

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ
ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

**Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту**

До захисту допущено:

Завідувач кафедри

_____ Олена ЧУМАЧЕНКО

«__» _____ 2023 р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

**за освітньо-професійною програмою «Системи і методи штучного інтелекту»
спеціальності 122 «Комп'ютерні науки»**

**на тему: «Розробка системи візуального ідентифікування транспорту в режимі
реального часу з використанням нейронних мереж»**

Виконав:

студент IV курсу, групи КІ-93

Бут Ігор Олегович _____

Керівник:

доцент, к.ф.-м.н., Барановська Леся Валеріївна _____

Консультант з нормоконтролю:

фах. 1-ї кат. Гончарук Максим Миколайович _____

Консультант з економічного розділу:

доцент, к.е.н., Рощина Надія Василівна _____

Рецензент:

доцент, к.т.н., Харченко Костянтин Васильович _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.

Студент _____

Київ, 2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп’ютерні науки»

Освітня програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ:

Завідувач кафедри

_____ Олена ЧУМАЧЕНКО

«___» _____ 2023 р.

ЗАВДАННЯ

на дипломну роботу студенту

Буту Ігорю Олеговичу

1. Тема роботи «Розробка системи візуального ідентифікування транспорту в режимі реального часу з використанням нейронних мереж», керівник роботи к.ф.-м.н., доцент Барановська Леся Валеріївна, затверджені наказом по університету від «30» травня 2023 р. № 2065-с
2. Термін подання студентом роботи 16.06.2023
3. Вихідні дані до роботи – система автоматичного ідентифікування транспорту.
4. Зміст роботи – 1. Дослідження предметної області; 2. – Теоретичні основи нейронних мереж для розпізнавання зображень; 3. Опис програмного продукту; 4. Функціонально-вартісний аналіз програмного продукту.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	доцент, к.е.н., Рощина Н.В.		

7. Дата видачі завдання 21.02.2023

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вивчення літератури за темою роботи	17.04.2023	Виконано
2.	Підготовка першого розділу	24.04.2023	Виконано
3.	Підготовка другого розділу	01.05.2023	Виконано
4.	Підготовка датасетів для моделі	04.05.2023	Виконано
5.	Розробка програмного продукту	06.05.2023	Виконано
6.	Підготовка третього розділу	28.05.2023	Виконано
7.	Підготовка економічної частини	03.06.2023	Виконано
8.	Підготовка презентації та доповіді	07.06.2023	Виконано
9.	Оформлення дипломної роботи	08.06.2023	Виконано

Студент

Ігор БУТ

Керівник

Леся БАРАНОВСЬКА

РЕФЕРАТ

Дипломна робота: 122 ст., 60 рис., 6 табл., 1 додаток, 45 джерел.

ШТУЧНИЙ ІНТЕЛЕКТ, НЕЙРОННІ МЕРЕЖІ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, ОДНОЕТАПНІ МОДЕЛІ ВИЯВЛЕННЯ

Об'єктом досліджень є автомобільний транспорт різних категорій з номерними знаками на кадрах з відеокамери.

Предметом досліджень є згорткові нейронні мережі для виявлення об'єктів та їх розпізнавання.

Програма написана мовою Python.

Актуальність роботи полягає в тому що в світі активно впроваджуються різні системи для спрощення методів оплати за користування платними автодорогами. Комп'ютерні рішення, такі як розпізнавання номеру транспортного засобу, дозволяють замінити потребу в будь-яких спеціальних пропускних пунктах, додаткових приладах чи інших видах перепусток.

Метою даної роботи є розробка програмного забезпечення для керування системою, яка може містити декілька камер для автоматичного розпізнавання транспортного засобу в режимі реального часу всіма камерами. Система повинна автоматично ідентифікувати проїжджаючий транспортний засіб за його номером та класифікувати за його типом.

В результаті роботи було досліджено різновиди моделей для виявлення об'єктів, навчено та протестовано вибрану модель, реалізовано систему розпізнавання в режимі реального часу.

ABSTRACT

Thesis: 122 pages, 60 figures, 6 tables, 1 appendix, 45 sources.

ARTIFICIAL INTELLIGENCE, NEURAL NETWORKS, CONVOLUTIONAL
NEURAL NETWORKS, ONE-STAGE DETECTION MODELS

The object of research is motor vehicles of different categories with license plates on the video camera footage.

The subject of research is convolutional neural networks for object detection and recognition.

The program is written in Python.

The relevance of the work lies in the fact that various systems are being actively implemented in the world to simplify payment methods for using toll roads. Computer solutions, such as vehicle license plate recognition, can replace the need for any special checkpoints, additional devices, or other types of passes.

The aim of this work is to develop software to control a system that can contain multiple cameras to automatically recognize a vehicle in real time by all cameras. The system should automatically identify a passing vehicle by its license plate number and classify it by its type.

As a result of the work, we investigated the types of models for object detection, trained and tested the selected model, and implemented a real-time recognition system.

ЗМІСТ

СПИСОК УМОВНИХ СКОРОЧЕНЬ	8
ВСТУП.....	9
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ІДЕНТИФІКАЦІЇ ТРАНСПОРТНОГО ЗАСОБУ	11
1.1 Актуальність роботи	11
1.2 Огляд наявних реалізацій оплати за проїзд на автобанах.....	12
1.3 Розпізнавання зображення	17
1.3.1 Виділення ознак	17
1.3.2 Тренування системи розпізнавання	20
1.3.3 Основні задачі розпізнавання	21
1.4 Постановка задачі.....	23
1.5 Висновок до розділу 1	24
РОЗДІЛ 2 ТЕОРЕТИЧНІ ОСНОВИ НЕЙРОННИХ МЕРЕЖ ДЛЯ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ.....	25
2.1 Нейронні мережі.....	25
2.1.1 Недоліки використання штучної нейронної мережі при розпізнаванні зображень	26
2.2 Згорткові нейронні мережі	28
2.2.1 Шар згортки	29
2.2.2 Шар активації	31
2.2.3 Шар агрегації.....	35
2.2.4 Повнозв'язний шар	36
2.2.5 Оптимізатори	37
2.3 Вибір моделі розпізнавання	42
2.3.1 Метрики для вибору моделі	42
2.3.2 Підходи в моделях розпізнавання	47
2.3.3 Вибір моделі для розпізнавання в реальному часі	48
2.4 Висновок до розділу 2	52
РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОЇ ЧАСТИНИ	53

3.1	Задачі системи	53
3.2	Модель виявлення	53
3.2.1	Вибір моделі YOLO	53
3.2.2	Архітектура моделі	56
3.2.3	Навчання моделі	63
3.3	Використані інструменти	71
3.3.1	Мова програмування та середа розробки	71
3.3.2	Інструменти розробки	71
3.4	Опис роботи системи	72
3.4.1	Модуль захоплення кадрів	73
3.4.2	Модуль опрацювання кадру	74
3.4.3	Модуль розпізнавання символів	76
3.4.4	Модуль створення JSON файлу	82
3.5	Висновок до розділу 3	84
РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО		
ПРОДУКТУ		86
4.1	Постановка задачі проектування	87
4.2	Обґрунтування функцій програмного продукту	87
4.3	Обґрунтування системи параметрів програмного продукту	90
4.4	Аналіз експертного оцінювання параметрів	93
4.5	Аналіз рівня якості варіантів реалізації функцій	97
4.6	Економічний аналіз варіантів розробки ПП	99
4.7	Вибір кращого варіанту ПП техніко-економічного рівня	105
4.8	Висновки до розділу 4	106
ВИСНОВОК		107
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ		109
ДОДАТОК А		114

СПИСОК УМОВНИХ СКОРОЧЕНЬ

ANN – Artificial Neural Network

ANPR – Automatic Number-Plate Recognition

AP – Average Precision

BCE – Binary Cross Entropy

CIoU – Complete Intersection over Union

CNN – Convolutional Neural Network

COCO – Common Objects in Context

DFL – Distribution Focal Loss

EETS – European Electronic Toll Service

FN – False Negative

FP – False Positive

FPN – Feature Pyramid Network

FPS – Frames Per Second

IoU – Intersection over Union

JSON – JavaScript Object Notation

mAP – mean Average Precision

NMS – Non-Maximum Suppression

OCR – Optical Character Recognition

RoI – Region of Interest

SSD – Single Shot Detector

TN – True Negative

TP – True Positive

YOLO – You Only Look Once

ВСТУП

Впровадження технологічних рішень допомагає людині полегшити собі життя, автоматизувати одноманітні операції та зменшити побічні витрати на їх виконання. За допомогою штучного інтелекту та машинного навчання вдалося створити системи, що можуть опрацьовувати дані по мірі їх надходження в систему. До таких даних зокрема належать і записи з відеокамер.

Розпізнавання зображень стало досить актуальним в останній час, дозволивши пришвидшити вже існуючі процеси та відкрити абсолютно нові, раніше неможливі сфери застосування. Однією з областей використання, де розпізнавання може значно змінити теперішні підходи являється система автобанів.

Раніше не було інструментів, які б без додаткових засобів могли б ідентифікувати транспортний засіб щоб дізнатися звідки він їде, скільки вже проїхав кілометрів платною дорогою, щоб в результаті сформувати рахунок водію. Проте зараз область ідентифікації транспорту починає активно змінюватися за рахунок введення автоматизованих системи розпізнавання транспорту, які не будуть потребувати ніяких додаткових засобів чи приладів.

Незважаючи на досягнутий прогрес у розпізнаванні в режимі реального часу, ця сфера все ще є викликом для програмістів. Основними умовами для використання такої програми на платних автомагістралях є її надійність, точність та гнучкість. Оскільки маємо справу з фінансовою сферою, недопустимі нарахування неправильних сум на рахунки користувачів через недостовірне ідентифікування. Також неправильне розпізнавання типу транспортного засобу чи його номеру може спричинити багато незручностей при майбутньому опрацюванні.

Метою даної дипломної роботи є дослідження методів та моделей розпізнавання, інтерпретації цих рішень для задачі ідентифікування транспортного засобу в режимі реального часу. Для цього планується використовувати найновіші технологічні рішення, попередньо дослідивши їх структуру та роботу. Система буде

включати технологію розпізнавання транспортного засобу, а також технологію для розпізнавання номерного знаку.

Основна робота полягає в створенні системи, повинна захоплювати кадр з відеокамери, після чого передавати його до моделі виявлення елементів на зображенні. Після того як буде виявлено транспортний номер, провести класифікацію транспортного засобу, якому належить даний номер та розпізнавання номеру. Результати роботи мають бути збережені у форматі, придатному для використання в подальшій обробці.

Таким чином об'єктом дослідження постає захоплений відеокадр з камери з наявними транспортними засобами. Предметом дослідження є нейронні мережі для виявлення об'єктів на зображенні.

Дана робота складається з чотирьох розділів. В першому проводиться дослідження актуальності теми та предметної області, задача обробки зображення та способи її реалізації, постановка задачі. В другому розділі проведено дослідження нейронних мереж, згорткових нейронних мереж для роботи з зображеннями, їх різновиди моделей розпізнавання, розглянуті метрики для дослідження та проведено порівняння. В третьому розділі розглянуті варіанти представлення вибраної моделі, а також проведено дослідження їх архітектури. Описано процес створення та роботи розробленої системи та продемонстровані результати роботи. В четвертому розділі проведено функціонально вартісний аналіз програмного продукту.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ІДЕНТИФІКАЦІЇ ТРАНСПОРТНОГО ЗАСОБУ

1.1 Актуальність роботи

Одним з критеріїв вибору даної теми стала актуальність поставленої задачі. Впровадження в Україні цифрового застосунку “Дія” показала, що за допомогою цифрових рішень можна відмовитися від багатьох витрат, таких як людський час та додаткові витрати на виконання певних послуг.

Впровадження цифрових рішень в галузі ідентифікування транспорту також може внести свої зміни в декількох можливих сценаріях використання.

Автоматизація платежів шляхом ідентифікації транспортного засобу. До даного сценарію можна віднести використання обчислювальної машини та системи розпізнавання для автоматичного ідентифікування транспортного засобу та розрахунку вартості послуг на паркінгах або пунктах оплати на автомагістралях. Замість використання фізичних датчиків для вимірювання часу та відстані, пройдених транспортним засобом, цей підхід дозволяє уникнути додаткових ресурсів шляхом використання комп’ютерного програмного забезпечення та технологій розпізнавання.

Запровадження даної системи на дорогах дозволить вирішувати одразу декілька задач:

- Автоматизована система відеонагляду за дотриманням правил дорожнього руху. Використовуючи програму розпізнавання можна буде ідентифікувати правопорушення на критичних ділянках та визначати їх ініціаторів.
- Виявлення викрадених машин. Розставлені пункти систем відеонагляду з розпізнаванням на дорогах міста, на виїздах з нього зможе пришвидшити пошук викраденого транспорту та зменшити

навантаження на правоохоронні органи, полегшивши їм роботу пошуку та перехоплення.

В даній дипломній роботі було реалізовано програмний продукт саме для його використання на автоматизованих системах автобанів.

1.2 Огляд наявних реалізацій оплати за проїзд на автобанах

Існує багато різновидів стягування плати з водія за користування автомагістраллю. Різні країни з платними автобанами використовують різні системи оплати або поєднують їх. Розглянемо найпоширеніші типи стягування плати а також їх плюси та недоліки.

Віньетки – форма оплати за користування автомагістральними шляхами країни у вигляді спеціальної наклейки, яка клеїться на лобове скло (рис. 1.1). Дані віньетки видаються на певний проміжок часу, і ці проміжки різняться в залежності від країни. Також віньетки відрізняються у вартості в залежності від типу транспортного засобу, на який вона клеїться. Найчастіше віньетки продають на АЗС, поштових відділеннях або прикордонних переїздах. Даний тип оплати використовується у багатьох європейських країнах, таких як Австрія, Болгарія, Чеська Республіка.[1]



Рисунок 1.1 – Приклад віньетки та її положення на лобовому склі[2]

До переваг оплати за користування дорогою віньєткою можна віднести:

- Одноразова плата за користування дорогою на певний проміжок часу;
- Відсутність потреби в оплаті при багаторазових виїздах з автомагістралі.

Недоліки:

- Потреба в обов'язковій зупинці для купівлі віньєтки;
- При незручних часових рамках дії віньєтки наявність переплати за невикористаний час;
- Проблема зчитування при наявності негоди.

Турнікети – спеціально обладнані пункти пропуску, на яких відбувається стягування платні за використання дороги (рис. 1.2). Допоки не відбудеться сплата, проїзд буде зачиненим для автотранспорту. В різних країнах тип таких турнікетів може відрізнятися. Наприклад, у Франції оплата відбувається постійно після проїзду певної ділянки розділеного маршруту. Зазвичай кожен такий пункт вже має виставлену вартість, яку має сплатити водій.

В Італії сплата відбувається одноразово. При заїзді на платні магістралі водій отримує спеціальний білет, на якому зазначена вся потрібна для системи інформація. При виїзді з автобану вираховується дистанція, яку проїхав транспорт та розраховується плата за користуванням дорогою.

Також ще багато інших країн використовують концепт турнікетів, серед яких є Іспанія, Португалія.[1]



Рисунок 1.2 – Турнікети для оплати проїзду [1]

Переваги:

- Відсутність необхідності покидати транспортний засіб;
- В залежності від реалізації – відсутність потреби в додаткових співробітниках та зменшення витрат.

Недоліки:

- В залежності від типу стягування платні, встановлення великої кількості турнікетів збільшує вартість проекту та підтримки;
- Збільшення часу на проходження пунктів оплати.

Для збільшення пропускної здатності на пунктах пропуску був розроблений Європейський електронний сервіс дорожніх зборів (EETS).

EETS створює можливість для користувачів платити за проїзд по платним дорогам усередині Європейського Союзу шляхом укладення єдиного договору з постачальником EETS та використання одного бортового пристрою для автоматичного стягування коштів. Це дозволяє водіям ефективно та зручно сплачувати за проїзд на платних дорогах в різних країнах Європи, замість використання різних систем та методів оплати.

На даний момент вже більшість європейських країн входять у зону дії EEST. Для автоматизації оплати та розпізнавання застосовується спеціальний пристрій, який ще називається транспондером (рис. 1.3), який потрібно окремо замовляти. Також при виборі плану пересування є наявність багатьох пакетів, що робить оплату більш вигідною в залежності від необхідних послуг.[3]



Рисунок 1.3 – Пристрій для оплати за проїзд Telepass device[4]

До переваг використання бортового пристрою можна віднести:

- Швидке проходження пунктів оплати;
- Контроль витрат при користуванні;
- Можливість обирати вигідний для користувача план без переплат.

Недоліки:

- Потреба в спеціальному бортовому пристрої;
- Потреба платити гроші за пристрій. В залежності від оператора, за пристрій доведеться заплатити одноразово, або платити орендні внески чи постійно підтримувати певний пакет послуг;
- Оскільки використовується електронний пристрій, він може загубитися, вийти з ладу або бути вкраденим.

Навіть при наявності вагомих переваг у використанні бортових пристроїв з програми EEST, ця опція потребує додаткових грошових витрат на самі пристрої та їх виробництво та покращення. Також гості Євросоюзу не мають змоги купляти ці бортові пристрої, або їх використання було б не раціональним для туристів.[3]

Через недоліки розглянутих вище способів, дослідимо систему, яка має найбільше шансів стати головною серед усіх платних автобанів. Розглянемо спосіб оплати за користування автобаном за допомогою ANPR. ANPR – це технологія для оптичного розпізнавання номерних знаків транспортного засобу для подальшого опрацювання цієї інформації в обчислювальних системах.[5] Зазвичай, системи ANPR використовуються на відкритих дорогах (рис. 1.4). Також такі відкриті дороги зазвичай підтримують використання транспондерів для безконтактного зчитування. У разі якщо транспондера не виявлено на транспортному засобі, відбувається зчитування номерного знаку. Головна особливість таких доріг полягає в тому, що на них відсутні будь-які пункти збору плати за користування дорогами та відсутність будь-яких воріт та турнікетів.

Оскільки немає необхідності сильно скидати швидкість та очікувати свою чергу на пунктах пропуску, пропускна спроможність таких пунктів найбільша серед усіх розглянутих варіантів. Оплата в такому разі відбувається в електронному форматі через спеціальні застосунки або вебсайти. Також рахунок може надходити у

вигляді електронної пошти на рахунок власника транспортного засобу. Серед країн, які використовують відкриті дороги є Великобританія, США, Польща, Сінгапур.[5]



Рисунок 1.4 – Відкриті ANPR ворота[5]

Розглянемо переваги та недоліки ANPR воріт.

Переваги:

- Найбільша пропускна здатність серед всіх платних доріг;
- Водій витрачає менше свого часу;
- Легкість сплати коштів ;
- Відсутність потреби наймати людей для пунктів пропуску;
- Легкість відстеження викраденого або розшукуваного авто.

Недоліки:

- Незареєстрований водій може проігнорувати сплату і скористатися дорогою безкоштовно;
- При збою в роботі камери або некоректному розпізнаванні можуть виникнути непорозуміння за нараховані рахунки;
- Вночі, при сильній негоді або забрудненості транспорту розпізнавання ускладнюється;
- Потреба в серйозному залізі для виконання розпізнавання.

1.3 Розпізнавання зображення

Головним завданням розпізнавання зображення є автоматизувати виявлення та ідентифікацію об'єкта, його патернів та ознак на зображенні. Метою є створення самостійної системи, яка зможе самостійно класифікувати та визначати об'єкти на зображенні.

Використання комп'ютерних технологій для розпізнавання типу транспортного засобу та його номерного знаку дозволяють поступово відмовлятися від застарілих методів ідентифікації учасника дорожнього руху.

Замість потреби в людському персоналі на пунктах пропуску на автобанах, впровадження комп'ютерних рішень пришвидшує проведення операцій щодо записування транспортного засобу та розрахунку платні за використання дороги.

1.3.1 Виділення ознак

Першим етапом розпізнавання кадру з фотографії або відео є виявлення зображення. На даному етапі зображення перетворюється на певний набір ознак які характеризують об'єкт на зображенні. Для різних підходів можна виділити різні класи ознак. Наприклад, для розпізнавання на основі зовнішнього вигляду, можна використовувати наступні ознаки:

- Геометричні ознаки. Вони відображають геометричні показники об'єкту, такі як розмір, форма, пропорція, тощо.
- Текsturні ознаки. До них можна віднести текстурні характеристики об'єктів на зображенні, такі як шорсткість, гладкість.

- Ознаки кольорів. Сюди відносяться колірні характеристики об'єктів або їх окремих частин.

Ще один підхід до виявлення об'єктів базується на основі аналізу руху об'єкта. При розгляді транспортної теми характеристики руху використовуються для відрізнєння транспортних засобів від статичних фонових сцен. До таких ознак можна віднести швидкість, траєкторію, патерн руху транспортного засобу.

Розглянемо методи, які найчастіше використовують для визначення розглянутих ознак. Такі методи можна поділити на три категорії:

- Моделі на основі зовнішнього вигляду;
- Моделі на основі руху;
- Нейронні мережі.

Розглянемо деякі існуючі моделі на основі зовнішнього вигляду.

а) Модель на основі частин виконує розбиття об'єкта на менші складові частини. Кожна частина моделюється окремо, і просторові відмінності, такі як розміри та взаєморозміщення між цими частинами, використовуються для виявлення та класифікації об'єктів. На прикладі використання для виявлення транспортного засобу, транспортний засіб розбивається на передню, задню та бокову частини. Кожна частина має свої характеристик, які потім використовуються для визначення присутності на зображенні.

б) Метод на основі ознак використовує репрезентативні описи цих ознак, що характеризують візуальний вигляд об'єкту. Ознаки, отримані на основі зовнішнього вигляду використовуються для визначення особливостей об'єкта, які роблять його унікальним. Наприклад при розгляді автомобіля, деякі з його ознак, такі як форма кузова, форма та колір номерного знаку перетворюються на більш загальні описи, які будуть використовуватися для виявлення об'єкту на зображенні.

Тепер розглянемо основні методи для виділення ознак на основі руху об'єкту.

а) Метод розрізнєння кадрів.

Цей метод базується на обчисленні різниці пікселів між послідовними кадрами для виділення рухомої області переднього плану. Якщо пікселі змінилися, це може означати що на зображенні відбулися зміни, тобто міг відбутися рух

(рис. 1.5). Він є простим у реалізації і часто застосовується для виявлення рухомих об'єктів на зображенні. Проте є дуже чутливим до шумів та змін освітлення через що потребує використання порогових значень та відсікання шумів.[6]

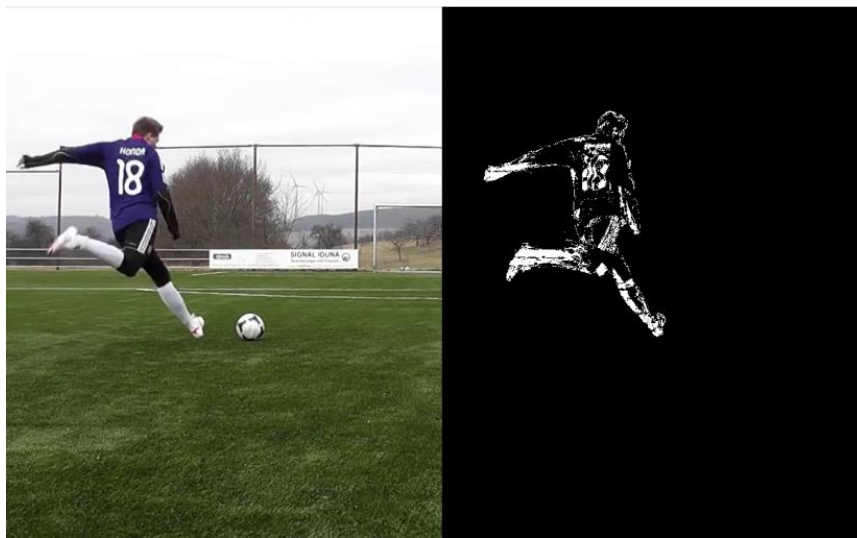


Рисунок 1.5 – Приклад виконання методу розрізнення кадрів[7]

б) Метод віднімання фону.

На відміну від методу розрізнення кадрів, даний метод використовує процес відокремлення об'єктів з переднього плану від фону, який формується шляхом усереднення послідовності зображень з метою створення моделі фону. Таким чином метод виділяє відмінності між поточним кадром і фоном, підкреслюючи рухомі об'єкти або передній план (рис. 1.6). Метод віднімання фону ефективно працює при умовах, де фон відносно однорідний, через що у випадках нестійких, динамічних фонів, такий метод не бажано використовувати.[6]

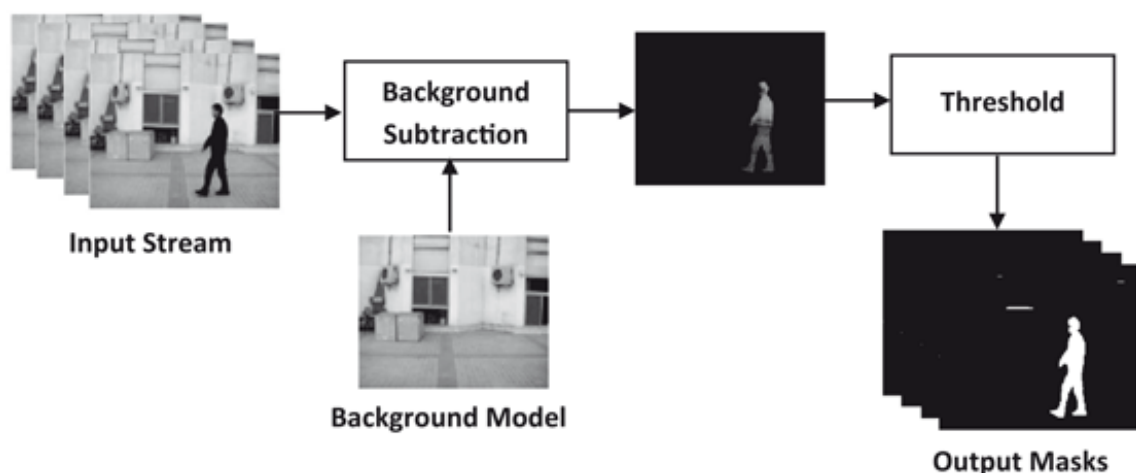


Рисунок 1.6 – Метод віднімання фону[8]

Серед нейронних мереж найчастіше для задачі розпізнавання зображень в сучасних системах використовується саме згорткова нейронна мережа (CNN) та її модифікації. Системи під керуванням CNN на даний момент демонструють великий успіх у використанні в системах з розпізнаванням в режимі реального часу. Нейронні мережі демонструють кращу гнучкість, кращу точність у задачах розпізнавання об'єктів, проте вимагають більше обчислювальної потужності.[6]

1.3.2 Тренування системи розпізнавання

Існує три моделі тренування системи розпізнавання:

а) Навчання з вчителем.

У даному типі навчання алгоритми використовуються для розрізнення об'єктів різних категорій на основі раніше позначених фотографій. На таких знімках об'єкт був попередньо позначений людиною, щоб навчити систему розпізнавати об'єкти на інших знімках. При цьому підході вхідні дані явно маркуються категоріями перед тим, як потрапляють до системи. З переглянутих методів є найточнішим і легке для навчання моделі, проте вимагає маркування даних і не може розпізнавати невідомі об'єкти.

б) Навчання без вчителя.

При навчанні без вчителя модель отримує набір немаркованих зображень і шляхом аналізу їхніх характеристик, атрибутів та особливостей визначає важливі схожості або відмінності між ними. Тоді система самостійно вивчає структуру та залежності у немаркованих даних для подальшого розпізнавання. Дана модель навчання не потребує попереднього маркування даних і може знаходити взаємозв'язки, які були непомічені людиною, проте вимагає більше часу та обчислювальних ресурсів, а також не може надати конкретної інформації про виявлену категорію.

в) Самонавчання.

Самостійне навчання є підмножиною неконтрольованого навчання і також використовує немарковані дані. Навчання здійснюється шляхом використання штучних міток, створених самою моделлю з немаркованих даних, які використовуються для навчання та розпізнавання. Модель корисна при потребі навчитися представляти інформацію з менш точних даних. Дана модель може допомогти розширити набір маркованих даних для тренувань моделей. Має спільні слабкі сторони з навчанням без вчителя.[9]

1.3.3 Основні задачі розпізнавання

До основних фундаментальних задач розпізнавання об'єктів можна віднести наступні операції:

- Класифікація
- Виявлення
- Сегментація

Класифікація відповідає за визначення приналежності одного об'єкту або кількох об'єктів на зображенні до якогось заздалегідь визначеного класу (рис.1.7).

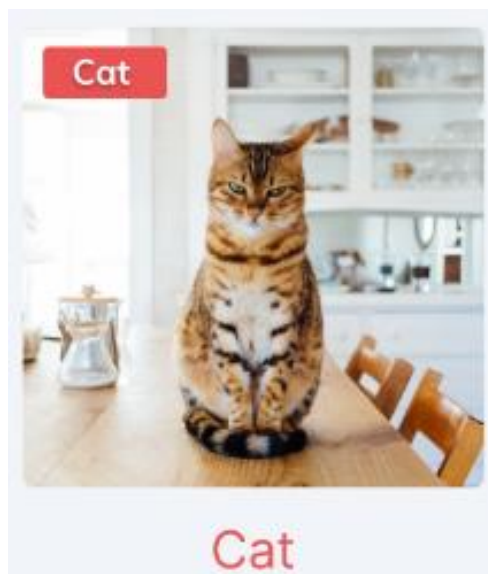


Рисунок 1.7 – Класифікація зображення[10]

Класифікація зображення відповідає за обробку всього зображення та присвоєння йому певної мітки на основі вмісту зображення.

Виявлення являє собою поєднання методу класифікації з локалізацією, яка вказує на місце знайденого об'єкта на зображенні (рис. 1.8). Задача виявлення полягає в знаходженні місця об'єкта, його розмірів, орієнтацію, положення та його клас.



Рисунок 1.8 – Виявлення об'єктів[10]

Сегментація зображення є призначенням міток або класів кожному пікселю зображення. Таким чином сегментація розділяє зображення на області, кожна з яких має свою форму, межі та клас. Це прибирає непотрібні пікселі з зображення при бажанні виділення лише конкретного об'єкта. Сегментацію можна поділити на семантичну сегментацію та сегментацію екземплярів (рис. 1.9). В першому випадку всі об'єкти будуть виділені та класифіковані як один клас, проте без розрізнення окремих екземплярів класу. Сегментація екземплярів є поєднанням виявлення та семантичної сегментації. Окрім сегментації ще також виконується розрізнення декількох екземплярів одного класу. Кожному екземпляру присвоюється своя мітка, завдяки чому дозволяється точно локалізувати та розрізнити об'єкти на зображенні. [10]

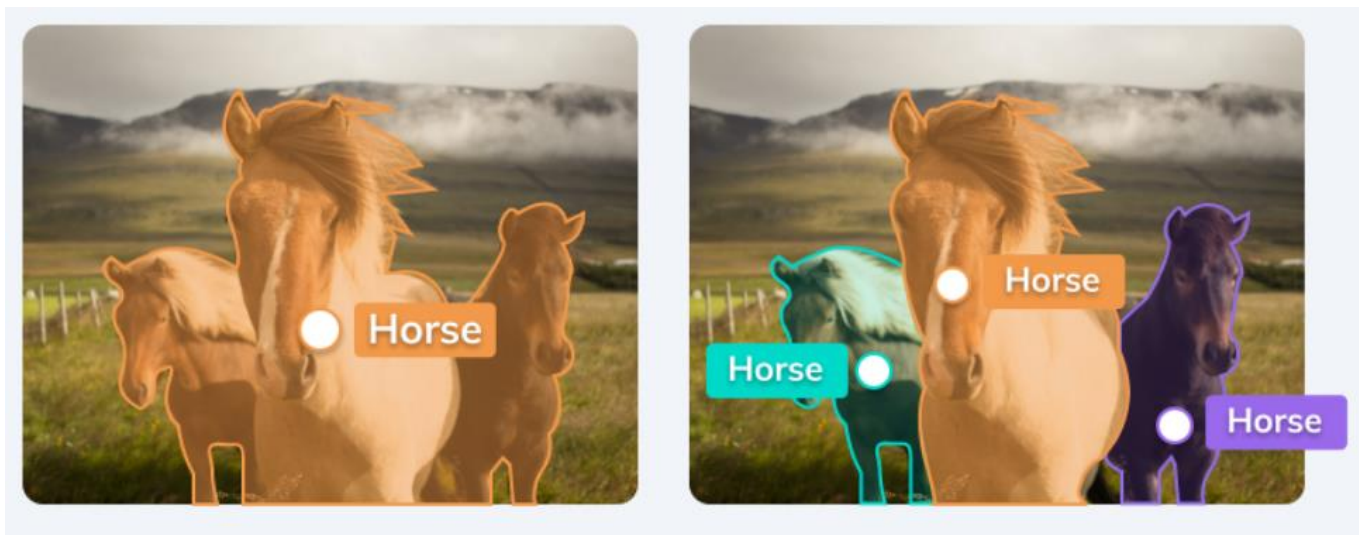


Рисунок 1.9 – Семантична сегментація та сегментація екземплярів[10]

1.4 Постановка задачі

Основним завданням є написання програми, яка зможе використовуватися на платних автобанах для зчитування типу та номеру транспортного засобу.

Задача передбачає виявлення об'єктів та їх класифікацію. Конкретно виділяються номерний знак та транспортний засіб. Отже, в нас буде використовуватися виявлення.

Дослідивши методи та підходи до вирішення задачі розпізнавання, можна вирішити як буде працювати програма. Розглянувши підходи до виділення ознак, було вирішено використовувати нейромережі для класифікації об'єктів.

Для навчання нашої нейромережі використаємо підхід навчання з вчителем. Тренування моделі буде відбуватися на попередньо підготовленому датасеті з фотографіями, на яких об'єкти були попередньо марковані. Кожен маркований об'єкт визначений класом, який нас цікавить.

Результати зберігатимуться у JSON-форматі, після чого його можна відправляти на сервер зберігання та обробки інформації для нарахування плати за користуванням автобаном транспортному засобу, який було ідентифіковано.

На даний момент планується створення системи, кожна камера якої відповідає за одну дорожню лінію, в одному напрямку дороги. В рамках подальшого розвитку планується покращити систему, використовуючи по одній камері високої роздільної здатності для кожної сторони руху, таким чином спростивши реалізацію проекту. Також можлива додаткові проектні рішення для збільшення сфер використання даного програмного забезпечення.

1.5 Висновок до розділу 1

В даному розділі спочатку було розглянуто актуальність розпізнавання та ідентифікування транспортних засобів. Розглянули сфери використання даної технології та обрали платні автомагістралі як сферу застосування.

Було досліджено існуючі методи стягування платні за користування платними автобанами. Кожен з даних методів був розглянутий, виділені основні переваги та недоліки.

Наступним етапом було дослідження розпізнавання образів. Ми розглянули такі етапи, як виділення ознак на зображенні. Було досліджено різновиди виділення ознак, а також різні методи кожного виду. Для кожного методу були розглянуті їх переваги та недоліки. Обрано найкращий.

Наступним етапом розглянули способи навчання системи для розпізнавання. Для кожної також були розглянуті переваги та недоліки і їх сфери застосування.

Розглянули основні задачі розпізнавання. Як вони відбувається та що роблять. В яких випадках використовуються.

Останнім етапом було поставлено задачу роботи. З досліджень, проведених в першому розділі, зробили план подальшої роботи та методів, якими будемо користуватися.

РОЗДІЛ 2 ТЕОРЕТИЧНІ ОСНОВИ НЕЙРОННИХ МЕРЕЖ ДЛЯ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ

2.1 Нейронні мережі

Штучна нейронна мережа (ANN) – це обчислювальна модель машинного навчання, яка складається з сукупності штучних нейронів, які поєднані між собою зв'язками. Штучні нейрони працюють в одній нейронній мережі для навчання та обробки вхідних даних.

Група нейронів, що розташована на одному рівні як етап обробки даних називається шаром. Нейрони попереднього шару з'єднані з нейронами наступного. Таким чином, після отримання вхідного сигналу на нейрон, відбувається обчислення та передача його результату до всіх підключених вузлів. Зазвичай нейронна мережа складається з трьох шарів: вхідний, вихідний та прихований (рис. 2.1).

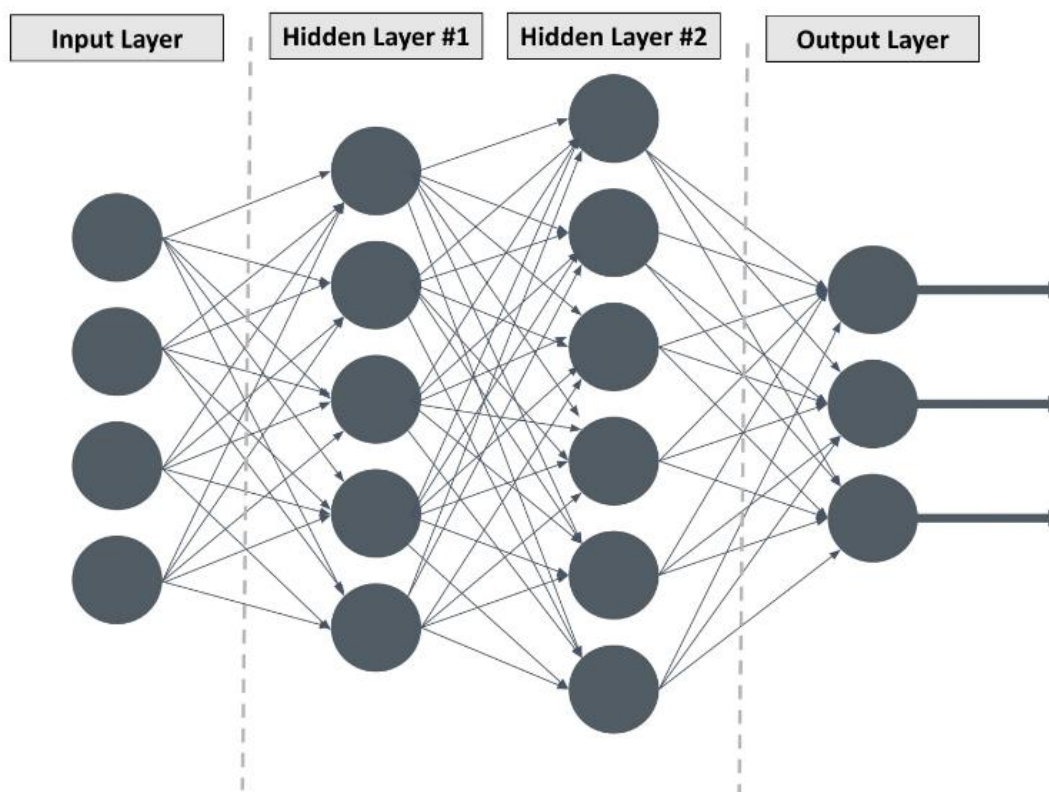


Рисунок 2.1 – Структура нейронної мережі [11]

В залежності від шару, в якому перебуває штучний нейрон, він може мати різні компоненти. Нейрон має наступну структуру:

- Один або декілька вхідних зв'язків (нейрони всіх шарів);
- Вага вхідного зв'язку (нейрони прихованого та вихідного шарів);
- Загальна сума (нейрони прихованого та вихідного шарів);
- Зміщення (нейрони прихованого та вихідного шарів);
- Функція активації (нейрони прихованого та вихідного шарів);
- Вихідний сигнал (нейрони всіх шарів).

Вхідний шар являється першим шаром нейронної мережі. Він не виконує ніяких обчислень. Його задача полягає в прийнятті вхідних сигналів та розподіленні їх по інших нейронах по зв'язках. Даний шар є спільним для усіх видів нейронних мереж. Кількість нейронів у вхідному шарі є кількістю вхідних ознак.

Прихованих шарів може бути один або декілька, та від їх кількості залежить глибина нейронної мережі. Вони виконують обчислення над вхідними даними та передають їх далі.

Вихідний шар відповідає за генерації вихідних прогнозів на основі результату прихованих шарів. Функція активації вихідного шару може відрізнятися від функції активації прихованих шарів в залежності від типу задачі.

2.1.1 Недоліки використання штучної нейронної мережі при розпізнаванні зображень

Хоч ANN і є потужним інструментом для прогнозування та прийняття даних на основі вхідної інформації, проте має певні обмеження, через що ANN не використовується в сучасних системах розпізнавання.

Однією з проблем ANN є її обмежена масштабованість. Через те що ANN потребує великої кількості параметрів для навчання, це може призвести до високого обчислювального навантаження та складності навчання.

Для комп'ютера зображення в колірній моделі RGB складається з трьох різних матриць. Кожна матриця представляє червоний, зелений і синій компоненти кольору для кожного пікселя зображення. При розгляді зображення розмірності n на n , ми отримуємо три окремі матриці тої ж розмірності (рис. 2.2). Перша матриця відображає червоний компонент для кожного пікселя, друга матриця – зелений компонент, а третя – синій. Кожен елемент цих матриць містить числове значення, що представляє інтенсивність кожного кольору для певного пікселя зображення.[12]

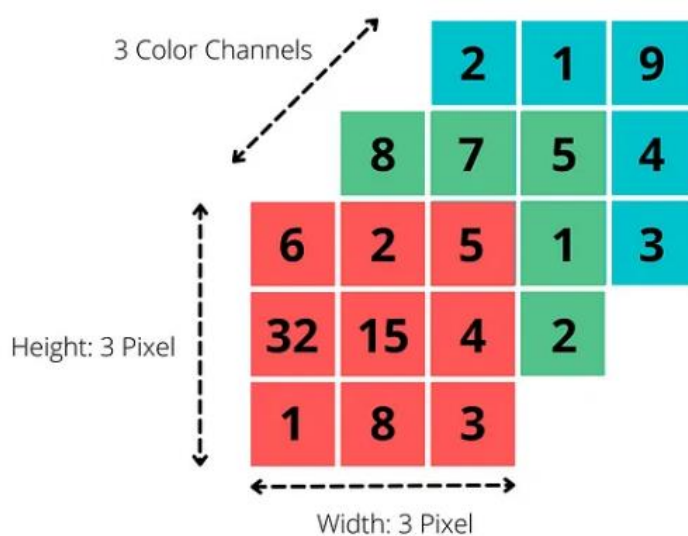


Рисунок 2.2 – Подача зображення RGB в обчислювальній машині[13]

Через це, для обробки зображень розміром 400 на 400 пікселів з трьома кольоровими каналами RGB, кількість ваг першого прихованого шару становитиме 480000. Це вимагає значних обчислювальних ресурсів і може спричиняти проблеми з перенавчанням.[12]

ANN потребує великої кількості маркованих навчальних даних, що може бути проблемою при наявності їх в малій кількості. Перенавчення моделі призводить до того, що моделі важче розпізнавати випадки, які не були в її тренувальній вибірці. Також після цього модель стає дуже чутлива до вхідних даних, через що невеликі зміни в зображенні можуть призвести до хибного розпізнавання.

Через велику кількість параметрів ANN не буде враховувати просторове положення об'єктів на зображенні, через що можуть бути суттєві обмеження при роботі з зображеннями, оскільки взаємне розташування об'єктів часто несе важливу інформацію.[13]

2.2 Згорткові нейронні мережі

Для того щоб вирішити ці проблеми, було вирішено доробити класичну нейронну мережу. Таким чином, з'явилася загорткова нейронна мережа (CNN). При її створенні, було вирішено використовувати підхід, який імітує сприйняття зображення людиною. Підсвідомо зображення розбивається на більше підзображень, після чого ми аналізуємо кожне для встановлення повної картини.[13]

Однією з основних задач CNN полягає в тому, щоб зменшити розмірність зображень, які мають високу роздільну здатність. Таке зменшення має зберігати основні особливості зображення для отримання хорошого прогнозування. Саме цим і займається операція згортки.

Згорткові нейронні мережі завдяки застосуванню відповідних фільтрів дозволяють вловлювати просторові та часові залежності на зображенні між об'єктами. Так фільтри захоплюють просторові особливості зображення, такі як краї, кути і текстури. Це відбувається завдяки використанню згорткових шарів, що пропускають фільтр через зображення. Зменшення кількості задіяних параметрів та повторне використання вагових коефіцієнтів допомагає моделі краще узагальнювати нові зображення. Повторне використання вагових коефіцієнтів дозволяє моделі вивчати більш складні ознаки з даних.[14]

Архітектура згорткової нейронної мережі також складається з трьох типів шарів: вхідний, вихідний та приховані шари. Приховані шари в свою чергу складається з:

- Шари згортки;

- Шари ReLU (шари активації);
- Шари агрегації;
- Повнозв'язний шар.

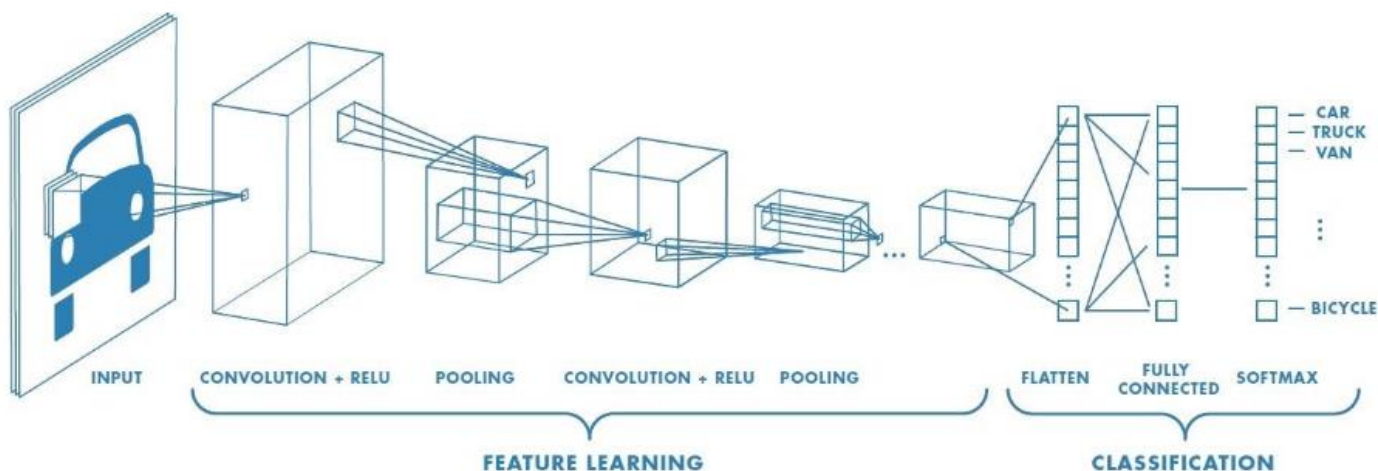


Рисунок 2.3 – Структура шарів згорткової нейронної мережі[14]

Усі види шарів можуть повторюватися кілька разів і їх порядок відповідає певним правилам. Мережа починається зі згорткового шару та закінчується повнозв'язним шаром. Перед повнозв'язним шаром зазвичай використовується завершальний шар агрегації.

2.2.1 Шар згортки

Шар згортки є першим шаром що витягує ознаки зображеного об'єкта. В цьому шарі використовується фільтр, або як його ще називають ядром, який проходить по всьому зображенню і виконує операцію згортки.

Фільтр у згортковій нейронній мережі є матрицею числових ваг, яка використовується для виявлення конкретних ознак у вхідних даних. Кожен фільтр може бути спеціалізованим на виявлення певного типу ознак, таких як краї, текстури або форми. Під час процесу навчання мережі ваги фільтрів оптимізуються

таким чином, щоб ефективно виділяти ці ознаки у вхідних даних. Кожен фільтр має невеликий розмір, наприклад матрицю 3 на 3 або більше. Фільтр рухається по зображенню з певним кроком, наприклад 1 піксель або більше (рис. 2.4). При кожному кроці фільтр виконує згортку з частинкою зображення, що знаходиться під ним.

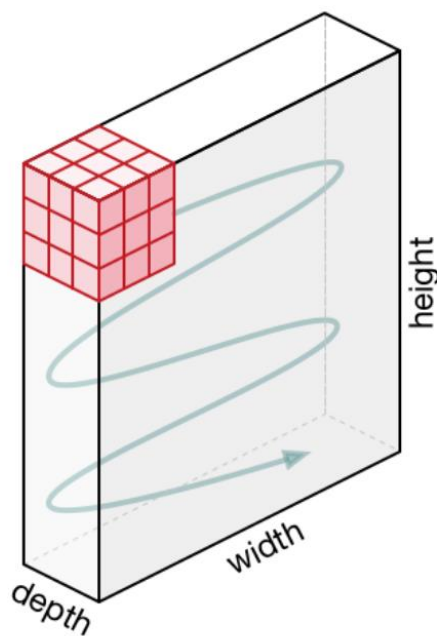


Рисунок 2.4 – Рух фільтру по зображенню з трьома кольоровими каналами[14]

Згортка – це операція об’єднання двох наборів інформації для створення третього. В CNN цими наборами виступають вхідні дані та фільтр. Згортка включає множення відповідних пікселів зображення на відповідні ваги фільтра і сумування отриманих значень (рис. 2.5). Цей процес повторюється для кожного фрагмента зображення, що потрапляє під фільтр. Результатом згортки виходить карта ознак.

Карта ознак відображає виявлені особливості на зображенні. Кожен піксель на карті ознак відповідає реакції фільтра на відповідний фрагмент зображення. Це дозволяє виокремлювати різні ознаки, такі як гострі краї, текстури або форми, що присутні на зображенні.

У процесі згортки, деякі ознаки можуть бути втрачені, але головні та унікальні ознаки зберігаються у вихідному масиві карти ознак. Щоб уникнути втрати важливої інформації, використовуються різні фільтри для створення багатьох карт

ознак. Кожна карта ознак, створена фільтром, здатна виділяти положення конкретних ознак, які присутні на вхідному зображенні. Таким чином, за допомогою різних фільтрів можна зберегти широкий спектр важливих ознак зображення. [11][13-15]

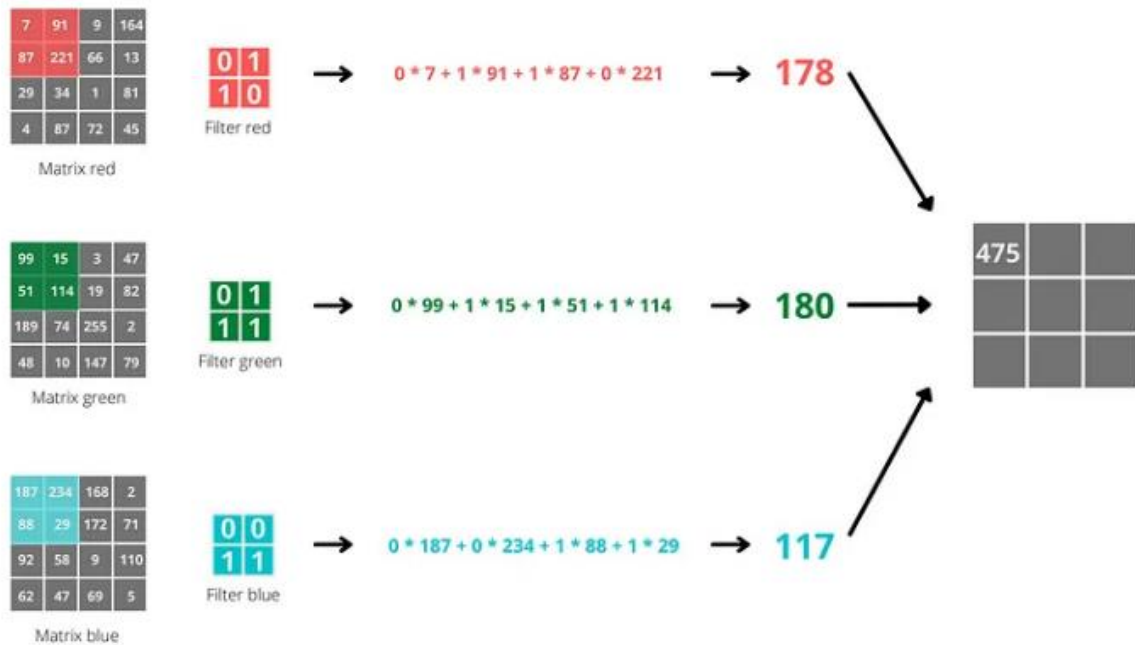


Рисунок 2.5 – Операція згортки[13]

2.2.2 Шар активації

Після проходження згортки, вихідні значення піддається функції активації. Згортка є лінійною операцією і робить лінійне перетворення входу. Без застосування шару активації мережа стане лінійною і не зможе вивчати складні нелінійні залежності та особливості на зображенні і узагальнювати нові дані. За допомогою функції активації можна додати нелінійності мережі, що дозволить їй вивчати складніші закономірності.[16]

В процесі розвитку CNN використовувалися наступні функції активації:

- Сигмоїдна функція;

- Гіперболічний тангенс;
- ReLU.

Сигмоїдна функція (рис. 2.6) визначається наступним чином:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (2.1)$$

де e – експонента.

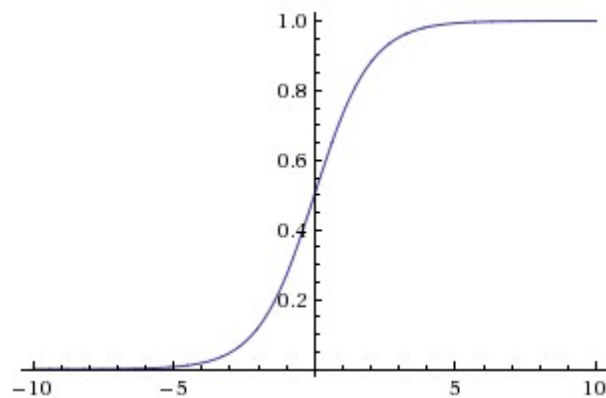


Рисунок 2.6 – Сигмоїдальна функція[17]

Сигмоїдна функція відображає будь-яке вхідне число у значеннях між 0 та 1. Велике від'ємне число наближається до 0, а велике додатне наближується до 1. Дана функція використовувалася оскільки вона мала швидкий та плавний перехід нейрона з 0 до 1.

Серед недоліків сигмоїдальної функції є проблема зникаючого градієнта. При великих від'ємних або додатних значеннях функції, результати функції по відношенню до вхідних значень не будуть сильно змінюватися. Це призводить до насичення функції і градієнт в цих ділянках стає дуже малий. За зникненням мається на увазі що малий градієнт функції активації буде помножений на градієнт функції втрат при зворотному поширенні для оновлення ваг. Якщо градієнт функції активації дуже малий, він буде наближатися до нуля загальний градієнт, а це означає, що сигнал не буде проходити через нейрони до ваг і вони не будуть оновлюватися ефективно. Це призводить до зупинки, або дуже повільного, навчального процесу.

Другою проблемою даного методу є те, що його виходи не центровані навколо нуля. Якщо наприклад всі вихідні дані позитивні, то градієнти під час зворотного поширення також будуть позитивними або негативними в залежності від градієнта функції активації. Тому під час оновлення ваг усі вони будуть або збільшені, або зменшені, і градієнтний потік стане зигзагоподібним. Як наслідок, ваговий вектор вимагатиме більше оновлень для ефективного навчання. Це означає, що кількість епох, необхідних для досягнення прийняттого рівня навчання, збільшується.[16-17]

Через це, потребується попередня підготовка даних, щоб середнє значення дорівнювало нулю.

Наступним варіантом є функція гіперболічного тангенсу (рис. 2.7). Можна сказати, що це масштабована сигмоїдальна функція:

$$\tanh(x) = 2\sigma(2x) - 1 = \frac{1}{1 + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} \quad (2.2)$$

На відміну від сигмоїдальної функції, дана функція стискає вхідне значення до діапазону $[-1, 1]$. Через це в неї немає проблеми з центруванням навколо нуля. І хоч тут також наявна проблема вмираючого градієнта, проте для гіперболічного тангенсу він сильніший, оскільки має ліпшу похідну. Через наявність експоненційної функції в даних алгоритмах він потребує більше ресурсів для обчислення.

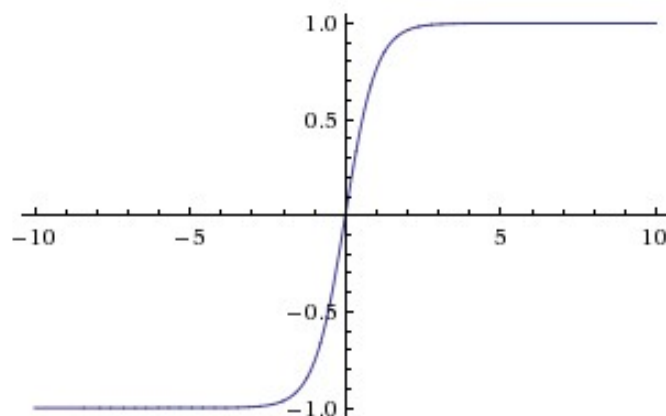


Рисунок 2.7 – Функція гіперболічного тангенсу[17]

Нині найчастіше використовується саме функція ReLU (рис. 2.8):

$$f(x) = \max(0, x) \quad (2.3)$$

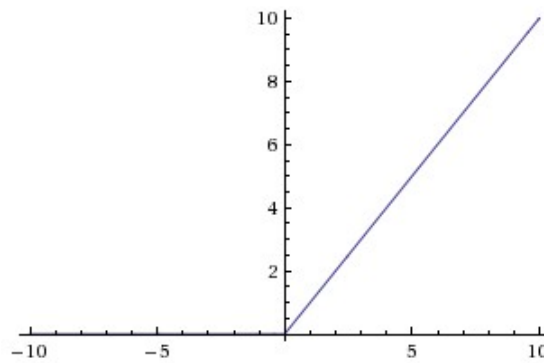


Рисунок 2.8 – ReLU функція[17]

Вона стала поширеною саме через свої переваги над вище розглянутими функціями. Похідна ReLU завжди дорівнює 0 при значеннях менших за нуль та 1 при значеннях які більше за нуль, за допомогою чого уникаємо вмирання градієнта при зворотному поширені. Також ReLU є простішим для використання та не потребує важких розрахунків.[16]

Оскільки функція виводить 0 при вхідних від’ємних даних, це дозволяє активувати менше нейронів, що призводить до розрідженості мережі. Менше нейронів буде проводити розрахунки та передавати сигнал далі, що зменшує погірбні для обчислень ресурси.

Проте ця особливість не завжди приносить переваги. У випадках, коли більшість нейронів мережі повертає нуль на виході, це викликає проблему “вмираючого ReLU”. Градієнт перестає проходити через них і ваги перестають оновлюватися, через що мережа не може продовжувати навчання. Коли нейрон застрягає у від’ємному діапазоні і видає 0, через нахил нахил ReLU він вже не зможе відновитися.[18]

Двома головними причинами вмирання ReLU є великий швидкість навчання, через що нові ваги можуть опинитися у дуже від’ємному діапазоні, та велике від’ємне зміщення.

Для того щоб позбавитися проблеми “вмираючого ReLU”, був створений Leaky ReLU (рис. 2.9), головною особливістю якого є використання малих невід’ємних ненульових значень, наприклад 0.01, на від’ємному відрізку[17]:

$$f(x) = \max(\alpha x, x) \quad (2.4)$$

Таким чином, замість того щоб приймати набувати значення 0 при $x < 0$, функція матиме невеликий від’ємний нахил. Завдяки цьому, при від’ємних значеннях забезпечується активація нейрону і він не помирає. Це підтримує здатність мережі до навчання.

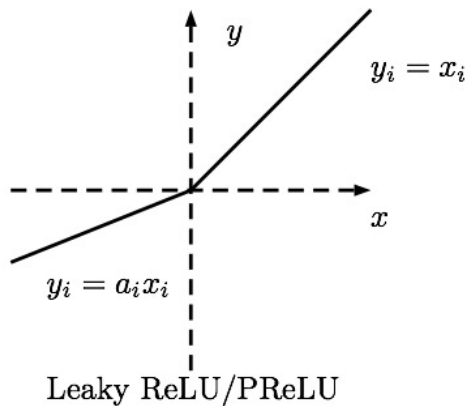


Рисунок 2.9 – Leaky ReLU функція[17]

2.2.3 Шар агрегації

Основним завданням даного шару є зменшення розмірності вхідних карт ознак, при цьому зберігши головні характеристики, при цьому зменшуючи кількість потрібних параметрів та обчислень.

Основними двома методами агрегації є усереднювальне та максимізаційне агрегування (рис. 2.10). Максимізаційне агрегування повертає максимальне

значення кожної досліджуваної області. Усереднювальне повертає середнє значення досліджуваної області.

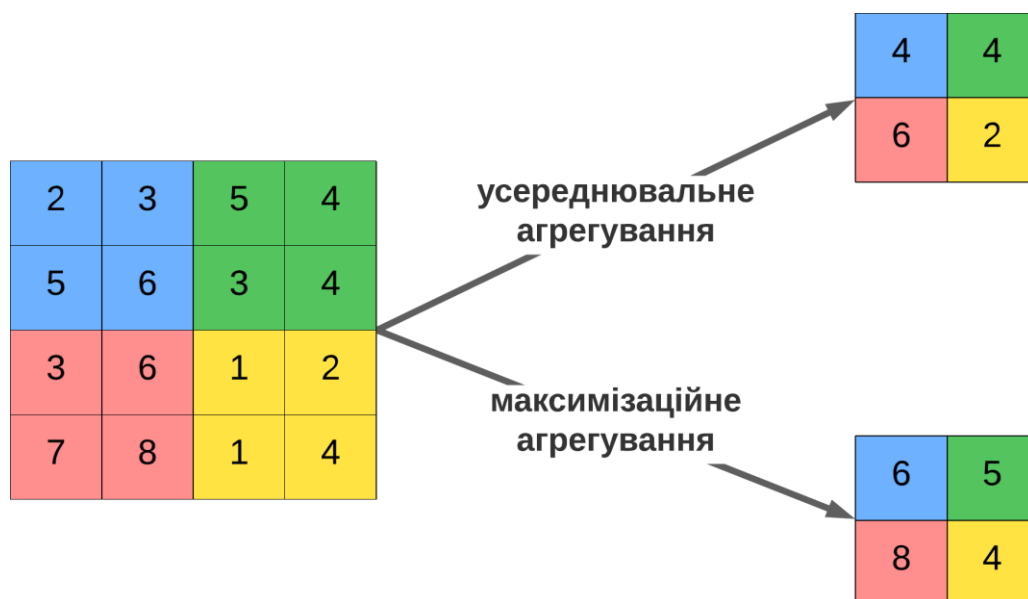


Рисунок 2.10 – Робота агрегації

Обидва методи фільтрують шум на зображенні, який не впливає на розпізнавання. Проте на практиці частіше використовується саме максимізаційне агрегування через його додаткові переваги. Після максимізаційного агрегування отримуємо карту ознак з найбільш домінуючими ознаками, які є незмінними до змін положення або обертань об'єкту. Також це дозволяє уникнути перенавчання.[15]

2.2.4 Повнозв'язний шар

Повнозв'язні шари являються останніми шарами мережі. Вхідними даними зазвичай є результати останнього шару агрегації, які проходять через етап розгортання. Об'єднана карта ознак, яка представляється трьохвимірною матрицею, перетворюється в одновимірний вектор шляхом розгортання елементів в послідовний ряд.

У повнозв'язному шарі мережі кожен нейрон з попереднього шару пов'язаний з кожним нейроном у наступному шарі. Головною задачею повнозв'язного шару є прогнозування за отриманими ознаками на основі навчальних даних. Кожне значення вхідного вектора представляє ймовірність належності ознаки до якогось класу. За кілька епох навчання модель здатна розпізнавати важливі та низькорівневі ознаки на зображеннях. Зворотне поширення помилки використовується для налаштування ваг моделі під час навчання, щоб знизити помилки та покращити точність класифікації.

У випадках бінарної класифікації, для визначення класу може використовуватися сигмоїдна функція активації. Вона буде показувати ймовірність належності до одного з двох класів.

Для багатокласової класифікації використовується функція активації softmax. Вона генерує розподіл ймовірностей приналежності до кожного з класів. Softmax бере вектор оцінок і стискає його до вектора значень, кожне з яких знаходиться в діапазоні від 0 до 1, і сума усіх значень становить 1.[15]

2.2.5 Оптимізатори

Одним з параметрів, який показує наскільки добре навчилася нейромережа є функція втрат. Функція втрат використовується для розрахунку різниці між передбаченим значенням моделі та справжнім значенням. Для гарного прогнозування функцію втрат потрібно зменшувати. Для цього використовують оптимізатори.

Оптимізатори використовуються для зміни параметрів в мережі, таких як швидкість навчання та її ваги для мінімізації функції втрат. Основною задачею оптимізаторів є знаходження оптимальних параметрів для забезпечення найкращої точності та зменшення втрат мережі.

Існує багато різновидів різних оптимізаторів (рис. 2.11). Розглянемо найбільш популярні методи, серед яких є SGD, Momentum, Adam.

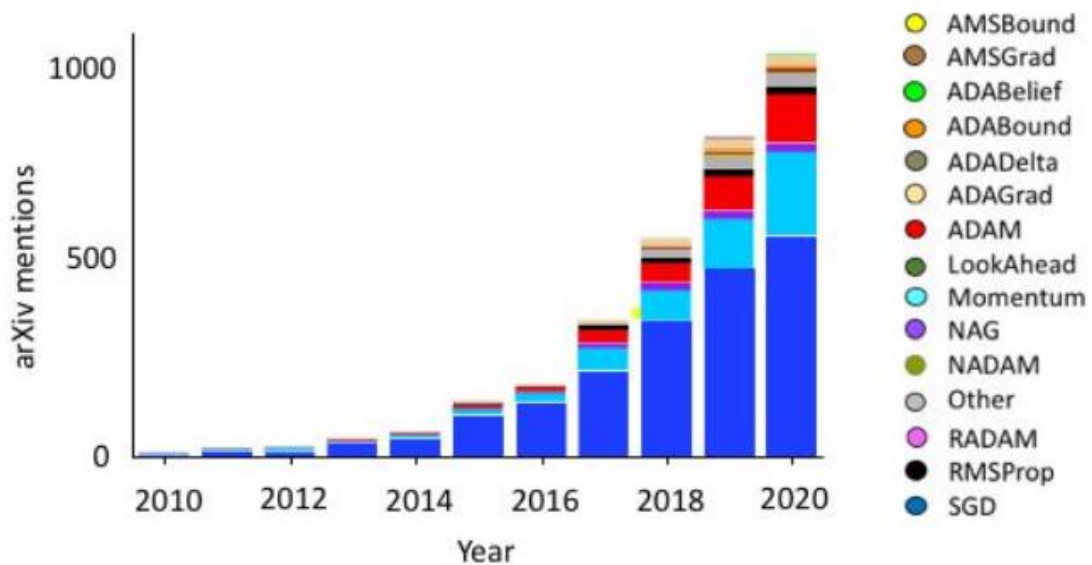


Рисунок 2.11 – Найбільш згадувані оптимізатори в онлайн-архіві arXiv[19]

а) Градієнтний спуск.

Градiєнтний спуск (SG) є одним з найбільш простих та поширених алгоритмів оптимізації, що використовується для оновлення параметрів моделі. Це ітераційний алгоритм оптимізації першого порядку для пошуку мінімум функції, яка в нашому випадку є функцією втрат мережі. Першим кроком є обчислення градієнту, який є похідною першого порядку від функції в теперішній точці. Далі відбувається переміщення у протилежному напрямку до збільшення градієнта, від теперішньої точки на обчислену величину. Метою є ітераційний рух до локального мінімуму, мінімізуючи втрати.[20] Даний алгоритм можна описати наступним чином[21]:

$$\theta = \theta - \eta \nabla J(\theta) \quad (2.5)$$

де η – параметр швидкості навчання;

θ – значення вагів;

$\nabla J(\theta)$ – градієнт функції втрат відносно значення вагів θ .

Основними перевагами градієнтного спуску є те, що його легко обчислювати та реалізувати. Проте, даний алгоритм може зациклюватися на локальних мінімумах. Також він потребує обчислення градієнта для всього набору даних, щоб змінити ваги. У випадку з великими наборами даних, це призводить до довгого очікування та потреби великої кількості пам'яті для обчислення.[22]

б) Стохастичний градієнтний спуск.

Для уникнення цих проблем було розроблено стохастичний градієнтний спуск (SGD). В ньому розглядається лише один приклад за раз, і через це функція втрат буде сильно коливатися на навчальних прикладах. Алгоритм роботи однієї епохи можна описати наступним чином[20]:

- 1) Обирається приклад з набору даних;
- 2) Вибраний приклад пропускається через мережу і обчислюється результат;
- 3) Обчислення градієнту функції втрат та використання його для оновлення вагів мережі;
- 4) Повторення кроків 1-3 для всіх прикладів з набору.

Алгоритм SGD записується наступним чином[21]:

$$\theta = \theta - \eta \nabla J(\theta; x_i; y_i) \quad (2.6)$$

де $\nabla J(\theta; x_i; y_i)$ – градієнт функції втрат відносно значення вагів θ з тренувальними прикладами $x_i; y_i$.

Частіше оновлення параметрів забезпечує швидшу збіжність моделі для даних великих розмірів. Оскільки використовується лише один приклад з вибірки, зменшується потреба в розмірі пам'яті та пришвидшується обчислення. Проте через великі коливання, алгоритм не зможе досягти мінімуму, проте буде знаходитися біля нього.[22]

в) Міні-пакетний градієнтний спуск.

Він є найкращою варіацією серед алгоритмів з градієнтним спуском. Досить часто саме він використовується при позначенні SGD при тренуваннях моделі. Замість того щоб використовувати весь набір даних чи тільки один приклад, весь набір даних ділиться на пакети з фіксованою кількістю прикладів. Алгоритм для виконання однієї епохи тут такий самий, як і для SGD, проте замість одного навчального прикладу ми використовуємо міні-пакет.[20] Відповідно, отримаємо наступну формулу [21]:

$$\theta = \theta - \eta \nabla J(\theta; B_i) \quad (2.7)$$

де $\nabla J(\theta; B_i)$ – градієнт функції втрат відносно значення вагів θ з тренувальними пакетами B_i .

Хоч даний алгоритм і потребує більше пам'яті ніж SGD, проте функція втрат в даному алгоритмі коливається набагато менше. Також розглядання пакетів з декількома прикладами дозволяє застосовувати векторну реалізацію для пришвидшення обчислення.[22]

Розглянемо алгоритми оптимізації градієнтного спуску.

а) Momentum.

Даний метод використовується для того щоб зменшити коливання функції втрат. Для цього використовується моментум, який визначається гіперпараметром. Він контролює швидкість накопичення моментуму під час оновлення параметрів мережі.

Мета моментуму полягає в тому, що він прискорює збіжність у важливих напрямках та зменшує коливання в незначних. Це досягається шляхом накопичення попередніх градієнтів і використання їх для впливу на оновлення параметрів мережі. Моментум дозволяє запам'ятовувати попередні напрямки зміни параметрів і надавати їм більшу вагу під час наступних оновлень.[21]

Алгоритм записується наступним чином[21]:

$$\theta = \theta - V_t; \quad V_t = \gamma V_{t-1} + \eta \nabla J(\theta) \quad (2.8)$$

де V_t – моментум на кроці t ;

γ – гіперпараметр моментуму, який зазвичай дорівнює 0.9.

Таким чином, моментум збільшує оновлення вздовж вимірів, де градієнти спрямовані в однакових напрямках, і зменшує де градієнти змінюють напрямки.[22]

б) Adam

Даний алгоритм можна розглядати як комбінацію алгоритму адаптивної швидкості навчання та моментуму. Adam адаптує швидкість навчання індивідуально для кожного параметра в мережі.. Крім збереження експоненційно спадаючого середнього квадратичного градієнта, яке позначається v_t , Adam також зберігає експоненційно спадаюче середнє минулих градієнтів, яке позначається m_t . Це дозволяє алгоритму Adam адаптивно налаштовувати швидкість навчання для кожного параметра відповідно до його важливості та змінювати швидкість навчання залежно від градієнтів у попередніх ітераціях. Таким чином основною ідеєю методу є поєднання прискорення збіжності у важливому напрямі, забезпечуючи більш точну та ефективну оптимізацію. Обчислення значень m_t та v_t відповідно[23]:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad (2.9)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \quad (2.10)$$

де m_t – оцінка першого моменту, який є середнім градієнтів;

v_t – другий момент, який є нецентрованою дисперсією градієнтів;

β_1, β_2 – гіперпараметри, які відповідають за експоненційну швидкість спадання ковзних середніх m_t та v_t , зазвичай приймають значення 0.9 та 0.999 відповідно;

g_t – градієнт на кроці t .

m_t та v_t при ініціалізації як нульові вектори зсунуті до нуля на початкових часових кроках. Для коригування цих зсувів розраховуються скореговані оцінки даних моментів[21]:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}; \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (2.11)$$

Ми використовуємо дані результати для оновлення параметрів вагів за наступною формулою[21]:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \quad (2.12)$$

де $\epsilon = 10^{-8}$.

Хоч даний метод і є складним в розрахунках, проте він здатний до масштабування швидкості навчання в поєднанні зі зменшенням коливань та збільшенням точності. Розглянувши популярні алгоритми оптимізації, можна сказати, що Adam є найкращим з них для використання при навчанні мережі.

2.3 Вибір моделі розпізнавання

2.3.1 Метрики для вибору моделі

Існує декілька метрик, які допоможуть визначитися з вибором моделі. Основними метриками, які зможуть показати якість моделі є FPS та точності. Для визначення інших показників, які вкажуть на вимоги до обчислювальної системи, використовуються параметри кількості операцій з плаваючою комою в секунду FLOPs та кількість параметрів моделі.

Кількість параметрів в моделі вказують на параметри, які необхідно вивчати під час тренування моделі. Параметри можуть включати ваги та зсуви в нейронних мережах. Кількість параметрів є індикатором складності моделі і може використовуватися для порівняння обсягу пам'яті, необхідного для зберігання моделі. Чим більше параметрів, тим більше пам'яті вимагається для зберігання та тренування моделі.

FLOPs вимірює обчислювальну складність моделі та вказує на кількість операцій з плаваючою точкою, які модель виконує за секунду. FLOPs є показником обчислювальної складності моделі і може використовуватися для порівняння ефективності різних моделей. Чим більше значення FLOPs, тим більше ресурсів вимагається для виконання моделі.

Значення FPS демонструє, скільки кадрів в секунду може обробити модель для визначення об'єктів на них. У випадку з визначенням машин, цей параметр завжди має контролюватися. Для стабільної роботи системи, бажане значення FPS має знаходитися в районі 15-30, оскільки це забезпечує достатню плавність картинки. Досягнення більшої кількості буде ще кращим, проте для цього потрібна оптимізація моделі та використання потужнішого комп'ютера.

Для перевірки точності розпізнавання моделі, використовується показник середньої точності моделі mAP . mAP є середнім значенням AP всіх класів, які розпізнає модель. AP – це середня точність розпізнавання класу в моделі. Формула для розрахунку mAP записується наступним чином[24]:

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i \quad (2.13)$$

де N – загальна кількість класів, які розпізнаються моделлю;

AP_i – показник AP для класу i .

Для розрахунку AP потрібні попередньо визначені метрики:

– IoU;

- Точність (Precision);
- Повнота (Recall);
- PR-крива (Precision-Recall curve).

IoU – це метрика, що показує співвідношення між перетином областей, істинним та прогнозованим, та об'єднанням цих областей (рис. 2.12).

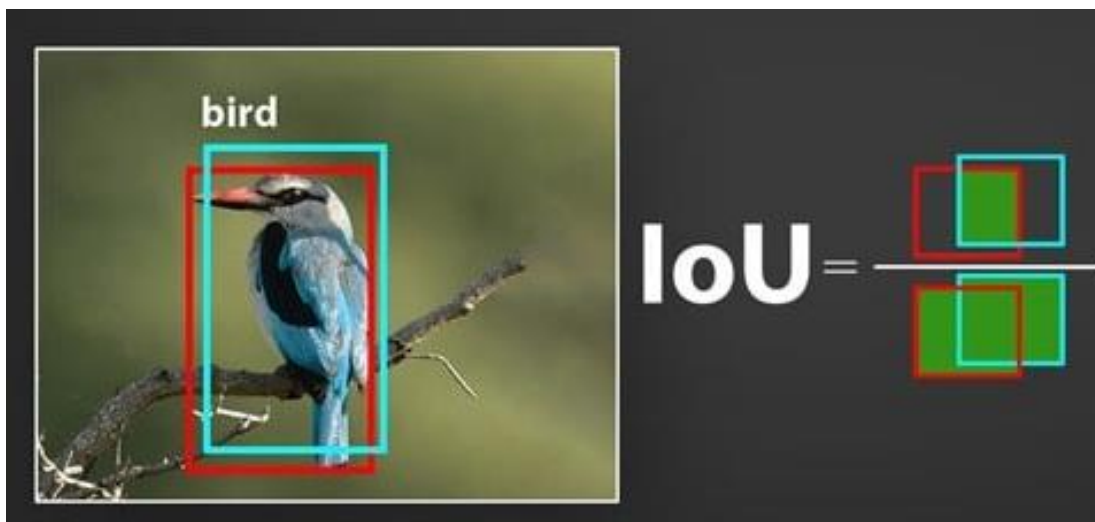


Рисунок 2.12 – Визначення метрики IoU[24]

Параметр IoU приймає значення α від 0 до 1, де 0 це відсутність розпізнавання, а 1 де прогнозована область співпала з істинною. Для визначення інших метрик зазвичай виставляється поріг $\alpha = 0.5$.

У випадку, коли α розпізнавання вища за виставлений поріг точності, таке прогнозування вважається вірно позитивним (True positive, TP). Якщо α менше порогу, або прогнозовано невірний клас, прогнозування вважається хибно позитивною (False positive, FP). У випадку відсутності будь-якого прогнозування для об'єкта, прогнозування позначається як хибно негативний (False negative, FN). Значення вірно негативний (True negative, TN) не використовується в розпізнаванні, оскільки нас не цікавить правильність визначення відсутності об'єкта (рис. 2.13).

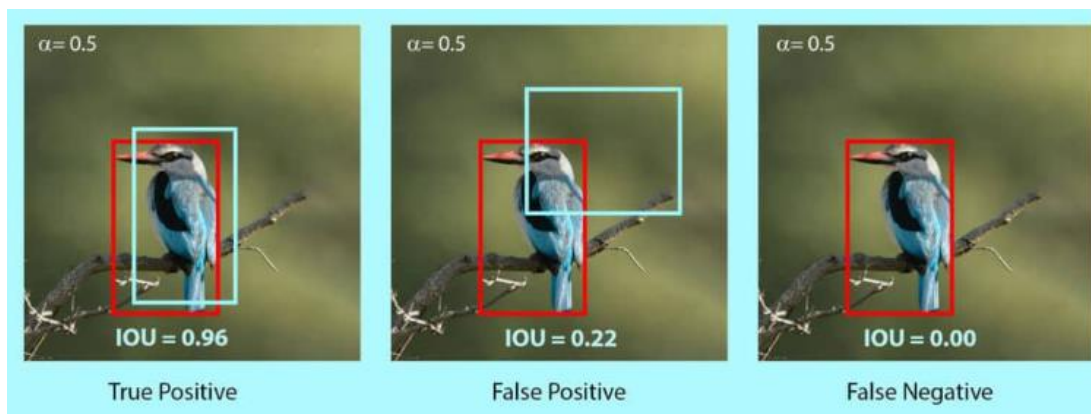


Рисунок 2.13 – Застосування IoU для визначення інших метрик[24]

Значення TP, FP, FN використовуються для визначення наступних метрик: точність та повнота.

Точність розраховується як співвідношення кількості вірно позитивних прикладів до загальної кількості об'єктів, які виявлені як позитивні $TP/(TP+FP)$. Повнота розраховується як відношення вірно позитивних прикладів до загальної кількості позитивних прикладів $TP/(TP+FN)$. Можна сказати, що точність показує нам наскільки точними є прогнози стосовно виявлених об'єктів, а повнота наскільки добре було виявлено всі позитивні випадки.[24]

Точність та повнота використовуються для побудування PR – кривої, яка показує, як добре модель може підтримувати високі показники класифікації позитивних випадків, зберігаючи при цьому високий рівень точності.

Для розрахунку AP, розраховується площа під PR – кривою класу. Такий підхід використовується в метриці PASCAL VOC 2010. Існує ще декілька підходів, в яких задано при яких порогах та яким чином розраховується AP. Ці підходи також мають свої датасети з фотографіями для перевірки якості моделі. Зазвичай в них входили базові класи для розпізнавання, по типу тварини, машини, люди. Розглянемо набори даних, які використовувалися та використовуються для перевірки якості роботи моделі:

а) PASCAL VOC 2005.

Даний підхід використовував замість площини під PR – кривої площину під ROC – кривою.

б) PASCAL VOC 2007.

Основною ідеєю даного підходу є використання одинадцяти точкової інтерполяції для зменшення впливу коливань PR – кривої на результат. Ділянка повноти ділиться на 11 рівнів. Для кожного рівня повноти шукається найбільша точність серед усіх порогових значень, величина яких перевищує або дорівнює поточному рівню повноти (рис. 2.14). Тоді AP розраховується як середнє суми одинадцяти інтерпольованих значень.[29]

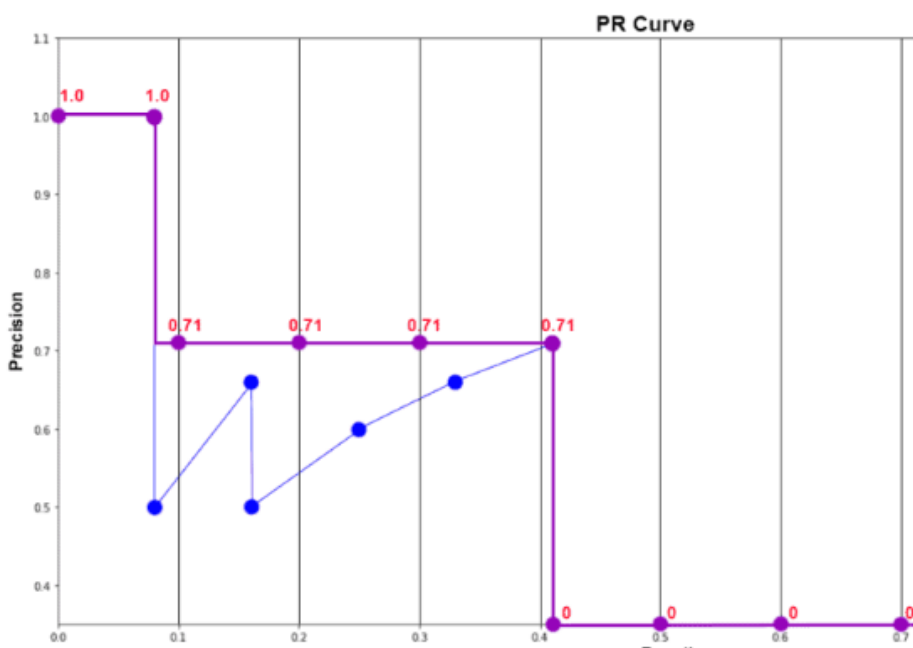


Рисунок 2.14 – Метод точкової інтерполяції[24]

в) MS COCO.

На даний момент MS COCO mAP є однією з найпоширеніших метрик для оцінки якості моделі. Даний підхід впровадив використання 101-бальної інтерполяційної версії AP. Замість оцінки при одному пороговому значенні для IoU, використовується 10 порогових значень, які мають діапазон від 0.5 до 0.95 при кроці 0.05.

З отриманих значень AP складається середнє значення mAP, яке і використовується для порівняння моделей.[24]

2.3.2 Підходи в моделях розпізнавання

В залежності від задачі, яка має виконуватися розпізнаванням, існують різні підходи та алгоритми для задоволення кожної з них. До основних задач виявлення об'єктів з зображення чи відеокадру можна віднести наступні два кроки:

- 1) Знайти об'єкти при їх наявності на зображенні;
- 2) Класифікувати кожен знайдений об'єкт, оцінити розмір використовуючи обмежувальну рамку.

В залежності від вимог до програми, ці кроки можна виконувати в один етап одночасно, або в два етапи. Таким чином, алгоритми поділяються на одноетапні та двоетапні.

а) Двоетапні моделі.

В двоетапних моделях, при першому проході пропонуються області знаходження об'єктів на зображенні. Такі області ще називається RoI (region of interest). Для визначення RoI використовуються методи комп'ютерного зору, або глибинні мережі. При другому проходженні ці області використовуються для вилучення ознак, які використовуються для класифікації об'єкта та уточнення пропонованих областей.

Двоетапні моделі досягають більшої точності в порівнянні з одноетапними за рахунок кращої локалізації об'єкта на зображенні. Це також дозволяє краще розпізнавати малі об'єкти або групу. Проте через більшу кількість операцій над зображенням, швидкість обробки кадрів FPS (frames per second) менша, ніж при використанні одноетапних моделей.

Програми які використовують двоетапні моделі, використовують обласні згорткові нейронні мережі (R-CNN) та його покращені версії (Fast-CNN, Faster-CNN, G-RCNN).[25]

б) Одноетапні моделі

На відміну від двоетапних моделей, одноетапні одразу визначають обмежувальні рамки об'єкта на зображенні, без попереднього пропонування областей на зображенні.

Даний алгоритм надає перевагу швидкості виявлення та класифікації, потребуючи менше часу на виконання меншої кількості операцій. Завдяки цьому одноетапні моделі є найкращим вибором для програм, які призначені для роботи в режимі реального часу.

Проте через відсутність поліпшеної локалізації об'єктів, як в двоетапних моделях, одноетапний не так добре справляється з розпізнаванням об'єктів неправильної форми чи групи невеликих об'єктів.

Серед найпопулярніших моделей, які використовуються для розпізнавання в режимі реального часу, є [25]:

- RetinaNet;
- SSD (Single Shot Detector);
- YOLO (You only look once).

2.3.3 Вибір моделі для розпізнавання в реальному часі

Оскільки ідеєю даної роботи є розпізнавання саме в реальному часі, порівняємо між собою різні алгоритми одноетапного визначення, оскільки вони будуть швидшими за двоетапні моделі. Проаналізуємо результати одноетапних моделей.

а) RetinaNet.

RetinaNet є однією з найбільш поширених одноетапних моделей для виявлення об'єктів. Вона розроблена таким чином, що вона оптимізує та збалансовує роботу однократних детекторів, щоб досягти більш ефективних результатів. В основі цієї архітектури лежить заміна перехресних ентропійних втрат,

що використовуються в попередніх моделях, на так звані фокусні втрати. На додачу до фокусних втрат, архітектура моделі заснована на використанні мереж ResNet і Feature Pyramid Network (FPN).

Структура FPN включає шлях зверху вниз і горизонтальні з'єднання, які додаються до кожного рівня FPN. FPN дозволяє злити інформативні деталі зображень з низькою роздільною здатністю з менш інформативними деталями зображень з високою роздільною здатністю. Кожен рівень FPN з'єднаний з повністю згортковими мережами, які включають дві незалежні підмережі: одну для класифікації та іншу для регресії. Класифікаційна мережа використовується для виконання багатокласового прогнозування, тоді як регресійна мережа розроблена для прогнозування відповідних обмежувальних рамок для класифікованих об'єктів.

Головною новацією RetinaNet є використання фокусних втрат (Focal Loss) у підмережі класифікації. Оскільки виявлення цілей у одноетапних алгоритмах зазвичай стикається з дисбалансом між кількістю позитивних і негативних прикладів, це може впливати на навчання моделі. Фокусні втрати призначають різну вагу важким прикладам, що ефективно вирішує проблему дисбалансу класів у моделі виявлення.[26-27]

б) SSD.

Для збільшення швидкості опрацювання зображення модель SSD усуває використання мережі пропозиції областей, використовуючи при цьому багатомасштабні ознаки та області за замовчуванням.

Роботу SSD можна розбити на три основні етапи. Першим етапом є виділення важливих карт ознак при витягуванні ознак. Цей етап складається тільки з повністю згорткових шарів. Після вилучення ознак, наступним етапом є ідентифікування та локалізація об'єктів, використовуючи при цьому повністю згорткову нейромережу. Основною метою другого етапу є створення найбільш відповідних карт меж для всіх карт ознак. Карта меж надає інформацію про об'єкт у вигляді обмежувальних рамок.

Після проходження двох етапів, останній відповідає за фільтрацію зайвих рамок. Остаточні результати виявлення генеруються за допомогою алгоритму немаксимального пригнічення (NMS), який виявляє дублікати рамок та зберігає

тільки найбільш впевнені прогнози. NMS дозволяє досягти як швидкості, так і точності виявлення.[26-27]

в) YOLO.

Архітектура YOLO заснована на трьох ключових методах, що дозволяють ефективно виявляти об'єкти та бути однією з найбільш поширених моделей серед програмних забезпечень. Перший елемент, який використовує модель YOLO – це залишкові блоки. Кожна клітина цієї сітки служить в якості центральних точок, для яких робиться окремий прогноз.

Другий метод включає врахування кожної з центральних точок під час створення обмежувальних рамок для конкретного прогнозу. Хоча класифікація відбувається ефективно для кожної клітини сітки, встановлення обмежувальних рамок для кожного прогнозу є більш складним завданням.

Останнім методом є використання метрики IoU, що допомагає обчислити найкращі обмежувальні рамки для виконання задачі виявлення об'єктів.[27]

З результатів досліджень на рисунку 2.15, які проводилися для виявлення фармацевтичних препаратів та на рисунку 2.16 на основі датасету COCO можемо сказати, що кожна з моделей може бути спеціалізована в певній сфері та ситуації. RetinaNet пропонує високі показники mAP, проте також потребує більше часу на опрацювання зображення. Це може бути корисним, у випадку коли нам не потрібний високий FPS, проте ми хочемо забезпечити високу точність виявлення.

SSD є середнім варіантом між YOLO та RetinaNet, маючи середній показник по точності та по швидкості опрацювання. Для того щоб створити програму для розпізнавання в режимі реального часу, яка зможе запускатися на слабкому залізі або мобільних пристроях, було розроблено MobileNetSSD. Це модель SSD яка використовує архітектуру MobileNet як основу. Це забезпечує можливість підтримки відносно високого FPS навіть на слабкому апараті, проте mAP такої моделі буде набагато нижчим в порівнянні зі звичайним SSD, або іншими моделями.

YOLO має найбільші показники швидкості, хоч і поступається іншим моделям у точності. Це досягається шляхом модифікування архітектури для підвищення швидкості опрацювання зображення. Хоч модель YOLO і з'явилася досить недавно,

завдяки широкому використанню та великій спільці розробників, оновлення моделей з різними змінами виходять досить часто. YOLO має ширшу вибірку моделей для кожної версії, що дозволяє використання саме цієї архітектури для різного типу обчислювальних машин, вибравши для себе пріоритетом або найбільшу швидкість або найбільшу точність.

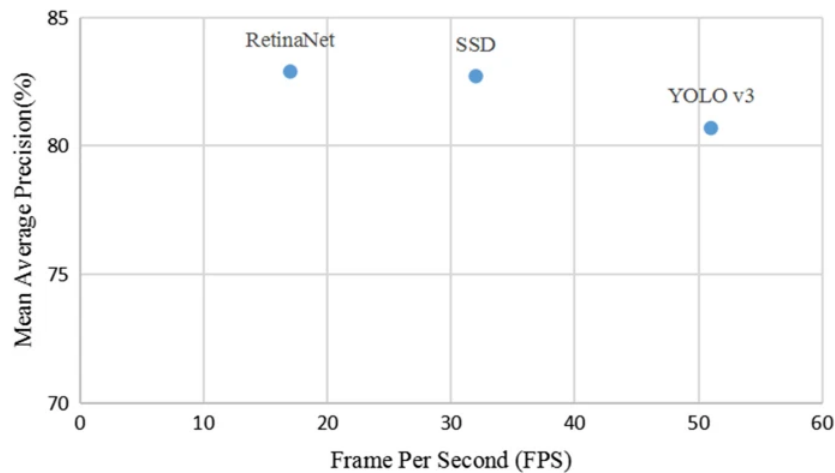


Рисунок 2.15 – Порівняння одноетапних моделей за FPS та mAP[26]

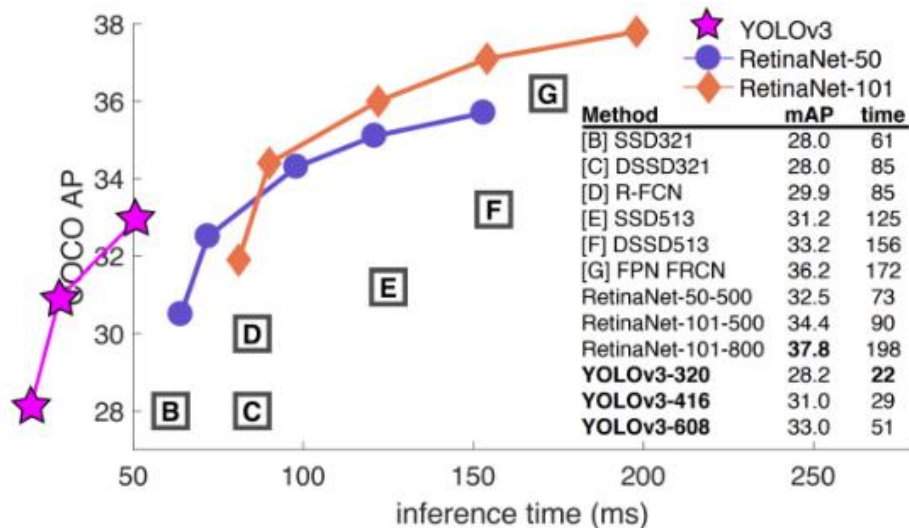


Рисунок 2.16 – Порівняння моделей за часом опрацювання та mAP[28]

Проаналізувавши різновиди одноетапних моделей, було обрано модель YOLO для розробки програмного забезпечення розпізнавання в режимі реального часу. Висока швидкість опрацювання для роботи в реальному часі, можливість підібрати модель в залежності від потужності комп'ютера та простота налаштування та

тренування завдяки великому хабу розробників стали основними факторами для вибору YOLO.

2.4 Висновок до розділу 2

В даному розділі було проведено дослідження використання нейромереж для задачі розпізнавання зображення. Розглянуто згорткову нейронну мережу як тип нейромережі, спеціально розробленої для розпізнавання зображень. Детально розглянуто кожний шар згорткової нейронної мережі, а саме шар згортки, шар активації та види функцій активацій, їх переваги та недоліки. За результатами досліджень виявилось, що Leaky ReLU є найкращим варіантом для більшості випадків. Також було досліджено шар агрегації та способи агрегації.

Досліджено найпопулярніші оптимізатори, які використовуються для навчання нейронної мережі. Розглянуто переваги та недоліки цих оптимізаторів. За результатами досліджень виявилось, що оптимізатор Adam є найкращим оптимізатором у більшості випадків.

Для того щоб обрати модель розпізнавання, досліджено різновиди метрик, які використовуються для дослідження моделі. Досліджено різновиди моделей виявлення, які поділяються на одноетапні та двоетапні. В результаті досліджень було обрано одноетапний алгоритм для використання в системі розпізнавання в режимі реального часу через його швидкість.

Розібрано алгоритм роботи популярних одноетапних моделей. За результатом досліджень було обрано модель YOLO як найшвидшу модель для визначення, що також поєднує в собі високу точність розпізнавання.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОЇ ЧАСТИНИ

3.1 Задачі системи

Система для розпізнавання та ідентифікування транспорту в режимі реального часу складається з комп'ютера, який проводить обчислення, та з підключених до нього камер. При збільшенні кількості камер потрібно також підвищувати потужність обчислювальної машини.

Система опрацьовує кадри з кожної камери. У випадку виявлення на кадрі номерного знаку, системою буде відслідковуватися номер, причому для розпізнавання номеру буде обрано найкращий кадр з отриманих. Паралельно з цим відслідковується і транспорт, який буде класифіковано.

Після того як номерний знак зник з поля зору, найкращий отриманий кадр знаку відправляється на процедуру оптичного розпізнавання символів (Optical Character Recognition, OCR), результатом якого є розпізнавання номеру.

Результатом роботи системи є JSON файл, який зберігає в собі інформацію про час отримання кадру транспортного засобу, його тип та номер, а також конвертоване в масив зображення. Даний JSON файл може використовуватися при майбутніх операціях з отриманою інформацією.

3.2 Модель виявлення

3.2.1 Вибір моделі YOLO

Сімейство YOLO налічує дуже багато моделей, які були випущені в різні роки (рис. 3.1). Оскільки YOLO має відкрите джерело, різні розробники доповнювали алгоритми та використовували різні основи для побудови моделі.



Рисунок 3.1 – Історія розвитку YOLO[29]

Кожна версія мала свої моделі різного розміру для застосування в залежності від потужності обчислювальної машини. Для вибору моделі розглянемо порівняння потужності та швидкості останніх версій YOLO на рисунку 3.2.

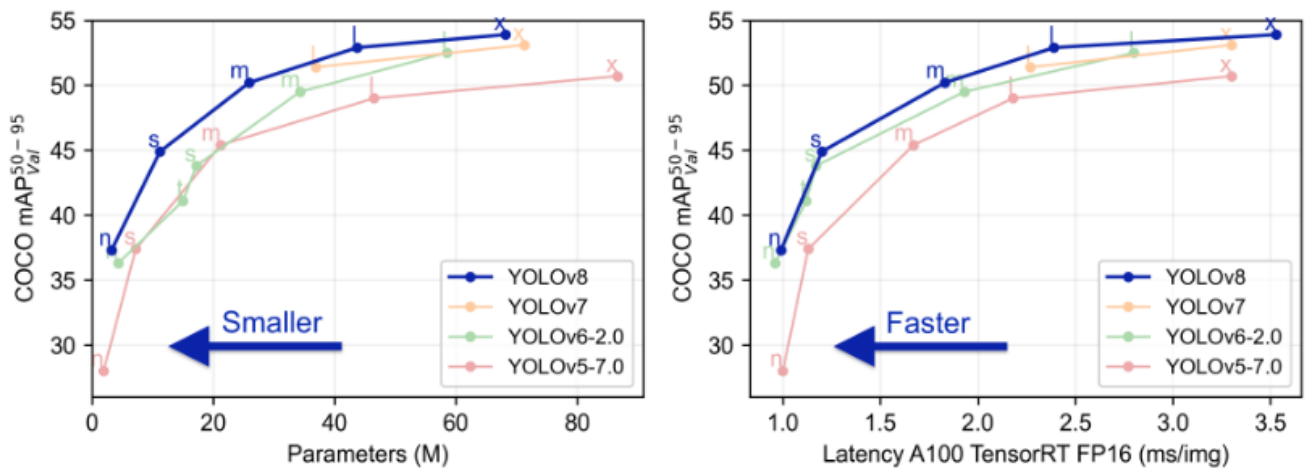


Рисунок 3.2 – Порівняння останніх моделей YOLO[29]

З порівняння моделей можемо сказати, що модель YOLOv5 є найменшою за розмірами і найнижчою за точністю та швидкістю серед інших моделей. Натомість модель YOLOv8 показує найкращі або подібні результати по відношенню до інших моделей. Тому для розробки системи розпізнавання оберемо модель YOLOv8.

Розглянемо варіанти моделі YOLOv8. Всього розрізняється 5 моделей YOLO: Nano, Small, Medium, Large, Extra Large, які позначаються як на рисунку 3.3.

Model	size (pixels)	mAP ^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)
YOLOv8n	640	37.3	80.4	0.99	3.2
YOLOv8s	640	44.9	128.4	1.20	11.2
YOLOv8m	640	50.2	234.7	1.83	25.9
YOLOv8l	640	52.9	375.2	2.39	43.7
YOLOv8x	640	53.9	479.1	3.53	68.2

Рисунок 3.3 – Порівняння моделей YOLOv8[29]

За результатами можемо сказати, що для кожної з моделей може мати свою сферу застосування. Також наявність GPU в обчислювальній машині може суттєво збільшити обчислювальну потужність моделі. За рахунок того що GPU здатна проводити паралельні обчислення, має більшу пам'ять та спеціалізована під роботу з векторами та матрицями обчислення параметрів мережі відбувається набагато швидше.

YOLOv8n має найменшу точність серед всіх інших моделей. Проте через її розмір та кількість параметрів вона може швидше навчитися на слабкому комп'ютері. Дана модель може використовуватися в мобільних пристроях або слабких системах, при цьому підтримуючи максимально можливий в порівнянні з іншими моделями FPS.

YOLOv8s має вже набагато більшу точність, проте і її розміри вже набагато більші. Це означає що моделі потрібно вже більше часу на опрацювання кадру, особливо коли опрацювання йде на CPU. Дана модель запропоновано використовувати у комбінації або з потужним процесором, або з слабкою GPU.

YOLOv8m має ще більшу точність, проте за часом опрацювання кадру вона набагато сильніше відстає від легших версій. Цю модель бажано використовувати у випадках, коли потрібно ще більше підвищити точність для розпізнавання більшої

кількості деталей та об'єктів на кадрі. Для підтримки її роботи у режимі реального часу вже бажано використовувати середню за потужністю GPU.

YOLOv8l та YOLOv8x використовуються, коли потрібна найвища точність розпізнавання, особливо коли об'єктів та деталей дуже багато на кадрі і вони можуть бути невеликими. Для підтримки роботи у реальному часі вже потрібно використовувати потужну GPU.

В нашому проєкті розпізнавання має бути достатньо точним, враховуючи що деталі на зображенні будуть не малі. Також планується, що система для розпізнавання транспортних засобів буде використовувати GPU з невеликим обсягом відеопам'яті, тому вирішено обрати модель YOLO small, оскільки вона підпадає під ці вимоги.

3.2.2 Архітектура моделі

Основна архітектура YOLOv8 складається з трьох складових: хребет(backbone), шия(neck) та голови(head). На рисунку 3.4 показано архітектуру мережі, причому тут голова та шия показані як один модуль. Можемо побачити, що архітектура належить до типу одноетапного виявлення.

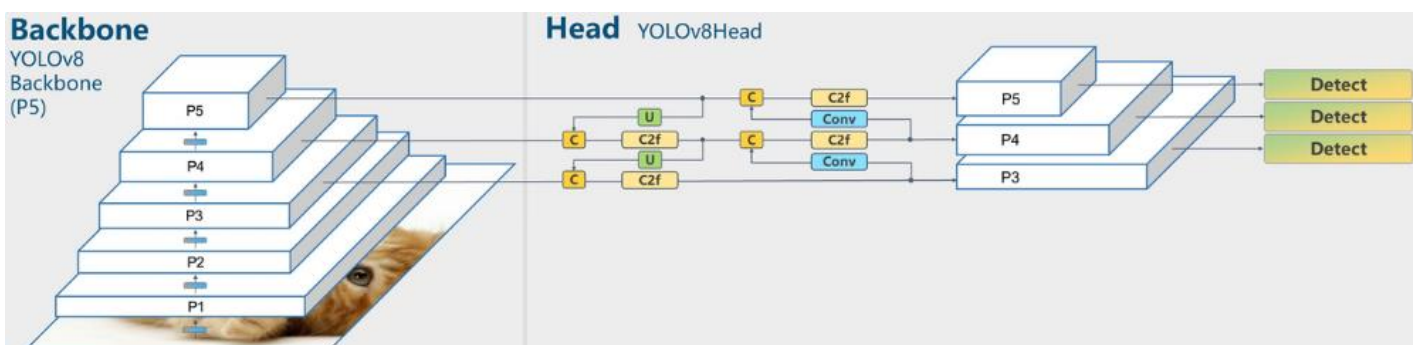


Рисунок 3.4 – Архітектура мережі YOLOv8 [30]

Однією з особливостей моделі YOLOv8 над іншими моделями сімейства є те, що модель виконує визначення без якоря. Це означає, що вона пророкує

безпосередньо центр об'єкта, а не зміщення від відомої опорної точки. Якорями називається заздалегідь визначений набір обмежувальних рамок з певними масштабами, які використовуються для виявлення класів об'єктів з таким же співвідношенням масштабів. Ці якорі вибираються на основі розміру об'єктів у навчальних даних та накладаються на зображення під час виявлення. В процесі навчання кожній області в навчальних даних зіставлялася якірна область, яка мала найвищий показник IoU. Потім модель навчилася прогнозувати зміщення і зміну масштабу від цього опорного контуру до базового контуру замість того, щоб безпосередньо прогнозувати опорний контур.

Перевагою безякірного виявлення є те що воно більш гнучке і ефективне. Якірне виявлення передбачає ручне визначення якірних рамок, які можуть працювати погано, якщо якорі не відповідають розподілу форм об'єктів у даних. Виявлення об'єктів без використання якорів допомагає скоротити кількість пропозицій для виявлення, що полегшує виконання NMS.[31]

Одноетапна модель YOLO складається з наступних компонентів:

а) Модуль вилучення ознак.

Хребет, або магістраль – це модуль, який використовується для вилучення ознак. Він відповідає за створення набору високорівневих ознак в різних масштабах та при різних роздільних здатностях, які потім використовуються рештою мережі. Цей підхід схожий до розглянутого в RetinaNet підходу FPN.

Для цього зображення пропускається через декілька згорткових шарів для отримання карт ознак. Шари організовані в ієрархічному порядку, де кожен шар фіксує особливості на різних рівнях. Ці шари виконують базові операції, такі як згортка, пакетна нормалізація та функція активації. В моделі ці шари об'єднані в етап Conv. В даній моделі як функція активації використовується SiLU, яка записується наступним чином[32]:

$$f(x) = x\sigma(x) \quad (3.1)$$

де σ – сигмоїда.

Порівняємо графіки функцій SiLU та ReLU на рисунку 3.5.

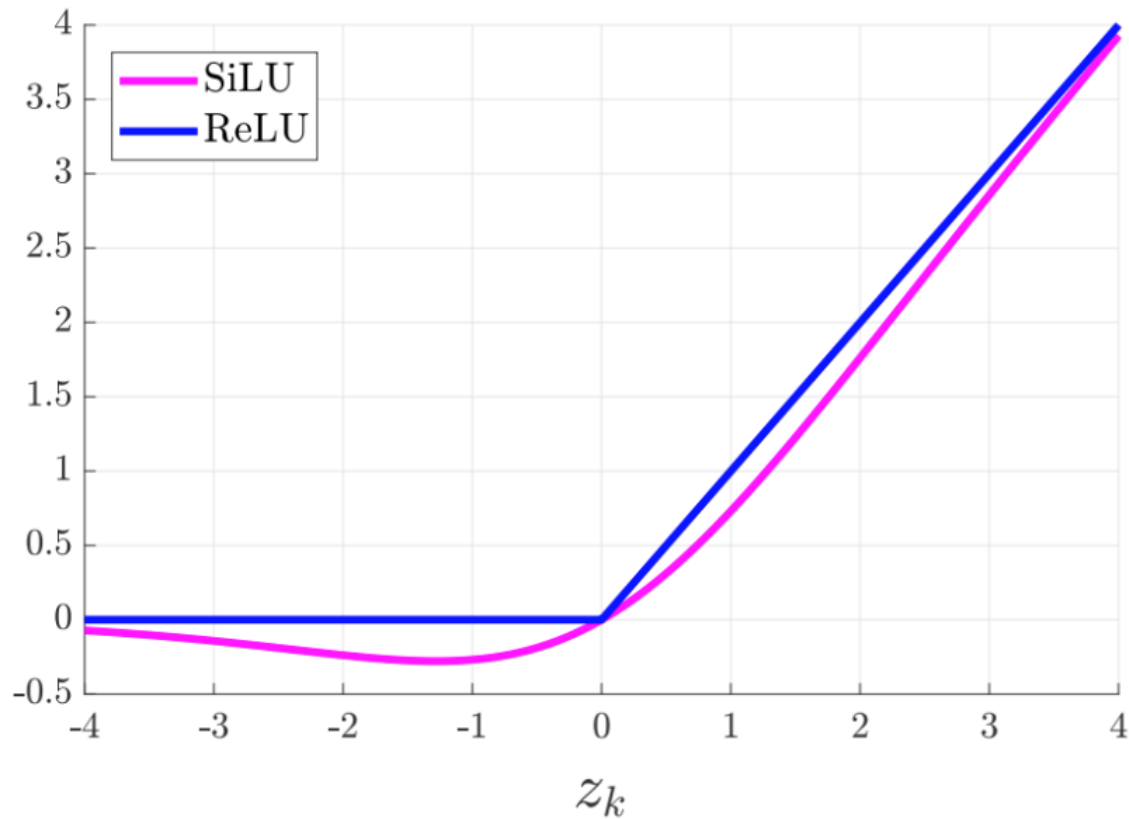


Рисунок 3.5 – Функція активації SiLU по відношенню до ReLU[32]

Можемо побачити, що перевагою SiLU над ReLU є те, що вона більш плавна, через що може вловлювати більше нелінійних ознак. Також функція SiLU позбавлена проблеми вмираючого ReLU, оскільки як і Leaky ReLU, виконує активацію на від'ємному діапазоні.

Основою для хребта є модифікація моделі CSPDarknet53, архітектура якого зображена на рисунку 3.6. Особливістю модифікації цієї мережі є те що вона використовує модуль C2f. Задача цього модулю заключається в тому що він розділяє вхідні карти ознак на дві частини, одна з яких переходить до наступного блоку згорткових шарів, тоді як друга частина проходить через декілька згорткових шарів.[31]

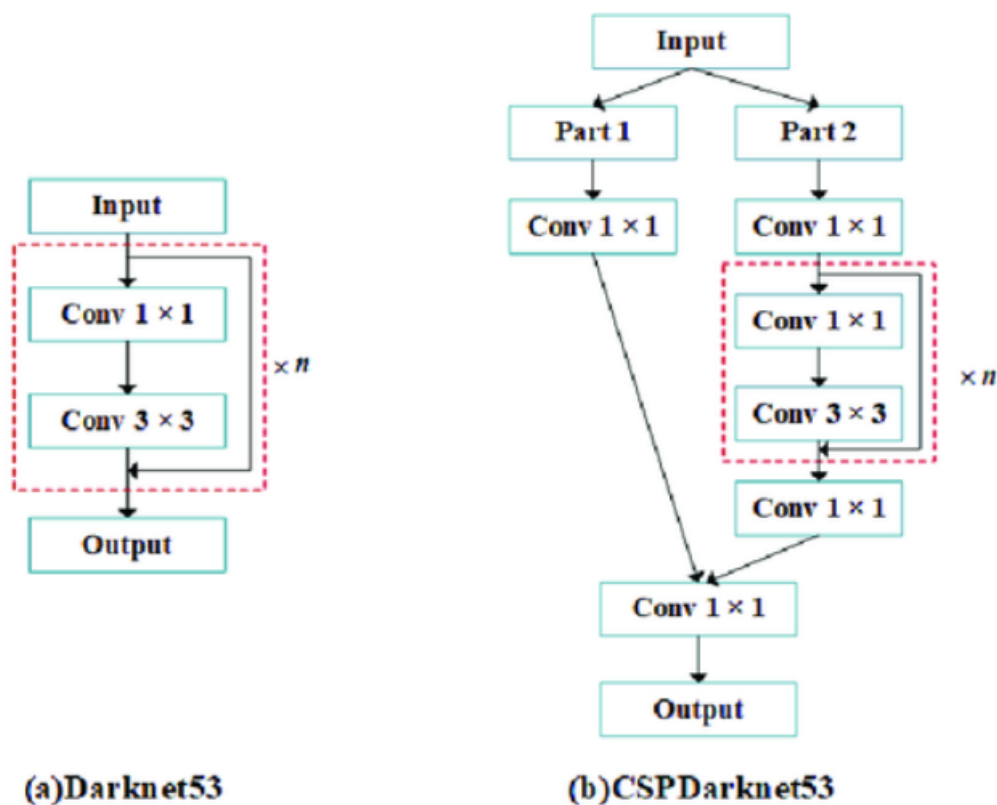


Рисунок 3.6 – Архітектура CSPDarknet53[33]

C2F-модулі в YOLOv8 використовують частково поперечний підхід (Cross Stage Partial, CSP), натхненний DenseNet, з наступних причин[34]:

- Пом'якшення проблеми зникаючого градієнта, оскільки коли мережі стають глибшими, сигналу помилки з виходу стає все важче поширюватися назад до ранніх шарів під час зворотного поширення. CSP допомагає вирішити цю проблему, посилюючи потік градієнтів через мережу.
- Для полегшення розповсюдження особливостей, оскільки архітектура CSP забезпечує кращий зв'язок між шарами в мережі, що дозволяє більш ефективно поширювати ознаки від ранніх шарів до пізніх.
- Заохочення повторного використання функцій, оскільки з'єднуючи шари безпосередньо і дозволяючи інформації обмінюватися між ними, CSP заохочує мережу використовувати і повторно використовувати функції на різних рівнях, підвищуючи ефективність використання функцій.

- Зменшити кількість параметрів: Дизайн CSP за своєю суттю є параметрично-ефективним, оскільки він забезпечує прямі зв'язки між рівнями, що допомагає зменшити кількість необхідних параметрів у мережі, що призводить до створення легшої та ефективнішої моделі.

Таким чином, використання C2F (рис. 3.7) як модуля CSP в основі YOLOv8 підвищує ефективність моделі, результативність і загальну продуктивність виявлення об'єктів.

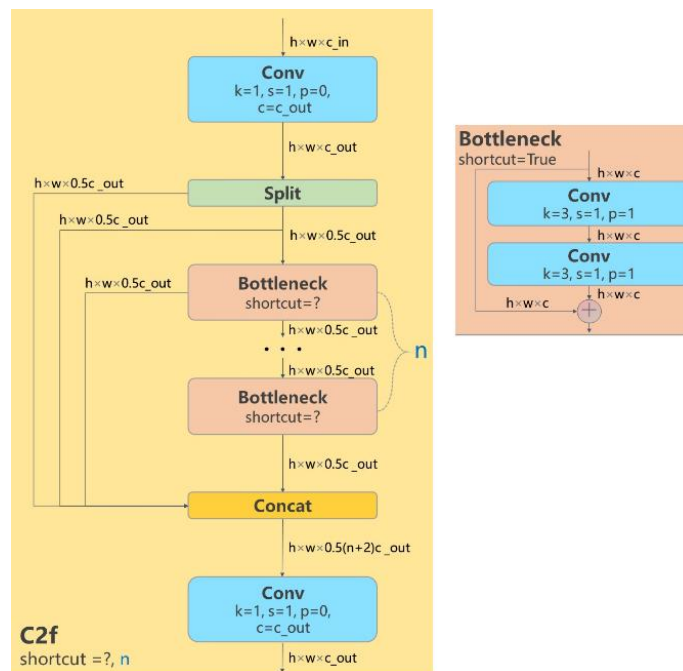


Рисунок 3.7 – Структура модулю C2f в YOLOv8[30]

Число n вказує на кількість повторень модулю Bottleneck в межах кожного рівня CSP. Це число може змінюватися в залежності від конкретного рівня в архітектурі та в залежності від типу моделі YOLOv8.

Модуль Bottleneck допомагає зменшити розмірність карт ознак, при цьому зменшивши обчислювальну складність мережі. Цей модуль також сприяє навчанню більш високорівневих ознак. Результатом роботи модулю C2f є агрегування карт ознак об'єктів з різних масштабів. Для виявлення об'єктів різних розмірів нам потрібні всі ознаки. Модуль бере об'єкти з високою роздільною здатністю з попереднього етапу і семантично багаті об'єкти з низькою роздільною здатністю з більш пізнього етапу та об'єднує їх вздовж виміру глибини. Таким чином

створюється нова карта об'єктів, яка містить як детальну просторову інформацію, так і глибоку семантичну інформацію.[33]

Заключним етапом магістралі є модуль швидкого об'єднання просторових пірамід SPPF (рис. 3.8).

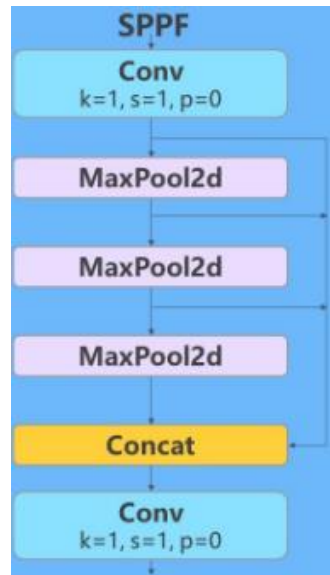


Рисунок 3.8 – Структура модулю SPPF[29]

Основною задачею модуля є те, що він агрегує вхідні дані і генерує вихід фіксованого розміру. Його перевага полягає в здатності значно розширити прийнятне поле і виділити найважливіші контекстові характеристики, при цьому не знижуючи продуктивність мережі [33].

б) Модуль агрегації.

Після проходження основної магістралі, перед переходом до голови моделі, отримані ознаки проходять через ший. Мета ший полягає в змішуванні та комбінуванні карт ознак, створених основною частиною моделі для підготовки їх для головної частини. Для цього ший також використовує модулі C2f, Conv, а також шари об'єднання та підвищення роздільної здатності зображення [34].

в) Модуль виявлення.

Голова отримує ознаки з ший і виконує остаточні завдання виявлення об'єктів. Голова видає три виходи, оскільки відбувається прогнозування в різних масштабах. Перший результат відповідає за виявлення більших об'єктів. Цей результат отримується з більш грубих карт з нижчою роздільною здатністю, але вищим рівнем

об'єктів. Другий результат відповідає за виявлення об'єктів середнього розміру, і цей результат отримується з карт об'єктів, які мають середню деталізацію. Третій результат відповідає за виявлення менших об'єктів. Він отримується з більш дрібних карт з вищою роздільною здатністю, але нижчим рівнем об'єктів.

Модуль визначення в голові застосовує модулі згортки та лінійних шарів (рис. 3.9). Карти ознак пропускаються через серію згорткових і лінійних шарів, за результатами яких визначаються функції втрати для обмежувальних рамок та класу об'єкта.

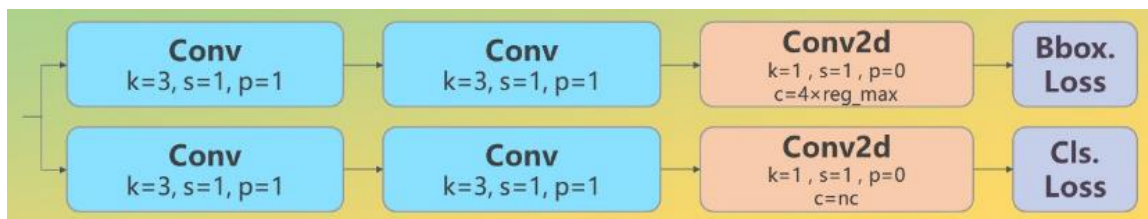


Рисунок 3.9 – Модуль виявлення[29]

Для визначення втрат обмежувальних рамок використовується повна втрата (CIoU) та розподільна фокусна втрата (DFL).

Втрати CIoU покращують прогнозування обмежувальних рамок при виявленні об'єктів, враховуючи не тільки перекриття між прогнозованою та основною областями, як втрата IoU, але й відстань між центрами та співвідношення сторін цих областей. Це робить прогнози моделі більш точними та відповідними за розміром. Також втрати CIoU збігаються швидше в порівнянні з іншими IoU втратами.[35]

DFL є вдосконаленим показник фокусних втрат RetinaNet, яка використовується для усунення дисбалансу і неузгодженості між завданнями класифікації та локалізації при виявленні об'єктів. Основна ідея DFL полягає в тому, щоб під час навчання більше фокусуватися на складних прикладах і менше на простих. Однак замість того, щоб зосереджуватися на прогнозованих ймовірностях класів, як фокусні втрати, DFL фокусується на прогнозованому IoU, що робить завдання локалізації і завдання класифікації більш узгодженими.[36]

Для отримання значення втрат для класифікації, в YOLOv8 використовується бінарна перехресна ентропія (BCE). BCE використовуються в задачах, де завдання

полягає в тому, щоб передбачити, чи належить екземпляр до певного класу чи ні. Втрати обчислюються на основі різниці між істинною міткою та передбаченою ймовірністю для цієї мітки. Якщо прогноз є впевненим і правильним, втрати є низькими; якщо прогноз є невпевненим і неправильним, втрати є високими. Якщо прогноз невпевнений, то втрати знаходяться десь посередині між ними. У випадку багатокласової класифікації, підрахунок втрат розглядається як кілька підзадач ВСЕ.[37]

3.2.3 Навчання моделі

Для навчання моделі YOLOv8 потрібне створення датасету. Для цього ми можемо використати сайт Roboflow [38], оскільки на даному сайті наявні усі інструменти, які потрібні для створення датасету.

На початку складається колекція зображень, які будуть використовуватися в датасеті. Після цього, використовуючи інструменти Roboflow, анотуємо кожне зображення, як зображено на рисунку 3.10.



Рисунок 3.10 – Анотація зображення

Контурами, кожному з яких відповідає його клас, виділяються об'єкти на зображенні. В даному випадку, помаранчевий контур відповідає класу “car”, в той час як червоний контур відповідає класу “plate”.

Оскільки Roboflow це відкрита платформа, на ній доступні створені раніше датасети, якими можна доповнити власноруч створений для збільшення кількості прикладів для моделі.

Для YOLOv8 дані представлені у наступному форматі (рис. 3.11): індекс класу, місцеположення центру об'єкту, ширина та висота об'єкту.

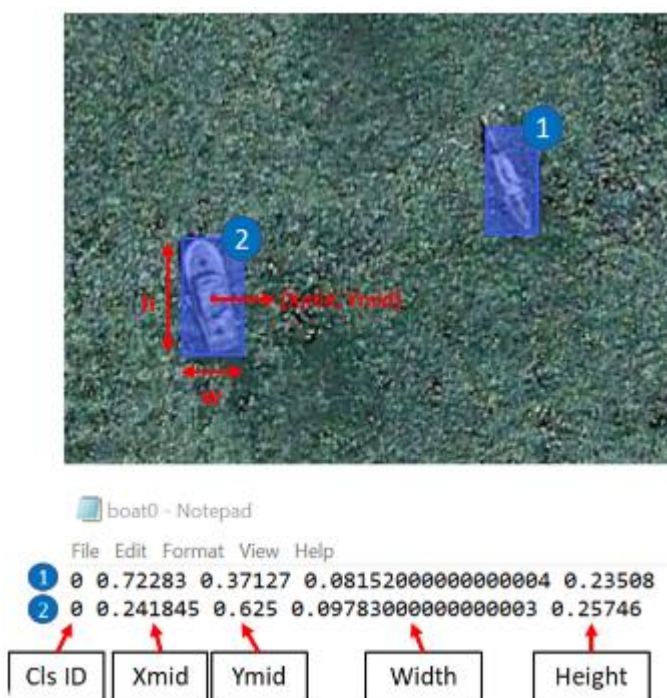


Рисунок 3.11 – Приклад інформаційного файлу анотованого зображення[39]

При генерації датасету, який буде готовий до використання, можна скористатися додатковими інструментами Roboflow. Зображення в датасеті можна піддати додатковій обробці за допомогою фільтрів. Таким чином, можна створити більше зображень для навчання, причому додаткова частина буде складатися з зображень з шумами, розмиттям, зміною положення. Таким чином, при навчанні створюється модель, яка більш пристосована до нештатних ситуацій та до різного роду шумів.

Після того, як було згенеровано датасет, його можна використовувати для абсолютно різних моделей навчання. Оскільки нас цікавить YOLOv8, ми імпортуємо його у відповідному форматі.

На рисунку 3.12 зображено розподіл зображень по класам. Ділення датасету на тренувальну, валідаційну та тестові вибірки відбулося у співвідношенні 70-20-10 відповідно.

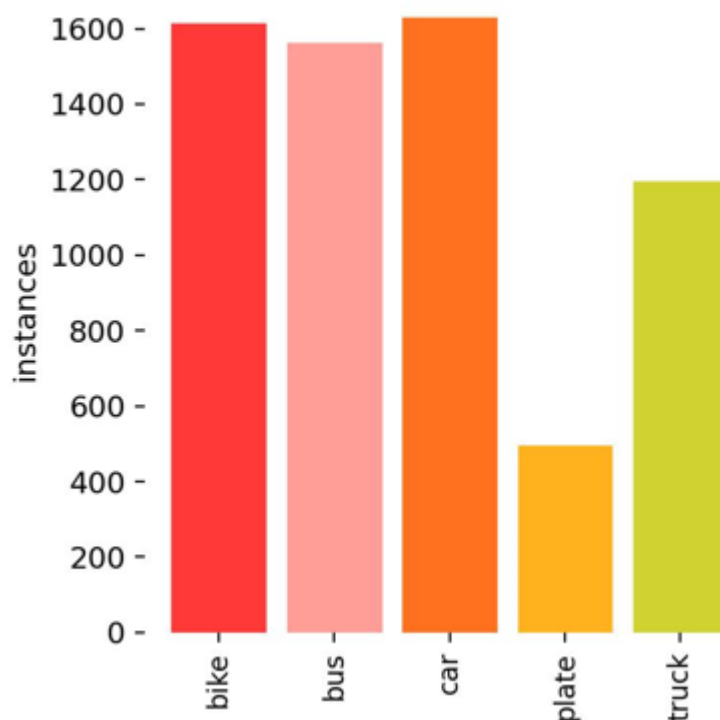


Рисунок 3.12 – Розподіл зображень у вибірці за класами

Як було розглянуто вище, для нашої роботи використаємо модель YOLOv8s, оскільки ми не знаємо наскільки потужні обчислювальні комп'ютери будуть використовуватися для роботи.

Для навчання задамо наступні параметри:

- mode = train;
- task = detect;
- epochs = 40;
- batch = -1;
- imgsz = 640;
- optimizer = Adam;

- lr0 = 0.0001;
- momentum = 0.9;
- model = yolov8s.pt.

Розглянемо задані параметри. Mode визначає режим в якому працює Yolo. Це може бути тренування, валідація, передбачення.

Task відповідає за тип задачі моделі YOLO, серед яких може бути виявлення, сегментація, класифікація. Оскільки ми хочемо виявляти об'єкти в режимі реального часу, обираємо опцію detect.

EPOCHS визначає кількість епох тренування моделі. При більшій кількості епох модель може краще навчитися.

Batch відповідає за кількість зображень у пакеті, який подається в модель для прогону. Більша кількість зображень в пакеті сприяє швидшому навчанню моделі, проте вимагає більше відеопам'яті. Параметр -1 дозволяє системі підібрати кількість зображень в пакеті в залежності від можливостей комп'ютера.

ImgSz визначає розмір вхідного зображення в модель. Більший розмір сприяє точнішому розпізнаванню, проте тренування буде довшим і ресурсоємнішим.

Optimizer дозволяє обрати оптимізатор для моделі. Серед можливих опцій є SGD, Adam, RMSProp. З розглянутих в попередньому розділі оптимізаторів обираємо Adam.

Замість того щоб навчати модель з нуля, можна використовувати раніше натреновану модель на датасеті COCO. Це пришвидшить та спростить процес навчання, оскільки будуть задані первинні параметри вагів.

Lr0 це швидкість навчання моделі, яка для оптимізатора Adam рекомендована як 0.0001. Momentum в YOLO представляється як параметр beta1, який рекомендовано виставляти на рівні 0.9.[40]

Після навчання моделі отримаємо характеристики моделі, зображені на рисунку 3.13.

	from	n	params	module	arguments
0	-1	1	928	ultralytics.nn.modules.conv.Conv	[3, 32, 3, 2]
1	-1	1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
2	-1	1	29056	ultralytics.nn.modules.block.C2f	[64, 64, 1, True]
3	-1	1	73984	ultralytics.nn.modules.conv.Conv	[64, 128, 3, 2]
4	-1	2	197632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
5	-1	1	295424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
6	-1	2	788480	ultralytics.nn.modules.block.C2f	[256, 256, 2, True]
7	-1	1	1180672	ultralytics.nn.modules.conv.Conv	[256, 512, 3, 2]
8	-1	1	1838080	ultralytics.nn.modules.block.C2f	[512, 512, 1, True]
9	-1	1	656896	ultralytics.nn.modules.block.SPPF	[512, 512, 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	591360	ultralytics.nn.modules.block.C2f	[768, 256, 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	148224	ultralytics.nn.modules.block.C2f	[384, 128, 1]
16	-1	1	147712	ultralytics.nn.modules.conv.Conv	[128, 128, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
18	-1	1	493056	ultralytics.nn.modules.block.C2f	[384, 256, 1]
19	-1	1	590336	ultralytics.nn.modules.conv.Conv	[256, 256, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]
21	-1	1	1969152	ultralytics.nn.modules.block.C2f	[768, 512, 1]
22	[15, 18, 21]	1	2117983	ultralytics.nn.modules.head.Detect	[5, [128, 256, 512]]
Model summary: 225 layers, 11137535 parameters, 11137519 gradients					
Model summary (fused): 168 layers, 11127519 parameters, 0 gradients					

Рисунок 3.13 – Структура моделі YOLOv8s до навчання та після

За результатами навчання отримали графіки метрик для перевірки якості навченої моделі. Розглянемо отримані результати.

На рисунку 3.14 зображені функції втрат, метрики точності та повноти, а також метрики mAP для значення IoU 50 та для ряду значень IoU 50-95.

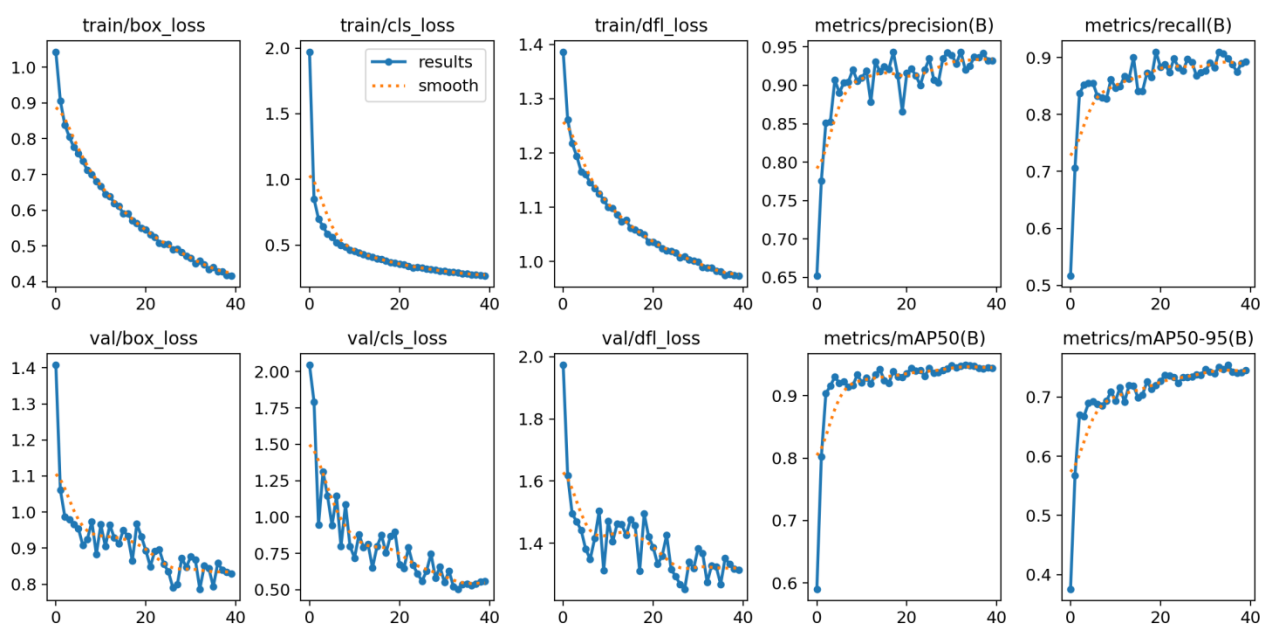


Рисунок 3.14 – Результати функцій втрат

На графіках функцій втрат зображені втрати при тренуванні та втрати при валідації для кожного типу втрат. Можемо побачити, що в середньому втрати при навчанні практично в два рази менші за отримані втрати при валідації. Це може свідчити про те що модель має надмірне заучування навчальних даних, проте показники втрат при валідації також мають чудові показники, що може свідчити про гарну роботу моделі.

Результати повноти та точності мають високі показники. Що може свідчити про те що модель точно розпізнає об'єкти на зображенні. Показники mAP також вказують на те, що об'єкти виділяються та класифікуються з високою точністю. Хоч при збільшенні порогу IoU значення точності падає, але точність все рівно висока.

Для того щоб дізнатися, чи можна використовувати дану модель, перевіримо її на тестовому датасеті для того щоб визначити, наскільки вона придатна до розпізнавання нових даних.

Першою з метрик перевірки виступає нормалізована матриця плутанини, зображеної на рисунку 3.15. На ній показано, наскільки точно модель змогла визначити клас, до якого належить об'єкт на зображенні.

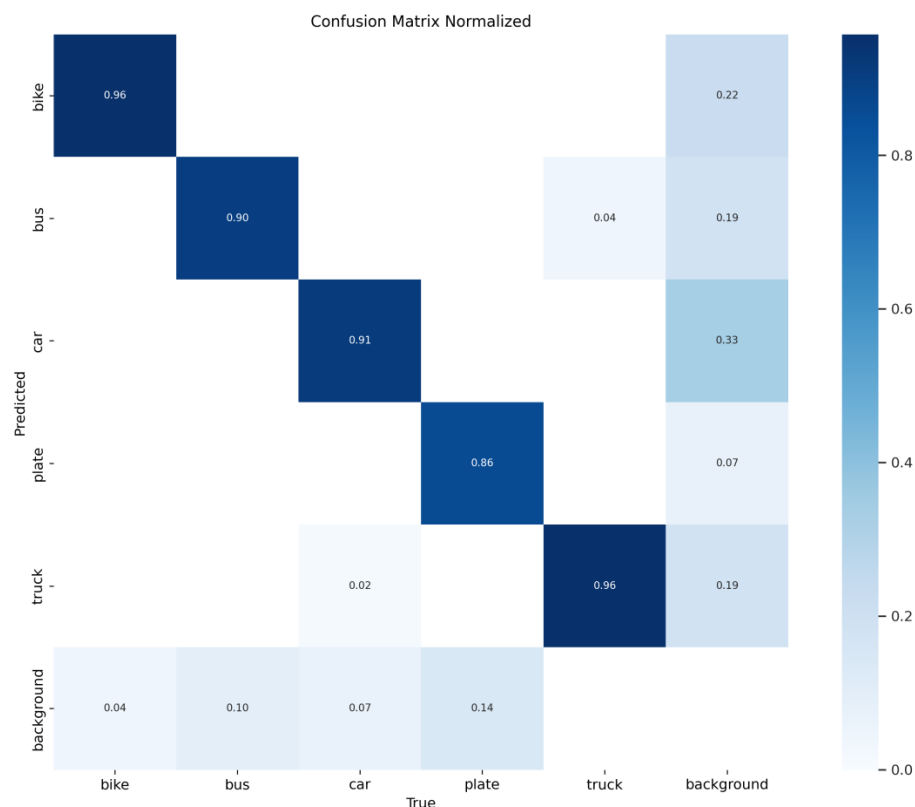


Рисунок 3.15 – Нормалізована матриця плутанини

За результатами бачимо, що модель чудово справляється з виявленням всіх класів техніки, проте має трохи нижчу точність виявлення номеру. Це пояснюється тим, що в наборі даних не всі номерні знаки на машинах були розмиті, через що розпізнавання відбувалося коректно, проте воно не зараховувалося через маркування. Показники для виявлення автобусів та машини трохи нижчі, проте все ще дуже добрі. Для покращення результатів потрібно оновити датасет, оскільки ми бачимо що модель з заданими параметрами показує чудові результати. Найкращий для розпізнавання датасет вдасться зібрати після встановлення камери на дорогах та збирання знімків з потрібного ракурсу. Після цього можна створити команду людей що буде маркувати дані або використовувати дану модель для доповнення навчальних даних.

Наступною метрикою йде PR – крива (рис. 3.16). За її результатами можемо побачити що модель чудово справляється з виявленням та класифікацією транспортних засобів, проте має низькі показники для машин. Це обумовлено наявністю прикладів з машинами, на яких анотовано тільки номер, через що є модель сприймає машину як машину, проте вона не виділена як об'єкт. Дана метрика використовується для розрахунку mAP моделі. При IoU=0.5 бачимо чудові показники точності виявлення на рівні 0.95.

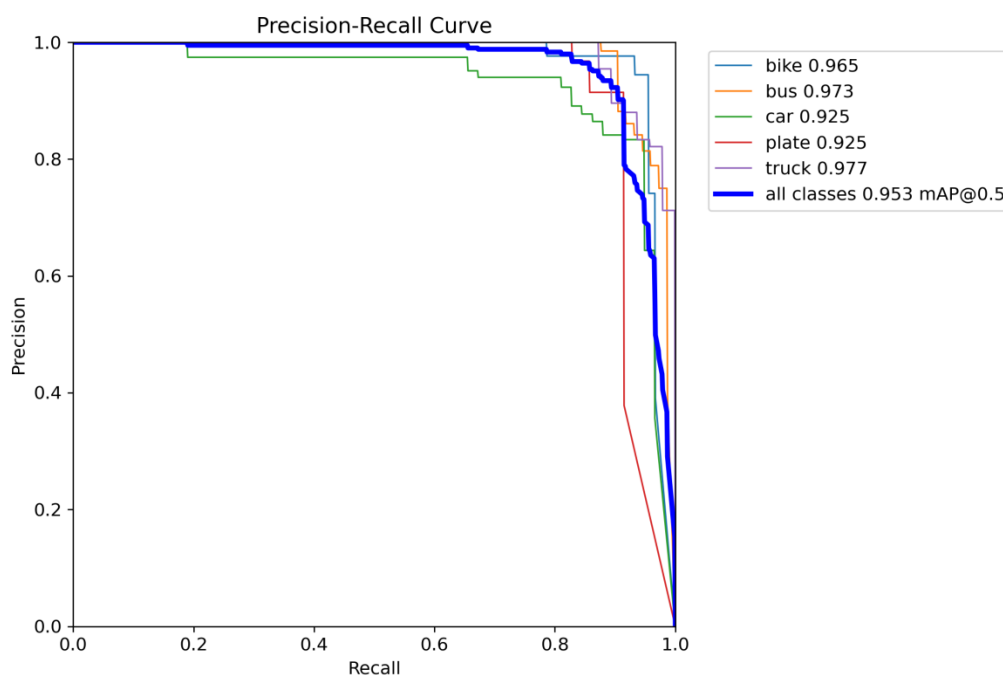


Рисунок 3.16 – PR-крива моделі

Ще однією метрикою для перевірки точності є F1 – крива (рис. 3.17). F1 це оцінка, основною задачею якої є поєднання оцінки точності та повноти в одне число. F1 дає цілісне уявлення про продуктивність моделі. Точність і повнота часто суперечать одне одному, утворюючи компроміс. Модель з високою точністю дуже впевнена у своїх позитивних прогнозах, але може пропустити багато реальних позитивних прикладів. І навпаки, модель з високим рівнем пригадування ідентифікує більшість позитивних прикладів, але може позначити багато негативних прикладів як позитивні. Об'єднуючі ці два параметри, оцінка F1 намагається збалансувати компроміс між цими двома оцінками. Можна сказати, що F1 більш комплексна оцінка точності ніж просто точність. Оцінка F1 розраховується наступним чином[41]:

$$F1 = \frac{2 * P * R}{P + R} \quad (3.2)$$

де P – точність;

R – повнота.

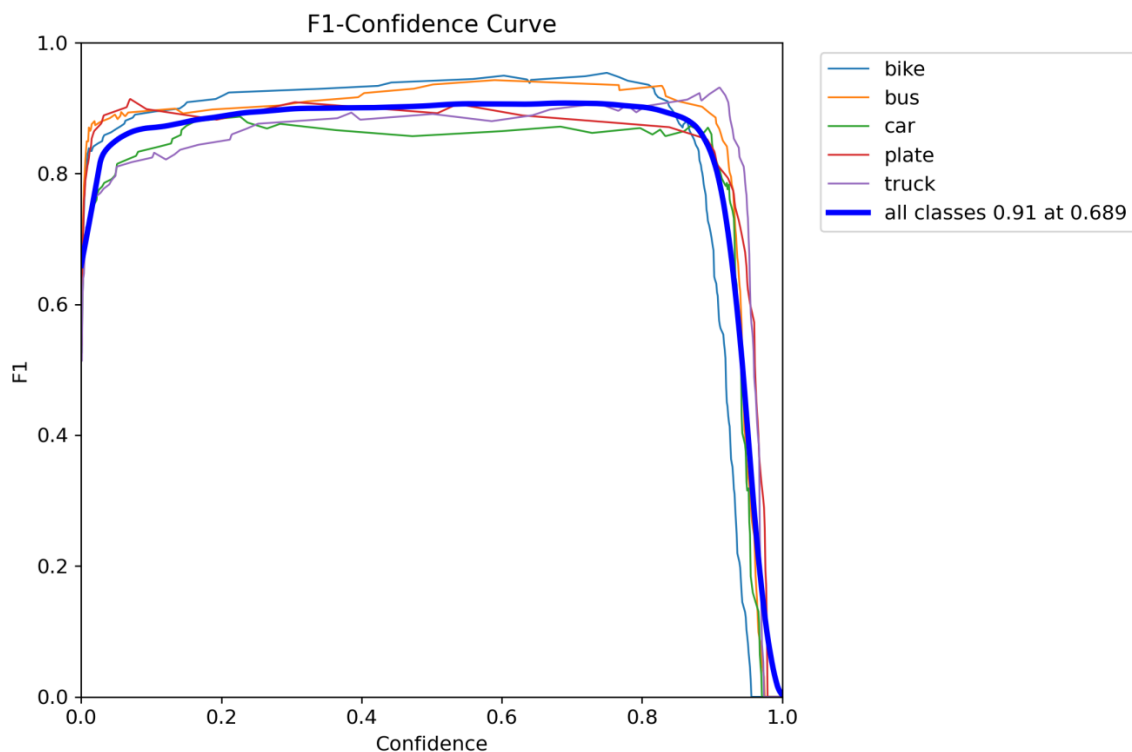


Рисунок 3.17 – Крива F1 моделі

Крива F1 формулюється, обчисливши значення F1 при всіх рівнях впевненості моделі в розпізнаванні. На рисунку 3.17 можемо побачити, що при впевненості моделі на рівні 0.69 маємо точність F1 0.91. Для роботи в реальному часі дані показники є чудовими.

Отже, розглянувши результати перевірки моделі на тестовому наборі. Можемо сказати що модель справляється зі своєю задачею і можна продовжувати розробку системи та впровадження даного розпізнавання.

3.3 Використані інструменти

3.3.1 Мова програмування та середа розробки

Для написання програми було використано мову Python, оскільки вона підтримує багато зручних бібліотек для використання камер та обробки зображень. Також є зручна підтримка моделей виявлення та розпізнавання.

Для навчання мережі було вибрано Google Colab, оскільки він дозволяє використовувати досить потужні сервера для масштабних розрахунків, які проводить модель. Так як в нас повноцінна програма, для її написання використовується PyCharm як одне з найкращих та оптимізованих середовищ програмування.

3.3.2 Інструменти розробки

Для написання програми використовувалися декілька бібліотек та програмних реалізацій.

Для тренування та інтеграції моделі YOLO було використано бібліотеку ultralytics. Ця бібліотека має інструменти для налаштування та тренування мереж, їх

використання, а також інсталювані попередньо треновані моделі, які можуть допомогти при навчанні моделі на новому датасеті.[42]

Для обробки зображень використовувалася бібліотека OpenCV. Вона надає величезний спектр інструментів для прийому вхідного зображення з камер, обробки цього зображення та показ результатів. Також бібліотека не навантажує систему, оскільки працює з використанням NumPy, що робить її швидшою. В програмі використовується для отримання зображення з камери, визначення кращого кадру, обрізання та модифікування кадру, конвертації кадру в числовий масив для зручнішого зберігання та транспортування.

NumPy використовується для зберігання отриманих за допомогою OpenCV кадрів.

EasyOCR це бібліотека, яка надає доступ до використання моделі EasyOCR, яка конкретно спеціалізується на аналізі та розпізнаванні тексту. Цей модель використовується для визначення транспортного номеру з отриманого зображення.[43]

Також було використано бібліотеку PyTorch, яка надала інструменти для підключення відеокарти за допомогою платформи CUDA для виконання обчислювальних операцій моделі. Це кардинально збільшило швидкість роботи моделі розпізнавання.

3.4 Опис роботи системи

Систему можна поділити на такі 4 підмодулі:

- Модуль захоплення кадрів;
- Модуль опрацювання кадру;
- Модуль розпізнавання символів;
- Модуль формування POST-запиту.

Схему роботи програми можна представити наступним чином (рис. 3.18):

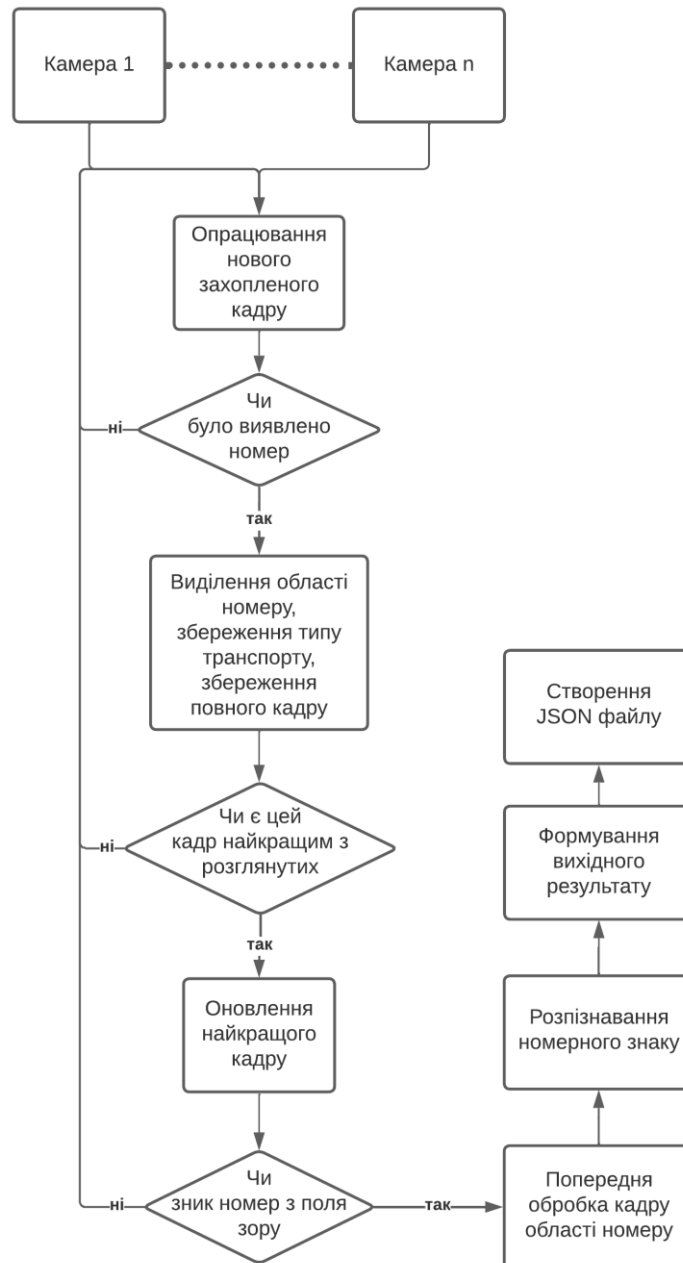


Рисунок 3.18 – Структура програми автоматичного виявлення

3.4.1 Модуль захоплення кадрів

При запуску системи спочатку перевіряється кількість камер, які підключені до комп'ютера. Це потрібно для визначення кількості каналів, з яких буде отримуватися відеопотік кадрів.

Після захоплення кадру, він передається до наступного модулю з позначкою з якої саме камери отримано цей кадр.

В залежності від швидкості з якою буде передано наступний кадр формується показник FPS, що вказує на швидкість роботи системи.

3.4.2 Модуль опрацювання кадру

Даний модуль відповідає за використання навченої моделі YOLO для виявлення на отриманому кадрі номерного знаку та транспортного засобу.

Програма передбачає знаходження камери з лівої сторони від дороги, оскільки на деяких вантажівках та машинах номерний знак знаходиться саме зліва. Також передбачається, що одна камера буде розрахована на одну автомобільну полосу.

Для того щоб спростити процес виявлення номеру транспортного засобу, сектором виявлення стала половина вхідного кадру, як зображено на рисунку 3.19. Таким чином ми швидше зможемо опрацювати ділянку з номером, уникнувши потреби виявлення двох номерів у випадку, якщо попереду ще буде транспортний засіб.

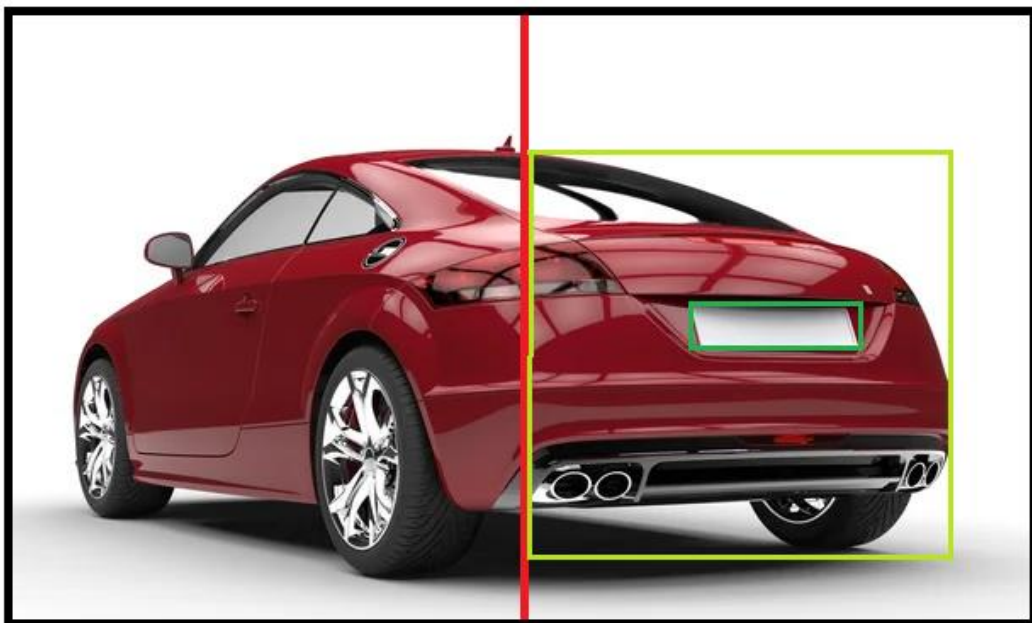


Рисунок 3.19 – Поділ кадру на зони виявлення

Хід роботи модулю визначення можна описати наступним чином:

Перший етап – виявлення номеру в заданому секторі вхідного кадру. У випадку, якщо модель почала виявляти транспортний номер, він піддається перевірці перед тим як перейти до наступного модулю. Для визначення зображення захопленого номеру найкращим, повинні виконуватися наступні умови:

- Номер знаходиться повністю в кадрі;
- Ширина номеру більша за висоту;
- Ширина номеру не зменшилася;
- Кадр є найчіткішим з попередньо захоплених.

Номер, який знаходиться повністю в кадрі означає, що всі кутові координати виявленого об'єкта знаходяться у кадрі повністю. Це дозволяє уникнути випадків коли не весь транспортний номер встиг потрапити у сектор виявлення, і якась його частина ще знаходиться за кадром.

Оскільки майже всі номерні знаки мають ширину більшу за довжину, ми виключаємо випадки виявлення лише певної частини номеру у випадку хибного розпізнавання чи ще якогось фактору. Через ці ж випадки зберігається кадр з найбільшою шириною, щоб не втратити важливу частину захопленого номера.

Для визначення чіткості кадру використовується оператор Лапласа. За допомогою нього визначаються краї зображення для вираховування різкості зображення. Чим більша сума масиву за результатом Лапласа, тим чіткішим і різкішим є зображення. Цей результат порівнюється з результатом попереднього найкращого кадру.

У випадку якщо всі умови враховані, то кадр номеру, який було виявлено останнім, вважається найкращим серед раніше виявлених.

Коли номерний знак зникає зі списку виявлених об'єктів, відбувається передача зображення цього номеру до наступного модулю. Для виключення випадку короткострокової втрати номера з поля зору, відбувається підрахунок кількості кадрів, на яких номерний знак не виявляється знову. У випадку якщо номер не виявлено протягом наступних 5 кадрів, вважається що машина проїхала і її можна зарахувати як транспорт що проїхав через пункт пропуску. До наступного модулю

передається зображення найчіткішого номера, повне зображення кадру, виявлений на кадрі клас автотранспорту. На рисунку 3.20 показано результат виявлення всіх видів транспортного засобу та номерних знаків на кадрах з двох різних камер.



Рисунок 3.20 – Результат роботи виявлення об’єктів з двох камер одночасно.

3.4.3 Модуль розпізнавання символів

Після того як з попереднього модулю отримали зображення номерного знаку, воно піддається первинній обробці для того щоб модель OCR було легше працювати з зображенням.

Моделі OCR конкретно спеціалізовані для роботи з текстом, через що є швидшими за інші інструменти для задачі розпізнавання символів. Є декілька основних OCR, які використовуються в програмах на Python, кожен з них має як свої переваги, так і недоліки. Для використання в розпізнаванні символів проведемо ряд експериментів, визначивши сильні та слабкі сторони кожного з них.[44]

Розглянемо перший на черзі OCR – TesseractOCR[45].

Переваги:

- Найшвидший і найбільш простий в розрахунках з розглянутих;
- Добре працює при рівному тексті високої роздільної здатності.

Недоліки:

- Потребує завантаження додаткового програмного забезпечення в систему;
- Потребує ретельного підготування зображення, особливо для зображення отриманого в динамічних умовах;
- Дуже чуттєвий до зміни положення символу. Малий нахил може призвести до відсутності розпізнавання символу.

Ще однією популярною моделлю виступає EasyOCR.

Переваги:

- Точніший за TesseractOCR;
- Немає чутливості до положення символу;
- Не потребує додаткових інсталяцій програмного забезпечення;
- Легке та розширене налаштування параметрів моделі.

Недоліки:

- Повільніший за TesseractOCR;
- Для уникнення неправильного розпізнавання також потребує попередню; обробку зображення.

Keras-OCR також є досить поширеним для задач розпізнавання символів.

Переваги:

- Найточніший з розглянутих;

- Легше розпізнавання кількох шрифтів, форматів та кольорів на зображенні.

Недоліки:

- Найповільніший з усіх розглянутих варіантів;
- Потребує постійної підвантаження даних з бібліотеки.

Отже, розглянувши дані моделі OCR, було вирішено працювати саме з EasyOCR. Оскільки текст на зображенні номера може бути не рівним, чіткість зображення може бути нижчою, EasyOCR краще розпізнає при таких випадках.

Перед тим як зображення буде передано до моделі розпізнавання, воно проходить етапи первинної обробки. Це підвищує точність розпізнавання символів на зображенні, що є важливим фактором при роботі з розпізнаванням номерів.

З попереднього модулю отримаємо найкращий кадр номерного знаку (рис. 3.21)



Рисунок 3.21 – Результат виявлення транспортного номеру та класифікація транспортного засобу

Якщо передати до моделі виділений кадр без збільшення розмірів, є вірогідність втрати важливої інформації або навпаки, виявлення відсутніх знаків. Для уникнення цього кадр попередньо збільшується в 6 разів по вертикалі та горизонталі. Таким чином, отримуємо збільшений номерний знак (рис. 3.22).



Рисунок 3.22 – Номерний знак до та після збільшення

Хоч новітні моделі OCR і підтримують розпізнавання таких зображень, провівши свої власні перетворення, вони все-ж таки можуть бути помилковими, або край зображення буде розпізнано як літеру, якою він не являється. Для цього продовжимо обробку зображення вручну.

Першим етапом є застосування гаусівського розмиття до зображення. Це дозволяє зменшити рівень шумів на зображенні, зробивши контури на зображенні більш гладкими та чіткішими (рис. 3.23).



Рисунок 3.23 – Застосування гаусівського розмиття

Наступним етапом обробки зображення є переведення відтінків зображення у сірі тони (рис. 3.24). Важливим аспектом даної обробки є те, що вона перетворює кольорові канали зображення в один канал сірого. Це в свою чергу сприяє простішому виявленню зміни інтенсивності пікселів, що спрощує подальшу обробку зображення.



Рисунок 3.24 – Переведення зображення у сірі відтінки

Попередній процес дозволяє перейти до наступного етапу, який переведе зображення в чорно-біле (рис. 3.25). Використання порогової функції виконує автоматичний підбір порогу для переведення частини пікселів в білий колір як фон, тоді як сильні контури перевести в чорний колір. Це дозволяє отримати зображення з двома можливими значеннями для пікселя, що дозволяє перейти до етапу визначення контурів.

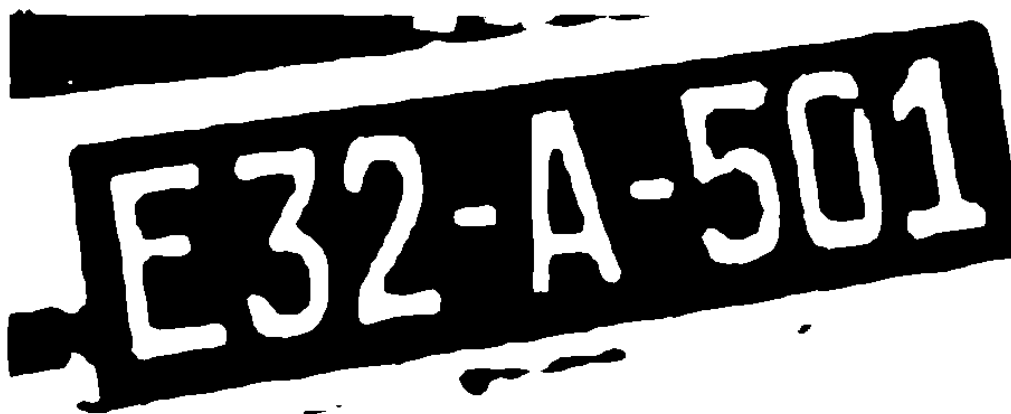


Рисунок 3.25 – Перетворення зображення в чорно-біле

Тепер ми можемо перейти до визначення контурів символів на зображенні. За допомогою функції виявлення контурів ми знаходимо області усіх різких переходів. Серед цих контурів є як і ті, що обводять потрібні символи, так і непотрібні, які виділяють середину 0 та А, а також краї зображення. Для цього застосовуємо декілька фільтрацій:

- Видаляємо поля, де ширина більша за довжину, оскільки всі символи вертикальні;
- Видаляємо області з мінімальною кількістю чорних пікселів;
- Видаляємо області які менші від областей літер на 70%.

За кінцевими результатами виділення контурів отримуємо виділені області, зображені на рисунку 3.26. У випадку, якщо на розпізнавання потрапить якась зайва область, вона малоймовірно буде розпізнана і буде просто проігнорована.



Рисунок 3.26 – Виділення областей символів

На розпізнавання передаються виділені на чорно-білому зображенні символи. Попередньо кожна літера інвертується, додаються поля для того щоб символ не був впритул до країв. Відбувається додаткове невелике збільшення кожного з кадрів. В результаті отримаємо окремі чорні літери на білому фоні (рис. 3.27).



Рисунок 3.27 – Окремі літери для розпізнавання

Наступним етапом є посимвольне розпізнавання. Кожна буква передається до моделі, після чого формується результат (рис. 3.28)

```
[([[0, 0], [621, 0], [621, 861], [0, 861]], 'E', 0.9977666702022283)]
[[[0, 0], [564, 0], [564, 843], [0, 843]], '3', 0.9999977350247811)]
[[[0, 0], [576, 0], [576, 846], [0, 846]], '2', 0.999999880790714)]
[[[0, 0], [546, 0], [546, 822], [0, 822]], 'A', 0.999926926040505)]
[[[0, 0], [579, 0], [579, 828], [0, 828]], '5', 0.9999896288186996)]
[[[0, 0], [564, 0], [564, 816], [0, 816]], '0', 0.6748894971417532)]
[[[0, 0], [573, 0], [573, 819], [0, 819]], '1', 0.9999992847443906)]
```

Рисунок 3.28 – Отримані результати при розпізнаванні символів

Після отримання повного транспортного номеру, результат передається до модулю створення JSON файлу, разом з найкращим кадром та визначеним типом транспорту.

3.4.4 Модуль створення JSON файлу

Для можливості подальшого використання результатів системи було вирішено створити модуль, який буде формувати JSON-POST файли для передачі опрацьованої інформації в пункти подальшої обробки. Для запису в файл з попереднього модулю передаються найкращий захоплений кадр, тип виявленого транспортного засобу та розпізнаний номер.

Для того щоб надіслати зображення у JSON файлі, воно попередньо конвертується в NumPy масив, який потім можна знову перетворити у зображення. Вміст створеного JSON файлу можна побачити на рисунку 3.29.

```

1 {"vehicle_type": "car", "number": "E32A501", "frame": [[[0, 5, 6],
  [0, 5, 6], [1, 6, 5], [4, 6, 6], [6, 5, 9], [7, 4, 6], [9, 5,
  4], [9, 4, 5], [7, 3, 9], [7, 3, 9], [10, 3, 6], [10, 5, 4], [12,
  4, 5], [10, 4, 5], [10, 5, 4], [7, 5, 4], [6, 5, 7], [1, 6, 5],
  [0, 6, 3], [0, 7, 4], [6, 5, 7], [8, 4, 9], [8, 3, 12], [8, 4,
  10], [11, 5, 6], [8, 6, 5], [0, 7, 4], [0, 7, 4], [1, 8, 5], [4,
  7, 5], [8, 6, 5], [10, 7, 2], [12, 11, 1], [9, 11, 0], [4, 8, 2],
  [4, 8, 3], [7, 5, 5], [6, 3, 5], [8, 4, 10], [10, 6, 11], [12,
  6, 7], [14, 7, 4], [16, 6, 6], [13, 6, 3], [7, 8, 0], [6, 10, 0],
  [11, 9, 1], [11, 11, 5], [6, 6, 6], [7, 6, 8], [12, 5, 10], [10,
  5, 7], [6, 7, 3], [5, 9, 4], [11, 6, 7], [11, 4, 7], [11, 5, 6],

```

Рисунок 3.29 – Приклад вмісту створеного JSON файлу

Для показу коректності зберігання інформації в JSON файл, реалізовано функцію читання цього файлу. Результатами читання зображені на рисунку 3.30. Як бачимо, було створено файл, в назві якого позначено дату захоплення кадру, тип транспортного засобу та його номер. Сам файл має перетворене зображення захопленого кадру.



Рисунок 3.30 – Вміст опрацьованого JSON файлу

В подальшому, за результатами перевірки передачі таких файлі та швидкістю передачі можна буде робити додаткове стискання зображення та оптимізації розміру файлу.

3.5 Висновок до розділу 3

В даному розділі було проведено аналіз існуючих моделей YOLO. YOLOv8 обрано як найкращу версію сімейства моделей YOLO, опираючись на результати досліджень. Розглянуті різні розміри моделі YOLOv8 для вибору найкращої для поставленої задачі за конкретних умов. Для даної програми було обрано модель розміру small. Досліджено архітектуру YOLOv8 як архітектуру одноетапного виявлення. Розглянуто такі складові як магістраль для витягування карт ознак з зображення, шия для агрегування отриманих карт ознак між собою для збереження основних та підсилення головних ознак, та голова для класифікації та виявлення об'єктів на зображенні.

Для використання моделі в мережі було проведено тренування моделі за заданими параметрами на створеному датасеті. Проведено аналіз результатів отриманої моделі за важливими метриками.

Перед початком розробки було вибрано програмне середовище для розробки та роботи програми, а також досліджено потрібні модулі та бібліотеки. Основними бібліотеками стали бібліотеки для захоплення зображення та його редагуванням, для роботи з моделями виявлення та розпізнавання тексту і для підключення відеокарти для обчислень.

Розроблено архітектуру програми, яку можна поділити на чотири модуля. Модуль захоплення зображення з відеокамери передає зображення з відеокамери до наступного модулю, при цьому показуючи вікна роботи програми та FPS системи. Модуль виявлення об'єктів на зображенні використовує навчену модель для

виявлення транспортного номеру та класифікації транспортного засобу, якому він належить. Модуль розпізнавання транспортного номеру відповідає за визначення символів з отриманого кадру номеру. Модуль формування JSON файлу формує дані для їх подальшого використання з метою подальшої обробки даних.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В даному розділі пройде оцінювання основних характеристик майбутньої системи візуального ідентифікування транспорту в режимі реального часу з використанням нейронних мереж.

Дана реалізація буде сприяти введенню нових систем для ідентифікування транспортного засобу на платних дорогах з метою спрощення та пришвидшення оплати для водіїв за користування дорогою не лише в Україні, а й в решті світу.

В даному дослідженні представлено різні варіанти реалізації для забезпечення найбільш правильної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) є технологією, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У даній роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи візуального ідентифікування транспорту в режимі реального часу. Оскільки рішення щодо проектування та реалізації компонентів, що розробляються, впливають на систему, кожна окрема підсистема має її задовольняти. Фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних, що є основою для розробки технічних вимог.

Визначаються наступні вимоги до програмного продукту:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка системи візуального ідентифікування транспорту в режимі реального часу. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування.

F_2 – вибір середовища розробки системи.

F_3 – вибір моделі розпізнавання символів.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1 :

а) Python.

б) Java.

Функція F_2 :

а) PyCharm.

б) DataSpell.

Функція F_3 :

а) EasyOCR.

б) TesseractOCR.

в) KerasOCR

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

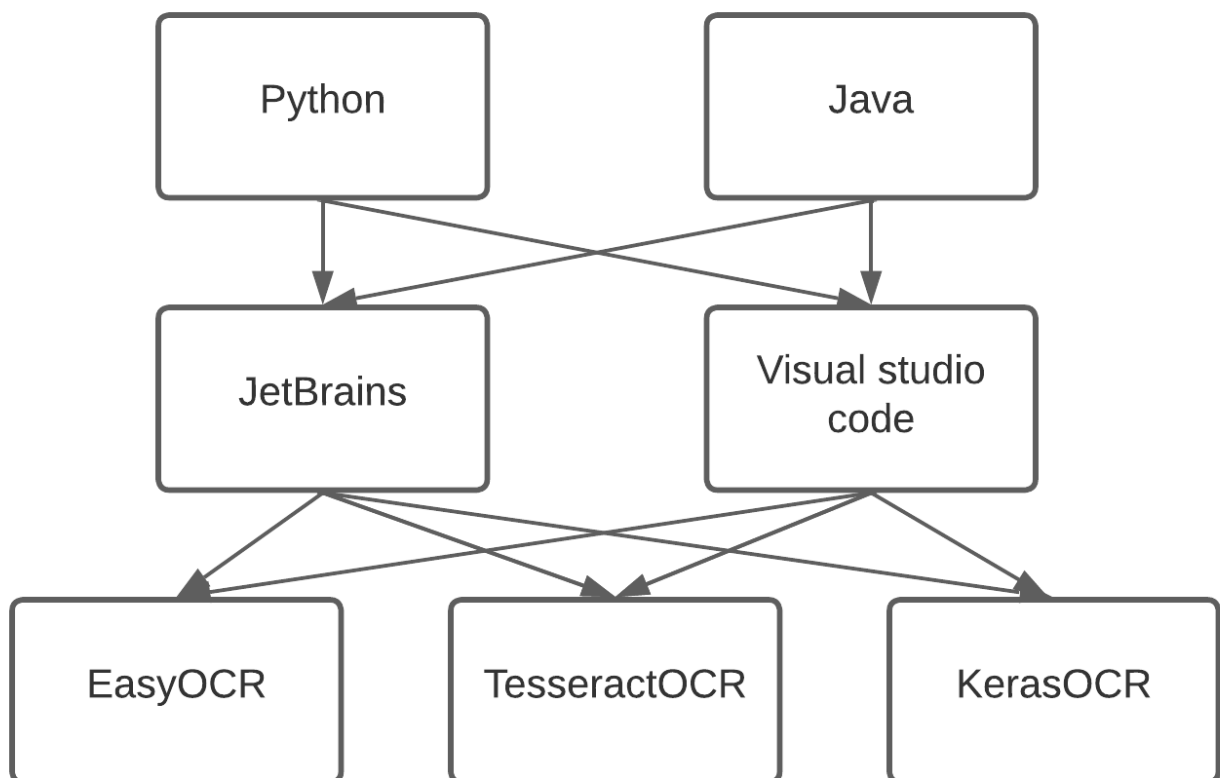


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	<i>A</i>	Простота синтаксису, велика кількість бібліотек, менша кількість рядків коду	Повільна швидкість роботи, потребує багато пам'яті, відсутність повноцінної багатопотоковості
	<i>B</i>	Висока надійність, висока швидкість, наявність багатопотоковості	Більший обсяг написання коду, витоки пам'яті, складність написання коду
F_2	<i>A</i>	Має середовища розробки для кожної мови, підтримка великої кількості інструментів, відмінна інтеграція з іншими інструментами	Потребує багато ресурсів комп'ютера, має багато специфічних налаштувань, через що на початку роботи може бути складним
	<i>B</i>	Підтримує багато мов програмування, велика кількість додаткових розширень, велика спілка розробників	Обмеженість функціоналу програного середовища, вимога до ресурсів комп'ютера, відсутність деяких функцій для різних мов програмування.
F_3	<i>A</i>	Простота використання, підтримка багатьох мов, точність роботи при різних умовах	Потребує більше потужностей комп'ютера
	<i>B</i>	Точність при високій якості зображення, багатомовність, швидкість	Залежність від параметрів налаштування, погане розпізнавання ненормованого тексту
	<i>B</i>	Гнучкість налаштування, висока точність при різних умовах	Потребує більше потужностей комп'ютера, повільна швидкість розпізнавання

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, оскільки вони не відповідають поставленим задачам перед системою. Ці варіанти відзначені у морфологічній карті.

Функція F_1 :

Надамо перевагу зручності та швидкості написання коду з підтримкою великої кількості бібліотек. Для спрощення написанню коду варіант Б буде відкинуто.

Функція F_2 :

Програма допускає обрання обох варіантів. Можливе використати варіанти А чи Б.

Функція F_3 :

Перший варіант є найбільш придатним для використання у системі. Обираємо варіант А.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

$$F_1a - F_2a - F_3a$$

$$F_1a - F_2b - F_3a$$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- X_1 – швидкодія мови програмування;
- X_2 – об'єм пам'яті для обчислень та збереження даних;
- X_3 – час навчання даних;
- X_4 – потенційний об'єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 - Основні параметри програмного продукту

Назва Параметра	Умовні позначенн я	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	6000	11000	17000
Об'єм пам'яті	X2	Мб	1024	512	256
Час попередньої обробки даних	X3	мс	7	5	4
Потенційний об'єм програмного коду	X4	кількість рядків коду	7000	3200	1100

За даними таблиці 4.3 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

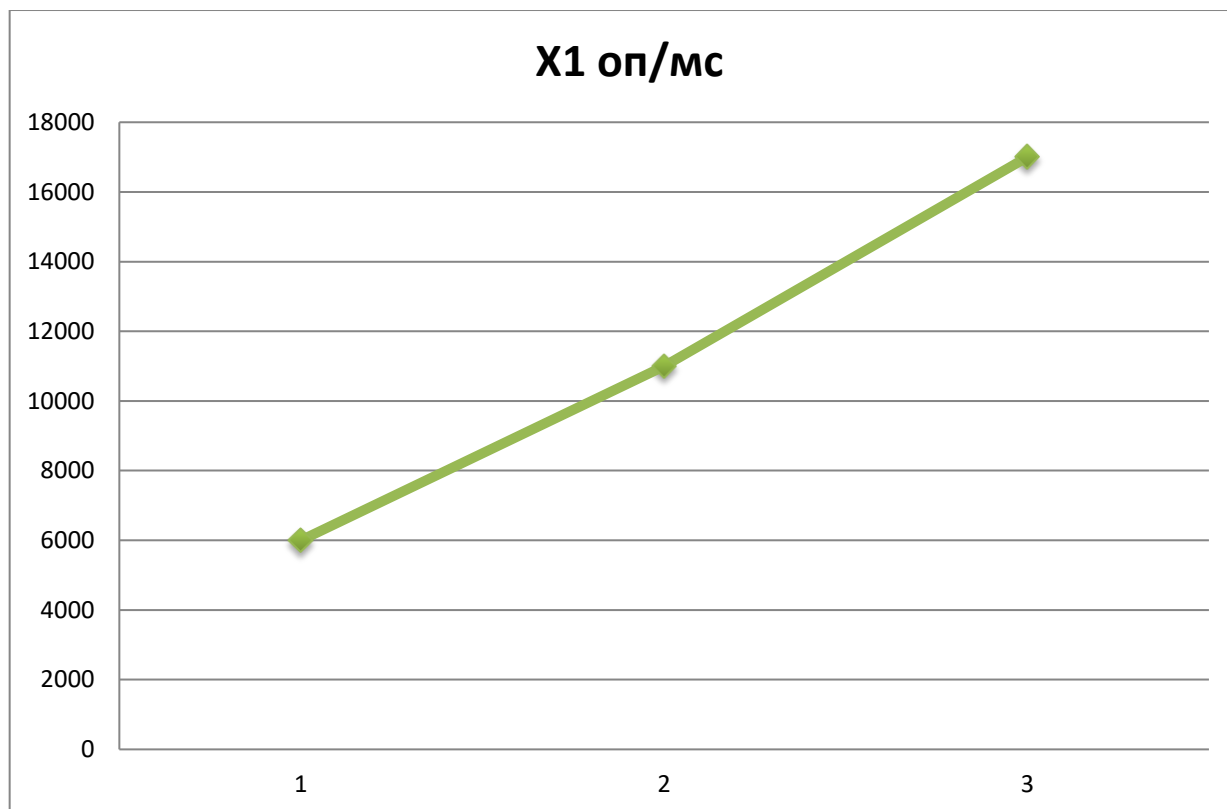


Рисунок 4.2 – X1, швидкодія мови програмування

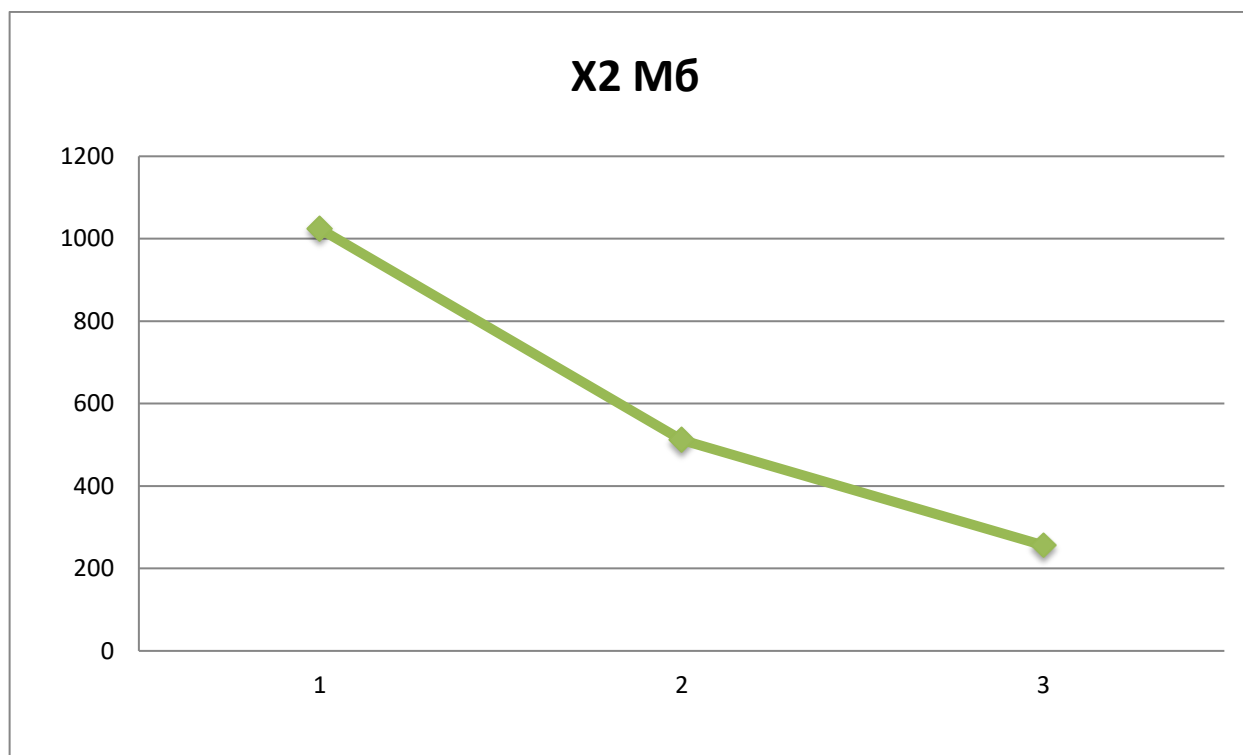


Рисунок 4.3 – X2, об'єм пам'яті

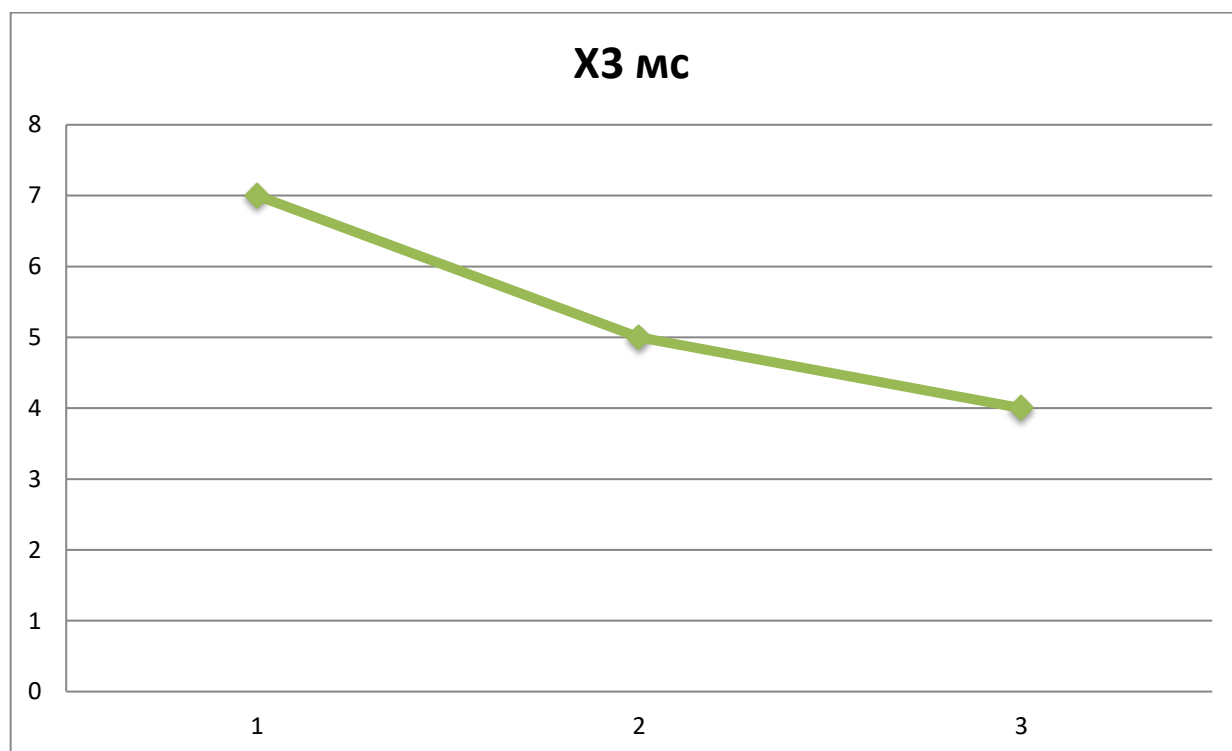


Рисунок 4.4 – X3, час попередньої обробки даних



Рисунок 4.5 – X4, потенційний об'єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
$X1$	Швидкодія мови програмування	Оп/мс	1	2	3	2	3	1	3	15.0	-9.50	90.25
$X2$	Об'єм пам'яті	Мб	2	1	6	3	5	5	7	29.0	4.50	20.25
$X3$	Час попередньої обробки даних	мс	7	5	3	4	5	5	3	32.0	7.50	56.25
$X4$	Потенційний об'єм програмного коду	Кількість рядків коду	4	6	2	5	1	3	1	22.0	-2.50	6.25
	Разом		14	14	14	14	14	14	14	98	0	173

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^N r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 98 \quad (4.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 24.5 \quad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T \quad (4.3)$$

Сума відхилень по всіх параметрах повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^N \Delta_i^2 = 173 \quad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 173}{7^2(4^3 - 4)} = 0,705 > W_k = 0,67 \quad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	>	<	<	<	<	<	<	0.5
X1 і X3	<	<	=	<	<	<	=	<	0.5
X1 і X4	<	<	>	<	>	<	>	<	0.5
X2 і X3	<	<	>	<	=	=	>	<	0.5

Продовження таблиці 4.4

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X2 і X4	<	<	>	<	>	>	>	>	1.5
X3 і X4	>	<	>	<	>	>	>	>	1.5

Числове значення, що визначає ступінь переваги i -го параметра над j -тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases} \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю $A = \|a_{ij}\|$.

Для кожного параметра зробимо розрахунок вагомості $K_{\epsilon i}$ за наступними формулами:

$$K_{\epsilon i} = \frac{b_i}{\sum_{i=1}^n b_i} \quad (4.7)$$

$$b_i = \sum_{j=1}^N a_{ij} \quad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\epsilon i} = \frac{b'_i}{\sum_{i=1}^n b'_i} \quad (4.9)$$

$$b'_i = \sum_{j=1}^N a_{ij} b_j \quad (4.10)$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\epsilon i}$	b_i^1	$K_{\epsilon i}^1$	b_i^2	$K_{\epsilon i}^2$
X1	1.0	0.5	0.5	0.5	2.5	0.16	9.25	0.16	34.125	0.16
X2	1.5	1.0	0.5	1.5	4.5	0.28	16.25	0.28	59.125	0.27
X3	1.5	1.5	1.0	1.5	5.5	0.34	21.25	0.36	77.875	0.36
X4	1.5	0.5	0.5	1.0	3.5	0.22	12.25	0.21	44.875	0.21
Всього:					16	1	59	1	216	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів $X2$ (об'єм пам'яті), $X3$ (час попередньої обробки даних) та $X4$ (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра $X1$ (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^n K_{ei,j} B_{i,j} \quad (4.11)$$

де n – кількість параметрів;

K_{ei} – коефіцієнт вагомості i -го параметра;

B_i – оцінка i -го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	11000	18	0.16	2.88
F3	А	X2	256	21	0.27	5.67
	Б	X3	512	14	0.36	5.04
F4	А	X4	1100	22	0.21	4.62

За даними з таблиці 4.6 за формулою:

$$K_K = K_{TY}[F_{1k}] + K_{TY}[F_{2k}] + \dots + K_{TY}[F_{zk}] \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 2.88 + 5.67 + 4.62 = 13.17;$$

$$K_{K2} = 2.88 + 5.04 + 4.62 = 12.54.$$

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;
2. Розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_O = T_P \cdot K_P \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М} \quad (4.13)$$

де T_P – трудомісткість розробки ПП;

K_P – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру ступеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_P = 36$ людино-днів. Поправочний коефіцієнт, який враховує вид

нормативно-довідкової інформації для першого завдання: $K_{\Pi} = 1.8$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ} = 0.9$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 36 \cdot 1.8 \cdot 0.9 = 58.32 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_p = 27$ людино-днів, $K_{\Pi} = 1.1$, $K_{СК} = 1$, $K_{СТ} = 0.8$:

$$T_2 = 27 \cdot 1.1 \cdot 0.8 = 23.76 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (58.32 + 23.76 + 4.8 + 23.76) \cdot 8 = 885.12 \text{ людино-годин.}$$

$$T_{II} = (58.32 + 23.76 + 6.91 + 23.76) \cdot 8 = 902.00 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 36568.60 грн., один аналітик в області даних з окладом 54852.90 грн.. Визначимо середню зарплату за годину за формулою:

$$C_{\text{ч}} = \frac{M}{T_m \cdot t} \text{ грн.} \quad (4.14)$$

де M – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$C_q = \frac{36568.60 + 36568.60 + 54852.90}{3 \cdot 21 \cdot 8} = 253.95 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$C_{зп} = C_q \cdot T_i \cdot K_d \quad (4.16)$$

де C_q – величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

K_d – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$\text{I.} \quad C_{зп} = 253.95 \cdot 885.12 \cdot 1.2 = 269729.99 \text{ грн.}$$

$$\text{II.} \quad C_{зп} = 253.95 \cdot 902.00 \cdot 1.2 = 274873.98 \text{ грн.}$$

Відрахування на єдиний соціальний внесок становить 22%:

$$\text{I.} \quad C_{вд} = C_{зп} \cdot 0.22 = 269729.99 \cdot 0.22 = 59340.60 \text{ грн.}$$

$$\text{II.} \quad C_{вд} = C_{зп} \cdot 0.22 = 274873.98 \cdot 0.22 = 60472.27 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години. (C_m)

Так як одна ЕОМ обслуговує одного програміста з окладом 36568.60 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$C_r = 12 \cdot M \cdot K_3 = 12 \cdot 36568.60 \cdot 0,2 = 87764.64 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{зп} = C_{г} \cdot (1 + K_3) = 87764.64 \cdot (1 + 0.2) = 105317.57 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{від} = C_{зп} \cdot 0.22 = 105317.57 \cdot 0.22 = 23169.86 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 20000 грн.

$$C_A = K_{TM} \cdot K_A \cdot Ц_{ПР} = 1.2 \cdot 0.25 \cdot 20000 = 6000 \text{ грн.,}$$

де K_{TM} – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

K_A – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_P = K_{TM} \cdot Ц_{ПР} \cdot K_P = 1.2 \cdot 20000 \cdot 0.08 = 1920.00 \text{ грн.,}$$

де K_P – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$\begin{aligned} T_{ЕФ} &= (D_K - D_B - D_C - D_P) \cdot t_3 \cdot K_B = (365 - 104 - 8 - 16) \cdot 8 \cdot 0.5 = \\ &= 948.00 \text{ годин,} \end{aligned}$$

де D_K – календарна кількість днів у році;

D_B, D_C – відповідно кількість вихідних та святкових днів;

D_P – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

K_B – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{EL} = T_{EF} \cdot N_C \cdot K_3 \cdot C_{EH} = 948.00 \cdot 0.2 \cdot 0.3 \cdot 4.79 = 272.46 \text{ грн.},$$

де N_C – середньо-споживча потужність приладу;

K_3 – коефіцієнтом зайнятості приладу;

C_{EH} – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$C_H = C_{PP} \cdot 0.67 = 20000 \cdot 0.67 = 13400 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{EKC} = C_{3П} + C_{ВІД} + C_A + C_P + C_{EL} + C_H \quad (4.17)$$

$$C_{EKC} = 105317.57 + 23169.86 + 6000.00 + 1920.00 + 272.46 + 13400.00 = 150079.89 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{M-Г} = C_{EKC} / T_{EF} = 150079.89 / 948.00 = 158.31 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов'язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_M = C_{M-\Gamma} \cdot T \quad (4.18)$$

$$\text{I.} \quad C_M = 158.31 \cdot 885.12 = 140123.35 \text{ грн.}$$

$$\text{II.} \quad C_M = 158.31 \cdot 902.00 = 142795.62 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$C_H = C_{ЗП} \cdot 0.67 \quad (4.19)$$

$$\text{I.} \quad C_H = 269729.99 \cdot 0.67 = 180719.10 \text{ грн.}$$

$$\text{II.} \quad C_H = 274873.98 \cdot 0.67 = 184165.56 \text{ грн.}$$

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_M + C_H \quad (4.20)$$

$$\text{I.} \quad C_{ПП} = 269729.99 + 60472.27 + 140123.35 + 180719.10 = 651044.71 \text{ грн.}$$

$$\text{II.} \quad C_{ПП} = 274873.98 + 59340.60 + 142795.62 + 184165.56 = 661175.76 \text{ грн.}$$

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{\text{ТЕР}j} = K_{kj} / C_{\Phi j} \quad (4.21)$$

$$K_{\text{ТЕР}1} = 13.17 / 651044.71 = 2.0229 \cdot 10^{-5},$$

$$K_{\text{ТЕР}2} = 12.54 / 661175.76 = 1.89662 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $K_{\text{ТЕР}1} = 2.0229 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $K_{\text{ТЕР}} = 2.0229 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- Мова програмування – Python;
- Простота використання та швидкість розробки програми
- Велика кількість бібліотек, наявність готових програмних реалізацій

Даний варіант виконання програмного комплексу дає користувачу зрозумілий та доступний для роботи функціонал, швидку та зручну реалізацію програми.

4.8 Висновки до розділу 4

В даній частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

За результатом виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу вибрано найкращий варіант реалізації програмного продукту.

ВИСНОВОК

В даній бакалаврській дипломній роботі було проведено дослідження використання моделей нейронних мереж для задачі ідентифікації та класифікації транспортного засобу в режимі реального часу.

В першому розділі було досліджено різні варіанти ідентифікації транспорту в області сплати грошей за користування платними автобанами. Також розглянуті основні методи для виявлення об'єктів на зображенні, за результатом дослідження яких виявилось, що нейронні мережі підходять найкраще для задачі розпізнавання. Також були розглянуті основні задачі розпізнавання, за результатом яких задачі виявлення була обрана як головна для роботи.

В другому розділі було розглянуто використання нейронних мереж для задачі розпізнавання, в результаті чого виявилось що найкращим варіантом для цієї задачі є використання згорткової нейронної мережі через свої додаткові шари, які пришвидшують та полегшують опрацювання зображень. Розглянуті всі етапи, які проходить зображення в згортковій нейронній мережі, а саме шар згортки, шар активації, шар агрегації та повнозв'язний шар. Розглянуті різні види функцій активацій та види агрегації. Також були розглянуті найпопулярніші види оптимізаторів для тренування нейронних мереж. Було досліджено основні види моделей розпізнавання а також метрики, за якими вони оцінюються. Розглянуті найпопулярніші моделі для використання в режимі реального часу, за результатами їх порівняння було обрано модель YOLO.

В третьому розділі було досліджено різновиди моделей сімейства YOLO, обрано найкращу модель для ситуації яка розглядається. Описано процес збирання навчальних даних та навчання моделі, перевірка її результати на тестових екземплярах. Показано алгоритм роботи програмного забезпечення для використання в системі розпізнавання та описано процес її роботи.

В четвертому розділі було проведено функціонально-вартісний аналіз програмного забезпечення, за результатом якого було обрано оптимальний варіант реалізації програмного продукту.

Під час створення роботи було реалізовано систему для ідентифікації та класифікації транспортного засобу в режимі реального часу з результатом у вигляді файлу, який може використовуватися для подальшої обробки отриманої інформації, яка може стосуватися не тільки оплати за користування автобаном.

Подальша робота полягає у покращенні даної системи для більш точного та безпомилкового виявлення, оскільки правильність розпізнавання номеру є дуже важливим параметром. Також можливе покращення системи для збільшення областей використання продукту та спрощення налаштування для досягнення потрібних результатів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Платні дороги в Європі 2019. [Електронний ресурс] – режим доступу: https://auto.24tv.ua/platni_dorohy_yevropy_tsiny_na_platni_dorohy_yak_i_de_platyty_n7076
2. Vignette for Motorway. [Електронний ресурс] – режим доступу: <https://guidetoslovenia.com/1956/how-to-pay-for-motorway-in-slovenia-vignette-for-motorway/>
3. Що таке EETS. [Електронний ресурс] – режим доступу: <https://etoll.gov.pl/ua/vantazhn/sistema-e-toll/eets/sho-take-eets/>
4. Telepass device. [Електронний ресурс] – режим доступу: <https://www.telepass.com/en/consumer/offers/telepass-pay-per-use>
5. Wikipedia contributors. Open road tolling. Wikipedia. [Електронний ресурс] – режим доступу: https://en.wikipedia.org/wiki/Open_road_tolling
6. Kiran, V. K., Parida, P., & Dash, S. Vehicle Detection and Classification: A Review. Springer eBooks, 45–56. [Електронний ресурс] – режим доступу: https://www.researchgate.net/publication/343482198_Vehicle_Detection_and_Classification_A_Review
7. Saml0w, V. a. P. B. Frame Differencing. Squawk-Fly. [Електронний ресурс] – режим доступу: <https://squawkfly.wordpress.com/2015/05/31/frame-differencing/>
8. Gulati, A. P. Vehicle Motion Detection using Background Subtraction. Analytics Vidhya. [Електронний ресурс] – режим доступу: <https://www.analyticsvidhya.com/blog/2022/03/vehicle-motion-detection-using-background-subtraction/>
9. Ramachandran, S. What Is Image Recognition? Nanonets AI & Machine Learning Blog. [Електронний ресурс] – режим доступу: <https://nanonets.com/blog/image-recognition/>

10. Rizzoli, A. The Ultimate Guide to Object Detection. V7. [Электронный ресурс] – режим доступа: <https://www.v7labs.com/blog/object-detection-guide>
11. Image Processing: How Do Image Classifiers Work? [Электронный ресурс] – режим доступа: <https://levity.ai/blog/how-do-image-classifiers-work>
12. Gavrilova, Y. Convolutional Neural Networks for Beginners. Serokell Software Development Company. [Электронный ресурс] – режим доступа: <https://serokell.io/blog/introduction-to-convolutional-neural-networks>
13. Lang, N. Breaking down Convolutional Neural Networks: Understanding the Magic behind Image Recognition. Medium. [Электронный ресурс] – режим доступа: <https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4>
14. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way | Saturn Cloud Blog. [Электронный ресурс] – режим доступа: <https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/>
15. Bonner, A. The Complete Beginner's Guide to Deep Learning: Convolutional Neural Networks and Image Classification. Medium. [Электронный ресурс] – режим доступа: <https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb>
16. Panneerselvam, L. Activation Functions and their Derivatives – A Quick & Complete Guide. Analytics Vidhya. [Электронный ресурс] – режим доступа: <https://www.analyticsvidhya.com/blog/2021/04/activation-functions-and-their-derivatives-a-quick-complete-guide/>
17. Difference of Activation Functions in Neural Networks in general. Data Science Stack Exchange. [Электронный ресурс] – режим доступа: <https://datascience.stackexchange.com/questions/14349/difference-of-activation-functions-in-neural-networks-in-general>
18. Leung, K. The Dying ReLU Problem, Clearly Explained - Towards Data Science. Medium. [Электронный ресурс] – режим

- доступу:<https://towardsdatascience.com/the-dying-relu-problem-clearly-explained-42d0c54e0d24>
- 19.iMerit Technology Services. Picking the Best ML Optimizer. iMerit. [Электронный ресурс] – режим доступа:<https://imerit.net/blog/picking-the-best-ml-optimizer-all-pbm/>
 - 20.Patrikar, S. Batch, Mini Batch & Stochastic Gradient Descent - Towards Data Science. Medium. [Электронный ресурс] – режим доступа:<https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a>
 - 21.Ruder, S. An overview of gradient descent optimization algorithms. ruder.io. [Электронный ресурс] – режим доступа:<https://www.ruder.io/optimizing-gradient-descent/>
 - 22.Doshi, S. Various Optimization Algorithms For Training Neural Network. Medium. [Электронный ресурс] – режим доступа:<https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6>
 - 23.Chauhan, N. S. Optimization Algorithms in Neural Networks - KDnuggets. KDnuggets. [Электронный ресурс] – режим доступа:
<https://www.kdnuggets.com/2020/12/optimization-algorithms-neural-networks.html>
 - 24.Kukil, & Kukil. Mean Average Precision (mAP) in Object Detection. LearnOpenCV – Learn OpenCV, PyTorch, Keras, Tensorflow With Examples and Tutorials. [Электронный ресурс] – режим доступа:<https://learnopencv.com/mean-average-precision-map-object-detection-model-evaluation-metric/>
 - 25.Boesch, G. Object Detection in 2023: The Definitive Guide. viso.ai. [Электронный ресурс] – режим доступа:<https://viso.ai/deep-learning/object-detection/>
 - 26.Tan, L., Huangfu, T., Wu, L., & Chen, W. Comparison of RetinaNet, SSD, and YOLO v3 for real-time pill identification. BMC Medical Informatics and Decision Making, 21(1). [Электронный ресурс] – режим доступа:<https://doi.org/10.1186/s12911-021-01691-8>

- 27.К, В. Object Detection Algorithms and Libraries. neptune.ai. [Электронный ресурс] – режим доступа:<https://neptune.ai/blog/object-detection-algorithms-and-libraries>
- 28.Hui, J. Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3). Medium. [Электронный ресурс] – режим доступа:<https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359>
- 29.YOLO models for Object Detection Explained [YOLOv8 Updated]. Encord. [Электронный ресурс] – режим доступа:<https://encord.com/blog/yolo-object-detection-guide/>
- 30.YOLOv8 Model. [Электронный ресурс] – режим доступа:<https://github.com/RangeKing>
- 31.Solawetz, J. What is YOLOv8? The Ultimate Guide. Roboflow Blog. [Электронный ресурс] – режим доступа:<https://blog.roboflow.com/whats-new-in-yolov8/>
- 32.Papers with Code - SiLU Explained. [Электронный ресурс] – режим доступа:<https://paperswithcode.com/method/silu>
- 33.Imane, C. YOLO v5 model architecture [Explained]. OpenGenus IQ: Computing Expertise & Legacy. [Электронный ресурс] – режим доступа:<https://iq.opengenus.org/yolov5/>
- 34.Solawetz, J. What is YOLOv5? A Guide for Beginners. Roboflow Blog. [Электронный ресурс] – режим доступа:<https://blog.roboflow.com/yolov5-improvements-and-evaluation/>
- 35.Khandelwal, R. Different IoU Losses for Faster and Accurate Object Detection. Medium. [Электронный ресурс] – режим доступа:<https://medium.com/analytics-vidhya/different-iou-losses-for-faster-and-accurate-object-detection-3345781e0bf>
- 36.Distribution Focal Loss. [Электронный ресурс] – режим доступа:<https://arxiv.org/pdf/2006.04388v1.pdf>

37. Martinek, V. Cross-entropy for classification - Towards Data Science. Medium. [Електронний ресурс] – режим доступу: <https://towardsdatascience.com/cross-entropy-for-classification-d98e7f974451>
38. Roboflow: [Електронний ресурс] – режим доступу: <https://roboflow.com/>
39. Le, B. Get Started with Training a YOLOv8 Object Detection Model. [Електронний ресурс] – режим доступу: <https://www.datature.io/blog/get-started-with-training-a-yolov8-object-detection-model>
40. Ultralytics. Classify. [Електронний ресурс] – режим доступу: <https://docs.ultralytics.com/usage/cfg/#train>
41. Kundu, R. F1 Score in Machine Learning: Intro & Calculation. V7. [Електронний ресурс] – режим доступу: <https://www.v7labs.com/blog/f1-score-guide>
42. Ultralytics. YOLOv8. Ultralytics YOLOv8 Docs. [Електронний ресурс] – режим доступу: <https://docs.ultralytics.com/models/yolov8/>
43. EasyOCR для розпізнавання символів. [Електронний ресурс] – режим доступу: <https://github.com/JaidedAI/EasyOCR>
44. Sami, T. “Tesseract” vs “Keras-OCR” vs “EasyOCR” - MLearning.ai - Medium. Medium. [Електронний ресурс] – режим доступу: <https://medium.com/mlearning-ai/tesseract-vs-keras-ocr-vs-easyocr-ec8500b9455b>
45. Tesseract Open Source OCR Engine (main repository). GitHub. [Електронний ресурс] – режим доступу: <https://github.com/tesseract-ocr/tesseract>

ДОДАТОК А

```
#FIRST MODULE WORK WITH CAMERAS
```

```
import cv2
import time
import Vehicle_detection as vd

def camera_amount():
    cameras_in_system = 0
    while True:
        cap = cv2.VideoCapture(cameras_in_system)
        if cap.read()[0]:
            cameras_in_system += 1
        else:
            break
        cap.release()
    print(cameras_in_system)
    return cameras_in_system

def start_multiple_cameras():
    fps_start_time = time.time()
    fps_frames = 0
    video_capture = {}

    cameras_amount = camera_amount()

    for i in range(cameras_amount):
        video_capture[i] = cv2.VideoCapture(i, cv2.CAP_DSHOW)

    for i in range(cameras_amount):
        cv2.namedWindow('Cam {}'.format(i))
```

```

while True:
    for i in range(cameras_amount):
        ret, frame = video_capture[i].read()
        if ret:
            fps_frames += 1
            if time.time() - fps_start_time >= 1:
                fps = fps_frames / (time.time() - fps_start_time)
                fps_frames = 0
                fps_start_time = time.time()

            frame = cv2.resize(frame, (640, 480))
            cv2.putText(frame, f"FPS: {fps:.2f}", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
            vd.detection(frame, i)
            cv2.imshow('Cam {}'.format(i), frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    for i in range(cameras_amount):
        video_capture[i].release()
    cv2.destroyAllWindows()

start_multiple_cameras()

#SECOND MODULE VEHICLE AND PLATE DETECTION

import numpy as np
from ultralytics import YOLO
import cv2
import Plate_recognition as pr

model = YOLO('best.pt')

```

```

distance_quality_data = np.zeros((3, 10))

best_plate_photo = {}
best_frame = {}
detected_vehicle_type = [""] * 10
def pre_ready(frame):
    height, width = frame.shape[:2]
    right_width = width // 2
    right_image = frame[:, right_width:, :]
    return right_image

def detection_sorted(detection_results):
    obj_locations = np.array(detection_results.xyxy.cpu(),
dtype="int")
    detected_obj = np.array(detection_results.cls.cpu(), dtype="int")
    location_sorted_indexes = np.argsort(obj_locations[:, 0])
    sorted_locations = obj_locations[location_sorted_indexes]
    sorted_objects = detected_obj[location_sorted_indexes]
    show_bbox= {key: value for key, value in zip(sorted_objects,
sorted_locations)}
    return sorted_locations, sorted_objects, show_bbox

def detection(frame, camera_used):
    detection_area = pre_ready(frame)
    results = model(detection_area, conf=0.45, device='0')
    srt_lct, srt_obj, bbox = detection_sorted(results[0].boxes)

    for obj_class, coordinates in bbox.items():
        x1, y1, x2, y2 = coordinates
        print(srt_lct)
        cv2.rectangle(detection_area, (x1, y1), (x2, y2), (0, 0, 225),
2)

        cv2.putText(detection_area, model.names[obj_class], (x1, y1 -
5), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 225, 0), 2)

```

```

    if 3 in srt_obj:
        plate_coordinates = np.array(srt_lct[np.argmax(srt_obj ==
3) ]).flatten()
        x1, y1, x2, y2 = plate_coordinates
        quality_rate = np.sum(cv2.Laplacian(detection_area[y1:y2,
x1:x2], cv2.CV_64F) ** 2)
        distance_quality_data[1, camera_used] = 5
        width = x2 - x1
        length = y2 - y1

        if (distance_quality_data[0, camera_used] < quality_rate and
plate_coordinates.all() > 0
            and distance_quality_data[2, camera_used] < width and
width > length):
            distance_quality_data[2, camera_used] = width
            distance_quality_data[0, camera_used] = quality_rate
            best_plate_photo[camera_used] =
np.asarray(detection_area[y1:y2, x1:x2])
            best_frame[camera_used] = np.asarray(frame)
            detected_vehicle_type[camera_used] =
results[0].names[srt_obj[0]]

        elif (distance_quality_data[0, camera_used] > 0 and
distance_quality_data[1, camera_used] > 0):
            distance_quality_data[1, camera_used] -= 1

        elif (distance_quality_data[0, camera_used] > 0 and
distance_quality_data[1, camera_used] == 0):
            pr.get_plate(best_plate_photo[camera_used],
best_frame[camera_used], detected_vehicle_type[camera_used])
            distance_quality_data[0, camera_used] = 0
            distance_quality_data[1, camera_used] = 5
            distance_quality_data[2, camera_used] = 0

```

```

        del best_plate_photo[camera_used]
        detected_vehicle_type[camera_used] = ""

#THIRD MODULE LICENSE PLATES RECOGNATION

import cv2
import easyocr
import numpy as np
import JSON_POST as JP

reader = easyocr.Reader(['en'], gpu=True)
detected_letters = {}
final_plate = []
width = []

def tresh_hold(letters, width):
    if len(width)>0:
        max_value = max(width)
        threshold = max_value * 0.7
        for key, value in list(letters.items()):
            if value[1] < threshold:
                del letters[key]
    return letters

def get_plate(plate, full_frame, vehicle_cls):
    plt_h, plt_w, _ = plate.shape
    resized = cv2.resize(plate, None, fx=6, fy=6)
    blur = cv2.GaussianBlur(resized, (3, 3), 0)
    gray = cv2.cvtColor(blur, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_OTSU |
cv2.THRESH_BINARY_INV)
    contours, _ = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    for contour in contours:

```

```

x, y, w, h = cv2.boundingRect(contour)
if(w < h):
    frame = cv2.bitwise_not(thresh[y:y + h, x:x + w])
    frame = cv2.copyMakeBorder(frame, 50, 50, 50, 50,
cv2.BORDER_CONSTANT, value=[255, 255, 255])
    white_pixels = np.count_nonzero(frame == 255)
    black_pixels = np.count_nonzero(frame == 0)
    coef = black_pixels/white_pixels
    if(coef > 0.05):
        width.append(w)
        frame = cv2.resize(frame, None, fx=3, fy=3)
        detected_letters[x] = (np.asarray(frame), w)
        cv2.rectangle(resized, (x, y), (x + w, y + h), (0, 0,
255), 2)
    cv2.imshow("r", resized)
    final_list = tresh_hold(detected_letters, width)

## STANDART PLATE
if (plt_w > plt_h * 2):
    sorted_dict = dict(sorted(final_list.items(), key=lambda x:
x[1][2]))
    for i, j in sorted_dict.items():
        letter = reader.recognize(j[0],
allowlist='123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ')
        if letter[0][1] != '':
            final_plate.append(letter[0][1])

## BIKE, SMALL PLATE
else:
    char_borders = []
    low_line = []
    top_line = []
    sorted_dict_square = dict(sorted(final_list.items(),
key=lambda x: -x[1][3]))

```

```

        min_h = min(sorted_dict_square.values(), key=lambda x:
x[1])[4]
        min_w = min(sorted_dict_square.values(), key=lambda x:
x[1])[1]
        for i in sorted_dict_square.values():
            temp_arr = []
            temp_arr.append(i[2])
            temp_arr.append(i[3])
            char_borders.append(temp_arr)
        for i in range(len(char_borders) - 1):
            if (char_borders[i][1] - char_borders[i + 1][1] > min_h):

low_line.append(sorted_dict_square[char_borders[i][0]][0])
                i += 1
                break
            else:

low_line.append(sorted_dict_square[char_borders[i][0]][0])
                for j in range(i, len(char_borders) - 1):
                    if (abs(char_borders[j][0] - char_borders[j + 1][0]) >
min_w * 2):

top_line.append(sorted_dict_square[char_borders[j][0]][0])
                        break
                    else:

top_line.append(sorted_dict_square[char_borders[j][0]][0])
                        top_line = top_line + low_line
                        for fp in range(len(top_line), len(char_borders)):

top_line.append(sorted_dict_square[char_borders[fp][0]][0])

                for i in top_line:
                    letter = reader.recognize(i,
allowlist='123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ')

```

```

        if letter[0][1] != '':
            final_plate.append(letter[0][1])

width.clear()
detected_letters.clear()
JP.create_json_post(full_frame, vehicle_cls, "".join(final_plate))
final_list.clear()

```

THE FOURTH MODULE GENERATES A JSON FILE

```

import json
import numpy as np
import time
import cv2
import os
location = "Kyiv_gate2_enter"

def create_json_post(full_frame, vehicle_cls, plate_number):

    ##CREATING JSON
    frame_in_arr = np.asarray(full_frame).tolist()

    data = {
        "vehicle_type": vehicle_cls,
        "number": plate_number,
        "frame": frame_in_arr
    }
    file_path = f"{time.strftime('%d_%m_%Y-%H_%M_%S')}.json"
    with open(file_path, "w") as json_file:
        json.dump(data, json_file)

    ##READING JSON
    with open(file_path, "r") as json_file:
        json_object = json.load(json_file)

```

```

vehicle = json_object["frame"]
vehicle = np.array(vehicle)
vehicle = vehicle.astype(np.uint8)
frame_data = os.path.splitext(file_path)[0]
number = json_object["number"]
vehicle_type = json_object["vehicle_type"]
cv2.imwrite(f"{frame_data}-{vehicle_type}-{number}.jpg", vehicle)

```

#MODEL TRAINING AND TESTING

```
!pip install ultralytics
```

```

import ultralytics
ultralytics.checks()
from ultralytics import YOLO

```

```
!pip install roboflow
```

```

from roboflow import Roboflow
rf = Roboflow(api_key="####")
project = rf.workspace("####").project("####")
dataset = project.version(2).download("yolov8")

```

```

!yolo task=detect mode=train model=yolov8s.pt
data=/content/####/data.yaml batch=-1 epochs=40 imgsz=640 cache=True
optimizer='Adam' momentum=0.9 lr0=0.0001 plots=True

```

```

!yolo task=detect mode=val
model=/content/runs/detect/train/weights/best.pt
data=/content/####/data.yaml name=test

```

```

!yolo task=detect mode=val
model=/content/runs/detect/train/weights/best.pt
data=/content/####/data.yaml name=test split='test'

```