МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інтелектуальна система розпізнавання образів у Web-контексті»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Ободяк В.К.**

Студента групи ІН.мз – 91с — **Матлахов В.І.**

СУМИ 2020

Сумський державний університет
(назва вузу)

Факультет *ЦЗДВФН* Кафедра *Комп'ютерних наук*
Спеціальність *«Інформатика»*

Затверджую:
зав.кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ
СТУДЕНТОВІ

Матлахову Віталію Івановичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інтелектуальна система розпізнавання образів у Web-контексті*
затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____
3. Вхідні данні до проекту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) аналітичний огляд методів розпізнавання образів; 2) постановка завдання й формування завдань дослідження; 3) опис основних положень, моделей нейронних мереж і критеріїв, що використовуються інтелектуальною системою; 4) розробка програмного забезпечення інтелектуальної системи; 5) аналіз результатів моделювання.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
_____
_____
_____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Аналітичний огляд методів розпізнавання образів. Постановка завдання й формування завдань дослідження.*		
2.	*Опис основних положень, моделей нейронних мереж і критеріїв, що використовуються інтелектуальною системою.*		
3.	*Розробка програмного забезпечення інтелектуальної системи*		
4.	*Аналіз результатів моделювання*		
5.	*Оформлення пояснювальної записки до дипломної роботи*		

Студент _____
(підпис)

Керівник роботи _____
(підпис)

РЕФЕРАТ

Записка: 68 стор., 33 рис., 6 табл., 7 додатків, 33 джерел.

Об’єкт дослідження — процес розпізнавання образів у Web-контексті.

Мета роботи — розробити та дослідити інтелектуальну систему для розпізнавання образів у Web-контексті в режимі реального часу.

Методи дослідження — в процесі проведення досліджень застосовувались технології проектування додатків, робота з мовами програмування JavaScript, Python, C та C++.

Результати — розроблено та досліджено інтелектуальну систему розпізнавання образів у Web-контексті в режимі реального часу у вигляді додатку до браузерів на базі Chromium з використанням серверних технологій.

ІНТЕЛЕКТУАЛЬНА СИСТЕМА РОЗПІЗНАВАННЯ ОБРАЗІВ У WEB-КОНТЕКСТІ, ДЕТЕКТОР ОБ'ЄКТІВ, DARKNET, YOLO, JAVASCRIPT, PYTHON

ЗМІСТ

ВСТУП

Комп’ютерний зір – це область штучного інтелекту, що дозволяє комп’ютерам та системам отримувати значущу інформацію з цифрових зображень, відео та інших візуальних входів, вживати заходів або робити рекомендації на основі цієї інформації використовуючи технології розпізнавання образів. Якщо штучний інтелект дозволяє комп’ютерам мислити, то комп’ютерний зір дозволяє їм бачити, спостерігати та розуміти.

Комп’ютерний зір працює майже так само, як людський зір, за винятком того, що люди мають перевагу. Людський зір під час навчання розрізняти предмети використовує також життєвий контекст, як далеко вони знаходяться, чи рухаються вони та багато іншого. Комп’ютерний зір навчає машини виконувати ці функції, але він повинен це робити як найшвидше за допомогою камер, даних та алгоритмів, а не сітківки, зорових нервів та зорової кори.

Комп’ютерне зір використовується у галузях, починаючи від сільського господарства і закінчуючи автомобілебудуванням, і ринок зростає. Очікується, що вона досягне 48,6 млрд. дол. США до 2022 року [1].

Розпізнавання образів потребує великої кількості даних. Комп’ютерний зір проводить аналіз даних знову і знову, поки не розпізнає відмінності і в кінцевому результаті розпізнає зображення. Наприклад, щоб навчити комп’ютер розпізнавати яблука, йому потрібно проаналізувати величезну кількість зображень яблук та предметів, пов’язаних з яблуками, щоб дізнатися відмінності та розпізнати яблуко.

Для цього використовуються дві основні технології: тип машинного навчання, який називається глибоким навчанням, та згорткова нейронна мережа (CNN).

Машинне навчання використовує алгоритмічні моделі, які дозволяють комп'ютеру навчатись контексту візуальних даних. Якщо через модель подається достатньо даних, комп'ютер проаналізує дані та навчиться відрізняти одне зображення від іншого. Алгоритми дозволяють машині вчитися самому, а не комусь програмувати її для розпізнавання зображення.

CNN допомагає машинному навчанню або моделі глибокого навчання аналізувати зображення, розбиваючи його на пікселі, яким присвоюються теги або мітки (label). Вона використовує мітки для виконання згортань (математична операція над двома функціями для отримання третьої функції) і робить прогнози щодо того, що «бачить». Нейронна мережа запускає згортки та перевіряє точність своїх прогнозів у серії ітерацій, поки прогнози не почнуть здійснюватися. Потім вона розпізнає або бачить зображення, подібні до людських.

Подібно до того, як людина бачить зображення на відстані, CNN спочатку розрізняє жорсткі краї та прості форми, а потім заповнює інформацію, виконуючи ітерації своїх прогнозів. CNN використовується для розуміння окремих зображень. Рекурентна нейронна мережа (RNN) використовується подібним чином для відео файлів, щоб допомогти комп'ютерам зрозуміти, як зображення в ряді кадрів пов'язані між собою.

Дана робота присвячена технологіям розпізнавання образів, використанню їх у web-контексті для вирішення таких задач як: статистичний аналіз web-ресурсів, батьківський контроль та інші.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Опис предметної області

Розпізнавання образів - це поширена проблема комп’ютерного зору, яка полягає у ідентифікації та визначенні об’єкта певних класів на зображенні. Локалізація об’єкта може здійснюватися різними способами, включаючи створення обмежувального вікна навколо об’єкта або позначення кожного пікселя на зображенні, що містить об’єкт (що називається сегментацією).

Розпізнавання образів вивчалося ще до популярності загорткових нейронних мереж в комп’ютерному зорі. Хоча CNN здатні автоматично виявляти більш складні та кращі функції, огляд звичайних методів бути корисним для кращого розуміння загальних принципів їх роботи.

Розпізнавання образів до глибокого навчання було багатоетапним процесом, починаючи з виявлення країв та вилучення об’єктів за допомогою таких методів, як SIFT (масштабонезалежне перетворення ознак), HOG (гістограма напрямлених градієнтів) тощо. Потім це зображення порівнювали з існуючими SVM шаблонами об’єктів, як правило, на багатомасштабних рівнях, щоб виявити та локалізувати об’єкти різних розмірів на зображенні [2].

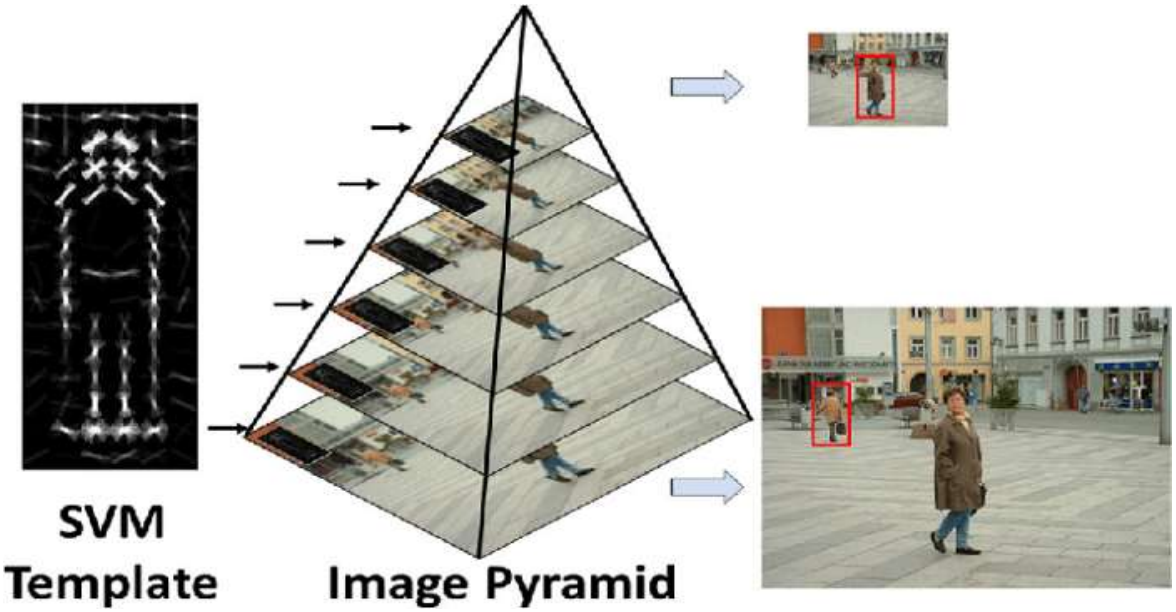


Рисунок 1.1 – Піраміда зображення з кількома масштабами [2].

Перетин над об'єднанням (IoU) або міра Жаккара - це оцінка, що використовується для вимірювання точності детектора об'єктів на певному наборі даних. Більш формально, для того, щоб застосувати міру Жаккара для оцінки довільного детектора об’єктів нам потрібні:

- істинні обмежувальні прямокутники із набору тестування, які описуюсь фактичне розташування об’єкту на зображенні;
- обмежувальні прямокутники, які описують прогнозоване розташування об’єкта детектором.

Ділення площі перекриття істинного та прогнозованого обмежувального прямокутника на площу їх об'єднання дає наш остаточний результат [3].

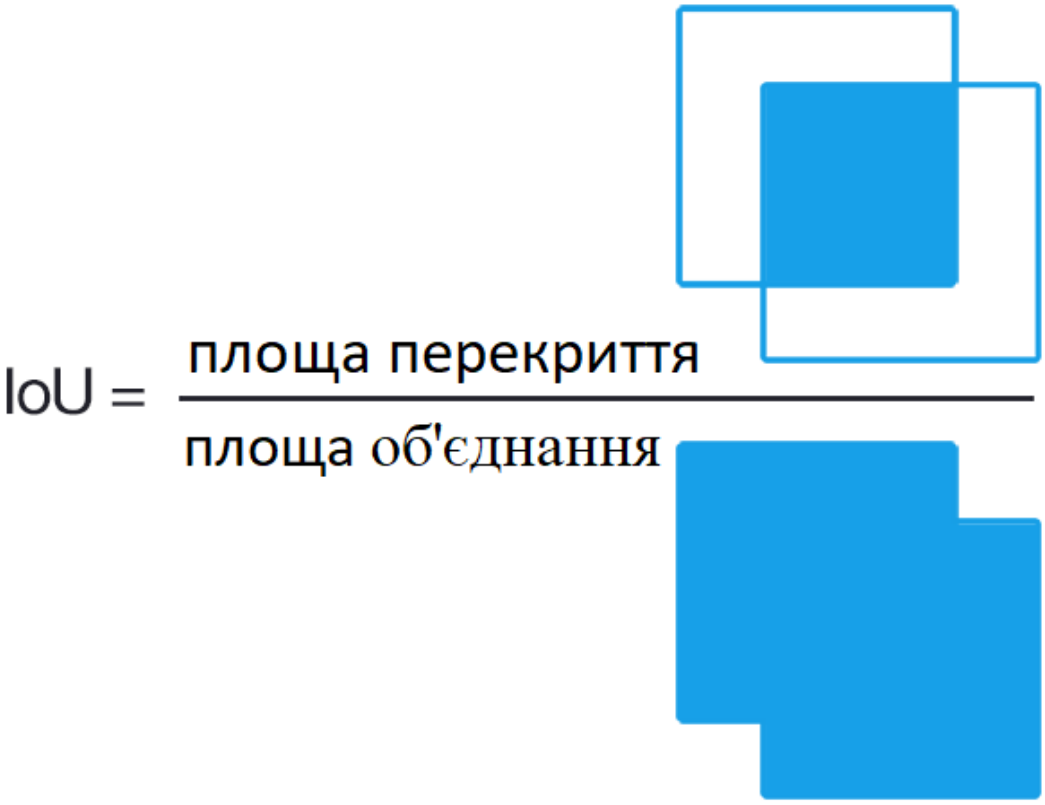


Рисунок 1.2 – Міра Жаккара схематично [3].

Середня точність (Average Precision, AP) – міра, яка описує наскільки точними є прогнози системи розпізнавання образів.

Середня повнота (Average Recall, AR) – міра, яка описує яку частку об’єктів система розпізнавання образів може розпізнати.

Середня точність та середня повнота – це дві загальні метрики, що використовуються для оцінки моделей розпізнавання.

Двоступеневе розпізнавання образів включає алгоритми, які спочатку ідентифікують обмежувальні прямокутники, які потенційно можуть містити об’єкти, а потім класифікують кожен обмежувальний прямокутник окремо. Для першого етапу потрібна регіональна мережа пропозицій, яка надає ряд регіонів, які потім передаються загальним архітектурам класифікації на основі глибокого навчання. Ієрархічний алгоритм групування в регіональній згортковій нейронній мережі (RCNN) (які є надзвичайно повільними), використання пулів CNN та ROI у Fast RCNN, якорі у Faster RCNN та багато інших методів та варіацій використовуються у цих регіональних мережах пропозицій (RPN). Відомо, що ці алгоритми розпізнають образи точніше, ніж їх одноетапні аналоги, але порівняно повільніші. З різними вдосконаленнями, пропонованими протягом багатьох років, поточним вузьким місцем у затримці роботи мереж двоступеневого виявлення об’єктів є крок RPN.

З урахуванням необхідності виявлення об’єктів у реальному часі було запропоновано багато архітектур одноступеневого виявлення об’єктів, таких як YOLO (та версії v2-v5) SSD, RetinaNet тощо, які намагаються поєднати крок виявлення та класифікації. Одним з головних досягнень цих алгоритмів було введення ідеї «регресування» прогнозованого обмежувального прямокутника об’єкта. Коли кожен обмежувальний прямокутник легко представляється з кількома значеннями (наприклад, xmin, xmax, ymin та ymax), стає простіше поєднувати крок виявлення та класифікації, що прискорює процес розпізнавання.

Наприклад, архітектура YOLO розділяє все зображення на менші сітчасті поля. Для кожної комірки сітки він передбачає ймовірності класу та координати x та y кожного обмежувального прямокутника, що проходить через цю комірку сітки [4].

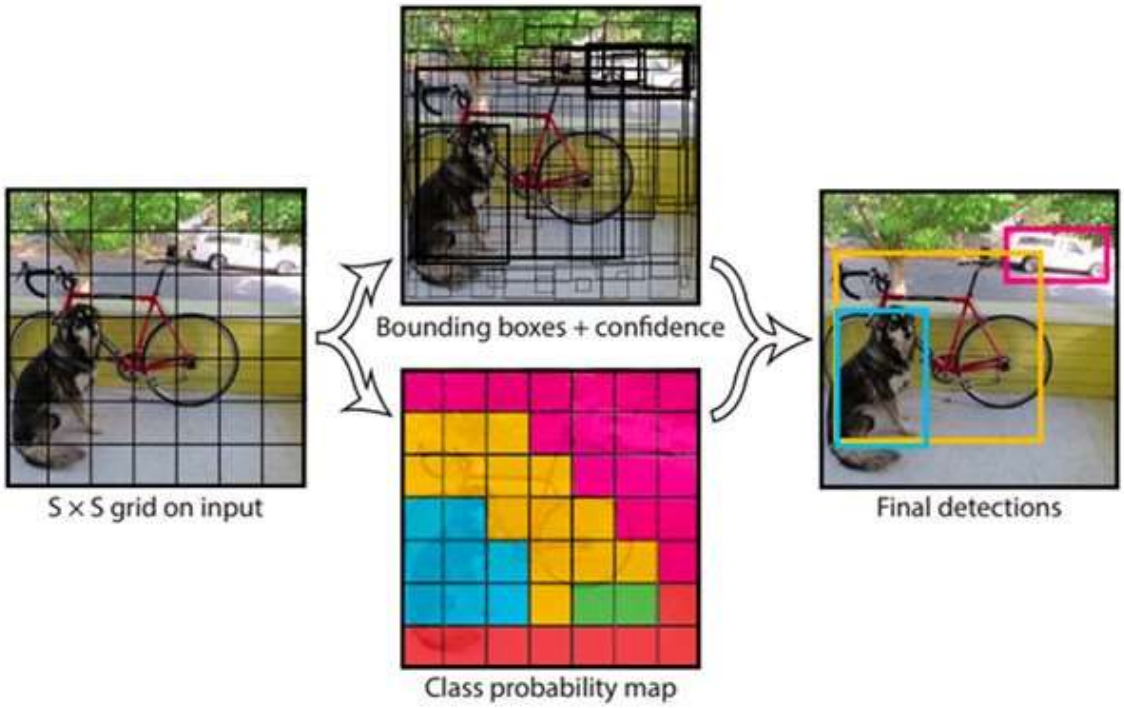


Рисунок 1.3 – Розпізнавання образів за допомогою YOLO архітектури [4].

Ці модифікації дозволяють однокроковим детекторам працювати швидше, а також працювати на глобальному рівні. Однак, оскільки вони працюють не на кожному прогнозованому обмежувальному прямокутнику окремо, це може призвести до їхньої гіршої роботи у випадку розташуванні менших об'єктів або подібних об'єктів поблизу.

Розпізнавання образів на основі теплової карти можна, в деякому сенсі, розглядати як продовження одноетапного виявлення об'єктів. У той час як алгоритми розпізнавання образів на основі одного етапу намагаються безпосередньо регресувати координати прогнозованого обмежувального прямокутника, виявлення об’єктів на основі теплової карти забезпечує розподіл ймовірності кутів / центру прогнозованого обмежувального прямокутника.

На основі розташування цих кутових / центральних піків на теплових картах передбачаються обмежувальні прямокутники. Оскільки для кожного класу можна створити різну теплову карту, цей метод також поєднує розпізнавання та класифікацію образів. Хоча розпізнавання образів на основі теплової карти в даний час є провідним новим дослідженням, воно все ще не

таке швидке, як звичайні одноетапні алгоритми виявлення об’єктів. Це пов'язано з тим, що ці алгоритми вимагають більш складних архітектур CNN, щоб отримати достатню точність.

На рисунку 1.4 схематично демонструється робота одної із таких архітектур. Мережа передбачає чотири екстремальні точки теплових карт (зверху теплова карта, накладена на вхідне зображення), і одну центральну теплову карту (нижній рядок ліворуч) для кожної категорії. Перераховуються комбінації піків (середній лівий) чотирьох екстремальних точкових теплових карт і обчислюється геометричний центр складеного обмежувального прямокутника (середній правий). Обмежувальний прямокутник виробляється тоді і тільки тоді, коли її геометричний центр має високу реакцію на центральній тепловій карті (знизу праворуч) [5].

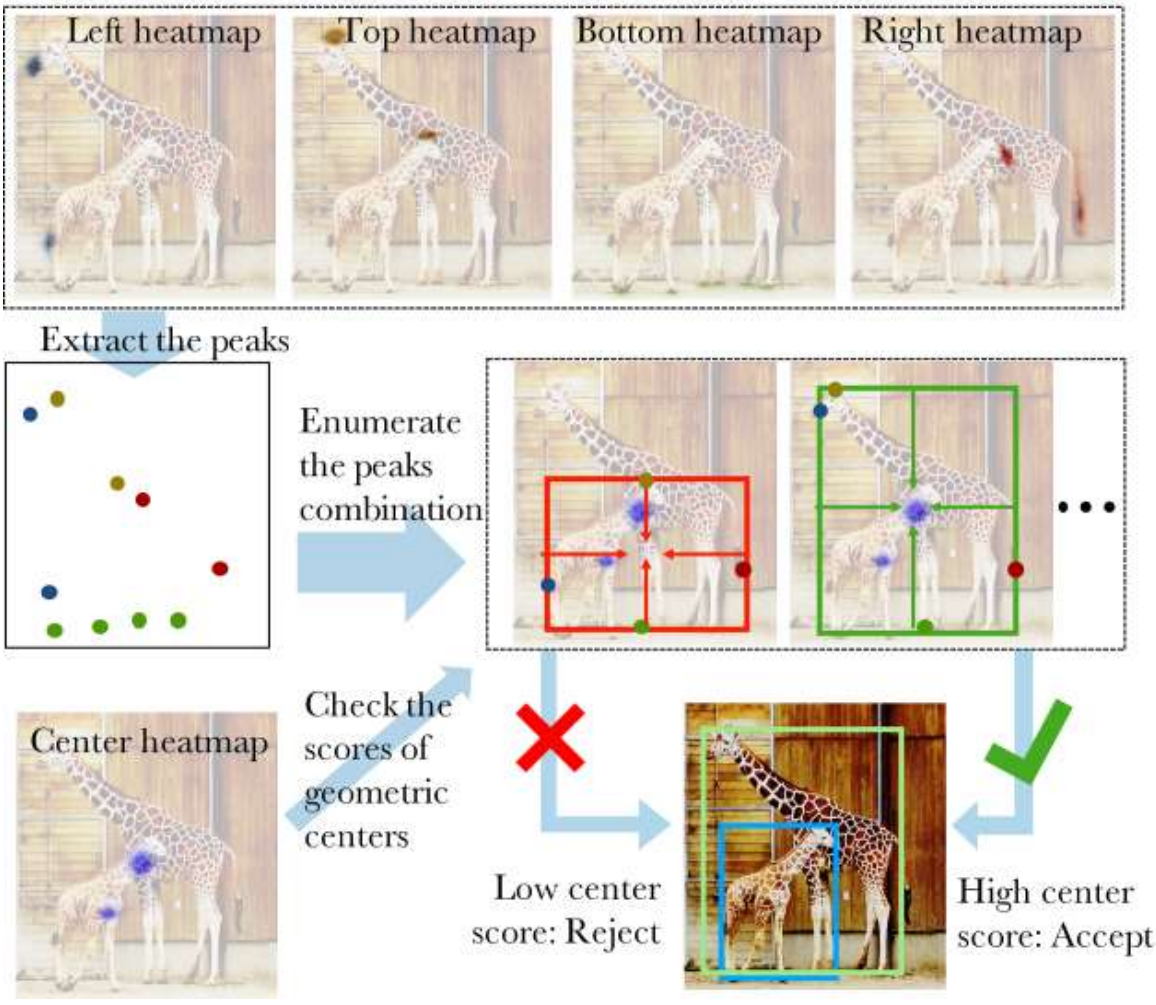


Рисунок 1.4 – Схема роботи згорткової нейронної мережі на основі теплової карти [5].

Для вирішення поставленої задачі необхідно створити інтелектуальну систему розпізнавання образів, яка спроможна виконувати виявлення об’єктів у реальному часі та робити це у web-контексті. На разі неможливо знайти систему, яка може задовольнити ці потреби. Одна з причин, те що браузерам необхідно забезпечувати максимальну безпеку користувачам, тому програмний код який виконується в середині браузера має обмеження пов’язані з виконанням низькорівневих завдань (робота з процесором, відеокартою та інше). Хоч ми і можемо використовувати відеокарту за допомогою посередника Canvas, проте цей інтерфейс не дає змоги використовувати технічне забезпечення на 100%, тому його використання не може гарантувати розпізнавання образів у реальному часі.

Використання серверних технологій може вирішити проблеми з обмеженнями браузерів, проте необхідно розглянути архітектури, які орієнтовані на розпізнавання образів у реальному часі:

- Faster R-CNN;
- YOLO;
- Single Shot MultiBox Detector (SSD).

Faster R-CNN складається з двох модулів. Перший модуль – це глибока повна згорткова мережа, яка пропонує регіони, а другий модуль – це Fast R-CNN детектор, який використовує запропоновані області. Вся система являє собою єдину уніфіковану мережу для виявлення об'єктів (рис. 1.5). Використовуючи нещодавно популярну термінологію нейронних мереж із механізмами «уваги», модуль RPN повідомляє модулю Fast R-CNN, де шукати.

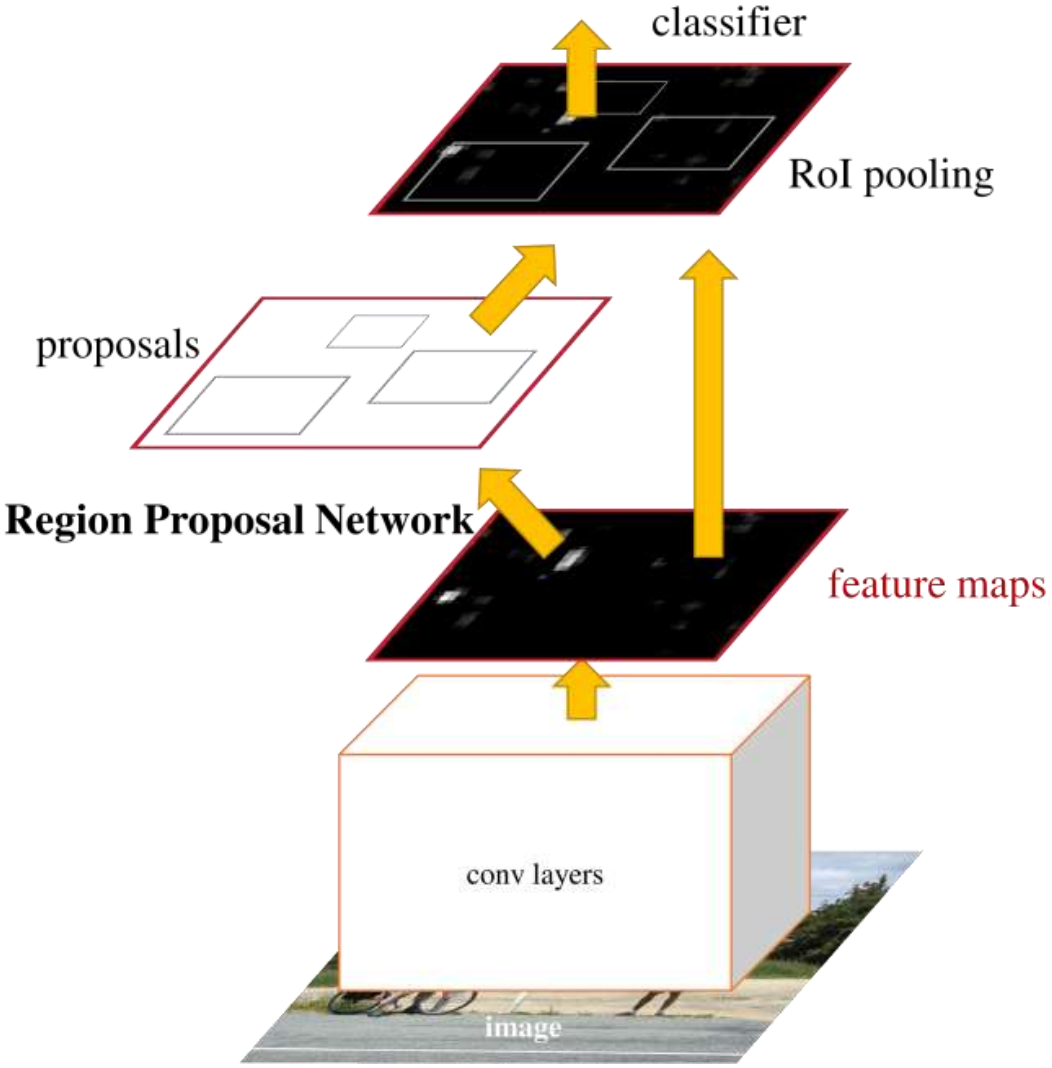


Рисунок 1.5 – Схема мережі Faster R-CNN [6].

Дана мережа має високу точність, від 65.7% до 75.9% mAP в залежності від методу тренування мережі та тренувальної вибірки (табл. 1.1). Пропускна спроможність системи продемонстрована у таблиці 1.2 [6]. Дані результати показують, що дану мережу можна використовувати для точного виявлення об’єктів, проте 5 фреймів за секунду може бути недостатньо.

Таблиця 1.1 – Середня точність Faster R-CNN [6].

Метод	**Тренувальна вибірка**	**mAP (%)**
SS	PASCAL VOC 12	65.7
SS	PASCAL VOC 07 + 12	68.4
RPN+VGG, shared	PASCAL VOC 12	67.0
RPN+VGG, shared	PASCAL VOC 07 + 12	70.4
RPN+VGG, shared	MS COCO + PASCAL VOC 07 + 12	75.9

Таблиця 1.2 – Кількість фреймів за секунду Faster R-CNN [6].

Система	**FPS**
SS + Fast R-CNN	0.5
RPN + Fast R-CNN	5

Підхід SSD заснований на зворотній мережі, що забезпечує пряму передачу даних, яка створює колекцію обмежувальних прямокутників фіксованого розміру та оцінки присутності екземплярів класу об’єктів у цих прямокутниках з подальшим етапом не максимальної суспензії для остаточного виявлення. Ранні мережеві рівні базуються на стандартній архітектурі, яка використовується для класифікації зображень високої якості (усічена перед будь-якими класифікаційними шарами), яку можна назвати базовою мережею. Потім додано в мережу допоміжну структуру для створення детекторів із такими ключовими характеристиками:

1. багатомасштабні карти ознак для виявлення – згорткові шари об’єктів усіченої базової мережі. Ці шари поступово зменшуються в

розмірі і дозволяють передбачати виявлення об’єктів в різних масштабах;

2. Згорткові прогнозувальники для розпізнавання – кожен доданий функціональний шар (або, за бажанням, існуючий рівень функцій з базової мережі) може створювати фіксований набір прогнозів виявлення за допомогою набору згорткових фільтрів;
3. Обмежувальні прямокутники та пропорції за замовчуванням – набір обмежувальних прямокутників за замовчуванням з кожною коміркою карти ознак для декількох карт ознак у верхній частині мережі. Обмежувальні прямокутники за замовчуванням викладають карту ознак згортково, так що положення кожного обмежувального прямокутника відносно відповідної комірки є фіксованим. У кожній комірці карти ознак ми передбачаємо зміщення щодо стандартних обмежувальних прямокутників за замовчуванням у комірці, а також оцінки за класом, які вказують на наявність екземпляра класу в кожному з цих обмежувальних прямокутників.

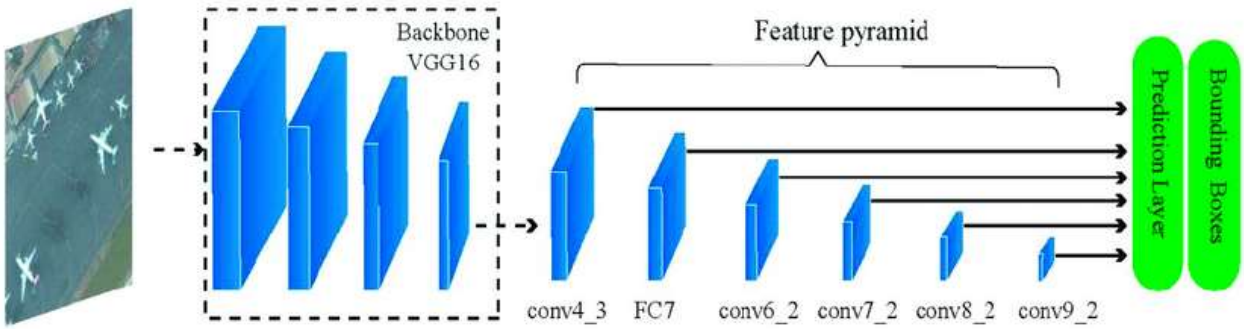


Рисунок 1.6 – Схема роботи моделі SSD [7].

Результати роботи даної архітектури для зображень розміном 300x300 та 512x512 продемонстровано на таблиці 1.3 [7].

Таблиця 1.3 – Результати роботи моделі SSD [7].

Метод	**mAP (%)**	**FPS**
SSD300	74.3	59
SSD512	76.8	22

YOLO – єдина нейронна мережа, яка використовується безпосередньо до повного зображення. Ця мережа ділить зображення на регіони і передбачає обмежувальні прямокутники та ймовірності для кожного регіону. Ці обмежувальні прямокутники зважуються за прогнозованими ймовірностями.

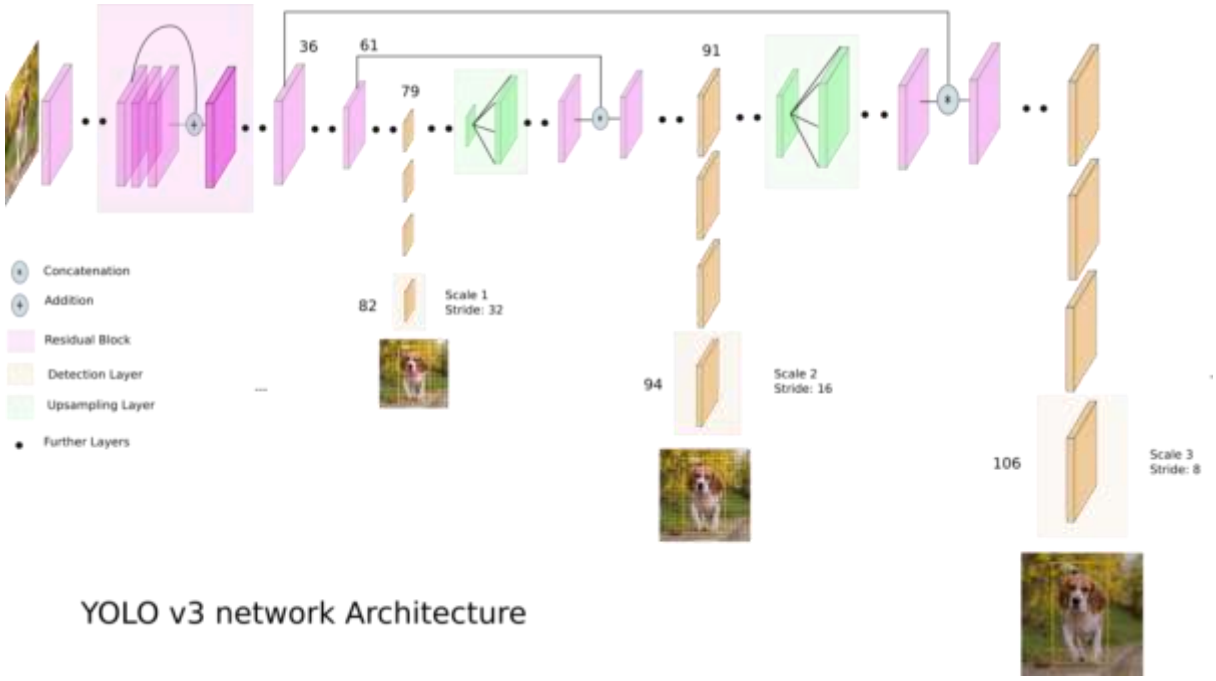


Рисунок 1.7 – Архітектура моделі YOLO v3 [8].

Ця модель має кілька переваг перед системами на основі класифікаторів. Вона розглядає ціле зображення під час тестування, тому його прогнози визначаються глобальним контекстом зображення. Вона також робить прогнози за допомогою єдиного використання мережі, на відміну від таких систем, як R-CNN, які виконуюсь прогнозування тисячі разів для одного зображення. Це робить модель YOLO надзвичайно швидкою, більш ніж в 1000 разів швидшою, ніж R-CNN, і в 100 разів, ніж Fast R-CNN. Результати тестування можна побачити в таблиці 1.4 [8].

Таблиця 1.4 – Результати тестування моделі YOLO [8].

Модель	**mAP (%)**	**FPS**
YOLOv3-320	51.5	45
YOLOv3-416	55.3	35
YOLOv3-608	57.9	20
YOLOv3-tiny	33.1	220

1.2 Актуальність розробки

Актуальністю розробки інтелектуальної системи розпізнавання образів у web-контексті в режимі реального часу є:

1. необхідність статистичного аналізу нетекстової інформації веб-ресурсів;
2. створення системи батьківського контролю, яка цензурує контент браузерів у реальному часі, адже більшість сучасних систем тільки блокують доступ до сумнівних веб-ресурсів та не можуть цензурувати контент фільмів і серіалів, які користувачі зазвичай переглядають в Інтернеті.

Інтелектуальна система розпізнавання образів у web-контексті дає можливість вирішити ці проблеми, надати більших можливостей для аналізу веб-ресурсів, створити безпечніші умови користуванням Інтернетом неповнолітнім.

1.4 Постановка задачі

За результатами проведеного аналітичного огляду задача з розробки інтелектуальної системи розпізнавання образів у веб-контексті в режимі

реального часу є актуальною. Для її вирішення необхідно виконати такі завдання:

1. обрати модель нейронної мережі, її імплементацію, методи та технології програмної реалізації;
2. розробити та дослідити інтелектуальну систему розпізнавання образів у Web-контексті в режимі реального часу;
3. провести тестування розробленої системи.

Створений продукт дозволить розпізнавати та локалізувати образи в режимі реального часу у web-контексті.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Вибір моделі нейронної мережі та її імплементації

Нейронна мережа YOLOv4 та фреймворк глибинного навчання Darknet (C/C++/CUDA) кращі у швидкості FPS та точності AP50:95 та AP50 для навчальної вибірки Microsoft COCO, ніж наступні фреймворки та нейронні мережі: Google TensorFlow EfficientDet, FaceBook Detectron RetinaNet / MaskRCNN, PyTorch Yolov3-ASFF та багато інших... YOLOv4 досягає точності 43,5% AP / 65,7% AP50 відповідно до тесту Microsoft COCO на швидкості 62 FPS з TitanV або 34 FPS з RTX 2070. На відміну від інших сучасних детекторів, YOLOv4 можна навчити всім, хто використовує ігровий графічний адаптер nVidia з 8–16 ГБ VRAM. Зараз не тільки великі компанії можуть навчити нейронну мережу на десятках графічних процесорів, використовуючи великі розміри міні-пакетів для досягнення більш високої точності. YOLOv4 надає контроль над штучним інтелектом звичайним користувачам, оскільки ця модель не вимагає великих міні-пакетів [9].

YOLOV4 є оптимальним для завдань виявлення об’єктів у реальному часі, оскільки мережа лежить на кривій оптимальності Парето на графіку AP (точність) / FPS (швидкість), яку можна побачити на рисунку 2.1.

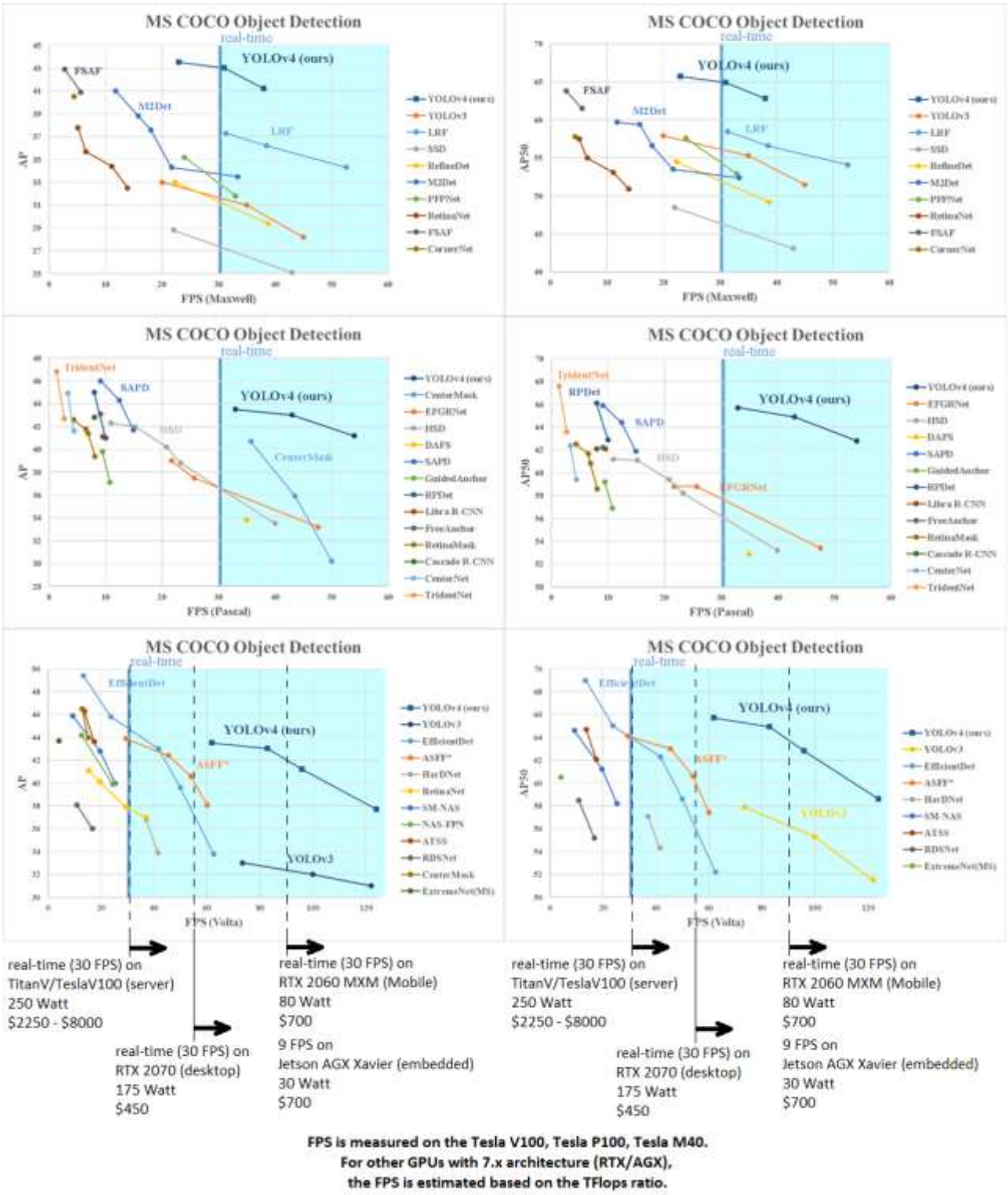


Рисунок 2.1 – Діаграми точності (AP) та швидкості (FPS) декількох нейронних мереж для виявлення об'єктів, виміряні на графічних адаптерах графічного процесора TitanV / TeslaV100, TitanXP / TeslaP100, TitanX / TeslaM40 для двох основних показників точності: AP50:95 та AP50 [9].

Найбільш практичні завдання мають мінімально необхідні вимоги до детекторів - це мінімально допустимі значення точності та швидкості. Зазвичай мінімально допустиме значення швидкості становить від 30 кадрів в секунду або більше для систем реального часу.

Як видно з діаграм, у системах реального часу з FPS 30 або більше:

- для YOLOv4–608 необхідно використовувати RTX 2070 за 450 доларів США (34 кадрів в секунду) з точністю 43,5% AP / 65,7% AP50;
- для EfficientDet-D2 необхідно використовувати TitanV, $ 2250 (42 FPS), з точністю 43,0% AP / 62,3% AP50;
- для EfficientDet-D0 необхідно використовувати RTX 2070 із ціною 450 доларів США (34 FPS) з точністю 33,8% AP / 52,2% AP50.

YOLOv4 вимагає в 5 разів дешевшого обладнання, але при цьому є більш точним, ніж EfficientDet-D2 (Google-TensorFlow). Можна використовувати EfficientDet-D0 (Google-TensorFlow) на дешевому обладнанні, але тоді точність буде на 10% нижчою. Крім того, деякі промислові системи мають обмеження щодо випромінювання тепла або застосування пасивної системи охолодження - у цьому випадку ви не можете використовувати TitanV, навіть якщо він доступний.

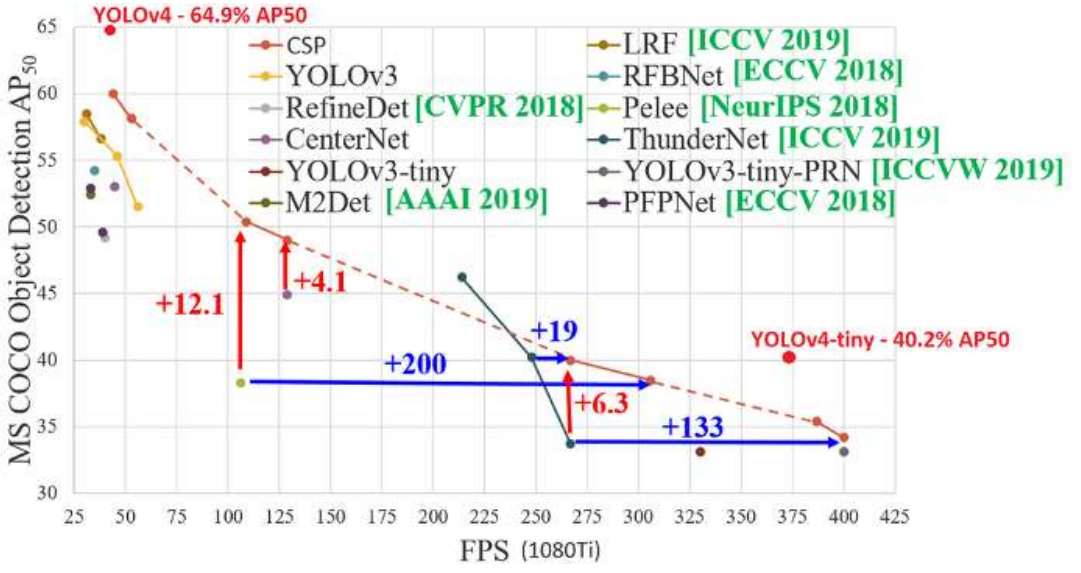
При використанні YOLOv4 (416x416) на графічному процесорі RTX 2080 Ti за допомогою TensorRT + tkDNN ми досягаємо вдвічі більшої швидкості, а при використанні batch = 4 швидкість навіть у 3–4 рази вище (табл. 2.1).

Таблиця 2.1 – Швидкість (FPS) нейронної мережі YOLOv4 320x320–608x608 на Darknet або tkDNN + TensorRT або OpenCV-dnn [9].

Size	Darknet FPS (avg)	tkDNN TensorRT FP32 FPS	tkDNN TensorRT FP16 FPS	OpenCV FP16 FPS	tkDNN TensorRT FP16 batch=4 FPS	OpenCV FP16 batch=4 FPS	tkDNN Speedup
320	100	116	**202**	171	**423**	384	**4.2x**
416	82	103	**162**	146	**284**	260	**3.5x**
512	69	91	**134**	125	**206**	190	**2.9x**
608	53	62	**103**	100	**150**	133	**2.8x**

nVidia GPU GeForce RTX 2080 Ti

Також була випущена крихітна модель YOLOv4 з надзвичайно високою швидкістю 370 FPS на GPU 1080ti або 16 FPS на Jetson Nano (max_N, 416x416, batch = 1, Darknet-framework). До 1000 кадрів в секунду за допомогою OpenCV / tkDNN-TensorRT (рис. 2.2).



- **1770 FPS** - on GPU RTX 2080Ti - (416x416, fp16, batch=4) tkDNN/TensorRT

- **1353 FPS** - on GPU RTX 2080Ti - (416x416, fp16, batch=4) OpenCV 4.4.0

- **39 FPS** - 25ms latency - on Jetson Nano - (416x416, fp16, batch=1) tkDNN/TensorRT

- **290 FPS** - 3.5ms latency - on Jetson AGX - (416x416, fp16, batch=1) tkDNN/TensorRT

- **20 FPS** on CPU ARM Kirin 990 Smartphone Huawei P40 (416x416, GPU-disabled, batch=1) Tencent/NCNN

- **42 FPS** - on CPU i7 7700HQ Laptop - (416x416, fp16, batch=1) OpenCV-dnn (OpenVINO backend)

Рисунок 2.2 – Порівняння yolov4-tiny з іншими моделями [9].

Іноді швидкість (FPS) деяких нейронних мереж вказується при використанні великого розміру партії або при тестуванні за допомогою спеціалізованого програмного забезпечення (TensorRT), яке оптимізує мережу і показує збільшене значення FPS. Порівняння деяких мереж на TRT з іншими мережами без TRT не є справедливим. Також використання великого розміру batch збільшує частоту кадрів в секунду, але також збільшує затримку (а не зменшує її) порівняно з batch = 1. Якщо мережа з batch = 1 показує 40 кадрів в секунду, а мережа з batch = 32 показує 60 кадрів в секунду, тоді затримка становитиме 25 мс для batch = 1, і ~ 500 мс для batch = 32, оскільки лише секунд буде оброблятися ~ 2 batch (32 зображення поштучно). З цієї причини використання batch = 32 неприйнятно у багатьох промислових системах. Тому ми показуємо результати лише з batch = 1 та без використання TensorRT на графіках порівняння.

Щоб навчати EfficientDet з розміром mini-batch = 128 необхідно використовувати декілька графічних процесора Tesla V100 32 ГБ, тоді як YOLOv4 навчається лише на одному графічному процесорі Tesla V100 32 ГБ з mini-batch = 8, і ви можете використовувати звичайну ігрову відеокарту GPU з 8–16 ГБ VRAM [9]. Як бачимо найоптимальніша модель нейронної мережі є YOLOv4.

Модель YOLOv4

Найцікавішою частиною статті про розробку YOLOv4 є те, які нові технології були обрані, модифіковані та інтегровані в неї. Вони роблять детектор більш придатним для навчання на одному графічному процесорі. Розглянемо деякі з них, щоб краще розуміти принцип роботи моделі.

Dense Block та DenseNet

Для підвищення точності можна розробити більш глибоку мережу, щоб розширити сприйнятливе поле та збільшити складність моделі. А щоб

полегшити складність тренувань, можна застосувати пропускні з'єднання. Цю концепцію можна розширити за допомогою дуже взаємопов’язаних шарів.

Dense Block (щільний блок) містить кілька шарів згортки, кожен шар H_i складається із пакетної нормалізації, ReLU, з подальшою згорткою. Замість того, щоб використовувати вихідні дані лише останнього шару, H_i приймає як вихідні дані всі попередні шари, а також оригінал. тобто x_0, x_1,… та x_{i-1}. Кожен H_i нижче виводить чотири карти ознак. Отже, на кожному шарі кількість карт ознак збільшується на чотири - швидкість зростання [18].

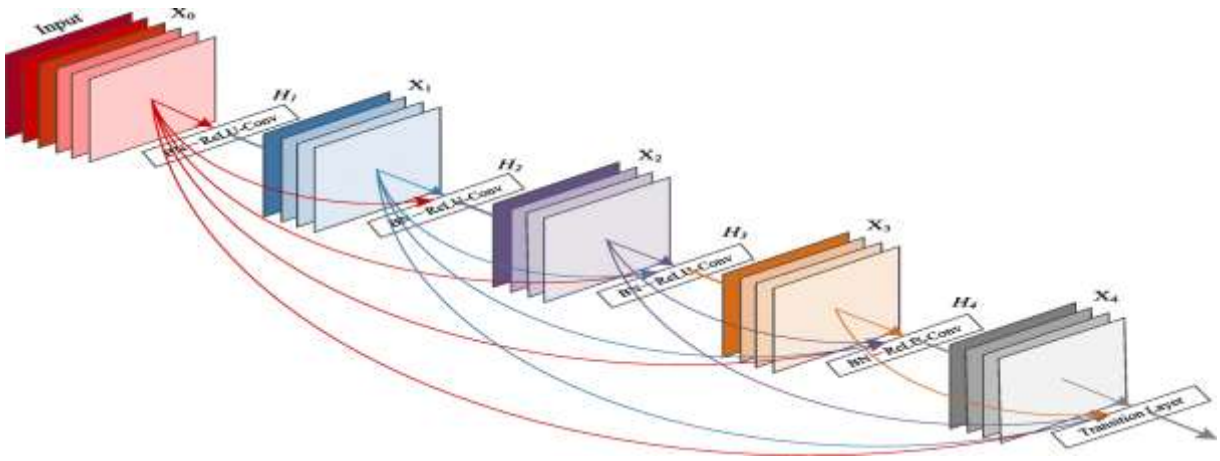


Рисунок 2.3 – П'ятишаровий щільний блок зі швидкістю росту k = 4 [18].

Тоді DenseNet може бути сформований шляхом складання декількох щільних блоків з перехідним шаром між тим, який складається із згортки та об'єднання.

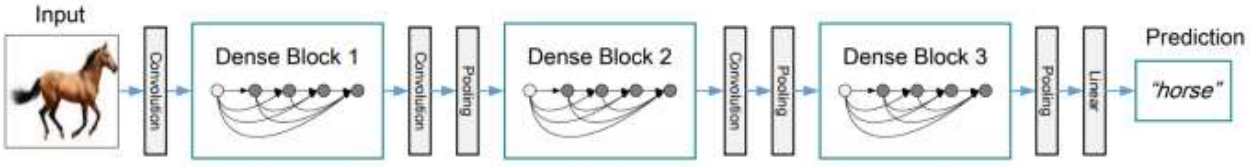


Рисунок 2.4 – Глибока DenseNet мережа з трьома щільними блоками [18].

Шари між двома сусідніми блоками називаються перехідними шарами та змінюють розміри карти ознак за допомогою згортки та об'єднання [18].

Перехресні часткові зв’язки (Cross-Stage-Partial-connections, CSP)

CSPNet розділяє вхідні карти ознак щільного блоку на дві частини. Перша частина x_0' обходить щільний блок і стає частиною вхідних даних для наступного перехідного шару. Друга частина x_0'' піде, думаючи про щільний блок, як показано на рисунку 2.5 [19].

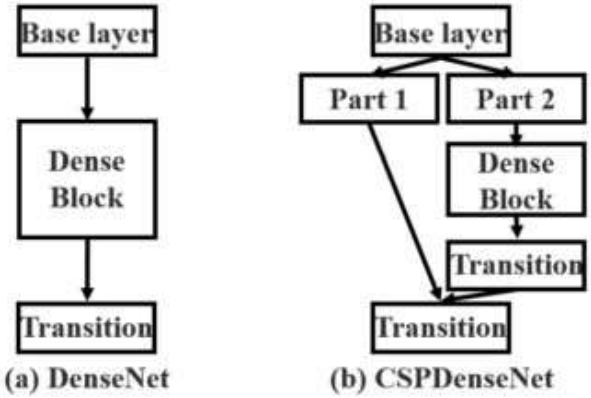


Рисунок 2.5 – Різні види функціональних стратегій злиття. (a) одноконтурний DenseNet, (b) запропонований CSPDenseNet [19].

Ця нова конструкція зменшує обчислювальну складність, розділяючи вхідні дані на дві частини - лише одна проходить через щільний блок.

CSPDarknet53

YOLOv4 використовує CSP-з'єднання Darknet-53, як магістраль для вилучення функцій (табл. 2.2).

Таблиця 2.2 – Darknet53 [20].

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

Модель CSPDarknet53 має вищу точність розпізнавання образів у порівнянні з конструкціями на основі ResNet, навіть якщо вони мають кращі показники класифікації [20]. Але точність класифікації CSPDarknet53 можна покращити за допомогою технології Mish та інших методів. Таким чином, остаточним вибором для YOLOv4 є CSPDarknet53.

Просторовий шар об'єднання піраміди (spatial pyramid pooling layer)

Spatial pyramid pooling layer (SPP) застосовує дещо іншу стратегію виявлення об'єктів різного масштабу. Він замінює останній шар об'єднання (після останнього згорткового шару) просторовим шаром об'єднання піраміди. Карти ознак просторово розділені на бункери m × m, де m, скажімо, дорівнює 1,

2 та 4 відповідно. Потім для кожного каналу застосовується максимальний пул для кожного каналу. Це формує подання фіксованої довжини, яке можна додатково проаналізувати за допомогою FC-шарів [21].

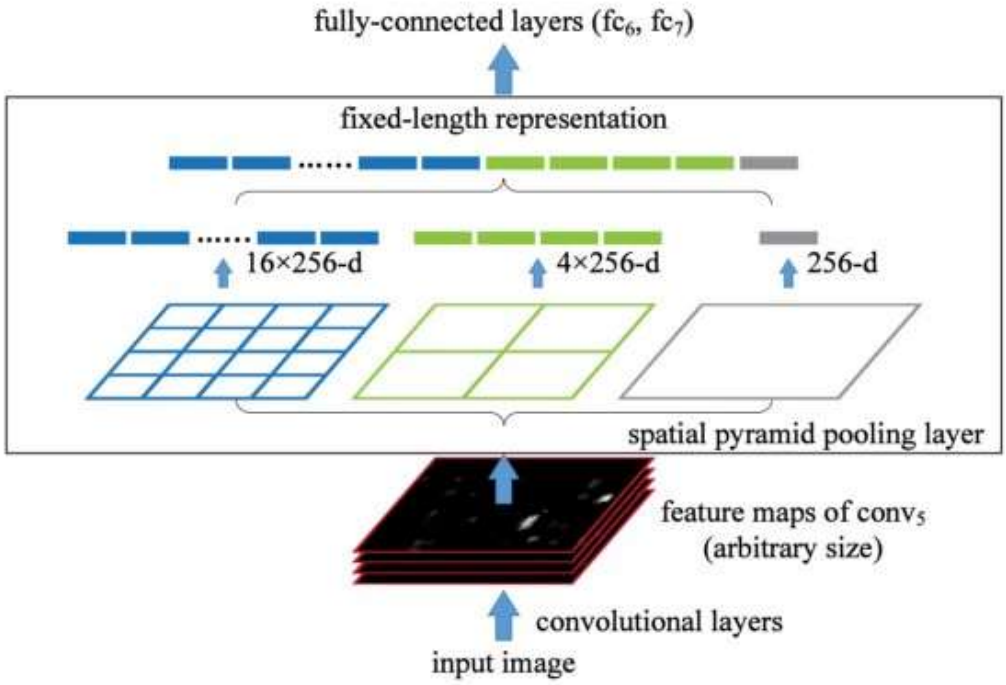


Рисунок 2.6 – Мережева структура з просторовим шаром об’єднання пірамід [21].

Багато моделей на базі CNN, що містять FC-шари, і тому приймають лише вхідні зображення певних розмірів. На відміну від них, SPP приймає зображення різного розміру. Тим не менше, існують такі технології, як мережі з повною згорткою (FCN), які не містять FC-шарів і приймають зображення різних розмірів. Цей тип дизайну особливо корисний для сегментації зображень, просторова інформація яких є важливою. Тому для YOLO перетворення 2-D карт ознак у 1-D вектор фіксованого розміру не є обов'язковим.

У моделі YOLO SPP модифікований для збереження вихідного просторового виміру. Максимальний пул застосовується до ковзного ядра

розміром, скажімо, 1×1, 5×5, 9×9, 13×13. Просторовий вимір зберігається. Карти ознак різних розмірів ядра потім об'єднуються разом як вихідні дані [22].

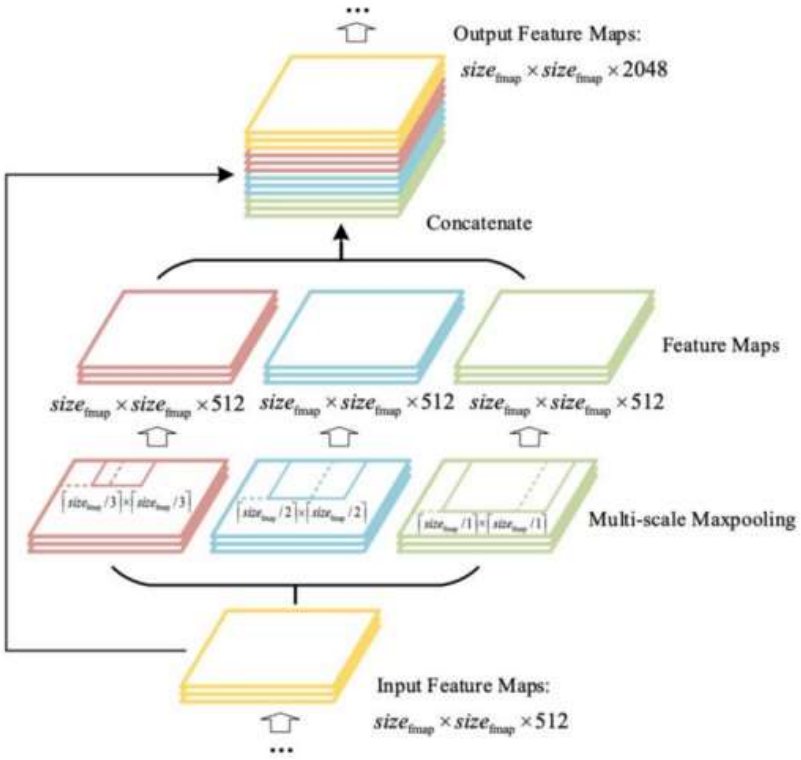


Рисунок 2.7 – Вдосконалене об'єднання просторових пірамід (SPP) [22].

Рисунок 2.8 демонструє, як SPP інтегровано в YOLO [22].

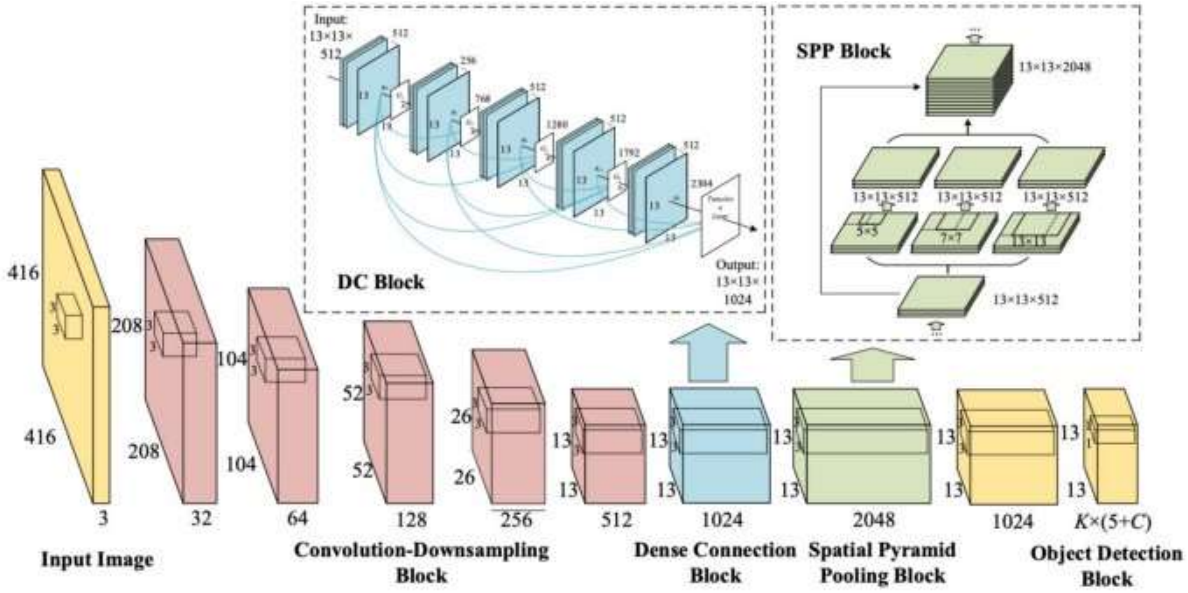


Рисунок 2.8 – Модель DC-SPP-YOLO [22].

Модуль просторової уваги (Spatial Attention Module, SAM)

«Увага» широко застосовується в дизайні DL. У SAM максимальний пул і середній пул застосовуються окремо для введення карт функцій для створення двох наборів карт функцій. Результати подаються в шар згортки з подальшою сигмовидною функцією для створення «просторової уваги» [23].

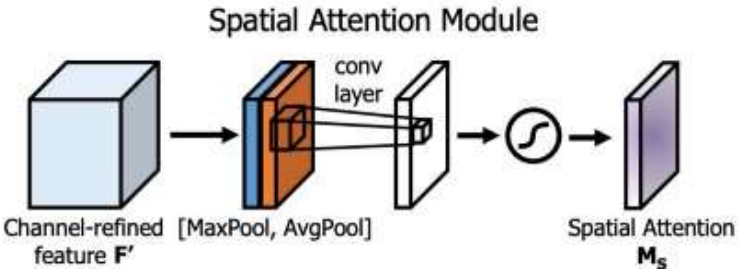


Рисунок 2.9 – Модуль просторової уваги [23].

Ця просторова маска уваги застосовується до вхідної функції для виведення уточнених карт ознак.

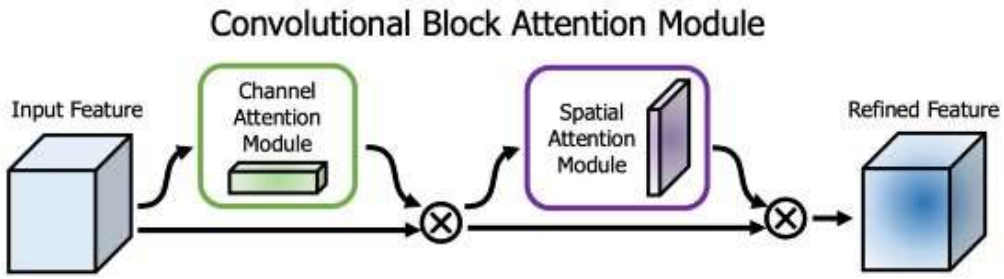


Рисунок 2.10 – Огляд модуля уваги до згорткового блоку [23].

У YOLOv4 модифікований SAM використовується без застосування максимального та середнього об'єднання (рис. 2.11) [24].

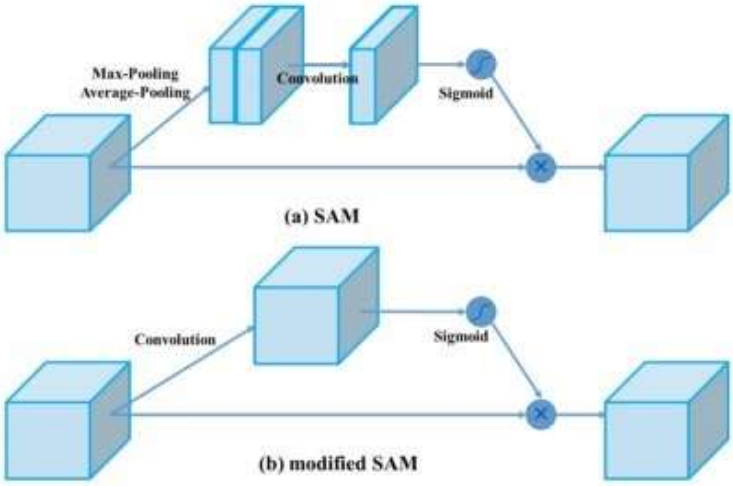


Рисунок 2.11 – Модифікований SAM у YOLOv4 [24].

Функція активації Mish

Припустимо, функція активації має форму, як на рисунку 2.12, з різними кандидатами на функції (наприклад, функцією косинуса) для одинарних або двійкових операторів. Ми можемо робити випадкові здогади, вибираючи ці функції та оцінюючи відповідні характеристики моделі на основі різних завдань (наприклад, класифікації) та наборів даних. Нарешті, ми можемо вибрати функцію активації, яка має найкращі результати.

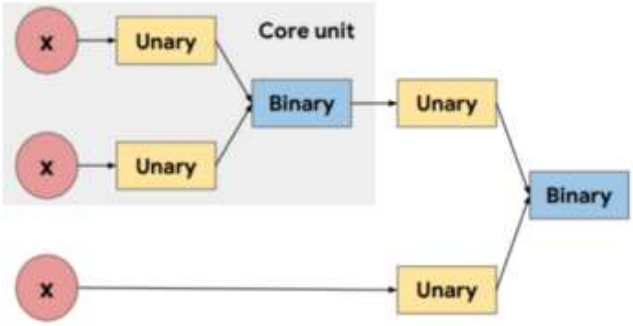


Рисунок 2.12 – Приклад структури функції активації [25].

Застосовуючи навчання з підсиленням, ми можемо ефективніше шукати простір рішень.

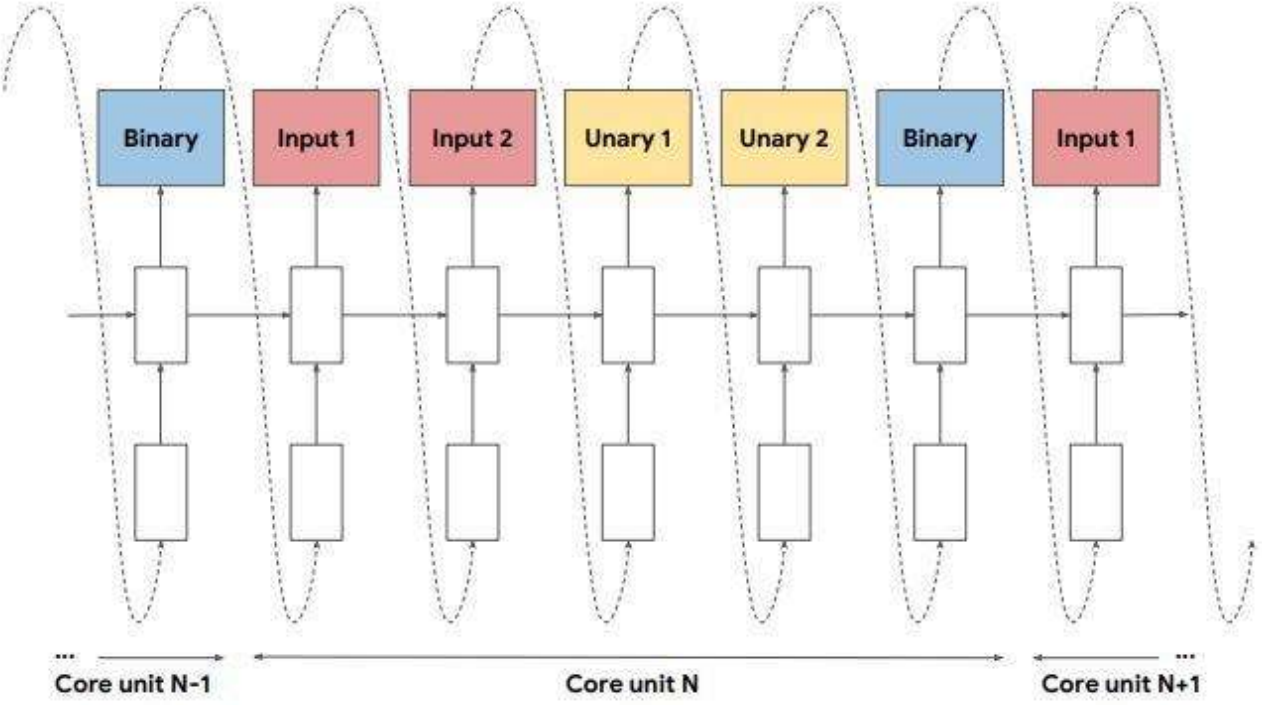


Рисунок 2.13 – Контролер RNN, який використовується для пошуку у великих просторах [25].

На кожному кроці він передбачає окремий компонент функції активації. Прогноз подається назад як вхід для наступного кроку часу в авторегресивному режимі. Контролер продовжує прогнозувати, доки не буде обрано кожен компонент функції активації. Контролер проходить навчання з підсиленням.

Використовуючи цей метод з подальшими експериментами, нова функція активації (рис. 2.14), звана Swish, показує кращу продуктивність, ніж ReLU та багато інших функцій активації [25].

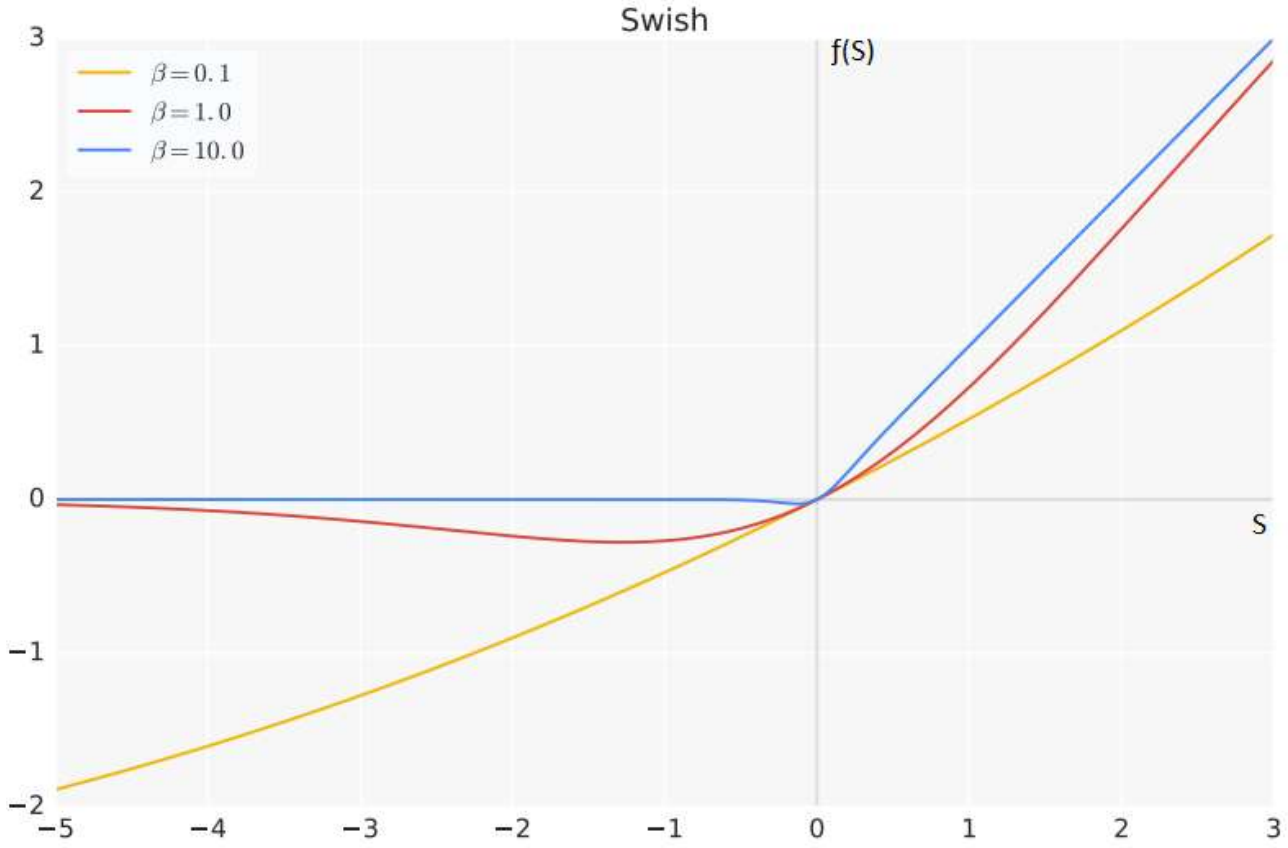


Рисунок 2.14 – Графік функції активації Swish f(S) з різними значеннями β, де S – вихідне значення суматора [25].

Mish - ще одна функція активації, близька до ReLU та Swish. Міш може перевершити їх у багатьох глибоких мережах у різних наборах даних [26]. Використання функції активації Mish для CSPDarknet53 і детектора збільшує обидві точності AP50 та AP50:95 в YOLOv4.

$$f(x) = x \cdot tanh(\varsigma(x))$$

де, $\varsigma(x) = \ln(1 + e^x)$функція активації softplus

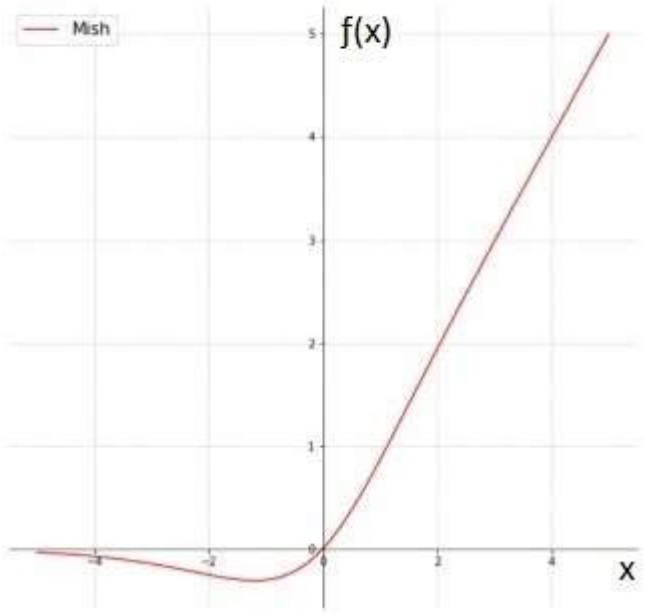


Рисунок 2.15 – Графік функції активації Mish [26].

2.2 Вибір засобів розробки програмно-апаратної частини системи

Для вирішення поставленої задачі необхідно створити програмне забезпечення за допомогою якого буде реалізована комунікація між імплементацією моделі нейронної мережі та клієнтською частиною системи. Потрібно використати серверні технології на базі NodeJS або Python з використанням протоколу WebSocket для передачі даних у реальному часі. Серверна технологія повинна не тільки приймати дані з клієнтської сторони, але й ефективно взаємодіяти з імплементацією.

WebSocket – це комп'ютерний протокол зв'язку, що забезпечує повнодуплексні канали зв'язку через єдине TCP-з'єднання [10][11]. Протокол WebSocket був стандартизований IETF як RFC 6455 в 2011 році, а API WebSocket в Web IDL стандартизований W3C.

WebSocket відрізняється від HTTP. Обидва протоколи розташовані на п’ятому рівні в моделі OSI і залежать від TCP на четвертому рівні. Хоча вони

різні, RFC 6455 зазначає, що WebSocket призначений для роботи через порти HTTP 443 і 80, а також для підтримки проксі-серверів HTTP і посередників, тим самим роблячи його сумісним з протоколом HTTP. Для досягнення сумісності WebSocket використовує оновлений заголовок HTTP для переходу з протоколу HTTP на протокол WebSocket [12].

Протокол WebSocket забезпечує взаємодію між веб-браузером (або іншим клієнтським додатком) та веб-сервером із меншими накладними витратами, ніж напівдуплексні альтернативи, такі як опитування HTTP, полегшуючи передачу даних у реальному часі від і до сервера. Це стає можливим завдяки забезпеченню стандартизованого способу відправки сервером вмісту клієнту без попереднього запиту клієнта та дозволяючи передавати повідомлення вперед-назад, залишаючи з'єднання відкритим. Таким чином, між клієнтом та сервером може відбуватися тривала двостороння розмова. Зв'язок зазвичай здійснюється через номер TCP-порту 443 (або 80 у випадку незахищених з'єднань).

Node.js – це міжплатформене середовище виконання з відкритим вихідним кодом для JavaScript, яке виконує код поза веб-браузером. Node.js дозволяє розробникам використовувати JavaScript для написання інструментів та запуску сценаріїв на стороні сервера для створення динамічного вмісту веб-сторінки до того, як сторінка буде відправлена у веб-браузер користувача. Отже, Node.js об'єднує розробку веб-додатків навколо однієї мови програмування, а не різних мов для серверних та клієнтських сценаріїв.

Node.js має модуль child_process, який надає можливість породжувати підпроцеси. Ця можливість насамперед надається функцією child_process.spawn(). Створений підпроцес за замовчуванням має канали для комунікації stdin, stdout та stderr, які встановлюються між батьківським процесом Node.js та породженим підпроцесом. Ці канали мають обмежену (і певну платформу) пропускну здатність. Якщо підпроцес пише в stdout, що перевищує ці обмеження, вихідні дані не відобразяться, підпроцес блокує

очікування, поки канал буфера не прийме більше даних. Це ідентично поведінці виводу даних в консолі. Також слід зазначити, що метод child_process.spawn() створює дочірній процес асинхронно, не блокуючи цикл подій Node.js [13]. Це дає нам можливість просто використовувати бінарну імплементацію моделі нейронної мережі.

Python – інтерпретована мова програмування високого рівня та загального призначення. Створена Гідо ван Россумом і вперше випущена в 1991 році, філософія дизайну Python підкреслює читабельність коду завдяки своєму помітному використанню значних пробілів. Його мовні конструкції та об’єктно-орієнтований підхід спрямовані на те, щоб допомогти програмістам писати чіткий логічний код для малих та великих проектів [14]. Python динамічно типізований та має збирач сміття. Він підтримує кілька парадигм програмування, включаючи структуроване (зокрема процедурне), об'єктно-орієнтоване та функціональне програмування. Python часто описують як мову, що включає батареї, завдяки своїй повній стандартній бібліотеці.

Ctypes - це бібліотека «чужих» функцій для Python. Вона надає типи даних сумісні з типами мови програмування C і дозволяє викликати функції в бібліотеках DLL або спільних бібліотеках. Його можна використовувати для обгортання цих бібліотек чистим кодом Python. Ви можете завантажити бібліотеки, отримуючи доступ до атрибутів цих об'єктів. cdll завантажує бібліотеки, які експортують функції, використовуючи стандартне правило виклику cdecl, тоді як бібліотеки windll викликають функції, використовуючи правило виклику stdcall. oledll також використовує правило виклику stdcall і передбачає, що функції повертають код помилки Windows HRESULT. Код помилки використовується для автоматичного підключення винятку помилки OSError, коли виклик функції не вдається [15].

Отже можна обрати кожну з цих технологій, як Node.js, так і Python, для виконання поставленого завдання, але для остаточного рішення необхідно буде провести декілька практичних тестів.

2.3 Вибір методу реалізації графічного інтерфейсу користувача

Поставлена задача потребує виконання виявлення об’єктів безпосередньо у веб-браузері, отже нам необхідно ін’єктувати програмний код в нього. Самий простий спосіб створити розширення для веб-браузера. На даний момент існує безліч браузерів, більшість яких підтримують встановлення розширень для них, але кожен має різний програмний інтерфейс для розширень. Основним критерієм для вибору веб-браузера при розробці розширення – є його популярність. Сьогодні найпопулярнішим є Chrome від Google, ним користується близько 62.6% від всіх користувачів веб-браузерами (рис. 2.16) [16].

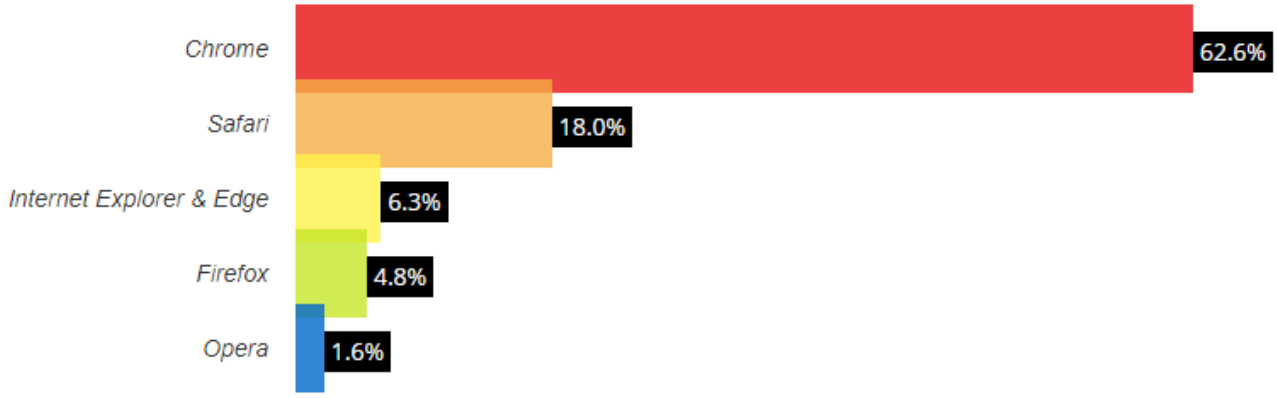


Рисунок 2.16 – Діаграма розподіл ринку веб-браузерів [16].

Розширення – це невеликі програми, які можуть змінювати досвід перегляду веб-сторінок. Вони дозволяють користувачам адаптувати функціональність та поведінку веб-браузера під індивідуальні потреби чи уподобання. Вони побудовані на веб-технологіях, таких як HTML, JavaScript та CSS. Розширення може включати декілька компонентів та набір функціональних можливостей. Інтерфейс користувача може варіюватися від простої піктограми, наприклад, до заміщення цілої сторінки. Файли розширень архівуються в єдиний пакет .crx, який користувач завантажує та встановлює. Це означає, що розширення не залежать від вмісту в Інтернеті, на відміну від звичайних веб-програм [17].

Розширення для веб-браузерів на базі Chromium мають такі основні файли як: маніфест, фоновий сценарій та контент сценарій. У маніфесті вказуються основні параметри розширення та дозвіл на використання тих чи інших програмних інтерфейсів. Фоновий сценарій виконується незалежно від контексту веб-сторінок. Контент сценарій ін’єктується безпосередньо в програмний код веб-сторінки та має можливість взаємодіяти з її DOM моделлю у закритому контексті від інших розширень, що не дає їм конфліктувати між собою. Веб-сторінки у браузері працюють як підпроцеси, проте веб-браузер на базі Chromium надає їм можливість комунікувати за допомогою подій та їх слухачів, що дає нам можливість передавати інформацію між фоновим та контент сценарієм. Це необхідно для реалізації потоку даних у вигляді зображень екрана користувача на серверну сторону системи розпізнавання образів.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Проектування системи

Для розробки системи необхідно виділити основні елементи, їх процеси взаємодії та потоки даних між ними. Основними елементами системи є:

- клієнтська сторона (розширення браузера);
- веб-сервер;
- детектор.

Узагальнено принцип роботи системи можна описати таким чином (рис. 3.1):

1. розширення передає зображення активної вкладки до веб-серверу;
2. веб-сервер передає дані детектору;
3. детектор передає результати розпізнавання веб-серверу;
4. веб-сервер передає результати розпізнавання до розширення;
5. розширення браузера відображує результати розпізнавання в активній вкладці.

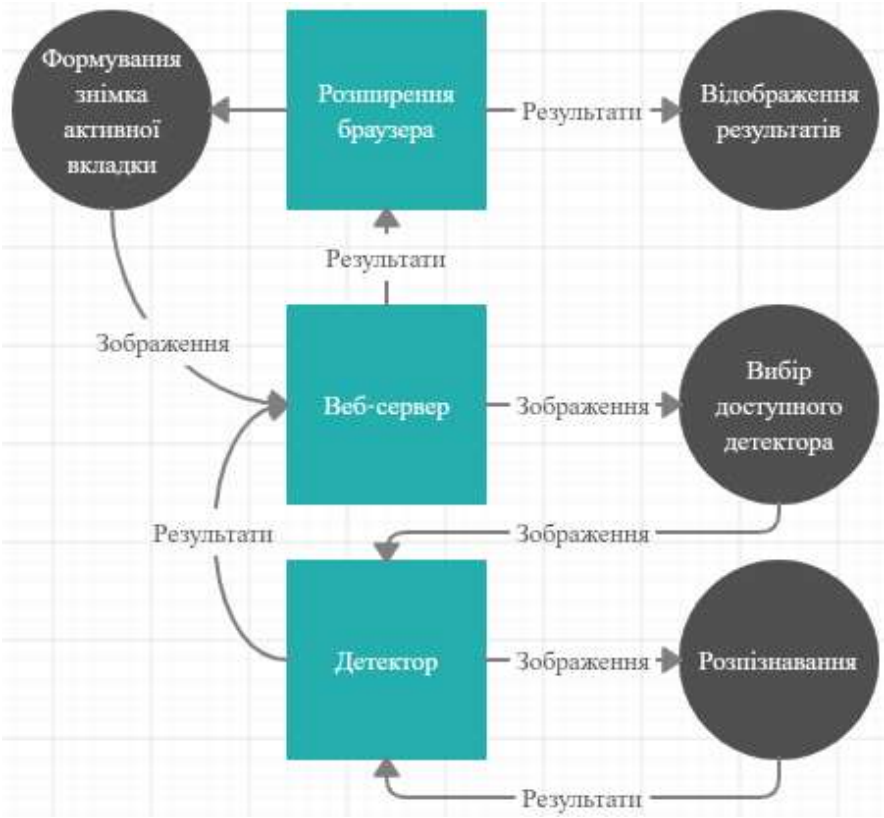


Рисунок 3.1 – Загальна DFD діаграма системи.

Клієнтська сторона має три компонента, які працюють у різному програмному контексті, а саме: popup вікно, фоновий та контент сценарії. Так як вони працюють незалежно один від одного необхідно створити комунікацію між ними, щоб реалізувати поставлені для розширення браузера завдання.

Popup вікно виступає загальним меню для розширення, тому його завданням є включення та виключення розпізнавання образів у браузері. При включенні розпізнавання вікно провокує подію яку прослуховує фоновий сценарій. Якщо подія включення спровокована, тоді фоновий сценарій повинен отримати дані екрану у контент сценарія, отримати дані зображення екрану, відправити зображення до веб-сервера, отримати результати та відправити їх контент сценарію, який повинен їх відобразити. Схематично на рисунку 3.2.

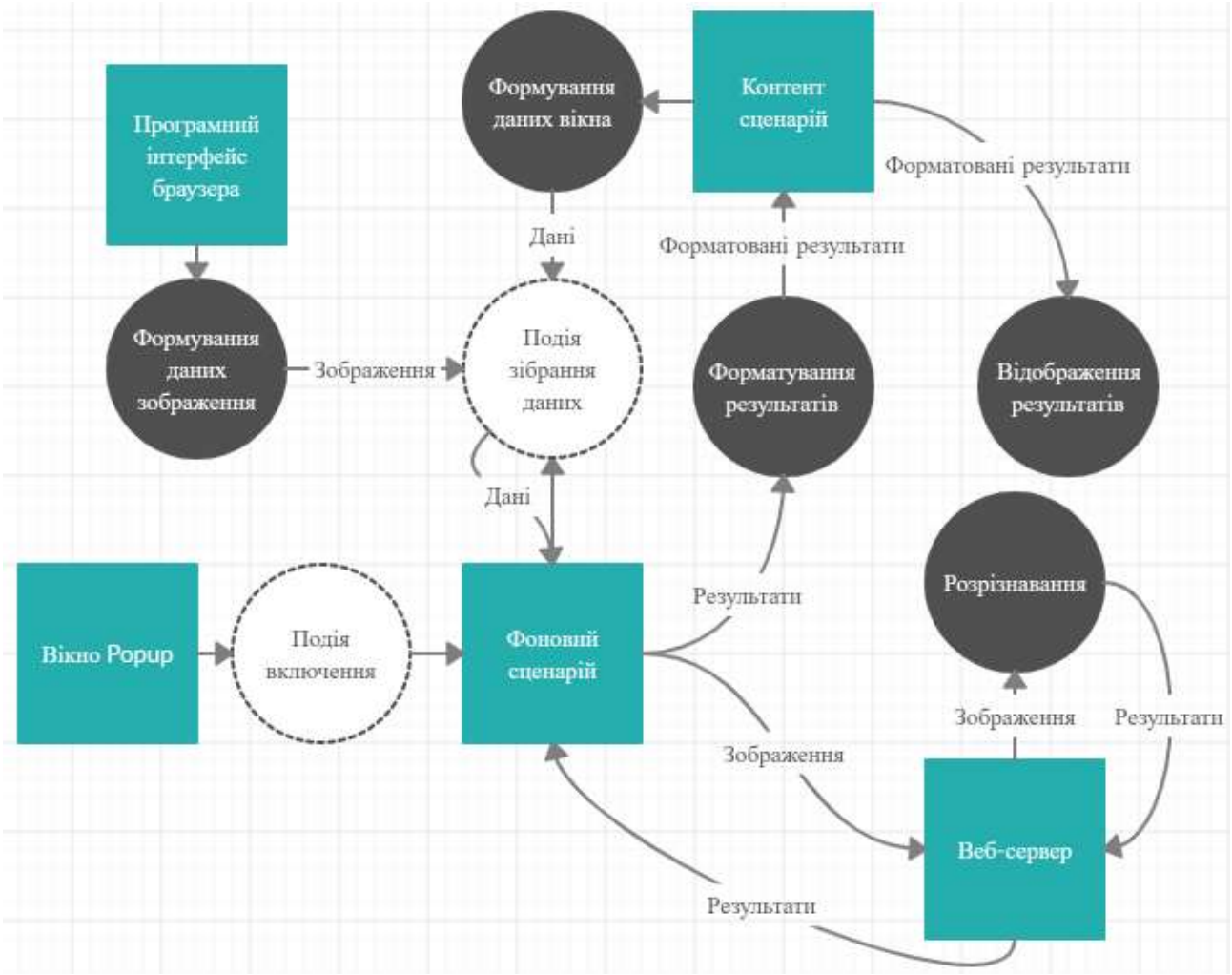


Рисунок 3.2 – DFD діаграма роботи розширення браузера системи.

Хоч елементи клієнтської сторони і працюють у різному контексті, але все ж таки вони ведуть комунікацію через посередника у вигляді браузера, тому можна створити UML діаграму для опису системи більш детально.

На стороні клієнта у нас є декілька елементів – це popup вікно, яке можна описати за допомогою класу popupForm, фоновий сценарій та контент сценарій.

Popup вікно повинно мати функцію рендерингу шаблону меню у вигляді HTML розмітки. Шаблон повинен відображати статус підключення до веб-серверу, роботи детектора, потоку даних на серверну сторону системи. Також меню має мати графічний інтерфейс користувача за допомогою якого можна

взаємодіяти з системою. Отже для реалізації нам необхідно реалізувати клас який буде мати такі особливості:

- метод рендеру шаблону;
- стадії підключення до веб-серверу;
- стадії роботи детектора;
- стадії роботи потоку даних;
- метод підключення та відключення до веб-серверу;
- метод запуску та зупинки детектора;
- метод запуску та зупинки патоку даних;
- метод синхронізації стадій системи.

Метод синхронізації стадій системи необхідний, тому що операції по взаємодії з серверною стороною виконує фоновий сценарій. А методи підключення, відключення, запуску та зупинки провокують події для передачі завдання фоновому сценарію. Тому крім безпосередніх реалізацій цих методів фоновий сценарій повинен зберігати стан системи, а також передавати результати розпізнавання контент сценарію.

Сценарій контенту виконує функцію відображення результатів розпізнавання, а тому повинен мати метод відображення та зберігати результати для оновлення з наступним фреймом.

Серверна сторона має контролер запитів, який має обробники запитів runDetector, stopDetector та frame, також від використовує сервіс провайдер, який реалізує ці методи та зберігає об’єкт детектора та його налаштування.

Клас детектора зберігає загальну інформацію для створення процесу, таку як назва файлу та його розташування, та містить методи взаємодії з ним.

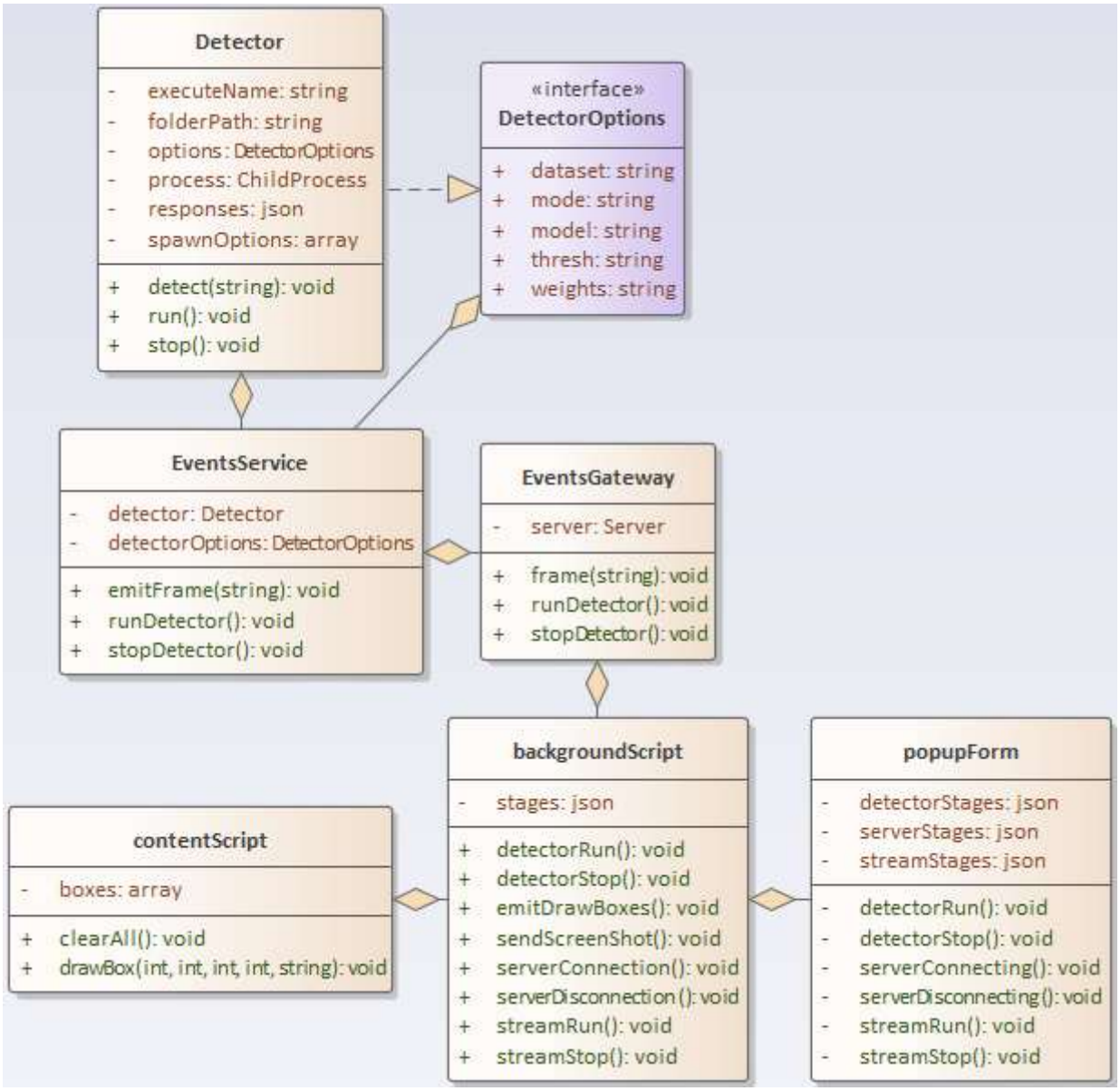


Рисунок 3.3 – UML діаграма системи.

3.2 Програмна реалізація

Для реалізації системи необхідно створити клієнтську частину та серверну, проте спершу слід скомпілювати імплементацію детектору, тому що серверна частина системи буде безпосередньо її використовувати. Ця імплементація використовує додаткове програмне забезпечення для покращення роботи детектора, таке як: CUDA, cuDNN та OpenCV.

Набір інструментів CUDA забезпечує середовище розробки для створення високопродуктивних додатків, прискорених графічним процесором. За

допомогою CUDA можна розробляти, оптимізувати та розгортати програми на вбудованих системах з прискореним графічним процесором, настільних робочих станціях, корпоративних центрах обробки даних та хмарних платформах. Набір інструментів включає прискорені GPU бібліотеки, засоби налагодження та оптимізації, компілятор C / C ++ та бібліотеку середовища виконання для побудови та розгортання додатка на основних архітектурах, включаючи x86, Arm та POWER. Використовуючи вбудовані можливості для розподілу обчислень між конфігураціями декількох графічних процесорів, можна розробляти додатки, що масштабуються від окремих робочих станцій графічного процесора до хмарних установок з тисячами графічних процесорів [27].

Бібліотека глибокої нейронної мережі CUDA (cuDNN) - це прискорена GPU бібліотека примітивів для глибоких нейронних мереж. cuDNN забезпечує високо налаштовані реалізації для стандартних підпрограм, таких як пряма та зворотна згортка, об'єднання, нормалізація та активація шарів [28].

OpenCV – це бібліотека з відкритим кодом, що включає кілька сотень алгоритмів комп'ютерного зору [29].

Імплементація має широкий програмний інтерфейс. Він підтримує роботу з зображеннями, масивами зображень, відео, веб-камерами та MJPEG потоками [30]. Проте робота з зображеннями відбувається через роботу з файловою системою, такий висновок можна зробити згідно з програмним кодом input = fgets(input, 256, stdin); [31]. Отже для передачі зображення необхідно створювати файл, зчитувати його та перезаписувати для наступного розпізнавання. Це створює додаткові витрати у часі, отже необхідно адаптувати програмний інтерфейс до нашої системи. Так як ми використовуємо серверні технології Nodejs з використанням WebSocket, тоді для передачі даних краще всього підійде текстова інформація. Зображення можна передавати у вигляді рядка з base64 кодуванням.

Base64 – це група схем кодування, які представляють двійкові дані у форматі рядка ASCII, перекладаючи їх у представлення radix-64. Схеми кодування Base64 зазвичай використовуються, коли існує потреба в кодуванні двійкових даних, які потрібно зберігати та передавати через носії, призначені для роботи з ASCII. Це гарантує, що дані залишаться незмінними без змін під час транспортування. Base64 зазвичай використовується в ряді додатків, включаючи електронну пошту через MIME та зберігання складних даних у XML [32].

Отже ми будемо передавати дані у вигляді base64. Детектору необхідно зчитати ці дані з stdin використовуючи цикл while поки не буде зчитаний весь рядок base64, адже консоль програми має обмеження в 1024 символів рядка, тому при передачі рядка більше 1024 символів дані будуть розділені на декілька рядків. Детальну реалізацію функції наведено у додатку А.

У інтерфейсі імплементації потрібно змінити метод, який ми будемо використовувати для розпізнавання образів, а саме метод test. Необхідно модифікувати його та методи які він використовує таким чином, щоб матриця зображення, яку використовує модель нейронної мережі, створювалась за допомогою декодування рядка base64. Детальний опис наведено у додатку Б.

Для інтегрування імплементації необхідно створити клас детектора на базі NodeJS з використанням модуля ChildProcess. Клас повинен мати функціонал створення та знищування процесу імплементації, та метод комунікації з ним. Для створення процесу використовується метод spawn модуля ChildProcess. Для комунікації інтерфейси stdin та stdout. Для отримання даних від процесу використовуються слухачі подій, такі як «data», «error» або «exit». Слід зазначити, що слухачі подій працюють асинхронно, тому для передачі зображення до процесу та отриманню результатів розпізнавання потрібно використовувати «обіцянки» (Promise), завдяки яких можна відслідковувати результати виконання розпізнавання за допомогою методів відкликання «вирішено» (resolve) та «відхилено» (reject). Детальну реалізацію наведено у

додатку В. Проте інтерфейс комунікації між основним та додатковим процесом, який працює асинхронно, може створити деяку затримку під час передачі даних, тому можна реалізувати серверну частину системи на основі мови програмування Python з використанням імплементації у вигляді динамічно приєднуваної бібліотеки та Python обгортки для Darknet. Детальний опис наведено у додатку Г.

Основним завданням клієнтської сторони є сбір даних, передача їх до серверної сторони, отримання результатів розпізнавання та відображення їх. Більшістю цих завдань може займатися фоновий сценарій розширення браузера, тому спершу краще реалізувати його.

Фоновий сценарій повинен мати функціонал підключення до серверу, відправлення запитів до нього та вміти обробляти дані результатів розпізнавання. Для підключення до серверної сторони використаємо Socket.io.

Socket.IO - це бібліотека, яка забезпечує взаємодію між браузером та сервером на основі подій в режимі реального часу у двосторонньому режимі [33].

Наш детектор у вигляді результатів розпізнавання відправляє відносні координати та розміри виявлених об’єктів. Тому необхідно створити метод за яким можна конвертувати відносні значення до абсолютних у контексті веб-сторінки. Для цього необхідно отримати дані розміру веб-сторінки, положення фокусу екрану користувача відносно веб-сторінки. Це можна зробити за допомогою теперішніх значень об’єкта «window» у програмному контексті веб-сторінки. Значення innerWidth вказує на теперішню ширину вікна, а innerHeight на теперішню висоту. PageXOffset та pageYOffset вказують на зміщення фокусу екрана користувача по X та Y, відповідно, початку веб-сторінки, який знаходиться у лівому, верхньому куті. За допомогою слухачів подій ми можемо отримати ці дані взаємодіючи з сценарієм контенту, який ін’єктується в програмний код веб-сторінки. Таким чином ми можемо визначити абсолютні координати та розміри виявлених об’єктів.

Для формування зображення екрану користувача програмним інтерфейсом браузера передбачено метод captureVisibleTab, який повертає рядок base64 зображення екрану активної веб-сторінки браузера. Для циклічного відправлення зображення можна використати метод setInterval. Після відправки зображення до серверної сторони ми отримуємо результати, які передаємо сценарію контенту активної веб-сторінки. Детальна реалізація фонового сценарія наведена у додатку Ґ.

Контент сценарій виконує дві функції, перша – передача даних про розміри та зміщення екрану користувача, друга – це відображення результатів. Відображення результатів можна реалізувати за допомогою створення елементів HTML-розмітки. Метод createElement об’єкта document дозволяє програмно створювати HTML елементи. Після створення елементу метод повертає його об’єкт. Завдяки цьому можна змінити дані елемента, вказавши його розташування, розмір та стилістичні ознаки. Детальний опис наведено у додатку Д.

Інтерфейсом користувача є випадаюче вікно при натисканні на значок розширення. Його можна реалізувати за допомогою Vue.js. Необхідно сформувати шаблон вікна на базі HTML-розмітки та створити методи, які забезпечать динамічне відображення стану роботи системи, та кнопки за допомогою який можна стан можна змінювати. Детальна реалізація наведено у додатку Е.

3.3 Тестування розробленої системи

Оскільки головною метою роботи є дослідження можливості використання системи розпізнавання образів у веб-контексті в реальному часі, тому головним показником є кількість кадрів за секунду.

Отже у результаті розробки програмного забезпечення отримано веб-сервер з детектором об’єктів та розширення для браузера, як клієнтську сторону системи. Спершу слід перевірити, яку кількість кадрів може обробляти

детектор. Для цього потрібно перевірити дві попередньо навчені моделі YOLOv4 та YOLOv4-tiny.

Під час першого тесту використовується повна модель YOLOv4 для обробки 250 фреймів відео та в результаті отримано 5.045 середнього значення кількості кадрів за секунду, що означає що на розпізнавання затрачається близько 200 мс. Цей результат є не задовільним, адже для обробки у реальному часі рекомендується близько 30 кадрів за секунду.

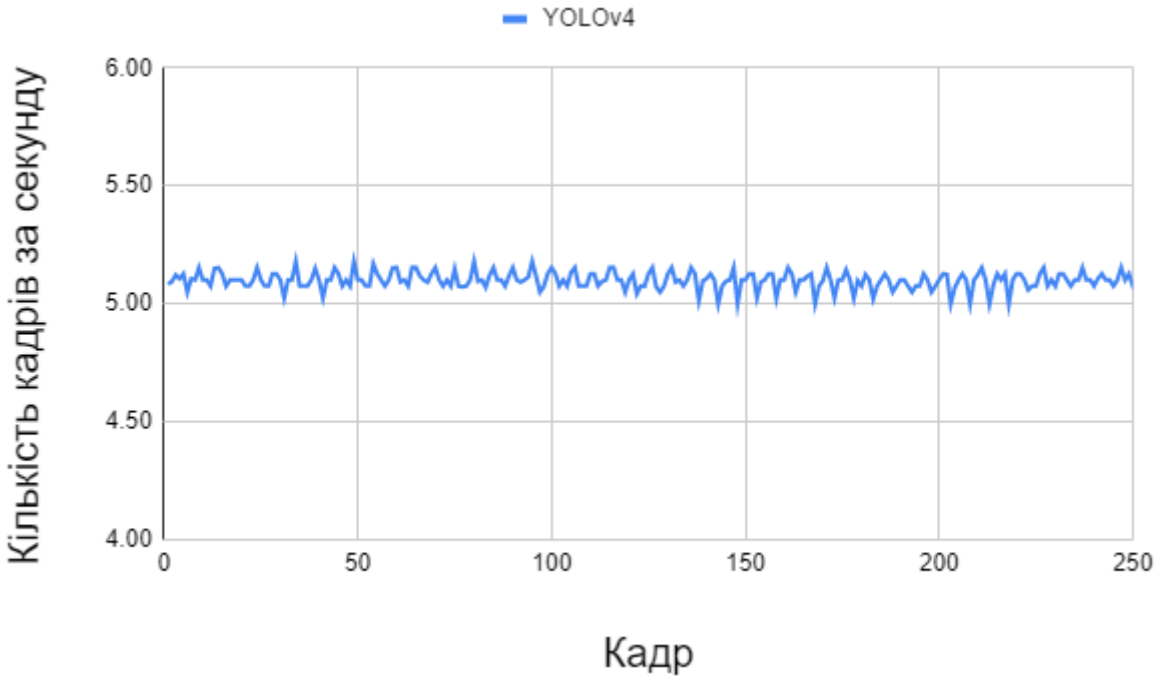


Рисунок 3.4 – Діаграма тестування FPS з використанням моделі YOLOv4.

У другому тесті було використано менш точну модель YOLOv4-tiny, проте більш швидку. Після обротки 250 кадрів середня кількість кадрів в секунду становить 63.153 кадрів в секунду. Цей показник удвічі перевищує необхідний та говорить про те, що на розпізнавання витрачається близько 15 мс.

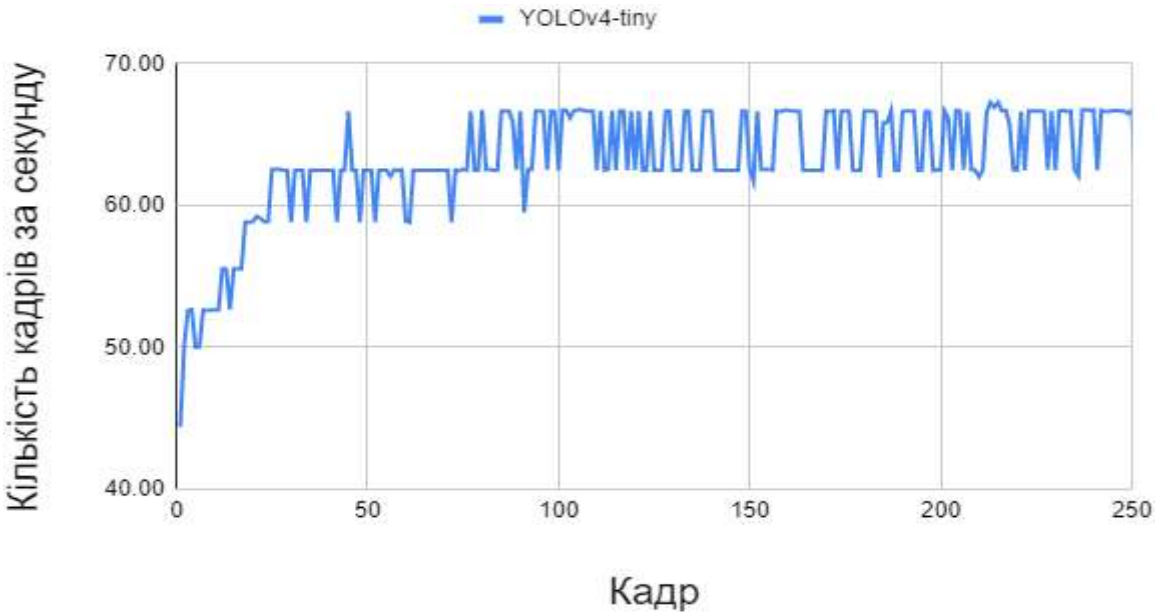


Рисунок 3.5 – Діаграма тестування FPS з використанням YOLOv4-tiny.

Необхідно також перевірити роботу розширення та всієї системи взагалі. Реалізоване розширення має меню, в якому можна вибрати модель нейронної мережі та кількість кадрів в секунду з якою будуть розпізнаватися образи на екрані.

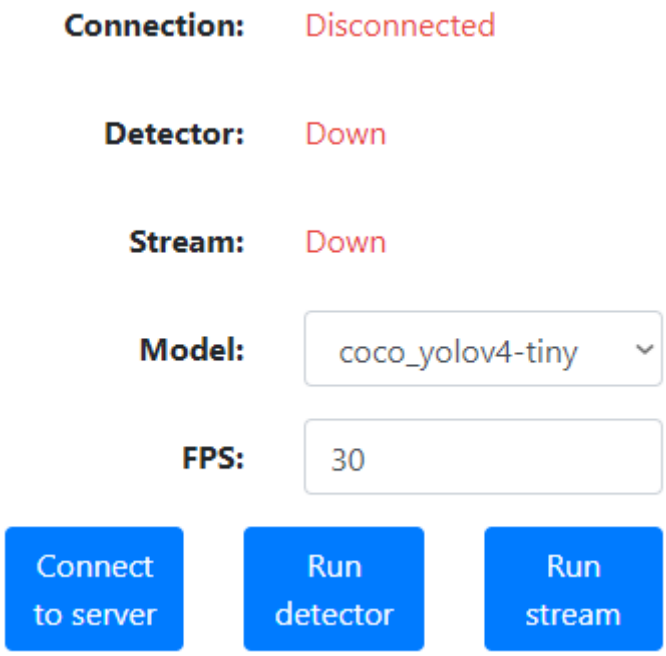


Рисунок 3.6 – Інтерфейс користувача розширення.

Для тесту було використано відео з сайту «YouTube» під назвою «HAWK pedestrian crossing "How To"». Результати роботи наведені на рисунках 3.7-3.9.

Рисунок 3.7 – Розпізнавання автомобілів під час перегляду відео.

Рисунок 3.8 – Розпізнавання світлофорів під час перегляду відео.

Рисунок 3.9 – Розпізнавання пішоходів під час перегляду відео.

Під час тестування використовувався персональний комп’ютер на базі процесора AMD Ryzen 5 1600 та графічного процесора NVIDIA GeForce GTX 1050 Ti. Результати тестування показують, що інтелектуальна система працює коректно.

ВИСНОВКИ

Розпізнавання образів у веб-контексті у реальному часі є важливим у сучасному, практичному аспекті. Його можна використовувати для різних цілей, такі як: цензурування контенту веб-сторінок, статистичного аналізу веб-ресурсів та інше. У процесі виконання завдання було виконано такі дослідження:

1. проведено аналітичний предметної області, виявлено проблеми та вибрано методи для їх вирішення;
2. обрано модель нейронної мережі, її імплементацію, методи та технології програмної реалізації;
3. розроблено та досліджено інтелектуальну систему розпізнавання образів у Web-контексті в режимі реального часу;
4. проведено тестування розробленої системи.

Під час аналітичного огляду предметної області зроблено висновок, що для реалізації поставленої задачі необхідно використовувати серверні технології по причині обмежень браузерів. Проектування перед реалізацією дало нам змогу зменшити час розробки програмного забезпечення.

Тестування показало, що використання серверних технологій та моделі YOLOv4 задовольняє поставлені вимоги для розпізнавання образів у веб-контексті в реальному часі.

Програму у майбутньому можна покращити створивши «балансувальника», завдання якого є розподіл потоку зображень на різні сервери, що забезпечить масштабування системи.

СПИСОК ЛІТЕРАТУРИ

1. 7 Amazing Examples Of Computer And Machine Vision In Practice Apr 8, 2019 by Bernard Marr - https://www.forbes.com/sites/bernardmarr/2019/04/08/7-amazing-examples-of-computer-and-machine-vision-in-practice/
2. An Energy-Efficient Hardware Implementationof HOG-Based Object Detection at 1080HD60 fps with Multi-Scale Support Amr Suleiman, Vivienne Sze, Journal of Signal Processing Systems, December 2015
3. Intersection over Union (IoU) for object detection by Adrian Rosebrock on November 7, 2016 - https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/
4. You Only Look Once:Unified, Real-Time Object Detection Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, University of Washington∗, Allen Institute for AI, Facebook AI Research 9 May 2016
5. Bottom-up Object Detection by Grouping Extreme and Center Points, Xingyi Zhou, Jiacheng Zhuo, Philipp Krähenbühl, Jan 23, 2019
6. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, 6 Jan 2016
7. SSD: Single Shot MultiBox Detector, Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg, UNC Chapel Hill 2Zoox Inc. Google Inc. University of Michigan, Ann-Arbor, 29 Dec 2016
8. YOLO: Real-Time Object Detection, Joseph Redmon – https://pjreddie.com/darknet/yolo/
9. YOLOv4 — the most accurate real-time neural network on MS COCO dataset. Aleksey Bochkovskiy 21 may 2020 – https://medium.com/@alexeyab84/yolov4-the-most-accurate-real-time-neural-network-on-ms-coco-dataset-73adfd3602fe

10. "Glossary:WebSockets". Mozilla Developer Network. 2015 – https://developer.mozilla.org/en-US/docs/Glossary/WebSockets
11. HTML5 WebSocket: A Quantum Leap in Scalability for the Web By Peter Lubbers & Frank Greco, Kaazing Corporation – http://www.websocket.org/quantum.html
12. RFC 6455 The WebSocket Protocol: Relationship to TCP and HTTP, Ian Fette; Alexey Melnikov (December 2011) – https://tools.ietf.org/html/rfc6455#section-1.7
13. Node.js v14.13.1 Documentation: Child process – https://nodejs.org/api/child_process.html
14. A Python Book: Beginning Python, Advanced Python, and Python Exercises, Dave Kuhlman April 22, 2012
15. ctypes — A foreign function library for Python – https://docs.python.org/3/library/ctypes.html
16. Browser & Platform Market Share: September 2020 – https://www.w3counter.com/globalstats.php
17. What are extensions? – https://developer.chrome.com/extensions
18. Densely Connected Convolutional Networks, Gao Huang, Zhuang Liu, Laurens van der Maaten, 28 Jan 2018 - https://arxiv.org/pdf/1608.06993.pdf
19. CSPNET: A NEW BACKBONE THAT CAN ENHANCE LEARNING CAPABILITY OF CNN, Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, 27 Nov 2019
20. YOLOv3: An Incremental Improvement, Joseph Redmon, Ali Farhadi, 8 Apr 2018
21. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition, Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, 23 Apr 2015
22. DC-SPP-YOLO: Dense Connection and Spatial Pyramid Pooling Based YOLO for Object Detection, Zhanchao Huang, Jianlin Wang, College of Information

Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China

23. CBAM: Convolutional Block Attention Module, Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon, 18 Jul 2018
24. YOLOv4: Optimal Speed and Accuracy of Object Detection, Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao, 23 Apr 2020
25. SEARCHING FOR ACTIVATION FUNCTIONS, Prajit Ramachandran∗, Barret Zoph, Quoc V. Le, 27 Oct 2017
26. Mish: A Self Regularized Non-Monotonic Activation Function, Diganta Misra, Landskape KIIT, Bhubaneswar, India, 13 Aug 2020
27. CUDA Toolkit – https://developer.nvidia.com/cuda-toolkit
28. NVIDIA cuDNN – https://developer.nvidia.com/cudnn
29. OpenCV: Introduction – https://docs.opencv.org/master/d1/dfb/intro.html
30. Yolo v4, v3 and v2 for Windows and Linux – https://github.com/AlexeyAB/darknet
31. AlexeyAB/darknet/src/detector.c – https://github.com/AlexeyAB/darknet/blob/d65909fbea471d06e52a2e4a41132380dc2edaa6/src/detector.c#L1640
32. Base64 – https://developer.mozilla.org/en-US/docs/Glossary/Base64
33. What Socket.IO is – https://socket.io/docs/

ДОДАТОК А

```
char* fgets_long(FILE* fp) {
    size_t size = 0, currlen = 0;
    char line[1024];
    char* ret = NULL, * tmp;
    while (fgets(line, sizeof line, fp)) {
        int wholeline = 0;
        size_t len = strlen(line);
        if (line[len - 1] == '\n') {
            line[len-- - 1] = 0;
            wholeline = 1;
        }
        if (currlen + len >= size) {
            size += (sizeof line) - (size ? 1 : 0);
            tmp = realloc(ret, size);
            if (tmp == NULL)
                break; // return all we've got so far
            ret = tmp;
        }
        memcpy(ret + currlen, line, len + 1);
        currlen += len;
        if (wholeline)
            break;
    }
    if (ret){
        tmp = realloc(ret, currlen + 1);
        if (tmp)
            ret = tmp;
    }
    return ret;
}
```

ДОДАТОК Б

```
extern "C" mat_cv *load_image_mat_cv_base64(const char *filename, int flag)
{
    cv::Mat *mat_ptr = NULL;
    try {
        std::string dec_jpg = base64_decode(filename);
        std::vector<uchar> data(dec_jpg.begin(), dec_jpg.end());
        cv::Mat mat = cv::imdecode(cv::Mat(data), flag);
        if (mat.empty()) {
            std::string shrinked_filename = filename;
            if (shrinked_filename.length() > 1024) {
                shrinked_filename.resize(1024);
                shrinked_filename = std::string("name is too long: ") + shrinked_filename;
            }
            cerr << "Cannot load image " << shrinked_filename << std::endl;
            std::ofstream bad_list("bad.list", std::ios::out | std::ios::app);
            bad_list << shrinked_filename << std::endl;
            //if (check_mistakes) getchar();
            return NULL;
        }
        cv::Mat dst;
        if (mat.channels() == 3) cv::cvtColor(mat, dst, cv::COLOR_RGB2BGR);
        else if (mat.channels() == 4) cv::cvtColor(mat, dst, cv::COLOR_RGBA2BGRA);
        else dst = mat;
        mat_ptr = new cv::Mat(dst);
        return (mat_cv *)mat_ptr;
    }
    catch (...) {
        cerr << "OpenCV exception: load_image_mat_cv \n";
    }
```

```
    if (mat_ptr) delete mat_ptr;
    return NULL;
}
cv::Mat load_image_mat_base64(char* filename, int channels) {
    int flag = cv::IMREAD_UNCHANGED;
    if (channels == 0) flag = cv::IMREAD_COLOR;
    else if (channels == 1) flag = cv::IMREAD_GRAYSCALE;
    else if (channels == 3) flag = cv::IMREAD_COLOR;
    else {
        fprintf(stderr, "OpenCV can't force load with %d channels\n", channels);
    }
    //flag |= IMREAD_IGNORE_ORIENTATION;    // un-comment it if you want

    cv::Mat* mat_ptr = (cv::Mat*)load_image_mat_cv_base64(filename, flag);
    if (mat_ptr == NULL) {
        return cv::Mat();
    }
    cv::Mat mat = *mat_ptr;
    delete mat_ptr;
    return mat;
}
extern "C" image load_image_cv_base64(char* filename, int channels) {
    cv::Mat mat = load_image_mat_base64(filename, channels);
    if (mat.empty()) {
        return make_image(10, 10, channels);
    }
    return mat_to_image(mat);
}
```

ДОДАТОК В

```
import {spawn} from 'child_process';
function trimToJSON(str: string) {
    str = str.trim();
    const start = '{';
    const end = '}';
    if (str.indexOf(start) !== -1 && str.indexOf(end) !== -1) {
        while (str.charAt(0) !== start) {
            str = str.substring(1)
        }
        while (str.charAt(str.length - 1) !== end) {
            str = str.substring(0, str.length - 2)
        }
        return str
    }
    return null
}
export interface DetectorOptions {
    mode: string;
    dataset: string;
    model: string;
    weights: string;
    url?: string;
    json_port?: string;
    mjpeg_port?: string;
    thresh?: string;
    ext_output?: boolean;
    dont_show?: boolean;
}
export class Detector {
    private process;
    private readonly folderPath: string = __dirname + "/dlls";
    private readonly executeName: string = "darknet.exe";
    private options: DetectorOptions;
    private spawnOptions: Array<string>;
    running: boolean = false;
    working: boolean = false;
    waiting: boolean = false;
    errored: boolean = false;
    private launchResolve;
    private launchReject;
    private detectionResolve;
```

```
    private detectionReject;
    private debug = true;
    private prevTime = Date.now() / 1000;
    constructor(options: DetectorOptions) {
        this.options = options;
        this.spawnOptions = this.getSpawnOptions()
    }
    getSpawnOptions() {
        let options = this.options;
        let spawnOptions: Array<string> = [
            "detector"
        ];
        spawnOptions.push(...[options.mode, options.dataset, options.model,
options.weights]);
        if (options.url) {
            spawnOptions.push(options.url)
        }
        if (options.json_port) {
            spawnOptions.push(...["-json_port", options.json_port])
        }
        if (options.mjpeg_port) {
            spawnOptions.push(...["-mjpeg_port", options.mjpeg_port])
        }
        if (options.thresh) {
            spawnOptions.push(...["-thresh", options.thresh])
        }
        if (options.ext_output) {
            spawnOptions.push("-ext_output")
        }
        if (options.dont_show) {
            spawnOptions.push("-dont_show")
        }
        return spawnOptions
    }
    setModel(model: string) {
        this.options.model = "./cfg/" + model + ".cfg";
        this.options.weights = "./" + model + ".weights";
    }
    setDataset(dataset: string) {
        this.options.weights = "./cfg/" + dataset + ".data";
    }
    onRunning(data) {
        if (data.toString().includes("Enter Image Base64")) {
            this.launchResolve(this.responses.up);
```

```
            this.working = true;
            this.running = false;
        }
    }
    onWaiting(str: string) {
        str = trimToJSON(str);
        if (str) {
            this.detectionResolve(str);
            this.waiting = false;
        }
    }
    onClose(code) {
        console.log('closing code: ' + code);
        this.working = false;
    }
    async run(dataset_model?: string): Promise<any> {
        if (this.process) {
            await this.stop()
        }
        if (dataset_model) {
            let [dataset, model] = dataset_model.split('_');
            this.setDataset(dataset);
            this.setModel(model);
            this.spawnOptions = this.getSpawnOptions();
        }
        this.running = true;
        this.process = spawn(this.executeName, this.spawnOptions, {
            cwd: this.folderPath
        });
        this.process.stdout.on('data', (data) => {
            if (this.running) {
                this.onRunning(data)
            }
            if (this.waiting) {
                this.onWaiting(data.toString())
            }
        });
        this.process.on('close', (code) => {
            this.launchReject(this.responses.errorRun);
            this.onClose(code)
        });
        return new Promise((resolve, reject) => {
            this.launchResolve = resolve;
            this.launchReject = reject
```

```
        });
    }
    async stop() {
        return new Promise(((resolve, reject) => {
            if (this.process && this.working) {
                let stopped = this.process.kill();
                if (stopped) {
                    console.log("Detector stopped.");
                    resolve(this.responses.down)
                }
                else {
                    reject(this.responses.errorStop)
                }
            }
            else {
                resolve(this.responses.down)
            }
        }))
    }
    async detect(base64: string): Promise<any> {
        return new Promise((resolve, reject) => {
            if (this.process && this.working && !this.waiting) {
                this.detectionResolve = resolve;
                this.detectionReject = reject;
                this.waiting = true;
                this.process.stdin.write(base64 + "\n");
            }
            else {
                reject(this.responses.errorDetect)
            }
        })
    }
}
```

ДОДАТОК Г

```
from ctypes import *
import math
import random
import os
import cv2
import numpy as np
import time
import darknet
import asyncio
import websockets
import base64
import json
netMain = None
metaMain = None
altNames = None
darknet_image = None
darknet_image_size = None
def run_yolo():
    global metaMain, netMain, altNames, darknet_image, darknet_image_size
    configPath = "./cfg/yolov4-tiny.cfg"
    weightPath = "./yolov4-tiny.weights"
    metaPath = "./cfg/coco.data"
    if not os.path.exists(configPath):
        raise ValueError("Invalid config path `" +
                         os.path.abspath(configPath) + "`")
    if not os.path.exists(weightPath):
        raise ValueError("Invalid weight path `" +
                         os.path.abspath(weightPath) + "`")
    if not os.path.exists(metaPath):
        raise ValueError("Invalid data file path `" +
                         os.path.abspath(metaPath) + "`")
    if netMain is None:
        netMain = darknet.load_net_custom(configPath.encode(
            "ascii"), weightPath.encode("ascii"), 0, 1)  # batch size = 1
    if metaMain is None:
        metaMain = darknet.load_meta(metaPath.encode("ascii"))
    if altNames is None:
        try:
            with open(metaPath) as metaFH:
                metaContents = metaFH.read()
                import re
```

```
            match = re.search("names *= *(.*)$", metaContents,
                              re.IGNORECASE | re.MULTILINE)
            if match:
                result = match.group(1)
            else:
                result = None
            try:
                if os.path.exists(result):
                    with open(result) as namesFH:
                        namesList = namesFH.read().strip().split("\n")
                        altNames = [x.strip() for x in namesList]
            except TypeError:
                pass
        except Exception:
            pass
    if darknet_image is None:
        darknet_image = darknet.make_image(darknet.network_width(netMain),
                                           darknet.network_height(netMain), 3)
    if darknet_image_size is None:
        darknet_image_size = [darknet.network_width(netMain),
darknet.network_height(netMain)]
def convertDetections(detections):
    objects = []
    for detection in detections:
        objects.append([detection[0].decode(), detection[1], detection[2],
darknet_image_size])
    return objects
def convertLabels(detections):
    for i in range(len(detections)):
        detections[i][0] = detections[i][0].decode()
def base64_to_image(base64_img: str):
    buff = base64.b64decode(base64_img)
    np_arr = np.frombuffer(buff, np.uint8)
    image = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
    return image
async def base64_handle(websocket, path):
    async for message in websocket:
        img = base64_to_image(message)
        img_resized = cv2.resize(img,
(darknet.network_width(netMain),darknet.network_height(netMain)),
                                 interpolation=cv2.INTER_LINEAR)
        darknet.copy_image_from_bytes(darknet_image, img_resized.tobytes())
        detections = darknet.detect_image(netMain, metaMain, darknet_image,
thresh=0.25)
```

```
        json_response = json.dumps(convertDetections(detections))
        await websocket.send(json_response)
if __name__ == "__main__":
    run_yolo()
    start_server = websockets.serve(base64_handle, "localhost", 8080)
    asyncio.get_event_loop().run_until_complete(start_server)
    asyncio.get_event_loop().run_forever()
```

ДОДАТОК Ґ

```
import {messages as msgs} from "./messages";
let socketUrl = null;
let socket = null;
let interval = null;
let stages = {
    server: null,
    detector: null,
    stream: null
};
let delay = 1000;
function serverConnection(url) {
    return new Promise((resolve, reject) => {
        if (socket) {
            if (socketUrl === url) {
                socket.open()
            } else {
                socket.close();
                socket = io(url);
            }
        }
        else {
            socket = io(url);
        }
        socket.on('connect', () => {
            stages.server = msgs.server.up.stage;
            resolve(msgs.server.up)
        });
        socket.on('disconnect', () => {
            stages.server = msgs.server.down.stage;
        });
    })
}
function serverDisconnection() {
    return new Promise(resolve => {
        if (socket) {
            socket.close();
        }
        stages.server = msgs.server.down.stage;
        resolve(msgs.server.down)
    })
}
```

```
function detectorRun(model) {
    return new Promise(((resolve, reject) => {
        if (socket && socket.connected) {
            socket.emit("runDetector", model, response => {
                stages.detector = response.stage;
                if (response.stage === msgs.detector.up.stage) {
                    resolve(response)
                } else {
                    reject(msgs.detector.error)
                }
            })
        }
        else {
            reject(msgs.detector.error)
        }
    }))
}
function detectorStop() {
    return new Promise(((resolve, reject) => {
        if (socket && socket.connected) {
            socket.emit("stopDetector",  null, response => {
                stages.detector = response.stage;
                if (response.stage === msgs.detector.down.stage) {
                    resolve(response)
                } else {
                    reject(msgs.detector.error)
                }
            })
        }
        else {
            reject(msgs.detector.error)
        }
    }))
}
function streamRun(fps) {
    return new Promise(((resolve, reject) => {
        if (socket && socket.connected) {
            startStream(fps);
            stages.stream = msgs.stream.up.stage;
            resolve(msgs.stream.up)
        }
        else {
            stages.stream = msgs.stream.down.stage;
            reject(msgs.stream.error)
```

```
        }
    }))
}
function streamStop() {
    return new Promise(((resolve) => {
        if (interval) {
            clearInterval(interval);
        }
        stages.stream = msgs.stream.down.stage;
        resolve(msgs.stream.down)
    }))
}
function getActiveTabScreenProps() {
    return new Promise((resolve, reject) => {
        chrome.tabs.query({active: true, currentWindow: true}, function(tabs) {
            chrome.tabs.sendMessage(tabs[0].id, {request: "windowProps"},
function(response) {
                console.log(response);
                resolve(response);
            });
        });
    })
}
function relativeToAbsolute(screenW, screenH, xOffset, yOffset, xRel, yRel, wRel,
hRel) {
    let absoluteCenterX = Math.round(screenW * xRel) + xOffset;
    let absoluteCenterY = Math.round(screenH * yRel) + yOffset;
    let absoluteW = Math.round(screenW * wRel);
    let absoluteH = Math.round(screenH * hRel);
    let absoluteX = absoluteCenterX - Math.round(absoluteW / 2);
    let absoluteY = absoluteCenterY - Math.round(absoluteH / 2);
    return {x: absoluteX, y: absoluteY, w: absoluteW, h: absoluteH};
}
function emitDrawBoxes(boxes) {
    chrome.tabs.query({active: true, currentWindow: true}, function(tabs) {
        chrome.tabs.sendMessage(tabs[0].id, {request: "drawBoxes", boxes: boxes},
function(response) {
            console.log(response);
            resolve(response);
        });
    });
}
function sendScreenShot() {
    if (socket && socket.connected) {
```

```
        getActiveTabScreenProps().then(screenProps => {
            let boxes = [];
            chrome.tabs.captureVisibleTab(dataUrl => {
                let base64 = dataUrl.split(',')[1];
                socket.emit("frame", base64, response => {
                    console.log(response);
                    for (let obj of response) {
                        let box = relativeToAbsolute(
                            screenProps.inner.w,
                            screenProps.inner.h,
                            screenProps.offset.x,
                            screenProps.offset.y,
                            obj.relative_coordinates.center_x,
                            obj.relative_coordinates.center_y,
                            obj.relative_coordinates.width,
                            obj.relative_coordinates.height,
                        );
                        box["label"] = obj.name + ": " + obj.confidence.toFixed(2) + "%";
                        boxes.push(box);
                    }
                    emitDrawBoxes(boxes);
                })
            });
        });
    }
}
function startStream(fps) {
    if (interval) {
        clearInterval(interval)
    }
    if (fps > 1 && fps <= 30) {
        delay = 1000 / fps;
    }
    interval = setInterval(sendScreenShot, delay)
}
chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {
    console.log(message);
    if (message.request === "stagesCheck") {
        console.log(stages);
        sendResponse(stages)
    }
    if (message.request === "serverConnection") {
        serverConnection(message.url).then(sendResponse).catch(sendResponse)
    }
```

```
    if (message.request === "serverDisconnection") {
        serverDisconnection().then(sendResponse)
    }
    if (message.request === "detectorRun") {
        detectorRun(message.model).then(sendResponse)
    }
    if (message.request === "detectorStop") {
        detectorStop().then(sendResponse)
    }
    if (message.request === "streamRun") {
        streamRun(message.fps).then(sendResponse)
    }
    if (message.request === "streamStop") {
        streamStop().then(sendResponse)
    }
    return true
});
```

ДОДАТОК Д

```
let delta = 15;
let boxes = [];
function innerResolution() {
   return {w: window.innerWidth, h: window.innerHeight}
}
function pageOffset() {
   return {x: window.pageXOffset, y: window.pageYOffset}
}
function windowProps() {
   return {inner: innerResolution(), offset: pageOffset()}
}
function clearAll() {
   for (let item of boxes) {
      item.element.remove();
   }
}
function drawBox(x, y, w, h, label) {
   let box = document.createElement("div");
   box.style.position = "absolute";
   box.style.left = x+"px";
   box.style.top = y+"px";
   box.style.width = w+"px";
   box.style.height = h+"px";
   box.style.border = "1px solid red";
   box.innerText = label;
   box.style.color = "red";
   boxes.push({x: x, y: y, w: w, h: h, element: box});
   document.body.appendChild(box);
}
chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {
   if (message.request === "windowProps") {
      sendResponse(windowProps())
   }
   if (message.request === "drawBoxes") {
      clearAll();
      for (let item of message.boxes) {
         drawBox(item.x,item.y, item.w, item.h, item.label);
      }
   }
   return true
});
```

ДОДАТОК Е

```
<template>
  <main class="py-4">
    <div class="container">
      <div class="row justify-content-center">
        <div class="col-md-12">
          <div class="card">
            <div class="card-body">
              <status-label :label="'Connection:'" :stages="serverStages"
:stage="serverStage"></status-label>
              <status-label :label="'Detector:'" :stages="detectorStages"
:stage="detectorStage"></status-label>
              <status-label :label="'Stream:'" :stages="streamStages"
:stage="streamStage"></status-label>
              <div class="form-group row">
                <label for="model" class="col-5 col-form-label text-right font-
weight-bold">Model:</label>
                <div class="col-7">
                  <select id="model" type="password" class="form-control" v-
model="model" required>
                    <option v-for="model in models" :value="model">{{ model
}}</option>
                  </select>
                </div>
              </div>
              <div class="form-group row">
                <label for="fps" class="col-5 col-form-label text-right font-
weight-bold">FPS:</label>
                <div class="col-7">
                  <input id="fps" type="number" class="form-control" v-
model="fps" placeholder="Frame rate" required>
                </div>
              </div>
              <div class="form-group row mb-0">
                <action-button :stages="serverStages"
:stage="serverStage"></action-button>
                <action-button :stages="detectorStages"
:stage="detectorStage"></action-button>
                <action-button :stages="streamStages"
:stage="streamStage"></action-button>
              </div>
            </div>
```

```
                </div>
            </div>
        </div>
    </div>
  </main>
</template>
<script>
  import {messages} from "../messages";
  export default {
    name: "popup-form",
    data: function () {
      return {
        serverStages: {…},
        serverStage: "down",
        detectorStages: {…},
        detectorStage: "down",
        streamStages: {…},
        streamStage: "down",
        models: [
          "coco_yolov4",
          "coco_yolov4-tiny"
        ],
        model: null,
        fps: null,
        url: "http://localhost:8080",
      }
    },
    mounted: function () {
      this.$nextTick(function () {
        this.localLoad(["model", "fps"]).then(items => {
          console.log(items);
          this.model = items.model;
          this.fps = items.fps;
        });
        this.stagesCheck();
      })
    },
    methods: {
      stagesCheck() {
        chrome.runtime.sendMessage({request: "stagesCheck"}, response => {
          console.log(response);
          if (response.server)  this.serverStage = response.server;
          if (response.detector)  this.detectorStage = response.detector;
          if (response.stream)  this.streamStage = response.stream;
```

```
        });
    },
    localLoad(keys) {
        return new Promise(resolve => {
            let obj  = {};
            if (keys instanceof Array) {
                for (let key of keys) {
                    obj[key] = null
                }
            }
            chrome.storage.local.get(obj, (items) => {
                resolve(items)
            })
        });
    },
    localSave(dataObject) {
        return new Promise(resolve => {
            chrome.storage.local.set(dataObject, function () {
                resolve()
            })
        });
    },
    serverConnecting() {
        this.serverStage = "upping";
        chrome.runtime.sendMessage({request: "serverConnection", url: this.url},
response => {
            console.log(response);
            if (response.stage === messages.server.up.stage) {
                this.onServerConnected()
            }
            if (response.stage === messages.server.down.stage) {
                this.onServerDisconnected()
            }
            if (response.stage === messages.server.error.stage) {
                this.onServerDisconnected()
            }
        });
    },
    onServerConnected() {
        this.serverStage = "up";
    },
    serverDisconnecting() {
        this.serverStage = "stopping";
        chrome.runtime.sendMessage({request: "serverDisconnection", url:
```

```
this.url}, response => {
                console.log(response);
                if (response.stage === "down") {
                    this.onServerDisconnected()
                }
                else {
                    //error
                    this.onServerDisconnected()
                }
            });
        },
        onServerDisconnected() {
            this.serverStage = "down";
        },
        // --- Detector Launching ---
        detectorRun() {
            if (this.serverStage !== messages.server.up.stage) {
                return
            }
            this.detectorStage = "upping";
            this.localSave({model: this.model});
            chrome.runtime.sendMessage({request: "detectorRun", model: this.model},
response => {
                console.log(response);
                if (response.stage === messages.detector.up.stage) {
                    this.onDetectorUp()
                }
                if (response.stage === messages.detector.down.stage) {
                    this.onDetectorDown()
                }
                if (response.stage === messages.detector.error.stage) {
                    this.onDetectorDown()
                }
            });
        },
        onDetectorUp() {
            this.detectorStage = "up"
        },
        detectorStop() {
            this.detectorStage = "stopping";
            chrome.runtime.sendMessage({request: "detectorStop"}, response => {
                console.log(response);
                if (response.status === messages.detector.down.stage) {
                    this.onDetectorDown()
```

```
            }
            else {
                //error
                this.onDetectorDown()
            }
        });
    },
    onDetectorDown() {
        this.detectorStage = "down"
    },
    streamRun() {
        if (this.detectorStage !== messages.detector.up.stage) {
            return
        }
        this.streamStage = "upping";
        this.localSave({model: this.model});
        chrome.runtime.sendMessage({request: "streamRun", fps: this.fps},
response => {
            console.log(response);
            if (response.stage === messages.stream.up.stage) {
                this.onStreamUp()
            }
            if (response.stage === messages.stream.down.stage) {
                this.onStreamDown()
            }
            if (response.stage === messages.stream.error.stage) {
                this.onStreamDown()
            }
        });
    },
    onStreamUp() {
        this.streamStage = "up"
    },
    streamStop() {
        this.streamStage = "stopping";
        chrome.runtime.sendMessage({request: "streamStop", model: this.model},
response => {
            console.log(response);
            if (response.status === messages.stream.down.stage) {
                this.onStreamDown()
            }
            else {
                //error
                this.onStreamDown()
```

```
                }
            });
        },
        onStreamDown() {
            this.streamStage = "down"
        }
    }
  }
</script>
```