Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інтелектуальних інформаційних технологій та автоматики
(повне найменування інституту, назва факультету (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Бакалаврська дипломна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ РОЗПІЗНАВАННЯ РУХОМИХ ОБ'ЄКТІВ

Виконав: студент 4 курсу, групи КН-22мс
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Бевз А.О.
(прізвище та ініціали)
Керівник: к.т.н., доцент кафедри КН
Озеранський В.С.
(прізвище та ініціали)
«____»_____2024 р.
Рецензент: к.т.н., доцент кафедри САІТ
Варчук І.В.
(прізвище та ініціали)
«07»06 2024 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
(прізвище та ініціали)
«07»06 2024 р.

Вінниця ВНТУ – 2024 рік

нницький національний технічний університет
акультет інтелектуальних інформаційних технологій та автоматизації
федра Комп'ютерних наук
вень вищої освіти перший (бакалаврський)
лузь знань – 12 «Інформаційні технології»
еціальність – 122 «Комп'ютерні науки»
світньо-професійна програма – «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
"07" 06 2024 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Бевзу Артему Олександровичу

Тема роботи: Програмний модуль розпізнавання рухомих об'єктів.
ївник роботи: Озеранський Володимир Сергійович, к.т.н., доцент
верджені наказом вищого навчального закладу "11" 03 2024 року №80

Строк подання студентом роботи 27. 05. 2024

Вихідні дані до роботи :
рмат вхідних зображень - jpg, jpeg або png, розмірність вхідних зображень – не енше 400x400, кількість класів розпізнавання – 80, обсяг навчальної вибірки – не енше 1000 зображень, обсяг тестової вибірки – не менше 100 зображень, мова ограмування – об'єктно-орієнтована.
ова програмування – об'єктно-орієнтована;

Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно робити):
уп, обгрунтування доцільності розробки програмного модуля розпізнавання комих об'єктів; аналіз переваг та недоліків існуючих програм для розпізнавання комих об'єктів; створення структури програмного модуля розпізнавання рухомих єктів; розробка алгоритму роботи програмного модуля розпізнавання рухомих єктів; висновки, перелік використаних джерел, додатки.

Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
ема алгоритму роботи модуля розпізнавання рухомих об'єктів;
грами компонентів модуля розпізнавання рухомих об'єктів;
альний вигляд інтерфейсу користувача модуля розпізнавання рухомих об'єктів;

Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дат	
		завдання видав	завдан прийн
1-3	Озеранський В.С., к.т.н., доцент		

6. Дата видачі завдання 07.03.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		При
		початок	закінчення	
1	Обґрунтування доцільності розробки програмного модуля розпізнавання рухомих об'єктів	06.05.2024	07.05.2024	
2	Розробка програмного модуля розпізнавання рухомих об'єктів	08.05.2024	09.05.2024	
3	Програмна реалізація модуля розпізнавання рухомих об'єктів	13.05.2024	14.05.2024	
4	Аналіз функціонування програмного модуля розпізнавання рухомих об'єктів	19.05.2024	19.05.2024	
5	Оформлення додатків	23.05.2024	26.05.2024	
6	Розробка інструкції користувача	26.05.2024	27.05.2024	
7	Оформлення матеріалів до захисту БДР	04.06.2024	06.06.2024	

Студент _____ Бевз
(підпис) (прізвище т

Керівник проекту (роботи) _____ Озерансь
(підпис) (прізвище т

АНОТАЦІЯ

УДК 621.374.415

Бевз А.О. Програмний модуль розпізнавання рухомих об’єктів. Бакалаврська дипломна робота складається з 65 сторінок формату А4, на яких є 31 рисунок, список використаних джерел містить 21 найменування.

В даній бакалаврській дипломній роботі досліджено процес розробки програмного модуля розпізнавання рухомих об’єктів.

Проведено огляд відомих методів для роботи з інтелектуальними обчисленнями у сфері комп’ютерного зору. Здійснено аналіз відомих програмних засобів розпізнавання рухомих об’єктів та як аналог до розроблюваного модуля було обрано програму Inception. Обґрунтовано доцільність використання для розпізнавання рухомих об’єтів згорткової нейронної мережі, а саме – попередньо натренованої нейронної мережі YOLOv2. Була проаналізована структура та порядок функціонування згорткової нейромережі YOLOv2. В роботі було проведено реалізацію програмного модуля розпізнавання об’єктів у зображеннях та модуля аналізу та візуалізації вихідних даних після обробки мовою програмування C# у середовищі Visual Studio.

Ключові слова: детектування, розпізнавання, нейронна мережа, тренування, YOLOv2, C#, Visual Studio.

ABSTRACT

Bevz A.O. Software module for recognition of moving objects. The bachelor's thesis consists of 65 pages of A4 format, on which there are 31 figures, the list of used sources contains 21 names.

In this bachelor's thesis, the process of developing a software module for recognizing moving objects is investigated.

An overview of known methods for working with intelligent computing in the field of computer vision was conducted. An analysis of well-known software tools for recognizing moving objects was carried out, and the Inception program was chosen as an analogue to the developed module. The expediency of using a convolutional neural network, namely a pre-trained YOLOv2 neural network, for the recognition of moving objects is substantiated. The structure and order of functioning of the YOLOv2 convolutional neural network were analyzed. In the work, the implementation of the software module for object recognition in images and the module for analysis and visualization of raw data after processing in the C# programming language in the Visual Studio environment was carried out.

Keywords: detection, recognition, neural network, training, YOLOv2, C#, Visual Studio.

ЗМІСТ

ВСТУП

Актуальність досліджень. У наш час роль інформації у всіх сферах людського життя є надзвичайно великою. Пошук та обробка інформації є одними із найбільш пріоритетних цілей будь-якої сучасної установи, починаючи з приватної компанії, що виробляє продукцію та надає послуги, закінчуючи передовими науковими лабораторіями. Оскільки об’єми інформації на сьогоднішній день є надзвичайно великими, то передусім постає задача автоматизації обробки цієї інформації.

Нині існує безліч підходів обробки візуальної інформації, тому що зараз вона є найпоширенішою. Однак питання автоматизованої обробки цифрових зображень або відео стають ключовими, адже потребують розробки більш складніших алгоритмів та використання штучного інтелекту для отримання значущих результатів. Натомість сучасний світ все більше й більше зацікавлений у таких технологіях як розпізнавання об’єктів [1].

Розпізнавання об'єктів – це ключова технологія, що стоїть за розвиненими системами допомоги водіям (ADAS), яка дозволяє автомобілям виявляти смуги руху або виявляти пішоходів для підвищення безпеки дорожнього руху. Розпізнавання об'єктів також важливе в таких сферах, як відеоспостереження [2], виявлення аномалій, підрахунок людей, самокеровані автомобілі, робототехніка, системи пошуку зображень або розпізнавання обличь.

Як правило, сучасні системи розпізнавання зображень є частиною хмарних сервісів [3, 4], доступ до яких є платним, а алгоритми обробки зображень розроблені для більш загальних випадків, що є незадовільним у більшості ситуацій. Дана предметна область є досить актуальною у таких галузях як медичні зображення, фізика, вимірювання та контроль якості в процесах виробництва, нейробіологія тощо. Саме тому розробка власного програмного модуля розпізнавання рухомих об’єктів є актуальною та доцільною.

Об’єкт дослідження – це процес розпізнавання рухомих об’єктів.

Предмет дослідження – програмні засоби розпізнавання рухомих об’єктів.

Мета дослідження – підвищення достовірності розпізнавання рухомих

об’єктів за рахунок використання попередньо натренованої згорткової нейронної мережі.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- проаналізувати відомі методи розпізнавання об’єктів на зображеннях та обрати напрямок досліджень;
- обгрунтувати вибір архітектури згорткової нейромережі;
- проаналізувати модель розпізнавання об’єктів на зображеннях на основі згорткової нейромережі;
- розробити алгоритм формування архітектури згорткової нейромережі;
- розробити програмне забезпечення розпізнавання об’єктів на зображеннях за допомогою нейронної мережі;
- провести тестування програмного модуля розпізнавання об’єктів.

1 ОБГРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ ПРОГРАМНОГО МОДУЛЯ РОЗПІЗНАВАННЯ РУХОМИХ ОБ’ЄКТІВ

1.1 Актуальність та галузі застосування розпізнавання об’єктів на зображеннях

У наш час приклади застосування комп’ютерного зору можна навести для будь-якої сфери. Дану галузь можна охарактеризувати як молоду, різноманітну та динамічну. І хоча існують більш ранні роботи, однак можна сказати, що тільки з кінця 1970-х почалося інтенсивне вивчення цієї проблеми, коли комп'ютери змогли управляти обробкою великих наборів даних, таких як зображення [5,6]. Однак ці дослідження зазвичай починалися з інших областей, і, отже, немає стандартної формулювання проблеми комп'ютерного зору. Також, і це навіть більш важливо, немає стандартного формулювання того, як повинна вирішуватися проблема комп’ютерного зору. Замість цього, існує маса методів для вирішення різних строго визначених завдань комп'ютерного зору, де методи часто залежать від завдань і рідко можуть бути узагальнені для широкого кола застосування. Багато з методів і додатків все ще знаходяться в стадії фундаментальних досліджень, але все більше число методів знаходить застосування в комерційних продуктах, де вони часто складають частину більшої системи, яка може вирішувати складні завдання (наприклад, в області медичних зображень [7] або вимірювання і контролю якості в процесах виробництва). У більшості практичних застосувань комп'ютерного зору комп'ютери попередньо запрограмовані для вирішення окремих завдань, але методи, засновані на знаннях, стають все більш загальними.

Одною з нових областей застосування комп’ютерного зору є автономні транспортні засоби, включаючи підводні, наземні (роботи, машини), повітряні. Рівень автономності змінюється від повністю автономних (безпілотних) до транспортних засобів, де системи, засновані на комп'ютерному зорі, підтримують водія або пілота в різних ситуаціях. Повністю автономні транспортні засоби використовують комп'ютерний зір для навігації, тобто для отримання інформації

про місце свого перебування, для створення карти навколишнього оточення, для виявлення перешкод. Вони також можуть бути використані для певних завдань, наприклад, для виявлення лісових пожеж. Прикладами таких систем можуть бути система попереджувальної сигналізації про перешкоди на машинах і системи автономної посадки літаків. Деякі виробники машин демонстрували системи автономного управління автомобілем, але ця технологія все ще не досягла того рівня, коли її можна запустити в масове виробництво [8]. Можливість ефективно обробити зображення та розпізнати на ньому пішохода чи тварину, які вибігли на дорогу – одне з головних завдань у цій сфері, тому вчені зі всього світу продовжують працювати над питанням швидкого та точного розпізнавання об’єктів у зображеннях.

Якщо потрібно розпізнавати рухомий об’єкт, то в програмі відеоаналітики вихідне відео читається кадр за кадром. Кожен кадр обробляється нейронною мережею. Це цикл, який продовжує працювати до кінця відео. Загальний принцип роботи системи відеоаналітики зображено на рисунку 1.1 [3].

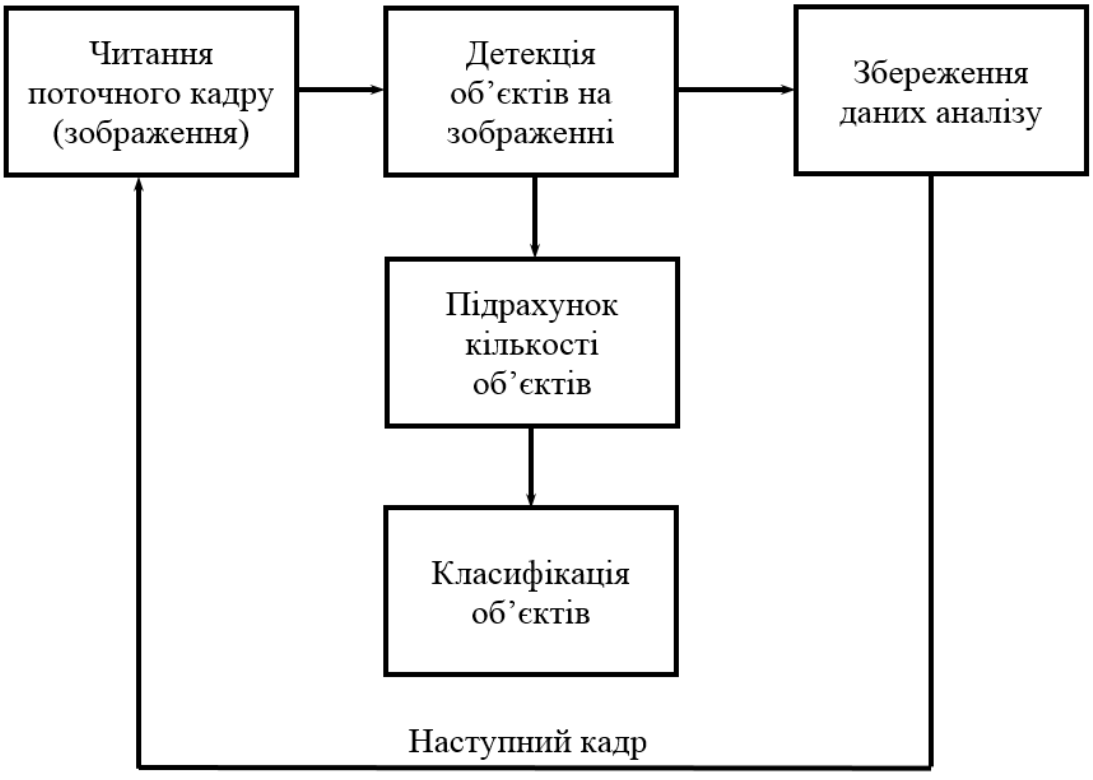

Рисунок 1.1 – Принцип роботи системи відеоаналітики

Для визначення об'єктів використовуються моделі на базі нейронних мереж типу CNN (convolutional neural network, згорткові нейронні мережі) та SSD (Single Shot MultiBox Detector). Ці моделі забезпечують високу точність та швидкість розпізнавання завдяки своїм архітектурним особливостям.

Згорткові нейронні мережі (CNN) є основою для багатьох сучасних систем розпізнавання об'єктів. Їх принцип роботи базується на використанні згорткових шарів, які виконують згортку (конволюцію) вхідних зображень з фільтрами (ядрами). Основні переваги CNN:

- CNN здатні автоматично виділяти важливі ознаки зображень, такі як контури, текстури та інші деталі.

- мережі навчаються від простих до складних ознак на різних рівнях. Перші шари виділяють низькорівневі ознаки (краї, кути), а останні шари — високорівневі (частини об'єктів, об'єкти);

- використання пулінг-шарів (наприклад, MaxPooling) дозволяє зменшити розмірність простору ознак, що знижує обчислювальні витрати і підвищує стійкість до зміщень та деформацій об'єктів.

Підхід SSD був опублікований наприкінці листопада 2016 року і привів до нового рекорду з точки зору продуктивності та точності для задач виявлення об'єктів.

SSD (Single Shot MultiBox Detector) є одним з ефективних методів для детекції об'єктів в реальному часі. Він поєднує в собі простоту та ефективність, використовуючи єдину згорткову мережу для виявлення об'єктів. Основні переваги SSD:

- єдине проходження (single shot) - SSD виконує як локалізацію, так і класифікацію об'єктів за одне проходження через мережу, що значно підвищує швидкість;

- мультискейл підхід використовує кілька згорткових шарів для прогнозування розташування об'єктів на різних масштабах, що покращує точність детекції дрібних та великих об'єктів;

- прямокутні межі (bounding boxes): кожен згортковий шар прогнозує серію

прямокутних меж (bounding boxes) та ймовірностей класів для цих меж, що забезпечує високу точність;

- Detector – нейронна мережа працює як класифікатор (детектор) об'єктів.

Таким чином, ми можемо класифікувати за 1 прохід кілька різних об'єктів і на одному кадрі визначати і людей і тварин і транспорт тощо.

Програма підрахунку об'єктів на відео, наприклад, може вважати різні типи об'єктів та враховувати їхнє знаходження у певній зоні.

Нейронна мережа також здатна “супроводжувати” об'єкти, таким чином визначивши на одному з кадрів машину і вона буде цією ж самою машиною і на інших кадрах.

Крім згаданих нейронних мереж архітектури CNN або SSD для відеоаналітики застосовуються ще інші моделі, залежно від поставленого завдання.

1.2 Огляд відомих методів розв’язання задачі

На сьогоднішній день існує велика кількість методів (алгоритмів) інтелектуальних обчислень для розв’язання задач у сфері комп’ютерного зору, зокрема [9]:

- нейронні мережі;
- дерева рішень;
- системи розмірковувань на основі аналогічних випадків;
- алгоритми визначення асоціацій і послідовностей;
- нечітка логіка;
- генетичні алгоритми.

Нейронна мережа [10] – це комп’ютерна система, яка за принципом своєї роботи схожа на біологічну нейронну мережу, тобто мозок. Така мережа складається з великої кількості вузлів, які є аналогами нейронів у мозку. Вузли поєднуються між собою штучним аналогом синапсів, що дозволяє їм обмінюватися інформаціє один між одним. Такі системи «навчаються» виконувати поставлену задачу на основі навчальних даних. Корекція результату відбувається за рахунок корекції ваг

ребер, які з’єднують нейрони. Нейроні мережі реалізують непрозорий процес, тобто побудована модель, як правило, не має чіткої інтерпретації.

Дерева рішень – цей метод широко застосовується в різних областях виробництва, фінансів і бізнесу, де часто зустрічаються задачі числового прогнозу. У результаті застосування цього методу, для навчальної вибірки даних створюється ієрархічна структура правил класифікації типу, «ЯКЩО... ТОДІ...», що мають вид дерева. Для того щоб вирішити, до якого класу віднести деякий об’єкт або ситуацію, треба відповісти на запитання, що стоїть у вузлах цього дерева, починаючи з його кореня. Питання можуть мати вигляд «Значення параметра А більше Х?» або «Значення змінної В належить підмножині ознак С?». Якщо відповідь позитивна, перехід до правого вузла наступного рівня, якщо негативна – то до лівого вузла; потім знову здійснюється процедура відповіді на питання, яке зв'язане з відповідним вузлом. Таким чином, зрештою, можна дійти до одного з кінцевих вузлів, де визначений клас об'єкта.

Системи розмірковувань на основі аналогічних випадків працюють таким чином, що для того, щоб зробити прогноз на майбутнє або вибрати правильне рішення, системи знаходять у минулому близькі аналоги наявної ситуації і вибирають ту ж відповідь, що і була для них правильною. Тому цей метод ще називають методом «найближчого сусіда».

Системи розмірковувань на основі аналогічних випадків дають гарні результати в різних задачах. Головний їхній недолік полягає в тому, що вони не створюють яких-небудь моделей або правил, що узагальнюють попередній досвід, – у виборі рішення вони базуються на всьому масиві доступних історичних даних, тому неможливо сказати, на основі яких конкретно факторів ці системи будують свої відповіді.

Алгоритми визначення асоціацій знаходять правила про окремі предмети, що з’являються разом в одній транзакції, наприклад, в одній покупці.

Нечітка логіка застосовується для масивів даних, у яких приналежність даних до якої-небудь групи є ймовірністю в інтервалі від 0 до 1. Чітка логіка маніпулює результатами, що можуть бути або істиною або неправдою. Нечітка логіка

застосовується в тих випадках, коли існує «може бути» у доповненні до «так» або «ні».

Генетичні алгоритми є могутнім засобом рішення різних комбінаторних задач і задач оптимізації. Особливістю генетичного алгоритму є акцент на використання оператора «схрещення», який виконує операцію рекомбінацію рішень-кандидатів, роль якої аналогічна ролі схрещення в живій природі. У результаті послідовної зміни поколінь виробляється рішення поставленої задачі, що уже не може бути далі поліпшено.

1.3 Обгрунтування вибору аналогу до програмного модуля розпізнавання рухомих об’єктів

На сьогоднішній день можна виділити декілька відомих програмних реалізацій, що здатні здійснювати розпізнавання об’єктів на зображеннях. Ці програмні рішення використовують передові технології машинного навчання та комп'ютерного зору, забезпечуючи високу точність і продуктивність. Серед найбільш популярних та ефективних програм можна відзначити наступні.

TensorFlow Object Detection API є одним з найпопулярніших інструментів для розпізнавання об'єктів, розроблених командою Google Brain. Ця бібліотека надає зручні інтерфейси для створення, тренування та використання моделей для детекції об'єктів.

Основні особливості:

- підтримка різних моделей: API підтримує різні моделі для детекції об'єктів, включаючи SSD, Faster R-CNN та EfficientDet;

- завдяки використанню передових архітектур нейронних мереж, TensorFlow Object Detection API забезпечує високу точність розпізнавання об'єктів.

- можливість налаштування моделей під конкретні завдання та використання власних даних для навчання.

YOLO (You Only Look Once) є однією з найшвидших та найефективніших моделей для детекції об'єктів, розробленою Джозефом Редмоном. Ця модель

відзначається своєю здатністю виконувати розпізнавання об'єктів в реальному часі.

Основні особливості:

- YOLO здатна обробляти до 45 кадрів в секунду на звичайному апаратному забезпеченні;

- однопрохідна обробка: модель виконує всі обчислення за одне проходження, що значно підвищує швидкість роботи;

- використовується в різних сферах, від відеоспостереження до автономного керування транспортними засобами.

OpenCV (Open Source Computer Vision Library) є однією з найстаріших та найвідоміших бібліотек для комп'ютерного зору. Вона містить широкий набір інструментів для обробки зображень та відео, включаючи засоби для розпізнавання об'єктів.

Основні особливості:

- підтримка різноманітних алгоритмів для обробки зображень, включаючи детекцію об'єктів, розпізнавання облич, трекінг та багато іншого;

- підтримка багатьох мов програмування, включаючи Python, C++, Java та інші;

- завдяки своїй гнучкості та потужності, OpenCV широко використовується в академічних та комерційних проектах.

Detectron2 є продовженням популярного фреймворку Detectron, розробленого Facebook AI Research (FAIR). Це сучасний інструмент для детекції та сегментації об'єктів.

Основні особливості:

- легко налаштовується та розширюється, що дозволяє створювати кастомізовані рішення для різних завдань.

- оптимізовані алгоритми забезпечують високу швидкість і точність;

- підтримка спільноти та активний розвиток проекту сприяють постійному вдосконаленню інструменту.

Microsoft Azure Custom Vision є частиною платформи Azure AI і надає хмарні сервіси для навчання та розгортання моделей розпізнавання об'єктів.

Основні особливості:

- простота використання - інтуїтивний інтерфейс дозволяє легко навчати та розгортати моделі без глибоких знань у сфері машинного навчання;

- можливість інтеграції з іншими хмарними сервісами для створення комплексних рішень;

- здатність обробляти великі обсяги даних та підтримка високого навантаження.

Сучасні програмні реалізації для розпізнавання об'єктів на зображеннях забезпечують високу точність, швидкість та гнучкість, що робить їх незамінними інструментами в різних галузях. Використання передових технологій машинного навчання та комп'ютерного зору дозволяє вирішувати складні задачі та відкриває нові можливості для автоматизації та аналізу даних.

До таких програм можна віднести BytePace [11], що є програмним інструментом на мові Python. Спочатку алгоритм був розроблений для виявлення осіб на зображеннях, але його можна натренувати на виявлення інших об'єктів. Для своєї роботи він використовує розбиття (splitting) зображення на області, оцінку яскравості в цих областях і відсікання областей, де класифікований об'єкт однозначно не знаходиться. З цього випливає головний недолік системи – її низька точність. Це може призводити до помилок в роботі (рис. 1.2).

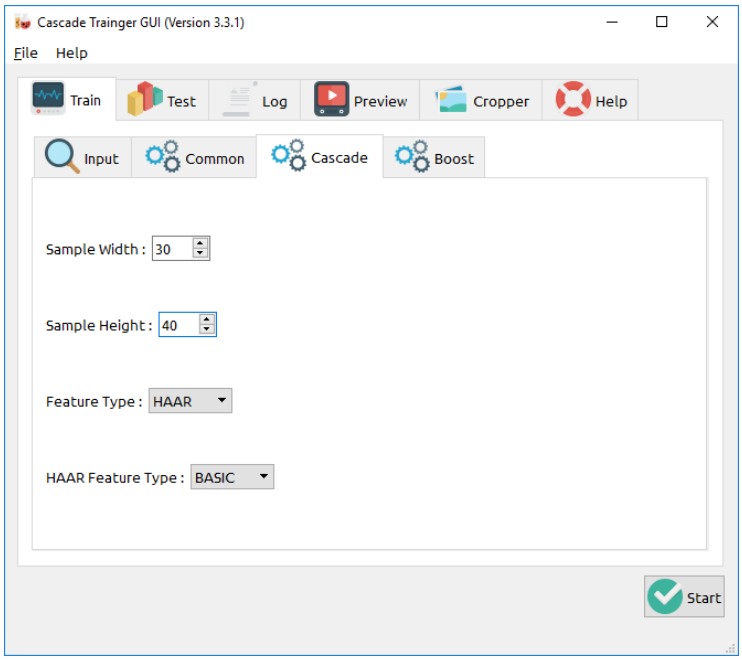

Рисунок 1.2 – Вікно програми BytePace

Ще однією реалізацією для розпізнавання об’єктів на зображеннях є програма Inception [12]. Принцип роботи цієї програми полягає у відкиданні зайвих векторів, що дозволяє підвищити ефективність та точність розпізнавання. Зайві вектори – це ті, які не ввійшли у взаємно знайдений кластер векторів. Наприклад, на одному зразку може бути тінь, яка розпізнається як межа об'єкта, а на наступному зразку цієї тіні може не бути, що ускладнює процес розпізнавання. Програма Inception здатна відфільтровувати подібні артефакти, що дозволяє зосередитися на справжніх межах об'єктів (рис. 1.3).

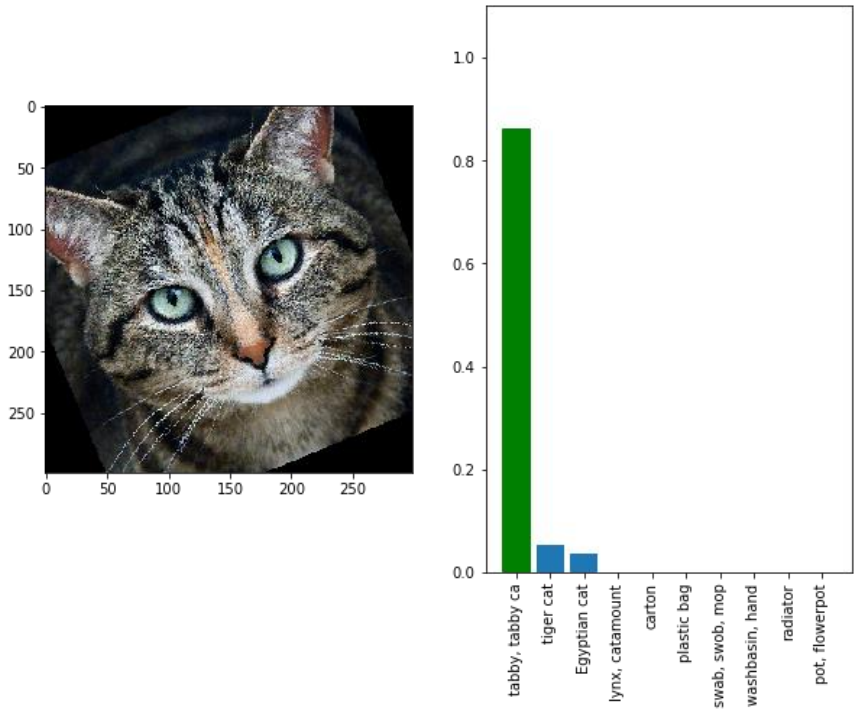

Рисунок 1.3 – Вікно програми Inception

Inception використовує складні алгоритми глибокого навчання та нейронних мереж для аналізу зображень. Однією з ключових особливостей є можливість масштабування та обробки зображень різної складності, що дозволяє програмі застосовуватися в різних галузях, таких як медична діагностика, автономне керування транспортними засобами, розпізнавання облич та інших.

Проте, не зважаючи на всі переваги, існують і певні недоліки використання

Inception. Одним із них є складність роботи з програмою. Налаштування та оптимізація моделі вимагають значних знань у галузі машинного навчання та програмування. Крім того, навчання моделі може займати багато часу та ресурсів, що може бути недоступним для менш обладнаних систем або менших організацій.

Іншим важливим недоліком є відносно низька достовірність роботи програми в деяких випадках. Це може бути пов'язано з тим, що Inception, як і будь-яка інша модель машинного навчання, сильно залежить від якості та кількості даних, на яких вона навчалася. Недостатня кількість навчальних зразків або їх низька якість можуть призвести до неправильної класифікації або невірного розпізнавання об'єктів.

Отже, хоча Inception представляє собою потужний інструмент для розпізнавання об'єктів на зображеннях, його використання потребує певних знань та ресурсів, а також може бути обмеженим у певних умовах через потенційну низьку достовірність результатів [12].

Отже, розглянуті програмні реалізації мають ряд недоліків таких як складність в освоєнні, низька швидкість та точність. З огляду на це, постає питання в розробці нового програмного засобу для класифікації зображень, який був би вільний від вказаних недоліків.

1.4 Постановка задачі

Для розробки програмного модуля розпізнавання рухомих об’єктів, було проаналізовано, з використанням літератури, особливості поставленої задачі з точки зору вимог до її розв’язання.

Для того, щоб програма працювала стабільно і підлягала масштабуванню, необхідно, щоб її внутрішня архітектура була простою та поділялася на прості модулі, які можна додавати або вилучати. Виконувані функції кожного програмного модуля повинні бути зручними для розширення та відлагодження.

Важливим етапом розробки програми є визначення досяжних технічних параметрів. У таблиці 1.1 відображена рекомендована конфігурація персонального комп’ютера для запуску програмного модуля.

Таблиця 1.1 – Рекомендована конфігурація

Тип процесора	64-розрядний (x64) процесор з тактовою частотою 3,2 ГГц
Об’єм оперативної пам’яті	16 ГБ для 64-разрядної системи
Місце на жорсткому диску	1 ГБ
Графічний пристрій	Графічний прискорювач, сумісний з DirectX11 та об’ємом відеопам’яті 4 ГБ
Операційна система	Windows 10

Отже, визначені вимоги до вхідних та вихідних даних, архітектури та виконуваних функцій програми, а також досяжних технічних параметрів. Виконання саме цих вимог дозволить розробити якісний та корисний програмний продукт.

1.5 Висновок

У першому розділі було розглянуто стан питання розпізнавання рухомих об’єктів. Проведено огляд відомих методів для роботи з інтелектуальними обчисленнями у сфері комп’ютерного зору. В ході аналізу предметної області розглянуто основні методи класифікації зображень та як найбільш перспективний, було обрано нейромережевий метод. Також було здійснено аналіз відомих програмних засобів розпізнавання рухомих об’єктів та як аналог до розроблюваного модуля було обрано програму Inception.

2 РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ РОЗПІЗНАВАННЯ РУХОМИХ ОБ'ЄКТІВ

2.1 Обґрунтування обраного методу розв'язання задачі

Проаналізувавши відомі методи для роботи з інтелектуальними обчисленнями у сфері комп'ютерного зору, було прийнято рішення використовувати нейронні мережі для вирішення задачі, адже вони є досить швидкими та точними в своїх обчисленнях. Загалом існує багато різновидів штучних нейронних мереж. Для поставленої задачі найбільше підходять такі 2 групи нейромереж:

- Згорткові нейронні мережі (з англ. Convolutional Neural Network або CNN);
- Рекурентні нейронні мережі (з англ. Recurrent Neural Network або RNN).

Згорткові нейронні мережі – це клас глибинних нейронних мереж, які широко використовуються у задачах обробки зображень [13]. CNN складаються з трьох типів шарів:

- convolutional layer (шар згортки);
- pooling layer (шар об'єднання);
- fully-connected layer (шар повного об'єднання).

Шар згортки аналізує вхідні дані за допомогою набору фільтрів. У результаті аналізу отримуємо набір особливостей, які подані у вигляді n-мірного масиву. Далі цей масив направляється у шар об'єднання, який зменшує кількість особливостей шляхом певних операцій над масивом. У результаті матимемо зжатий набір особливостей. Це необхідно для того, щоб пришвидшити роботу алгоритму. Послідовність із шарів згортки та об'єднання повторюється декілька раз, у залежності від виду нейронної мережі. Після цього остаточний набір властивостей надсилається до шару повного об'єднання. На цьому етапі проводиться аналіз усіх властивостей, отриманих у ході роботи нейронної мережі та визначається фінальний результат.

Рекурентні нейронні мережі – це клас нейронних мереж які дозволяють аналізувати вхідні дані в контексті попередніх або наступних даних[14]. Такий вид

нейронних мереж є особливо ефективним при аналізі зв’язаних послідовностей, таких як текст, відео, аудіо, тощо.

У рекурентних нейронних мережах вхідні дані обробляються в одному шарі. Це дозволяє будувати нейронні мережі з різним відношенням вхідних даних до вихідних. Дані у рекурентній нейронній мережі обробляються послідовно. Саме це й дозволяє проводити аналіз поточних даних у контексті попередніх. Хоча, у такого підходу є недолік, адже для аналізу наступного блоку даних необхідно очікувати результат аналізу попереднього, що обмежує можливість використання.

Хоча обидва види нейронних мереж можна використати для вирішення поставленої задачі розпізнавання об’єктів у зображеннях, але популярними підходами, що застосовуються у наш час, є використання згорткових нейронних мереж (CNN), таких як R-CNN та YOLOv2 [15], які автоматично вчаться виявляти об'єкти в зображеннях. У свою чергу, такі нейронні мережі розпізнавання об’єктів бувають двох типів – двоступеневі та одноступеневі.

Початковий етап двоступеневих мереж, таких як R-CNN, ідентифікує підмножини регіонів зображення, які можуть містити об’єкт. На другому етапі класифікуються об’єкти регіонів. Двоступеневі мережі можуть досягати дуже точних результатів розпізнавання об’єктів, однак вони, як правило, повільніше, ніж одноступеневі мережі.

У одноступеневих мережах, таких як YOLOv2, CNN виробляє мережеві передбачення для регіонів у всьому зображенні за допомогою обмежуючих прямокутників, і потім прогнози декодуються для створення остаточних обмежуючих прямокутників для об’єктів [16]. Одноступеневі мережі можуть бути набагато швидшими, ніж двоступеневі, але вони можуть не досягати однакового рівня точності, особливо для зображень, що містять невеликі об’єкти.

Нині існує два підходи, щоб розпочати роботу над розпізнаванням об’єктів у зображеннях за допомогою нейромережі.

З однієї сторони, можна почати працювати з нейронною мережею «з нуля», тобто створити та натренувати власний розпізнавач об’єктів. Для цього необхідно розробити архітектуру мережі, а також зібрати величезний об’єм тренувальних

даних, на яких власна мережа буде вчитися розпізнавати ті чи інші об’єкти. Звичайно, результати використання такої мережі можуть бути чудовими, однак з огляду на те, що потрібно вручну налаштовувати шари та ваги нейронної мережі, затрати часу на налаштування декількох мільйонів параметрів, а також на власне тренування мережі (сотні годин роботи GPU) будуть просто величезними.

Тому з іншої сторони, існує варіант використання попередньо натренованої нейронної мережі. Цей метод може забезпечити більш швидкі результати, оскільки розпізнавачі об'єктів вже пройшли навчання на тисячах, а то й мільйонах зображень, і все що залишається – точно налаштувати таку мережу для власного додатку, що не потребує великих обсягів обчислювальних ресурсів.

Отже, провівши аналіз відомих методів для роботи з інтелектуальними обчисленнями у сфері комп’ютерного зору, для розробки програмного модуля розпізнавання об’єктів у зображеннях було обрано алгоритм, який базується на використанні попередньо натренованої згорткової нейронної мережі YOLOv2, яка, порівняно зі своїми аналогами [16], є досить швидкою та точною в обчисленнях.

2.2 Структура та порядок функціонування згорткової нейромережі

YOLOv2 – нейромережа з одноступеневою архітектурою, яка складається з 19 шарів згортки та 5 шарів об’єднання. Вона розглядає ціле зображення під час розпізнавання, тому її вхідний шар приймає тензор 1 x 3(RGB) x 416px x 416px. Мережа приймає вхідні дані і передає їх через різні шари для створення вихідних даних (рис. 2.1).

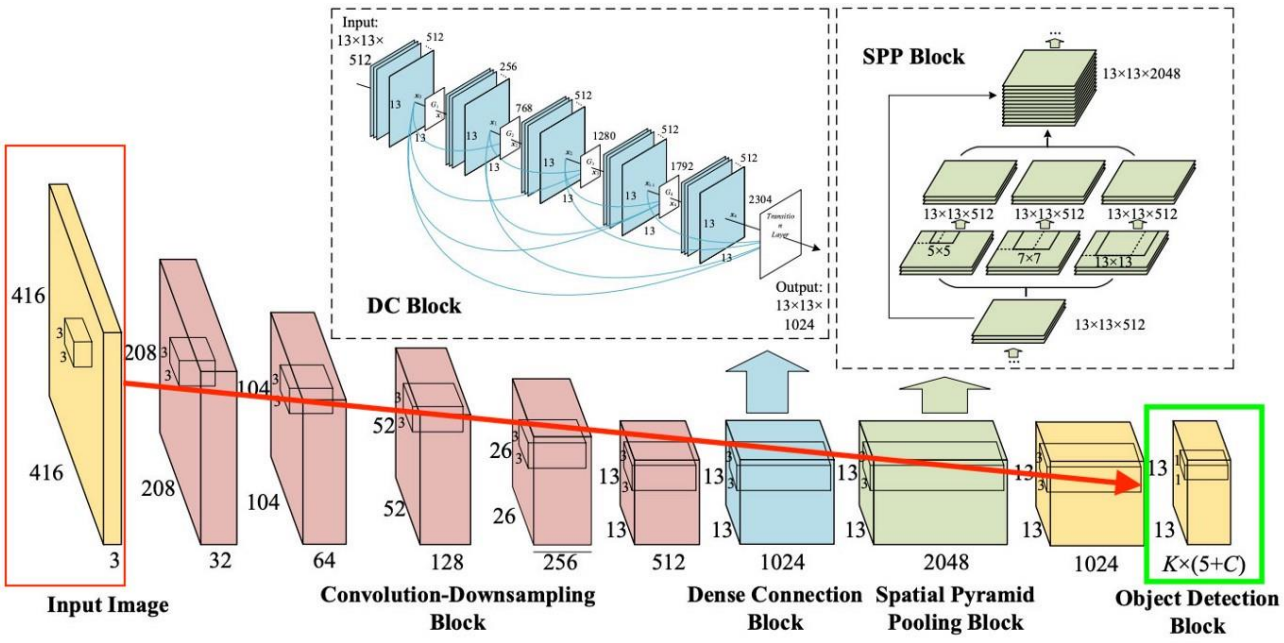

Рисунок 2.1 – Структура нейромережі YOLOv2

Вихідний шар має розмірність 1 x 13 x 13 x (k x (1 + 4 + 80)), де 13 x 13 – кількість комірок сітки, на яке поділяється зображення; k – кількість полів прив`язки (anchor boxes); 80 – кількість класів об’єктів, які розпізнає нейромережа. Оскільки авторами мережі було визначено значення k = 5 як найбільш оптимальне, то вихідний тензор має розмірність 1 x 13 x 13 x 425 (рис. 2.2).

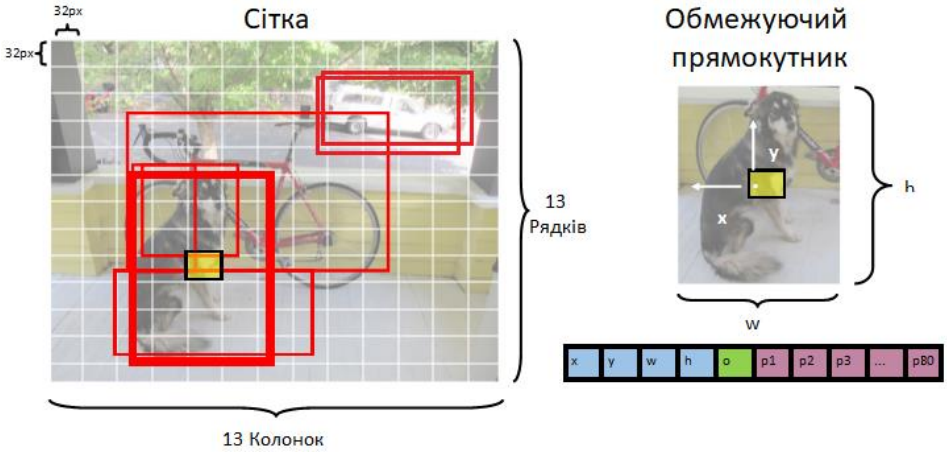

Рисунок 2.2 – Графічне відображення вихідних даних нейронної мережі YOLOv2

Отже, на виході, для кожної комірки сітки (32px x 32px) міститься 425 параметрів (рис. 2.2):

- x – координата по осі X для центру обмежуючого прямокутника щодо комірки сітки, з якою він пов’язаний;
- y - координата по осі Y для центру обмежуючого прямокутника щодо комірки сітки, з якою він пов’язаний;
- w – ширина обмежуючого прямокутника;
- h – висота обмежуючого прямокутника;
- o – значення достовірності того, що об’єкт існує в межах обмежуючого прямокутника, також відоме як оцінка присутності об’єкта (від 0 до 1);
- p1-p80 – ймовірності для кожного з 80 класів об’єктів, прогнозованих нейромережею (від 0 до 1).

2.3 Розробка алгоритму роботи програмного модуля розпізнавання об’єктів

Щоб розробити програмний модуль розпізнавання об’єктів у зображеннях, необхідно розробити його загальний алгоритм роботи. Схема алгоритму зображена на рисунку 2.3.

Загальний алгоритм роботи програми має такі кроки: завантаження попередньо натренованої нейронної мережі, отримання вхідних даних у вигляді цифрового зображення користувача, обробка вхідних даних шляхом пошуку та розпізнавання об’єктів за допомогою нейромережі, обробка масиву вихідних даних, які створила мережа, фільтрація вихідних даних, створення зображення з обмежувальними прямокутниками, з назвами розпізнаних класів та достовірністю їх розпізнання.

Схема алгоритму роботи програми складається з операцій, що розташовані у певній пронумерованій послідовності, при виконанні яких наведено конкретний результат у форматі визначених процесів. Програма працює до її закриття користувачем.

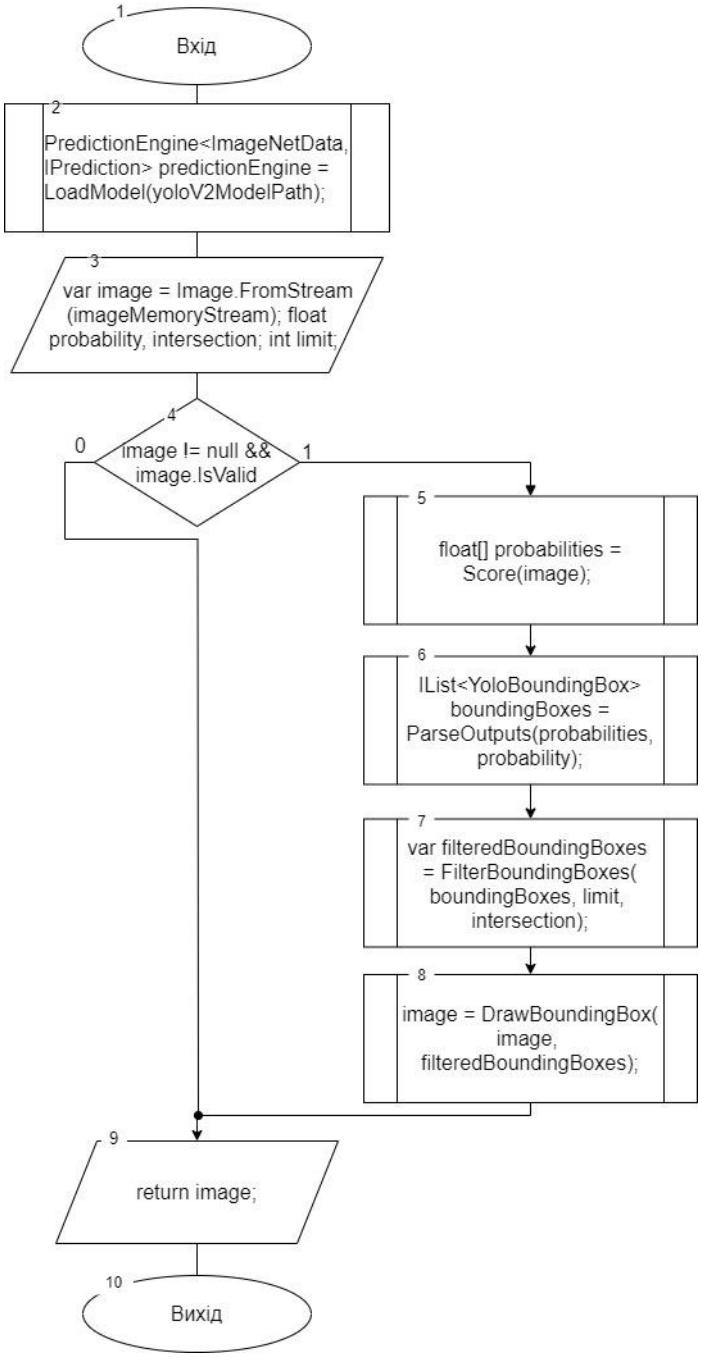

Рисунок 2.3 – Схема загального алгоритму роботи програми

2.4 Розробка структури програмного забезпечення

Для того, щоб програма працювала стабільно і підлягала масштабуванню, необхідно, щоб її структура була простою та поділялася на прості модулі, які можна додавати або вилучати. Структура програми наведена на рисунку 2.4.

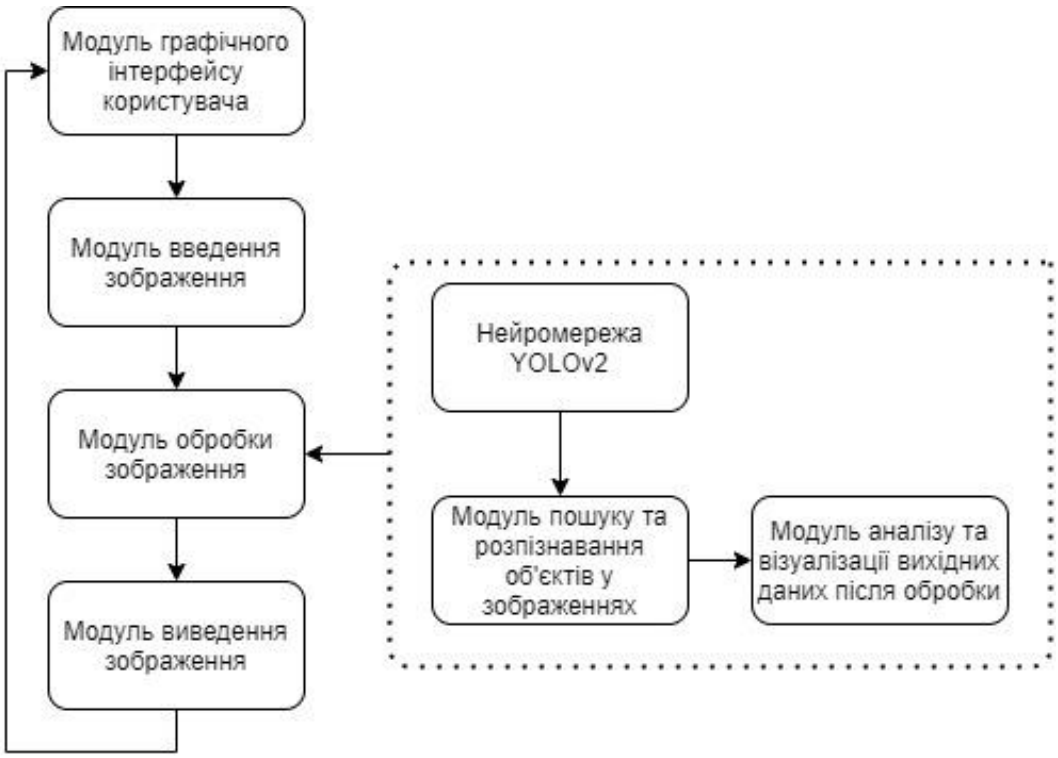

Рисунок 2.4 – Структура програмного забезпечення розпізнавання об’єктів

Як видно з рисунку 2.4, графічний інтерфейс користувача пов’язаний з модулем введення та виведення зображення і відповідає за відображення форми завантаження нового зображення та відображення результуючого після розпізнавання зображення користувача. Модуль введення контролює отримання вхідного зображення допустимого формату (.jpg, .jpeg, .png.) з оригінального відеоряду, приводить його до розміру (416x416), який приймає обрана нейромережа та передає його у модуль обробки.

Модуль обробки зображення виконує пошук і розпізнавання об’єктів за допомогою нейронної мережі, що забезпечує високу точність та швидкість аналізу зображень. Основні етапи роботи модуля включають в себе попередню обробку зображень, використання навченої моделі для ідентифікації об'єктів, а також аналіз і візуалізацію вихідних даних після обробки.

На етапі попередньої обробки зображення піддаються різноманітним трансформаціям, таким як масштабування, нормалізація кольорів, видалення шумів тощо. Це дозволяє покращити якість зображень та підготувати їх до подальшого аналізу нейронною мережею.

Наступний етап полягає у використанні нейронної мережі, яка виконує пошук і розпізнавання об'єктів на зображеннях. Нейронна мережа, зазвичай, складається з кількох шарів, включаючи згорткові шари, які виконують виділення ознак, та повнозв'язні шари, які відповідають за класифікацію. Під час обробки зображень модель аналізує кожен піксель та визначає, чи належить він до певного об'єкта, використовуючи попередньо навчені вектори ознак.

Після розпізнавання об'єктів, результати аналізу проходять етап візуалізації. Модуль створює кінцеве зображення, на якому виділяє знайдені об'єкти, їхні класи, а також вказує назву кожного об'єкта та рівень достовірності його розпізнавання. Це дозволяє користувачеві легко інтерпретувати результати роботи модулю та отримати уявлення про виявлені об'єкти на зображенні.

Результуюче зображення повертається користувачеві через графічний інтерфейс. Це зображення може включати різні графічні елементи, такі як прямокутники, які окреслюють об'єкти, підписи з назвами об'єктів, а також значення достовірності, що відображають впевненість моделі в правильності розпізнавання. Графічний інтерфейс забезпечує зручний спосіб взаємодії з користувачем, дозволяючи переглядати, зберігати або подальше використовувати результати обробки зображення (рис. 2.5).

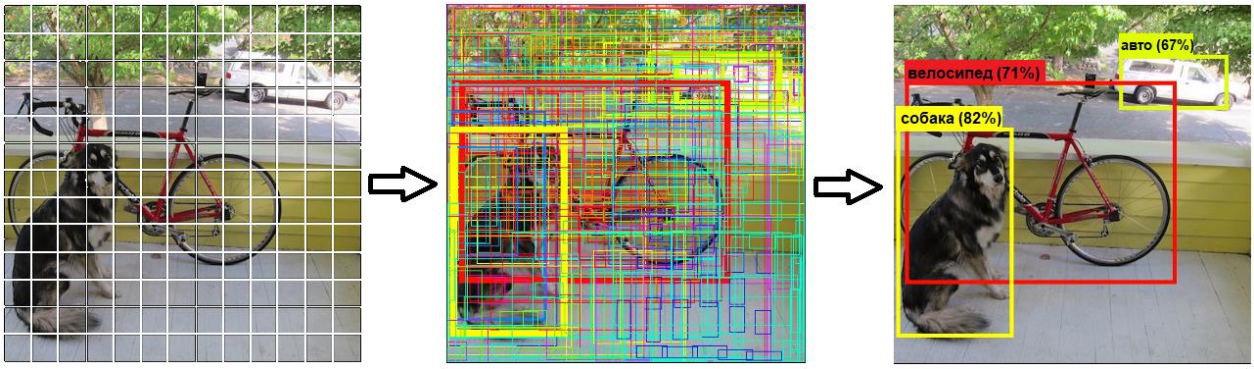

Рисунок 2.5 – Графічне відображення початкових, проміжних та кінцевих даних обробки зображення

На етапі проектування було також створено діаграму класів програмного забезпечення розпізнавання об’єктів, наведену на рисунку 2.6.

Рисунок 2.6 – Діаграма класів програмного забезпечення

Діаграма класів є ключовим компонентом у процесі об'єктно-орієнтованого проєктування, оскільки вона відображає структуру системи, визначає її основні компоненти, їх взаємозв'язки та взаємодію.

Діаграма класів для програмного забезпечення розпізнавання об’єктів включає декілька основних елементів:

- **Клас ImageProcessor** - відповідає за завантаження, попередню обробку та зберігання зображень. Цей клас може мати методи для масштабування, нормалізації, фільтрації та інших трансформацій зображень, необхідних для підготовки до аналізу;

- **Клас NeuralNetwork** - реалізує основну логіку нейронної мережі, яка виконує розпізнавання об’єктів. Він включає шари мережі (згорткові, об'єднувальні, повнозв'язні), методи для навчання, передбачення та оцінки якості моделі;

- **Клас ObjectDetector** - використовує результати роботи нейронної мережі для виявлення об’єктів на зображенні. Він містить методи для визначення координат об’єктів, їх класифікації та оцінки достовірності розпізнавання;

- **Клас Visualization** - відповідає за візуалізацію результатів розпізнавання. Він містить методи для створення графічних елементів, таких як прямокутники навколо об'єктів, підписи з назвами об'єктів та значення достовірності;

- **Клас UserInterface** - забезпечує взаємодію з користувачем. Він включає методи для завантаження зображень, відображення результатів та надання користувачеві можливостей для збереження або подальшого аналізу результатів;

- **Клас Logger** - відповідає за ведення журналу подій та помилок, що можуть виникати під час роботи програмного забезпечення. Це дозволяє відслідковувати роботу системи та проводити діагностику у разі виникнення проблем.

2.5 Висновок

У розділі було обґрунтовано доцільність використання для розпізнавання рухомих об’єтів згорткової нейронної мережі, а саме – попередньо натренованої нейронної мережі YOLOv2. Була проаналізована структура та порядок функціонування згорткової нейромережі YOLOv2. Також було розроблено алгоритм роботи програми та структуру програмного змодуля розпізнавання рухомих об’єктів на зображеннях, отриматих з відеофрагментів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ РОЗПІЗНАВАННЯ РУХОМИХ ОБ'ЄКТІВ

3.1 Варіантний аналіз вибору засобів розробки

Вибір правильних технології для реалізації будь-якого програмного додатку є важливим етапом процесу розробки, адже правильна технологія дозволить значно полегшити процес реалізації. Розглянемо інтегровані середовища розробки (англ. Integrated development environment – IDE) для ефективної розробки програмного модуля. Для вибору середовища програмування розглянуто наступні IDE [17]:

1. Eclipse.
2. Qt Creator.
3. Microsoft Visual Studio.

Eclipse – безкоштовне модульне інтегроване середовище розробки програмного забезпечення [18]. Розробляється і підтримується Eclipse Foundation і включає проекти, такі як платформа Eclipse, набір інструментів для розробників на мові Java, візуальні конструктори GUI тощо. Написаний в основному на Java, може бути використаний для розробки додатки на Java і, за допомогою різних плаґінів, на інших мовах програмування, включаючи Ada, C, C++, COBOL, Fortran, Perl, PHP, Python, R, Ruby (включно з каркасом Ruby on Rails), Scala, Clojure та Scheme. Розробник – Eclipse Foundation., ліцензування – безкоштовне, поточна версія – 4.8.

Eclipse являє собою фреймворк із низкою особливостей:

- можливість розробки програмного забезпечення на багатьох мовах програмування (рідною є Java);
- крос-платформна;
- модульна, призначена для подальшого розширення незалежним розробниками.

Інтерфейс середовища розробки Eclipse наведено на рисунку 3.1.

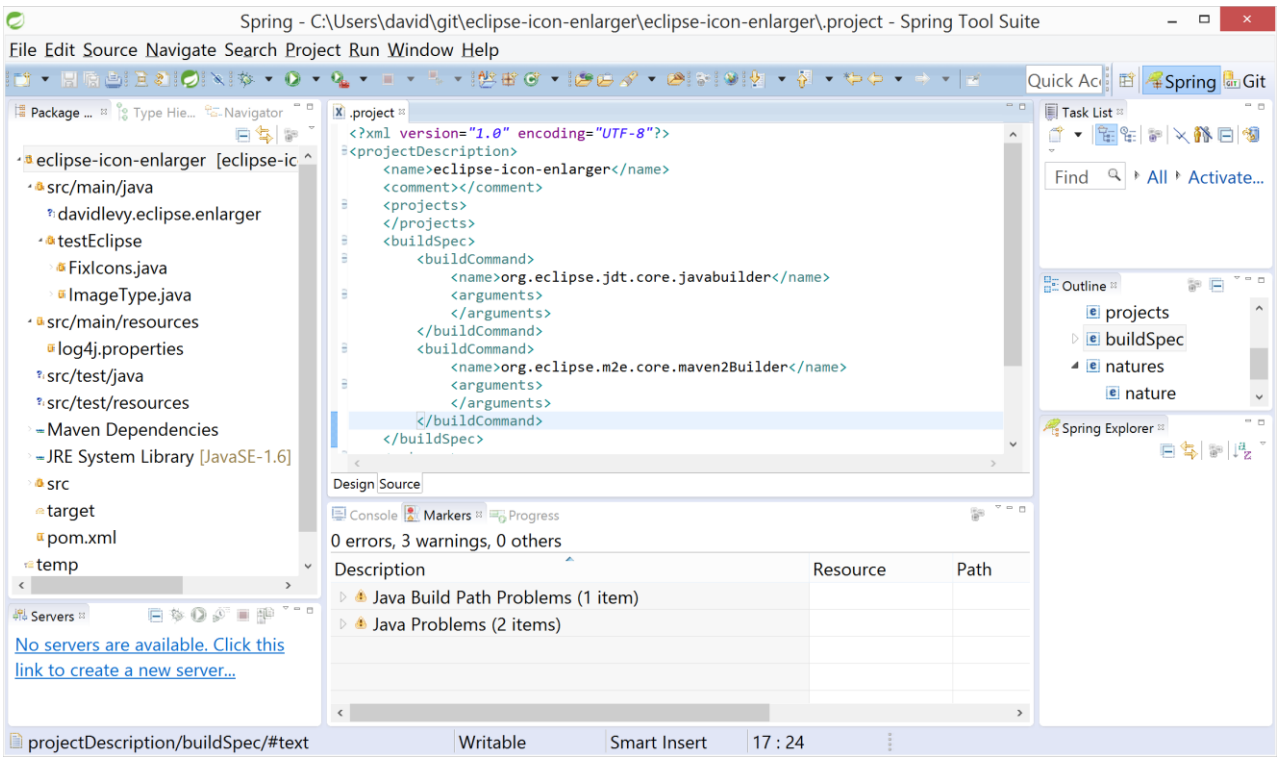

Рисунок 3.1 – Інтерфейс середовища розробки Eclipse

Qt Creator – безкоштовний крос-платформний інструментарій розробки програмного забезпечення мовою програмування C++[19]. Дозволяє запускати написане за його допомогою ПЗ на більшості сучасних операційних систем, просто компілюючи текст програми для кожної операційної системи без зміни коду. Містить всі основні класи, які можуть бути потрібні для розробки прикладного програмного забезпечення, починаючи з елементів графічного інтерфейсу й закінчуючи класами для роботи з мережею, базами даних, OpenGL, SVG і XML. Програма створена компанією Qt Development Frameworks, поточна версія – 4.7.0.

Qt Creator також може бути використаним в багатьох інших мовах програмування: Ada (QtAda), C# (Qyoto/Kimono), Java (Qt Jambi), Qt Jambi, Pascal, Perl, PHP (PHP-Qt), Ruby (QtRuby), та Python (PyQt,PySide).

Фрагмент роботи програми Qt Creator наведено на рисунку 3.2.

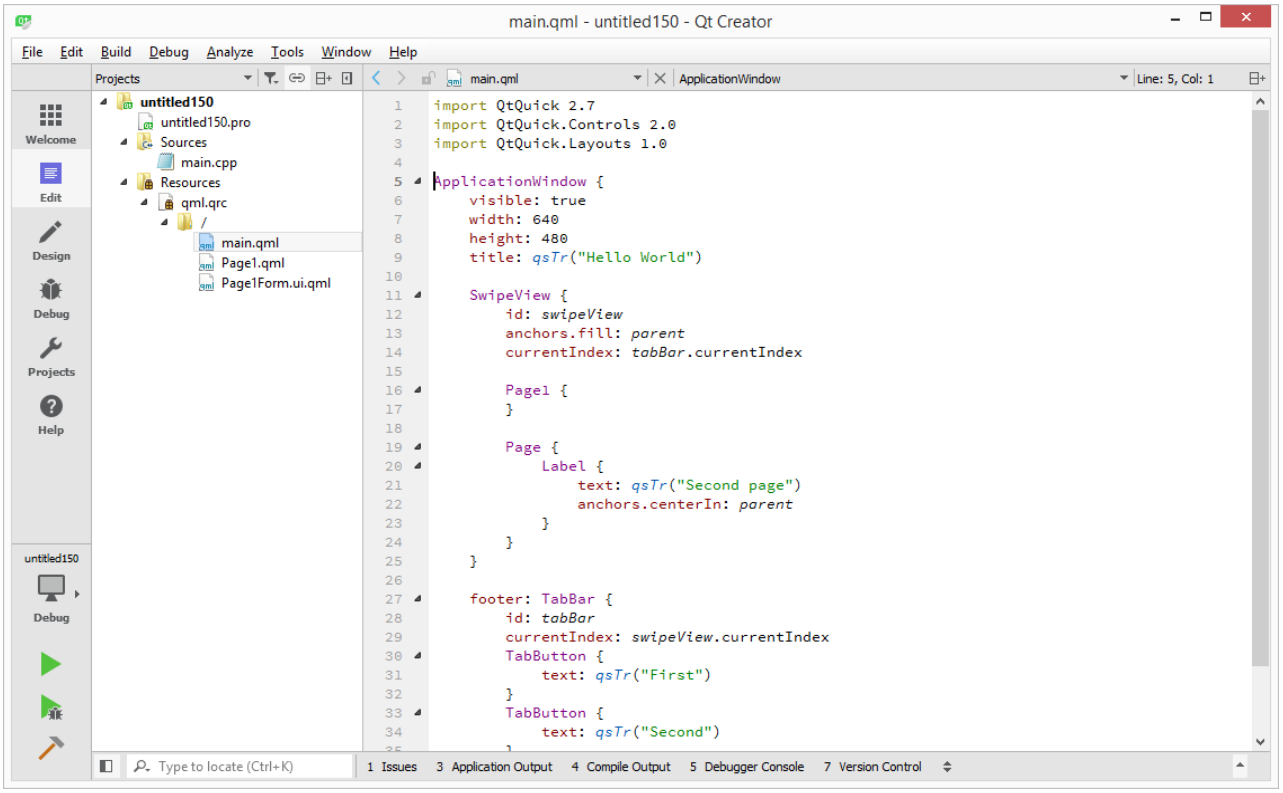

Рисунок 3.2 – Інтерфейс середовища розробки Qt Creator

Visual Studio 2019 – кросплатформне середовище розробки від корпорації Microsoft, що дає змогу розробляти як консольні й графічні додатки для настільних комп'ютерів, так і веб-сайти, веб-додатки, веб-служби, а також мобільні додатки [20]. Visual Studio дає змогу розробляти графічний інтерфейс додатків за допомогою фреймворків Windows Forms та WCF (Windows Presentation Foundation). MSVS має три збірки для розробників з різним професійним рівнем, а саме: Community, Professional, Enterprise, що містять можливості створення діаграм класів та функції юніт-тестування коду. Visual Studio підтримує 36 різних мов програмування, серед яких найбільш використовуваними є: C, C++, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, HTML та CSS. Програма створена компанією Microsoft, поточна версія – 16.6.0.

Інтерфейс середовища розробки Visual Studio 2019 Professional наведено на рисунку 3.3.

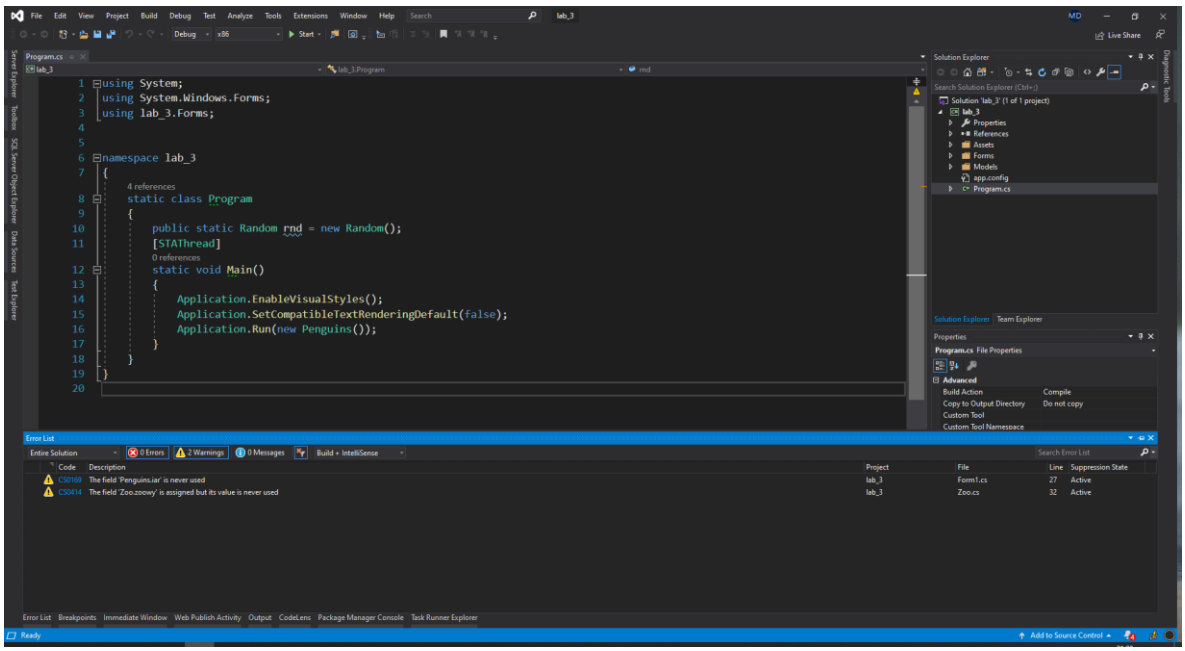

Рисунок 3.3 – Інтерфейс середовища розробки Visual Studio 2019 Professional

Аби вибрати, яке саме інтегроване середовище розробки найбільше підходить для реалізації модуля розпізнавання об’єктів у зображеннях, було складено таблицю 3.1 з порівнянням вище описаних IDE:

Таблиця 3.1 – Порівняння інструментальних засобів розробки

	Eclipse	Qt Creator	Visual Studio
Підтримка роздільних вікон для декількох моніторів	+	-	+
Оптимізація коду	-	+	+
Статичний аналіз коду	-	-	+
Вбудована підтримка системи контролю версій Git	-	-	+
Вбудовані засоби для створення та тренування нейронних мереж	-	-	+

Аналізуючи дані таблиці, середовище розробки Visual Studio краще за аналоги, тому даний засіб розробки є оптимальним для реалізації програми, оскільки містить усі необхідні можливості для його реалізації.

Для вибору мови програмування було складено таблицю 3.2 з порівнянням популярних об’єктно-орієнтованих мов програмування.

Таблиця 3.2 - Порівняльний аналіз мов програмування

Мова / Характеристика	Python	C#	Java
Кросплатформність додатків	–	+	+
Автоматична збірка сміття	+	+	+
Перевантаження операторів	+	+	–
Підтримка бібліотек штучного інтелекту	+	+	–

Мова C# заслужено вважається однією з кращих мов програмування багатоцільового застосування [21], саме через це та переваги, наведені в таблиці 3.2, для реалізації програми з використанням нейромережі обрано саме її.

.NET Core – це модульна платформа для розробки програмного забезпечення з відкритим вихідним кодом, заснована на .NET Framework. Сумісна з такими операційними системами як Windows, Linux і macOS. Модульність .NET Core працює таким чином, що кожна програма може працювати з різними модулями і не залежить від єдиного оновлення платформи.

Важливою особливістю платформи .NET Core при розробці проекту є те, що до її складу входить бібліотека ML.NET [22]. Ця бібліотека надає можливість створювати алгоритми з використанням штучного інтелекту та підключати нейромережі, розроблені за допомогою інших технології та мов програмування.

3.2 Розробка модуля розпізнавання об’єктів у зображеннях

Модуль розпізнавання об’єктів у зображеннях оголошений як клас ModelScorer і містить два методи – LoadModel та Score. Метод LoadModel визначає послідовність функцій бібліотеки ML.NET для створення конвеєра розпізнавань, а метод Score – отримує вхідні дані та за допомогою створеного конвеєра розпізнавань повертає масив чисел з ймовірностями для кожного з 80 класів об’єктів. Код класу з даними методами наведено нижче.

```
public class ModelScorer : IModelScorer
{
    private readonly MLContext mlContext;
    private readonly PredictionEngine<ImageNetData, IPrediction> predictionEngine;

    public ModelScorer(IModel onnxModel)
    {
        this.mlContext = new MLContext();
        this.predictionEngine = LoadModel(onnxModel);
    }

    public float[] Score(ImageNetData image) =>
this.predictionEngine.Predict(image).PredictedLabels;

    private PredictionEngine<ImageNetData, IPrediction> LoadModel(IModel onnxModel)
    {
        Console.WriteLine("Завантаження моделі.");
        Console.WriteLine($"Розташування моделі: {onnxModel.ModelPath}");
        var data = mlContext.Data.LoadFromEnumerable(new List<ImageNetData>());
        var pipeline = mlContext.Transforms.LoadImages(
                outputColumnName: onnxModel.ModelInput,
                imageFolder: "",
                inputColumnName: nameof(ImageNetData.ImagePath))
            .Append(mlContext.Transforms.ResizeImages(
                outputColumnName: onnxModel.ModelInput,
                imageWidth: 416,
                imageHeight: 416,
                inputColumnName: onnxModel.ModelInput,
                resizing: ImageResizingEstimator.ResizingKind.Fill))
            .Append(mlContext.Transforms.ExtractPixels(
                outputColumnName: onnxModel.ModelInput,
                inputColumnName: onnxModel.ModelInput))
            .Append(mlContext.Transforms.ApplyOnnxModel(
                outputColumnName: onnxModel.ModelOutput,
                inputColumnName: onnxModel.ModelInput,
                modelFile: onnxModel.ModelPath));



        var model = pipeline.Fit(data);

        var predictionEng = mlContext.Model.CreatePredictionEngine<ImageNetData,
IPrediction>(model);

        return predictionEng;
    }
}
```

3.3 Розробка модуля аналізу та візуалізації вихідних даних після обробки

Наступним етапом розробки програми розпізнавання об’єктів у зображеннях є створення класів та функцій для аналізу та візуалізації вихідних даних після обробки.

Перш за все необхідно створити клас OutputParser, який міститиме усі необхідні поля та методи для аналізу даних. Код класу наведено далі.

```
public class OutputParser
{
    // Кількість рядків та сповпців у сітці, на яку поділяється зображення
    public const int ROW_COUNT = 13;
    public const int COL_COUNT = 13;
    // Кількість елементів, що містяться у комірці (x, y, height, width,
confidence).
    public const int BOX_INFO_FEATURE_COUNT = 5;
    // Назви 80 класів об’єктів, які може розпізнавати модель
    private readonly string[] classLabels;
    // Заздалегідь визначені розміри обмежуючих прямокутників у комірці
    private readonly (float x, float y)[] boxAnchors;

    public OutputParser(IOnnxModel<(float, float)> onnxModel)
    {
        boxAnchors = onnxModel.Anchors;
        classLabels = new[]
        {
            "людина", "велосипед", "машина", "мотоцикл", "літак", "автобус",
"поїзд",  "вантажівка", "човен", "світлофор", "пожежний гідрант", "знак зупинки",
"паркомат", "лавка", "птах", "кіт", "собака", "кінь", "вівця", "корова", "слон",
"ведмідь", "зебра", "жирафа", "рюкзак", "парасолька", "сумочка", "краватка",
"валіза", "фрісбі", "лижі", "сноуборд", "спортивний м'яч", "повітряний змій",
"бейсбольна бита", "бейсбольна рукавиця", "скейтборд", "дошка для серфінгу",
тенісна ракетка", "пляшка", "келих", "чашка", "виделка", "ніж", "ложка", "миска",
"банан", яблуко", "сендвіч", "апельчин", "брокколі", "морква", "хот-дог", "піца",
"пончик", "пиріг", стілець", "диван", "вазон", "ліжко", "стіл", "туалет",
"телевізор", "ноутбук", "миша", "пульт", "клавіатура", "мобільник",
"мікрохвильовка", "духовка", "тостер", "раковина", "холодильник", "книга",
"годинник", "ваза", "ножиці", "плюшевий ведмедик", "фен", "зубна щітка"
        };
    }
}
```

Додамо в даний клас метод ParseOutputs, який буде обробляти масив вихідних даних, створених мережею. Цей метод стане важливим компонентом процесу розпізнавання об'єктів, оскільки він перетворює сирі результати, отримані від нейронної мережі, у структуровану та зрозумілу для користувача інформацію.

Метод ParseOutputs буде виконувати такі основні завдання:

- Отримання вихідних даних - прийом масиву вихідних даних від нейронної мережі, що можуть включати ймовірності класів, координати об'єктів на зображенні та інші метрики.

- Ідентифікація об'єктів - визначення об'єктів на зображенні на основі ймовірностей класів. Це включає порівняння ймовірностей з пороговими значеннями для прийняття рішення про те, чи є певний об’єкт.

```
public IList<BoundingBox> ParseOutputs(float[] onnxModelOutputs, float
probabilityThreshold = .3f)
{
    var boxes = new List<BoundingBox>();
    for (int row = 0; row < ROW_COUNT; row++)
    {
        for (int column = 0; column < COL_COUNT; column++)
        {
            for (int box = 0; box < boxAnchors.Length; box++)
            {
                var channel = box * (classLabels.Length + BOX_INFO_FEATURE_COUNT);
                var boundingBoxPrediction =
ExtractBoundingBoxPrediction(onnxModelOutputs, row, column, channel);
                var mappedBoundingBox = MapBoundingBoxToCell(row, column, box,
boundingBoxPrediction);
                if (boundingBoxPrediction.Confidence < probabilityThreshold)
                    continue;
                var classProbabilities =
ExtractClassProbabilities(onnxModelOutputs, row, column, channel,
boundingBoxPrediction.Confidence);
                var (topProbability, topIndex) = classProbabilities
                    .Select((probability, index) => (Score: probability, Index:
index)).Max();
                if (topProbability < probabilityThreshold)
                    continue;
                boxes.Add(new BoundingBox
                {
                    Dimensions = mappedBoundingBox,
                    Confidence = topProbability,
                    Label = classLabels[topIndex],
                    BoxColor = BoundingBox.GetColor(topIndex)
                });
            }
        }
    }
    return boxes;
}
```

Далі, коли координати всіх обмежуючих прямокутників були витягнуті з вихідних даних мережі, необхідно провести додаткову фільтрацію для видалення прямокутників, що перекриваються. Додамо метод FilterBoundingBoxes під методом ParseOutputs, код якого наведено далі:

```
public IList<BoundingBox> FilterBoundingBoxes(IList<BoundingBox> boxes, int limit,
float threshold)
{
    var filteredBoxes = new bool[boxes.Count];
    var sortedBoxes = boxes.OrderByDescending(b => b.Confidence).ToArray();
    var results = new List<BoundingBox>();
```

```
  for (int i = 0; i < boxes.Count; i++)
    {
        if (filteredBoxes[i])
        {
            continue;
        }
        results.Add(sortedBoxes[i]);
        if (results.Count >= limit)
        {
            break;
        }

        for (var j = i + 1; j < boxes.Count; j++)
        {
            if (filteredBoxes[j])
            {
                continue;
            }
            if (IntersectionOverUnion(sortedBoxes[i].Rect, sortedBoxes[j].Rect) >
threshold)
            {
                filteredBoxes[j] = true;
            }
            if (filteredBoxes.Count(b => b) <= 0)
            {
                break;
            }
        }
    }
    return results;
}
```

Останнім етапом, додамо метод DrawBoundingBox, який до вхідного зображення додає обмежуючі прямокутники за визначеними координатами, виділяючи об’єкти розпізнаних класів з їх назвою та достовірністю розпізнання, та повертає результуюче зображення користувачеві. Код методу наведено далі.

```
public Image DrawBoundingBox(string inputImageLocation, string outputImageLocation,
string imageName, Image inputImage = null)
{
    var image = inputImage ?? Image.FromFile(Path.Combine(inputImageLocation,
imageName));
    var originalImageHeight = image.Height;
    var originalImageWidth = image.Width;
    foreach (var box in this._filteredBoundingBoxes)
    {
        // Отримати розміри обмежуючого прямокутника
        var x = (uint)Math.Max(box.Dimensions.X, 0);
        var y = (uint)Math.Max(box.Dimensions.Y, 0);
        var width = (uint)Math.Min(originalImageWidth - x, box.Dimensions.Width);
        var height = (uint)Math.Min(originalImageHeight - y,
box.Dimensions.Height);

        // Змінити розмір зображення до початкового
        x = (uint)originalImageWidth * x / ImageNetSettings.imageWidth;
```

```
        y = (uint)originalImageHeight * y / ImageNetSettings.imageHeight;
        width = (uint)originalImageWidth * width / ImageNetSettings.imageWidth;
        height = (uint)originalImageHeight * height / ImageNetSettings.imageHeight;
        using (Graphics thumbnailGraphic = Graphics.FromImage(image))
        {
            thumbnailGraphic.CompositingQuality = CompositingQuality.HighQuality;
            thumbnailGraphic.SmoothingMode = SmoothingMode.HighQuality;
            thumbnailGraphic.InterpolationMode =
InterpolationMode.HighQualityBicubic;

            // Визначити параметри тексту
            Font drawFont = new Font("Arial", 12, FontStyle.Bold);
            SizeF size = thumbnailGraphic.MeasureString(box.Description, drawFont);
            SolidBrush fontBrush = new SolidBrush(Color.Black);
            Point atPoint = new Point((int)x, (int)y - (int)size.Height + 20);
            Pen pen = new Pen(box.BoxColor, 3.2f);
            SolidBrush colorBrush = new SolidBrush(box.BoxColor);

            // Намалювати обмежуючий прямокутник на зображенні
            thumbnailGraphic.DrawRectangle(pen, x, y, width, height);

            // Намалювати текст на зображенні
            thumbnailGraphic.FillRectangle(colorBrush, (int)x, (int)(y -
size.Height + 20), (int)size.Width, (int)size.Height);
            thumbnailGraphic.DrawString(box.Description, drawFont, fontBrush,
atPoint);
        }
    }
     if (inputImage == null)
     {
         if (!Directory.Exists(outputImageLocation))
         {
             Directory.CreateDirectory(outputImageLocation);
         }
         image.Save(Path.Combine(outputImageLocation, imageName));
     }
     return image;
}
```

3.4 Висновок

У третьому розділі було обґрунтовано вибір середовища розробки, мови програмування та технологій, які будуть використовуватися при розробці програми та наведено їх основні переваги. У результаті аналізу було обрано середовище розробки Visual Studio, мову програмування C# та технології ML.NET для розробки модулів обробки зображень і .NET Core для розробки графічного інтерфейсу користувача. Було проведено реалізацію програмних модулів, а саме модуля розпізнавання об’єктів у зображеннях та модуля аналізу та візуалізації вихідних даних після обробки.

4 АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ ПРОГРАМИ

4.1 Аналіз методів тестування

Тестування програмного забезпечення – це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись [23]. Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою оцінки. Може оцінюватись: відповідність вимогам, якими керувалися проектувальники та розробники, правильна відповідь для усіх можливих вхідних даних, виконання функцій за прийнятний час, практичність, сумісність з програмним забезпеченням та операційними системами, відповідність задачам замовника [24].

Методи тестування програмного додатку:

- статичне тестування;
- динамічне тестування;
- тестування «білої скриньки»;
- тестування «чорної скриньки».

Статичне тестування – тестова діяльність, що пов'язана з аналізом результатів розробки програмного забезпечення. Воно передбачає перевірку програмних кодів, контроль та перевірку програми без запуску на комп'ютері. На етапі статичного тестування перевіряється вся документація, отримана як результат життєвого циклу програми.

Динамічне тестування – тестова діяльність, що передбачає експлуатацію програмного продукту. Динамічне та статичне тестування доповнюють одне одного. Динамічні методи застосовуються в процесі безпосереднього виконання програми. Коректність програмного засобу перевіряється на безлічі тестів або наборів підготовлених вхідних даних. При прогоні кожного тесту збираються та аналізуються дані про відмови та збої в роботі програми.

Тестування «білої скриньки» – тестування, у якому відома внутрішня

структура програми, а досліджуються внутрішні елементи програми і зв’язки між ними. Об'єктом тестування тут є не зовнішня, а внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним. Зазвичай аналізуються керуючі зв'язки елементів, рідше – інформаційні зв'язки. Тестування за принципом «білої скриньки» характеризується ступенем, в якому тести виконують або покривають логіку (вихідний код) програми.

У цьому випадку формуються тестові варіанти, в яких:

- гарантується перевірка всіх незалежних маршрутів програми;
- знаходяться гілки True, False для всіх логічних рішень;
- виконуються всі цикли (у межах їхніх кордонів та діапазонів);
- аналізується правильність внутрішніх структур даних.

Тестування «чорної скриньки» – тестування, у якому відомі функції додатку, а досліджується робота кожної функції на всій області визначення. При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе. Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 10^10. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок, а забезпечує пошук наступних категорій помилок:

- некоректних чи відсутніх функцій;
- помилок інтерфейсу;
- помилок у зовнішніх структурах даних або в доступі до бази даних;
- помилок характеристик (необхідна ємність пам'яті тощо).

Подібні категорії помилок способами «білої скриньки» не виявляються.

У випадку тестування розробленої програми, використовується тестування за методом «чорної скриньки», так як воно найкраще підходить для тестування готових додатків, що не передбачають доступ до вихідного коду тестувальником.

4.2 Тестування програмного модуля розпізнавання об’ктів

Для тестування програми за методом «чорної скриньки» було розроблено і виконано варіанти тестування для основних функцій та варіантів використання, доступних для користувача. Результати тестування наведені в таблиці 4.1.

Таблиця 4.1. – Варіанти тестування за методом «чорної скриньки»

№ п/п	Вхідні дані	Методика	Вихідні дані	Відповідність
1	Зображення без об’єктів	Запустити програму, вибрати зображення	Зображення без виділених об’єктів	Відповідає (див. рис. 4.1)
2	Зображення з 1 об’єктом	Запустити програму, вибрати зображення	Зображення з 1 виділеним об’єктом	Відповідає (див. рис. 4.2)
3	Зображення з 2 об’єктами	Запустити програму, вибрати зображення	Зображення з 2 виділеними об’єктами	Відповідає (див. рис. 4.3)
4	Зображення з 3 об’єктами	Запустити програму, вибрати зображення	Зображення з 3 виділеними об’єктами	Відповідає (див. рис. 4.4)
5	Зображення з повними та неповними об’єктами	Запустити програму, вибрати зображення	Зображення з виділеними повними та неповними об’єктами	Відповідає (див. рис. 4.5)
6	Зображення з безліччю об’єктів на передньому і задньому плані	Запустити програму, вибрати зображення	Зображення з виділеними об’єктами переднього плану	Відповідає (див. рис. 4.6)

Тест №1. Після запуску програми з обраним зображенням без чітко визначених об’єктів, слід натиснути кнопку «Upload File». Очікуваний результат: програма повертає зображення без виділених об’єктів. Отриманий результат відповідає очікуваному. Результат тесту – успіх. Результат цього тесту зображено на рисунку 4.1.

Рисунок 4.1 – Результат тесту №1

Тест №2. Після запуску програми з обраним зображенням з одним чітко визначеним об’єктом, слід натиснути кнопку «Upload File». Очікуваний результат: програма повертає зображення з 1 виділеним об’єктом. Отриманий результат відповідає очікуваному. Результат тесту – успіх. Результат цього тесту зображено на рисунку 4.2.

Рисунок 4.2 – Результат тесту №2

Тест №3. Після запуску програми з обраним зображенням з двома чітко визначеними об’єктами, слід натиснути кнопку «Upload File». Очікуваний результат: програма повертає зображення з 2 виділеними об’єктами. Отриманий результат відповідає очікуваному. Результат тесту – успіх. Результат цього тесту зображено на рисунку 4.3.

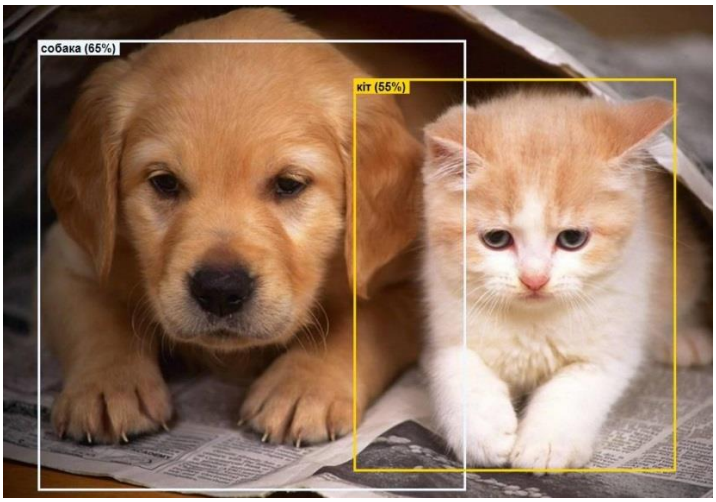

Рисунок 4.3 – Результат тесту №3

Тест №4. Після запуску програми з обраним зображенням з трьома чітко визначеними об’єктами, слід натиснути кнопку «Upload File». Очікуваний результат: програма повертає зображення з 3 виділеними об’єктами. Отриманий результат відповідає очікуваному. Результат тесту – успіх. Результат цього тесту зображено на рисунку 4.4.

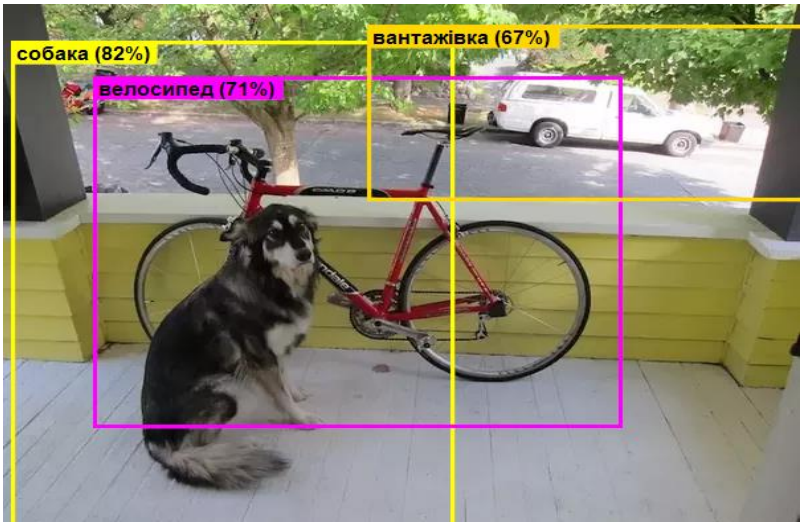

Рисунок 4.4 – Результат тесту №4

Тест №5. Після запуску програми з обраним зображенням з повними та неповними об’єктами, слід натиснути кнопку «Upload File». Очікуваний результат: програма повертає зображення з виділеними повними та неповними об’єктами. Отриманий результат відповідає очікуваному. Результат тесту – успіх. Результат цього тесту зображено на рисунку 4.5.

Рисунок 4.5 – Результат тесту №5

Тест №6. Після запуску програми з обраним зображенням з безліччю чітко визначених об’єктів на передньому і задньому плані, слід натиснути кнопку «Upload File». Очікуваний результат: програма повертає зображення з виділеними об’єктами переднього плану. Отриманий результат відповідає очікуваному. Результат тесту – успіх. Результат цього тесту зображено на рисунку 4.6.

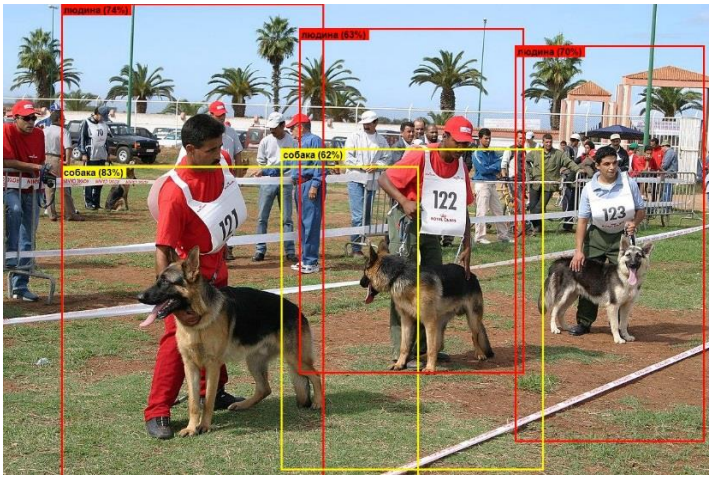

Рисунок 4.6 – Результат тесту №6

Отже, у ході тестування програмного модуля розпізнавання об’єктів у зображеннях було проведено 6 тестів. Згідно з результатами тестування, робота програми на 100% відповідає результатам тестів, що підтверджує її функціональність.

Для доведення досягнення поставленої в роботі мети – підвищення достовірності роботи програмного модуля розпізнавання об’єктів на зображеннях – було протестовано роботу розробленої програми та програми-аналога Inception [12] на 100 зображеннях із тестової вибірки. Результати тестування представлені у таблиці 4.2.

Таблиця 4.2 – Результати тестування розробленого програмного модуля та програми-аналога Inception

Програмний засіб	Кількість зображень у тестовій вибірці	Сумарна кількість об’єктів на зображеннях	Кількість правильно розпізнаних об’єктів	Достовірність розпізнавання об’єктів на зображеннях, %
Програма Inception	100	240	211	87,9
Розроблений програмний модуль	100	240	229	95,4

Із таблиці 4.2 видно, що розроблений програмний модуль має вищу достовірність розпізнавання об’єктів на зображеннях (95,4%), ніж аналогічна програма (87,9%), а значить достовірність розпізнавання об’єктів на зображеннях покращена на 7,5%, тобто мета роботи досягнута.

4.3 Висновок

У четвертому розділі було розглянуто різні методи тестування, та визначено, що для даної нейромережевої системи найоптимальнішим є тестування методом «чорної скриньки». Під час тестування було розглянуто декілька можливих випадків розпізнавання об’єктів у зображеннях. В результаті тестування програми було доведено її повну працездатність та відповідність поставленому завданню. Серед переваг програми можна відзначити її більшу достовірність у розпізнаванні великих об’єктів – тих, які знаходяться на передньому плані і займають певну частину зображення. До недоліків нейромережевої системи можна віднести неможливість розпізнавання дуже малих об’єктів заднього плану зображення. Розроблений програмний модуль має вищу достовірність розпізнавання об’єктів на зображеннях (95,4%), ніж аналогічна програма (87,9%), а значить достовірність розпізнавання об’єктів на зображеннях покращена на 7,5%, тобто мета роботи досягнута

ВИСНОВКИ

Під час виконання бакалаврської дипломної роботи було розроблено програмний модуль розпізнавання рухомих об’єктів.

В роботі було розглянуто стан питання розпізнавання рухомих об’єктів. Проведено огляд відомих методів для роботи з інтелектуальними обчисленнями у сфері комп’ютерного зору. Здійснено аналіз відомих програмних засобів розпізнавання рухомих об’єктів та як аналог до розроблюваного модуля було обрано програму Inception. Обґрунтовано доцільність використання для розпізнавання рухомих об’єтів згорткової нейронної мережі, а саме – попередньо натренованої нейронної мережі YOLOv2. Була проаналізована структура та порядок функціонування згорткової нейромережі YOLOv2. Також було розроблено алгоритм роботи програми та структуру програмного змодуля розпізнавання рухомих об’єктів на зображеннях, отриматих з відеофрагментів. В роботі було проведено реалізацію програмного модуля розпізнавання об’єктів у зображеннях та модуля аналізу та візуалізації вихідних даних після обробки мовою програмування C# у середовищі Visual Studio. Тестування довело, що розроблений програмний модуль має вищу достовірність розпізнавання об’єктів на зображеннях (95,4%), ніж аналогічна програма (87,9%), а значить достовірність розпізнавання об’єктів на зображеннях покращена на 7,5%.

Отже всі поставлені задачі виконано, мету роботи досягнуто.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Object Detection [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Object_detection.
2. Tang, Y.; Zhang, C.; Gu, R. Vehicle detection and recognition for intelligent traffic surveillance system. Multimed. Tools Appl. 2017, 76, 5817–5832.
3. Згорткова нейронна мережа – просте пояснення CNN та її застосування – [Електронний ресурс] – https://evergreens.com.ua/ua/articles/cnn.html
4. Bradski G. Learning OpenCV - Computer Vision with the OpenCV Library / G. Bradski.,2018 – 580 с.
5. Машинне навчання з використанням мікрокомп’ютерів – [Електронний ресурс] – Режим доступу: http://man.gov.ua/files/49/Machine_Nav4ann_Mogilniy.pdf
6. Python documentation – [Електронний ресурс] – https://www.python.org/doc/
7. Image Classification on ImageNet – [Електронний ресурс] – Режим доступу: https://paperswithcode.com/sota/image-classification-on-imagenet
8. Machine vision [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Machine_vision.
9. Dougherty G. / Digital Image Processing for Medical Applications / G. Dougherty – Cambridge University Press. – ISBN 978-0-5218-6085-7.
10. Pan, C.; Sun, M.; Yan, Z. The Study on Vehicle Detection Based on DPM in Traffic Scenes. In Proceedings of the International Conference on Frontier Computing, Tokyo, Japan, 13–15 July 2016; pp. 19–27.
11. Куссуль Н. М. Інтелектуальні обчислення: навчальний посібник (із грифом МОН України) / Н. М. Куссуль, А. Ю. Шелестов, А. Н. Лавренюк. - К.: «Наукова думка», 2006. — 186 с. ISBN 966-00-0592-X.
12. Руденко О.В. Штучні нейронні мережі: Навчальний посібник / О.В.Руденко, Є.В.Бодянський. - Харків : ТОВ «Компанія СМІТ», 2006. — 404 с. - ISBN 966-8630-73-X.
13. Krizhevsky A. ImageNet Classification with Deep Convolutional Neural Networks [Електронний ресурс]. URL: https://papers.nips.cc/paper/4824imagenet-classification-with-deep-convolutional-neural-networks.pdf.

14. Inceptionv3 [Електронний ресурс]. URL: https://se.mathworks.com/help/deeplearning/ref/inceptionv3.html
15. Глибокі нейронні мережі для вирішення завдань розпізнавання і класифікації зображення [Електронний ресурс]. – Режим доступу: http://itcm.comp-sc.if.ua/2017/Sineglazov.pdf
16. Recurrent neural network [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Recurrent_neural_network.
17. What is Object Detection [Електронний ресурс] – Режим доступу до ресурсу: https://www.mathworks.com/discovery/object-detection.html/.
18. Redmon, J.; Farhadi, A. YOLO9000: Better, Faster, Stronger. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 6517–6525.
19. Eclipse desktop & web IDEs [Електронний ресурс] – Режим доступу: https://www.eclipse.org/ide/.
20. ML.NET – Machine Learning made for .NET [Електронний ресурс] – Режим доступу: https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet/.
21. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп’ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. В. Сілагін, І. В. Богач. – Вінниця : ВНТУ, 2023. – (PDF, 54 с.).

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмний модуль розпізнавання рухомих об'єктів

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 80,5% Схожість 19,5%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Бевз А.О.

Керівник роботи Озеранський В.С.

ДОДАТОК Б

ІНСТРУКЦІЯ КОРИСТУВАЧА

Так як додаток не вимагає інсталяції, то для початку роботи з ним достатньо запустити файл ObjectDetector.Web.exe. За замовчуванням відкривається вікно браузера, в якому можна побачити робочу область програми (рис. Б.1).

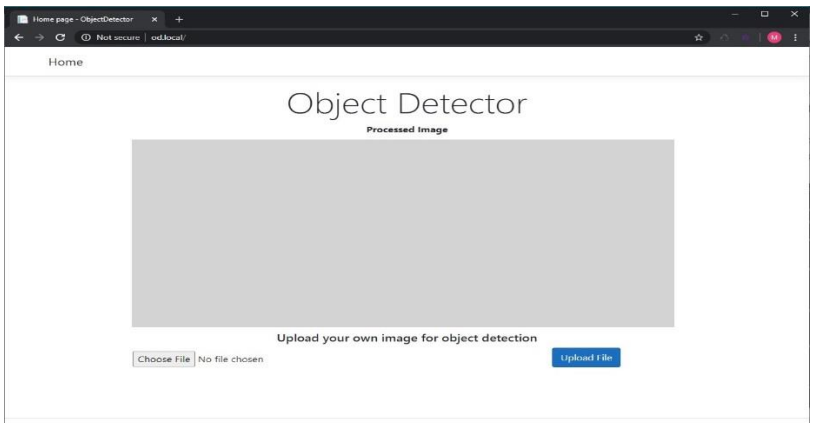

Рисунок Б.1 – Головне вікно програми

Для того, аби вибрати зображення для розпізнавання об’єктів у ньому, слід натиснути кнопку «Choose File», яка відкриває вікно вибору зображення з файлової системи користувача (рис. Б.2). Допустимі формати файлів: .jpg, .jpeg, .png. Після обрання необхідного файлу зображення, необхідно натиснути кнопку «Open», після чого файл буде доданий у програму для обробки.

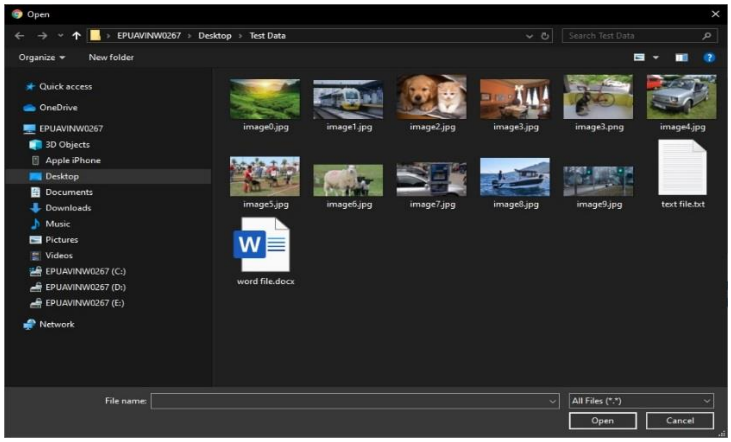

Рисунок Б.2 – Вікно вибору зображення

Після того, як зображення обрано, слід натиснути кнопку «Upload File». Додаток виконує усі необхідні обчислення та повертає результуюче зображення з виділеними класами об’єктів, їх назвою та достовірністю розпізнавання у робочу область програми. Також програма повертає кількість затраченого часу в мілісекундах на розпізнавання об’єктів у зображенні. Приклад роботи додатку наведено на рисунку Б.3.

Для того, аби повторно виконати розпізнавання об’єктів у зображенні, слід знову натиснути кнопку «Choose File» та повторити вищеописаний процес. Програма працює до повного закриття головного вікна користувачем.

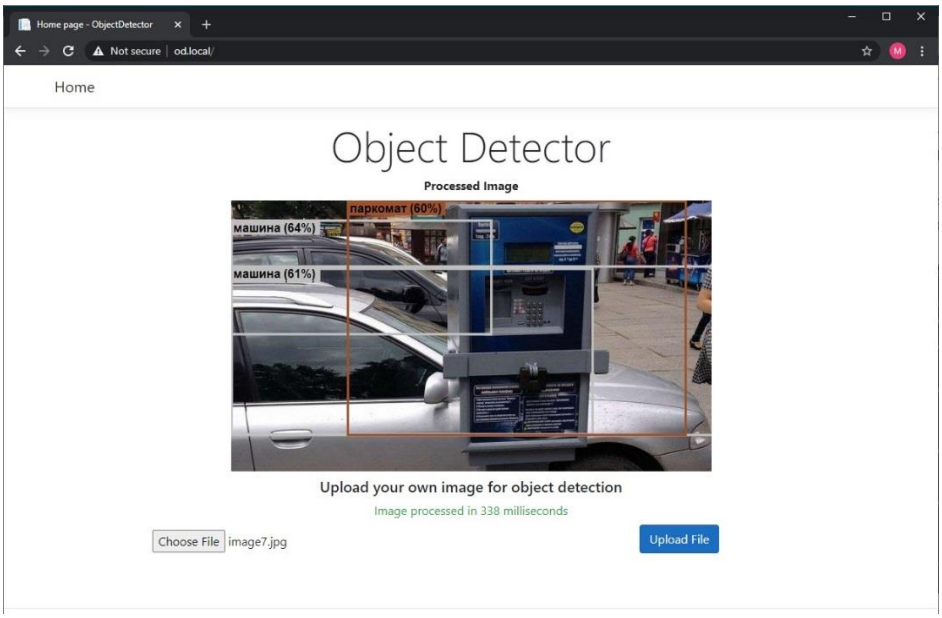

Рисунок Б.3 – Приклад роботи додатку

ДОДАТОК В
ЛІСТИНГ ПРОГРАМИ

YoloV2ObjectDetectionService.cs

```
namespace ObjectDetector.Services.V2.Services
{
    using System;
    using System.Collections.Generic;
    using System.Drawing;
    using System.Drawing.Drawing2D;
    using System.IO;
    using ObjectDetector.Infrastructure.DataStructures;
    using ObjectDetector.Infrastructure.YoloParser;
    using ObjectDetector.Services.Interfaces;
    using ObjectDetector.Services.Models;
    using ObjectDetector.Services.V2.DataModels;

    public class YoloV2ObjectDetectionService : ITinyYoloV2ObjectDetectionService,
IYoloV2ObjectDetectionService
    {
        private IList<YoloBoundingBox> _filteredBoundingBoxes;
        private readonly IYoloV2Model _model;
        private readonly YoloV2ModelScorer _modelScorer;
        private readonly YoloV2OutputParser _outputParser;

        public YoloV2ObjectDetectionService(IYoloV2Model model)
        {
            IYoloV2Model model1;
            this._model = model;
            this._modelScorer = new YoloV2ModelScorer(this._model);
            this._outputParser = new YoloV2OutputParser(this._model);
        }

        public void DetectObjectsUsingModel(ImageNetData imageInputData, float
probability = .3F, int limit = 5, float intersection = .5F)
        {
            var probabilities = _modelScorer.Score(imageInputData);
            var boundingBoxes = _outputParser.ParseOutputs(probabilities, probability);
            this._filteredBoundingBoxes =
_outputParser.FilterBoundingBoxes(boundingBoxes, limit, intersection);
        }

        public void DrawBoundingBox(string inputImageLocation, string
outputImageLocation, string imageName)
        {
            Image image = Image.FromFile(Path.Combine(inputImageLocation, imageName));
            var originalImageHeight = image.Height;
            var originalImageWidth = image.Width;

            foreach (var box in this._filteredBoundingBoxes)
            {
                var x = (uint)Math.Max(box.Dimensions.X, 0);
                var y = (uint)Math.Max(box.Dimensions.Y, 0);
                var width = (uint)Math.Min(originalImageWidth - x,
box.Dimensions.Width);
                var height = (uint)Math.Min(originalImageHeight - y,
```

```
box.Dimensions.Height);
                // Resize To Image
                x = (uint)originalImageWidth * x / ImageNetSettings.imageWidth;
                y = (uint)originalImageHeight * y / ImageNetSettings.imageHeight;
                width = (uint)originalImageWidth * width / ImageNetSettings.imageWidth;
                height = (uint)originalImageHeight * height /
ImageNetSettings.imageHeight;

                using (Graphics thumbnailGraphic = Graphics.FromImage(image))
                {
                    thumbnailGraphic.CompositingQuality =
CompositingQuality.HighQuality;
                    thumbnailGraphic.SmoothingMode = SmoothingMode.HighQuality;
                    thumbnailGraphic.InterpolationMode =
InterpolationMode.HighQualityBicubic;
                    // Define Text Options
                    Font drawFont = new Font("Arial", 12, FontStyle.Bold);
                    SizeF size = thumbnailGraphic.MeasureString(box.Description,
drawFont);
                    SolidBrush fontBrush = new SolidBrush(Color.Black);
                    Point atPoint = new Point((int)x, (int)y - (int)size.Height + 20);
                    // Define BoundingBox options
                    Pen pen = new Pen(box.BoxColor, 3.2f);
                    SolidBrush colorBrush = new SolidBrush(box.BoxColor);
                    // Draw bounding box on image
                    thumbnailGraphic.DrawRectangle(pen, x, y, width, height);
                    // Draw text on image
                    thumbnailGraphic.FillRectangle(colorBrush, (int)x, (int)(y -
size.Height + 20), (int)size.Width, (int)size.Height);
                    thumbnailGraphic.DrawString(box.Description, drawFont, fontBrush,
atPoint);
                }
            }

            if (!Directory.Exists(outputImageLocation))
            {
                Directory.CreateDirectory(outputImageLocation);
            }
            image.Save(Path.Combine(outputImageLocation, imageName));
        }
    }
}
```

YoloV2ModelScorer.cs

```
namespace ObjectDetector.Services.V2.Services
{
    using System;
    using System.Collections.Generic;
    using Microsoft.ML;
    using Microsoft.ML.Transforms.Image;
    using ObjectDetector.Infrastructure.DataStructures;
    using ObjectDetector.Services.Interfaces;
    using ObjectDetector.Services.Models;
    using ObjectDetector.Services.V2.DataModels;

    public class YoloV2ModelScorer : IModelScorer
    {
        private readonly MLContext mlContext;
        private readonly PredictionEngine<ImageNetData, YoloV2Prediction>
```

```
predictionEngine;

        public YoloV2ModelScorer(IYoloV2Model onnxModel)
        {
            this.mlContext = new MLContext();
            this.predictionEngine = LoadModel(onnxModel);
        }

        public float[] Score(ImageNetData image) =>
this.predictionEngine.Predict(image).PredictedLabels;

        private PredictionEngine<ImageNetData, YoloV2Prediction> LoadModel(IYoloV2Model
onnxModel)
        {
            Console.WriteLine("Завантаження моделі.");
            Console.WriteLine($"Розташування моделі: {onnxModel.ModelPath}");
            Console.WriteLine($"Default parameters: image
size=({ImageNetSettings.imageWidth},{ImageNetSettings.imageHeight})");

            var data = mlContext.Data.LoadFromEnumerable(new List<ImageNetData>());
            var pipeline = mlContext.Transforms.LoadImages(
                    outputColumnName: onnxModel.ModelInput,
                    imageFolder: "",
                    inputColumnName: nameof(ImageNetData.ImagePath))
                .Append(mlContext.Transforms.ResizeImages(
                    outputColumnName: onnxModel.ModelInput,
                    imageWidth: ImageNetSettings.imageWidth,
                    imageHeight: ImageNetSettings.imageHeight,
                    inputColumnName: onnxModel.ModelInput,
                    resizing: ImageResizingEstimator.ResizingKind.Fill))
                .Append(mlContext.Transforms.ExtractPixels(
                    outputColumnName: onnxModel.ModelInput,
                    inputColumnName: onnxModel.ModelInput))
                .Append(mlContext.Transforms.ApplyOnnxModel(
                    outputColumnName: onnxModel.ModelOutput,
                    inputColumnName: onnxModel.ModelInput,
                    modelFile: onnxModel.ModelPath));

            var model = pipeline.Fit(data);
            var predictionEng = mlContext.Model.CreatePredictionEngine<ImageNetData,
YoloV2Prediction>(model);

             return predictionEng;
        }
    }
}
```

YoloV2OutputParser.cs

```
namespace ObjectDetector.Infrastructure.YoloParser
{
    using System;
    using System.Collections.Generic;
    using System.Drawing;
    using System.Linq;
    using ObjectDetector.Core.Interfaces;
    using ObjectDetector.Infrastructure.DataStructures;

    public class YoloV2OutputParser
    {
```

```
        private class BoundingBoxPrediction : BoundingBoxDimensions
        {
            public float Confidence { get; set; }
        }

    public const int ROW_COUNT = 13;
    public const int COL_COUNT = 13;

    public const int BOX_INFO_FEATURE_COUNT = 5;

    private readonly string[] classLabels;

    private readonly (float x, float y)[] boxAnchors;


    public YoloV2OutputParser(IOnnxModel<(float, float)> onnxModel)
    {
        classLabels = onnxModel.Labels;
        boxAnchors = onnxModel.Anchors;
    }
        private float Sigmoid(float value)
        {
            var k = MathF.Exp(value);
            return k / (1.0f + k);
        }

        private float[] Softmax(float[] classProbabilities)
        {
            var max = classProbabilities.Max();
            var exp = classProbabilities.Select(v => MathF.Exp(v - max));
            var sum = exp.Sum();
            return exp.Select(v => v / sum).ToArray();
        }

ML.NET flattens
        private int GetOffset(int row, int column, int channel)
        {
            const int channelStride = ROW_COUNT * COL_COUNT;
            return (channel * channelStride) + (column * COL_COUNT) + row;
        }

        private BoundingBoxPrediction ExtractBoundingBoxPrediction(float[] modelOutput,
int row, int column, int channel)
        {
            return new BoundingBoxPrediction
            {
                X = modelOutput[GetOffset(row, column, channel++)],
                Y = modelOutput[GetOffset(row, column, channel++)],
                Width = modelOutput[GetOffset(row, column, channel++)],
                Height = modelOutput[GetOffset(row, column, channel++)],
                Confidence = Sigmoid(modelOutput[GetOffset(row, column, channel++)])
            };
        }
        private BoundingBoxDimensions MapBoundingBoxToCell(int row, int column, int box,
BoundingBoxPrediction boxDimensions)
        {
            const float CELL_WIDTH = ImageNetSettings.imageWidth / COL_COUNT;
            const float CELL_HEIGHT = ImageNetSettings.imageHeight / ROW_COUNT;
```

```
            var mappedBox = new BoundingBoxDimensions
            {
                X = (row + Sigmoid(boxDimensions.X)) * CELL_WIDTH,
                Y = (column + Sigmoid(boxDimensions.Y)) * CELL_HEIGHT,
                Width = MathF.Exp(boxDimensions.Width) * CELL_WIDTH * boxAnchors[box].x,
                Height = MathF.Exp(boxDimensions.Height) * CELL_HEIGHT *
boxAnchors[box].y,
            };
            // The x,y coordinates from the (mapped) bounding box prediction represent
the center
            // of the bounding box. We adjust them here to represent the top left
corner.
            mappedBox.X -= mappedBox.Width / 2;
            mappedBox.Y -= mappedBox.Height / 2;

            return mappedBox;
        }
        // IoU (Intersection over union) measures the overlap between 2 boundaries.
        private float IntersectionOverUnion(RectangleF boundingBoxA, RectangleF
boundingBoxB)
        {
            var areaA = boundingBoxA.Width * boundingBoxA.Height;
            var areaB = boundingBoxB.Width * boundingBoxB.Height;
            if (areaA <= 0 || areaB <= 0)
            {
                return 0;
            }
            var minX = MathF.Max(boundingBoxA.Left, boundingBoxB.Left);
            var minY = MathF.Max(boundingBoxA.Top, boundingBoxB.Top);
            var maxX = MathF.Min(boundingBoxA.Right, boundingBoxB.Right);
            var maxY = MathF.Min(boundingBoxA.Bottom, boundingBoxB.Bottom);
            var intersectionArea = MathF.Max(maxY - minY, 0) * MathF.Max(maxX - minX,
0);
            return intersectionArea / (areaA + areaB - intersectionArea);
        }
        public IList<YoloBoundingBox> ParseOutputs(float[] onnxModelOutputs, float
probabilityThreshold = .3f)
        {
            var boxes = new List<YoloBoundingBox>();
            for (int row = 0; row < ROW_COUNT; row++)
            {
                for (int column = 0; column < COL_COUNT; column++)
                {
                    for (int box = 0; box < boxAnchors.Length; box++)
                    {
                        var channel = box * (classLabels.Length +
BOX_INFO_FEATURE_COUNT);
                        var boundingBoxPrediction =
ExtractBoundingBoxPrediction(onnxModelOutputs, row, column, channel);
                        var mappedBoundingBox = MapBoundingBoxToCell(row, column, box,
boundingBoxPrediction);

                        if (boundingBoxPrediction.Confidence < probabilityThreshold)
                            continue;
                        var classProbabilities =
ExtractClassProbabilities(onnxModelOutputs, row, column, channel,
boundingBoxPrediction.Confidence);
                        var (topProbability, topIndex) = classProbabilities
                            .Select((probability, index) => (Score: probability, Index:
```

```
index)).Max();
                        //var (topProbability, topIndex) = classProbabilities
                        //    .Select((probability, index) => (Score: probability,
Index: index))
                        //    .OrderByDescending(result => result.Score).First();
                        if (topProbability < probabilityThreshold)
                            continue;
                        boxes.Add(new YoloBoundingBox
                        {
                            Dimensions = mappedBoundingBox,
                            Confidence = topProbability,
                            Label = classLabels[topIndex],
                            BoxColor = YoloBoundingBox.GetColor(topIndex)
                        });
                    }
                }
            }
            return boxes;
        }
        public IList<YoloBoundingBox> FilterBoundingBoxes(IList<YoloBoundingBox> boxes,
int limit, float threshold)
        {
            var filteredBoxes = new bool[boxes.Count];
            var sortedBoxes = boxes.OrderByDescending(b => b.Confidence).ToArray();
            var results = new List<YoloBoundingBox>();
            for (int i = 0; i < boxes.Count; i++)
            {
                if (filteredBoxes[i])
                {
                    continue;
                }
                results.Add(sortedBoxes[i]);
                if (results.Count >= limit)
                {
                    break;
                }
                for (var j = i + 1; j < boxes.Count; j++)
                {
                    if (filteredBoxes[j])
                    {
                        continue;
                    }
                    if (IntersectionOverUnion(sortedBoxes[i].Rect, sortedBoxes[j].Rect)
> threshold)
                    {
                        filteredBoxes[j] = true;
                    }

                    if (filteredBoxes.Count(b => b) <= 0)
                    {
                        break;
                    }
                }
            }
            return results;
        }
    }
}
```

ДОДАТОК Г
ГРАФІЧНА ЧАСТИНА

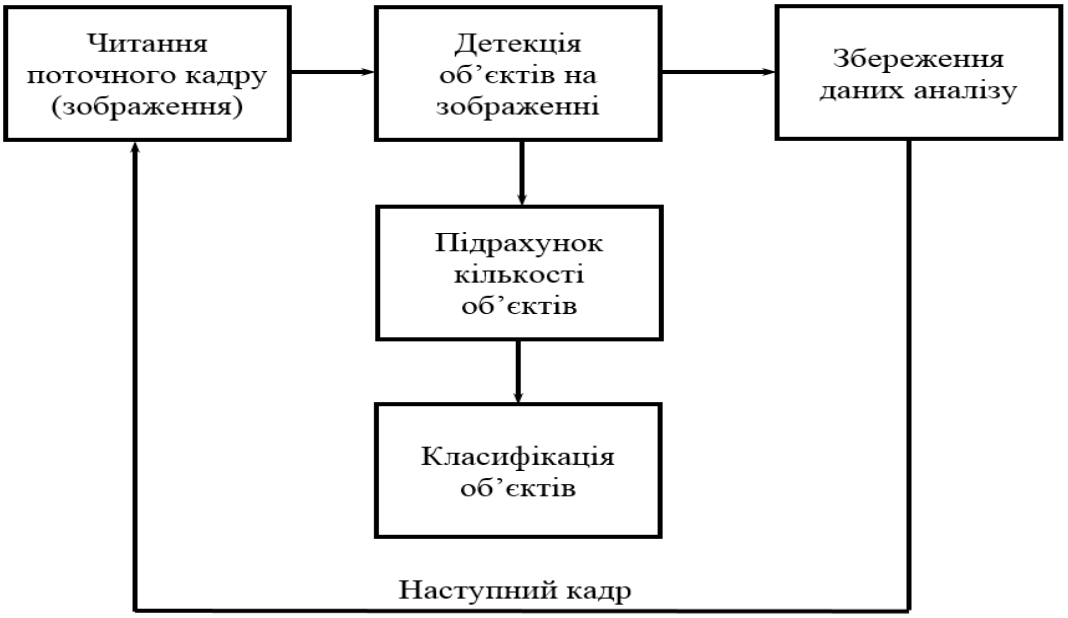

Рисунок Г.1 – Принцип роботи системи розпізнавання рухомих об'єктів

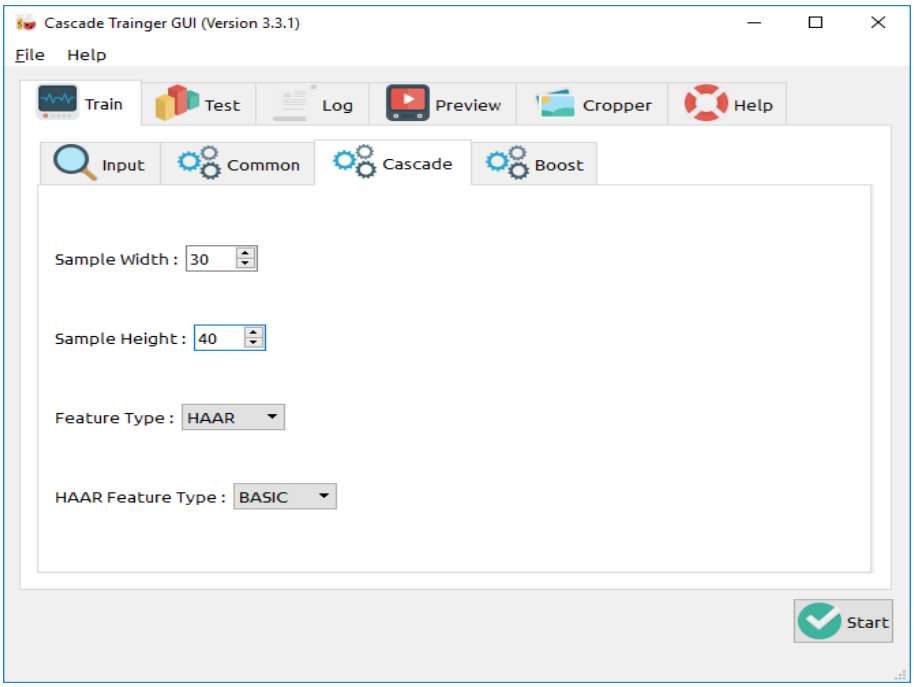

Рисунок 1.2 – Вікно програми BytePace

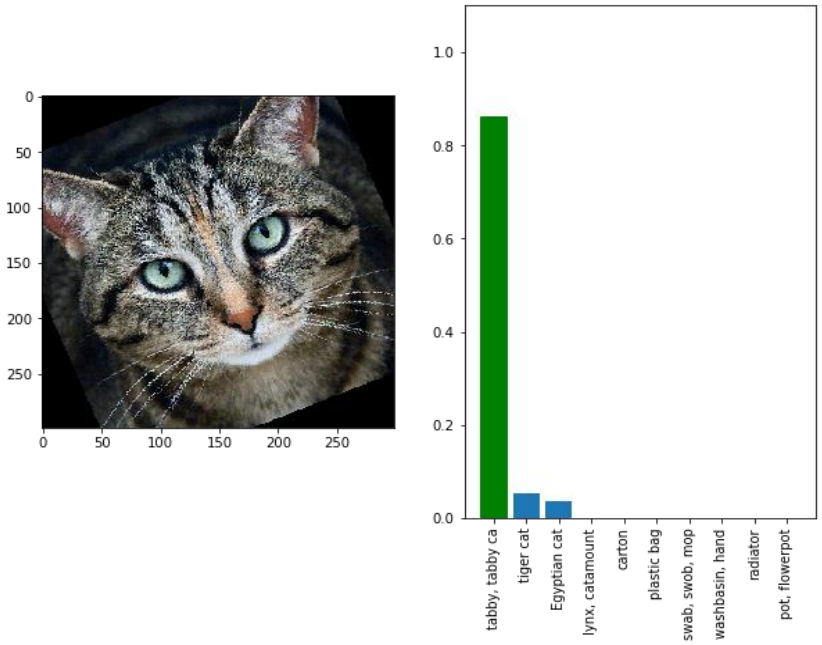

Рисунок Г.2 – Приклади програм аналогів

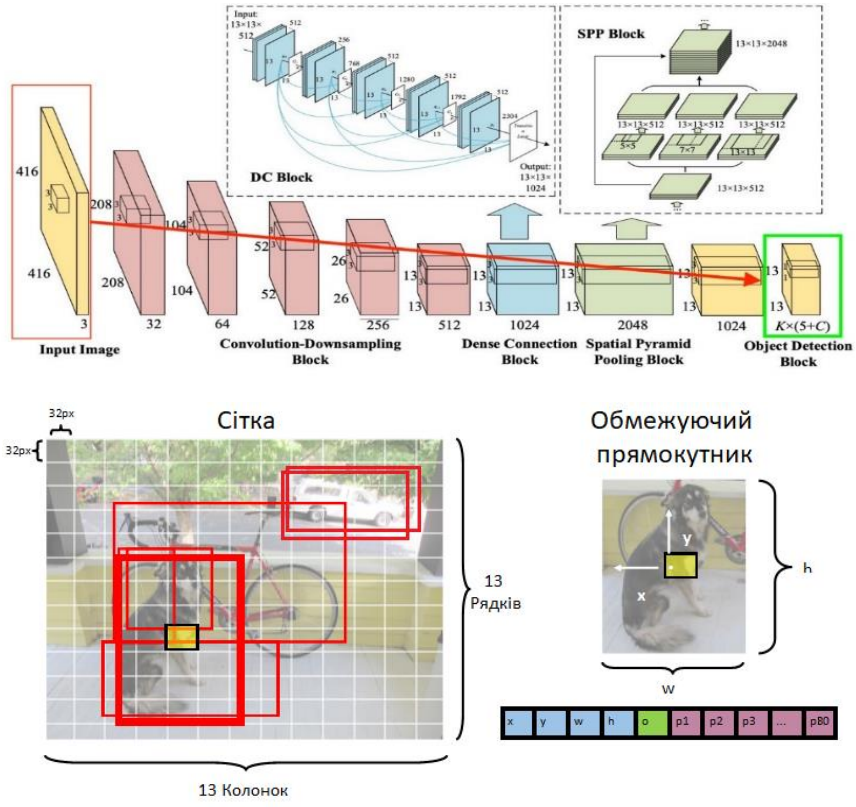

Рисунок Г.3 – Структура нейромережі YOLOv2

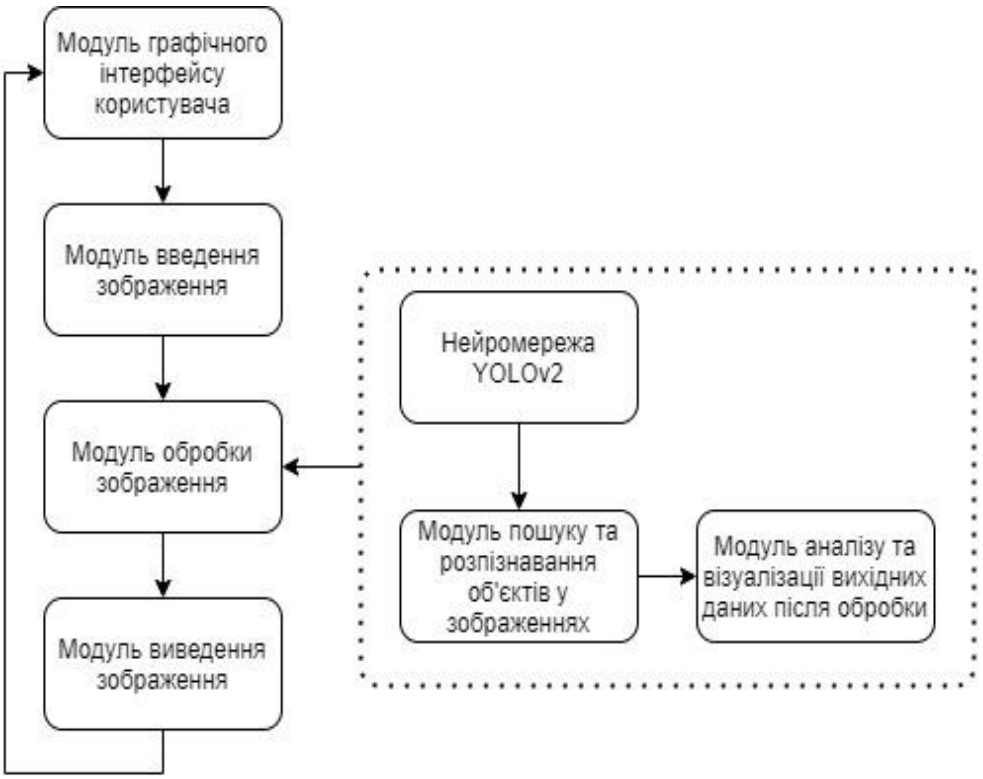

Рисунок Г.4 – Структура програмного забезпечення розпізнавання об’єктів

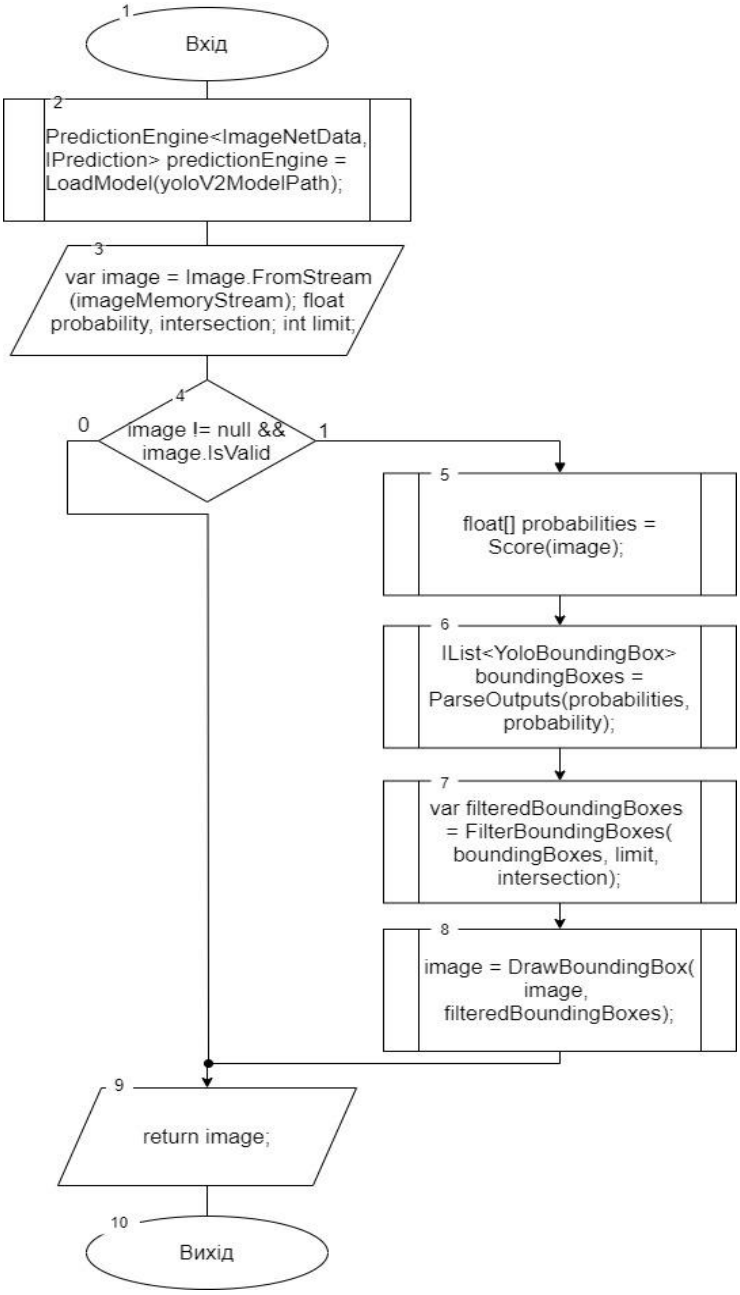

Рисунок Г.5 – Алгоритм роботи програми розпізнавання об’єктів

Рисунок Г.6 – Діаграма класів програмного забезпечення

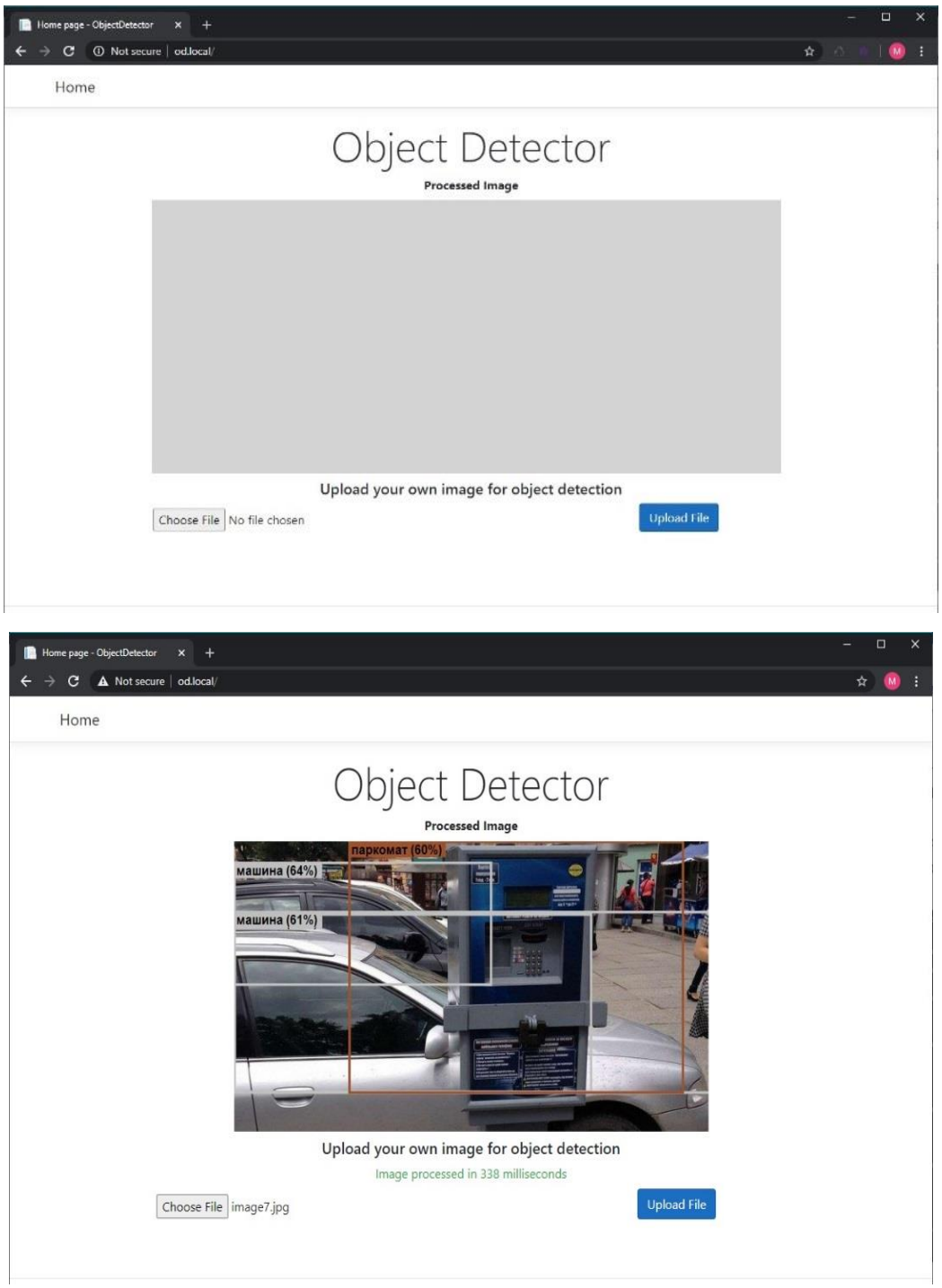

Рисунок Г.7 – Приклад роботи програмного модуля розпізнавання об’єктів