

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Розробка інтелектуальної системи виявлення та
класифікації об'єктів з використанням методів
глибокого навчання»**

Виконав студент групи ПЗм-23-2
БАЛАМУТ Дмитро Олегович

Керівник роботи:
ЛОБАНЧИКОВА Надія Миколаївна

Рецензент:
ЛОКТИКОВА Тамара Миколаївна

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«01» жовтня 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: БАЛАМУТ Дмитро Олегович

Керівник роботи: ЛОБАНЧИКОВА Надія Миколаївна

Тема роботи: **«Розробка інтелектуальної системи виявлення та класифікації об'єктів з використанням методів глибокого навчання»**,
затверджена Наказом закладу вищої освіти від «01» жовтня 2024 р. № 520/с
Вихідні дані для роботи: **методи та дослідження комп'ютерного зору для виявлення об'єктів, застосовуючи моделі глибокого навчання.**

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Лобанчикова Н.М.	3.10.2024р	3.10.2024р
2	Лобанчикова Н.М.	3.10.2024р	3.10.2024р
3	Лобанчикова Н.М.	3.10.2024р	3.10.2024р

РЕФЕРАТ

Кваліфікаційна робота присвячена розробці інтелектуальної системи виявлення та класифікації об'єктів з використанням методів глибокого навчання.

Завдання полягає в аналізі та виборі найбільш ефективних методів і архітектур нейронних мереж для розпізнавання об'єктів на зображеннях.

Кваліфікаційна робота складається з програмного комплексу, який включає в себе нейронну мережу, серверну частину, а також веб-інтерфейс та пояснювальної записки. Пояснювальна записка складається з вступу, трьох розділів, висновків, списку використаних джерел, який містить 27 найменувань, 29 рисунки, та 6 додатків.

Проаналізовано предметну область. Описано різні методи машинного навчання та архітектури нейронних мереж, що використовуються для задачі розпізнавання об'єктів. Проведено огляд сучасних підходів і сформульовано завдання на розробку програмного продукту. Розроблено та навчено нейронну мережу, яка здатна ефективно класифікувати об'єкти на зображеннях. Створено структуру веб-додатку, який складається з серверної та клієнтської частин, що забезпечують взаємодію користувача з нейронною мережею через зручний інтерфейс. Здійснено навчання системи комп'ютерного зору до отримання максимального результату. Порівняно основні показники навчання та проведено тестування нейронної мережі у вигляді веб-додатку, що показав позитивні результати у задачі класифікації об'єктів.

Висновок містить підсумок проведеного дослідження.

Ключові слова: НЕЙРОННА МЕРЕЖА, КОМП'ЮТЕРНИЙ ЗІР, PYTHON, МАШИННЕ НАВЧАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, YOLO.

ABSTRACT

The qualification work is dedicated to the development of an intelligent system for detecting and classifying objects using deep learning methods. The task is to analyze and select the most effective methods and neural network architectures for object recognition in images.

The qualification work consists of a software package, which includes a neural network, a server part, as well as a web interface and explanatory note. The explanatory note consists of an introduction, three chapters, conclusions, and a list of 27 references, 29 figures, and 6 appendices.

The subject area has been analyzed. Various machine learning methods and neural network architectures used for object recognition tasks have been described. A review of modern approaches was conducted, and the task for developing the software product was formulated. A neural network was developed and trained, capable of efficiently classifying objects in images. The structure of the web application was created, consisting of both server-side and client-side parts, ensuring user interaction with the neural network through an intuitive interface. The training of the computer vision system was carried out until optimal results were achieved. Key learning metrics were compared, and the neural network was tested in the form of a web application, showing positive results in the object classification task.

The conclusion contains a summary of the conducted research.

Keywords: NEURAL NETWORK, COMPUTER VISION, PYTHON, MACHINE LEARNING, ARTIFICIAL INTELLIGENCE, YOLO.

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	10
1.1. Огляд архітектур глибокого навчання	10
1.2. Підходи на основі сітки для виявлення об'єктів	21
1.3. Огляд існуючих бібліотек	24
Висновки до першого розділу	27
РОЗДІЛ 2. АЛГОРИТМ КОМП'ЮТЕРНОГО ЗОРУ	28
2.1. Вибір методів та архітектур для побудови системи	28
2.1.1. Метод виявлення об'єктів	28
2.1.2. Семантична сегментація зображень	32
2.1.3. Алгоритм відстеження об'єктів	36
2.2. Тренування модулі на спеціальному наборі даних	41
Висновки до другого розділу	47
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ	48
3.1. Платформа для розробки	48
3.2. Інструменти розробки	49
3.3. Розробка програмного забезпечення для обробки даних	54
3.4. Інтеграція та тестування моделі у веб-середовищі	63
3.5. Проєктування та тестування моделі у веб-застосунку	66
Висновки до третього розділу	70
ВИСНОВКИ	71
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	72
ДОДАТКИ	75
Додаток А	76
Додаток Б	80
Додаток В	84
Додаток Г	87
Додаток Ґ	88
Додаток Д	100

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CNN – Convolutional neural network

YOLO – You Only Live Once

SegNet – Segmentation Network

DeepSORT – Deep Simple Online and Realtime Tracking

ВСТУП

Комп'ютерний зір має великий потенціал для автоматизації та покращення різних процесів у різних сферах. Зокрема, застосування таких систем у транспорті дозволяє ефективно розв'язувати проблему безпеки та управління дорожнім рухом. Використання алгоритмів глибокого навчання для виявлення та відстеження об'єктів на дорогах дозволяє значно покращити контроль за ситуацією на дорогах та зменшити людську помилку при прийнятті рішень. Одним із важливих напрямків розвитку таких технологій є створення інтелектуальних транспортних систем, здатних автоматично розпізнавати об'єкти, відстежувати транспортні засоби та пішоходів, а також виявляти порушення правил дорожнього руху.

Актуальність теми. Використання систем комп'ютерного зору для розпізнавання світлофорів, пішохідних переходів, дорожніх смуг та відстеження транспортних засобів є важливим кроком до розвитку інтелектуальних транспортних систем, які здатні значно покращити ситуацію на дорогах. Системи, що використовують алгоритми машинного навчання, мають можливість адаптуватися до змінних умов дорожнього руху та точно реагувати на різні ситуації в реальному часі. Це робить їх ефективним інструментом для зниження аварійності та покращення якості управління транспортним потоком.

Метою дослідження є розробка інтелектуальної системи виявлення та класифікації об'єктів з використанням методів глибокого навчання. Система повинна дозволяти аналізувати ситуацію на дорозі в реальному часі, виявляти важливі об'єкти, такі як світлофори, пішохідні переходи та транспортні засоби, а також класифікувати їх для покращення безпеки дорожнього руху.

Встановлена мета обумовлює наступні завдання:

1. Провести аналіз та узагальнення архітектур глибокого навчання, методів, методів комп'ютерного зору для розпізнавання об'єктів на дорогах.
2. Розробити алгоритми та реалізувати систему для виявлення світлофорів, пішохідних переходів та дорожніх смуг.

3. Здійснити інтеграцію алгоритмів глибокого навчання для відстеження транспортних засобів та пішоходів.

4. Розробка веб-інтерфейсу для візуалізації результатів роботи системи.

5. Тестування та оптимізація розробленої системи для досягнення високої точності та швидкості в реальних умовах.

Об'єктом дослідження є технології побудови інтелектуальних систем, комп'ютерного зору та глибокого навчання для аналізу зображень і відео з транспортних засобів.

Предметом дослідження є методи, засоби та алгоритми побудови та програмної реалізації інтелектуальної системи виявлення та класифікації об'єктів з використанням методів глибокого навчання.

Наукова новизна

- проведений аналіз та узагальнення методів і архітектур для побудови інтелектуальної системи виявлення та класифікації об'єктів з використанням методів глибокого навчання;
- проведене інтегрування методів та алгоритмів для інтелектуального розпізнавання транспортних засобів з використанням технологій комп'ютерного зору для обробки інформації з відео;
- проведене інтегрування методів детекції об'єктів, таких як YOLOv4, а також сегментації з використанням моделей MobileNet і SegNet для підвищення ефективності розпізнавати об'єкти на дорозі, такі як світлофори, пішохідні переходи та транспортні засоби;
- інтеграція методів відстеження автомобілів (за допомогою алгоритму DeepSORT) та детекції пішоходів, що дозволило реалізувати функції виявлення порушень, таких як проїзд на червоне світло або рух без дотримання дорожніх смуг.

Практичне значення:

Практичне значення роботи полягає у програмній реалізації запропонованих інтеграційних методів. Розроблена система комп'ютерного

зору може бути використана для покращення безпеки на дорогах, забезпечення ефективного моніторингу та управління транспортними потоками, а також автоматизації процесу контролю за дотриманням правил дорожнього руху. Вона може знайти застосування у розробці інтелектуальних транспортних систем, що використовуються в містах та на автомагістралях.

Тези були відправлені на VIII Всеукраїнську науково-практичну конференцію 'Нові інформаційні технології управління бізнесом', яка відбудеться 12 лютого 2025 року.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Огляд архітектур глибокого навчання

Сьогодні глибоке навчання є підгалуззю машинного навчання, яка фокусується на використанні багат шарових нейронних мереж для аналізу складних даних.

Глибоке навчання – це не єдиний підхід, а скоріше клас алгоритмів і топологій, які можна застосовувати до широкого спектру проблем. Архітектури глибокого навчання є основою для багатьох сучасних технологій штучного інтелекту (ШІ), таких як розпізнавання образів, автоматичний переклад, обробка природної мови, а також для задач, пов'язаних із класифікацією та виявленням об'єктів [1]. Вони існують вже понад семидесяти років, але нові архітектури та графічні процесори (GPU) вивели їх на передові позиції штучного інтелекту.

Хоча глибоке навчання, звичайно, не є новим, воно переживає вибухове зростання завдяки поєднанню глибоко шаруватих нейронних мереж і використання GPU для прискорення їх виконання. Великі дані також сприяли цьому зростанню.

Оскільки глибоке навчання ґрунтується на тренуванні нейронних мереж з використанням зразків даних і винагородженні їх на основі їхнього успіху, то чим більше даних, тим краще для побудови структур глибокого навчання.

Кількість архітектур і алгоритмів, що використовуються в глибокому навчанні, є широкою та різноманітною [2]. Цей розділ досліджує шість архітектур глибокого навчання, що охоплюють останні двадцять років. Зокрема, згорткові нейронні мережі (CNN) є із найстаріших підходів у цьому списку, але також і із найбільш використовуваних у різних застосуваннях.

Рисунок 1.1 показує архітектуру глибокого навчання на навчання з учителем та навчання без учителя, та представляє кілька популярних архітектур глибокого навчання [1].

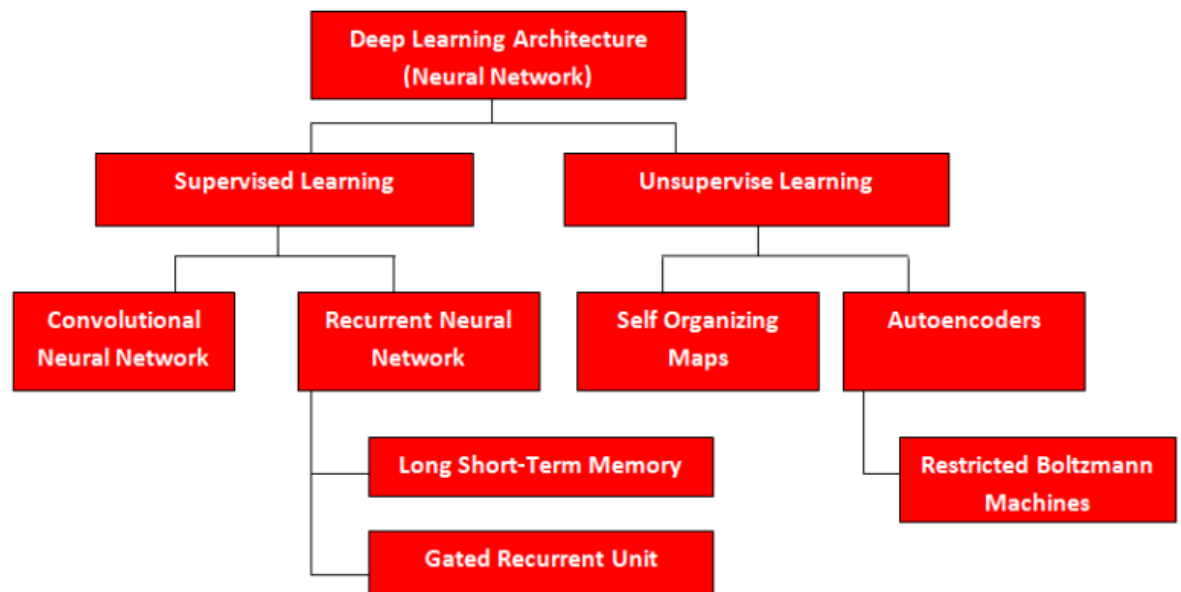


Рис. 1.1. Архітектура глибокого навчання

Навчання з учителем (Supervised learning) відноситься до проблемної області, де ціль, яку потрібно передбачити, чітко позначена в даних, що використовуються для навчання. У цьому розділі представляю дві найпопулярніші архітектури глибокого навчання з учителем – згорткові нейронні мережі та рекурентні нейронні мережі [3].

Зазвичай згорткові нейронні мережі використовуються для завдань комп'ютерного зору, але їх також можна використовувати для аналізу інших задач. Згорткова нейронна мережа (CNN) – це багатошарова нейронна мережа, біологічно натхненна зоровою корою головного мозку тварин. Ця архітектура особливо корисна в застосунках обробки зображень.

Першу CNN створив Ян ЛеКун. На той час архітектура була зосереджена на розпізнаванні рукописних символів, таких як інтерпретація поштових індексів. Як глибока мережа, ранні шари розпізнають особливості (такі як краї), а пізніші шари рекомбінують ці особливості у атрибути вищого рівня вхідних даних.

Архітектура LeNet CNN складається з кількох шарів, які реалізують вилучення ознак, а потім класифікацію (див. рис.1.2) [2]. Зображення ділиться

на рецептивні поля, які подаються на згортковий шар, який потім вилучає ознаки з вхідного зображення.

Наступним кроком є пулінг, який зменшує розмірність вилучених ознак (шляхом пониження дискретизації), зберігаючи при цьому найважливішу інформацію (зазвичай, за допомогою максимального пулінгу). Потім виконується ще один крок згортки та пулінгу, який подається на повністю зв'язаний багатошаровий перцептрон. Останній вихідний шар цієї мережі — це набір вузлів, які ідентифікують ознаки зображення (в даному випадку, вузол на кожне ідентифіковане число). Ви навчаєте мережу за допомогою зворотного поширення помилки.

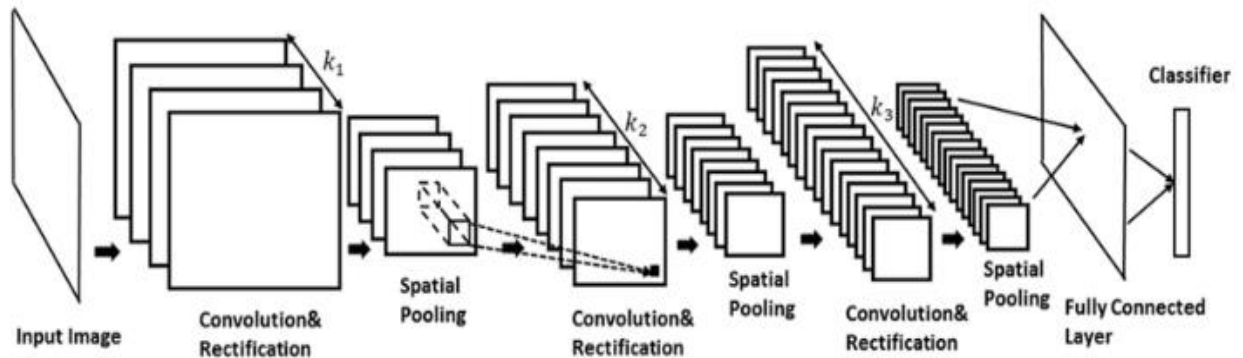


Рис. 1.2. Діаграма згорткової нейронної мережі (CNN)

Використання глибоких шарів обробки, згортки, пулінгу та повністю зв'язаного шару класифікації відкрило двері для різних нових застосувань нейронних мереж глибокого навчання [3]. Окрім обробки зображень, CNN успішно застосовується для розпізнавання відео та різних завдань у сфері обробки природної мови.

Рекурентна нейронна мережа (RNN) є однією з фундаментальних архітектур мереж, на основі якої будуються інші архітектури глибокого навчання (див. рис.1.3). Основна відмінність між типовою багатошаровою мережею та рекурентною мережею полягає в тому, що замість повністю прямого зв'язку, рекурентна мережа може мати зв'язки, які повертаються до попередніх шарів (або до того ж самого шару). Цей зворотний зв'язок дозволяє RNN зберігати пам'ять про минулі вхідні дані та моделювати задачі в часі.

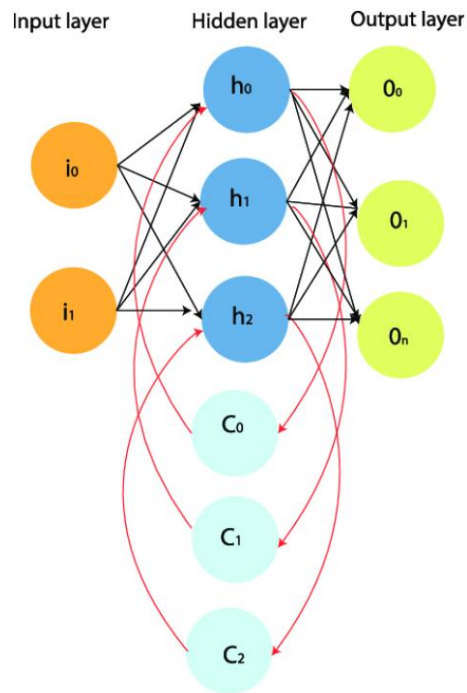


Рис. 1.3. Діаграма рекурентна нейронна мережа (RNN)

RNN складаються з багатого набору архітектур. Ключовою відмінністю є зворотний зв'язок всередині мережі, який може проявлятися від прихованого шару, вихідного шару або деякої їх комбінації [6].

LSTM було розроблено у 1997 році, проте за останні роки ця архітектура RNN набула значної популярності завдяки її застосуванню в різних галузях. Наприклад, LSTM використовується у смартфонах та інших звичних продуктах. Відмінність LSTM від традиційних нейронних мереж полягає у впровадженні концепції комірки пам'яті (див. рис. 1.4). Ця комірка здатна зберігати значення як на короткий, так і на тривалий час залежно від вхідних даних, що дозволяє запам'ятовувати важливу інформацію, а не лише останнє обчислене значення. Комірка пам'яті LSTM містить три ворота (гейти), які керують потоками інформації. Вхідні ворота (гейти) визначають, коли нові дані можуть потрапити до комірки. Ворота забуття вирішують, коли слід видалити існуючу інформацію, щоб звільнити місце для нових даних. Вихідні ворота контролюють, коли інформація з комірки використовується у вихідному сигналі. Комірка також має ваги, які впливають на роботу воріт.

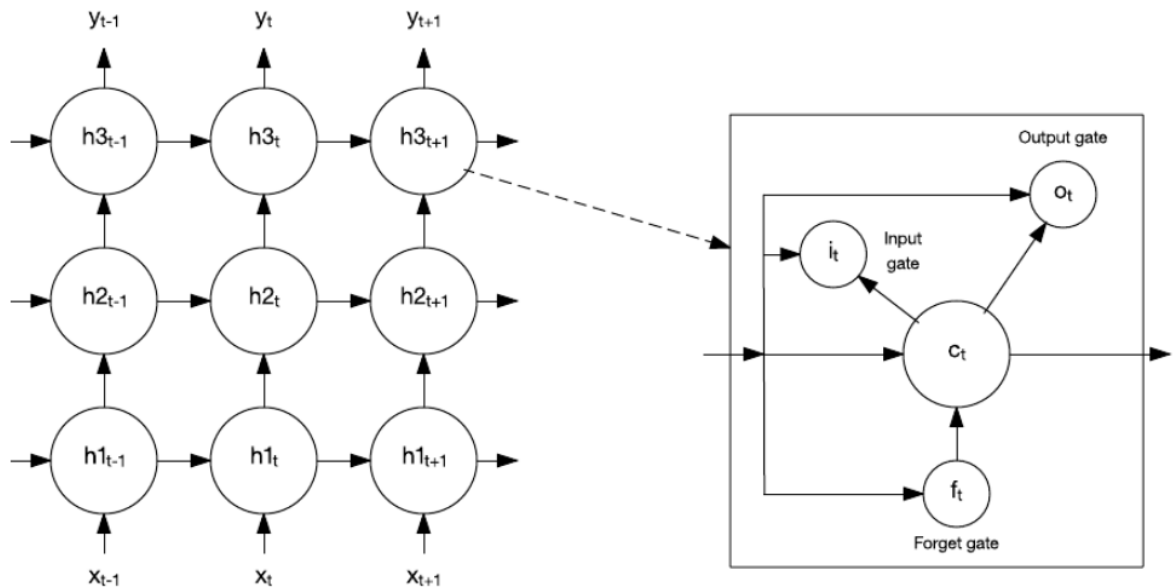


Рис. 1.4. Мережі LSTM

Алгоритм навчання, зазвичай BPTT, оптимізує ці ваги залежно від помилок, отриманих на виході мережі.

Останнім часом використання CNN і LSTM дозволило створити системи автоматичного підписування зображень та відео, [7] які формують підписи природною мовою. У таких системах CNN виконує обробку зображень або відео, а LSTM навчається перетворювати результати CNN у текстовий опис природною мовою.

У 2014 році було запропоновано спрощення архітектури LSTM під назвою "згортковий рекурентний блок" (GRU). Ця модель має лише два ворота, виключаючи вихідні ворота, що є в LSTM. Ворота GRU складаються з воріт оновлення та скидання.

Ворота оновлення визначають, яку частину вмісту попередньої комірки слід зберегти. Ворота скидання відповідають за те, як нові вхідні дані поєднуються з попереднім вмістом комірки [6]. GRU може моделювати стандартну RNN, якщо встановити значення воріт скидання на 1, а воріт оновлення — на 0.

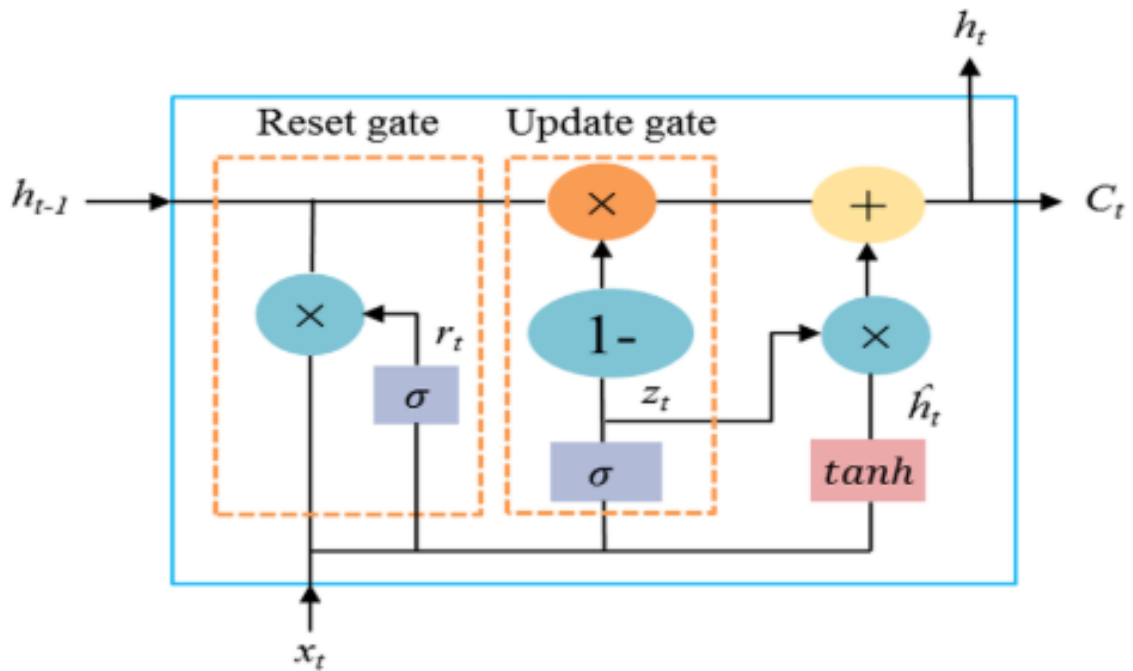


Рис. 1.5. Мережа GRU

GRU є простішою моделлю порівняно з LSTM, її навчання відбувається швидше, а виконання є більш ефективним. Проте LSTM має вищу здатність до вираження складних залежностей, і за наявності більшого обсягу даних може забезпечувати кращі результати.

Навчання без учителя (Unsupervised learning) [3] відноситься до проблемної області, де в даних, що використовуються для навчання, немає цільової мітки. Розглянемо дві архітектури глибокого навчання без учителя: самоорганізуючі карти (SOM) та автокодувальники [2]. Також розглянемо, як будуються глибокі мережі переконань та глибокі мережі стекування на основі базової архітектури без учителя.

Самоорганізуюча карта (SOM) була винайдена доктором Теуво Кохоненом у 1982 році і була широко відома як карта Кохонена. SOM – це нейронна мережа без учителя, яка створює кластери вхідного набору даних шляхом зменшення розмірності вхідних даних (див. рис.1.6). SOM відрізняються від традиційної штучної нейронної мережі кількома способами.

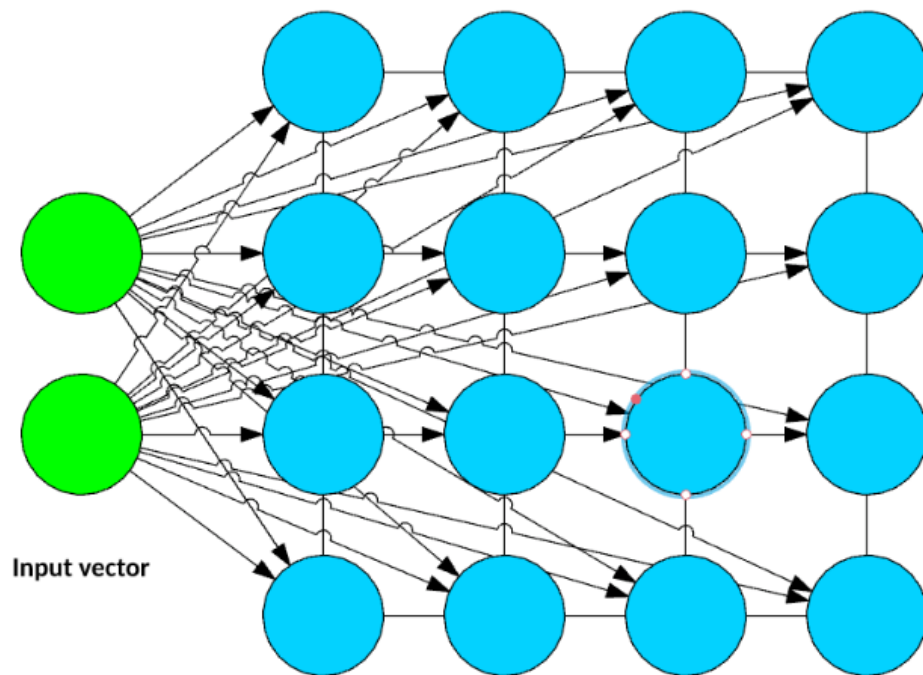


Рис. 1.6. Діаграма самоорганізуюча карта (SOM)

Перша відмінність полягає в тому, що ваги служать характеристикою вузла. Після нормалізації вхідних даних спочатку обирається випадковий вхід. Випадкові ваги, близькі до нуля, ініціалізуються для кожної ознаки вхідного запису. Ці ваги тепер представляють вхідний вузол. Кілька комбінацій цих випадкових ваг представляють варіації вхідного вузла.

Обчислюється евклідова відстань між кожним з цих вихідних вузлів та вхідним вузлом. Вузол з найменшою відстанню оголошується найбільш точним представленням входу та позначається як найкраща відповідна одиниця або ВМУ [3]. З цими ВМУ як центральними точками, інші одиниці обчислюються аналогічно та призначаються до кластера, від якого вони знаходяться на найменшій відстані.

Радіус точок навколо ваг ВМУ оновлюється на основі близькості. Радіус зменшується. Далі, в SOM не застосовується функція активації, і оскільки немає цільових міток для порівняння, немає поняття обчислення помилки та зворотного поширення. Хоча історія винайдення автокодувальників є нечіткою, перше відоме використання автокодувальників було знайдено у ЛеКуна в 1987 році. Цей варіант ШНМ (штучної нейронної мережі)

складається з 3 шарів: вхідного, прихованого та вихідного шарів, (див. рис.1.7).

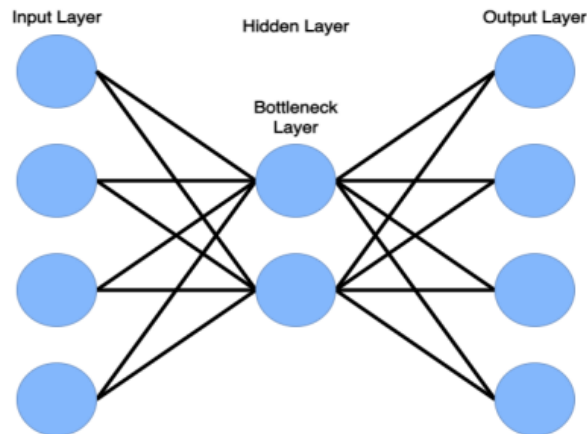


Рис. 1.7. Діаграма Autoencoders

Спочатку вхідний шар кодується в прихований шар з використанням відповідної функції кодування. Кількість вузлів у прихованому шарі значно менша, ніж кількість вузлів у вхідному шарі. Цей прихований шар містить стиснене представлення оригінального входу.

Вихідний шар має на меті реконструювати вхідний шар за допомогою функції декодера. Під час фази навчання різниця між вхідним та вихідним шаром обчислюється за допомогою функції помилки, і ваги коригуються для мінімізації помилки.

На відміну від традиційних методів навчання без учителя, де немає даних для порівняння з виходами, автокодувальники навчаються безперервно, використовуючи зворотне поширення [3]. З цієї причини автокодувальники класифікуються як самокеровані алгоритми.

Глибоке навчання представлене спектром архітектур, які можуть створювати рішення для широкого кола проблемних областей. Ці рішення можуть бути зосереджені на прямому поширенні або рекурентними мережами, що дозволяють враховувати попередні вхідні дані.

Обмежена машина Больцмана (RBM) є двошаровою нейронною мережею, що складається з вхідного шару та прихованого шару. Як показано на діаграмі (див.рис.1.8), в RBM кожен вузол прихованого шару з'єднаний з

кожним вузлом видимого (вхідного) шару. У традиційній машині Больцмана вузли в межах вхідного та прихованого шарів також мають з'єднання між собою. Проте через високу обчислювальну складність в Обмеженій машині Больцмана такі внутрішньшарові з'єднання відсутні [10].

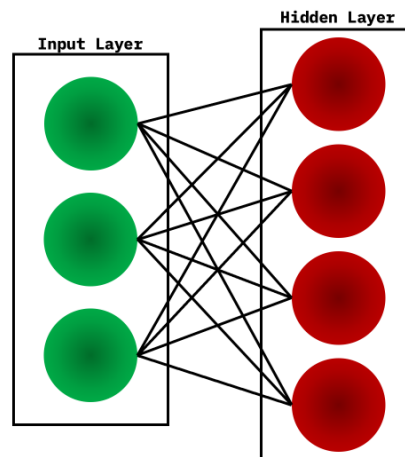


Рис. 1.8. Діаграма RBM

Під час навчання Обмежені машини Больцмана (RBM) використовують стохастичний підхід для розрахунку ймовірнісного розподілу тренувального набору даних. На початку навчання кожен нейрон активується випадковим чином. Модель також має приховані та видимі зсуви (bias).

Прихований зсув використовується для побудови активації під час прямого проходу, а видимий зсув допомагає у реконструкції вхідних даних. Оскільки в RBM реконструйовані дані завжди відрізняються від оригінальних, ці моделі також відомі як генеративні. Через вбудовану випадковість одні й ті самі передбачення можуть давати різні результати. Це є ключовою відмінністю від автокодувальника, який є детерміністичною моделлю.

DBN (глибинна мережа Больцмана) є типовою архітектурою мережі, але з новим алгоритмом навчання. DBN — це багатошарова мережа (зазвичай глибока та містить багато прихованих шарів), в якій кожна пара з'єднаних шарів є RBM (див. рис.1.9). Таким чином, DBN представляється як стопка з RBM. У DBN вхідний шар відповідає за сирі сенсорні дані, а кожен прихований шар навчається абстрактним уявленням цього вводу. Вихідний шар, який обробляється дещо інакше, ніж інші шари, виконує класифікацію

мережі. Навчання відбувається в два етапи: неспостережне попереднє навчання (pretraining) та контрольоване доопрацювання (fine-tuning).

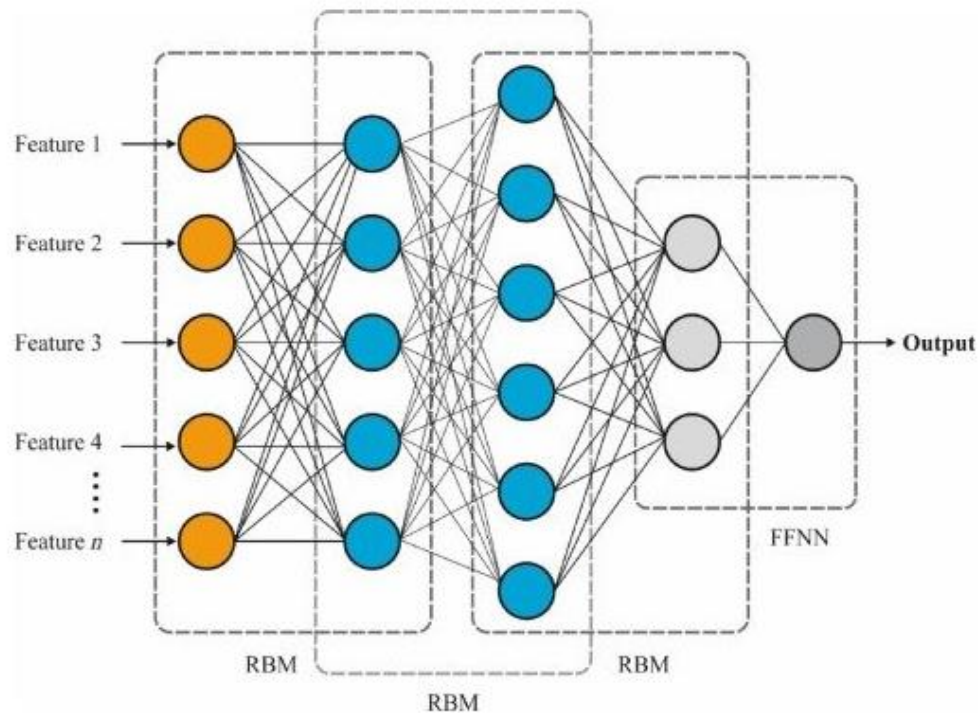


Рис. 1.9. Діаграма RBM

У процесі неспостережного попереднього навчання кожен RBM навчається реконструювати свій вхід (наприклад, перший RBM реконструює вхідний шар у перший прихований шар). Наступний RBM навчається аналогічно, але перший прихований шар розглядається як вхідний (видимий) шар [12], і RBM тренується, використовуючи виходи з першого прихованого шару як входи. Цей процес триває, поки не буде попередньо навчений кожен шар.

Коли попереднє навчання завершено, починається етап доопрацювання. На цьому етапі вихідним вузлам присвоюються мітки, щоб надати їм значення (що вони представляють у контексті мережі). Потім застосовується повне навчання мережі, яке здійснюється за допомогою або методу градієнтного спуску, або зворотного поширення помилок для завершення процесу навчання.

Остаточна архітектура – це DSN, або глибока випукла мережа. DSN відрізняється від традиційних архітектур глибокого навчання тим, що, хоча

вона складається з глибокої мережі, насправді це набір окремих мереж, кожна з яких має свої власні приховані шари.

Ця архітектура є відповіддю на одну з проблем глибокого навчання — складність навчання. Кожен шар у глибокій архітектурі експоненційно збільшує складність навчання, тому DSN розглядає навчання не як єдину задачу, а як набір окремих задач навчання. DSN складається з набору модулів, кожен з яких є підмережею в загальній ієрархії DSN. У одній реалізації цієї архітектури створюються три модулі для DSN [15].

Кожен модуль складається з вхідного шару, одного прихованого шару та вихідного шару. Модулі розташовуються один над одним, де вхід модуля складається з виходів попереднього шару та оригінального вектору вводу. Таке шарування дозволяє загальній мережі навчатися складнішим класифікаціям, ніж це було б можливо за допомогою одного модуля.

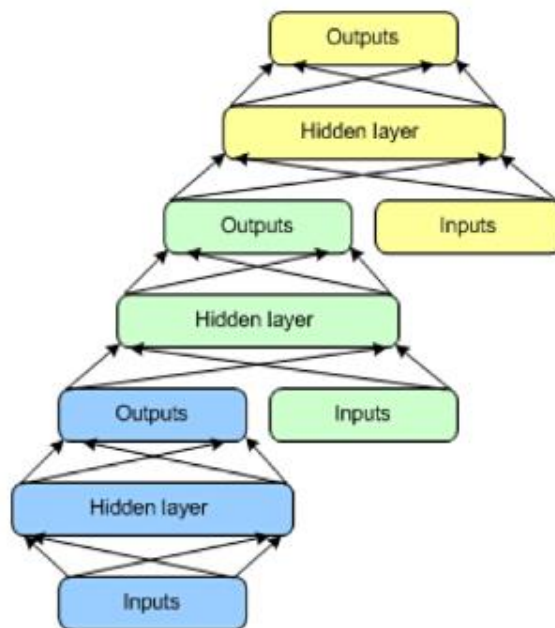


Рис. 1.10. Діаграма RBM

DSN дозволяє навчати окремі модулі ізольовано, що робить цей процес ефективним завдяки можливості паралельного навчання. Навчання з учителем реалізується як зворотне поширення помилки для кожного модуля, а не для всієї мережі. Для багатьох задач DSN може показувати кращі результати, ніж типові DBN, що робить їх популярною та ефективною архітектурою мереж.

1.2. Підходи на основі сітки для виявлення об'єктів

В комп'ютерному зорі однією з ключових проблем є ідентифікація та локалізація предметів на візуальних даних, таких як зображення та відео.

Сучасні методи глибокого навчання часто використовують архітектури, що базуються на нейромережах, включаючи згорткові нейромережі (CNN), а також більш розвинені моделі, що інтегрують мережеві або регіональні підходи для оптимізації процесу виявлення об'єктів.

Серед найбільш ефективних підходів для виявлення об'єктів є методи, що базуються на сітці, зокрема ті, що використовують згорткові нейромережі (CNN) для розбиття зображень на регулярні або нерегулярні сітки. Кожен елемент сітки відповідає за виявлення об'єктів, які потрапляють у його межі. Такі підходи дозволяють одночасно розпізнавати та локалізувати кілька об'єктів на одному зображенні.

Одним із найбільш популярних методів є архітектура YOLO (You Only Look Once), яка реалізує концепцію сітки для виявлення об'єктів. Цей підхід розбиває зображення на фіксовану сітку та одночасно виконує ідентифікацію об'єктів у кожній клітинці сітки. Завдяки такій архітектурі YOLO здатний ефективно виконувати детекцію в реальному часі, що особливо важливо для застосувань в інтелектуальних транспортних системах.

Вперше представлена у 2016 році, YOLO здійснила революцію в цій галузі, запропонувавши виявлення об'єктів у реальному часі з вражаючою швидкістю та точністю.

На відміну від традиційних методів, які обробляють зображення в кілька етапів, YOLO розглядає виявлення об'єктів як єдину задачу регресії, прогножуючи обмежувальні рамки та ймовірності класів безпосередньо з повних зображень за одну оцінку. Цей підхід значно підвищує швидкість та ефективність, дозволяючи YOLO обробляти зображення з вражаючою частотою кадрів, зберігаючи при цьому високу точність.

Архітектура YOLO побудована на основі згорткових нейронних мереж (CNN), і вона еволюціонувала через різні версії, кожна з яких покращувала попередню з точки зору продуктивності, точності та адаптивності до різних середовищ і завдань у комп'ютерному зорі.

Виявлення об'єктів за допомогою YOLO докорінно змінило те, як машини інтерпретують візуальні дані [20], пропонуючи потужне та ефективне рішення для ідентифікації та класифікації об'єктів на зображеннях.

В основі YOLO лежить спрощення процесу виявлення об'єктів шляхом його зведення до єдиної задачі регресії. Замість багаторазової обробки зображення для ідентифікації об'єктів, YOLO ділить зображення на сітку та прогнозує обмежувальні рамки, ймовірності класів та показники впевненості для кожної клітинки сітки за один прохід.

Цей оптимізований підхід дозволяє YOLO досягати швидкості виявлення в реальному часі, що робить його ідеальним для застосувань, які потребують швидкого та точного розпізнавання об'єктів, таких як автономне водіння, спостереження та робототехніка.

З розвитком YOLO він включав у себе все більш складні методи для покращення як швидкості, так і точності.

Ранні версії, такі як YOLOv1, були зосереджені на базовому прогнозуванні на основі сітки, але наступні ітерації запровадили нововведення, такі як опорні рамки, покращені мережі вилучення ознак та багатомасштабне виявлення, що значно покращило продуктивність. Пізніші версії, такі як YOLOv4 та YOLOv5, ще більше оптимізували архітектуру, інтегрувавши передові концепції, такі як часткові мережі між етапами (CSPNet) та мережі агрегації шляхів (PANet), щоб збалансувати швидкість з високою точністю. Ці досягнення зробили модель YOLO універсальним інструментом, здатним впоратися з широким спектром реальних завдань у виявленні об'єктів, від виявлення дрібних об'єктів у захаращених сценах до підтримки продуктивності на різних наборах даних.

Існують основні ключові особливості:

Виявлення за один прохід – на відміну від традиційних алгоритмів виявлення об'єктів, які потребують кількох проходів по зображенню, моделі YOLO розглядають виявлення об'єктів як єдину задачу регресії. Вони прогнозують обмежувальні рамки та ймовірності класів за одну оцінку, що значно скорочує час обчислень;

Обробка в реальному часі – YOLO розроблено для виявлення об'єктів у реальному часі, обробляючи зображення з високою частотою кадрів без шкоди для точності [19]. Це робить його ідеальним для застосувань, де швидкість є критичною, таких як автономні транспортні засоби та аналіз живого відео;

Уніфікована архітектура – архітектура YOLO базується на єдиній згортковій нейронній мережі (CNN), яка обробляє як вилучення ознак, так і виявлення об'єктів. Цей уніфікований підхід контрастує з моделями, які розділяють ці завдання на різні мережі, що призводить до більш ефективного та оптимізованого процесу;

Прогнозування на основі сітки – YOLO ділить зображення на сітку та призначає кожній клітинці сітки завдання прогнозування об'єктів у межах її кордонів. Цей підхід спрощує процес виявлення та дозволяє одночасно виявляти кілька об'єктів у різних місцях зображення.

YOLO складається з основних компонентів та функціональних можливостей, які сприяють його швидкості та точності виявлення об'єктів:

Прогнозування на основі сітки – YOLO ділить вхідне зображення на сітку клітинок [16]. Кожна клітинка відповідає за прогнозування обмежувальних рамок та пов'язаних з ними ймовірностей класів для об'єктів, які потрапляють в її межі. Цей підхід на основі сітки дозволяє YOLO виявляти кілька об'єктів у різних місцях зображення;

Основа згорткової нейронної мережі (CNN) – YOLO використовує CNN для вилучення ознак з вхідних зображень. Базова мережа, така як Darknet у YOLOv3, обробляє зображення через кілька шарів, захоплюючи ієрархічні ознаки, які є вирішальними для виявлення об'єктів у різних масштабах;

Виявлення за один прохід – На відміну від традиційних методів, які потребують кількох етапів обробки, виявлення YOLO працює за один прохід через мережу. Це включає прогнозування обмежувальних рамок та ймовірностей класів безпосередньо з усього зображення, що робить YOLO надзвичайно швидким;

Обмежувальні рамки та опорні рамки – YOLO прогнозує обмежувальні рамки навколо об'єктів разом з показниками впевненості, які вказують на ймовірність того, що рамка містить об'єкт. У пізніших версіях YOLO було введено опорні рамки, попередньо визначені форми рамок, які допомагають покращити точність виявлення, особливо для об'єктів різних розмірів.

Підходи на основі сітки є важливими інструментами для створення високопродуктивних систем комп'ютерного зору, здатних ефективно і точно виявляти та класифікувати об'єкти в різних умовах. Вони активно застосовуються в інтелектуальних транспортних системах для покращення безпеки дорожнього руху, моніторингу та управління транспортними потоками.

1.3. Огляд існуючих бібліотек

Для реалізації проєкту необхідно визначитися з мовою програмування. Одним із потужних і популярних, також і один із простих мов програмування є Python.

Python підтримується на всіх основних операційних системах та платформах, що робить її зручним інструментом для розробки в різних середовищах. Мова популярна із-за її гнучкості та універсальності, яка може конкурувати з іншими мовами програмування, такі як C++, Java та інші.

Python є однією з найпопулярніших мов програмування, яка здобула величезне визнання серед розробників завдяки своїй простоті, зручності та потужності. Вона активно використовується в різноманітних галузях, таких як веб-розробка, наука про дані, штучний інтелект, автоматизація, розробка програмного забезпечення та багато інших. Python має дуже зрозумілий і

легкий для освоєння синтаксис. Це робить мову ідеальним вибором для новачків, які тільки починають вивчати програмування, а також для досвідчених розробників, які шукають швидкий спосіб реалізувати свої ідеї. Python має величезну кількість бібліотек, що дозволяє швидко вирішувати різноманітні задачі. Для обробки даних широко використовуються бібліотеки як-от NumPy, Pandas, Matplotlib, а для машинного навчання — TensorFlow, PyTorch, scikit-learn та інші.

Python став основною мовою для роботи з великими даними та розробки моделей машинного навчання завдяки потужним бібліотекам, таким як scikit-learn, TensorFlow, Keras, PyTorch та XGBoost.

За допомогою Python можна створювати веб-додатки за допомогою фреймворків, таких як Django і Flask. Python пропонує простоту інтеграції з іншими технологіями, такими як бази даних (MySQL, PostgreSQL) і інтерфейси API.

Основні характеристики Python:

Має простий синтаксис, що робить її легкою для вивчення, особливо для початківців;

Виконується інтерпретатором, що означає, що код виконується построчно, без необхідності компіляції в машинний код перед запуском; Об'єктно-орієнтована;

Працює на різних операційних системах, що дозволяє запускати один і той же код на різних платформах.

Python через свою популярність та зручність використання широко використовується в різних областях, таких як наука про дані та машинне навчання, веб-розробка, розробка ігор, автоматизація та скриптинг, розробка програмного забезпечення, аналіз даних та візуалізація.

Python постачається з великою кількістю вбудованих модулів та функцій, що охоплюють широкий спектр завдань. Python є популярною мовою для задач машинного навчання та комп'ютерного зору завдяки своїй простоті

та має величезну екосистему сторонніх бібліотек, які розширюють його функціональні можливості, зокрема:

TensorFlow – це фреймворк для машинного навчання, розроблений Google. Надає інструменти для створення, навчання та розгортання моделей глибокого навчання. `tensorflow-estimator` та `tensorflow-gpu` є компонентами TensorFlow, що відповідають за оцінку моделей та використання GPU для прискорення обчислень.

Keras – високорівневий API для нейронних мереж, що працює поверх TensorFlow (або інших бекендів). Спрощує процес створення та навчання моделей глибокого навчання.

scikit-learn – бібліотека для машинного навчання, що містить алгоритми для класифікації, регресії, кластеризації, зменшення розмірності та інших задач. Може використовуватись для попередньої обробки даних, оцінки моделей та інших задач машинного навчання.

NumPy – бібліотека для наукових обчислень з Python, що надає підтримку багатовимірних масивів та матриць, а також широкий набір математичних функцій. Використовується для ефективної роботи з числовими даними, що є основою для обробки зображень та машинного навчання.

scikit-image – бібліотека для обробки зображень, що надає широкий набір інструментів для фільтрації, сегментації, геометричних перетворень, аналізу зображень та багато іншого. Використовується для попередньої обробки зображень, аналізу форми об'єктів та інших задач, пов'язаних з обробкою візуальних даних.

Matplotlib – бібліотека для візуалізації даних, що дозволяє створювати графіки, діаграми та інші візуальні представлення даних. Може використовуватись для візуалізації результатів роботи моделей та аналізу даних.

Оглянувши бібліотеки можна зазначити, що кращі для рішення поставленої задачі підійдуть TensorFlow/Keras.

1.4. Висновки до першого розділу

Було проведено аналіз предметної області, що охоплює огляд архітектур глибокого навчання, підходів на основі сітки для виявлення об'єктів та існуючих бібліотек для реалізації проєкту інтелектуальної транспортної системи.

Розглянуто основні типи архітектур глибокого навчання, зокрема навчання з учителем та навчання без учителя. Підкреслено важливість згорткових нейронних мереж (CNN) для задач комп'ютерного зору та їх еволюцію.

Особливу увагу було приділено підходам на основі сітки для виявлення об'єктів, де ключовим методом є YOLO.

Обґрунтовано вибір мови програмування Python як основної для проєкту, завдяки її простоті, великій кількості бібліотек для машинного навчання та комп'ютерного зору, а також кросплатформенності.

РОЗДІЛ 2. АЛГОРИТМ КОМП'ЮТЕРНОГО ЗОРУ

2.1. Вибір методів та архітектур для побудови системи

Проведемо аналіз та виберемо основні методи для роботи.

2.1.1. Метод виявлення об'єктів

Однією із задач роботи було обрати для створення системи виявлення та класифікації об'єктів для опису методів та архітектури. Особлива увага приділяється глибокому навчанню, технічним аспектам, а також методам комп'ютерного зору.

Під час вибору архітектури для задачі виявлення об'єктів я розглядав декілька сучасних моделей, включаючи найновішу версію YOLOv11. Однак, я зупинив свій вибір на YOLOv4, оскільки вона відповідає моїм дослідницьким цілям та технічним можливостям, а саме:

YOLOv4 демонструє оптимальне співвідношення між продуктивністю (до 60 кадрів за секунду) та точністю (високий mAP). Це робить її придатною для застосувань у реальному часі.

Є добре вивченою і широко використовуваною моделлю, що має значну кількість наукових робіт та прикладних досліджень. Я хотів краще зрозуміти її архітектуру та можливості.

Має менші вимоги до апаратного забезпечення порівняно з YOLOv11, що дозволило мені використовувати доступні GPU для навчання та тестування.

Хоча YOLOv11 забезпечує вищу точність, її обчислювальна складність та вимоги до обладнання виявилися менш придатними для мого проєкту. Водночас, YOLOv4 дозволяє глибше зрозуміти основи архітектури YOLO, які є спільними для всіх її версій.

Для реалізації системи буду використовувати такі бібліотеки: TensorFlow та Keras – побудова нейронних мереж; OpenCV – обробка

зображень та інтеграції комп'ютерного зору; cuDNN – оптимізація роботи GPU під час навчання.

YOLOv4 використовує низку новітніх функцій, які разом підвищують його ефективність. До них належать зважені залишкові з'єднання (WRC), часткові з'єднання на перехресних етапах (CSP) [13], нормалізація міні-пакетів (CmBN), самонавчання через змагальну підготовку (SAT), активація Mish, мозаїчне збільшення даних, регуляризація DropBlock та втрата CIoU. Поєднання цих компонентів дозволяє досягти передових результатів.

Зазвичай детектор об'єктів складається з декількох частин: вхідних даних, основної мережі (backbone), проміжної ланки (neck) та вихідної частини (head). Основна мережа YOLOv4, попередньо навчена на ImageNet, використовується для прогнозування класів і координат об'єктів (див. рис.2.1). Для неї можуть застосовуватися такі моделі, як VGG, ResNet, ResNeXt або DenseNet. Проміжна ланка збирає карти ознак з різних етапів і зазвичай включає кілька шляхів "знизу вгору" та "зверху вниз". Вихідна частина відповідає за фінальні передбачення об'єктів і їх класифікацію.

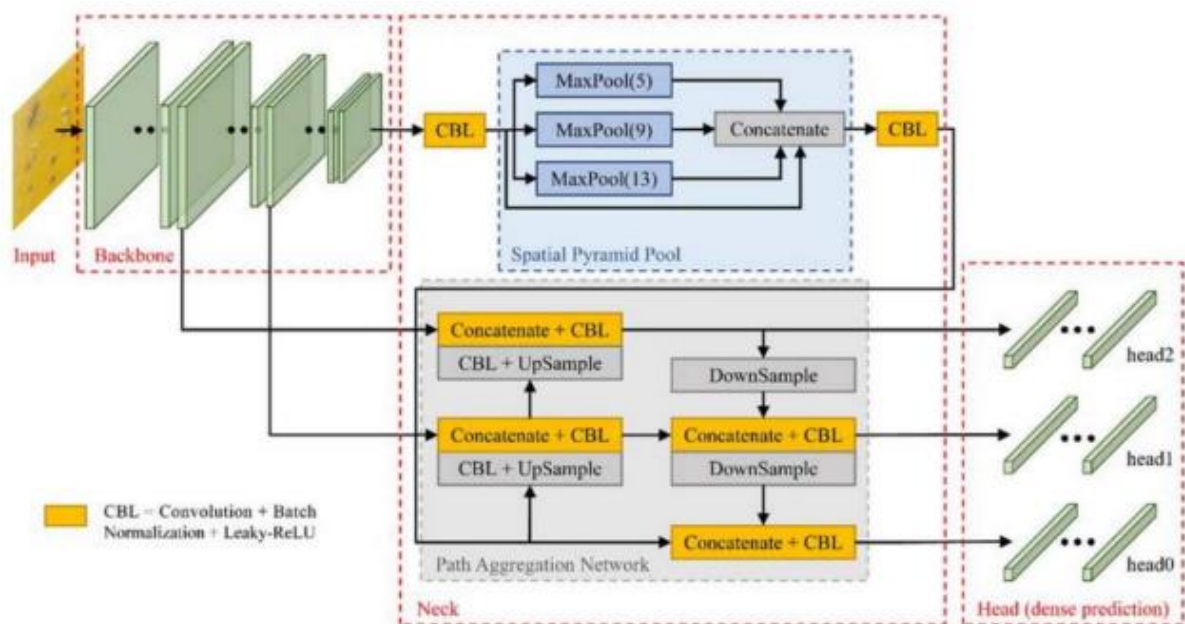


Рис. 2.1. Архітектура YOLOv4

Існують два типи моделей: одностадійні та двостадійні детектори об'єктів. У двостадійних детекторах процес детекції складається з двох етапів:

спочатку визначаються регіони інтересу, а потім ці регіони класифікуються для визначення наявності об'єкта. YOLOv4 [13], як одностадійний детектор об'єктів, є більш точним і швидким, ніж двостадійні детектори, такі як R-CNN і Fast R-CNN.

YOLOv4 створена на основі ідеї одноетапного підходу, що дозволяє одночасно визначати координати об'єктів на зображенні та їхні категорії.

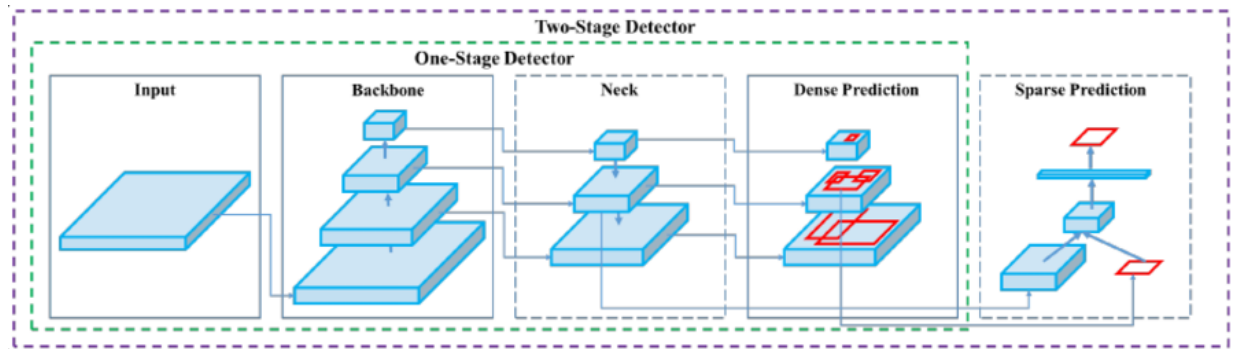


Рис. 2.2 Архітектура моделей об'єктного детектування

Як показано на рисунку 2.2, сучасні об'єктні детектори зазвичай мають чотири основні частини:

Input (вхідні дані) зображення, без особливих пояснень. **Backbone** (основа): використовується для попереднього витягу характеристик із зображень. Популярні моделі включають VGG16, ResNet, EfficientNet, CSPDarknet53 тощо. Зазвичай ці моделі попередньо тренуються на ImageNet. **Neck** (нейронна «шия»): інтегрує карти характеристик із різних шарів backbone. Популярні моделі: SPP, FPN, PAN тощо.

Head (голова): отримує об'єднані характеристики з neck і використовується для передбачення обмежувальних рамок (bounding box, bbox).

Голови поділяються на два типи: одностадійні (dense): прогнозують наявність bbox у кожній сітці.

Двостадійні (sparse): завдяки ROI pooling виконують прогнозування тільки для областей інтересу (ROI).

Основна архітектура YOLOv4 включає наступне: Backbone (CSPDarknet53), Neck (SPP + PAN), Head (YOLO).

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1x	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2x	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8x	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8x	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4x	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Рис. 2.3 Схеми Darknet53

Як показано на рисунку 3.2, Darknet53 складається з 53 шарів згортки (conv. layers). Усі вони враховані, включаючи останній повнозв'язний шар (FC), який реалізований через згортку 1×1. Власне, 52 шари служать для основної мережі.

Кожен згортковий шар містить: Conv2D (2D-згортка); BatchNormalization (нормалізація батчу), LeakyReLU (у YOLOv4 використовується Mish як активаційна функція).

Вхідний розмір даних для Darknet53 зазвичай становить 256×256, хоча у статті використовували розмір 512×512.

CSPNet (Cross Stage Partial Network). CSPNet дозволяє зменшити обчислювальну складність CNN на 10–20% без втрати точності, а іноді навіть із покращенням.

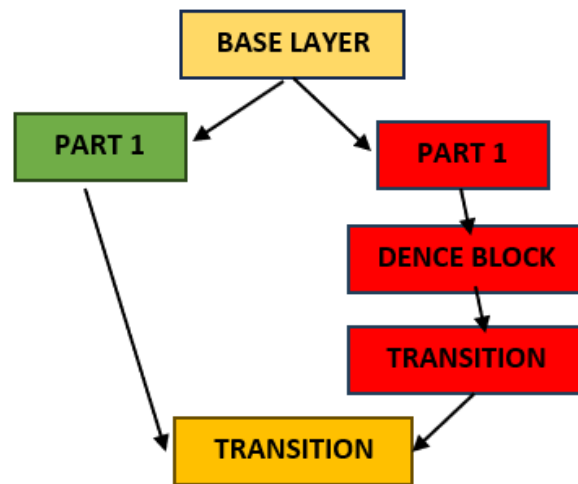


Рисунок 2.4 CSPNet

Як показано на рисунку 2.4, CSPNet ділить базовий шар на дві частини: одна залишається незмінною, а інша проходить через трансформацію. Потім обидві частини об'єднуються (concatenate) і проходять додаткову трансформацію. У CSPDarknet53 базовим шаром виступає feature map згорткового шару перед кожним ResBlock Body, а трансформація здійснюється через додатковий згортковий шар.

SPP (Spatial Pyramid Pooling) та PAN (Path Aggregation Network). SPP використовується для обробки даних на різних масштабах, що дозволяє враховувати глобальний контекст. PAN використовується для кращої передачі характеристик між шарами за допомогою методів підвищення (upsampling) та зниження (downsampling) роздільної здатності.

Реалізація алгоритму буде представлена у розділі 3.

2.1.2. Семантична сегментація зображень

Спектр застосувань семантичної сегментації є досить широким, охоплюючи задачі від аналізу сцен та встановлення взаємозв'язків між об'єктами до забезпечення функціонування безпілотних транспортних засобів. Традиційні методи, що базувалися на обробці низькорівневих візуальних сигналів, значною мірою поступилися місцем алгоритмам машинного навчання, які набули широкої популярності.

Особливо варто відзначити значні досягнення глибокого навчання в таких областях, як розпізнавання рукописних символів, обробка мовлення, класифікація зображень за категоріями та ідентифікація об'єктів на зображеннях [5, 6]. В даний час спостерігається підвищений інтерес до методів семантичної розмітки на рівні окремих пікселів [7-16]. Однак, деякі з сучасних підходів намагаються безпосередньо використовувати архітектури глибокого навчання, спочатку призначені для задач категоризації, для вирішення задач піксельної розмітки [7].

Отримані результати, хоча і є перспективними, часто характеризуються недостатньою точністю [3]. Основною причиною цього є зниження роздільної здатності карт ознак внаслідок застосування операцій максимального пулінгу та субдискретизації. Саме потреба у відновленні роздільної здатності ознак до рівня вхідного зображення для здійснення піксельної класифікації стала ключовою мотивацією для розробки архітектури SegNet. При цьому, отримані ознаки повинні забезпечувати точну локалізацію меж об'єктів.

Архітектура SegNet розроблена як ефективне рішення для задач семантичної сегментації на рівні пікселів. Її створення було зумовлено, в першу чергу, потребами аналізу дорожніх сцен, що передбачає моделювання зовнішнього вигляду об'єктів, їх форми (наприклад, автомобілі, пішоходи) та встановлення просторових відношень між різними класами об'єктів (наприклад, дорога та тротуар).

В типових дорожніх умовах значна частина пікселів належить до великих об'єктів, таких як дорога або будівля, що вимагає від мережі генерування гладких сегментацій. Водночас, система повинна ефективно розрізняти об'єкти за їх формою, навіть якщо вони мають невеликі розміри. Таким чином, збереження інформації про межі об'єктів у представленні зображення є критично важливим. З точки зору обчислювальної ефективності, мережа повинна характеризуватися низьким споживанням пам'яті та швидким часом обробки під час інференсу, (див. рис. 2.5).



Рис. 2.5. Прогнози SegNet на дорогах і в приміщеннях

Додатковою перевагою є можливість наскрізного навчання (end-to-end training) з використанням ефективних методів оптимізації, таких як стохастичний градієнтний спуск (SGD) [17], що забезпечує більшу відтворюваність результатів. Саме необхідність задоволення цим критерієм стала основою для розробки архітектури SegNet.

SegNet має мережу енкодера та відповідну мережу декодера, за якою слідує фінальний шар для піксельної класифікації. Ця архітектура ілюструється на рис. 3. Мережа енкодера складається з 13 згортальних шарів, які відповідають першим 13 згортальним шарам мережі VGG16 [1], розробленої для класифікації об'єктів. Тому ми можемо ініціалізувати процес навчання з ваг, натренованих для класифікації на великих наборах даних [41]. Також можна відкинути повнозв'язні шари на користь збереження більш детальних карт ознак на найглибшому виході енкодера. Це також значно зменшує кількість параметрів у мережі енкодера SegNet (з 134 млн до 14,7 млн), порівняно з іншими сучасними архітектурами [2], [4] (див. таблицю 6). Кожен шар енкодера має відповідний шар декодера, тому мережа декодера складається з 13 шарів. Кінцевий вихід декодера передається до багатокласового софтмакс-класифікатора для отримання ймовірностей класів для кожного пікселя окремо.

SegNet має мережу енкодера та відповідну мережу декодера, за якою слідує фінальний шар для піксельної класифікації. Ця архітектура ілюструється на рис. 3. Мережа енкодера складається з 13 згортальних шарів,

які відповідають першим 13 згортальним шарам мережі VGG16 [1], розробленої для класифікації об'єктів. Тому ми можемо ініціалізувати процес навчання з ваг, натренованих для класифікації на великих наборах даних [41]. Також можна відкинути повнозв'язні шари на користь збереження більш детальних карт ознак на найглибшому виході енкодера. Це також значно зменшує кількість параметрів у мережі енкодера SegNet (з 134 млн до 14,7 млн), порівняно з іншими сучасними архітектурами [2], [4] (див. таблицю 6). Кожен шар енкодера має відповідний шар декодера, тому мережа декодера складається з 13 шарів. Кінцевий вихід декодера передається до багатокласового софтмакс-класифікатора для отримання ймовірностей класів для кожного пікселя окремо.

Кожен енкодер у мережі енкодера виконує згортання з фільтровою банкою для отримання набору карт ознак. Потім ці карти нормалізуються за допомогою пакетної нормалізації [5], [6]. Далі застосовується елементно-оброблена ректифікована лінійна нелінійність (ReLU) $\max(0, x)$. Після цього виконується макс-пулінг з вікном 2×2 та кроком 2 (без перекриття вікна), і результуючий вихід піддається субдискретизації на коефіцієнт 2. Макс-пулінг використовується для досягнення інваріантності до зсувів у просторі для малих зміщень вхідного зображення. Субдискретизація забезпечує великий контекст вхідного зображення (просторове вікно) для кожного пікселя на карті ознак. Хоча кілька шарів макс-пулінгу та субдискретизації можуть досягти більшої інваріантності до зсувів для стійкої класифікації, це супроводжується втратою просторової роздільної здатності карт ознак. Все більш втрачене (деталі меж) представлення зображення не є корисним для сегментації, де важлива чітка розмежованість меж. Тому необхідно захоплювати та зберігати інформацію про межі в картах ознак енкодера до виконання субдискретизації. Якщо пам'ять під час інференції не обмежена, то всі карти ознак енкодера (після субдискретизації) можна зберігати. Однак це зазвичай не є можливим у практичних застосуваннях, тому ми пропонуємо більш ефективний спосіб зберігання цієї інформації. Він полягає у збереженні

тільки індексів макс-пулінгу, тобто запам'ятовуються місця максимального значення ознаки в кожному вікні пулінгу для кожної карти ознак енкодера. В принципі це можна зробити, використовуючи 2 біти для кожного вікна пулінгу 2×2 , що значно ефективніше зберігати порівняно з запам'ятовуванням карт ознак у плаваючій точці. Як ми покажемо пізніше в цій роботі, вона використовує ту ж саму мережу енкодера, що й SegNet-Basic, але з технікою декодування FCN, яка використовується у всіх його декодерах. Ліворуч на рис. 3 зображена техніка декодування, яка використовується в SegNet (також SegNet-Basic), де в кроці апсемплінгу немає навчання. Однак апсемплінговані карти згортатимуться з навчальними багатоканальними фільтрами декодера для збільшення їх рідких входів. Кожен фільтр декодера має таку ж кількість каналів, як і кількість апсемплінгованих карт ознак. Менший варіант — це варіант, де фільтри декодера є одноканальними, тобто вони згортатимуть лише відповідну апсемплінговану карту ознак. Цей варіант (SegNet-Basic-SingleChannelDecoder) значно зменшує кількість навчальних параметрів і час інференції.

2.1.3. Алгоритм відстеження об'єктів

Існують багато сучасних алгоритмів відстеження об'єктів. Вони є важливою частиною системи комп'ютерного зору, який дозволяє визначити місце розташування об'єктів на серії зображень, що змінюються з часом.

У даній роботі використовується алгоритм DeepSort, який є удосконаленим методом для відстеження рухомих об'єктів, базуючись на їхніх візуальних ознаках. Алгоритм SORT отримав покращення завдяки впровадженню методу глибокої асоціації, що базується на технологіях глибокого навчання (Deep Learning). Цей підхід допомагає ефективніше справлятися із затуленням об'єктів та зміною точки зору. Завдяки глибокому навчанню, алгоритм здатний глибше аналізувати візуальні характеристики об'єктів, що підвищує точність та надійність відстеження, особливо у складних умовах.

Цей алгоритм, як і інші алгоритми відстеження, використовує загальну структуру (див. рис.2.6):

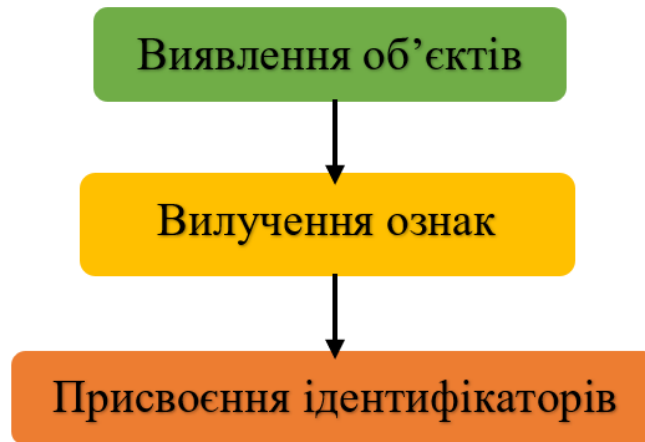


Рис. 2.6. Структура DeepSORT

Різні алгоритми відстеження реалізують кожну частину по-різному, але загалом шаблон залишається незмінним. Deep SORT як ключовим компонентом відстеження множинних об'єктів (MOT)

Відстеження множинних об'єктів (MOT) – це процес визначення положення та подальшого супроводу декількох об'єктів у відеопослідовності або серії зображень протягом певного часу. Ця задача є надзвичайно важливою для комп'ютерного зору та систем спостереження, оскільки дозволяє аналізувати рух та взаємодію багатьох об'єктів одночасно.

Основна мета MOT полягає у створенні траєкторій руху для кожного об'єкта в кадрі та збереженні їх ідентифікації між кадрами, навіть у випадках, коли вони частково або повністю перекриваються іншими об'єктами або тимчасово зникають з поля зору.

Для досягнення цієї мети алгоритми відстеження множинних об'єктів зазвичай використовують комбінацію методів виявлення об'єктів, асоціації даних та прогнозування траєкторій. Ці алгоритми також повинні вирішувати складні завдання, такі як обробка перекриттів, збереження ідентифікації об'єктів у скупчених сценах та адаптація до змін зовнішнього вигляду або руху об'єктів.

Алгоритм SORT є фундаментом системи Deep SORT, яка представляє собою передовий метод для відстеження багатьох об'єктів у відео. SORT, що розшифровується як "просте онлайн відстеження в реальному часі", є простим, але дієвим алгоритмом для асоціації даних та ініціалізації відстеження в сценарії з багатьма об'єктами. Спочатку він використовує підхід на основі виявлення для зіставлення об'єктів у послідовних кадрах, а потім застосовує фільтр Калмана для прогнозування майбутньої позиції об'єктів та корекції зіставлення на основі фактичних вимірювань.

Завдяки своїй ефективності та точності, цей алгоритм отримав широке застосування та служить основою для більш просунутої системи DeepSORT, яка покращує процес асоціації даних за допомогою методів глибокого навчання для подальшого підвищення якості відстеження.

Поєднуючи сильні сторони алгоритму SORT з глибоким навчанням, DeepSORT досяг найвищих результатів у задачах відстеження множинних об'єктів, (див. рис.2.7).

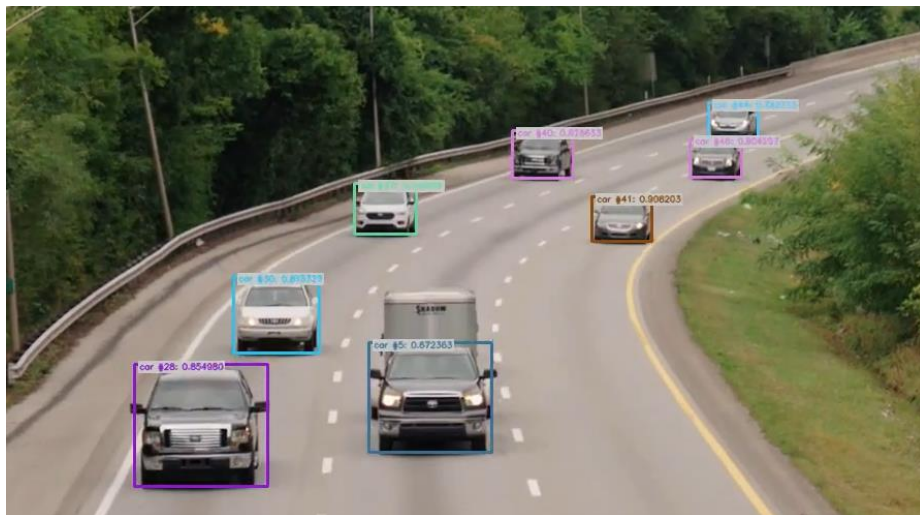


Рис. 2.7. Приклад застосування алгоритму SORT

Алгоритм Deep SORT застосовується для супроводу багатьох об'єктів у відеопотоці. Він є розвитком алгоритму SORT (просте онлайн-відстеження в реальному часі), в якому для цієї задачі використовується фільтр Калмана. Deep SORT використовує метрику глибокої асоціації, що ґрунтується на характеристиках зовнішнього вигляду, які отримуються за допомогою

глибокої згорткової нейронної мережі. Завдяки цьому алгоритм ефективно працює в умовах, коли об'єкти можуть ненадовго зникати з поля зору або перекриватися іншими об'єктами у відео (див. рис.2.8)). Для ідентифікації та супроводу окремих об'єктів між кадрами Deep SORT застосовує присвоєння ID, що є надзвичайно важливим для таких сфер, як системи відеоспостереження, безпілотні транспортні засоби та взаємодія людини з комп'ютером.

Алгоритм діє у два етапи: спочатку відбувається виявлення об'єктів, а потім ці виявлення співставляються з вже існуючими траєкторіями. В цілому, Deep SORT демонструє значне підвищення точності та надійності відстеження в порівнянні з традиційними методами, що робить його цінним інструментом в області комп'ютерного зору та штучного інтелекту.

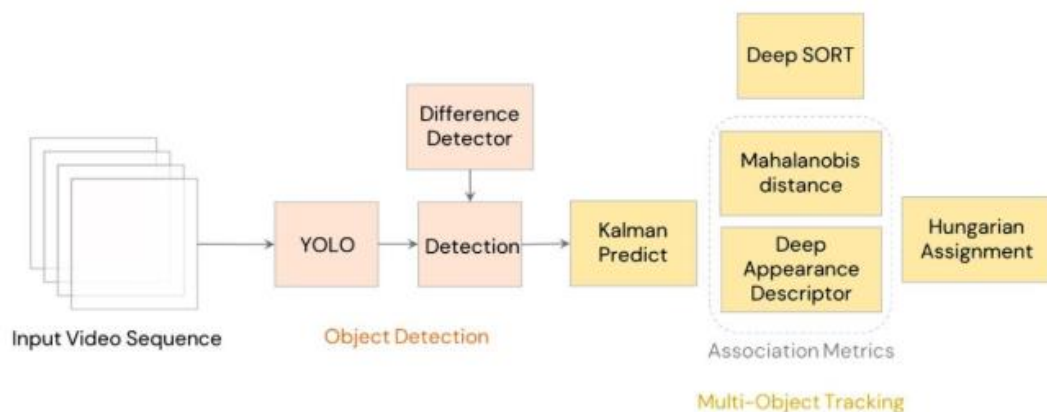


Рис. 2.8. Архітектура Deep SORT

Deep SORT складається з чотирьох основних компонентів.

Виявлення та вилучення ознак. Deep SORT починає з виявлення об'єктів, часто використовуючи згорткову нейронну мережу (CNN), таку як YOLO, для ідентифікації об'єктів у кадрі. Кожне виявлення пов'язане з багатовимірним дескриптором зовнішнього вигляду, отриманим за допомогою іншої CNN. Ці дескриптори кодують зовнішній вигляд виявлених об'єктів і використовуються для зіставлення.

Фільтр Калмана для прогнозування стану. Подібно до SORT, Deep SORT використовує фільтр Калмана для прогнозування стану. Стан об'єкта зазвичай

включає його положення, швидкість та прискорення. Фільтр Калмана прогнозує стан кожного об'єкта в поточному кадрі на основі його останнього відомого стану, враховуючи динаміку руху об'єкта.

Асоціація даних у Deep SORT суттєво відрізняється від свого попередника. Замість того, щоб покладатися виключно на IoU (Intersection over Union – перетин над об'єднанням), він використовує комбінацію інформації про рух та ознак зовнішнього вигляду.

Deep SORT керує життєвими циклами траєкторій, використовуючи кілька евристичних методів. Він вводить такі поняття, як підтвердження траєкторії, коли траєкторія підтверджується лише після виявлення в кількох послідовних кадрах, що зменшує кількість хибних спрацювань. Існує також параметр віку для видалення старих траєкторій, які не були виявлені останнім часом, що гарантує, що система підтримує лише активні траєкторії.

Серед переваг Deep SORT слід відмітити наступне. Завдяки використанню дескрипторів зовнішнього вигляду, Deep SORT може повторно ідентифікувати об'єкти після того, як вони були закриті. Дескриптори зовнішнього вигляду допомагають розрізняти схожі об'єкти. Його можна комбінувати з будь-яким сучасним детектором. Незважаючи на додаткову складність, Deep SORT може працювати майже в реальному часі завдяки ефективним механізмам вилучення ознак та асоціації.

SORT дуже добре працює з точки зору точності відстеження. Але SORT повертає доріжки з великою кількістю перемикачів ідентифікаторів і виходить з ладу в разі оклюзії. Це пов'язано з використанням матриці асоціацій. DeepSORT використовує кращу метрику асоціації, яка поєднує дескриптори руху та зовнішнього вигляду. DeepSORT можна визначити як алгоритм відстеження, який відстежує об'єкти не лише на основі швидкості та руху об'єкта, але й зовнішнього вигляду об'єкта.

2.2. Тренування моделі на спеціальному наборі даних

Для успішної реалізації системи інтелектуального виявлення та класифікації об'єктів у транспортному середовищі було проведено навчання моделі глибокого навчання на спеціалізованих наборах даних. Основна мета полягала у створенні моделі, здатної ефективно обробляти складні відео- та зображення транспортних потоків, визначати ключові об'єкти та здійснювати їх точну класифікацію.

У цьому параграфі розглянемо модель YOLO версії 4.

Для навчання моделі було використано MS COCO Dataset – для навчання моделі YOLO v4, яка використовується для виявлення об'єктів, таких як автомобілі, пішоходи та світлофори.

Моделі для розпізнавання об'єктів постійно вдосконалюються, демонструючи покращення як у швидкості, так і в ефективності роботи.

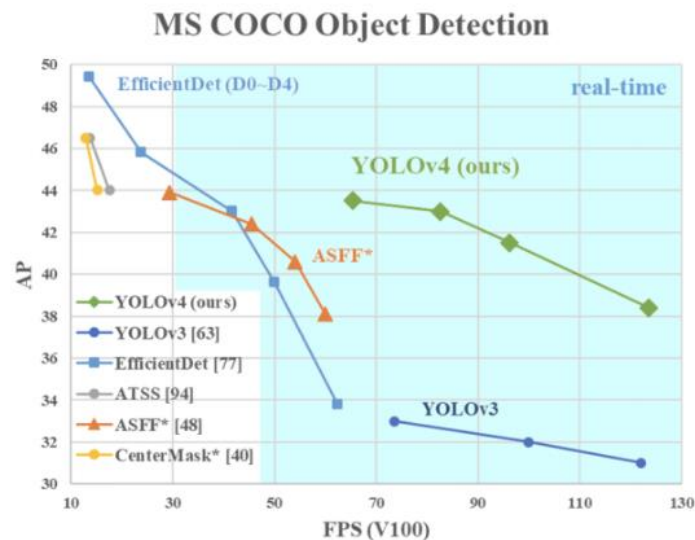


Рис. 2.9. Діаграма

Загальні моделі розпізнавання об'єктів зазвичай перевіряються на наборі даних COCO, який містить широкий спектр об'єктів і класів. Ідея полягає в тому, що якщо модель добре справляється з таким завданням, вона зможе якісно адаптуватися до нових наборів даних.

Однак використання методів глибинного навчання, що застосовуються в дослідженнях, може бути складним на практиці, особливо для розпізнавання унікальних об'єктів.

Використовуючи YOLOv4, так як це не нова версія, але також використовується, і також користується функціями, унікальними для YOLOv4. Серед них: WRC, CSP, CmBN, SAT, активація Mish, мозаїчне розширення даних, DropBlock-регуляризація та CIoU-втрати. Для навчання використовуємо фреймворк Darknet.

Для навчання YOLOv4 нам знадобляться наступні ресурси: GPU з відповідними драйверами, OpenCV, Налаштований cuDNN поверх драйверів GPU.

Далі ми створюємо свій набір даних. Після завантаження набору даних потрібно створити 8 папок з іменами класів і скопіювати відповідні зображення з папки "train" у ці папки. Для цього я написав простий Python-скрипт, який створює вісім папок з зображеннями танців.

Ви також можете зробити це за допомогою скрипту. Після завершення цього посту ви будете добре обізнані у створенні моделі нейронної мережі для класифікації восьми категорій індійських класичних танців.

Перший крок — це позначення зображень у форматі YOLO. YOLO вимагає специфічного типу розмітки. Формат розмітки для зображення повинен бути у такому вигляді: <object-class> <x_center> <y_center> <width> <height>. Тут <object-class> — це ціле число об'єкта від 0 до (класів-1). У нашому випадку це буде від 0 до 7. <x_center>, <y_center>, <width>, <height> — це значення типу float, що відносяться до ширини та висоти зображення і можуть бути в межах від [0.0 до 1.0]. Цей крок є основою для YOLO, тому його потрібно виконати точно.

Для розмітки зображень ми будемо використовувати інструмент анотації labellmg. Після завантаження/клонування потрібно відкрити Anaconda Prompt з правами адміністратора, перейти до директорії labellmg і встановити необхідну бібліотеку ruqt командою: `conda install ruqt=5`. Після встановлення

pyqt, використовуємо наступну команду: `pyrcc5 -o libs/resources.py resources.qrc`. Це встановить інструмент анотації.

Тепер створіть текстовий файл під назвою `classes.txt` у кореневій директорії `labellmg` і додайте туди наші цільові імена класів у такому форматі:

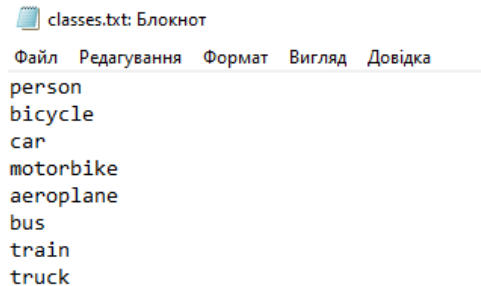


Рис. 2.10. Імена класів

Тепер, коли налаштування `labellmg` завершено, ми можемо перейти до анотації зображень пізніше. Спочатку підготуємо деякі файли та внесемо зміни в директорію YOLO для тренування нашого власного набору даних.

Для цього перейдемо до директорії YOLO (`\vcpkg-master\installed\x64-windows\tools\darknet\cfg`) і виконаємо наступні кроки:

Скопіювати файл `yolov4-custom.cfg`, вставити його туди і перейменувати в `yolo-obj.cfg`;

Відкрити файл `yolo-obj.cfg` і внести такі зміни; перейти до директорії `\vcpkg-master\installed\x64-windows\tools\darknet\data` і створити файл `obj.names`; скопіювати вміст файлу `classes.txt` туди. створити файл `obj.data` в тій самій директорії з таким вмістом.

Створити папку з ім'ям `obj` всередині директорії (`\vcpkg-master\installed\x64-windows\tools\darknet\data`).

Тепер наші зміни в YOLO завершено. Далі перейдемо до анотації зображень: `python labellmg.py /dataset/images/bharatanatyam classes.txt` - це відкриває інтерфейс для анотації зображень, де я міг вибрати відповідну папку для збереження анотацій. Після цього я вручну створював обмежувальні прямокутники навколо об'єктів на кожному зображенні, вибирав відповідний клас (наприклад, "bharatanatyam") і зберігав анотацію.

У цьому розділі я описав кроки, необхідні для анотації зображень та підготовки їх до навчання моделі YOLOv4 на користувацькому наборі даних. Спочатку, після створення папок з класами та завантаження зображень, необхідно було виконати анотацію зображень за допомогою інструменту labelImg.

Я повторив цей процес для кожного класу в наборі даних (інші танці), змінюючи шлях для збереження файлів анотацій, щоб вони зберігалися у відповідних папках для кожного класу.

Після завершення анотацій я скопіював відповідні файли .jpg та .txt з кожної папки у загальну папку obj, яку я створив раніше в процесі налаштування YOLO.

Наступним кроком було створення текстового файлу train.txt в директорії \vcpkg-master\installed\x64-windows\tools\darknet\data. У цьому файлі я вказав імена всіх зображень разом з їх шляхами. Шлях до зображень був таким: /vcpkg-master/installed/x64-windows/tools/darknet/data/obj

```
D:/Softwares/vcpkg-master/installed/x64-windows/tools/darknet/data/obj/1.jpg
D:/Softwares/vcpkg-master/installed/x64-windows/tools/darknet/data/obj/10.jpg
D:/Softwares/vcpkg-master/installed/x64-windows/tools/darknet/data/obj/100.jpg
D:/Softwares/vcpkg-master/installed/x64-windows/tools/darknet/data/obj/101.jpg
D:/Softwares/vcpkg-master/installed/x64-windows/tools/darknet/data/obj/102.jpg
D:/Softwares/vcpkg-master/installed/x64-windows/tools/darknet/data/obj/103.jpg
D:/Softwares/vcpkg-master/installed/x64-windows/tools/darknet/data/obj/104.jpg
D:/Softwares/vcpkg-master/installed/x64-windows/tools/darknet/data/obj/105.jpg
D:/Softwares/vcpkg-master/installed/x64-windows/tools/darknet/data/obj/106.jpg
D:/Softwares/vcpkg-master/installed/x64-windows/tools/darknet/data/obj/107.jpg
D:/Softwares/vcpkg-master/installed/x64-windows/tools/darknet/data/obj/109.jpg
D:/Softwares/vcpkg-master/installed/x64-windows/tools/darknet/data/obj/111.jpg
```

Рис. 2.11. Шляхи до зображень

Для тренування моделі YOLOv4 на користувацькому наборі даних я написав Python-скрипт, що допомагає автоматизувати деякі етапи процесу. Після підготовки даних та анотацій ми можемо приступити до навчання моделі.

Спочатку я відкрив PowerShell з правами адміністратора та перейшов до кореневої директорії Darknet. Для запуску процесу навчання я використав наступну команду: .\darknet.exe detector train data/obj.data cfg/yolo-obj.cfg yolov4.conv.137

Коли команда була виконана, Darknet завантажує шари з файлу ваг і починає тренування на нашому користувацькому наборі даних. Всі налаштування, зокрема кількість класів, шлях до тренувальних даних та місце збереження навчальних ваг, беруться з файлу obj.data, який я створив раніше.

Оскільки ми встановили max_batches рівним 16000 для 8 класів, навчання займає значний час. У моєму випадку весь процес тренування тривав близько 18 годин, тому я рекомендую вимкнути режим сну на комп'ютері і запускати тренування вночі.

Після завершення тренування я побачив на екрані відповідні повідомлення про успішне завершення процесу навчання, що свідчить про готовність моделі до використання.

```
Tensor Cores are used.
16000: 0.418116, 0.640757 avg loss, 0.000010 rate, 9.109000 seconds, 1024000 images
Saving weights to D:\Softwares\vcpgk-master\installed\x64-windows\tools\darknet\backup\yolo-obj_16000.weights
Saving weights to D:\Softwares\vcpgk-master\installed\x64-windows\tools\darknet\backup\yolo-obj_last.weights
Saving weights to D:\Softwares\vcpgk-master\installed\x64-windows\tools\darknet\backup\yolo-obj_final.weights
PS D:\Softwares\vcpgk-master\installed\x64-windows\tools\darknet>
```

Рис. 2.12. Шляхи до зображень

Ця команда для тренування також зберігає графік "втрати проти кількості ітерацій" в кореневій директорії Darknet, який виглядатиме наступним чином:

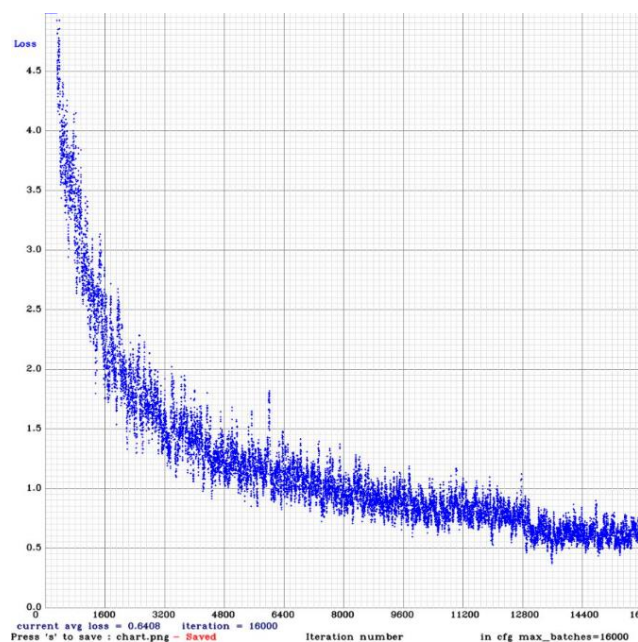


Рис. 2.13. Графік «втрати проти кількості ітерацій»

У результаті тренування можна побачити, що на початку середнє значення втрати було дуже високим, але в кінці тренування воно досягло значення 0.6408. Остаточнє середнє значення втрати може варіюватися від 0.05 (для невеликої моделі та простого набору даних) до 3.0 (для великої моделі та складного набору даних). Отже, ми отримали дуже хорошу модель. Тепер ми можемо протестувати нашу модель на невідомих зображеннях.

Для цього потрібно виконати наступну команду в тому ж вікні PowerShell: `.\darknet.exe detector test data/obj.data cfg/yolo-obj.cfg backup/yolo-obj_final.weights /dataset/test/6.jpg -dont_show -out_filename`

Тут необхідно лише змінити шлях до тестового зображення відповідно до вашої системи. Вказана команда передбачить клас на заданому зображенні з виведенням впевненості в консоль PowerShell, а також збереже зображення з обведенням і класом в кореневій директорії Darknet під іменем "predictions.jpg".

```
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, cls_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
Total BFLOPS 59.614
avg_outputs = 490698
Allocate additional workspace_size = 52.43 MB
Loading weights from backup/yolo-obj_final.weights...
seen 64, trained: 1024 K-images (16 Kilo-batches_64)
Done! Loaded 162 layers from weights-file
```

Рис. 2.14. Результат виконання команди

Відповідне зображення з передбаченням буде виглядати наступним чином:

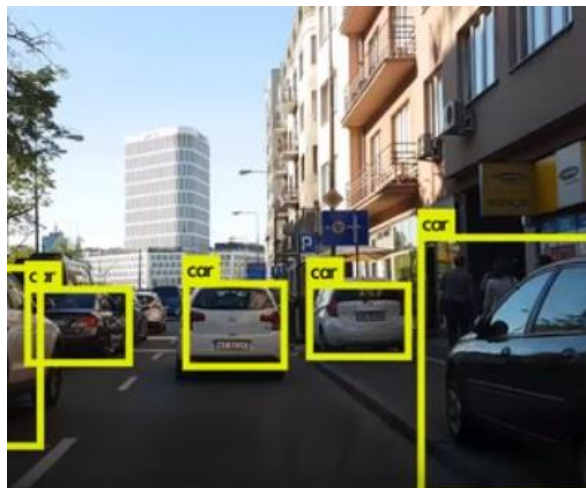


Рис. 2.15. Результат тренування

Модель працює так, як і очікувалося. Хочу зазначити, що для отримання дуже гарної моделі YOLO необхідно мати щонайменше 2000 різних зображень для кожного класу або більше, а також тренувати модель на кількість ітерацій, що дорівнює $2000 * \text{кількість класів}$ або більше. Через обмеження моєї системи я використовував ширину та висоту зображення 416, але якщо ви отримуєте більшу середню втрату, вам слід збільшити роздільну здатність мережі до 608 або 832. Також, якщо ви не хочете виявляти певний об'єкт, додайте зображення цього об'єкта з порожніми текстовими файлами в папку obj.

Висновки до другого розділу

Було проведено аналіз сучасних методів і архітектур для побудови інтелектуальної системи виявлення та класифікації об'єктів з використанням методів глибокого навчання. Розглянуто основні підходи до вирішення задач виявлення об'єктів, семантичної сегментації зображень і алгоритмів відстеження об'єктів.

Також обґрунтовано вибір методів, таких як YOLO, SSD і Faster R-CNN для виявлення об'єктів, а також UNet і DeepLab для семантичної сегментації. Для відстеження об'єктів було запропоновано використання алгоритмів SORT та DeepSORT, які забезпечують високу точність і ефективність роботи.

Для виявлення об'єктів вибрано модель YOLO, яка забезпечує швидкість і точність при детекції кількох об'єктів на зображеннях. Для семантичної сегментації зображень обрано методи на основі глибоких нейронних мереж, що дозволяє точно сегментувати пікселі об'єктів. Для відстеження об'єктів у динамічних сценах використовуються алгоритми DeepSORT і Kalman Filter, які дозволяють стабільно відстежувати об'єкти в реальному часі. Ці методи забезпечують високу точність і ефективність у вирішенні завдань комп'ютерного зору.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ

3.1 Платформа для розробки

Для розробки програми, яка передбачає використання методів глибокого навчання для класифікації об'єктів, було обрано середовище розробки Visual Studio Code (VS Code). Це потужний і популярний редактор коду з відкритим вихідним кодом, який підтримує численні мови програмування та різноманітні інструменти для ефективної розробки програмного забезпечення.



Рис. 3.1. Visual Studio Code

Visual Studio Code пропонує користувачам інтуїтивно зрозумілий інтерфейс і багатий набір функцій, що робить його ідеальним вибором для проектів, пов'язаних із машинним навчанням та глибоким навчанням.

Серед основних можливостей VS Code можна виділити:

Підтримує мови Python, C++, JavaScript, TypeScript та багато інших. Це дозволяє розробникам працювати з різними технологіями в одному середовищі без необхідності постійно переключатися між різними інструментами.

Включає потужну функцію автодоповнення коду, яка значно прискорює процес програмування. Завдяки інтеграції з IntelliSense, редактор надає корисні підказки та пропозиції під час написання коду, що зменшує ймовірність помилок і підвищує продуктивність розробника.

Має величезну кількість розширень і плагінів, які дозволяють інтегрувати популярні бібліотеки для машинного навчання та глибокого навчання, такі як TensorFlow, PyTorch, Keras та інші. Це дозволяє легко

налаштувати середовище для виконання алгоритмів на GPU, що є важливим для ефективної роботи з великими даними та складними моделями.

Має потужний дебагер, який дозволяє розробникам ефективно виявляти та виправляти помилки в коді. Це особливо корисно при роботі з нейронними мережами та великими обсягами даних, де складно відслідковувати, де саме виникає проблема.

Також є будований термінал, який дозволяє безпосередньо виконувати команди для запуску моделей, установки пакетів через `pip` або `conda`, а також виконувати скрипти для обробки даних і навчання моделей.

Дозволяє зручно працювати навіть з великими проектами, забезпечуючи швидкий доступ до всіх файлів і ресурсів. Це особливо важливо при розробці проектів, пов'язаних із обробкою великих обсягів даних, де потрібно постійно взаємодіяти з численними модулями та бібліотеками.

Visual Studio Code є потужним і універсальним середовищем розробки, яке надає всі необхідні інструменти для створення інтелектуальних систем виявлення та класифікації об'єктів, що використовують передові методи глибокого навчання. Завдяки своїй гнучкості, інтеграції з різними технологіями та наявності потужних функцій для розробки, VS Code є ідеальним вибором для даного проекту.

3.2 Інструменти розробки

Сьогодні існують багато інструментів для вирішення складних поставлених задач. Для реалізації проекту інтелектуальної системи було обрано ряд інструментів і технологій, які дозволяють ефективно обробляти великі обсяги відеоданих за допомогою сучасних методів комп'ютерного зору.

Основним мовним інструментом для розробки є мова програмування Python, яка добре підтримує роботу з бібліотеками машинного навчання та обробки зображень.

```
def sum(a, b):
    return (a + b)

a = int(input('Enter 1st number: '))
b = int(input('Enter 2nd number: '))

print(f'Sum of {a} and {b} is {sum(a, b)}')

Enter 1st number:
2
Enter 2nd number:
3
Sum of 2 and 3 is 5
```



Рис. 3.2. Приклад програмного коду на Python

Python - це відкритий та крос-платформний інструмент програмування, який стрімко набирає популярності вже більше десяти років. Вперше мову було презентовано у 1991 році. Основною реалізацією Python є CPython, написана мовою C. Саме CPython є найпоширенішим варіантом використання цієї мови програмування.

Python є однією з найбільш популярних мов програмування для розробки проектів у сфері штучного інтелекту завдяки своїй простоті у використанні, багатій екосистемі бібліотек та зручному синтаксису.

Завдяки численним розширенням, Python стала багатоцільовою мовою. Вона успішно застосовується у наукових обчисленнях, моделюванні, веб-розробці та багатьох інших сферах.

Еволюція мови Python відбувається завдяки чітко регламентованому процесу, що включає генерацію, обговорення, відбір та впровадження ідей щодо її вдосконалення. Ці пропозиції оформлюються у вигляді документів, відомих як PEP (Python Enhancement Proposal – Пропозиції щодо покращення Python). Цікавим фактом є те, що найменування мови жодним чином не пов'язане з плазунами.

Натомість, воно є відсиланням до популярного британського комедійного телевізійного шоу 1970-х років під назвою "Літаючий цирк Монті Пайтона". Однак, попри це, у масовій свідомості назва "Python" частіше асоціюється саме зі змією, а не з телепередачею. Це підтверджується візуальними елементами, такими як значки файлів у операційних системах KDE та Microsoft Windows, а також логотип веб-сайту python.org (до версії

2.5), на яких зображено зміїні голови. Одним із головних завдань розробників Python є створення захопливої та цікавої у використанні мови програмування, що знайшло своє відображення у її назві, інспірований "Монті Пайтоном".

Для реалізації завдань обробки відео та аналізу зображень були використані популярні бібліотеки, такі як TensorFlow та Keras для роботи з нейронними мережами, а також OpenCV для обробки відео та зображень.

Використання TensorFlow дозволяє скористатися потужними можливостями для тренування та застосування глибоких нейронних мереж, таких як YOLO для виявлення об'єктів і DeepSort для відстеження транспортних засобів.

OpenCV – це бібліотека, яка надає потужні інструменти для обробки зображень і відео, що є необхідним для задач, таких як виявлення дорожніх знаків, світлофорів, пішоходів, а також для відстеження транспортних засобів.

OpenCV бере свій початок в компанії Intel у 1999 році, завдяки зусиллям Гері Бредскі. Перший публічний реліз відбувся у 2000 році. Згодом Вадим Писаревський приєднався до Гері Бредскі, очоливши російську команду розробників OpenCV в Intel. У 2005 році OpenCV було використано у Stanley – автомобілі, що став переможцем DARPA Grand Challenge 2005 року. Подальший активний розвиток проєкту продовжився за підтримки Willow Garage під керівництвом Гері Бредскі та Вадима Писаревського. На сьогоднішній день OpenCV підтримує значну кількість алгоритмів, пов'язаних з комп'ютерним зором та машинним навчанням, і його функціональність постійно розширюється.

Вона доступна на різних платформах, включаючи Windows, Linux, OS X, Android, iOS тощо. Крім того, активно розробляються інтерфейси на основі CUDA та OpenCL для забезпечення високошвидкісних операцій на GPU.

OpenCV-Python є Python API для OpenCV. Він поєднує в собі найкращі якості C++ API OpenCV та мови Python.

Для візуалізації вихідних даних були використані кілька інструменти Vue.js для демонстрації та Node.js обробки результатів.



Рис. 3.3. VueJS та NodeJS

Vue.js – це прогресивний JavaScript-фреймворк, який використовувався для побудови інтерфейсу користувача. Vue.js дозволяє швидко створювати інтерактивні та динамічні веб-сторінки, надаючи зручний спосіб взаємодії з користувачем, що особливо важливо для відображення результатів роботи системи в реальному часі.

Будучи прогресивним JavaScript-фреймворком, Vue робить створення інтерфейсів користувача простішим та приємнішим. Він був розроблений з метою поступового впровадження та уникнення багатьох недоліків, властивих існуючим монолітним фреймворкам.

У роботі використовую Vue 2.6.14, яка є стабільною версією цього фреймворку.

Також застосовую основні технології та бібліотеки:

Vue Router – використовується для організації навігації між різними сторінками вашого додатка. Це дозволяє користувачам переміщатися між різними секціями інтерфейсу без перезавантаження сторінки.

Element UI – бібліотека компонентів інтерфейсу користувача для Vue.js, яка забезпечує велику кількість готових до використання компонентів, таких як кнопки, форми, таблиці, модальні вікна і інше.

Axios – бібліотека для роботи з HTTP-запитами, яка дозволяє вашому додатку відправляти асинхронні запити на сервер для отримання або відправки

даних. Це необхідно для зв'язку з бекендом, обробки відео та виведення результатів на інтерфейс.

Vue Hover Mask – спеціальна бібліотека для створення масок або ефектів при наведенні миші на елементи інтерфейсу.

Core-JS – бібліотека для забезпечення сумісності з старими браузерами та поліфіллів, щоб ваш додаток працював на більшості сучасних браузерів.

Node.js – використовувався для обробки вихідних даних з Python-компіляцій та для організації зв'язку між серверною частиною та інтерфейсом користувача. Node.js дозволяє ефективно обробляти асинхронні запити, що важливо для роботи з відеопотоками та отриманням результатів детекції об'єктів у реальному часі. За допомогою Node.js було реалізовано запуск Python-скриптів, що відповідають за обробку відео та виявлення об'єктів, а також передачу результатів на фронтенд для візуалізації.

Основні бібліотеки Node.js, які використовую в роботі:

Express – бібліотека є популярним веб-фреймворком для Node.js, який значно полегшує створення серверних додатків. Вона надає потужні інструменти для організації маршрутизації, обробки запитів і відповідей, а також для роботи з різними типами даних, зокрема для створення RESTful API. Завдяки Express стає можливим швидко налаштувати сервер і розробляти масштабовані додатки.

Path – модуль в Node.js використовується для роботи з шляхами файлів і директорій. Цей модуль дозволяє працювати з файловою системою незалежно від операційної системи, забезпечуючи крос-платформенну сумісність. Завдяки path, розробник може маніпулювати шляхами файлів без необхідності враховувати особливості кожної операційної системи.

FS (File System) – модуль дозволяє взаємодіяти з файловою системою. Це вбудований модуль Node.js, який забезпечує функції для читання та запису файлів, а також для створення та видалення директорій.

Бібліотека spawn з модуля child_process дозволяє запускати нові процеси з Node.js додатка, що особливо корисно для виконання зовнішніх

команд або програм. Завдяки `spawn`, можна запускати зовнішні програми, отримувати результат виконання та працювати з потоками вводу/виводу: запуск сторонніх команд або скрипті; асинхронне отримання результатів виконання процесу; підтримка обробки потоків вводу/виводу.

Це бібліотека для обробки завантаження файлів у форму через HTTP-запити типу `multipart/form-data`. Вона дозволяє працювати з файлами, що передаються через форми, зберігати їх на сервері або обробляти в реальному часі.

Бібліотека `multer` забезпечує зручне і гнучке управління файлами: підтримка завантаження одного або кількох файлів; налаштування обмежень на типи і розмір файлів; збереження файлів у тимчасових директоріях або в пам'яті.

3.3 Розробка програмного забезпечення для обробки даних

Головною метою даного проєкту є розробка інтелектуальної системи, яка здатна здійснювати виявлення та класифікацію об'єктів на відеозаписах, що створює основу для впровадження сучасних інтелектуальних транспортних систем (ІТС). Ця система повинна використовувати найсучасніші методи глибокого навчання для аналізу відеопотоку, що дозволить забезпечити глибоке розуміння дорожньої обстановки, визначення потенційних ризиків і забезпечення оперативного реагування на небезпечні ситуації.

Розробка такої системи має на меті вирішення кількох ключових задач, кожна з яких є невіддільною частиною створення повноцінного інструменту для аналізу транспортних потоків. По-перше, необхідно створити алгоритми, здатні точно ідентифікувати об'єкти, такі як транспортні засоби, пішоходи, дорожні знаки, світлофори, розмітка, а також інші елементи дорожньої інфраструктури. Це вимагає використання сучасних архітектур нейронних мереж, таких як YOLOv4 для виявлення об'єктів та SegNet для виконання семантичного сегментування.

По-друге, особливе значення має здатність системи до обробки відеопотоку в режимі реального часу. Це вимагає оптимізації швидкості обчислень, оскільки відеопотік складається з безперервної послідовності кадрів, кожен з яких має бути проаналізований з високою точністю. З цією метою використовуються алгоритми відстеження об'єктів, такі як DeepSORT, які дозволяють встановити часове контекстне співвідношення між кадрами, що дає змогу ефективно відстежувати рух транспортних засобів і пішоходів.

Ще одним важливим завданням є збір та підготовка навчальних даних. Для досягнення високої точності система потребує обширного датасету з якісно анотованими прикладами різних дорожніх ситуацій. Усі дані мають бути належним чином оброблені та аугментовані, щоб забезпечити стійкість моделі до змін умов освітлення, погоди, ракурсів камер та інших факторів.

Після розробки алгоритмів і навчання моделі наступним етапом стане її інтеграція в інтелектуальну транспортну систему. Це включає розгортання системи на серверній інфраструктурі, оптимізованій для використання GPU, що значно прискорить обчислення. Крім того, буде створено веб-додаток для моніторингу результатів, який дозволить користувачам переглядати аналіз транспортних потоків і отримувати візуалізовані результати в реальному часі.

Після впровадження системи буде проведено її тестування. Цей етап включає оцінку точності та швидкодії моделі, визначення її ефективності в різних сценаріях, таких як виявлення транспортних засобів, що порушують правила дорожнього руху, ідентифікація пішоходів на пішохідних переходах, а також аналіз динаміки транспортних потоків. Усі результати тестування будуть детально проаналізовані з метою визначення можливостей для подальшого вдосконалення системи.

Таким чином, реалізація даного проєкту спрямована на створення високоефективної інтелектуальної системи, яка не тільки сприятиме підвищенню безпеки на дорогах, але й стане основою для подальшого розвитку інтелектуальних транспортних систем.

Наступні пункти будуть показувати процес розробки програмного забезпечення, призначеного для ефективної обробки даних. Описано також реалізацію алгоритмів, спрямованих на забезпечення точності й продуктивності системи.

Перед розробкою проєкту, складемо чітку структури, яка забезпечить ефективну організацію процесів збору, обробки та використання даних (див. рис.3.4). Такий підхід дозволяє оптимізувати роботу з відео- та зображувальними даними, забезпечуючи високу продуктивність і зручність взаємодії для кінцевих користувачів. [23]

На першому рівні даних відбувається збір даних. Цей рівень слугує початковим етапом, де всі необхідні вхідні дані збираються для подальшого аналізу.

Другий рівень відповідає за аналіз і перетворення зібраних даних. Відео та зображення подаються на модуль обробки, де виконується модульний аналіз і розпізнавання. Після обробки результати виводяться у вигляді зображень чи відео, а також проходять через етап перевірки відповідно до встановлених правил ідентифікації.

Третій рівень забезпечує зручність використання отриманих даних. Візуалізація дозволяє користувачам взаємодіяти з системою через інтерфейс, тоді як обробка інформації інтегрує результати для їх подальшого використання. Зібрані дані зберігаються у сховищі.

Рівень 1: Джерело даних	
Відео	Зображення
Рівень 2: Термінал обробки	
Вхід відео та зображення	Вихід відео та зображення
Модульний аналіз та розпізнавання	Визначення за правилами
Рівень 3: Платформа	
Інтерфейс	
Обробка даних	
Зберігання даних	

Рис. 3.4. Структура проєкту

Після побудови структури перейлесо на саму розробку програмного забезпечення.

Основою роботи моделі слугує алгоритм YOLOv4 (You Only Look Once), який налаштований для роботи з попередньо тренованими вагами та набором анкерів, що визначають розміри об'єктів для виявлення. [24] Реалізовані функції включають:

Компонентів YOLOv4. Архітектура мережі включає складові, такі як Darknet body, механізми downsampling та resblock. Ці елементи забезпечують обчислення ознак на різних масштабах, що є важливим для точності детекції.

Функції Mish. [10] Вона була імплементована як альтернатива стандартним активаційним функціям, що покращує гладкість виходу мережі та стабільність під час тренування.

Обробка вихідних шарів мережі. Для кожного з рівнів передбачення (19x19, 38x38, 76x76) реалізовано механізми об'єднання інформації з різних шарів мережі, що дозволяє ефективніше обробляти великі й малі об'єкти.

Реалізація функцій оцінки якості детекції. Для оцінки точності передбачень були використані методи обчислення Intersection over Union (IoU), Generalized IoU (GIoU), та Distance IoU (DIoU) [12]. Вони дозволяють коректно обчислювати втрати та оновлювати параметри мережі.

Передпроцесинг та постпроцесинг даних. Для роботи з реальними зображеннями було створено функції нормалізації вхідних даних, підготовки анкерних рамок, а також механізми фільтрації результатів з використанням методу Non-Maximum Suppression. Також модуль підтримує завантаження параметрів мережі з файлів форматів .h5, .txt та інших, що забезпечує гнучкість використання та налаштування.

Основна частина коду представлена у додатку «В».

Наступним кроком буде демонстрація процесу детекції об'єктів на відео з використанням нейронної мережі YOLOv4. В даному випадку виділемо автомобілів:

```
import os
```

```

import numpy as np
from Detect.yolo import YOLOv4Detector
import Track.video as video
from PIL import Image
import matplotlib.pyplot as plt

dataloader = video.Video("data/dp.mp4")
img = dataloader.retrieve_frame()
img = Image.fromarray(img[..., ::-1])
yolo4 = YOLOv4Detector()
yolo4.set_target_classes("car")
bboxes, _, sc = yolo4.detect_objects(img)
output_dir = "result"
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
for i, bbox in enumerate(bboxes):
    img = np.array(img)
    x1, y1, width, height = int(bbox[0]), int(bbox[1]), int(bbox[2]),
int(bbox[3])
    x2 = x1 + width
    y2 = y1 + height
    if x2 > x1 and y2 > y1:
        roi = img[y1:y2, x1:x2, :]
        plt.imshow(roi)
        plt.axis('off')
        output_path = os.path.join(output_dir, f"roi_{i+1}.png")
        plt.savefig(output_path, bbox_inches='tight', pad_inches=0)
        plt.close()
        print(f"Saved: {output_path}")
    else:
        print(f"Invalid bounding box coordinates: {bbox}")

```

Результатом обробки буде виділення регіонів інтересу (ROI), які зберігаються як окреме зображення у форматі .png в директорії result. Це зображення міститиме тільки ті ділянки, що були визначені як важливі для подальшого аналізу або обробки, що можна побачити на прикладах, зображених на рис. 3.5 та рис. 3.6.



Рис. 3.5. Вхідне зображення

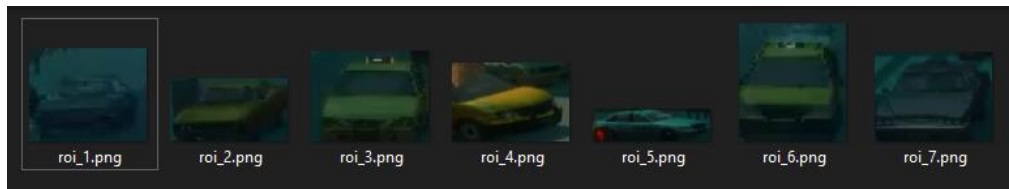


Рис. 3.6. Вихідні зображення

Далі проведемо опис модулю «Сегментація», як наступного кроку по обробці візуальних даних. Сегментація нам потрібна для поліпшення якості розпізнавання.

Наступним кроком є обробку візуальних даних на основі моделі SegNet, адаптованої до архітектури MobileNet, (див. рис.3.7) [21]. Основна мета розробки полягала у створенні ефективного засобу сегментації зображень, який поєднує високу точність із низькими вимогами до апаратного забезпечення.

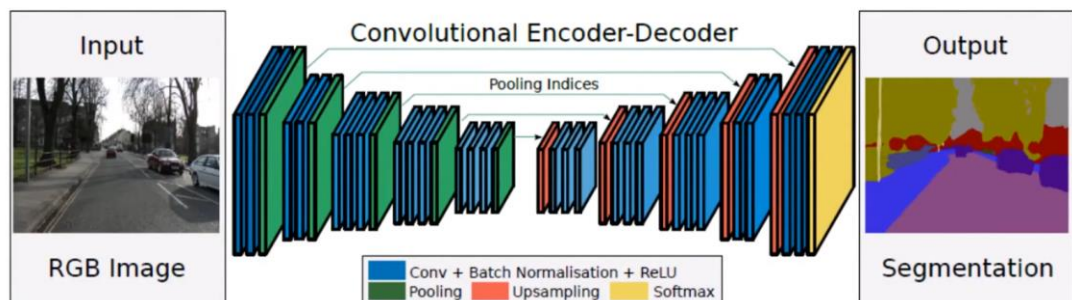


Рис. 3.7. SegNet/MobileNet

В основі використано модель MobileNet, яка є легкою згортковою нейронною мережею. Енкодер складається з кількох послідовних згорткових блоків, що обробляють вхідні зображення для виділення багаторівневих характеристик.

Ключовою особливістю цієї моделі є застосування глибинних згорток, які дозволяють зменшити кількість параметрів і прискорити обчислення. Для забезпечення нелінійності було використано функцію активації ReLU6, яка обмежує значення виходу в межах від 0 до 6, що сприяє стабільності моделі. Код представлений у додатку «В»

На основі отриманих характеристик з енкодера було розроблено декодер SegNet. Ця частина відповідає за відновлення просторової роздільної здатності вихідного зображення, поступово збільшуючи розміри висоти та ширини за допомогою операцій Upsampling.

Декодер складається з кількох послідовних етапів, які включають згорткові шари, нормалізацію (BatchNormalization) і збільшення розмірів (UpSampling2D). Такий підхід дозволяє зберігати якість сегментації навіть на етапах збільшення роздільної здатності.

Також реалізовано функцію створення кінцевої моделі SegNet, яка поєднує енкодер і декодер у єдину архітектуру. Вихідний шар моделі забезпечує класифікацію пікселів зображення на відповідні класи за допомогою шару Softmax.

Для забезпечення універсальності та зручності роботи передбачено можливість налаштування кількості класів, розмірів вхідного зображення та рівня деталізації обраного енкодера. Код представлений у додатку «В»

Реалізовані функції дозволяють адаптувати модель до специфічних вимог, зокрема шляхом зміни параметрів енкодера, декодера чи обчислювальних ресурсів. Це забезпечує гнучкість у використанні моделі для різних завдань, таких як сегментація дорожнього полотна, розпізнавання об'єктів на супутникових знімках чи аналіз медичних зображень.

Розглянемо процес розробки програмного забезпечення, орієнтованого на обробку даних, що забезпечує ефективне виявлення та відстеження об'єктів у відеопослідовностях. Розроблене рішення базується на інтеграції алгоритмів машинного навчання з класичними методами обробки даних.

```
import numpy as np

class BoundingBoxDetection:
    def __init__(self, bbox, score, descriptor, label):
        self.bbox = np.asarray(bbox, dtype=np.float32)
        self.score = float(score)
        self.descriptor = np.asarray(descriptor, dtype=np.float32)
        self.label = label
    def to_bbox_coordinates(self):
        coordinates = self.bbox.copy()
        coordinates[2:] += coordinates[:2]
        return coordinates
    def to_center_aspect(self):
        coordinates = self.bbox.copy()
        coordinates[:2] += coordinates[2:] / 2
        coordinates[2] /= coordinates[3]
        return coordinates
```

Код реалізує кілька ключових компонентів. Одним з основних модулів є клас `BoundingBoxDetection`, який використовується для опису обмежувальних прямокутників (bounding boxes), що визначають позицію об'єктів на зображенні.

Даний клас надає можливості для конвертації координат обмежувальних прямокутників у різні формати, що зручно для подальшого аналізу та обчислень. Крім того, клас підтримує збереження додаткових характеристик об'єктів, таких як дескриптори ознак, оцінки достовірності та мітки.

Для відстеження об'єктів було впроваджено механізм Калманівського фільтру. Основним компонентом цього підходу є клас `BoxTracker`, що моделює рух об'єкта через його параметри, такі як центр прямокутника, його висота, та аспектне співвідношення. Реалізація включає етапи прогнозування та корекції стану, а також проєкцію стану у простір вимірювань. Це забезпечує плавний і стійкий процес відстеження навіть при частковій втраті даних.

Загалом, розроблене програмне забезпечення дозволяє вирішувати широкий спектр задач, пов'язаних з аналізом об'єктів у відеопослідовностях, зокрема виявлення, відстеження та асоціація об'єктів, а також їх фільтрація та оптимізація результатів виявлення.

Після розробки ключових модулів, переходимо до фінального стану. Основна частина системи обробки даних, відповідає за об'єднання функціональності різних модулів для досягнення єдиної мети аналізу відеоматеріалів.

Код реалізує обробку відео в реальному часі з використанням методів детекції, трекінгу об'єктів і сегментації. Основною точкою входу є функція `main`, яка ініціалізує всі необхідні моделі, налаштовує параметри й керує виконанням процесу. Розпочинається з завантаження моделей за допомогою функції `load_models`.

Це включає ініціалізацію детектора YOLOv4, трекера MultiTargetTracker, а також сегментаційної моделі на основі MobileNet. Завантажені моделі забезпечують можливість детекції об'єктів (людей, автомобілів), трекінгу їх переміщення, а також сегментації дорожніх розміток і пішохідних переходів.

```
def load_models():
    """Loads the models needed for detection, tracking, and segmentation."""
    yolo_detector = YOLOv4Detector()
    # Deep Sort setup
    metric = nn_matching.NeighborDistanceMetric("cosine", max_cosine_dist,
nn_budget)
    tracker_instance = MultiTargetTracker(metric)
    model_file = "Tracking/DeepSort/model_data/market1501.pb"
    feature_encoder = gdet.create_bbox_feature_extractor(model_file,
batch_size=1)
    # MobileNet Segmentation Model
    segmentation_model = mobilenet_based_segnet(n_classes=NUM_CLASSES,
input_height=IMG_HEIGHT, input_width=IMG_WIDTH)
    segmentation_model.load_weights(model)
```

Далі, в процесі `process_video`, кожен кадр відео обробляється послідовно. Спочатку кадр аналізується на наявність статичних об'єктів, таких як пішохідні переходи, світлофори та дорожні смуги, використовуючи функцію `static_process`. Потім відбувається основна обробка кадру: детектор об'єктів виявляє автомобілі та людей, трекер оновлює інформацію про їхні траєкторії, а сегментаційна модель уточнює деталі про дорожню інфраструктуру.

Для кожного кадру виконується обчислення стану світлофорів, аналіз поведінки автомобілів у контексті дорожніх правил, а також реєстрація порушень, таких як проїзд на червоне світло або недотримання пішохідних переходів. Інформація про всі ці аспекти записується до бази трекера (`tracker_database`) та підсумовується у фінальних результатах.

Програма також генерує відео з візуалізацією результатів, додаючи на кожен кадр графічні позначення, такі як траєкторії руху, зони пішохідних переходів, статус світлофорів тощо. Для цього використовуються спеціальні функції візуалізації, як-от `plot_video_info`, `plot_cars_rect`, `plot_zebra_crossing`.

Основна частина коду інтегрує всі модулі та забезпечує повний цикл обробки даних – від попереднього аналізу та детекції об'єктів до візуалізації результатів і запису підсумкової інформації. Це дозволяє створювати систему аналізу, яка може ефективно застосовуватися в задачах моніторингу дорожнього руху або безпеки.

3.4 Інтеграція та тестування моделі у веб-середовищі

Концепція архітектури веб-додатку описує структуру та взаємодію його складових частин. Вона визначає, яким чином компоненти системи співпрацюють та розподіляють функціональні обов'язки для досягнення максимальної ефективності та забезпечення можливості масштабування. Однією з найпоширеніших парадигм у проектуванні веб-додатків є модель "клієнт-сервер".

У цій моделі веб-додаток складається з двох ключових елементів: клієнта та сервера. Клієнт, яким найчастіше виступає веб-оглядач, відображає

інтерфейс для користувача та отримує введені ним дані. Сервер, своєю чергою, обробляє запити, що надходять від клієнта, реалізує бізнес-логіку та забезпечує доступ до необхідної інформації.

Функціональність веб-додатків забезпечується завдяки злагодженій роботі різноманітних компонентів. Клієнтська частина, відома як Front-end, відповідає за візуалізацію інтерфейсу користувача, тоді як серверна частина, або Back-end, обробляє запити користувачів та генерує відповіді.

Різноманітні архітектурні патерни слугують основою для організації та структурування компонентів веб-додатків. Найбільш популярними з них є монолітна, клієнт-серверна, розподілена та мікросервісна архітектури.

Front-End частина. Основним інструментом для розробки інтерфейсу є Vue. Основні аспекти були розглянуті в розділі «Інструменти розробки». Для створення UI частини необхідно завантажити Vue і обов'язково вибрати версії. В даній роботі використовується друга версія. Після встановлення Vue і усіх необхідних пакунків переходимо до створення сторінок.

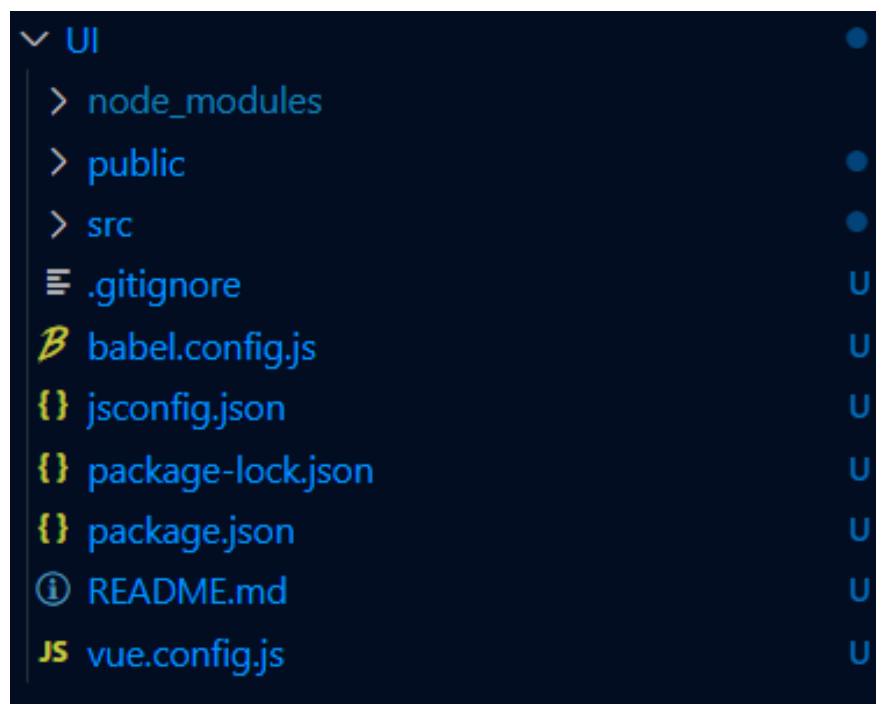


Рис. 3.8. Результат завантаження Vue

Основними сторінками будуть дві: сторінка для завантаження, то сторінка для демонстрації результатів.

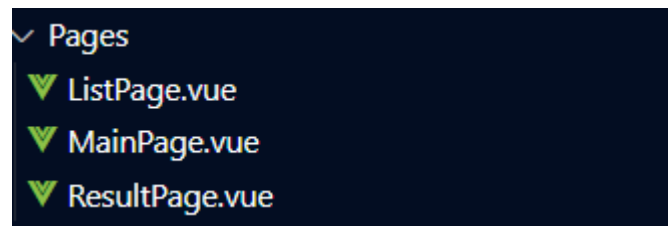


Рис. 3.9. Сторінки для frontend

Реалізуємо форму завантаження відео та фонового зображення. Даний код представлений у додатку «Г».

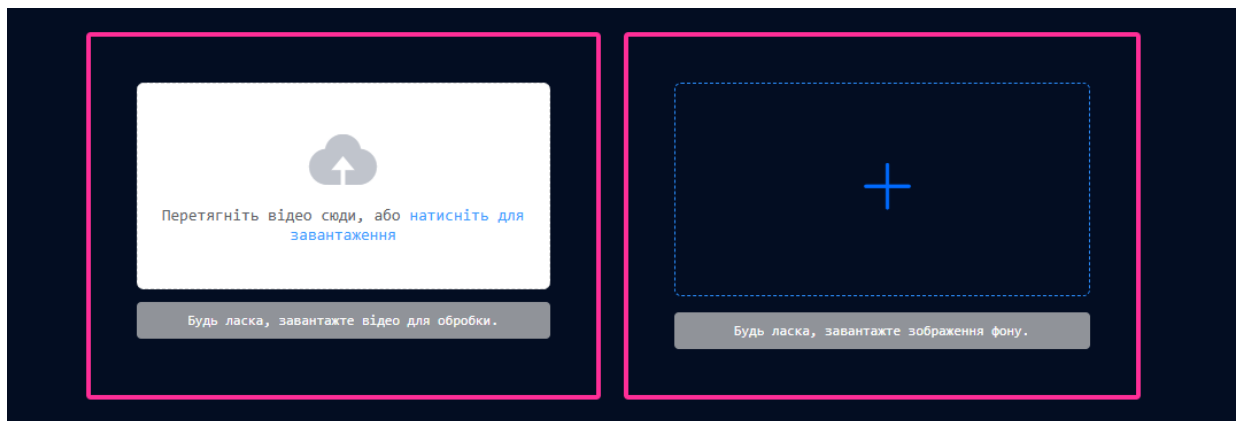


Рис. 3.10. Форма завантаження вхідних даних

Для обробки запитів перейдемо у back-end частину.

Back-End частина. У цьому проєкті back-end частина виконує кілька ключових завдань, які забезпечують функціональність системи обробки відео та взаємодію з користувачем через веб-інтерфейс. Основні особливості back-end частини:

Node.js: Використовується для створення серверного оточення та забезпечення обробки запитів клієнтів. Менеджер пакетів для Node.js, використовується для встановлення залежностей, необхідних для запуску веб-сервера. Основна мова програмування, яка посиляє запит на python для запуску скрипта main.py для розробки моделей, обробки відео та виконання алгоритмів, таких як YOLOv4, SegNet, MobileNet і DeepSORT. Весь код серверної частини представлений у додатку Г.

Зазначу, що у проєкті також використовую базу даних MongoDB для зберігання інформації оброблених даних. Основна таблиця складається із таких рядків:

Id – Унікальне число, яке також служить для посилання на збережений результат у директорії;

Name – назва результату вибраним користувачем;

Date – час збереження;

Timer – кількість секунд обробки даних.

Представлений код реалізації збереження інформації у БД.

```
<button @click="saveProject">Зберегти</button>
...
async saveProject() {
  // Відкриття prompt для введення назви проєкту
  const name = prompt('Введіть назву проєкту:');
  if (!name) {
    alert('Назва проєкту не може бути пустою!');
    return;
  }

  try {
    // Відправка даних на сервер
    const response = await axios.post('http://localhost:8000/savedata', {
      name,
      timer: this.timer,
    });

    // Показ повідомлення про успіх
    this.message = response.data.message;
  } catch (error) {
    console.error('Помилка збереження:', error);
    this.message = 'Помилка збереження проєкту';
  }
}
}
```

Зв'язок з БД та основні операції представлені у додатку «Д».

3.5 Проєктування та тестування моделі у веб-застосунку

Проєкт передбачає інтерактивну взаємодію користувача із веб-додатком для аналізу відеоматеріалів дорожнього руху. Основний процес включає кілька ключових етапів, які реалізовані як на фронтенді, так і на бекенді.

На головній сторінці користувач може завантажити відеофайл із записом дорожнього руху, який буде оброблено. Також необхідно додати статичне фонове зображення сцени, яке використовується для визначення середовища. Реалізація представлена на рис. 3.11-3.15.

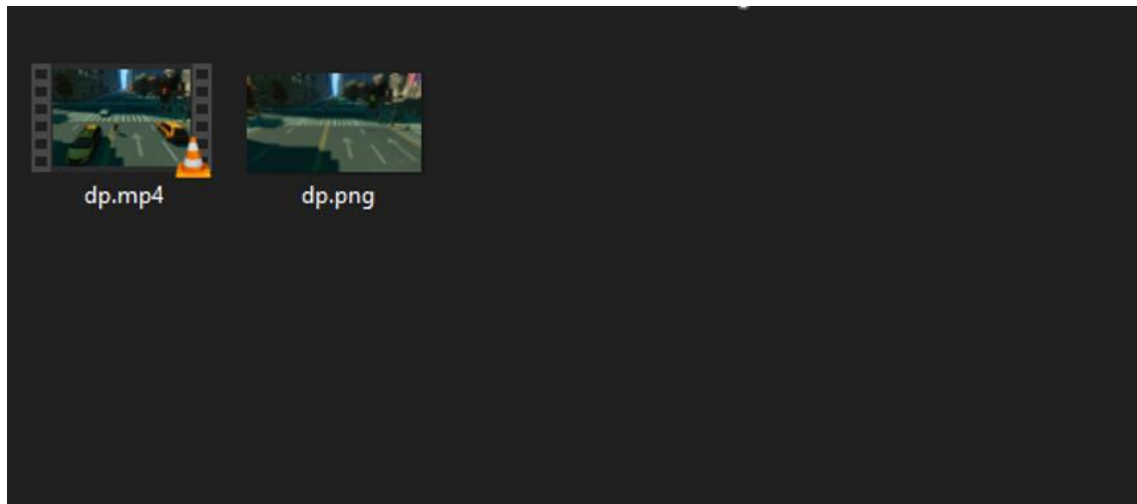


Рис. 3.11. Вибірка вхідних даних

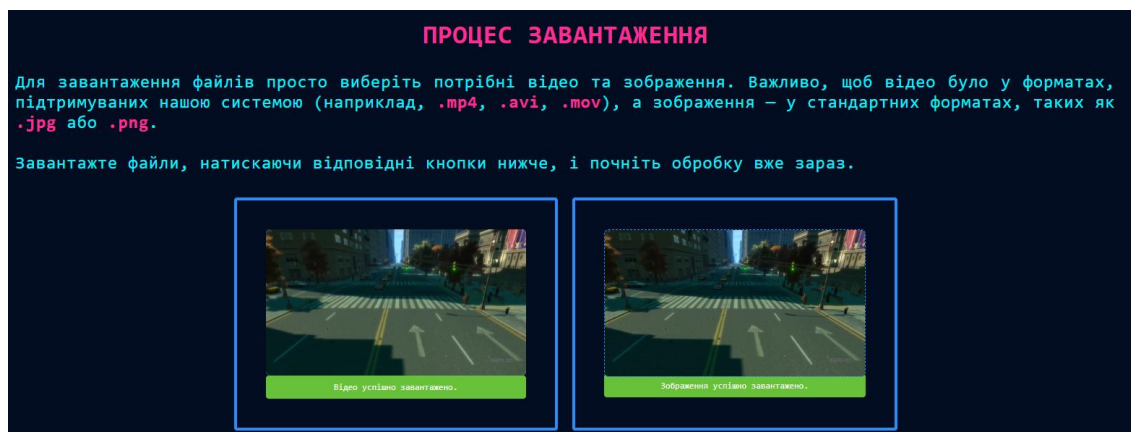


Рис. 3.12. Результат завантаження

Після завантаження матеріалів користувач натискає кнопку "Почати обробку". Це запускає передачу відео та зображень на бекенд для аналізу (див. рис.3.13).



Рис. 3.13. Процес обробки даних

Після завершення обробки користувач автоматично перенаправляється на сторінку результатів. Згенеровані фрейми представлені на рис.3.14. Результат згенерованого фрейму представлено на рис.3.15.



Рис. 3.14. Згенеровані фрейми

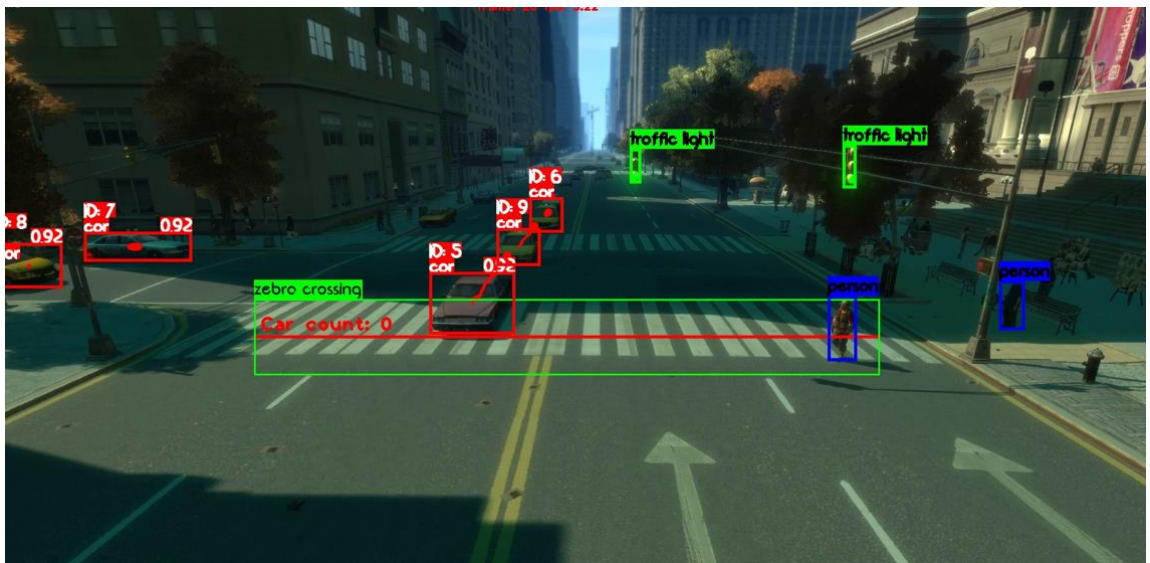


Рис. 3.15. Результат згенерованого фрейму

У правій колонці сторінки результатів розміщується таблиця, яка містить інформацію про: час кожного порушення; ID порушення; додаткові дані про транспортні засоби. У користувача є можливість завантажити оброблене відео або результати в табличному форматі, рис.3.16.

Відео можна відтворити прямо на сторінці, щоб переглянути аналіз потоку транспорту. Такий підхід дозволяє ефективно виконувати аналіз дорожнього руху в режимі 2D. Об'єднання front-end і back-end частин забезпечує зручний інтерфейс для користувачів, а також автоматизує обробку великих обсягів відеоданих за допомогою інноваційних алгоритмів комп'ютерного бачення.

Година	ID	Порушення правил дорожнього руху
1.25	101	Транспортний засіб не поступився пішоходам
5.50	102	Транспортний засіб рухається не за правилами
12.30	103	Транспортний засіб проїхав на червоне світло

Рис. 3.16. Огляд результат таблиці

На завершення хочеться підкреслити досягнення цього проєкту. Було успішно об'єднано сучасні алгоритми комп'ютерного зору для автоматизованого аналізу дорожнього руху, зокрема для розпізнавання дорожніх смуг та об'єктів у динамічному середовищі. Проєкт демонструє ефективне поєднання сучасних підходів у комп'ютерному зорі та алгоритмічній оптимізації, створюючи платформу для розширення можливостей у сфері аналізу відеоданих дорожнього руху.

Одним з ключових аспектів цього проєкту є застосування методів сегментації зображень та аналізу об'єктів, що дозволяє не тільки виявляти дорожні смуги, але й ідентифікувати та відстежувати рухомі об'єкти, такі як транспортні засоби, пішоходи та велосипедисти. Завдяки використанню алгоритмів глибинного навчання, вдалося досягти високої точності в складних умовах, включаючи змінні погодні умови та різноманітні дорожні ситуації.

Висновки до третього розділу

Було здійснено детальний опис процесу реалізації інтелектуальної системи, починаючи з вибору платформи та інструментів розробки до інтеграції та тестування моделі у веб-середовищі.

Особливу увагу було приділено розробці програмного забезпечення для обробки даних, що включає розробку структур проєкту та окремих модулів, таких як «Виявлення», «Сегментація» та «Відстеження», які складають основну частину обробки даних.

Ретельне тестування та інтеграція моделі в веб-застосунок підтвердили її ефективність у реальних умовах, що засвідчило готовність системи до подальшого використання та вдосконалення. Таким чином, розроблена система забезпечує необхідні функції для ефективної обробки даних та їх інтеграції у веб-середовище, що дозволяє успішно реалізувати завдання в межах проєкту.

ВИСНОВКИ

У кваліфікаційній магістерській роботі було розроблено систему інтелектуального розпізнавання транспортних засобів з використанням технологій комп'ютерного зору для обробки інформації з відео. Завдяки застосуванню сучасних методів детекції об'єктів, таких як YOLOv4, а також сегментації з використанням моделей MobileNet і SegNet, система здатна ефективно розпізнавати об'єкти на дорозі, такі як світлофори, пішохідні переходи та транспортні засоби.

Особливістю проекту є інтеграція таких методів, як відстеження автомобілів (за допомогою алгоритму DeepSORT) та детекція пішоходів, що дозволяє реалізувати різноманітні функції, зокрема виявлення порушень, таких як проїзд на червоне світло або рух без дотримання дорожніх смуг.

Проект є важливим кроком у розвитку інтелектуальних транспортних систем, які мають потенціал значно покращити ситуацію на дорогах, знижуючи затори, підвищуючи безпеку та ефективність транспортування. Завдяки використанню передових технологій комп'ютерного зору, таких як YOLOv4 для виявлення об'єктів і MobileNet з SegNet для сегментації зображень, система здатна виконувати складні завдання, зокрема розпізнавання світлофорів, пішохідних переходів та дорожніх смуг.

Завдяки простому інтерфейсу командного рядка та веб-серверу, що надає доступ до результатів, проект орієнтований на практичне використання в реальних умовах. Однак на цьому етапі проект потребує подальшого вдосконалення, зокрема, для забезпечення стабільної роботи на різних платформах і підвищення надійності веб-сервера.

В цілому, система демонструє високий потенціал для подальшої інтеграції в інтелектуальні транспортні системи, що, в свою чергу, може мати значний вплив на підвищення безпеки на дорогах і покращення якості дорожнього руху в майбутньому.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація YOLO [Електронний ресурс]. – Режим доступу: <https://yolo-docs.readthedocs.io/en/latest/>
2. Bishop, Christopher M. Neural Networks for Pattern Recognition. Оксфорд: Oxford University Press, 1995.
3. Ren, P. M., Wang, L., Fang, W., Song, S. L., Djahel, S. Новий підхід до підрахунку людей у реальному часі на основі Squeeze YOLO. Міжнародний журнал біоінспірованих, 2020, том X, випуск X, стор. X–X, 90-112 с.
4. TensorFlow: Сегментація зображень [Електронний ресурс]. – Режим доступу: <https://www.tensorflow.org/tutorials/images/segmentation>
5. Villan, A. F. (2019). YOLO для виявлення об'єктів - Освоєння OpenCV 4 з Python. [Електронний ресурс]. // Villan Режим доступу: <https://www.oreilly.com/library/view/mastering-opencv-4/9781789344912/c3e77fbd-1684-4609-9d70-5c553fe3a580.xhtml>
6. Javed R., Rahim M. S. M., Saba T., Rehman A. 2020 Netw. Model. Anal. Heal. Informatics Bioinforma. 9 1. 20-34 с.
7. Building Machine Learning and Deep Learning Models on Google Cloud Platform A Comprehensive Guide for Beginners Book © 2019, 4-16 с.
8. CUDA Toolkit Documentation. [Електронний ресурс]. Режим доступу: <https://docs.nvidia.com/cuda/>
9. NVIDIA cuDNN. [Електронний ресурс]. Режим доступу: <https://docs.nvidia.com/cudnn/index.html>
10. Zheng, D. 2022. Degree Project in Technology: Use of Improved Deep Learning and DeepSORT for Vehicle Estimation. Second cycle, 14-25 с.
11. Badrinarayanan, V., Kendall, A., Cipolla, R. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. arXiv preprint. 2015. [Електронний ресурс]. Режим доступу: <https://arxiv.org/abs/1511.00561>
12. OpenCV modules. [Електронний ресурс]. Режим доступу: <https://docs.opencv.org/4.x/index.html>

13. Object Detection with YOLO and OpenCV. [Електронний ресурс].
Режим доступу: <https://www.geeksforgeeks.org/object-detection-with-yolo-and-opencv/>
14. Ultralytics YOLO Docs. YOLOv4: High-Speed and Precise Object Detection. [Електронний ресурс]. Режим доступу: <https://docs.ultralytics.com/models/yolov4/>
15. Northwood, C. The Full Stack Developer: Your Essential Guide to the Everyday Skills Expected of a Modern Full Stack Web Developer, 23-45 с.
16. Shukla, Nishant, Fricklas, Kenneth. Machine Learning with TensorFlow. Книга. 272 стор. 2018., 1-25 с.
17. Milosevic, Nemanja. Introduction to Convolutional Neural Networks. Берклі, Каліфорнія: Apress, 2020.
18. Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. О'Рейлі, 2019.
19. Géron, Aurélien. TensorFlow 1.x Deep Learning Cookbook: Over 90 Unique Recipes to Solve Artificial-Intelligence Driven Problems with Python. Пакарді, 2017.
20. Vasilev, Ivan, and Slater, Daniel. Python Deep Learning. Пакарді, 2018.
21. Lane, Howard, and Hapke. Natural Language Processing in Action. Маннінг, 2020.
22. Buduma, Nithin, Nikhil Buduma, and Joe Papa. Fundamentals of Deep Learning: Designing Next-Generation Machine Learning Algorithms. O'Reilly, 2017.
23. Patterson, Josh, and Adam Gibson. Deep Learning: A Practitioner's Approach.* O'Reilly, 2017.
24. Docs. Object Detection YOLOv4. [Електронний ресурс]. Режим доступу: <https://nodejs.org/docs/latest/api/>

25. Nielsen, Michael. Neural Networks and Deep Learning. 2015. Determination Press. [Электронный ресурс]. Режим доступа: <http://neuralnetworksanddeeplearning.com/>
26. Elser, Sören. Frontend vs. Backend vs. Full-Stack – The Difference Explained. 2022. ElevateX GmbH.
27. Macrae, Callum. Vue.js: Up and Running: Building Accessible and Performant Web Apps. O'Reilly Media, 2018.

ДОДАТКИ

Лістинг файлу yolo_model.py

```

from functools import wraps
import numpy as np
import tensorflow as tf
from keras import backend as K
from keras.engine.base_layer import Layer
from keras.layers import Conv2D, Add, ZeroPadding2D, UpSampling2D, Concatenate,
MaxPooling2D
from keras.layers.advanced_activations import LeakyReLU
from keras.layers.normalization import BatchNormalization
from keras.models import Model
from keras.regularizers import l2

from Detect.utils import compose

class Mish(Layer):
    def __init__(self, **kwargs):
        super(Mish, self).__init__(**kwargs)
        self.supports_masking = True

    def call(self, inputs):
        return inputs * K.tanh(K.softplus(inputs))

    def get_config(self):
        config = super(Mish, self).get_config()
        return config

    def compute_output_shape(self, input_shape):
        return input_shape

@wraps(Conv2D)
def DarknetConv2D(*args, **kwargs):
    darknet_conv_kwargs = {'kernel_regularizer': l2(5e-4)}
    darknet_conv_kwargs['padding'] = 'valid' if kwargs.get('strides')==(2,2) else
'same'
    darknet_conv_kwargs.update(kwargs)
    return Conv2D(*args, **darknet_conv_kwargs)

def DarknetConv2D_BN_Leaky(*args, **kwargs):
    no_bias_kwargs = {'use_bias': False}
    no_bias_kwargs.update(kwargs)

```

```

    return compose(
        DarknetConv2D(*args, **no_bias_kwargs),
        BatchNormalization(),
        LeakyReLU(alpha=0.1))

def DarknetConv2D_BN_Mish(*args, **kwargs):
    no_bias_kwargs = {'use_bias': False}
    no_bias_kwargs.update(kwargs)
    return compose(
        DarknetConv2D(*args, **no_bias_kwargs),
        BatchNormalization(),
        Mish())

def resblock_body(x, num_filters, num_blocks, all_narrow=True):
    preconv1 = ZeroPadding2D((1,0),(1,0))(x)
    preconv1 = DarknetConv2D_BN_Mish(num_filters, (3,3), strides=(2,2))(preconv1)
    shortconv = DarknetConv2D_BN_Mish(num_filters//2 if all_narrow else num_filters,
    (1,1))(preconv1)
    mainconv = DarknetConv2D_BN_Mish(num_filters//2 if all_narrow else num_filters,
    (1,1))(preconv1)
    for i in range(num_blocks):
        y = compose(
            DarknetConv2D_BN_Mish(num_filters//2, (1,1)),
            DarknetConv2D_BN_Mish(num_filters//2 if all_narrow else num_filters,
    (3,3)))(mainconv)
        mainconv = Add()([mainconv,y])
    postconv = DarknetConv2D_BN_Mish(num_filters//2 if all_narrow else num_filters,
    (1,1))(mainconv)
    route = Concatenate()([postconv, shortconv])
    return DarknetConv2D_BN_Mish(num_filters, (1,1))(route)

def darknet_body(x):
    x = DarknetConv2D_BN_Mish(32, (3,3))(x)
    x = resblock_body(x, 64, 1, False)
    x = resblock_body(x, 128, 2)
    x = resblock_body(x, 256, 8)
    x = resblock_body(x, 512, 8)
    x = resblock_body(x, 1024, 4)
    return x

def make_last_layers(x, num_filters, out_filters):
    x = compose(
        DarknetConv2D_BN_Leaky(num_filters, (1,1)),

```

```

        DarknetConv2D_BN_Leaky(num_filters*2, (3,3)),
        DarknetConv2D_BN_Leaky(num_filters, (1,1)),
        DarknetConv2D_BN_Leaky(num_filters*2, (3,3)),
        DarknetConv2D_BN_Leaky(num_filters, (1,1)))(x)
y = compose(
    DarknetConv2D_BN_Leaky(num_filters*2, (3,3)),
    DarknetConv2D(out_filters, (1,1)))(x)
return x, y

def yolo4_body(inputs, num_anchors, num_classes):
    darknet = Model(inputs, darknet_body(inputs))
    #19x19 head
    y19 = DarknetConv2D_BN_Leaky(512, (1,1))(darknet.output)
    y19 = DarknetConv2D_BN_Leaky(1024, (3,3))(y19)
    y19 = DarknetConv2D_BN_Leaky(512, (1,1))(y19)
    maxpool1 = MaxPooling2D(pool_size=(13,13), strides=(1,1), padding='same')(y19)
    maxpool2 = MaxPooling2D(pool_size=(9,9), strides=(1,1), padding='same')(y19)
    maxpool3 = MaxPooling2D(pool_size=(5,5), strides=(1,1), padding='same')(y19)
    y19 = Concatenate()([maxpool1, maxpool2, maxpool3, y19])
    y19 = DarknetConv2D_BN_Leaky(512, (1,1))(y19)
    y19 = DarknetConv2D_BN_Leaky(1024, (3,3))(y19)
    y19 = DarknetConv2D_BN_Leaky(512, (1,1))(y19)
    y19_upsample = compose(DarknetConv2D_BN_Leaky(256, (1,1)), UpSampling2D(2))(y19)
    #38x38 head
    y38 = DarknetConv2D_BN_Leaky(256, (1,1))(darknet.layers[204].output)
    y38 = Concatenate()([y38, y19_upsample])
    y38 = DarknetConv2D_BN_Leaky(256, (1,1))(y38)
    y38 = DarknetConv2D_BN_Leaky(512, (3,3))(y38)
    y38 = DarknetConv2D_BN_Leaky(256, (1,1))(y38)
    y38 = DarknetConv2D_BN_Leaky(512, (3,3))(y38)
    y38 = DarknetConv2D_BN_Leaky(256, (1,1))(y38)
    y38_upsample = compose(DarknetConv2D_BN_Leaky(128, (1,1)), UpSampling2D(2))(y38)

    #76x76 head
    y76 = DarknetConv2D_BN_Leaky(128, (1,1))(darknet.layers[131].output)
    y76 = Concatenate()([y76, y38_upsample])
    y76 = DarknetConv2D_BN_Leaky(128, (1,1))(y76)
    y76 = DarknetConv2D_BN_Leaky(256, (3,3))(y76)
    y76 = DarknetConv2D_BN_Leaky(128, (1,1))(y76)
    y76 = DarknetConv2D_BN_Leaky(256, (3,3))(y76)
    y76 = DarknetConv2D_BN_Leaky(128, (1,1))(y76)

    #76x76 output

```

```

y76_output = DarknetConv2D_BN_Leaky(256, (3,3))(y76)
y76_output = DarknetConv2D(num_anchors*(num_classes+5), (1,1))(y76_output)

#38x38 output
y76_downsample = ZeroPadding2D(((1,0),(1,0)))(y76)
y76_downsample = DarknetConv2D_BN_Leaky(256, (3,3),
strides=(2,2))(y76_downsample)
y38 = Concatenate()([y76_downsample, y38])
y38 = DarknetConv2D_BN_Leaky(256, (1,1))(y38)
y38 = DarknetConv2D_BN_Leaky(512, (3,3))(y38)
y38 = DarknetConv2D_BN_Leaky(256, (1,1))(y38)
y38 = DarknetConv2D_BN_Leaky(512, (3,3))(y38)
y38 = DarknetConv2D_BN_Leaky(256, (1,1))(y38)

y38_output = DarknetConv2D_BN_Leaky(512, (3,3))(y38)
y38_output = DarknetConv2D(num_anchors*(num_classes+5), (1,1))(y38_output)

#19x19 output
y38_downsample = ZeroPadding2D(((1,0),(1,0)))(y38)
y38_downsample = DarknetConv2D_BN_Leaky(512, (3,3),
strides=(2,2))(y38_downsample)
y19 = Concatenate()([y38_downsample, y19])
y19 = DarknetConv2D_BN_Leaky(512, (1,1))(y19)
y19 = DarknetConv2D_BN_Leaky(1024, (3,3))(y19)
y19 = DarknetConv2D_BN_Leaky(512, (1,1))(y19)
y19 = DarknetConv2D_BN_Leaky(1024, (3,3))(y19)
y19 = DarknetConv2D_BN_Leaky(512, (1,1))(y19)

y19_output = DarknetConv2D_BN_Leaky(1024, (3,3))(y19)
y19_output = DarknetConv2D(num_anchors*(num_classes+5), (1,1))(y19_output)

yolo4_model = Model(inputs, [y19_output, y38_output, y76_output])

return yolo4_model

```

Лістинг файлу segnet.py

```

from keras.models import Model
from keras.layers import Input, Conv2D, DepthwiseConv2D, ZeroPadding2D,
BatchNormalization, Activation
import keras.backend as backend

IMAGE_LAYOUT = 'channels_last'

def custom_relu6(value):
    return backend.relu(value, max_value=6)

def convolutional_block(inputs, filters, alpha, kernel_size=(3, 3), stride=(1, 1)):
    """
    Побудова блоку згортки для вхідних даних.
    """
    channels_axis = 1 if IMAGE_LAYOUT == 'channels_first' else -1
    adjusted_filters = int(filters * alpha)
    padded_input = ZeroPadding2D(padding=(1, 1), name='conv_pad',
data_format=IMAGE_LAYOUT)(inputs)
    conv_layer = Conv2D(
        adjusted_filters, kernel_size, data_format=IMAGE_LAYOUT,
        padding='valid', use_bias=False, strides=stride, name='conv_layer'
    )(padded_input)
    norm_layer = BatchNormalization(axis=channels_axis,
name='batch_norm')(conv_layer)
    return Activation(custom_relu6, name='relu_activation')(norm_layer)

def depthwise_convolutional_block(inputs, pointwise_filters, alpha, depth_mult=1,
stride=(1, 1), block_id=1):
    """
    Побудова блоку глибинної згортки для моделі.
    """
    channels_axis = 1 if IMAGE_LAYOUT == 'channels_first' else -1
    adjusted_filters = int(pointwise_filters * alpha)
    padded_input = ZeroPadding2D((1, 1), data_format=IMAGE_LAYOUT, name=f'-
padding_block_{block_id}')(inputs)
    depthwise_layer = DepthwiseConv2D(
        (3, 3), data_format=IMAGE_LAYOUT, padding='valid',
        depth_multiplier=depth_mult, strides=stride, use_bias=False,
        name=f'depthwise_conv_{block_id}'
    )(padded_input)

```

```

        depthwise_norm = BatchNormalization(axis=channels_axis, name=f'-
depthwise_norm_{block_id}')(depthwise_layer)
        activated_depthwise = Activation(custom_relu6, name=f'-
depthwise_relu_{block_id}')(depthwise_norm)
        pointwise_layer = Conv2D(adjusted_filters, (1, 1), data_format=IMAGE_LAYOUT,
padding='same', use_bias=False, strides=(1, 1), name=f'-
pointwise_conv_{block_id}')(activated_depthwise)
        pointwise_norm = BatchNormalization(axis=channels_axis, name=f'-
pointwise_norm_{block_id}')(pointwise_layer)
        return Activation(custom_relu6, name=f'-
pointwise_relu_{block_id}')(pointwise_norm)

def build_mobilenet_encoder(input_height=224, input_width=224,
pretrained_weights='imagenet'):
    """
    Побудова енкодера MobileNet із зазначеними параметрами.
    """
    alpha = 1.0
    depth_multiplier = 1
    input_tensor = Input(shape=(input_height, input_width, 3))
    x = convolutional_block(input_tensor, 32, alpha, stride=(2, 2))
    x = depthwise_convolutional_block(x, 64, alpha, depth_multiplier, block_id=1)
    feature1 = x
    x = depthwise_convolutional_block(x, 128, alpha, depth_multiplier, stride=(2, 2),
block_id=2)
    x = depthwise_convolutional_block(x, 128, alpha, depth_multiplier, block_id=3)
    feature2 = x
    x = depthwise_convolutional_block(x, 256, alpha, depth_multiplier, stride=(2, 2),
block_id=4)
    x = depthwise_convolutional_block(x, 256, alpha, depth_multiplier, block_id=5)
    feature3 = x
    x = depthwise_convolutional_block(x, 512, alpha, depth_multiplier, stride=(2, 2),
block_id=6)
    for i in range(7, 12):
        x = depthwise_convolutional_block(x, 512, alpha, depth_multiplier,
block_id=i)
        feature4 = x
    x = depthwise_convolutional_block(x, 1024, alpha, depth_multiplier, stride=(2,
2), block_id=12)
    x = depthwise_convolutional_block(x, 1024, alpha, depth_multiplier, block_id=13)
    feature5 = x

    return input_tensor, [feature1, feature2, feature3, feature4, feature5]

```

```

DATA_FORMAT = 'channels_last'
def decoder_segnet(features, num_classes, upsampling_steps=3):
    """
    Декодер SegNet для розгортання характеристик до початкового розміру.
    """
    assert upsampling_steps >= 2
    decoded = features
    decoded = ZeroPadding2D((1, 1), data_format=DATA_FORMAT)(decoded)
    decoded = Conv2D(512, (3, 3), padding='valid', data_format=DATA_FORMAT)(decoded)
    decoded = BatchNormalization()(decoded)
    # Перше збільшення розміру hw до 1/8
    decoded = UpSampling2D((2, 2), data_format=DATA_FORMAT)(decoded)
    decoded = ZeroPadding2D((1, 1), data_format=DATA_FORMAT)(decoded)
    decoded = Conv2D(256, (3, 3), padding='valid', data_format=DATA_FORMAT)(decoded)
    decoded = BatchNormalization()(decoded)
    # Додаткові етапи збільшення розміру hw
    for _ in range(upsampling_steps - 2):
        decoded = UpSampling2D((2, 2), data_format=DATA_FORMAT)(decoded)
        decoded = ZeroPadding2D((1, 1), data_format=DATA_FORMAT)(decoded)
        decoded = Conv2D(128, (3, 3), padding='valid',
data_format=DATA_FORMAT)(decoded)
        decoded = BatchNormalization()(decoded)
    # Останнє збільшення розміру hw до 1/2
    decoded = UpSampling2D((2, 2), data_format=DATA_FORMAT)(decoded)
    decoded = ZeroPadding2D((1, 1), data_format=DATA_FORMAT)(decoded)
    decoded = Conv2D(64, (3, 3), padding='valid', data_format=DATA_FORMAT)(decoded)
    decoded = BatchNormalization()(decoded)

    # Вивід результату з розміром h_input/2, w_input/2, num_classes
    output = Conv2D(num_classes, (3, 3), padding='same',
data_format=DATA_FORMAT)(decoded)

    return output

def build_segnet_model(num_classes, encoder_fn, input_height=416, input_width=608,
encoder_level=3):
    """
    Функція для побудови моделі SegNet.
    """
    # Отримання вхідних даних та рівнів з енкодера
    input_img, encoder_levels = encoder_fn(input_height=input_height,
input_width=input_width)

```

```

# Вибір потрібного рівня з енкодера
features = encoder_levels[encoder_level]
# Побудова декодера
decoded_output = decoder_segnet(features, num_classes, upsampling_steps=3)
# Формування вихідного результату
reshaped_output = Reshape((int(input_height / 2) * int(input_width / 2), -
1))(decoded_output)
final_output = Softmax()(reshaped_output)
# Створення моделі
segnet_model = Model(input_img, final_output)
return segnet_model

def mobilenet_based_segnet(n_classes, input_height=224, input_width=224,
encoder_level=3):
    """
    Створення SegNet на основі MobileNet.
    """
    model = build_segnet_model(n_classes, build_mobilenet_encoder,
input_height=input_height, input_width=input_width, encoder_level=encoder_level)
    model.model_name = "mobilenet_segnet"
    return model

```

```

import numpy as np
import scipy.linalg

class BoundingBoxDetection:
    def __init__(self, bbox, score, descriptor, label):
        self.bbox = np.asarray(bbox, dtype=np.float32)
        self.score = float(score)
        self.descriptor = np.asarray(descriptor, dtype=np.float32)
        self.label = label
    def to_bbox_coordinates(self):
        coordinates = self.bbox.copy()
        coordinates[2:] += coordinates[:2]
        return coordinates
    def to_center_aspect(self):
        coordinates = self.bbox.copy()
        coordinates[:2] += coordinates[2:] / 2
        coordinates[2] /= coordinates[3]
        return coordinates

chi_square_95 = { 1: 3.8415, 2: 5.9915, 3: 7.8147, 4: 9.4877, 5: 11.070, 6: 12.592,
7: 14.067, 8: 15.507, 9: 16.919 }

class BoxTracker(object):

    def __init__(self):
        dims, timestep = 4, 1.0
        self.motion_matrix = np.eye(2 * dims, 2 * dims)
        for i in range(dims):
            self.motion_matrix[i, dims + i] = timestep
        self.update_matrix = np.eye(dims, 2 * dims)
        self.position_weight = 1. / 20
        self.velocity_weight = 1. / 160

    def initialize_track(self, observation):
        initial_position = observation
        initial_velocity = np.zeros_like(initial_position)
        initial_state = np.r_[initial_position, initial_velocity]
        uncertainties = [
            2 * self.position_weight * observation[3],
            2 * self.position_weight * observation[3],
            1e-2,

```

```

        2 * self.position_weight * observation[3],
        10 * self.velocity_weight * observation[3],
        10 * self.velocity_weight * observation[3],
        1e-5,
        10 * self.velocity_weight * observation[3]
    ]
    covariance_matrix = np.diag(np.square(uncertainties))
    return initial_state, covariance_matrix

def predict_state(self, state, covariance):
    position_uncertainties = [
        self.position_weight * state[3],
        self.position_weight * state[3],
        1e-2,
        self.position_weight * state[3]
    ]
    velocity_uncertainties = [
        self.velocity_weight * state[3],
        self.velocity_weight * state[3],
        1e-5,
        self.velocity_weight * state[3]
    ]
    motion_covariance = np.diag(np.square(np.r_[position_uncertainties,
velocity_uncertainties]))
    predicted_state = np.dot(self.motion_matrix, state)
    predicted_covariance = np.linalg.multi_dot((
        self.motion_matrix, covariance, self.motion_matrix.T)) +
motion_covariance
    return predicted_state, predicted_covariance

def project_state(self, state, covariance):
    uncertainty = [
        self.position_weight * state[3],
        self.position_weight * state[3],
        1e-1,
        self.position_weight * state[3]
    ]
    innovation_covariance = np.diag(np.square(uncertainty))
    projected_mean = np.dot(self.update_matrix, state)
    projected_covariance = np.linalg.multi_dot((
        self.update_matrix, covariance, self.update_matrix.T))
    return projected_mean, projected_covariance + innovation_covariance

```

```

def correct_state(self, predicted_state, predicted_covariance, measurement):
    projected_mean, projected_covariance = self.project_state(predicted_state,
predicted_covariance)
    cholesky_factor, lower = scipy.linalg.cho_factor(
        projected_covariance, lower=True, check_finite=False)
    kalman_gain = scipy.linalg.cho_solve(
        (cholesky_factor, lower), np.dot(predicted_covariance,
self.update_matrix.T).T,
        check_finite=False).T
    innovation = measurement - projected_mean
    updated_state = predicted_state + np.dot(innovation, kalman_gain.T)
    updated_covariance = predicted_covariance - np.linalg.multi_dot((
        kalman_gain, projected_covariance, kalman_gain.T))
    return updated_state, updated_covariance

def calculate_gating_distance(self, predicted_state, predicted_covariance,
measurements, use_position_only=False):
    predicted_state, predicted_covariance = self.project_state(predicted_state,
predicted_covariance)
    if use_position_only:
        predicted_state, predicted_covariance = predicted_state[:2],
predicted_covariance[:2, :2]
        measurements = measurements[:, :2]
    cholesky_decomp = np.linalg.cholesky(predicted_covariance)
    diff = measurements - predicted_state
    transformed_diff = scipy.linalg.solve_triangular(
        cholesky_decomp, diff.T, lower=True, check_finite=False,
        overwrite_b=True)
    squared_distance = np.sum(transformed_diff * transformed_diff, axis=0)
    return squared_distance

```

Лістинг файлу main.py

```

import json
import warnings
from collections import deque
warnings.filterwarnings("ignore")
from Detect.yolo import YOLOv4Detector
import Track.video as video
from Track.DeepSort.deep_sort import nn_matching
from Track.DeepSort.deep_sort.tracker import MultiTargetTracker
from Track.DeepSort.tools import generate_detections as gdet
from Track.utils import *
from Segment.segnet_mobile.segnet import mobilenet_based_segnet
from Segment.zebra_crossing import IMG_WIDTH, IMG_HEIGHT, NUM_CLASSES
from background import static_process
from utils import *
max_cosine_dist = 0.5
nn_budget = None
nms_max_overlap = 0.45
min_box_area = 2500
tracker_database = {}
vehicle_centers = [deque(maxlen=30) for _ in range(9999)]
def load_models():
    yolo_detector = YOLOv4Detector()
    metric = nn_matching.NeighborDistanceMetric("cosine", max_cosine_dist, nn_budget)
    tracker_instance = MultiTargetTracker(metric)
    model_file = "Tracking/DeepSort/model_data/market1501.pb"
    feature_encoder = gdet.create_bbox_feature_extractor(model_file, batch_size=1)
    segmentation_model = mobilenet_based_segnet(n_classes=NUM_CLASSES,
input_height=IMG_HEIGHT, input_width=IMG_WIDTH)
    segmentation_model.load_weights("Segmentation/segnet_mobile/weights.h5")
    return yolo_detector, tracker_instance, feature_encoder, segmentation_model
def get_video_fps(video_filepath):
    command = f"ffprobe -v error -select_streams v -show_entries stream=r_frame_rate
-of json {video_filepath}"
    response = os.popen(command).read()
    data = json.loads(response)
    fps = eval(data.get('streams')[0].get('r_frame_rate'))
    return fps

def process_video(video_filepath, background_image, output_directory, models, fps):
    result_root = output_directory
    create_directory_if_not_exists(result_root)

```

```

# Load video
video_loader = video.Video(video_filepath)
save_directory = os.path.join(output_directory, "frame")
create_directory_if_not_exists(save_directory)
frame_id = 0
timer = Stopwatch()
car_info = ""
yolo_detector, tracker_instance, feature_encoder, segmentation_model = models
zebra_rect, traffic_lights_bboxes, zebra_box, lanes =
static_process(background_image, yolo_detector, segmentation_model)
yolo_detector.set_target_classes(["person", "car"])
for frame in video_loader:
    if frame_id % 20 == 0:
        print(f"{frame_id} ({1. / max(1e-5, timer.avg_time):.2f} fps)")
        timer.start()
        update_tracker(frame, yolo_detector, feature_encoder, tracker_instance,
nms_max_overlap)
        traffic_light_state = get_environment(frame, traffic_lights_bboxes)
        online_car_ids = []
        online_person_ids = []
        frame_id, car_info = process_frame(
            frame, tracker_instance, car_info,
            traffic_light_state, online_car_ids,
            online_person_ids, zebra_rect, zebra_box, lanes, traffic_lights_bboxes,
            frame_id, timer, save_directory, fps)
    with open(os.path.join(output_directory, "result.txt"), "w") as f:
        f.write(car_info)
    output_video_path = os.path.join(output_directory, "output_video.mp4")
    cmd_str = f"ffmpeg -f image2 -i {os.path.join(output_directory,
'frame')}/%05d.jpg -b 5000k -c:v libx264 {output_video_path}"
    os.system(cmd_str)
def main(args):
    models = load_models()
    video_filepath = args.input_video
    background_image_path = args.input_background
    output_directory = args.output_dir
    fps = get_video_fps(video_filepath)
    process_video(video_filepath, background_image_path, output_directory, models,
fps)
if __name__ == '__main__':
    main(get_args())

```

Лістинг файлу MainPage.vue

```

<template>
  <div id="MainPage">
    <Menu/>
    <Header/>
    <div class="main">
      <div class="main-container">
        <div class="main-container-info">
          <h2>ПРОЦЕС ЗАВАНТАЖЕННЯ</h2>
          <p>
            Для завантаження файлів просто виберіть потрібні відео та зображення. Ва-
            жли-во, щоб відео було у форматах, підтримуваних нашою системою (наприклад,
            <span>.mp4</span>, <span>.avi</span>, <span>.mov</span>), а зображення – у стандарт-
            них форматах, таких як <span>.jpg</span> або <span>.png</span>.
            <br><br>Завантажте файли, натискаючи відповідні кнопки нижче, і почніть
            обро-бку вже зараз.
          </p>
        </div>
        <div class="main-container-download">
          <div class="main-container-download-video" :class="videoStyle">
            <ele-upload-video
              :fileSize="20"
              :responseFn="handleResponse"
              action="http://127.0.0.1:8000/upload/video"
              name="video"
              :value="video"
            />
            <el-tag class="infotag" v-if="!VideoUploadSuccess" type="info"
              effect="dark">Будь ласка, завантажте відео для обробки.</el-tag>
            <el-tag class="infotag" v-if="VideoUploadSuccess" type="success"
              effect="dark">Відео успішно завантажено.</el-tag>
          </div>
          <div class="main-container-download-image" :class="imageStyle">
            <div id="ele-upload-img">
              <el-upload

                class="avatar-uploader"
                action="http://127.0.0.1:8000/upload/image"
                :show-file-list="false"
                :on-success="handleUploadSuccess"
                :on-error="handleUploadError"
                name="image"

```

```

    >
    
    <i v-else class="el-icon-plus avatar-uploader-icon"></i>
  </el-upload>
</div>
<el-tag class="infotag" v-if="!ImageUploadSuccess" type="info"
effect="dark">Будь ласка, завантажте зображення фону.</el-tag>
<el-tag class="infotag" v-if="ImageUploadSuccess" type="success"
effect="dark">Зображення успішно завантажено.</el-tag>
</div>
</div>
</div>
<div class="main-container-info">
  <p>
    Натисніть кнопку <span>«Почати обробку»</span>.
    Після завантаження файлів система почне автоматичну обробку. Ви отримаєте
    ре-зультат, який допоможе вам аналізувати ситуацію на дорозі, виявляти порушення пра-
    вил ру-ху, відстежувати транспортні засоби та пішоходів, а також вивчати інші аспекти
    дорожнього руху. Час обробки залежатиме від розміру відео та кількості зображень.
  </p>
</div>
<div>
  <a href="javascript:void(0)" class="main-button"
@click="chuliButtonHandler">Почати обробку</a>
</div>
</div>
<Footer/>
</div>
</template>

```

```

<script>
import axios from "axios";
import Menu from "@/Components/Menu.vue";
import Header from "@/Components/Header.vue";
import Footer from "@/Components/Footer.vue";
export default {
  name: "MainPage",
  components: {
    Menu,
    Header,
    Footer,
  },
  data() {

```

```

return {
  video: null,
  VideoUploadSuccess: false,
  ImageUploadSuccess: false,
  imageUrl: ""
};
},
computed: {
  videoStyle: function() {
    return {
      borderstyle: !this.VideoUploadSuccess,
      borderstylesuccess: this.VideoUploadSuccess
    };
  },
  imageStyle: function() {
    return {
      borderstyle: !this.ImageUploadSuccess,
      borderstylesuccess: this.ImageUploadSuccess
    };
  }
},
methods: {
  handleUploadError(error) {
    this.$message({
      type: "error",
      message: "Не вдалося завантажити зображення, будь ласка, оновіть сторінку та спробуйте ще раз."
    });
    console.log("error", error);
  },
  handleResponse(response, file) {
    this.video = "/IncomingData/upload_video.mp4";
    this.VideoUploadSuccess = true;
  },
  handleUploadSuccess(response, file, fileList) {
    this.$message.success("Зображення успішно завантажено!");
    this.ImageUploadSuccess = true;
    this.imageUrl = URL.createObjectURL(file.raw);
  },
  chuliButtonHandler() {
    if (this.VideoUploadSuccess && this.ImageUploadSuccess) {
      let loading = this.$loading({
        lock: true,

```

```

        text: "Обробка триває",
        spinner: "el-icon-loading",
        background: "rgba(0, 0, 0, 0.75)"
    });
    axios({
        url: "http://127.0.0.1:8000/api/process",
        method: "get",
        timeout: 0
    })
        .then(response => {
            if (response.data.message === "process finish") {
                loading.close();
                this.$router.push({ path: "/result" });
            }
        })
        .catch(error => {
            console.log(error);
        });
    } else {
        this.$message({
            type: "error",
            message: "Будь ласка, спочатку завантажте відео та зображення!"
        });
        /* this.$router.push({ path: "/result" }); для налагодження*/
    }
},
mounted() {
    axios
        .get("http://127.0.0.1:8000/api/cleardir")
        .then()
        .catch(error => {
            console.log(error);
        });
}
};
</script>

```

Лістинг файлу ResultPage.vue

```

<template>
  <div class="root">
    <Menu/>

```

```

<div class="return">
  <el-button type="primary" icon="el-icon-arrow-left" @click="handleReturn">Поне-
редня сторінка</el-button>
</div>
<div class="container-title"><h2>Оброблені дані</h2></div>
<div class="container-result">
  <div class="videoplaywrapper">
    <video
      id="videoplayer"
      :autoplay="false"
      :src="videoUrl"
      ref="videoPlayer"
      height="480"
      width="854"
      autoplay loop muted
      src="../../Server/Processes/ProcessedData/output_video.mp4"
    >Ваш браузер не підтримує тег Video.</video>

    <div class="video-controls">
      <button @click="rewindVideo"><i class="fas fa-undo-alt fa-2x"></i></button>
      <button @click="playVideo"><i class="fas fa-play fa-2x"></i></button>
      <button @click="pauseVideo"><i class="fas fa-pause fa-2x"></i></button>
      <button @click="fastForwardVideo"><i class="fas fa-redo-alt fa-
2x"></i></button>
    </div>
  </div>

  <div class="table-wrapper">
    <ul class="table-target">
      <li><a class="link-table" @click.prevent="showCarOrPerson('show_car');"
: class="{ 'link-active': selectedCarOrPerson === 'show_car' }">Транспорт</a></li>
      <li><a class="link-table" @click.prevent="showCarOrPerson('show_person');"
: class="{ 'link-active': selectedCarOrPerson === 'show_person' }">Пішоходи</a></li>
    </ul>
    <div class="table-container">
      <table class="table-main" :class="{ 'deactive': selectedTableOrPhoto ===
'show_photo' }">
        <thead>
          <tr>
            <th class="course-date" style="width: 120px;">Година</th>
            <th style="width: 90px;">ID</th>
            <th>Порушення правил дорожнього руху</th>

```

```

    </tr>
</thead>
<tbody :class="{ 'deactive': selectedCarOrPerson === 'show_person' }">
    <!-- Перевірка, чи є дані в таблиці -->
    <tr v-if="tableData.length === 0">
        <td colspan="3" style="text-align: center; font-style: italic; color:
#aaa;">
            Порушників не виявлено
        </td>
    </tr>
    <tr v-for="(row) in tableData">
        <td><a href="#" @click.prevent="seekToTime(row.timeStamp)">{{
row.timeStamp }}</a></td>
        <td>{{ row.carID }}</td>
        <td>{{ row.illegalInfo }}</td>
    </tr>
</tbody>
<tbody :class="{ 'deactive': selectedCarOrPerson === 'show_car' }">
    <tr>
        <td><a href="#" @click.prevent="seekToTime(1.25)">1.25</a></td>
        <td>101</td>
        <td>Транспортний засіб не поступився пішоходам</td>
    </tr>
    <tr>
        <td><a href="#" @click.prevent="seekToTime(5.50)">5.50</a></td>
        <td>102</td>
        <td>Транспортний засіб рухається не за правилами</td>
    </tr>
    <tr>
        <td><a href="#" @click.prevent="seekToTime(12.30)">12.30</a></td>
        <td>103</td>
        <td>Транспортний засіб проїхав на червоне світло</td>
    </tr>
    <tr>
        <td><a href="#" @click.prevent="seekToTime(20.75)">20.75</a></td>
        <td>104</td>
        <td>Транспортний засіб рухається не за правилами</td>
    </tr>
</tbody>
</table>
<div :class="{ 'deactive': selectedTableOrPhoto === 'show_table' }">
    <div :class="{ 'deactive': selectedCarOrPerson === 'show_person' }">
        <div v-if="tableData.length === 0">

```

```

        <p style="
        text-align: center;
        font-style: italic;
        color: #aaa;
        font-size: 22px;
        margin-top: 92px;
        font-family: Consolas, 'Courier New', monospace;
        /* font-weight: 600; */
        /* color: #0297fc; */
        text-shadow: 1px 1px 2px #000;
        ">
            Порушників не виявлено
        </p>
    </div>

</div>

<div :class="{ 'deactive': selectedCarOrPerson === 'show_car' }">
    <div v-if="tableData.length === 0">
        <p style="
        text-align: center;
        font-style: italic;
        color: #aaa;
        font-size: 22px;
        margin-top: 92px;
        font-family: Consolas, 'Courier New', monospace;
        /* font-weight: 600; */
        /* color: #0297fc; */
        text-shadow: 1px 1px 2px #000;
        ">
            Порушників не виявлено
        </p>
    </div>
</div>

</div>
</div>

<ul class="table-target">
    <li><a
        class="link-table"
        :class="{ 'link-active': selectedTableOrPhoto === 'show_table' }"
        @click.prevent="showTableOrPhoto('show_table')"
```

```

        >Таблиця</a></li>
      <li><a
        class="link-table"
        :class="{ 'link-active': selectedTableOrPhoto === 'show_photo' }"
        @click.prevent="showTableOrPhoto('show_photo')"
        >Знімки</a></li>
    </ul>
  </div>
</div>

<div class="downloadbutton">
  <div class="downloadvideo">
    <button type="primary" round @click="handleDownloadVideo">Завантажити вихідне
відео</button>
  </div>
  <div class="downloadtxt">
    <button type="primary" round @click="handleDownloadTxt">Завантажити вихідний
ре-зультат</button>
    <button @click="saveProject">Зберегти</button>
    <p v-if="message">{{ message }}</p>
  </div>
</div>
<Footer />
</div>
</template>

<script>
import axios from "axios";
import Menu from "@/Components/Menu.vue";
import Footer from "@/Components/Footer.vue";
export default {
  name: "result",
  components: {
    Menu,
    Footer
  },
  data() {
    return {
      videoUrl: "",
      tableData: [],
      timer: 26,
      message: '',
      selectedCarOrPerson: 'show_car',

```

```

        selectedTableOrPhoto: 'show_table'
    };
},
mounted: function() {
    this.videoUrl = "/ProcessedData/output_video.mp4";
    axios
        .get("http://127.0.0.1:8000/api/videodata")
        .then(response => {
            this.tableData = JSON.parse(JSON.stringify(response.data.videodata));
        })
        .catch(error => {
            console.log(error);
        });
},
methods: {
    // Відтворити відео
    playVideo() {
        const video = this.$refs.videoPlayer;
        video.play();
    },
    // Пауза відео
    pauseVideo() {
        const video = this.$refs.videoPlayer;
        video.pause();
    },

    // Перемотати відео назад на 2 секунди
    rewindVideo() {
        const video = this.$refs.videoPlayer;
        video.currentTime = Math.max(0, video.currentTime - 2);
    },

    // Перемотати відео вперед на 2 секунди
    fastForwardVideo() {
        const video = this.$refs.videoPlayer;
        video.currentTime = Math.min(video.duration, video.currentTime + 2);
    },

    showCarOrPerson(divName) {
        this.selectedCarOrPerson = divName;
    },
    showTableOrPhoto(divName) {
        this.selectedTableOrPhoto = divName;
    }
}

```

```

    },

    seekToTime(time) {
        const video = this.$refs.videoPlayer;
        if (video) {
            video.currentTime = time;
            video.play();
        }
    },

    tableRowClassName({row, rowIndex}) {
        if (row.date >= new Date()) {
            return 'warning-row';
        } else if (rowIndex === 3) {
            return 'success-row';
        }
        return '';
    },

    handleReturn() {
        this.$router.push({ path: "/" });
    },

    handleDownloadVideo() {
        let aTag = document.createElement("a");
        aTag.href = "ProcessedData/output_video.mp4";
        aTag.download = "Вихідне відео.mp4";
        document.body.appendChild(aTag);
        aTag.click();
        document.body.removeChild(aTag);
    },

    handleDownloadTxt() {
        let aTag = document.createElement("a");
        aTag.href = "/ProcessedData/result.txt";
        aTag.download = "Вихідний результат.txt";
        document.body.appendChild(aTag);
        aTag.click();
        document.body.removeChild(aTag);
    },

    async saveProject() {
        // Відкриття prompt для введення назви проєкту
        const name = prompt('Введіть назву проєкту:');
        if (!name) {

```

```

    alert('Назва проєкту не може бути пустою!');
    return;
}

try {
    // Відправка даних на сервер
    const response = await axios.post('http://localhost:8000/savedata', {
        name,
        timer: this.timer,
    });

    // Показ повідомлення про успіх
    this.message = response.data.message;
} catch (error) {
    console.error('Помилка збереження:', error);
    this.$message({
        type: "error",
        message: "Помилка збереження проєкту"
    });
}
}

};
</script>

```

Лістинг файлу index.js

```
const express = require('express');
const path = require('path');
const fs = require('fs');
const { spawn } = require('child_process');
const multer = require('multer');
const extractTableData = require('./processfile.js');
const { convertToMp4, isH264 } = require('./format.js');
const config = require('./setting.js');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const cors = require('cors');
const history = require('connect-history-api-fallback');

const app = express();
const upload = multer({ dest: './Processes/IncomingData/' });

// Enable history mode for SPA compatibility
app.use(history({
  verbose: true,
  htmlAcceptHeaders: ['text/html', 'application/xhtml+xml']
}));

// Serve static files
app.use(express.static(path.join(__dirname, '../dist')));
app.use(express.static(path.join(__dirname, '../Processes')));

// Set CORS headers
app.all('*', (req, res, next) => {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Headers", "X-Requested-With");
  res.header("Access-Control-Allow-Methods", "PUT,POST,GET,DELETE,OPTIONS");
  res.header("X-Powered-By", '3.2.1');
  res.header("Content-Type", "application/json;charset=utf-8");
  next();
});

// Middleware
app.use(bodyParser.json());
app.use(cors());

// Start server
```

```

app.listen(8000, () => {
  console.log("Server is running on port 8000");
});

// Video upload endpoint
app.post('/upload/video', upload.single("video"), (req, res) => {
  const extension = req.file.originalname.split('.').pop();
  const newFilePath = path.join(path.dirname(req.file.path), "upload_video.mp4");

  if (extension === "avi") {
    convertToMp4(req.file.path, newFilePath, res);
  } else if (extension === "mp4") {
    if (isH264(req.file.path)) {
      fs.renameSync(req.file.path, newFilePath);
      res.send("Video uploaded successfully");
    } else {
      convertToMp4(req.file.path, newFilePath, res);
    }
  }
});

// Image upload endpoint
app.post('/upload/image', upload.single('image'), (req, res) => {
  const extension = req.file.originalname.split('.').pop();
  const renamedPath = path.join(path.dirname(req.file.path),
`upload_image.${extension}`);
  fs.renameSync(req.file.path, renamedPath);
  res.send("Image uploaded successfully");
});

// Video processing endpoint
app.get('/api/process', (req, res) => {
  const pythonExecutable = config.PYTHONEXE_PATH;
  const scriptPath = config.PYTHONFILE_PATH;
  const inputVideoPath = path.join(__dirname, "..", "Processes", "IncomingData",
"upload_video.mp4");
  const inputImagePath = path.join(__dirname, "..", "Processes", "IncomingData",
"upload_video.png");
  const outputDirectory = path.join(__dirname, "..", "Processes", "ProcessedData");

  const args = [
    scriptPath,
    '--input_video', inputVideoPath,

```

```

        '--input_background', inputImagePath,
        '--output_dir', outputDirectory
    ];

    console.log("Launching Python script...");

    const pythonProcess = spawn(pythonExecutable, args);

    pythonProcess.stdout.on('data', (data) => {
        console.log(`stdout: ${data}`);
    });

    pythonProcess.stderr.on('data', (data) => {
        console.error(`stderr: ${data}`);
    });

    pythonProcess.on('close', () => {
        res.json({ message: 'Processing complete' });
    });
});

// Endpoint to retrieve video data
app.get('/api/videodata', (req, res) => {
    const resultFilePath = path.join(__dirname, "..", "Processes", "ProcessedData",
    "result.txt");
    const tableData = [];
    extractTableData(tableData, resultFilePath);
    res.json({ videodata: tableData });
});

// Endpoint to download processed video
app.get("/api/downloadvideo", (req, res) => {
    const videoFilePath = path.join(__dirname, "..", "Processes", "ProcessedData",
    "output_video.mp4");
    res.download(videoFilePath, 'video.mp4');
    res.set({ "Content-Type": "application/octet-stream" });
});

// Endpoint to download processed text file
app.get("/api/downloadtxt", (req, res) => {
    const textFilePath = path.join(__dirname, "..", "Processes", "ProcessedData",
    "result.txt");
    res.download(textFilePath, 'result.txt');
});

```

```

    res.set({ "Content-Type": "application/octet-stream" });
  });

// Connect to MongoDB
mongoose.connect('mongodb://127.0.0.1:27017/TrafficSystem', {})
  .then(() => console.log('Successfully connected to MongoDB (TrafficSystem)'))
  .catch(err => {
    console.error('MongoDB connection error:', err.message);
    process.exit(1);
  });

// Database schema and model
const TrafficSchema = new mongoose.Schema({
  id: String,
  name: String,
  date: Date,
  timer: Number
});
const TrafficModel = mongoose.model('Traffic', TrafficSchema);

const processedDirectory = path.join(__dirname, '..', 'Processes', 'ProcessedData');
const savedDirectory = path.join(__dirname, '..', 'Saved');

// Save project data
app.post('/savedata', async (req, res) => {
  const { name, timer } = req.body;
  const timestamp = new Date();
  const projectId = `${timestamp.getFullYear()}${timestamp.getMonth() +
1}${timestamp.getDate()}${timestamp.getHours()}${timestamp.getMinutes()}${timestamp.g
etSeconds()}`;

  const newEntry = new TrafficModel({
    id: projectId,
    name,
    date: timestamp,
    timer
  });

  try {
    await newEntry.save();

    const projectDirectory = path.join(savedDirectory, projectId);
    fs.mkdirSync(projectDirectory, { recursive: true });
  }

```

```

    const files = fs.readdirSync(processedDirectory);
    files.forEach(file => {
        const source = path.join(processedDirectory, file);
        const destination = path.join(projectDirectory, file);
        if (fs.lstatSync(source).isDirectory()) {
            recursivelyCopyDirectory(path.join(processedDirectory, 'frame'),
path.join(projectDirectory, 'frame'));
        } else {
            fs.copyFileSync(source, destination);
        }
    });

    res.status(201).json({ message: 'Project saved successfully', project:
newEntry });
  } catch (err) {
    console.error('Error saving project:', err);
    res.status(500).json({ message: 'Error saving project' });
  }
});

function recursivelyCopyDirectory(source, destination) {
  if (!fs.existsSync(destination)) {
    fs.mkdirSync(destination, { recursive: true });
  }
  const entries = fs.readdirSync(source);
  entries.forEach(entry => {
    const sourcePath = path.join(source, entry);
    const destinationPath = path.join(destination, entry);
    if (fs.lstatSync(sourcePath).isDirectory()) {
      recursivelyCopyDirectory(sourcePath, destinationPath);
    } else {
      fs.copyFileSync(sourcePath, destinationPath);
    }
  });
}

// Clear uploaded and processed files
app.get("/api/cleardir", (req, res) => {
  const uploadDirectory = path.join(__dirname, "..", "Processes", "IncomingData");
  const uploadedFiles = fs.readdirSync(uploadDirectory);
  uploadedFiles.forEach(file => {
    fs.unlinkSync(path.join(uploadDirectory, file));
  });
});

```

```

});

const processedFiles = fs.readdirSync(processedDirectory);
processedFiles.forEach(file => {
  fs.unlinkSync(path.join(processedDirectory, file));
});

res.status(200).end();
});

function getTableData(tableData, filePath) {
  let fileContent = fs.readFileSync(filePath);
  fileContent.toString().split(/\s?\n/).forEach(function (item, index, array) {
    let itemArray = item.split(",");
    if (+itemArray[2] || +itemArray[3] || itemArray[4]) {
      if (+itemArray[2]) {
        tableData.push({
          timeStamp: itemArray[0],
          carID: itemArray[1],
          illegalInfo: "Транспортний засіб не поступився пішоходам"
        });
      }
      if (+itemArray[3]) {
        tableData.push({
          timeStamp: itemArray[0],
          carID: itemArray[1],
          illegalInfo: "Транспортний засіб рухається не за правилами"
        });
      }
      if (+itemArray[4]) {
        tableData.push({
          timeStamp: itemArray[0],
          carID: itemArray[1],
          illegalInfo: "Транспортний засіб проїхав на червоне світло"
        });
      }
    }
  })
}

function getTableData(tableData, filePath) {
  let fileContent = fs.readFileSync(filePath);
  fileContent.toString().split(/\s?\n/).forEach(function (item, index, array) {
    let itemArray = item.split(",");
    if (+itemArray[2] || +itemArray[3] || itemArray[4]) {

```

```

if (+itemArray[2]) {
    tableData.push({
        timeStamp: itemArray[0],
        carID: itemArray[1],
        illegalInfo: "Транспортний засіб не поступився пішоходам"
    });
} if (+itemArray[3]) {
    tableData.push({
        timeStamp: itemArray[0],
        carID: itemArray[1],
        illegalInfo: "Транспортний засіб рухається не за правилами"
    });
} if (+itemArray[4]) {
    tableData.push({
        timeStamp: itemArray[0],
        carID: itemArray[1],
        illegalInfo: "Транспортний засіб проїхав на червоне світло"
    });
}
}
}
}
}

```