytics import YOLO

if __name__ == "__main__":
    # Load a model
    model = YOLO("yolov8n.yaml")

    # Using the model
    results = model.train(data="config.yaml", epochs=50)
```

Рисунок 3.14 – Код для запуску навчання

Завантажуємо з офіційного сайту pre-trained модель YOLOv8 nano, та далі запускаємо навчання з 50 епохами на локальному пристрої, характеристики якого були описані у п.п. 1.3.

Після чого запускаємо код та отримаємо наступне:

```
D:\UNIVERSITY_WORKS\2_KRM\venv\Scripts\python.exe D:\UNIVERSITY_WORKS\2_KRM\YOLOv8_model\main.py

                   from  n    params  module                                       arguments
  0                  -1  1       464  ultralytics.nn.modules.conv.Conv             [3, 16, 3, 2]
  1                  -1  1      4672  ultralytics.nn.modules.conv.Conv             [16, 32, 3, 2]
  2                  -1  1      7360  ultralytics.nn.modules.block.C2f             [32, 32, 1, True]
  3                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  4                  -1  2     49664  ultralytics.nn.modules.block.C2f             [64, 64, 2, True]
  5                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  6                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  7                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  8                  -1  1    460288  ultralytics.nn.modules.block.C2f             [256, 256, 1, True]
  9                  -1  1    164608  ultralytics.nn.modules.block.SPPF            [256, 256, 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  1    148224  ultralytics.nn.modules.block.C2f             [384, 128, 1]
 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  1     37248  ultralytics.nn.modules.block.C2f             [192, 64, 1]
 16                  -1  1     36992  ultralytics.nn.modules.conv.Conv             [64, 64, 3, 2]
 17           [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                  -1  1    123648  ultralytics.nn.modules.block.C2f             [192, 128, 1]
 19                  -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
 20            [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  1    493056  ultralytics.nn.modules.block.C2f             [384, 256, 1]
 22       [15, 18, 21]  1    897664  ultralytics.nn.modules.head.Detect           [80, [64, 128, 256]]
YOLOv8n summary: 225 layers, 3157200 parameters, 3157184 gradients, 8.9 GFLOPs
```

Рисунок 3.15 – Архітектура YOLOv8, яка використовувалась для навчання моделі

Мережа складається із 225 шарів, які містять у сумі 3011043 параметрів. Архітектура побудована з послідовності шарів з використанням різних типів згорток, конкатенації та блоків згорткових операцій. Мережа має глибоку структуру, що дозволяє їй виявляти об'єкти різних розмірів та складності на зображеннях з високою швидкістю, використовуючи 8.9 GFLOPs обчислень як і вказано у таблиці 3.1 з характеристиками моделей YOLOv8.

Після формування архітектури нейронна мережа переходить до навчання:

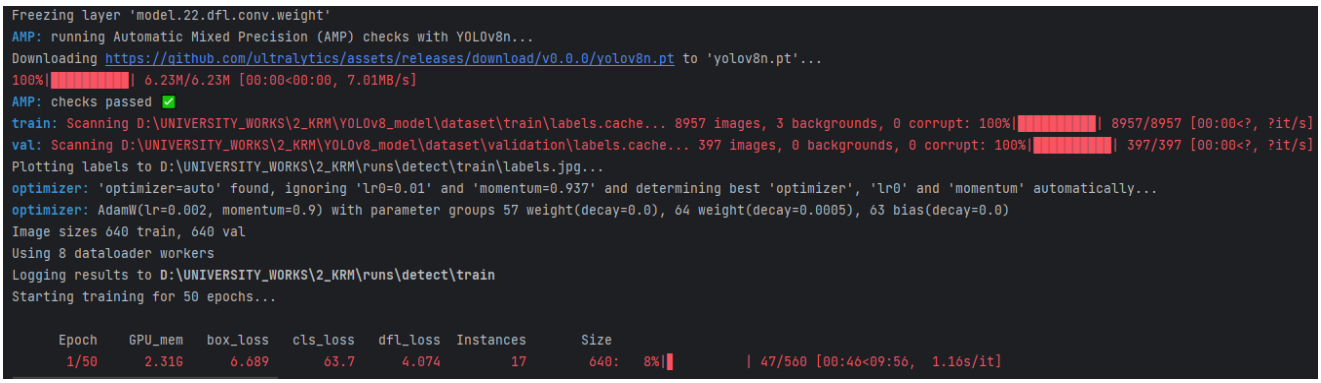

Рисунок 3.16 – Процес навчання нейронної мережі

В даному навчанні можна в реальному часі відслідкувати наступні параметри навчання:

1) box_loss – значення, яке вказує на середню помилку у прогнозуванні координат обрамлень (bounding boxes) для об'єктів на зображенні. На першій епосі дане значення достатньо високе;

2) cls-loss – середня помилка в класифікації об'єктів;

3) dfl-loss – помилка відстеження (defocus loss) або втрата чіткості у виявленні об'єктів;

4) Instances – кількість виявлених об'єктів моделлю на тестовому або валідаційному наборі даних;

5) Size – розмір вхідного зображення, для навчання нейронної мережі було подано зображення розміром 640x640.

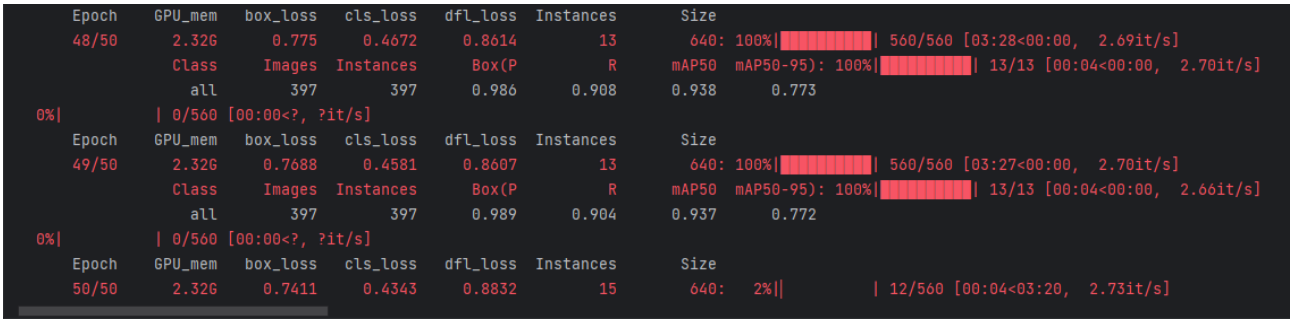

Рисунок 3.17 – Останні 3 епохи навчання нейронної мережі

Як можна побачити із наступних даних фінальної епохи – середня помилка у прогнозуванні координат об’єкту, який треба розпізнати став значно менше, що свідчить про підвищену точність, те ж саме можна сказати про інші параметри, середня помилка із значення 63.7 впала до 0.43 на останній епосі, значення помилки відстеження теж впало з 4.074 до 0.8832, що є гарним результатом, останнім кроком при навчанні була валідація даних:

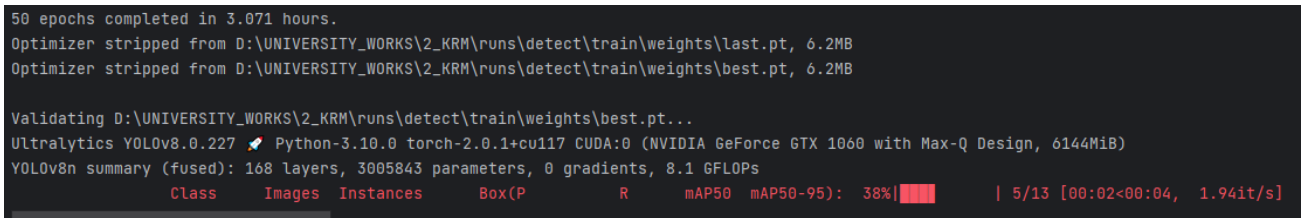

Рисунок 3.18 – Валідація даних

Із результатів можна побачити, що модель навчалась 3 години та було збережено два оптимізатори – найкращий та останній створений.

Як можна побачити, впродовж навчання нейронна мережа на кожній епосі знижала свою помилку що на тренувальному, що на валідаційному наборі даних. Проведемо аналітику отриманої моделі та подивимося на графіки навчання:

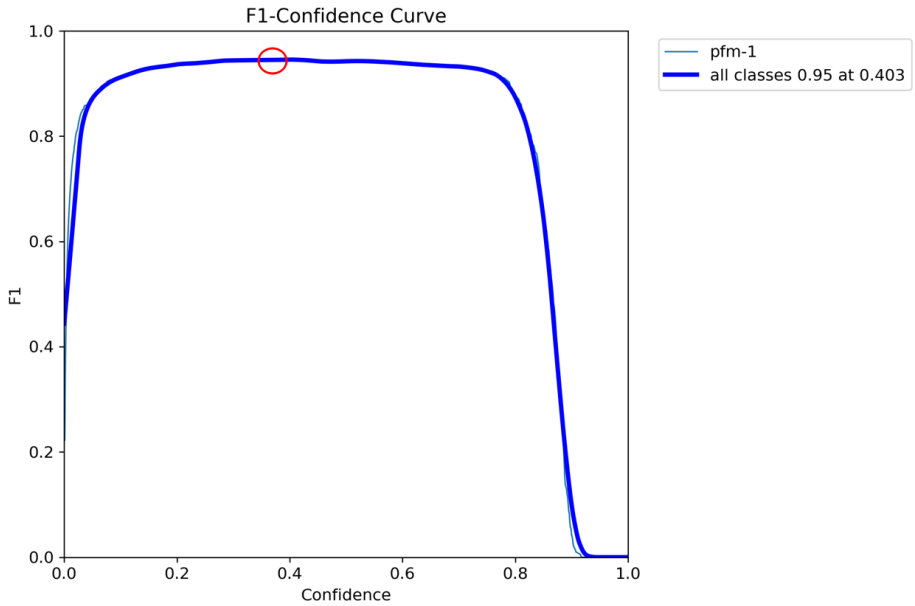

Рисунок 3.19 – Крива довіри F1 (F1-Confidence Curve)

Було отримано графік, який відображає залежність міри F1 від рівня впевненості моделі (confidence level). Крива довіри F1 показує найкращу оцінку F1 0,95 із порогом довіри 0,403 (виділено червоним кружком).

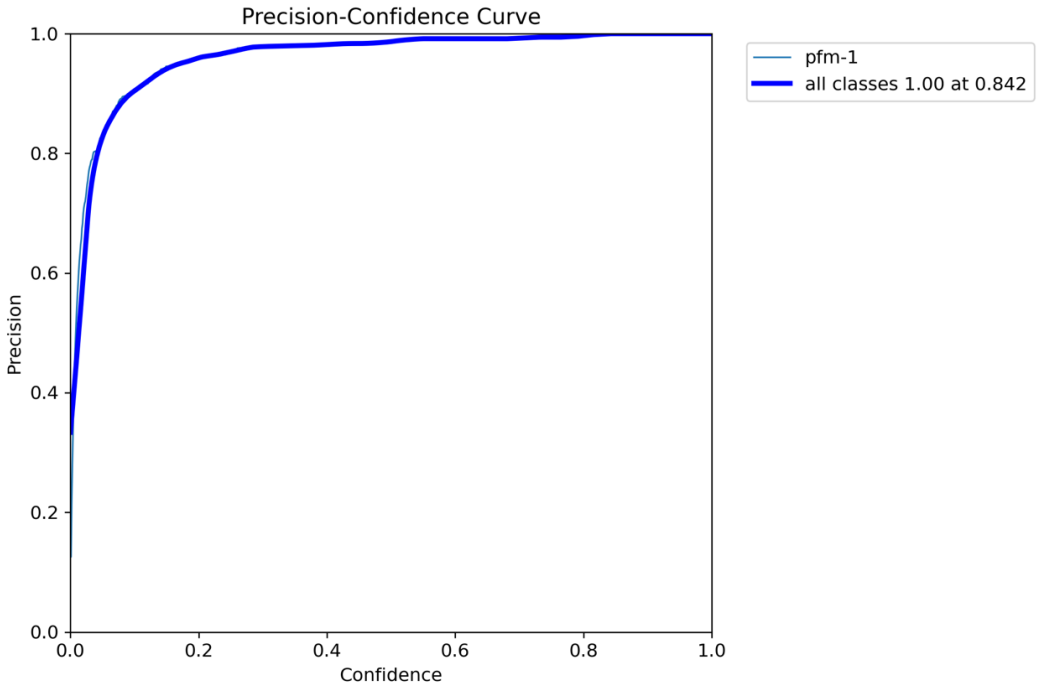

Рисунок 3.20 – Крива точності від впевненості показує найкращу оцінку точності 1 із порогом довіри 0.842

Виведемо графіки точності навчання моделі на тестовому та валідаційному наборі даних:

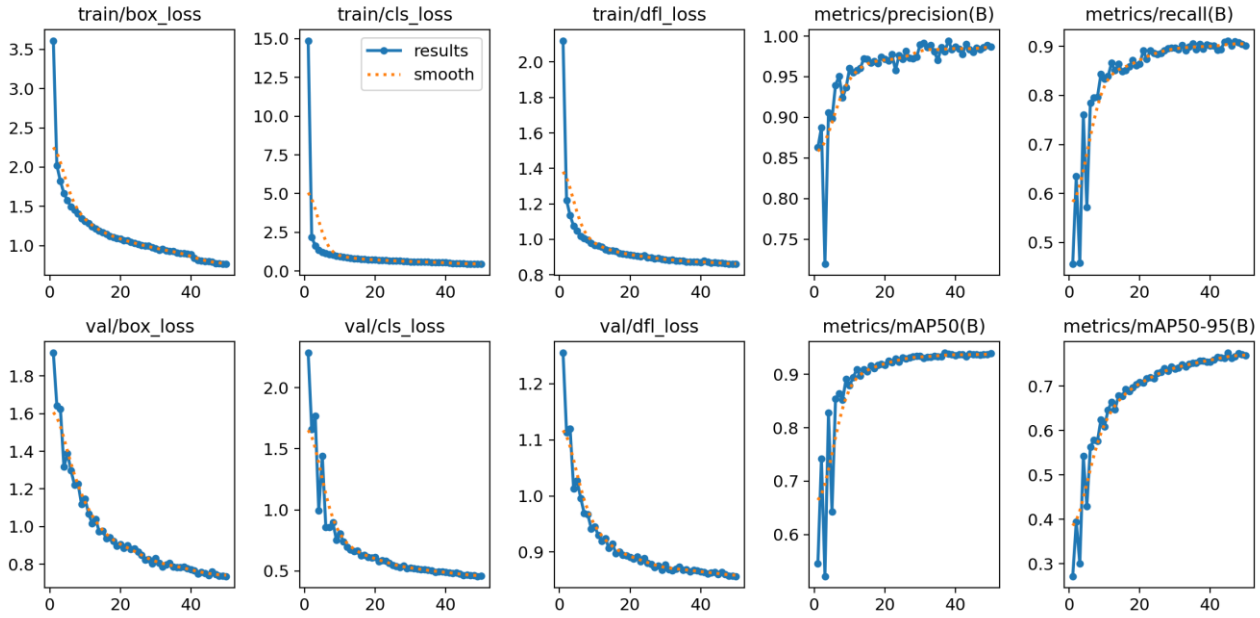

Рисунок 3.21 – Графіки навчання моделі на 50 епохах

Значення metrics/precision(B) (точність) визначає, який відсоток мін ПФМ-1, визначених моделлю як точний клас, дійсно є об’єктами цього класу, значення metrics/recall(B) (повнота) визначає, який відсоток дійсних об'єктів певного класу було виявлено моделлю. Точність даних вимірювань на 50 епохах зростає. Показник mAP50(B) вимірює середню точність при використанні порогу впевненості 50% для виявлення об'єктів, mAP50-90(B) вимірює середню точність в діапазоні від 50% до 95% порогу впевненості. З даних графіків також можна помітити зростання точності детектування мін ПФМ-1.

Після чого було вирішено запустити тестування нейронної мережі на тестових зображеннях (397 зображень) після валідації даних. Спочатку завантажуємо звичайні тестові зображення без аугментації:

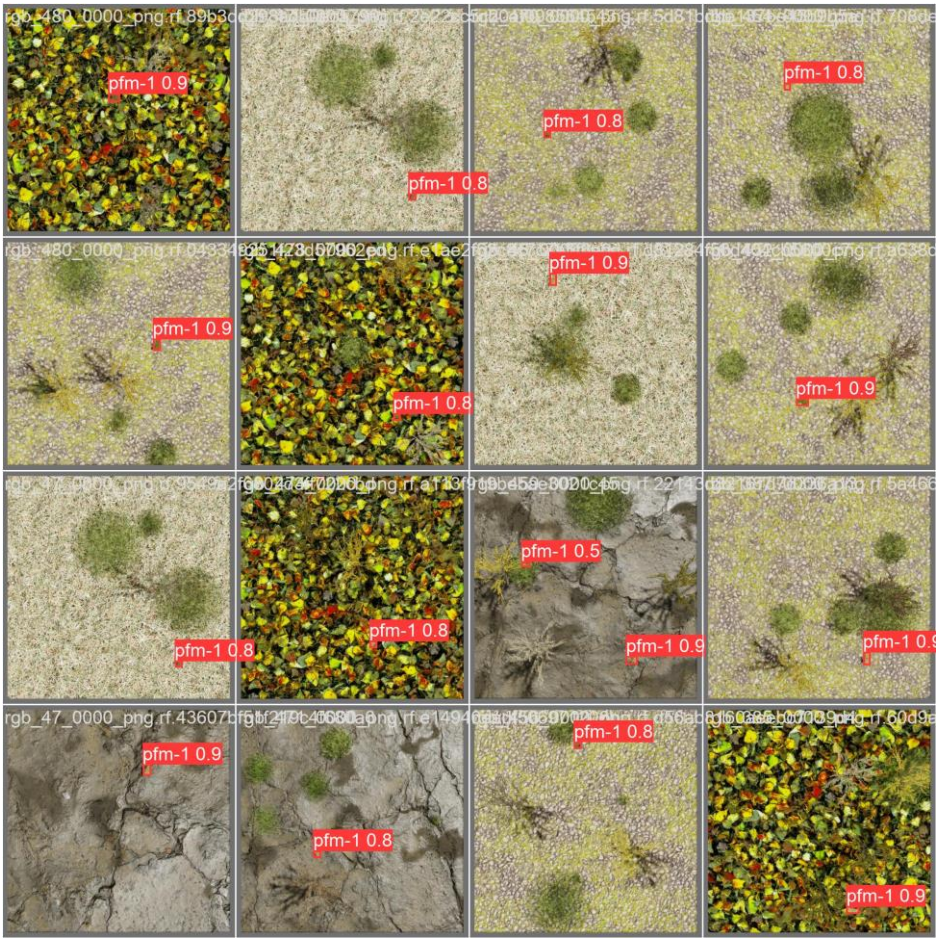

Рисунок 3.22 – Тестування моделі на тестових даних, середня точність дорівнює 0.9 при максимальній точності 1

Далі завантажимо аугментовані зображення, тобто оброблені чорно-білим фільтром, змішуємо їх (тобто беремо 4 зображення та поєднуємо їхні частини), та змінюємо яскравість зображення від -25% до +25% щоб запобігти перенавчанню.

Як можна побачити – модель гарно справляється із аугментованими зображеннями, що свідчить про відсутність перенавчання. Максимальна точність 95% на тестових даних. Далі використаємо смартфон та спробуємо зняти відео монітору де зображено міни ПФМ-1 з валідаційного набору даних.

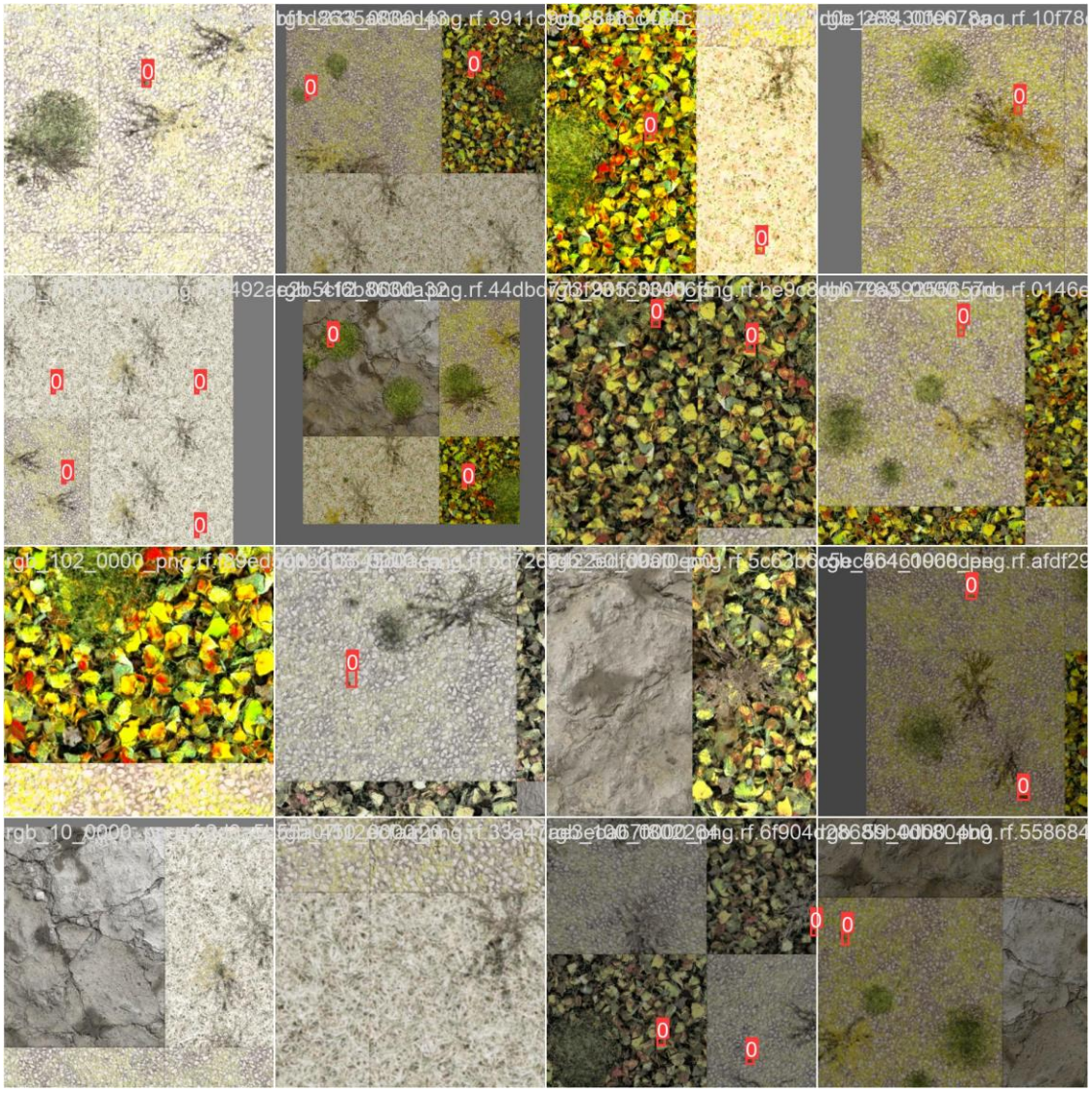

Рисунок 3.23 – Розпізнавання мін ПФМ-1 на аугментованих зображеннях

Запускаємо тестування та передаємо у параметрах створену модель нейронної мережі:

```
D:\UNIVERSITY_WORKS\2_KRM\venv\Scripts\python.exe D:\UNIVERSITY_WORKS\2_KRM\YOLOv8_model\predict_on_video.py

0: 640x384 1 pfm-1, 67.8ms
Speed: 5.0ms preprocess, 67.8ms inference, 7.0ms postprocess per image at shape (1, 3, 640, 384)

0: 640x384 (no detections), 14.0ms
Speed: 4.0ms preprocess, 14.0ms inference, 2.0ms postprocess per image at shape (1, 3, 640, 384)

0: 640x384 1 pfm-1, 12.0ms
Speed: 3.0ms preprocess, 12.0ms inference, 3.0ms postprocess per image at shape (1, 3, 640, 384)
```

Рисунок 3.24 – Процес обробки відео для розпізнавання мін ПФМ-1

У результаті на дуже складному фреймі було зупинено відео та зроблено скріншот, де людське око не може розгледіти міну, бо вона дуже схожа за кольоровою палітрою та ховається серед листя:

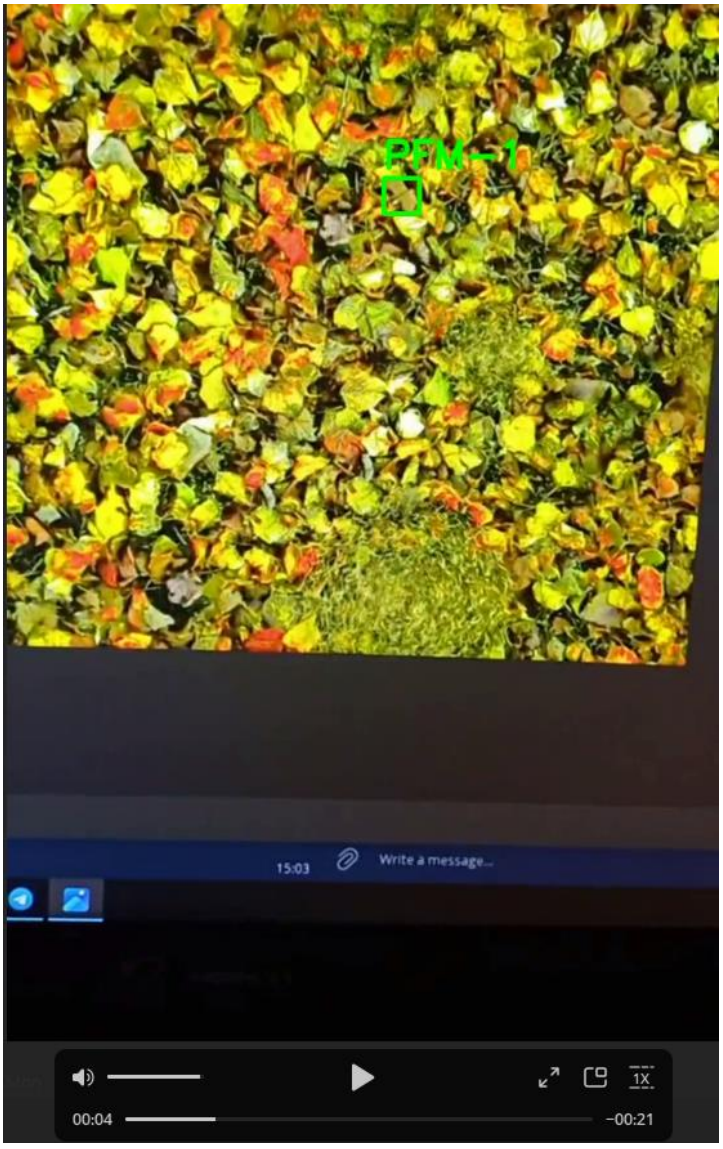

Рисунок 3.25 – Кадр із відео, де нейронна мережа розпізнає міну ПФМ-1 серед листя

Аналізуючи відео можна помітити, що іноді нейронна мережа поводить себе дивно та помічає міни там, де їх нема в дуже рідкісних випадках, але загалом працює без перебоїв.

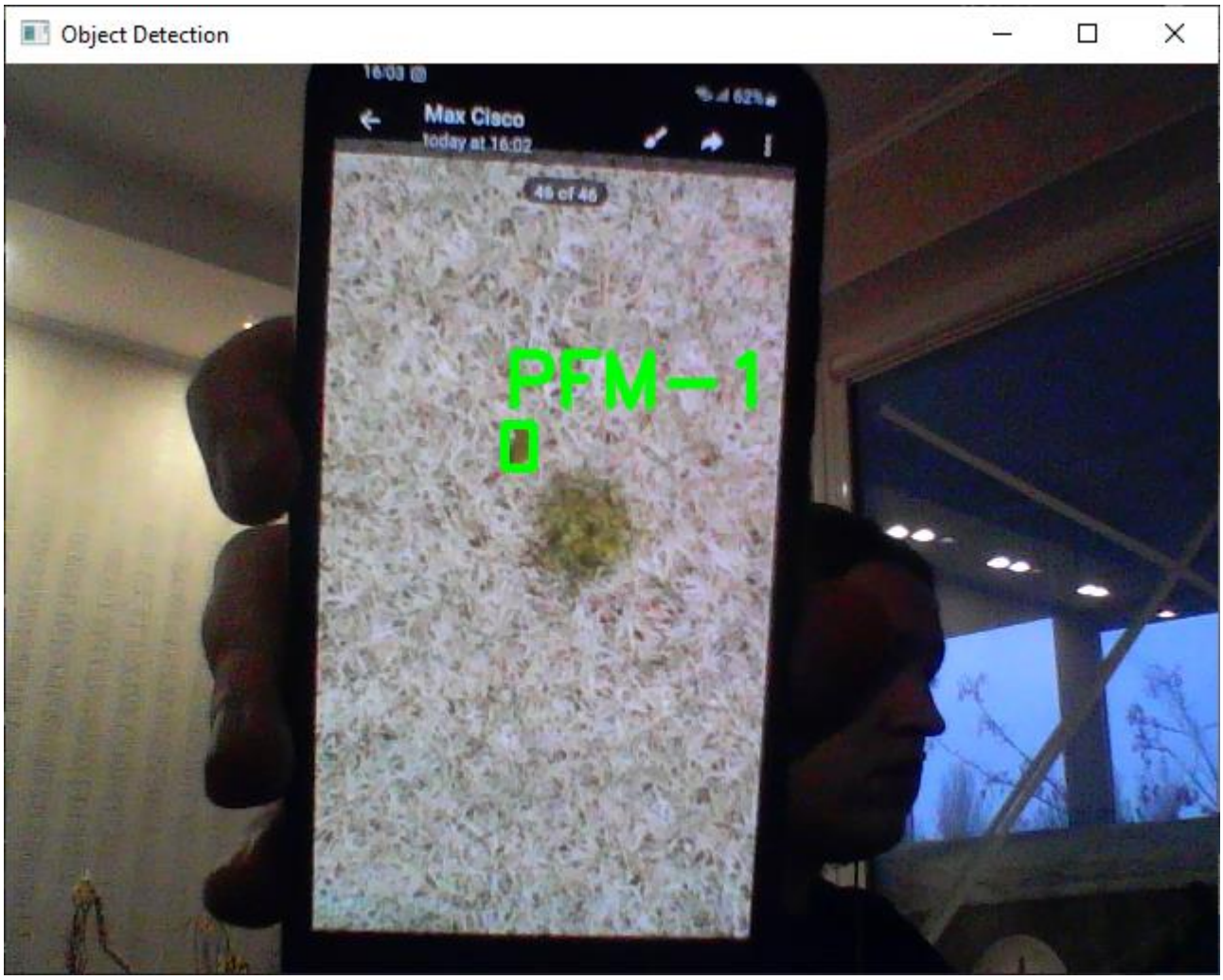

Рисунок 3.26 – Тестування роботи моделі використовуючи веб-камеру

Протестувавши вхідні зображення з веб-камери можна зробити висновок, що окрім правильності навчання моделі ще дуже важливу роль у даній задачі грає якість вхідного зображення. Наприклад, знімаючи відео на смартфоні із гарною якістю (1200x700) точність моделі складає близько 90%, а якщо намагатися зробити це через веб-камеру, де розмір вхідного зображення 480x640 та зменшена якість, то точність падає до 60-70%. Тому при розробці потрібно враховувати подібні моменти та розраховувати точність більше детально.

Максимальна точність на тестових даних складає 95%, але при спробі на реальних прикладах помітно що точність знижується через низку факторів (якість,

освітлення, позиція). Тому було вирішено модифікувати архітектуру YOLOv8n для отримання кращих результатів розпізнавання.

3.4 Модифікація моделі YOLOv8n, 100 епох, квантування нейронної мережі

Після створення першої згорткової нейронної мережі можна було побачити, що звичайна НМ без модифікацій має в середньому результат 50-60% на дуже простих даних. Тому було вирішено перейти до вже готової архітектури YOLOv8n і без налаштувань навчити модель на 50 епохах. Це дало значно кращі результати, із чого можна зробити висновок, що використання вже натренованої моделі ефективніше, ніж створювати власну згорткову нейронну мережу.

Модифікація та оптимізація потрібна, тому що найчастіше під час імпорту моделі на пристрій, який буде її обробляти, перед розробником постане безліч проблем. Попередня модель показала непогані результати, але все ще потребує довчання та оптимізації.

Для чого потрібна оптимізація:

1) збільшення пропускної здатності (зниження затримки), корисно як для хмарних сервісів, так і для edge-девайсів у вигляді мобільних пристроїв;

2) розгортання моделей на edge-пристроях з обмеженнями по обробці, пам'яті та/або енергоспоживанню;

3) зменшення розміру моделі для прискореного оновлення моделі та зниження витрат на зберігання моделей;

4) оптимізація моделі корисна для обладнання з обмеженнями або оптимізацією для операцій з фіксованою точкою;

5) для апаратних прискорювачів спеціального призначення.

Існує декілька видів оптимізації:

1) Pruning – усунення частини параметрів нейронної мережі;

2) Quantization – зменшення точності оброблюваних типів даних;

3) Knowledge distillation – оновлення топології вихідної моделі до більш ефективної, зі зменшеною кількістю параметрів і швидшим виконанням;

4) Weight clustering – скорочення кількості унікальних параметрів у вагах моделі;

5) OpenVino, TensorRT – бібліотеки, за допомогою яких можна оптимізувати моделі.

Серед усіх цих методів було обрано квантування моделі. Квантування моделі – це популярний метод оптимізації глибокого навчання, під час якого дані моделі – як параметри мережі, так і активації – перетворяться з подання з плаваючою комою на подання з більш низькою точністю, наприклад, із використанням 8-бітних цілих чисел.

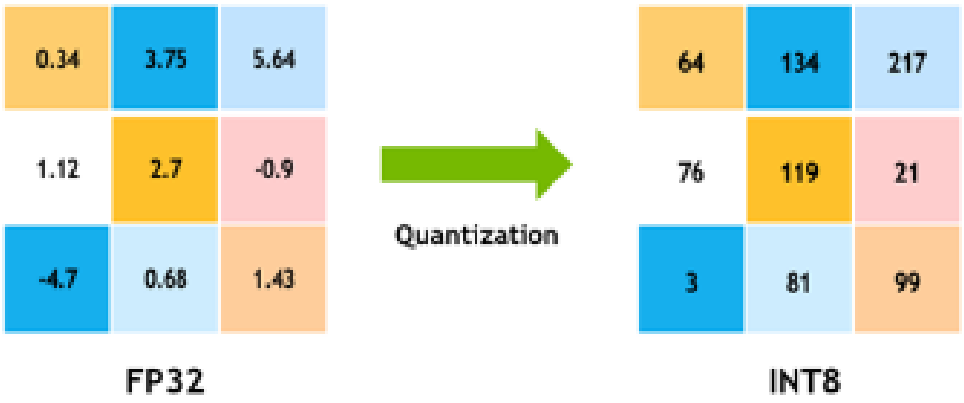

Рисунок 3.27 – Процес квантування max-pooling

Даний метод дає наступні переваги:

– під час опрацювання 8-бітних цілочисельних даних графічні процесори NVIDIA використовують швидші та дешевші 8-бітові тензорні ядра для обчислення операцій згортання і множення матриць. Це дає більшу пропускну здатність обчислень;

– переміщення даних із пам’яті в обчислювальні елементи (потокові мультипроцесори в графічних процесорах NVIDIA) вимагає часу й енергії, а також

виділяє тепло. Зниження точності даних активації та параметрів із 32-бітних чисел із плаваючою комою до 8-бітних цілих чисел призводить до 4-кратного скорочення даних, що економить електроенергію і знижує виділене тепло;

– зменшення обсягу пам’яті означає, що модель потребує менше місця для зберігання, менше параметрів для оновлення, використання кешу вище тощо.

Існують різні типи квантування, в даному випадку було використано float32 до float16. Головне пам’ятати, що квантування - це наближення.

Загалом, чим ближче наближення, тим менше зниження точності можна очікувати. Якщо все квантувати до float16, то пам’ять скоротиться вдвічі і, ймовірно, точність не буде втрачено, але й не отримаємо в результаті дуже сильного прискорення. З іншого боку, квантування за допомогою int8 може призвести до набагато швидшої обробки, але точність результатів, імовірно, буде гіршою. У деяких випадках це навіть не спрацює і може знадобитися навчання з урахуванням квантування.

Відтепер навчання буде проходити на Google Colabaratory:

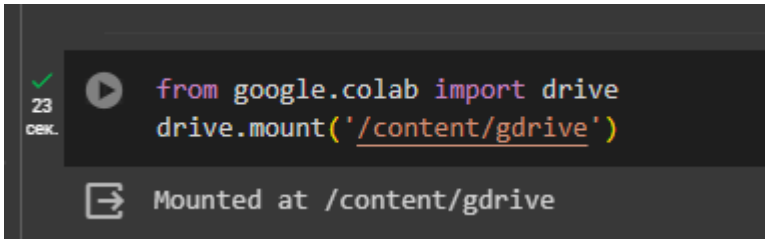

Рисунок 3.28 – Підключаємо власний Google Drive

Після цього створюємо все як у попередньому проекті та завантажуємо також набір даних:

Рисунок 3.29 – Завантаження тренувальних та валідаційних зображень разом із анотаціями

Після цього створюємо файл конфігурації для навчання нейронної мережі та завантажуємо його на диск:

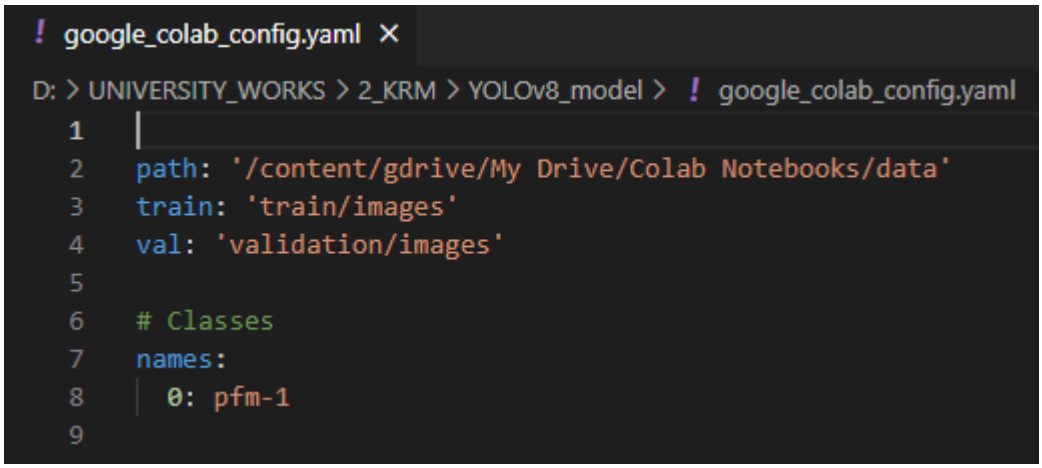

Рисунок 3.29 – Файл конфігурації для навчання

Далі встановлюємо необхідні бібліотеки для оточення та пишемо необхідний код для навчання:

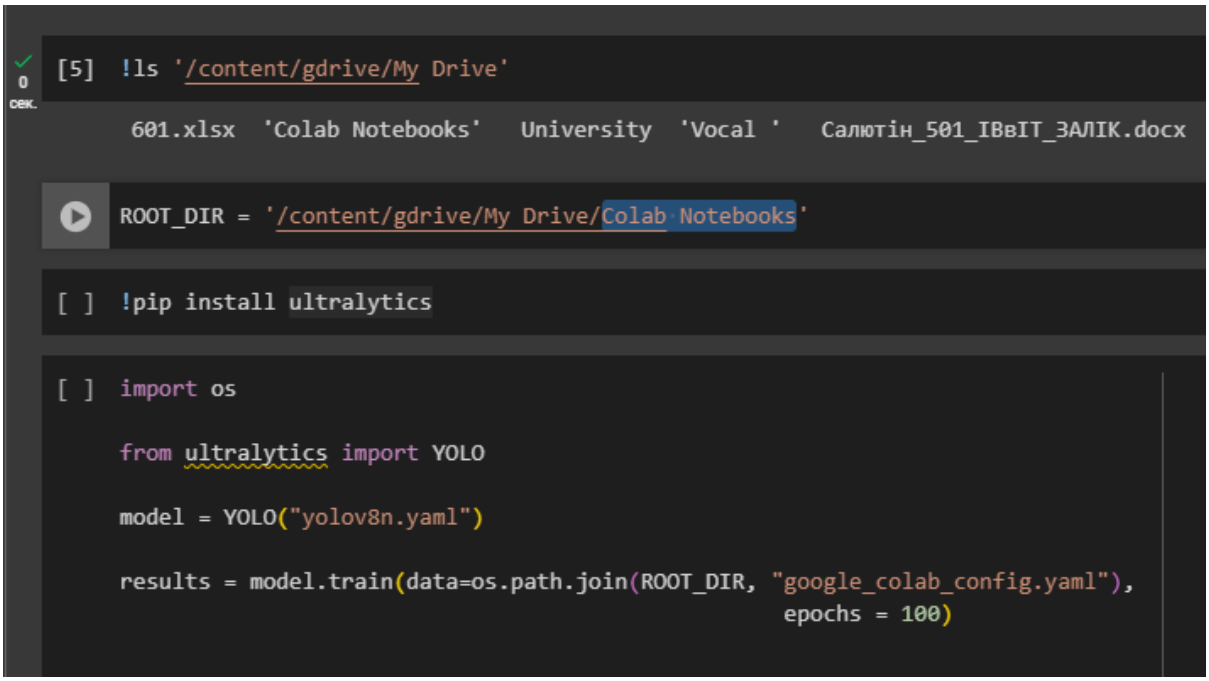

Рисунок 3.30 – Код для створення моделі у Google Colabaratory

Після чого запускаємо навчання нейронної мережі:

```
Freezing layer 'model.22.dfl.conv.weight'
AMP: running Automatic Mixed Precision (AMP) checks with YOLOv8n...
AMP: checks passed ✅
train: Scanning E:\project_nn\data\train\labels.cache... 8957 images, 3 backgrounds, 0 corrupt: 100%|██████████| 8957/8957 [00:00<?, ?it/s]
val: Scanning E:\project_nn\data\validation\labels.cache... 397 images, 0 backgrounds, 0 corrupt: 100%|██████████| 397/397 [00:00<?, ?it/s]
Plotting labels to runs\detect\train5\labels.jpg...
  0%|          | 0/560 [00:00<?, ?it/s]optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum' automatically...
optimizer: SGD(lr=0.01, momentum=0.9) with parameter groups 77 weight(decay=0.0), 84 weight(decay=0.0005), 83 bias(decay=0.0)
100 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      1/100      8.32G      6.609      69.09      4.016         15        640:   2%||          | 11/560 [00:54<43:33,  4.76s/it]
```

Рисунок 3.31 – Процес навчання моделі

Навчившись було отримано наступне:

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    100/100       2.3G     0.6211     0.3445     0.8257         13        640: 100%|██████████| 560/560 [03:05<00:00,  3.02it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 13/13 [00:04<00:00,  3.00it/s]
                   all        397        397      0.977      0.919      0.951      0.799

100 epochs completed in 5.518 hours.
Optimizer stripped from runs\detect\train\weights\last.pt, 6.3MB
Optimizer stripped from runs\detect\train\weights\best.pt, 6.3MB

Validating runs\detect\train\weights\best.pt...
Ultralytics YOLOv8.0.229 🚀 Python-3.10.11 torch-2.1.2+cu121 CUDA:0 (NVIDIA GeForce GTX 1060 6GB, 6144MiB)
YOLOv8n summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 13/13 [00:04<00:00,  2.71it/s]
                   all        397        397      0.977      0.919      0.951      0.799
Speed: 0.3ms preprocess, 4.7ms inference, 0.0ms loss, 1.2ms postprocess per image
Results saved to runs\detect\train

Process finished with exit code 0
```

Рисунок 3.32 – Результати навчання моделі

При 100 епохах нейронна мережа навчалася 5 з половиною годин, модель важить 6.3 мегабайти. Передивимося процес навчання, щоб побачити графіки:

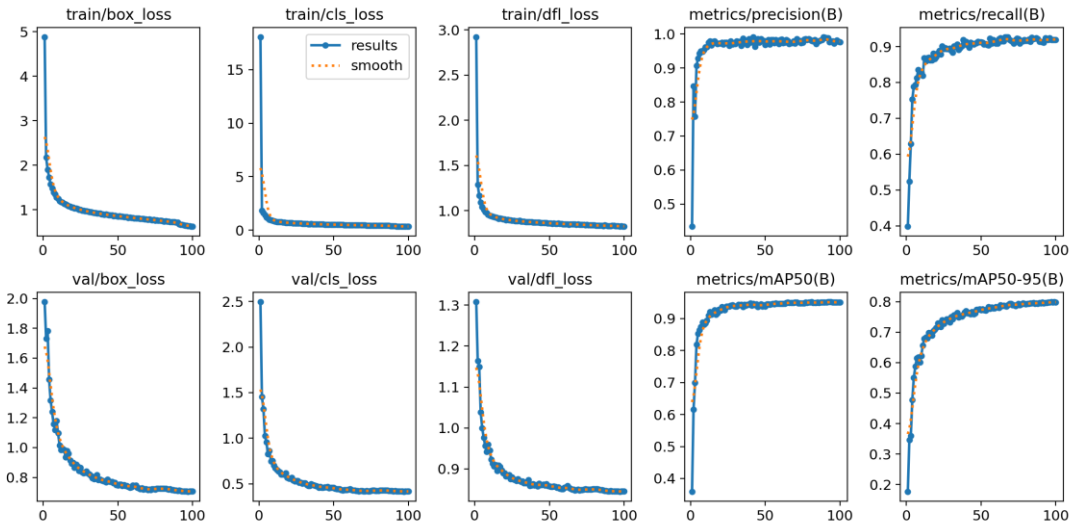

Рисунок 3.33 – Графіки процесу навчання моделі

Порівнявши значення на фінальних епохах можна отримати наступні дані – при 50 епохах втрата розпізнавання об’єкту на зображенні (box_loss) дорівнювала 0.766, при 100 епохах помилка знизилась до 0.62113.

На усіх інших графіках також помітно покращення, квантування прискорило навчання нейронної мережі та дозволило не втратити точність на 100 епохах, а навіть навпаки покращити точність розпізнавання. Квантування допомогло покращити навчання використовуючи YOLOv8 nano натреновану модель, однак одних графіків замало, тому було перевірено роботу нейронної мережі за допомогою тестового набору даних, веб-камери та відео, яке буде надано нейронній мережі для визначення мін на зображеннях.

Перевіримо даний набір використовуючи веб-камеру пристрою:

Рисунок 3.34 – Перевірка роботи нейронної мережі використовуючи веб-камеру з низькою роздільною здатністю картинки

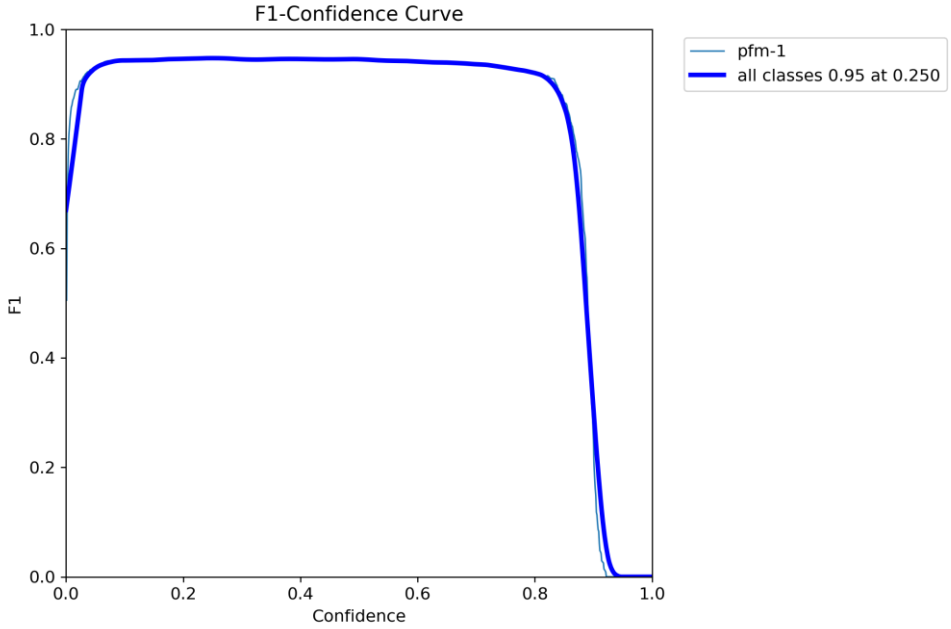

Рисунок 3.35 – Крива довіри від впевненості показує найкращу оцінку точності 0.95 із порогом довіри 0.25

Як можна побачити, рівень довіри понизився у порівнянні із минулою моделлю (при однакових F1=0.95), це пов’язано із квантуванням даних під час навчання, що може знизити точність або впевненість.

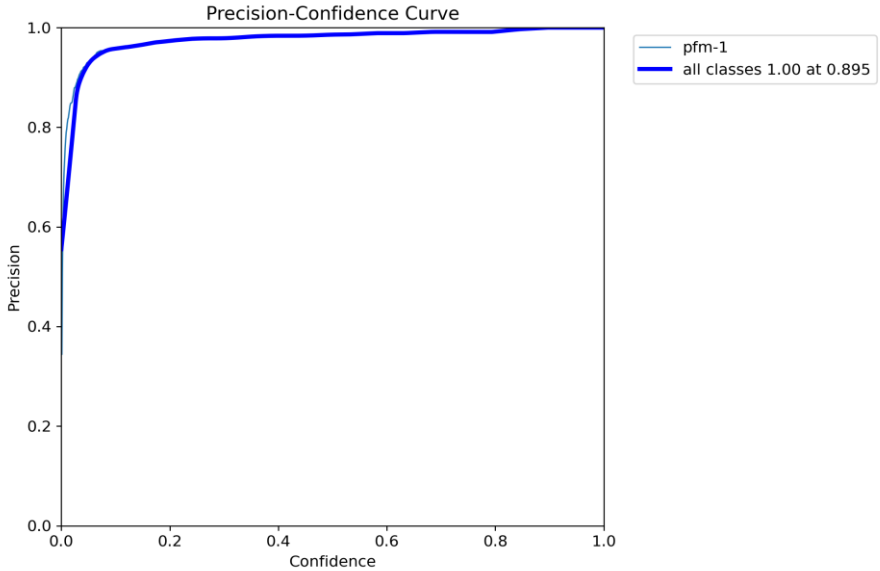

Рисунок 3.36 - Крива точності від впевненості показує найкращу оцінку точності 1 із порогом довіри 0.895

В даному випадку можна побачити, що впевненість при 100 епохах виросла до 0.895 (минуле значення при 50 епохах – 0.842)

При використанні даної моделі було помічено знатні покращення роботи на машині, де розроблялася система, також точність ідентифікації та розпізнавання мін ПФМ-1 при низькій роздільній здатності теж покращилось, враховуючи, що телефон із зображенням міни було переміщено в різні сторони під час тестування. Протестуємо систему на звичайних та аугментованих даних:

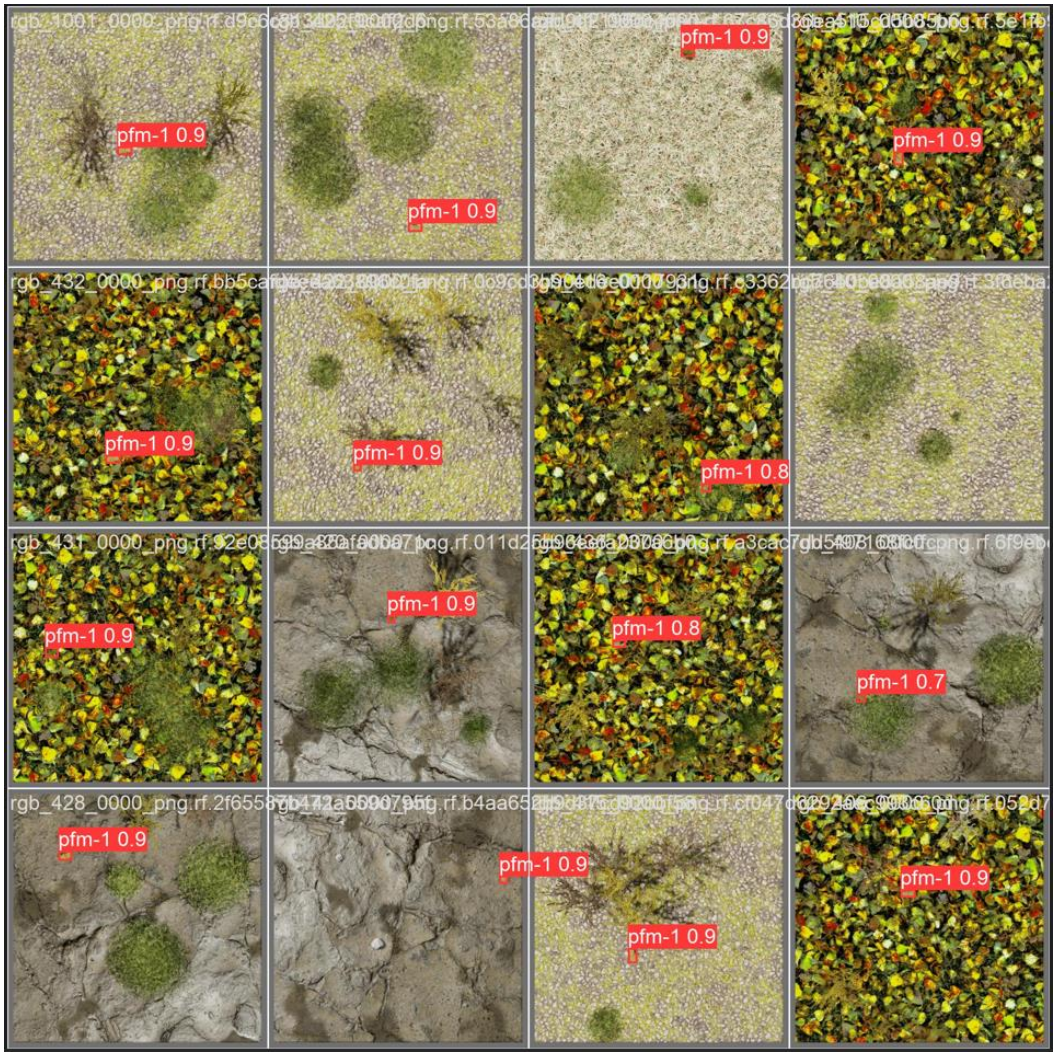

Рисунок 3.37 – Розпізнавання мін на звичайних зображеннях

Можна побачити, що в середньому результат дорівнює 90% при розпізнаванні мін на полях. І головне, що нейронна мережа за весь час при праці із

даними не розпізнала міни там, де їх нема, це можна помітити на рис. 3.37, де є пусті зображення без підпису, тобто без помилкової ідентифікації класифікованих об’єктів.

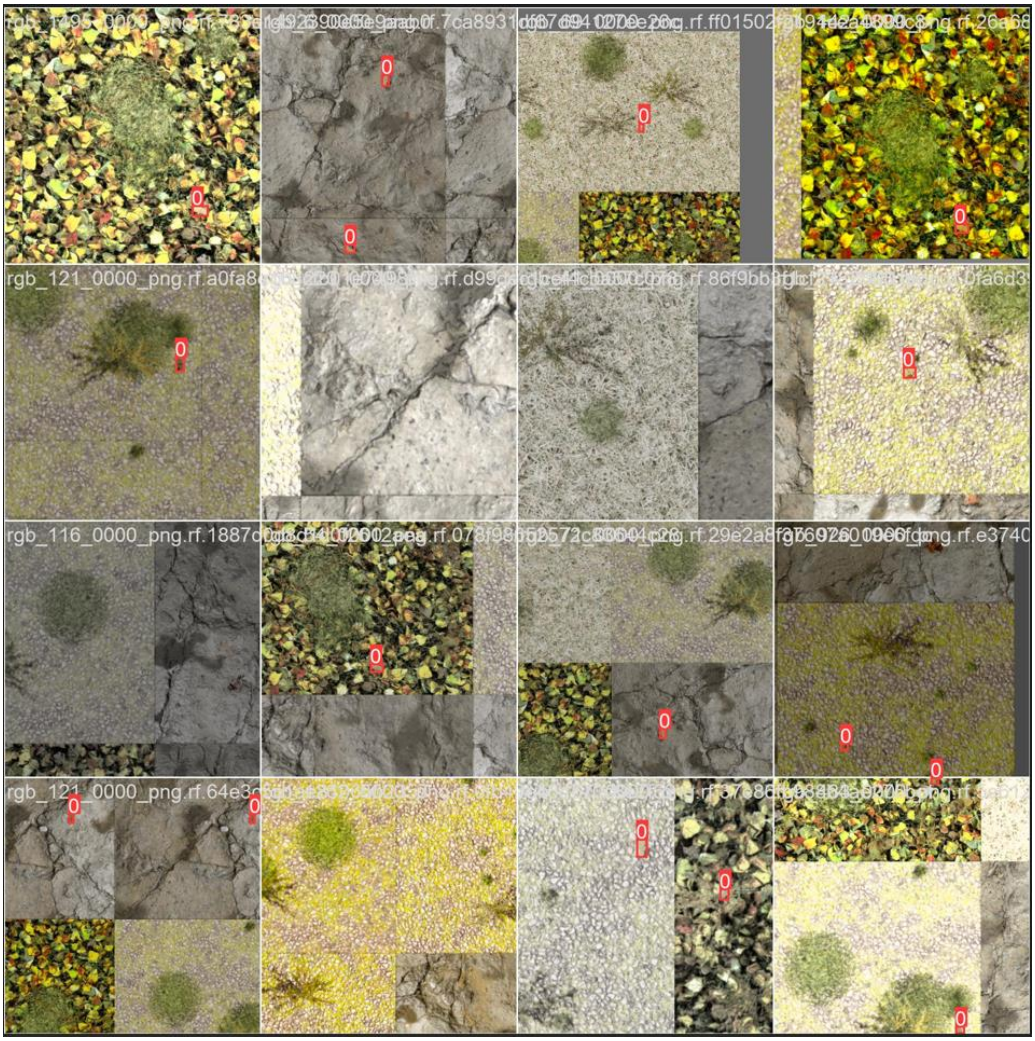

Рисунок 3.38 – Розпізнавання мін на аугментованих зображеннях

На аугментованих зображеннях нейронна мережа видає точність 89-90% в середньому, та також оброблює зображення без помилкової класифікації на зображеннях, де немає мін.

Рисунок 3.39 – Одне із тестових зображень розпізнавання міни ПФМ-1

Квантування у нейронних мережах є важливим методом оптимізації, який забезпечує оптимальний баланс між точністю та ресурсами. Цей підхід дозволяє зменшити обсяг пам'яті, необхідний для збереження параметрів моделі, та обчислювальні витрати під час інференсу.

Використання квантування дозволяє ефективно використовувати нейронну мережу на різних платформах, зокрема на пристроях з обмеженими обчислювальними можливостями, мобільних пристроях та вбудованих системах. Отримуючи високу швидкодію та зменшуючи споживання енергії, використання квантування стає необхідним елементом під час навчання нейронних мереж, забезпечуючи їхню ефективність та доступність для широкого спектру застосувань. Додаток А містить увесь код для навчання та тренування нейронних мереж CNN та YOLOv8. Після навчання оптимальної моделі можна переходити до імпорту моделі у вигляді застосунку на Android.

3.5 Імпорт моделі в NNAPI, створення застосунку

Для того щоб розробити мобільну версію моделі нейронної мережі для початку необхідно перевести його із формату «best_model.pt» у «best_model.pb», для цього встановлюємо необхідні бібліотеки:

```
(venv) PS D:\UNIVERSITY_WORKS\2_KRM\YOLOv8_model> pip install onnx
Collecting onnx
  Downloading onnx-1.15.0-cp310-cp310-win_amd64.whl.metadata (15 kB)
Requirement already satisfied: numpy in d:\university_works\2_krm\venv\lib\site-packages (from onnx) (1.26.2)
Collecting protobuf>=3.20.2 (from onnx)
  Using cached protobuf-4.25.1-cp310-abi3-win_amd64.whl.metadata (541 bytes)
Downloading onnx-1.15.0-cp310-cp310-win_amd64.whl (14.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 14.3/14.3 MB 8.2 MB/s eta 0:00:00
Downloading protobuf-4.25.1-cp310-abi3-win_amd64.whl (413 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 413.4/413.4 kB 6.4 MB/s eta 0:00:00
Installing collected packages: protobuf, onnx
Successfully installed onnx-1.15.0 protobuf-4.25.1
```

```
(venv) PS D:\UNIVERSITY_WORKS\2_KRM\YOLOv8_model> pip install tf2onnx
Collecting tf2onnx
  Downloading tf2onnx-1.15.1-py3-none-any.whl.metadata (1.2 kB)
```

Рисунок 3.40 – Встановлення необхідних бібліотек

Після цього пишемо код для переведення моделі у необхідний формат та отримаємо наступне:

```
(venv) PS D:\UNIVERSITY_WORKS\yolov5>  python export.py --weights best.pt --include tflite --img 640
export: data=D:\UNIVERSITY_WORKS\yolov5\data\coco128.yaml, weights=['best.pt'], imgsz=[640], batch_siz
=False, opset=17, verbose=False, workspace=4, nms=False, agnostic_nms=False, topk_per_class=100, topk_
YOLOv5  v7.0-254-gba63208 Python-3.10.0 torch-2.1.2+cpu CPU
```

Рисунок 3.41 – Процес переведення звичайного формату моделі у формат TensorFlow Lite

```
TensorFlow SavedModel: starting export with tensorflow 2.13.1...

ONNX: starting export with onnx 1.15.0 opset 17...
============ Diagnostic Run torch.onnx.export version 2.0.1+cu117 ============
verbose: False, log level: Level.ERROR
======================== 0 NONE 0 NOTE 0 WARNING 0 ERROR ========================

ONNX: simplifying with onnxsim 0.4.35...
ONNX: export success ✅ 5.7s, saved as 'runs\detect\100_ep_quanted\weights\best.onnx' (11.6 MB)
TensorFlow SavedModel: running 'onnx2tf -i "runs\detect\100_ep_quanted\weights\best.onnx" -o "runs\detect\100_ep_quanted\weights\best_saved_model"
TensorFlow SavedModel: export success ✅ 78.7s, saved as 'runs\detect\100_ep_quanted\weights\best_saved_model' (29.2 MB)
```

Рисунок 3.42 – Вдале конвертування моделі з формату «model.pb» у «best_float32.tflite»

Після отримання облегшеної моделі для мобільних пристроїв з використанням апаратного прискорення можна переходити до розробки застосунку, для цього створюємо звичайний проект в Android Studio:

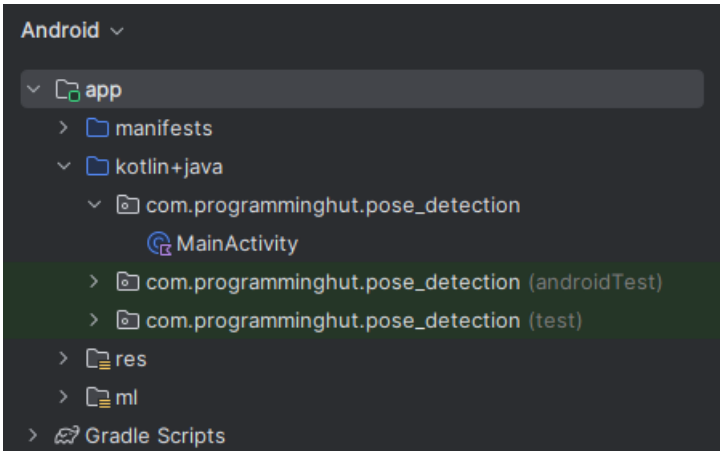

Рисунок 3.43 – Створення проекту для майбутнього застосунку

Одразу після цього переходимо до імпорту підготовленої моделі, для цього додаємо наступний файл:

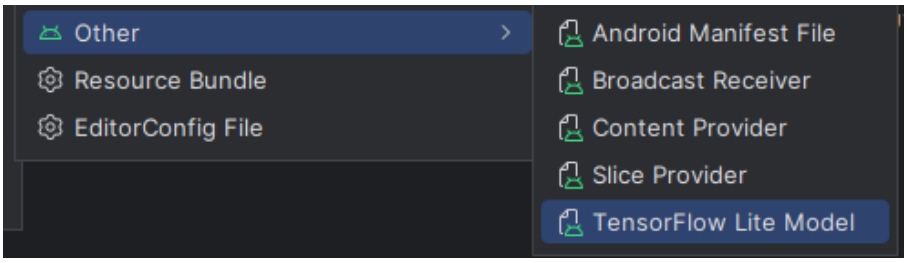

Рисунок 3.44 – Імпорт TFLite моделі

Обираємо необхідний файл, а саме створену попередньо модель:

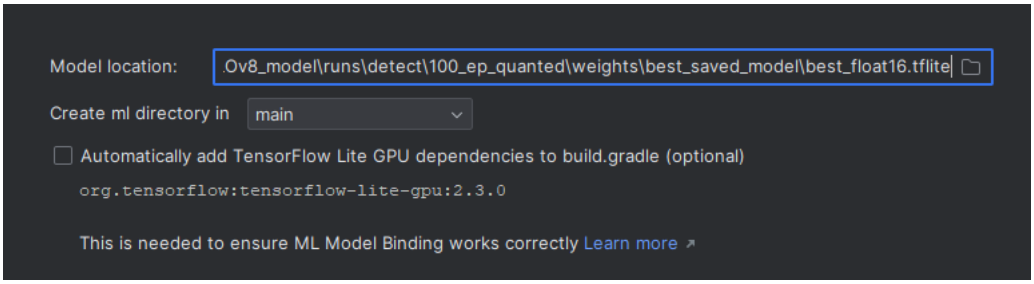

Рисунок 3.45 – Вибір шляху для імпорту моделі

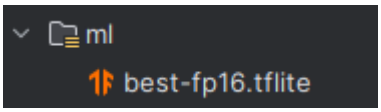

Рисунок 3.46 – Імпорт моделі у середовище

Виконав імпорт моделі налаштовуємо файл «activity_main.xml», який відповідає за головне меню застосунку, його робимо просто із чорним фоном, бо на весь екран буде виводитися вхідне зображення з камери застосунку. До файлу «MainActivity.kt» додамо логіку обробки моделі та її запуск, а також роботу із камерою, та у результаті отримаємо наступний результат:

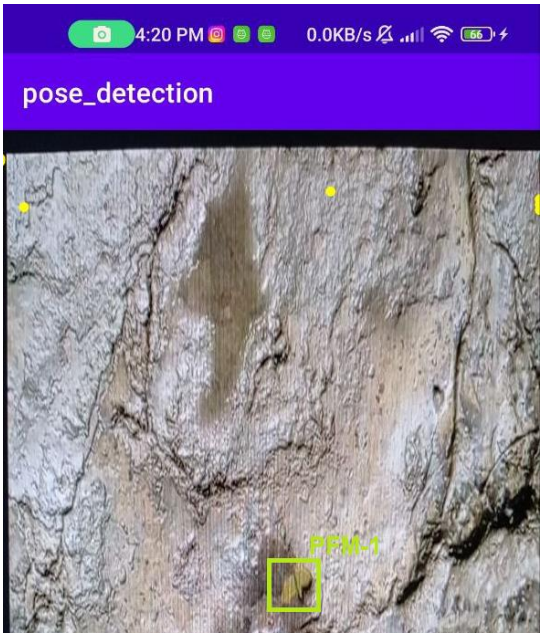

Рисунок 3.47 – Результат розпізнавання міни використовуючи камеру телефона

Після побудови проекту було проведено тестування імпортованої моделі на окремих тестових файлах. Аналізуючи роботу Android-застосунку можна зробити висновок, що у середньому на пристрої Redmi Note 9 із слабким процесором MediaTek Helio G85 камера видавала 35-45 кадрів в секунду під час роботи застосунку. Після квантування моделі було змінено ваги з формату fp32 на fp16 було помічено, що швидкість роботи моделі покращилась і навіть на слабких специфікаціях видає гарний результат, але точність впала до 85% максимум, що є

гарним результатом. У додатку Б представлено програмну реалізацію Android-застосунку.

3.6 Порівняння навчених моделей нейронних мереж

Головною задачею під час розробки системи комп’ютерного зору для розпізнавання та виявлення наземних мін ПФМ-1 – створення точної та надійної моделі нейронної мережі, яка дозволила б використовувати її на мобільних пристроях, або будь-яких інших які мають сучасний процесор та операційну систему Android.

Перша модель згорткової нейронної мережі показала погані результати навчання через застарілість архітектури та малої кількості шарів, які було використано для навчання. Також проблемою було використання 200 епох для навчання, даний підхід змусив перенавчитися нейронну мережу та виявляти окремі зони на зображенні та фон, но не самі міни. Також дана архітектура не підтримує object tracking, що створює проблему використання даної нейронної мережі в реальному часі через неоптимізоване використання ресурсів пристроїв. Тому було прийнято рішення відмовитися від даної архітектури.

Друга модель це була сучасна натренована архітектура YOLOv8, уважно виявивши усі специфікації та види моделі було обрано nano модель та 50 епох для навчання. Вона показувала вже більш гарні результати та мала можливість провести валідацію вхідних даних після кожної епохи щоб виявити такі проблеми як перенавчання та помилки у вхідних даних. Після тестування було помічено, що дана модель має гарний відсоток розпізнавання, але дуже важка навіть для сучасної машини, тому було вирішено перейти до створення третьої моделі.

Третя модель – це YOLOv8 nano з квантуванням, замість використання fp32 було переведено ваги до формату fp16, та замість 50 епох було проведено 100. У результаті було отримано таку саму модель по точності, на даних, які не були у наборі для навчання, але ще облегшену версію, що дозволило запустити її на

Android-пристроях. Дана інформація свідчить про те, що використання саме третьої моделі підходить для задачі розпізнавання та виявлення наземних мін ПФМ-1.

Таблиця 3.2 – Порівняння навчених нейронних мереж

Модель	box_loss	cls_loss	dfl_loss	precision	recall	epochs
CNN	1.0252	0.9463	1.5602	0.841	0.752	200
YOLOv8	0.7411	0.4343	0.8832	0.937	0.94	50
Квантована YOLOv8	0.6211	0.3445	0.8257	0.951	0.948	100

Висновки до 3 розділу

Було створено набір даних для навчання нейронної мережі через відсутність заздалегідь готового в спеціально створених для подібних задач ресурсах, також було проведено додаткову анотацію даних для навчання з вчителем через значний набір достовірних даних для навчання алгоритмів. Також було проведено додаткову роботу із аугментацією даних для кращого тестування моделей після навчання. Було розроблено скрипти для навчання нейронних мереж, а саме: Convolutional Neural Network, YOLOv8 nano 50 епох та квантована YOLOv8 nano 100 епох. Після аналізу та тестування усіх навчених мереж було виявлено, що краща модель це квантована YOLOv8 nano, тому її було конвертовано у формат TensorFlow Lite та імпортовано у Android-застосунок як окрему систему для розпізнавання та виявлення наземних мін ПФМ-1. У результаті точність розпізнавання у системі на мобільному пристрої дорівнює 85% на даних, які не були внесені до навчального набору.

ВИСНОВКИ

При виконанні кваліфікаційної роботи було досліджено та проаналізовано проблему розпізнавання та виявлення наземних мін ПФМ-1 за допомогою технології комп’ютерного зору та нейронних мереж. Проаналізовано існуючі матеріали та статті по розпізнаванню об’єктів використовуючи алгоритми комп’ютерного зору та сучасні архітектури нейронних мереж. Створення набору даних та їх анотація була проведена у сервісі Computer Vision Annotation Tool. Головні представники двох груп алгоритмів розпізнавання об’єктів CNN та YOLOv8 були навчені з використанням підготовленого набору даних. Експериментальним чином були отримані метрики моделей. Результати показали, що обидві моделі мають свої переваги та недоліки, але було виділено, що CNN потрібно дотренувати на додаткових даних, щоб дана мережа могла розпізнавати міни PFM-1 у режимі реального часу, маючи при цьому повільну швидкість обробки. YOLOv8 трохи втрачає точність на нових наборах даних, але володіє швидкістю обробки в декілька раз вище, ніж CNN, що дозволяє використовувати її для обробки вхідного зображення в режимі реального часу. Із недоліків YOLOv8 можна виділити навантаження на систему під час роботи, тому було квантовано дану модель та навчено на більшій кількості епох, що дало той самий результат із меншим навантаженням на систему. Дана модель підлягала тестуванню на даних, які не входили в навчальний набір даних, результатом було точне розпізнавання без критичних помилок.

У спеціальній частині з охорони праці та безпеки у надзвичайних ситуаціях було сформовано правила поведінки персоналу при знаходженні вибухонебезпечних пристроїв поза роботою на відпочинку, або в міській місцевості. Також виконано аналіз умов праці у офісному приміщенні де йде розробка стартапів, проведено замір норми клімату за допомогою гігрометру психометричного, у результаті було отримано інформацію, що приміщення повністю відповідає поставленим вимогам.

Завдання кваліфікаційної роботи було виконано в повній мірі, зокрема:

1) проаналізовано сучасний стан задачі розпізнавання та виявлення наземних мін ПФМ-1 шляхом аналізу подібних публікацій та відомих рішень;

2) створено власний набір даних для навчання;

3) обрано та проаналізовано архітектури нейронних мереж для розпізнавання та виявлення наземних мін ПФМ-1;

4) розглянуто алгоритми та API для оптимізації моделей нейронних мереж;

5) досліджено показники навчання нейронних мереж, у результаті кращу модель було імпортовано у розроблену систему розпізнавання та виявлення наземних мін ПФМ-1.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Jasper Baur, Gabriel Steinberg Applying Deep Learning to Automate UAV-Based Detection of Scatterable Landmines, 2020. 4 p.

2. Oleksandr A. Pryshchenko, Lorenzo Capineri Implementation of an Artificial Intelligence Approach to GPR Systems for Landmine Detection, 2022. 15-25 p.

3. Dan Munteanu, Diana Moina Sea Mine Detection Framework Using YOLO, SSD and EfficientDet Deep Learning Models, 2022. 1-6 p.

4. Venceslav Kafedziski, Sinisha Pecov Detection and Classification of Land Mines from Ground Penetrating Radar Data Using Faster R-CNN, 2018. 5-10 p.

5. Yi Xiao, Xinqing Wang Object Detection Based on Faster R-CNN Algorithm with Skip Pooling and Fusion of Contextual Information, 2020. 7-8 p.

6. Prasanta Das, Angshuman Chakraborty, Ravi Sankar Deep Learning-Based Object Detection Algorithms on Image and Video, 2023. 1-2 p.

7. Afdhal Afdhal, Khairun Saddami Real-Time Object Detection Performance of YOLOv8 Models for Self-Driving Cars in a Mixed Traffic Environment, 2023. 5-6p.

8. Muralidhar Pullakandam, Keshav Loya Weapon Object Detection Using Quantized YOLOv8, 2023. 8 p.

9. Ram Bawankule, Vaishnavi Gaikwad Visual Detection of Waste using YOLOv8, 2023. 10 p.

10. M. Rizk, I. Bayad Human Detection in Thermal Images Using YOLOv8 for Search and Rescue Missions, 2023. 7 p.

11. Baur, J., Steinberg, G. Detecting PFM-1 Landmines, 2022. 5 p.

12. Гангало І. М., Лісовий Д. О., Жебка В. В. Розпізнавання об’єктів за допомогою технологій комп'ютерного зору. Державний університет телекомунікацій, Київ, 2022. 5-6 с.

13. Бондар Ілля Обробка відео для розпізнавання та ідентифікації об'єктів, Київ, 2021. 10 с.

14. Лісовий В.Ю. Система розпізнавання зброї для камер відеоспостереження на основі методів комп'ютерного зору. Київ, 2018. 115 с.

15. Омельченко С. О. Використання комп’ютерного зору для розпізнавання образів. Київ, 2021. 15 с.

16. Клетте Рейнхард Комп’ютерний зір. ДМК Прес, 2019. 506 с.

17. Aishwarya Singh DeepMind’s Computer Vision Algorithm Brings the Power of Imagination to Build 3D Scenes from 2D Images, 2019. 3 p.

18. Sundas Iftikhar, Sundas Iftikhar Deep Learning-Based Pedestrian Detection in Autonomous Vehicles: Substantial Issues and Challenges, 2022. 10-20 p.

19. Мазур М. В. Задача виявлення та класифікації об'єктів за допомогою згорткових нейронних мереж, ВНТУ, 2019. 366 с.

20. Чару Аггарвал Нейронні мережі і глибоке навчання. Навчальний курс, Вильямс, 2020. 645-650 с.

21. Сєнін Ю.І. Згорткові нейронні мережі для визначення авторства текстів, 2021. 4 с.

22. Колберт Р., Уинстон Д., Ботту Л. Обробка природної мови (майже) з нуля, Журнал досліджень машинного навчання, 2011. 512 с.

23. Bharati P., Pramanik A. Deep learning techniques r-cnn to mask r-cnn: a survey. 2020. 655 p.

24. Xu Z., Jia R. Fast R-CNN method for detecting green mangoes in complex scenes using picking robots, 2020. 647 p.

25. Ross Girshick Fast R-CNN, IEEE International Conference on Computer Vision (ICCV), 2015. 15 p.

26. Shaoqing Ren, Kaiming He Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, IEEE, 2016. 1137 p.

27. Raducu Gavrilescu, Cristian Zet Faster R-CNN:an Approach to Real-Time Object Detection, IEEE, 2018. 15 p.

28. Bin Liu, Wencang Zhao Study of object detection based on Faster R-CNN, IEEE 2018. 12 p.

29. Xu Li, Tianxuan Hao Faster R-CNN-LSTM Construction Site Unsafe Behavior Recognition Model, MDPI, 2023. 4-5 p.

30. Ultralytics YOLOv8 Docs. URL: https://docs.ultralytics.com/ (дата звернення: 10.10.2023).

31. Адарша Шивананда, Акшай Кулкарні Проекти комп'ютерного зору з PyTorch: Проектування та розробка моделей виробничого рівня. APress, 2022. 222 с.

32. Кумар С. Підхід до класифікації образів та розпізнавання об'єктів на основі м'яких обчислень: Єдина концепція, 2014. 155 с.

33. Саймон Х. Нейронні мережі. Повний курс. 2-е видання, 2020. 870 с.

34. Чару Аггарвал Нейронні мережі і глибоке навчання. Навчальний курс, 2020. 350 с.

35. Рутковський Д. Нейронні мережі, генетичні алгоритми і нечіткі системи 2-е вид, 2014. 256 с.

36. Рашид Т. Створюємо нейронну мережу, 2018. 208 с.

37. Ніхиль Будума, Ніколас Локашо Основи глибокого навчання. Створення алгоритмів для штучного інтелекту наступного покоління, 2019. 151 с.

38. Метт Харрісон Машинне навчання. Кишеньковий довідник. Короткий посібник з методів структурованого машинного навчання на Python, 2020. 320 с.

39. Вей Л., Драгомір А. SSD: Single Shot MultiBox Detector, 2016. 100 с.

40. Charalampos Symeonidis, Ioannis Mademlis Improving Neural Non-Maximum Suppression for Object Detection, IEEE 2019. 5 p.

41. Tutiani, Sutresna Wati Implementation of Convolutional Neural Network (CNN) in Android-Based Acne Detection Applications, IEEE 2022. 12 p.

42. Yu-Xin Ding, Wei-Guo Zhao Automaticlly Learning Featurs Of Android Apps Using CNN, IEEE 2018. 5 p.

43. Xusheng Xiao, Shao Yang An Image-Inspired and CNN-Based Android Malware Detection Approach, IEEE 2020. 7 p.

44. Madushi H. CNN for User Activity Detection Using Encrypted In-App Mobile Data, MDPI 2022. 3 p.

45. Computer Vision Annotation Tool. URL: https://app.cvat.ai (дата звернення: 11.11.2023).

46. Khaled A. Data Augmentation in Classification and Segmentation: A Survey and New Strategies, MDPI 2023. 3-4 p.

ДОДАТОК А

Програмний код для навчання нейронної мережі

Файл-конфігурації config.yaml

```
path: 'D:\UNIVERSITY_WORKS\2_KRM\YOLOv8_model\dataset'
train: 'train\images'
val: 'validation\images'
test: 'test\images'
# Classes
names:
  0: pfm-1
```

Скрипт train.py

```
from ultralytics import YOLO

if __name__ == "__main__":
    # Load a model
    model = YOLO("yolov8n.yaml")
    print(“CAN YOU HEAR THE SILENCE?”)
    print(“CAN YOU SEE THE DARK?”)
    print(“CAN YOU FIX THE BROKEN?”)
    print(“CAN YOU FEEL?”)
    print(“CAN YOU FEEL MY HEART?”)

    # Using the model
    results = model.train(data="config.yaml", epochs=50)
```

Скрипт predict_webcamera.py

```
import os
import cv2
from ultralytics import YOLO

# change here path to needed model (change 50ep)
model_path = os.path.join('.', 'runs', 'detect',
'100_ep_quanted', 'weights', 'last.pt')
model = YOLO(model_path)  # load a custom model
```

```
threshold = 0.5

cap = cv2.VideoCapture(0)
if not cap.isOpened():
    raise IOError("Cannot open webcam")

while True:
    ret, frame = cap.read()
    if not ret:
        break

    results = model(frame)[0]

    for result in results.boxes.data.tolist():
        x1, y1, x2, y2, score, class_id = result

        if score > threshold:
            cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), (0, 255, 0), 4)
            cv2.putText(frame, results.names[int(class_id)].upper(), (int(x1), int(y1 - 10)),
                        cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0, 255, 0), 3, cv2.LINE_AA)

    cv2.imshow('Object Detection', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Скрипт predict_webcamera.py

```
import os
```

```
import cv2
from ultralytics import YOLO

# change here path to needed model (change 50ep)
model_path = os.path.join('.', 'runs', 'detect', '100_ep_quanted', 'weights', 'last.pt')
model = YOLO(model_path)  # load a custom model
threshold = 0.5
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    raise IOError("Cannot open webcam")
while True:
    ret, frame = cap.read()
    if not ret:
        break
    results = model(frame)[0]
    for result in results.boxes.data.tolist():
        x1, y1, x2, y2, score, class_id = result
        if score > threshold:
            cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), (0, 255, 0), 4)
            cv2.putText(frame, results.names[int(class_id)].upper(), (int(x1), int(y1 - 10)),
                        cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0, 255, 0), 3, cv2.LINE_AA)

    cv2.imshow('Object Detection', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

ДОДАТОК Б

Програмний код системи

Дизайн AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.programminghut.pose_detection">

    <uses-permission android:name="android.permission.CAMERA"/>

    <application
        android:allowBackup="true"

android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Pose_detection"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

```
    </application>
</manifest>
```

MainActivity.kt

```
package com.programminghut.pose_detection
import android.annotation.SuppressLint
import android.content.Context
import android.content.pm.PackageManager
import android.graphics.*
import android.hardware.camera2.CameraCaptureSession
import android.hardware.camera2.CameraDevice
import android.hardware.camera2.CameraManager
import android.hardware.camera2.CaptureRequest
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.os.Handler
import android.os.HandlerThread
import android.util.Log
import android.view.Surface
import android.view.TextureView
import android.widget.ImageView
import com.programminghut.pose_detection.ml.BestFp16
import
com.programminghut.pose_detection.ml.LiteModelMovenetSingleposeLight
ningTfliteFloat164
import org.tensorflow.lite.DataType
import org.tensorflow.lite.support.image.ImageProcessor
import org.tensorflow.lite.support.image.TensorImage
import org.tensorflow.lite.support.image.ops.ResizeOp
import org.tensorflow.lite.support.tensorbuffer.TensorBuffer

class MainActivity : AppCompatActivity() {

    val paint = Paint()
```

```
        lateinit var imageProcessor: ImageProcessor
        lateinit var model: BestFp16
        lateinit var bitmap: Bitmap
        lateinit var imageView: ImageView
        lateinit var handler:Handler
        lateinit var handlerThread: HandlerThread
        lateinit var textureView: TextureView
        lateinit var cameraManager: CameraManager
        override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState)
            setContentView(R.layout.activity_main)
            get_permissions()

            imageProcessor                                          =
ImageProcessor.Builder().add(ResizeOp(416,                      416,
ResizeOp.ResizeMethod.BILINEAR)).build()
            model = BestFp16.newInstance(this)
            imageView = findViewById(R.id.imageView)
            textureView = findViewById(R.id.textureView)
            cameraManager = getSystemService(Context.CAMERA_SERVICE)
as CameraManager
            handlerThread = HandlerThread("videoThread")
            handlerThread.start()
            handler = Handler(handlerThread.looper)

            paint.setColor(Color.GREEN)

            textureView.surfaceTextureListener                      =
object:TextureView.SurfaceTextureListener{
                override      fun      onSurfaceTextureAvailable(p0:
SurfaceTexture, p1: Int, p2: Int) {
                    open_camera()
                }
```

```
                override fun onSurfaceTextureSizeChanged(p0:
SurfaceTexture, p1: Int, p2: Int) {}

                override fun onSurfaceTextureDestroyed(p0:
SurfaceTexture): Boolean {return false}
                override fun onSurfaceTextureUpdated(p0:
SurfaceTexture) {
                    bitmap = textureView.bitmap!!
                    var tensorImage = TensorImage(DataType.FLOAT32)
                    tensorImage.load(bitmap)
                    tensorImage                                    =
imageProcessor.process(tensorImage)
val inputFeature0 = TensorBuffer.createFixedSize(intArrayOf(1, 416,
416, 3), DataType.FLOAT32)
                    inputFeature0.loadBuffer(tensorImage.buffer)

                    val outputs = model.process(inputFeature0)
                    val               outputFeature0               =
outputs.outputFeature0AsTensorBuffer.floatArray
                    var                  mutable                   =
bitmap.copy(Bitmap.Config.ARGB_8888, true)
                    var canvas = Canvas(mutable)
                    var h = bitmap.height
                    var w = bitmap.width
                    var x = 0
                    Log.d("output__",
outputFeature0.size.toString())
                    while (x <= outputFeature0.size - 4) {
                        val confidence = outputFeature0[x + 2]
                        if (confidence > 0.70) {
                            val left = outputFeature0[x] * w
                            val top = outputFeature0[x + 1] * h
```

```
                                val right = outputFeature0[x + 2] * w
                                val bottom = outputFeature0[x + 3] * h
                                paint.style = Paint.Style.STROKE //
Установка стиля на пустой прямоугольник
                                canvas.drawRect(left, top, right,
bottom, paint)
                                val label = "pfm-1"
                                if (label == "pfm-1") {
                                    paint.style = Paint.Style.FILL
                                    paint.textSize = 30f
                                    val textWidth =
paint.measureText(label)
                                    val textHeight = paint.textSize
                                    paint.color = Color.WHITE
                                    canvas.drawRect(left, top -
textHeight, left + textWidth, top, paint)
                                    paint.color = Color.BLACK
                                    canvas.drawText(label, left, top,
paint)
                                }
                            }
                            x += 4
                        }
                        imageView.setImageBitmap(mutable)
                    }
                }
            }
            override fun onDestroy() {
                super.onDestroy()
                model.close()
            }
            @SuppressLint("MissingPermission")
            fun open_camera(){
```

```
            cameraManager.openCamera(cameraManager.cameraIdList[0],
object:CameraDevice.StateCallback(){
                override fun onOpened(p0: CameraDevice) {
                    var               captureRequest               =
p0.createCaptureRequest(CameraDevice.TEMPLATE_PREVIEW)
                    var                  surface                   =
Surface(textureView.surfaceTexture)
                    captureRequest.addTarget(surface)
                    p0.createCaptureSession(listOf(surface),
object:CameraCaptureSession.StateCallback(){
                        override        fun        onConfigured(p0:
CameraCaptureSession) {

p0.setRepeatingRequest(captureRequest.build(), null, null)
                        }
                        override      fun      onConfigureFailed(p0:
CameraCaptureSession) {
                        }
                    }, handler)
                }
                override fun onDisconnected(p0: CameraDevice) {
                }
                override fun onError(p0: CameraDevice, p1: Int) {
                }
            }, handler)
        }
        fun get_permissions(){
    if(checkSelfPermission(android.Manifest.permission.CAMERA)    !=
PackageManager.PERMISSION_GRANTED){

requestPermissions(arrayOf(android.Manifest.permission.CAMERA), 101)
            }
        }
```

```
        override fun onRequestPermissionsResult(  requestCode: Int,
permissions: Array<out String>, grantResults: IntArray  ) {
            super.onRequestPermissionsResult(requestCode,
permissions, grantResults)
            if(grantResults[0] != PackageManager.PERMISSION_GRANTED)
get_permissions()
        }
```