МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ САДІВНИЦТВА

Факультет економіки і підприємництва

Кафедра інформаційних технологій

Допущено до захисту
"_____" _____ 2023 р.
Зав. кафедри, к.т.н., доцент
_____ **Ліщук Р.І.**

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

СИСТЕМА ПІДРАХУНКУ ТРАНСПОРТНИХ ЗАСОБІВ

Виконав – студент 2 курсу, групи 21к-кн
освітньої програми "Комп'ютерні науки"
спеціальності 122 Комп'ютерні науки
Кушнерик Богдан Олександрович

Науковий керівник – к.е.н., доцент
Ліщук Роман Ігорович

Умань – 2023

Перелік скорочень

YOLO – You Only Look Once

IoU – Intersection over Union

CNN – Convolution Neural Network

GPU – Graphics Processing Unit

Зміст

Вступ 4
Розділ 1 Дослідження технологій детектування об’єктів. 6
1.1. Огляд методів детектування 6
1.2. Постановка задачі 17
Розділ 2 Огляд програмних засобів для виконання роботи. 18
2.1. Архітектура YOLOv7. 18
2.2. Додаткові технології та програмне забезпечення для реалізації проекту. 26
РОЗДІЛ 3 Підготовка даних та виконання завдання 28
3.1. Підготовка даних та тренування нейронної мережі YOLOv7. 28
3.2. Детектування та підрахунок об’єктів у відео потоці. 36
Висновок 38
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 39
Додатки 41

ВСТУП

Детектування об'єктів є одними з найсучасніших і популярних функцій штучного інтелекту і технологій комп'ютерного зору. Однією з головних причин популяризації та збільшення використання технологій комп’ютерного зору в короткі терміни є те, що вони забезпечують відмінний розвиток бізнесу та можливості отримання прибутку для різних підприємств, а також можуть заощадити кошти в майбутній перспективі.

Комп'ютерний зір значно просунувся, але все ще залишається проблемним у відповідності точності людського сприйняття. Завдання детектування об'єктів на зображенні, яке вважалося надзвичайно складною проблемою лише кілька років тому, зараз здійсненне і навіть було розроблено такими компаніями, як Google та IBM.[1]

Серед різних сфер застосування детектування об’єктів, велику актуальність в багатьох галузях набуло детектування транспортних засобів, зокрема в транспортному бізнесі, міському плануванні та управлінні дорожнім рухом, безпеці дорожнього руху, екології та інших сферах. Також, детектування транспортних засобів за допомогою камер в реальному часі може використовуватися для оптимізації роботи міської інфраструктури, забезпечення комфортного і безпечного життя мешканців, детектування аварій на дорогах, відслідковування заторів, підрахунку пасажиропотоку та інших завдань, які можуть забезпечити більш ефективне та безпечне використання транспорту у містах.

В даній роботі метою є розробка підсистеми підрахунку автомобілів, що пересікають певну (створену додатково) лінію, для системи детектування об’єктів в відео.

Існує різноманітна кількість варіантів детектування об’єктів на відео. В роботі будуть розглядатися деякі з найвідоміших методів та, на основі їх огляду, буде обрана система для подальшого використання.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ДЕТЕКТУВАННЯ ОБ’ЄКТІВ.

1.1. Огляд методів детектування

На початку роботи постає завдання вибору класифікатора для його подальшої модифікації.

Детектування об'єктів може бути виконано за допомогою традиційних методів обробки зображень або сучасних мереж глибокого навчання.

1. Методи обробки зображень, як правило, не вимагають історичних даних для навчання і не контролюються за своєю природою. OpenCV є популярним інструментом для обробки зображень. Плюси: Отже, ці завдання не вимагають анотованих зображень, де люди позначають дані вручну (для контрольованого навчання). Мінуси: Ці методи обмежені багатьма факторами, такими як складні сценарії (без одноколірного фону), оклюзія (частково приховані об'єкти), освітлення та тіні, а також ефект безладу.
2. Методи глибокого навчання, як правило, залежать від контрольованого або неконтрольованого навчання, причому контрольовані методи є стандартом у завданнях комп'ютерного зору. Продуктивність обмежена обчислювальною потужністю графічних процесорів, яка швидко зростає з кожним роком. Плюси: Детектування об'єктів глибокого навчання значно стійкіше до оклюзії, складних сцен і складного освітлення. Мінуси: Потрібна величезна кількість навчальних даних; Процес анотування зображень трудомісткий і дорогий. Наприклад, маркування 500.000 зображень для навчання користувацького алгоритму детектування об'єктів DL вважається невеликим набором даних. Однак багато еталонних

наборів даних (MS COCO, Caltech, KITTI, PASCAL VOC, V5) забезпечують доступність маркованих даних.

Одним з поширених методів детектування об’єктів є каскади Хаара. Каскади Хаара – це алгоритм детектування об’єктів машинного навчання, який ідентифікує об’єкти на зображенні та відео.

Алгоритм можна пояснити в чотири етапи: Розрахунок функцій Хаара; Створення цілісних образів; Використання AdaBoost; Реалізація каскадних класифікаторів. Важливо пам'ятати, що цей алгоритм вимагає багато позитивних зображень (зображення що містять об’єкт детектування) і негативних зображень (зображення, на яких відсутні об’єкти детектування) для навчання класифікатора, подібно до інших моделей машинного навчання.[4]

Насамперед необхідно зібрати функції Хаара. В свою чергу функція Хаара – це, по суті, розрахунки, які виконуються на суміжних прямокутниках областей у певному місці у вікні виявлення [4]. Використовується три види функцій: Значення функцій двох прямокутників – це різниця між сумами пікселів у двох прямокутних областях (верхні два приклади на рис. 1); функція трьох прямокутників обчислює суму пікселів в двох зовнішніх прямокутниках та віднімає суму середнього прямокутника (дві середні функції на рис. 1); та використання функцій чотирьох прямокутників обчислює різницю між сумами пікселів в прямокутниках по діагоналі (нижній приклад на рис. 1.1). [5]

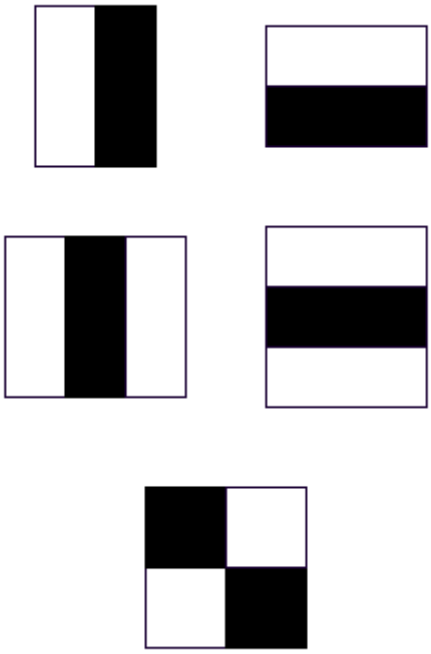

Рис. 1.1 Приклади функцій Хаара

Функції буває складно визначити для великого зображення. Для вирішення цієї проблеми використовуються інтегральне зображення, за допомогою якого зменшується кількість операцій.

Суть інтегрального зображення полягає в тому, що для кожної точки в зображення визначається сума всіх пікселів в прямокутнику, обмеженому лінією, проведеною з нижньої сторони пікселя до лівої сторони, та лінії, проведеної з правої сторони пікселя в верх. [6] За допомогою отриманого масиву можна скоротити необхідну кількість операцій, необхідних для визначення суми значень пікселів в певній частині зображення. В [5] наводиться приклад (за основу взятий рис. 1.2) формули визначення суми елементів в прямокутнику D в такому вигляді:

$$D = 4 + 1 – (2 + 3)$$

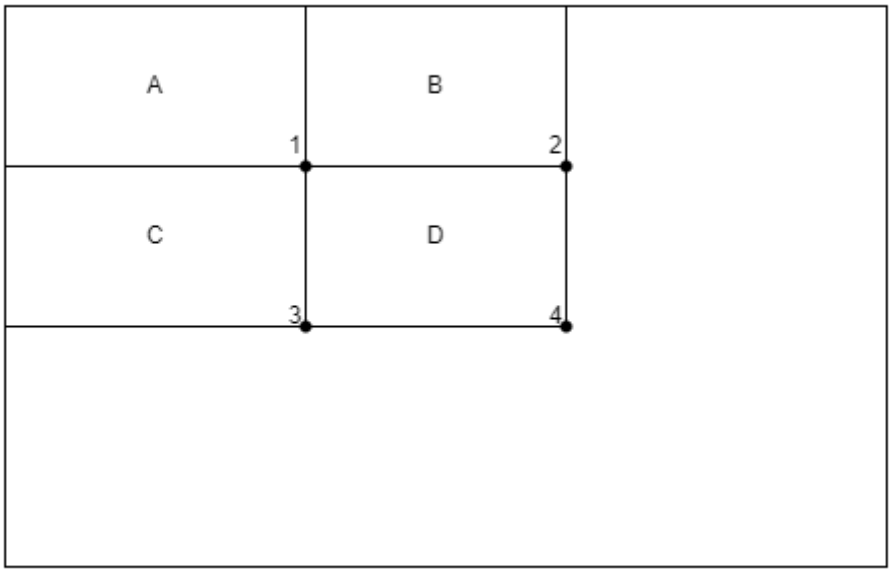

Рис. 1.2 Приклад визначення меж зображення для чотирьох точок

В результаті, з використання інтегрального зображення можна визначити суму елементів в прямокутнику використовуючи лише 4 посилання на суми певних масивів. Тобто для визначення функції з, наприклад, 2 прямокутників знадобиться лише 8 чисел, що значно скорочує час обробки даних.[5]

Оскільки майже всі функції Хаара будуть не релевантні при детектуванні об’єктів (оскільки важливі лише особливості об’єкта), то для визначення найкращих функцій використовується навчання AdaBoost. [4]

Adaboost по суті обирає найкращі функції та навчає класифікатор їх використовувати. Він використовує комбінацію «слабких класифікаторів» для створення «сильного класифікатора», який алгоритм може використовувати для детектування об'єктів. Слабкі учні створюються шляхом переміщення вікна над вхідним зображенням та обчисленням функцій Хаара для кожного підрозділу зображення. Ця різниця порівнюється з вивченим порогом, який відокремлює не об'єкти від об'єктів. Останній крок об’єднує цих слабких учнів у сильного за допомогою. [4]

Каскадний класифікатор складається з серії етапів, де кожен етап є сукупністю слабких учнів. Слабкі учні навчаються за допомогою бустингу, що дозволяє отримати високоточний класифікатор із середнього прогнозу всіх слабких учнів. На підставі цього прогнозу класифікатор або приймає рішення вказати, що об'єкт був знайдений, або переходить до наступної області. [4]

В кінці хочеться зауважити, що каскади Хаара є не дуже точним, а також при підвищенні точності він ускладнюється що приводить до збільшення часу детектування. Але даний детектор не вимагає надто великих обчислювальних потужностей. Також позитивною стороною каскадів Хаара є наявність навчених моделей для певних об’єктів.

Для детектування об’єктів також може бути використана згорткова нейронна мережа (CNN), яка є класом нейронних мереж, що спеціалізуються на обробці даних, що мають сітчасту топологію, таку як зображення.

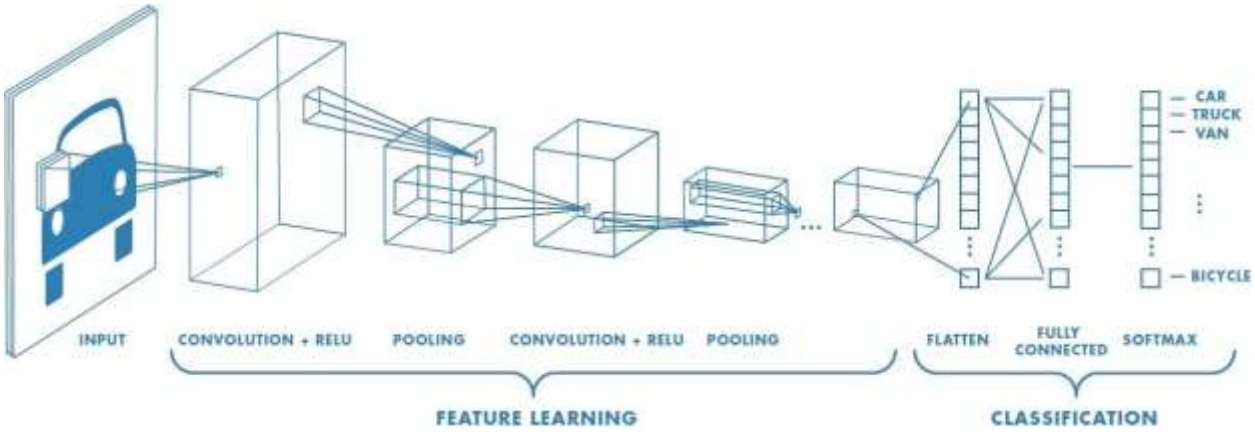

Рис. 1.3 Архітектура CNN

CNN є послідовністю шарів (рис. 1.3), і кожен шар перетворює один об’єм в інший за допомогою диференційованої функції. До загорткової нейронної мережі відносяться такі шари, як [7]:

1. Вхідний шар – це шар, в якому надаються вхідні дані в нашій моделі. Вхідними даними можуть бути як фото, так і послідовність фото;

2. Згортковий шар – це шар, який використовується для вилучення функції з набору вхідних даних. Він застосовує набір навчальних фільтрів, відомих як ядра, до вхідних зображень;
3. Шар активації – додаючи функцію активації до виводу попереднього шару , шар активації додає мережі не лінійності. Далі, він застосовує функцію активації по елементу до виходу шару згортки.
4. Шар об'єднання – цей шар періодично вставляється в CNN. Його основною функцією є зменшення розміру карт об’єктів, що робить обчислення швидким, скорочує пам'ять, а також запобігає переоснащенню.
5. Зведення – отримані карти ознак зводяться в одновимірний вектор після шарів згортки та об'єднання, для того щоб в подальшому можна було передати в повністю зв’язаний шар для категоризації або регресії.
6. Повністю пов'язаний шар – він бере вхідні дані з минулого шару та обчислює фінальну класифікацію або завдання регресії.
7. Шар виведення – вихідні дані з повністю пов’язаних шарів далі подаються в логістичну функцію для завдань класифікації, яка перетворює вихідні дані кожного класу в оцінку ймовірності кожного класу.

Шар згортки є фундаментальним компонентом архітектури CNN. Згортка це спеціалізований тип лінійної операції, що використовується для вилучення ознак, де невеликий масив чисел, званий ядром, застосовується на вході, який є масивом чисел, званим тензором. По-елементний добуток між кожним елементом ядра і вхідним тензором обчислюється в кожному місці тензора і підсумовується для отримання вихідного значення у відповідному положенні вихідного тензора, званого картою ознак (рис. 1.4). Ця процедура повторюється, застосовуючи кілька ядер для формування довільної кількості карт ознак, які представляють різні характеристики вхідних тензорів. Різні ядра можуть розглядатися як екстрактори різних

функцій. Двома ключовими гіперпараметрами, які визначають операцію згортки, є розмір і кількість ядер. [8]

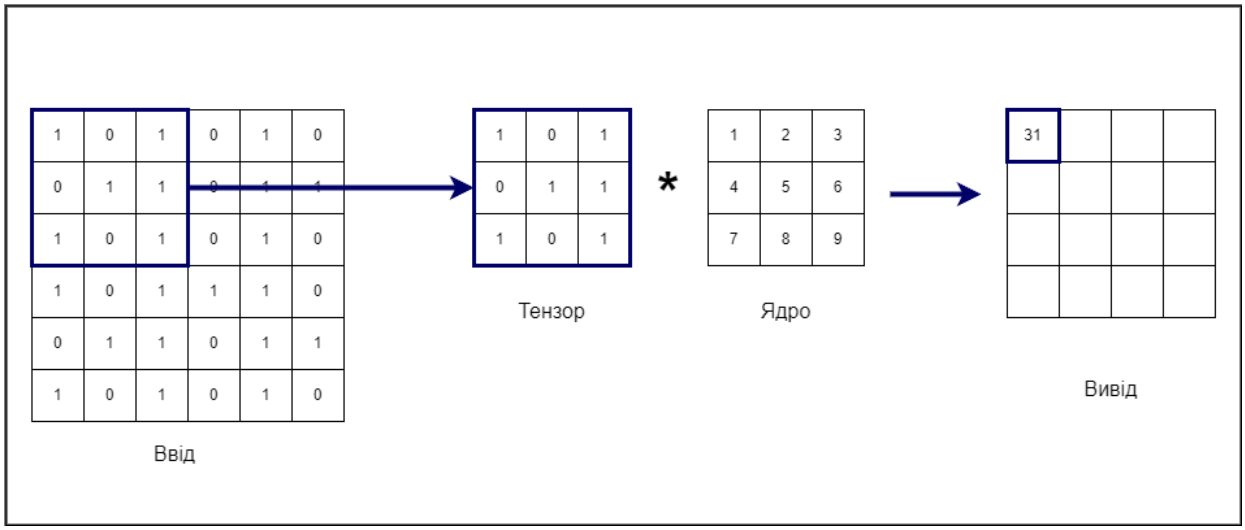

Рис. 1.4 Приклад згортки

Відстань між двома послідовними позиціями ядра називається кроком, який також визначає операцію згортки. Загальний вибір кроку - 1; Однак крок, більший за 1, іноді використовується для того, щоб досягти зниження роздільної здатності карт характеристик. [8]

До переваг CNN можна віднести те, що він: добре виявляє закономірності та функції в зображеннях, відео та аудіо сигналах; його стійкість по відношенню до інваріантності трансляції, обертання та масштабування; наскрізне навчання, а отже,відсутність необхідності ручного вилучення функцій; його можливість обробляти великі обсяги даних і досягати високої точності.

До недоліків же можна віднести: високу вартість; потреба в великій кількості обчислювальних ресурсів; необхідність в великих обсягах маркованих даних; обмежена можливість інтерпретації, через що важко зрозуміти чому навчилась мережа. [8]

Ще одним шляхом для детектування об’єктів є використання нейронної мережі YOLO («You Only Look Once»). Обробка зображень за допомогою YOLO досить проста та зрозуміла. Система змінює розмір зображення, запускає єдину згорткову мережу на зображенні та визначає порогові значення отриманих виявлень за достовірністю моделі (рис. 1.5).

Рис. 1.5 Система детектування YOLO

Для детектування об’єктів на зображенні дана мережа проводить один прохід через це зображення (через це YOLO отримала свою назву). Одна згорткова мережа одночасно прогнозує кілька обмежувальна рамка і ймовірності класу для цих рамок. YOLO тренується на повних зображеннях і безпосередньо оптимізує продуктивність детектування. Ця уніфікована модель має кілька переваг порівняно з традиційними методами детектування об’єктів [9].

По-перше YOLO є надзвичайно швидким. Автори визначають кадрове детектування як проблему регресії, через що потреба в складному конвеєрі відсутня. Ми просто запускаємо нашу нейронну мережу на новому зображенні під час тестування, щоб передбачити детектування. Крім того, YOLO досягає високої середньої точності, порівняно з іншими системами реального часу.

По-друге, на відміну від ковзаючи вікон і методів на основі пропозицій регіонів, YOLO бачить все зображення під час навчання і тренування, тому він неявно кодує контекстну інформацію про класи, а також їх зовнішній вигляд. В порівнянні з Fast R-CNN, YOLO видає менше половини фонових помилок.

По-третє, YOLO вивчає узагальнені представлення об’єктів. Після навчання на природних зображеннях та тестуванні на творах мистецтва YOLO значно перевершує найкращі методи детектування, такі як DPM і R-CNN. Оскільки YOLO дуже узагальнений, ймовірність його поломки при застосуванні до нових доменів або неочікуваних вхідних даних менша [9].

YOLO все ще відстає від найсучасніших систем детектування за точністю. Незважаючи на те, що він може швидко ідентифікувати об’єкти на зображеннях, йому важко точно локалізувати деякі об’єкти, особливо маленькі.

Дана мережа використовує характеристики всього зображення, щоб передбачити кожну обмежувальну рамку. Вона також прогнозує всі обмежувальні рамки в усіх класах для зображення одночасно. Це означає, що наша мережа загально міркує про зображення та всі об’єкти на зображенні.

Конструкція YOLO забезпечує наскрізне навчання та швидкість у реальному часі, зберігаючи високу середню точність. Система ділить вхідне зображення на сітку S × S. Якщо центр об’єкта потрапляє в комірку сітки, ця комірка відповідає за детектування цього об’єкта (приклад зображений на рис. 1.6). [9]

Рис. 1.6 Визначення сітки прогнозування об’єкта

Кожна комірка сітки передбачає обмежувальні рамки (B) та оцінки для цих рамок. Ці показники достовірності відображають, наскільки модель впевнена в тому, що рамка містить об’єкт, а також наскільки точна, на її думку, рамка, що вона передбачає. Якщо в цій комірці немає об’єкта, оцінка достовірності має бути нульовою. В іншому випадку ми хочемо, щоб оцінка достовірності дорівнювала перетину над об'єднанням (IoU) між прогнозованим блоком і основною істиною. Формулою IoU є площа перетину поділена на площу союзу (рис. 1.7) [10].

Рис. 1.7 Метрика IoU

Кожна обмежувальна рамка складається з п’яти передбачень, а саме: x та y, що представляють центр об’єкта (прогнозується відносно всього зображення в наслідок чого отримане значення буде в діапазоні [0, 1]); w (weight) та h (height) – ширина та довжина обмежувальної рамки (діапазон значення, як і в координатах центра рівний [0, 1]); прогноз достовірності представляє IOU між передбачуваним блоком і будь-яким базовим блоком істинності. Кожна комірка сітки також передбачає ймовірності умовного класу C, Pr(Classi |Object). Ці ймовірності залежать від комірки сітки, що містить об’єкт. Ми лише попередньо визначаємо набір класових ймовірностей на комірку сітки, незалежно від кількості блоків B. Під час тестування ми множимо умовні класові ймовірності та індивідуальні прогнози достовірності блоків, що дає нам оцінки достовірності для

кожного блоку. Ці показники кодують як ймовірність того, що цей клас з’явиться в полі, так і те, наскільки добре прогнозований блок відповідає об’єкту.

Рис. 1.8 демонструє як система моделює детектування як проблему регресії. Вона ділить зображення на сітку розміром S × S і для кожної комірки сітки передбачає обмежувальні рамки (B), достовірність цих рамок та ймовірності класу C. Ці передбачення кодуються як тензор S × S × (B * 5 + C). [9]

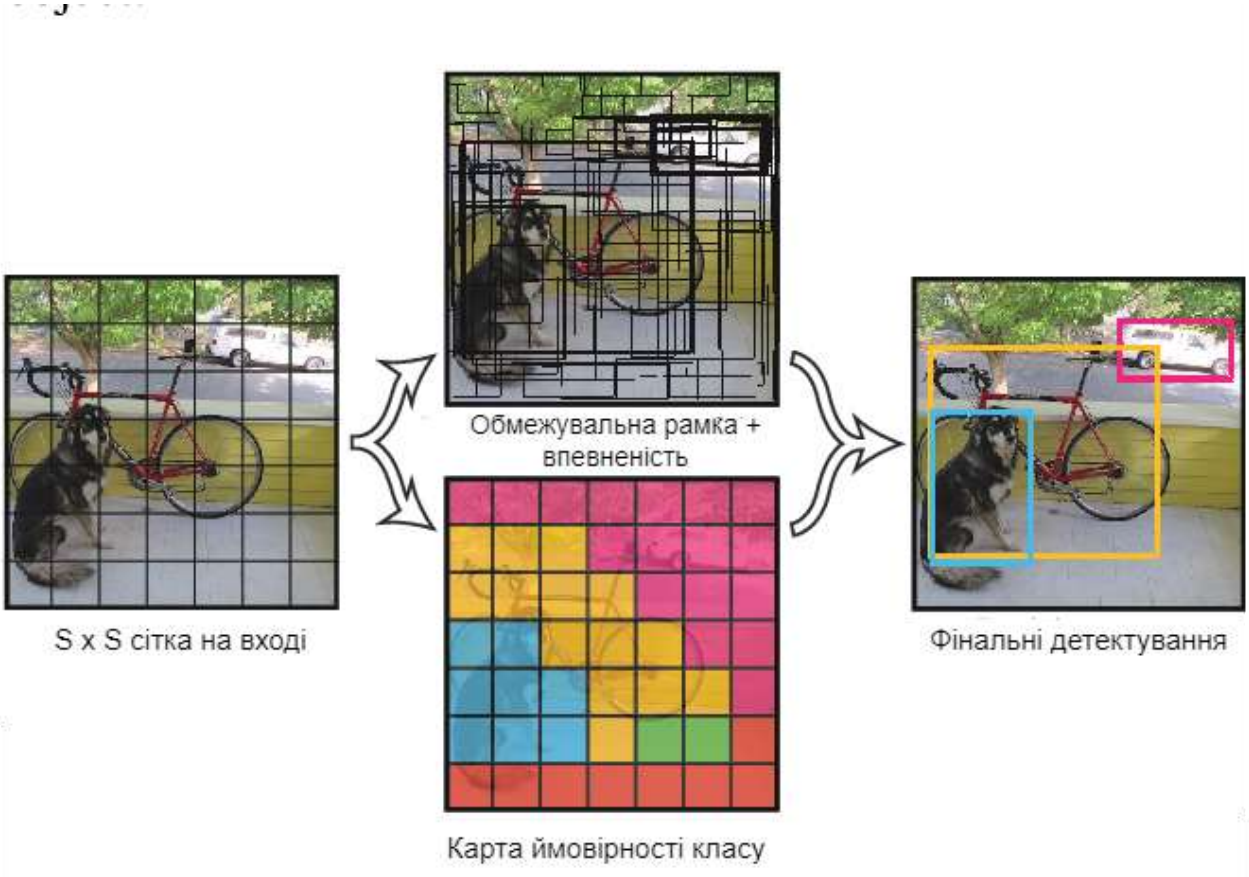

Рис. 1.8 Модель YOLO

До позитивних сторін YOLO можна віднести швидкість обробки зображень, Гарна середня точність розпізнання та можливість неявного кодування контекстної інформації. До негативних сторін даної мережі можна віднести її складність детектувати близькі об’єкти.

Також слід зауважити, що для роботи з нейронною мережею необхідна наявність GPU.

Для виконання роботи була обрана нейронна мережа YOLO, а саме її 7 версія, за рахунок швидкості детектування, наявності в неї простої архітектури та через те що вона вимагає мінімальних навчальних даних, що полегшує впровадження та адаптацію до нових завдань.

1.2. Постановка задачі

У даній роботі необхідно підрахувати кількість транспортних засобів на відео. Для цього потрібно виконати такі завдання:

1. Проаналізувати методи детектування об’єктів за допомогою штучних нейронних мереж та вибрати оптимальний.
2. Провести тренування обраної мережі та отримати ваги для подальшого детектування об’єктів на відео;
3. Зробити можливим підрахунок транспортних засобів мережею та вивести їх лічильник на відео;
4. Проаналізувати результат підрахунку кількості транспортних засобів.

РОЗДІЛ 2 ОГЛЯД ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ВИКОНАННЯ РОБОТИ.

2.1. Архітектура YOLOv7.

Модель YOLO організована у вигляді загорткової нейронної мережі. Її початкові згорткові шари витягують ознаки із зображення, тоді як повністю пов’язані шари передбачають вихідні ймовірності та координати. Архітектура даної мережі заснована на моделі класифікації зображень GoogLeNet. Вона складається з 24 згорткових шарів, за якими слідують 2 повністю зв’язані шари. Замість початкових модулів, які використовує GoogLeNet, просто використовуються шари скорочення 1×1, за якими слідують згорткові шари 3 × 3. На рис. 2.1 зображений вигляд повної мережі. [9]

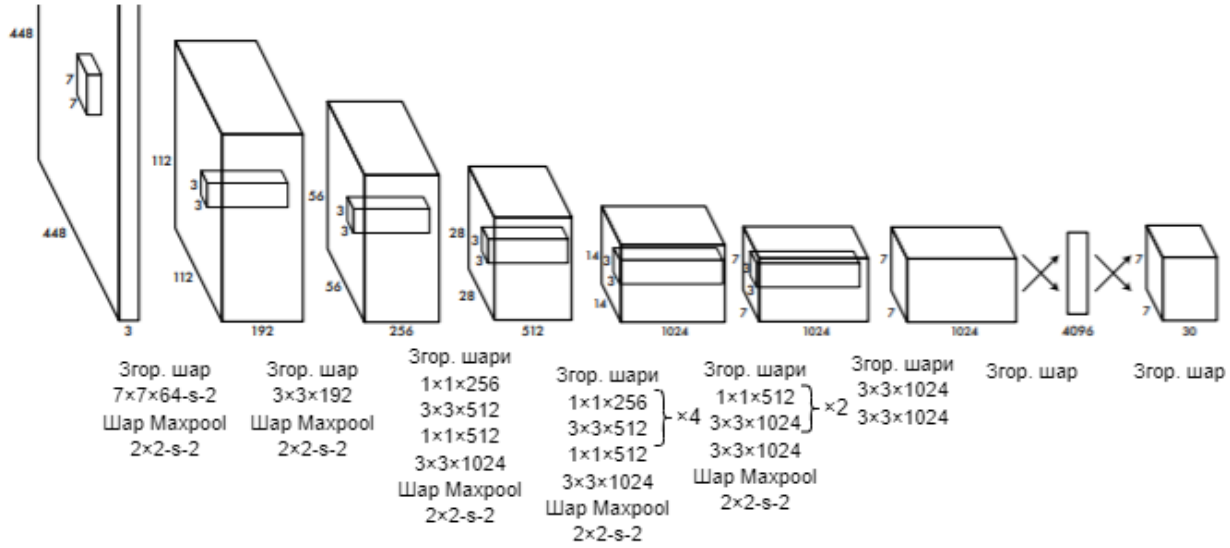

Рис. 2.1 Архітектура YOLO

Дана мережа детектування складається з 24 згорткових шарів, за якими слідують 2 повністю зв’язані шари. Чергування згорткових шарів 1×1 зменшує простір об’єктів із попередніх шарів. Остаточним результатом мережі є тензор прогнозів 7 × 7 × 30.

Fast YOLO використовує нейронну мережу з меншою кількістю згорткових шарів (9 а не 24) і меншою кількістю фільтрів у цих шарах.

Окрім розміру мережі, усі параметри навчання та тестування однакові для YOLO та Fast YOLO.

Останній рівень передбачає ймовірність класу та координати обмежувальної рамки. Ми використовуємо функцію лінійної активації для останнього шару, а всі інші шари використовують наступну випрямлену лінійну активацію [9]:

$$\Phi(x) = \begin{cases} x, \text{якщо } x > 0 \\ 0.1x, \text{в іншому випадку} \end{cases}$$

Мережа оптимізує помилку суми квадратів у вихідних даних, використовуємо похибку суми квадратів, оскільки її легко оптимізувати, однак вона не повністю відповідає нашій меті максимізації середньої точності. Він зважує помилку локалізації однаково з помилкою класифікації, що може бути не ідеальним. Крім того, на кожному зображенні багато комірок сітки не містять жодного об’єкта. Це підштовхує показники «впевненості» цих комірок до нуля, часто переважаючи градієнт комірок, які містять об’єкти. Це може призвести до нестабільності моделі, спричиняючи раннє розбіжність у навчанні. Щоб виправити це, ми збільшуємо втрати від передбачень координат обмежувальної рамки та зменшуємо втрати від достовірних передбачень для рамок, які не містять об’єктів. Для цього ми використовуємо два параметри, λ_{coord} і λ_{noobj}. Ми встановлюємо λcoord = 5 і λnoobj = .5.

Похибка суми квадратів також однаково зважує помилки у великих і малих рамках. Ця метрика помилок має відображати, що невеликі відхилення у великих рамках мають менше значення, ніж у малих рамках. Як частково вирішити цієї проблеми, прогнозується квадратний корінь із ширини та висоти обмежувальної рамки замість безпосередньо ширини та висоти. YOLO передбачає кілька обмежувальних рамок на комірку сітки. Під час навчання ми хочемо, щоб за кожен об’єкт відповідав лише один

предиктор обмежувальної рамки. Призначається один предиктор, який буде «відповідальним» за прогнозування об’єкта, на основі якого передбачення має найвищий поточний IOU з основною правдою. Це призводить до спеціалізації між предикторами обмежувальної рамки. Кожен предиктор покращує прогнозування певних розмірів, пропорцій чи класів об’єктів, покращуючи загальне запам’ятовування.

Під час навчання оптимізується наступна функція втрат, що складається з кількох частин:

$$\lambda_{\text{coord}} \sum_{i=0}^{S2} \sum_{j=0}^{B} 1_{ij}^{obj}$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w_i}} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h_i}} \right)^2 \right]$$

$$+ \sum_{i=0}^{S2} \sum_{j=0}^{B} 1_{ij}^{obj} \left(\sqrt{C_i} - \sqrt{\hat{C}_i} \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S2} \sum_{j=0}^{B} 1_{ij}^{noobj} \left(\sqrt{C_i} - \sqrt{\hat{C}_i} \right)^2$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S2} 1_{i}^{obj} \sum_{c \in classes} \left(p\left|i(c) - \hat{p}_i(c) \right.\right)^2$$

де 1_i^{obj} означає, що об’єкт з’являється в комірці i, а 1_{ij}^{obj} означає, що j-й обмежувальна рамка у комірці і «відповідає» за це передбачення.

Зауважте, що функція втрат лише штрафує за помилку класифікації, якщо об’єкт присутній у цій комірці сітки. Він також штрафує лише помилку координат обмежувальної рамки, якщо цей предиктор «відповідає» за ground truth рамку (тобто має найвищий IoU з усіх предикторів у цій комірці сітки). [9]

Так само, як під час навчання, прогнозування детектування для тестового зображення вимагає лише однієї оцінки мережі. YOLO надзвичайно швидкий під час тестування, оскільки для нього потрібна лише одна оцінка мережі, на відміну від методів на основі класифікатора. Конструкція сітки забезпечує просторове розмаїття передбачень обмежувальної рамки. В більшості випадків, те, до якої комірки потрапляє

об'єкт, є зрозумілим, і мережа передбачає лише одну рамку для кожного об'єкта. Однак деякі великі об’єкти або об'єкти поблизу межі кількох комірок можуть бути локалізовані кількома комірками. Для виправлення цих чисельних помилок можна використовувати максимальне придушення. Хоча це не критично для продуктивності але не максимальне придушення додає 2-3% у mAP. [9]

YOLO накладає сильні просторові обмеження на передбачення обмежувальної рамки, оскільки кожна комірка сітки передбачає лише дві рамки та може мати лише один клас. Це просторове обмеження обмежує кількість найближчих об’єктів, які дана модель може передбачити. YOLO бореться з дрібними об’єктами, які з’являються групами. Ця модель вчиться передбачати обмежувальні рамки з даних, їй важко узагальнити об’єкти з новими або незвичними пропорціями чи конфігураціями. Також вона використовує відносно грубі функції для прогнозування обмежувальних рамок, оскільки її архітектура має кілька рівнів зменшення дискретизації вхідного зображення. Нарешті, поки ми тренуємося на функції втрат, яка наближає продуктивність детектування, наша функція втрат розглядає помилки однаково в малих обмежувальних рамках і великих обмежувальних рамках. Невелика помилка у великій рамці, як правило, нешкідлива, але невелика помилка в маленькій рамці має набагато більший вплив на IoU. Нашим основним джерелом помилок є неправильна локалізація. [9]

Розглядаючи безпосередньо сьому версію YOLO в статті [11] було запропоновано використання Extended-ELAN (E-ELAN), на основі ELAN. Незалежно від довжини шляху градієнта та кількості обчислювальних блоків у великомасштабній ELAN, вона досягла стабільного стану. Якщо необмежену кількість обчислювальних блоків укладають у стек, цей стабільний стан може бути зруйновано, а коефіцієнт використання параметрів зменшиться. Запропонований E-ELAN використовує потужність розширення, перемішування та злиття для досягнення

здатності постійно покращувати здатність до навчання мережі без руйнування вихідного шляху градієнта. З точки зору архітектури, E-ELAN змінює лише архітектуру обчислювального блоку, тоді як архітектура перехідного рівня повністю не змінюється. Стратегія розробників полягає у використанні групової згортки для розширення каналу та потужності обчислювальних блоків. Застосовується той самий груповий параметр і множник каналу до всіх обчислювальних блоків обчислювального рівня. Потім карту функцій, розраховану кожним обчислювальним блоком, буде перемішано в g груп відповідно до встановленого параметра групи g, а потім об'єднуємо їх разом. У цей час кількість каналів у кожній групі карти функцій буде такою ж, як кількість каналів у вихідній архітектурі. Нарешті, ми додаємо g груп карт функцій для виконання кардинальності злиття. На додаток до підтримки оригінальної архітектури дизайну ELAN, E-ELAN також може керувати різними групами обчислювальних блоків для вивчення більш різноманітних функцій (зображення архітектур на рис. 2.2). [11]

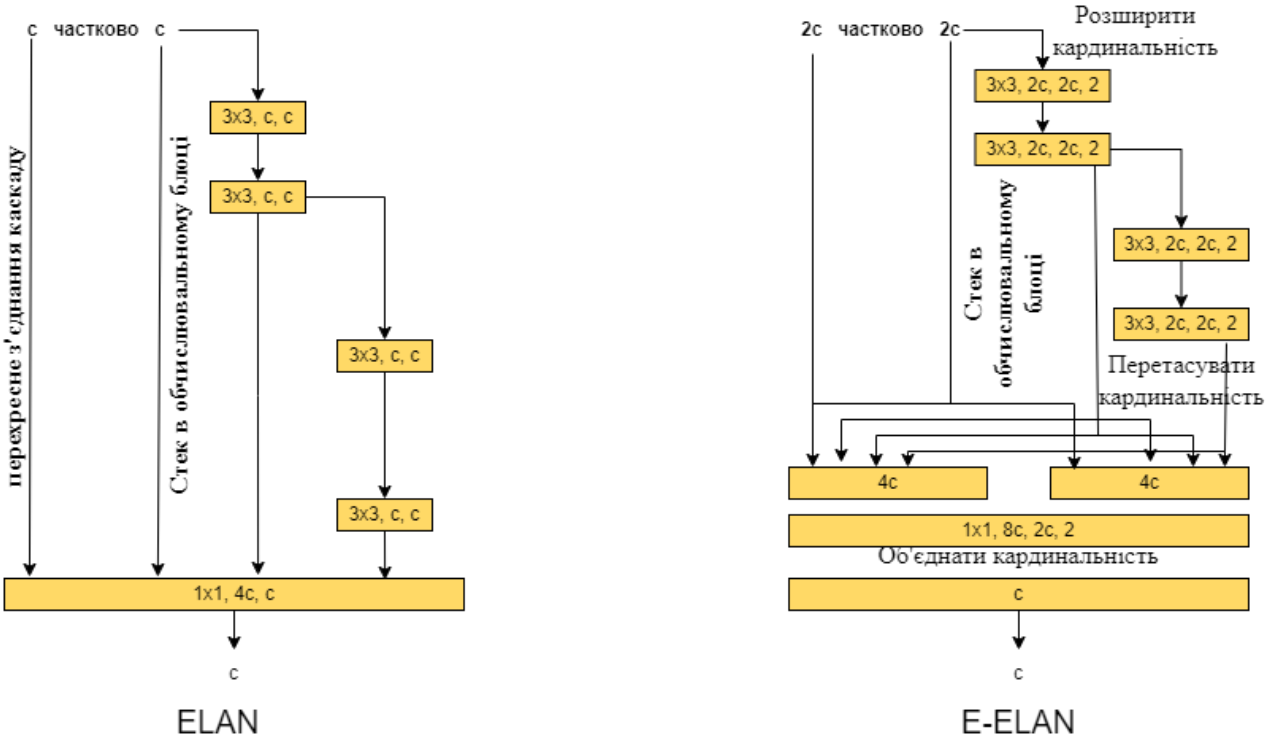

Рис. 2.2 Розширені ефективні рівні агрегаційних мереж ELANта E-ELAN

Запропонований розширений ELAN (E-ELAN) взагалі не змінює шлях передачі градієнта вихідної архітектури, але використовує групову згортку для збільшення кардинальності доданих функцій і комбінування функцій різних груп у спосіб перетасування та злиття кардинальності. Цей спосіб роботи може покращити функції, отримані за допомогою різних карт функцій, і покращити використання параметрів і обчислень. [11]

Основною метою масштабування моделі є коригування деяких атрибутів моделі та створення моделей різних масштабів для задоволення потреб у різних швидкостях висновку.

Наведені вище методи в основному використовуються в таких архітектурах, як PlainNet або ResNet. Коли ці архітектури виконують масштабування вгору або вниз, ступінь входу та спаду кожного рівня не змінюватимуться, тож ми можемо незалежно аналізувати вплив кожного фактора масштабування на кількість параметрів і обчислення. Однак, якщо ці методи застосувати до архітектури на основі конкатенації, ми виявимо, що коли масштабування вгору або вниз виконується на глибині, ступінь рівня трансляції, який знаходиться відразу після обчислювального блоку на основі конкатенації, зменшиться або збільшення, як показано на рис. 2.3. [11]

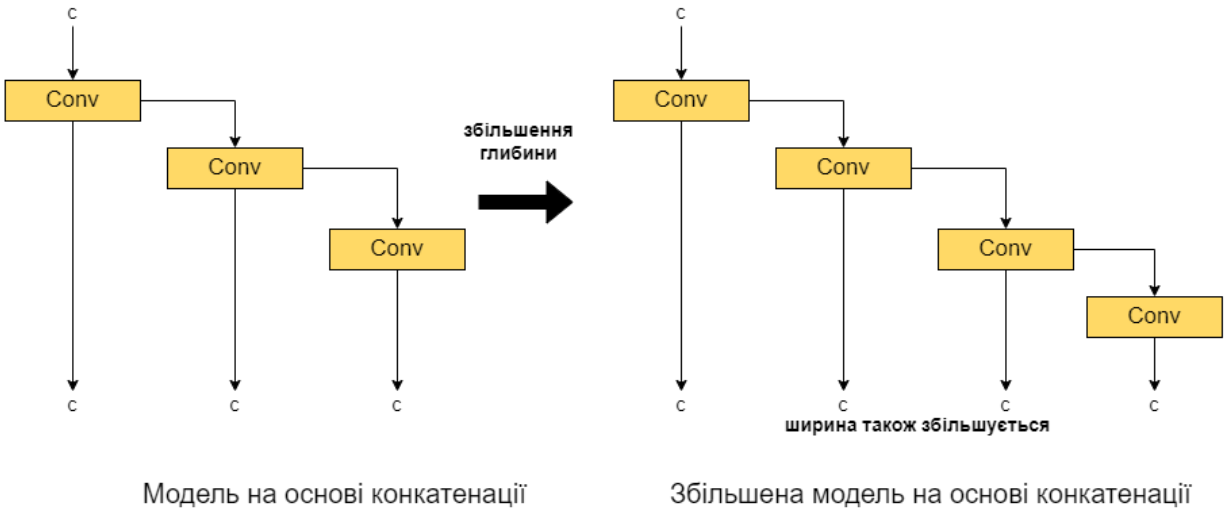

Рис. 2.3 Масштабування глибини на моделях на основі конкатенації

З наведеного вище явища випливає, що для нас є неможливим аналіз різних коефіцієнтів масштабування окремо для моделі на основі конкатенації, натомість вони повинні розглядатися разом. Тому був запропонований відповідний метод масштабування складеної моделі для моделі на основі конкатенації. Коли проводиться масштабування коефіцієнта глибини обчислювального блоку, також необхідно обчислити зміну вихідного каналу цього блоку. Потім виконується масштабування коефіцієнта ширини з однаковою кількістю змін на перехідних шарах (результат зображено на рис. 2.4). Запропонований складений метод масштабування може підтримувати властивості, які модель мала при початковому проектуванні, і підтримувати оптимальну структуру. [11]

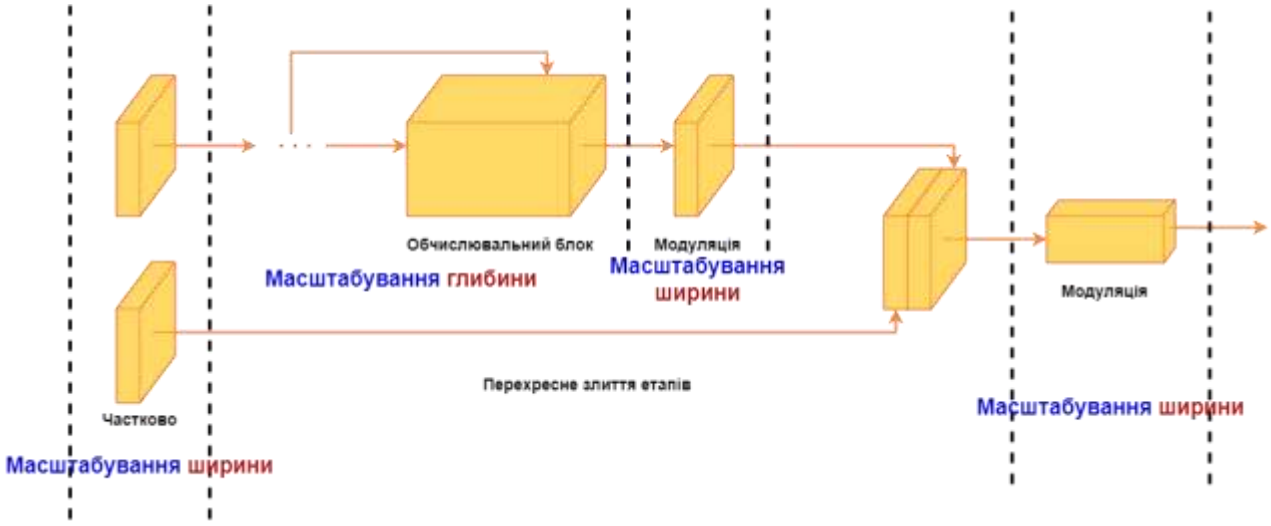

Рис. 2.4 Масштабування глибини на моделях на основі конкатенації

Також автори даної мережі зосередили свою увагу на деяких оптимізованих модулях і методах оптимізації, які можуть збільшити вартість навчання для підвищення точності детектування об’єктів, але без збільшення вартості логічного висновку. Запропоновані модулі та методи оптимізації були названі навчальними bag of freebies (дослівний переклад – пакет халяви). [11]

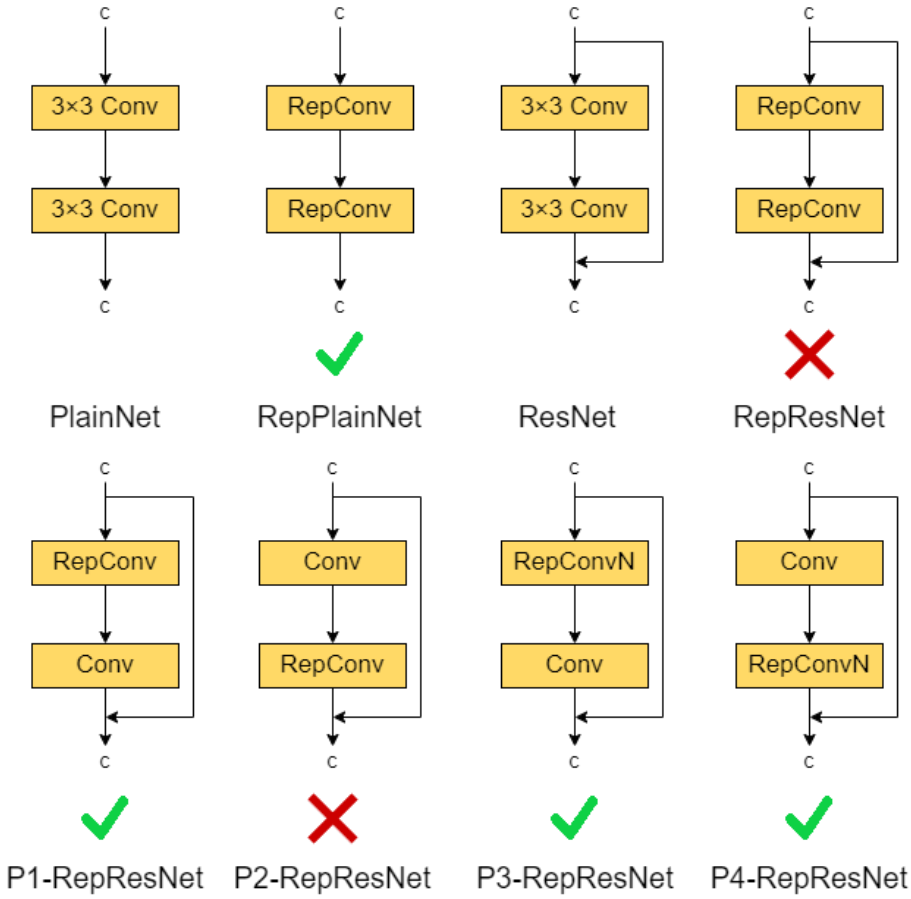

Рис. 2.5 Запланована повторно параметризована модель

Ці, так би мовити, халяви - одні з трюків, які автори використовували на тренуваннях, але оригінальні концепції не були ними запропоновані.

Одним із прикладів bag of freebies є використання RepConv без підключення ідентичності (RepConvN) для проектування архітектури запланованої повторно параметризованої згортки за для того щоб, коли згортковий шар із залишком або конкатенацією замінюється повторно параметризованою згорткою, не було тотожного зв’язку. На рисунку 2.5 наведено приклад розробленої авторами "запланованої повторно параметризованої згортки", що використовується в PlainNet та ResNet. [11]

2.2. Додаткові технології та програмне забезпечення для реалізації проекту.

Для виконання завдання буде використовуватися відео з YouTube, з якого будуть створюватися дані для навчання.

Для початку постає завдання підготовки даних для тренування мережі, а саме поділу відео на фото та їх анотації. Є багато шляхів виконати це завдання, наприклад використання веб-ресурсів, що надають необхідні для цього функціонал, додатків (частіше лише поділу відео) або використання певних бібліотек для мов програмування. Для цього буде використовуватись Roboflow.

Roboflow – це інструмент, який полегшує завдання комп’ютерного зору в області глибокого навчання. Він дає змогу розробникам створювати програми для комп’ютерного бачення, незалежно від їхніх навичок чи досвіду. Він підтримує детектування об'єктів і моделі класифікації. До необхідного нам функціоналу відноситься: анотація набору даних; попередня обробка набору даних; експорт набору даних.

Далі буде проводитись навчання та використовуючи отримані ваги провести детектування відео. Для цього використовується нейронна мережа YOLOv7, а як середовище розробки буде взятий Google Colaboratory (надалі Colab).

Colab – це хмарне середовище, яке дозволяє будь-кому, хто має доступ до інтернету, експериментувати з машинним навчанням і кодуванням для штучного інтелекту. Він надає можливість писати та виконувати код Python, ділитися своїм кодом і редагувати його одночасно з іншими людьми, а також документувати все, об’єднавши це в єдиний блокнот. Важливим плюсом Colab є можливість надання ним доступу до графічних процесорів без оплати.

Для отримання вихідного коду YOLOv7 використовується GitHub, з якого він буде копіюватися в Colab. GitHub - це веб-сайт і хмарний сервіс, який допомагає розробникам зберігати і керувати своїм кодом, а також відстежувати і контролювати зміни в своєму коді.

РОЗДІЛ 3 ПІДГОТОВКА ДАНИХ ТА ВИКОНАННЯ ЗАВДАННЯ

3.1. Підготовка даних та тренування нейронної мережі YOLOv7.

Як вже згадувалося раніше, першим завданням при виконанні даної роботи є збір даних та, з їх використанням, проведення тренування мережі і, в результаті, отримання ваг для можливості детектування об’єктів.

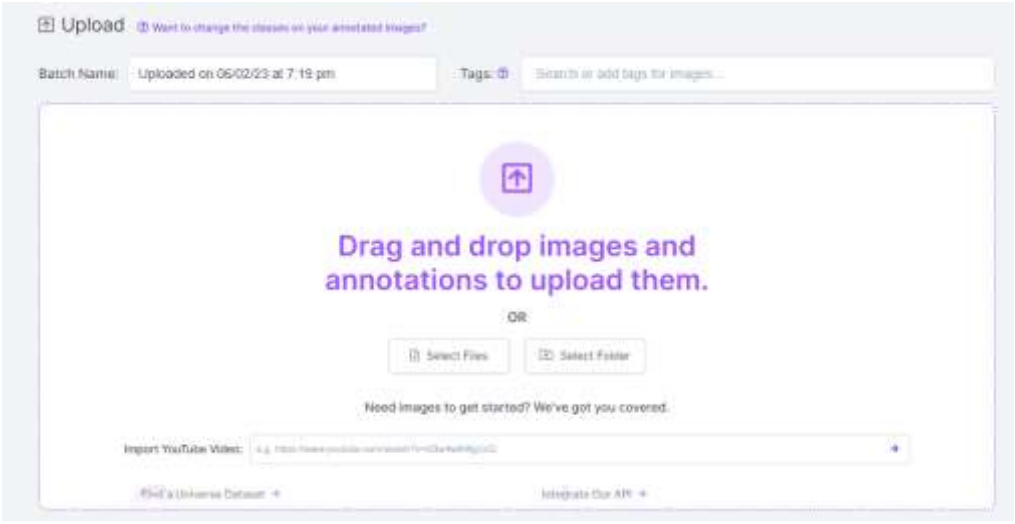

Рис. 3.1 Вікно для завантаження зображень

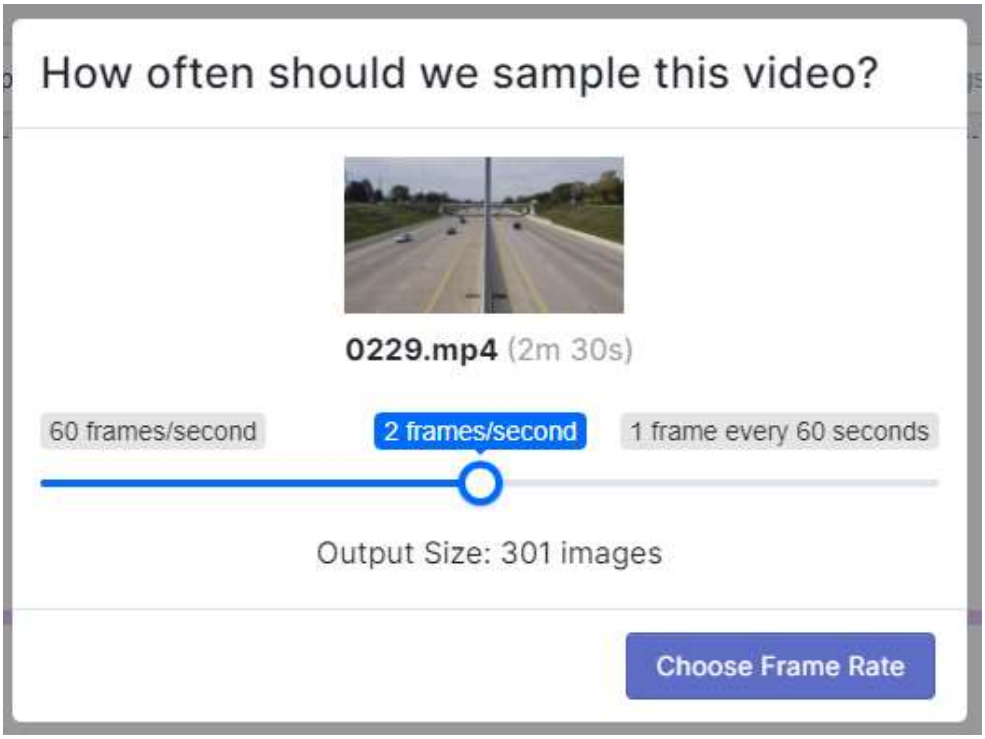

Рис. 3.2 Вікно вибору кількості кадрів на секунду при поділі відео

Відео для проведення тренування та подальшого детектування було взяте з Youtube (посилання на відео: https://www.youtube.com/watch?v=KBsqQez-O4w&ab_channel=NickMartinez). Використовуючи його отримуємо набір даних (зображення для проведення навчання). Досить зручним варіантом для виконання даного завдання є використання функціоналу Roboflow. При створенні проекту (рис. 3.1) надається можливість завантажити відео та обрати кількість кадрів на секунду при його поділу (рис. 3.2).

Після створення набору зображень переходимо до їх анотування. Для цього після завантаження і підтвердження додавання зображень переходимо в розділ анотування та натискаємо на не анотовані зображення (рис. 3.3). Для навчання мережі рекомендується брати для навчання 1000 зображень з 10000 анотованих об’єктів, але в даній роботі буде достатньо лише 190 зображень через роботу лише з одним відео з якого й будуть створюватися дані для навчання, а також визначення лише одного класу при тренуванні.

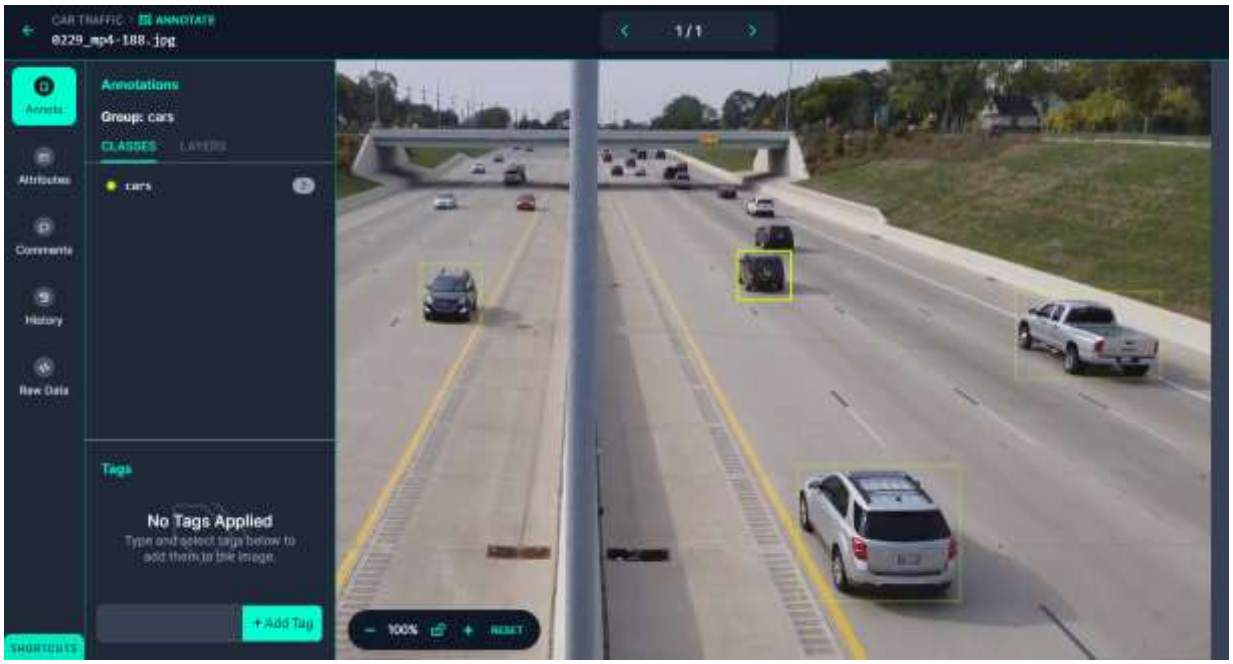

Рис. 3.3 Вікно анотації зображень

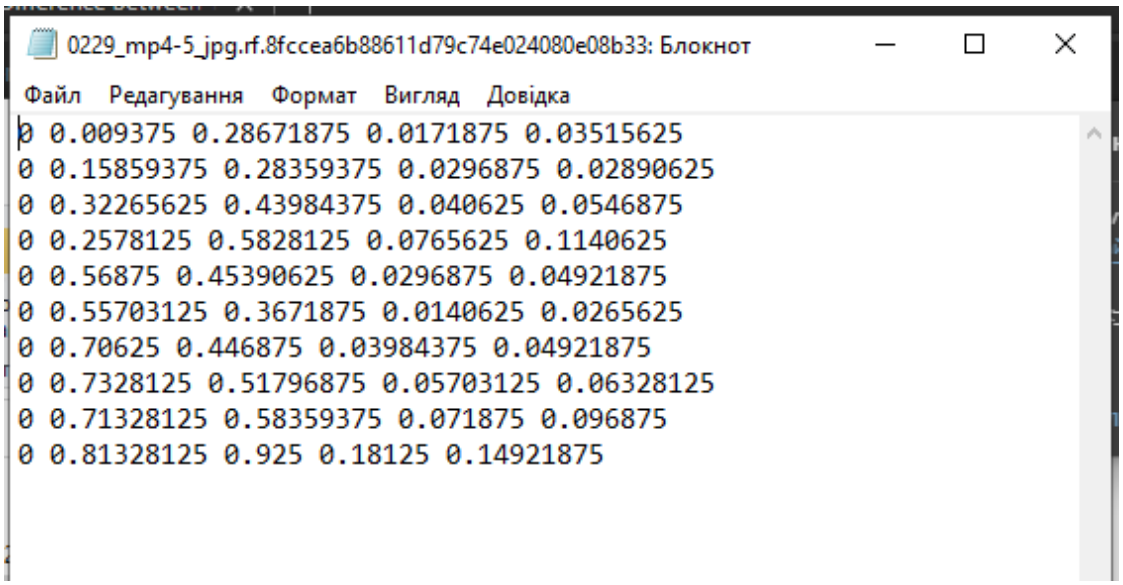

Рис. 3.4 Вміст файлу label.

Далі створюємо набір даних з оброблених зображень. Для цього необхідно буде обрати пропорції поділу між папкою valid та train. До кожного зображення створюється файл з описом рамок (рис. 3.4). Рекомендовані пропорції поділу зводиться до 80% для папки train та 20% для папки valid.

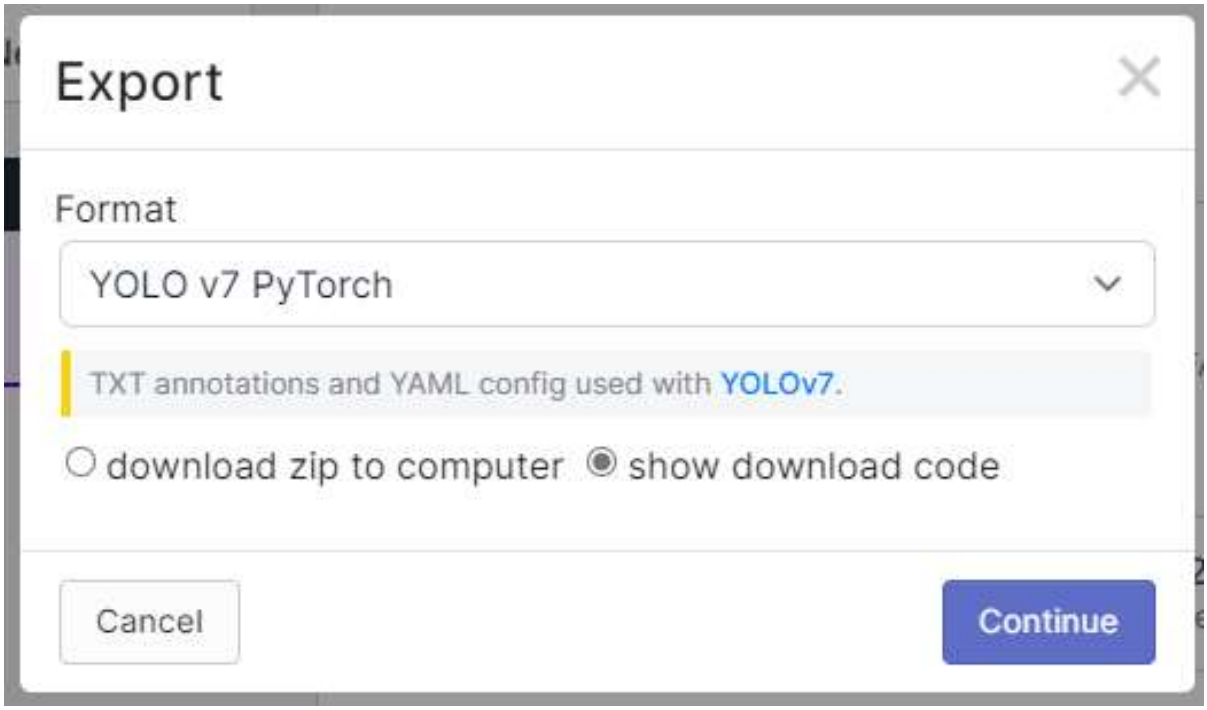

Рис. 3.5 Вікно для експорту даних

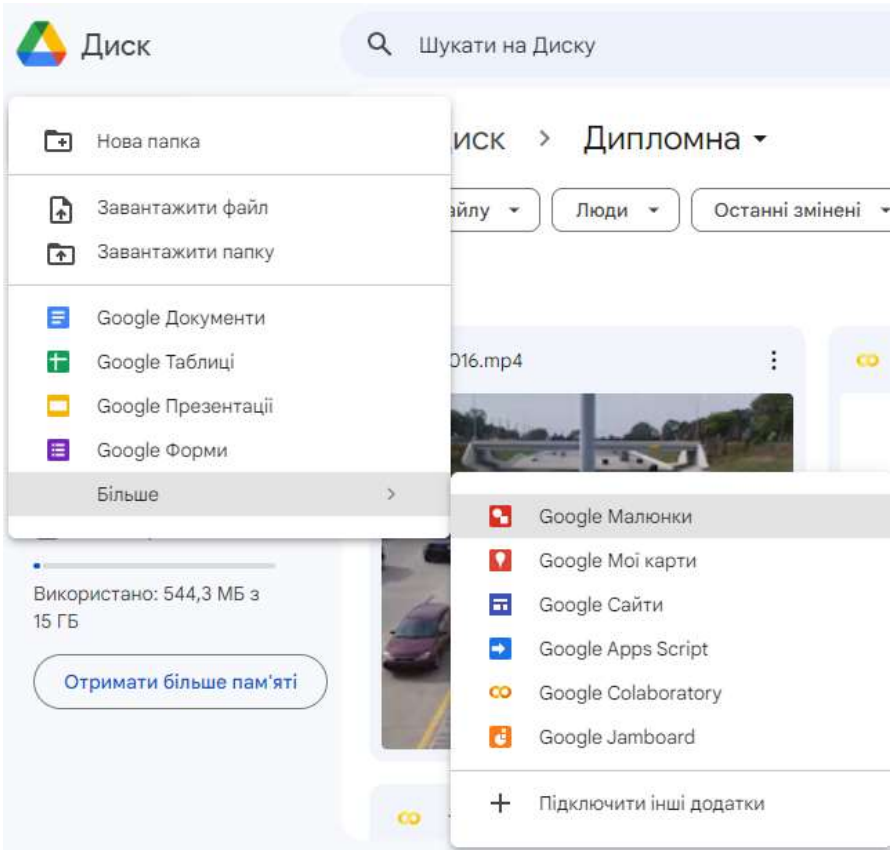

Рис. 3.6 Приклад створення файлу Colab в Google Drive

В даній роботі, дані були імпортовані в проект за допомогою коду завантаження (рис 3.5).

Наступним кроком постає завдання проведення тренування за допомогою YOLOv7. Для цього створюємо файл Google Colaboratory в GoogleDrive (рис. 3.6) та змінюємо апаратне прискорення на GPU (graphics processing unit).

Далі імпортуємо репозиторій YOLOv7 з GitHub (на рис. 3.7 зображена рядок коду для виконання даного завдання). Після цього також проводиться імпорт даних, які використовуються для навчання мережі а також ваги yolov7 на основі яких буде проводитись навчання (посилання на завантаження можна знайти в репозиторії YOLOv7 на GitHub).

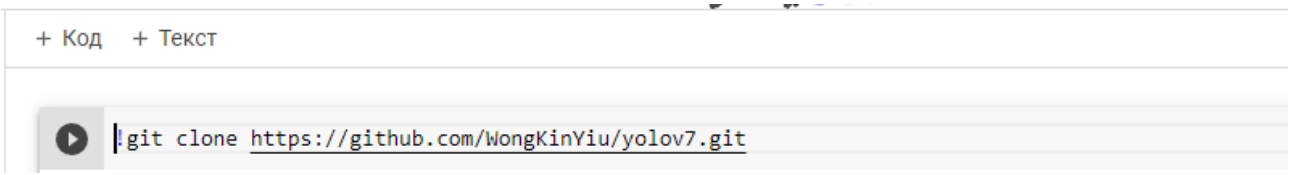

Рис. 3.8 Рядок коду для імпорту репозиторію YOLOv7

Далі, для зручності подальшої роботи з файлами, заходимо в папку yolov7 (рис. 3.9). Потім встановлюємо необхідні бібліотеки за допомогою, наявного в папці yolov7, файлу requirements.txt ввівши код зображений на рис. 3.10.

```
[ ]  %cd yolov7/
     !ls
```

Рис. 3.9 Рядок коду для переходу в папку yolov7

```
!pip install -r requirements.txt
```

Рис. 3.10 Рядок коду для встановлення додаткових

Далі необхідно внести зміни до файлів для відповідності тренуванню, а саме: до вмісту файлу coco.yaml (назву змінюємо coco_custome.yaml), місцезнаходження якого yolov7/data/, вносяться зміни, зображені на рис. 3.11; в файлі обраної моделі yolov7.yaml (шлях до файлу - /yolov7/cfg/training/) вноситься зміна до кількості класів (зображено на рис. 3.12).

```
1 train: /content/yolov7/car-traffic-1/train
2 val: /content/yolov7/car-traffic-1/valid
3
4 # number of classes
5 nc: 1
6
7 # class names
8 names: [ 'car']
```

Рис. 3.11 Фінальний вигляд файлу coco.yaml

```
1 # parameters
2 nc: 1  # number of classes
3 depth_multiple: 1.0  # model depth multiple
4 width_multiple: 1.0  # layer channel multiple
5
6 # anchors
7 anchors:
```

Рис. 3.12 Фінальний вигляд файлу yolov7.yaml

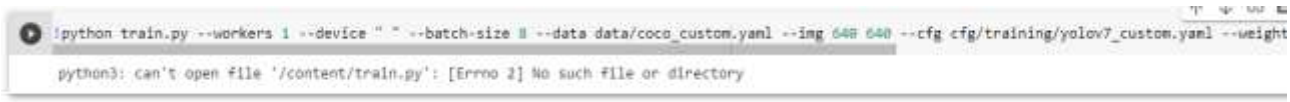

Рис. 3.13 Рядок коду для запуску тренування

Тепер можна починати тренування. Для цього вводимо рядок !python train.py та в тому ж рядку визначаємо необхідні змінні, що передаються в цей файл (рис. 3.13), одними з яких є: img – роздільна здатність наданих зображень; workers – скільки під процесів розпаралелювати під час навчання; batch_size – визначає кількість зразків оброблених до створення оновленої моделі; epochs – кількість епох для тренування, а саме це значення відповідає загальній кількості проходів, яке модельне навчання пройде по всьому набору даних; hyp – файл YAMP, який описує вибір гіперпараметрів для навчання проекту; weights – попередньо натренеровані ваги, з яких ми хочемо знову почати навчання(у випадку відсутності ваг ставимо значення «’ ’»); cfg – файл конфігурації архітектури моделі; data – файл DATA YAMP містить інформацію про набір даних, щоб допомогти керувати YOLO. [2]

Особливу увагу при описі тренувань хочеться звернути на характеристику epochs. При проведенні першого тренування було вирішено встановити значення цього показника на 50 епох. Результат детектування об’єктів на відео за допомогою отриманих ваг (за результат тренування я беру найкраще значення за всі епохи) був невтішним, а саме основним недоліком було те, що розпізнання вантажних автомобілів зазвичай було помилковим. Далі кількість епох була збільшена до 100 та провів нове навчання, ваги з якого краще визначали транспортні засоби але також був присутній недолік, що кидався в очі – це розпізнання двох автомобілів, один з яких закриває значну частину іншого, як одного об’єкта в певних випадках, що могло призвести до помилок, а саме меншого числа автомобілів, що пересікли лінію, від фактичного. При наступному тренуванні було проведено 200 епох. Визначення об’єктів

було більш менш вірним, але була присутня недостатня точність в певні моменти відео.

Для остаточної надійності в кінці було проведено тренування на 500 епох що дало задовільний результат точності детектування транспортних засобів.

Також потрібно звернути увагу на формат виведення результатів тренування після кожної епохи(Рис. 3.14 – 3.17).

```
Epoch   gpu_mem       box       obj       cls     total    labels  img_size
0/199     1.69G   0.09001   0.02417         0    0.1142       102       640: 100% 18/18 [00:25<00:00,  1.41s/it]
          Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 3/3 [00:08<00:00,  2.78s/it]
            all          47         421     0.0205     0.0095    0.000491    8.15e-05

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
1/199     6.53G   0.08351   0.02159         0    0.1051       103       640: 100% 18/18 [00:07<00:00,  2.26it/s]
          Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 3/3 [00:02<00:00,  1.22it/s]
            all          47         421     0.0296      0.116     0.00658     0.00133

Epoch   gpu_mem       box       obj       cls     total    labels  img_size
2/199     8.16G   0.07593   0.02107         0   0.09701        70       640: 100% 18/18 [00:08<00:00,  2.21it/s]
          Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 3/3 [00:01<00:00,  2.03it/s]
            all          47         421       0.29      0.233      0.177      0.0455
```

Рис. 3.14 Приклад результатів тренування на епохах 0 - 2

```
 Epoch   gpu_mem       box       obj       cls     total    labels  img_size
53/199     8.18G   0.03234   0.01267         0   0.04501        92       640: 100% 18/18 [00:08<00:00,  2.23it/s]
          Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 3/3 [00:00<00:00,  3.11it/s]
            all          47         421       0.747      0.686      0.781      0.416

 Epoch   gpu_mem       box       obj       cls     total    labels  img_size
54/199     8.18G    0.0535   0.01133         0   0.06484        77       640: 100% 18/18 [00:08<00:00,  2.24it/s]
          Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 3/3 [00:01<00:00,  2.93it/s]
            all          47         421       0.899      0.736      0.861      0.523

 Epoch   gpu_mem       box       obj       cls     total    labels  img_size
55/199     8.18G   0.04001   0.01276         0   0.05278       125       640: 100% 18/18 [00:07<00:00,  2.42it/s]
          Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 3/3 [00:00<00:00,  3.16it/s]
            all          47         421       0.871      0.671      0.822       0.47
```

Рис. 3.15 Приклад результатів тренування на епохах 53 - 55

```
  Epoch   gpu_mem       box       obj       cls     total    labels  img_size
192/199     8.18G   0.02165  0.009287         0   0.03093        63       640: 100% 18/18 [00:07<00:00,  2.26it/s]
          Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 3/3 [00:00<00:00,  3.13it/s]
            all          47         421       0.961      0.941      0.988      0.763

  Epoch   gpu_mem       box       obj       cls     total    labels  img_size
193/199     8.18G   0.02177  0.008533         0    0.0303        71       640: 100% 18/18 [00:07<00:00,  2.37it/s]
          Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 3/3 [00:01<00:00,  2.30it/s]
            all          47         421       0.944      0.957      0.987      0.767

  Epoch   gpu_mem       box       obj       cls     total    labels  img_size
194/199     8.18G   0.01954  0.009839         0   0.02938        97       640: 100% 18/18 [00:07<00:00,  2.44it/s]
          Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 3/3 [00:00<00:00,  3.01it/s]
            all          47         421       0.963      0.938      0.988      0.765
```

Рис. 3.16 Приклад результатів тренування на епохах 192 - 194

```
      Epoch    gpu_mem       box       obj       cls     total    labels  img_size
    497/499     8.48G  0.008248  0.005923         0   0.01417        94       640: 100% 18/18 [00:08<00:00,  2.05it/s]
               Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 3/3 [00:01<00:00,  2.27it/s]
                 all          47         421      0.942      0.969       0.984       0.777

      Epoch    gpu_mem       box       obj       cls     total    labels  img_size
    498/499     8.48G  0.008494  0.006134         0   0.01463       118       640: 100% 18/18 [00:09<00:00,  1.90it/s]
               Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 3/3 [00:01<00:00,  2.83it/s]
                 all          47         421      0.961      0.948       0.981       0.774

      Epoch    gpu_mem       box       obj       cls     total    labels  img_size
    499/499     8.48G  0.008334  0.006115         0   0.01445        89       640: 100% 18/18 [00:09<00:00,  1.97it/s]
               Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 3/3 [00:02<00:00,  1.25it/s]
                 all          47         421      0.962       0.95        0.98       0.771
```

Рис. 3.17 Приклад результатів тренування на епохах 497 - 499

В цих результатах P (Precios - точність) – визначає наскільки точно модель класифікує об’єкти під час навчання; R (Recall - повнота) – визначає наскілки ефективно модель виявляє всі наявні об’єкти в зображенні; mAP (mean Average precision – середня точність) – вимірює якість детектування об’єктів і обчислюється як середнє значення показника точності для різних значень порогу впевненості моделі в присутності об’єкту. На наданих вище прикладах результатів видно що mAP є зі значенням 0.5 та 0.5:0.95. різниця між ними полягає в тому, що в першому випадку визначається середня точність для порогу 0.5, а в другому розглядається ширший діапазон порогів, тобто починаючи з 0.5 і збільшуючи його (з кроком 0.05) до 0.95, для кожного порогу обчислюється точність детектування.

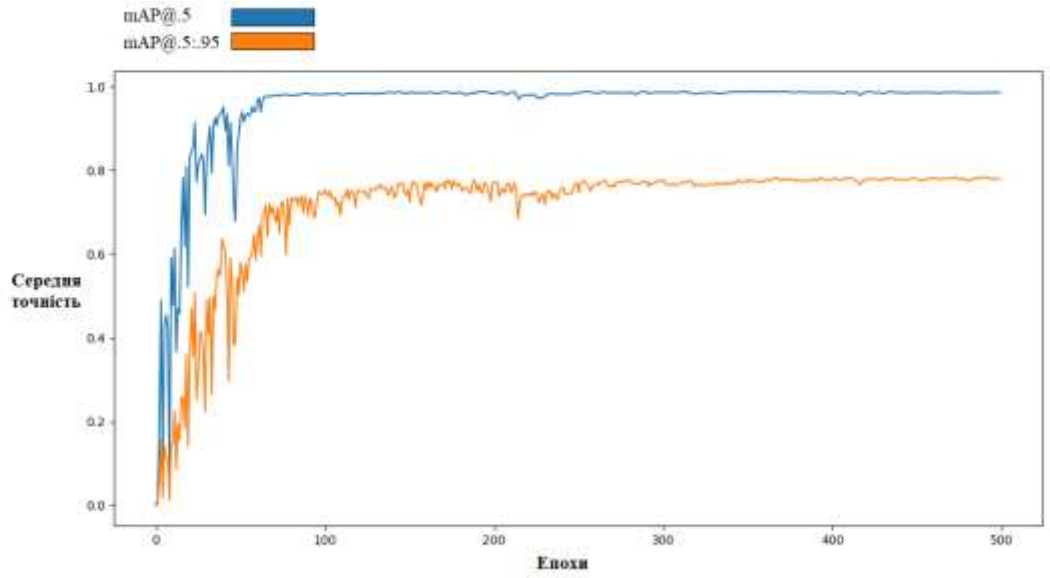

Рис. 3.18 Графік зміни aMP при різних епохах

Потім обчислюється середнє значення точності детектування для всіх порогів у діапазоні 0.5-0.95. На рис. 3.18 зображено графік зміни mAP даних в кожній епосі, що демонструє вплив кількості епох на дані значення.

3.2. Детектування та підрахунок об’єктів у відео потоці.

Після проведення тренування та отримання ваг для детектування об’єктів постає завдання відстеження об’єктів та їхнього підрахунку при пересіченні лінії. Для відстеження об’єктів використовується посилання на попереднє фото з використанням Евклідової відстані між центром об’єктів.

Евклідова відстань – це “звичайна” прямолінійна відстань між двома точками евклідового простору.

Формула Евклідової відстані [3]:

$$\sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

Під час детектування об’єктів відбувається прохід через всі визначенні об’єкти. Під час цього продовжується пошук найближчого об’єкта на минулому фото до теперішнього об’єкта та, при умові що відстань між ними не перевищує певного значення, теперішній об’єкт записується в масив з індексом найближчого. Саме для підрахунку об’єктів їм надається значення в залежності від розташування відносно встановленої лінії. При відстеженні об’єктів на наступних фото це значення унаслідується. При перетині цієї лінії значення об’єкта змінюється на нейтральне а лічильник збільшується на 1.

Для виконання даного алгоритму були внесені зміни в код в файлі detect.py. Вивід лінії та лічильників на екран здійснюється за допомогою бібліотеки opencv.

Для виконання детектування в новому файлі Colab клонується репозиторій YOLOv7, проводиться вхід в папку /yolov7/ та завантажуються ваги, отримані з навчання, і відео, на якому буде проводитися детектування.

Далі запускається файл detect.py кодом, зображеним на рис. 3.19. В результаті отримуємо результат (рис. 3.20).

```
!python detect.py --weights /content/epoch_500.pt --conf 0.7 --img-size 640 --source /content/016.mp4
```

Рис. 3.19 Рядок коду для запуску детектування відео

Рис. 3.20 Результат детектування

Рис. 3.20 демонструє вдале виконання завдання, а саме детектування транспортних засобів, їх підрахунок, та виведення лічильника на екран. При всьому цьому, середня точність є висока, не враховуючи зовсім малі об’єкти (точність коливається від 0.7 до 0.95 враховуючи що при детектуванні був встановлений поріг 0.7). Отриманого рівня детектування об’єктів достатньо для виконання завдання.

ВИСНОВОК

При виконанні даної роботи було проведено дослідження систем детектування об’єктів. Спочатку було проведено дослідження каскадів Хаара але через певні недоліки, а саме через те що вони є недуже точними а також залежність часу детектування від точності моделі.

Наступною була досліджена мережа CNN, недоліками якої виявилися необхідність великої кількості обчислювальних ресурсів, високу вартість організації мережі та обмежена можливість інтерпретації, через що важко зрозуміти чому навчилась мережа.

В кінці було досліджено YOLOv7. До мінусів даної мережі відносяться складність розпізнання малих об’єктів, складність при розпізнанні об’єктів на малих сценаріях та потреба обчислювальних ресурсів.

В результаті була обрана мережа YOLOv7 недоліки якої не є критичними для виконання даного завдання, а також враховуючи її переваги, а саме швидкодію та високу середню точність.

Було проведена підготовка даних до тренування, а саме їх анотація з використанням Roboflow. Після цього було проведено тренування обраної мережі, в результаті якого були отримані ваги для подальшого детектування транспортних засобів (робота виконувалась в Google Colaboratory). Для тренування було обрано використання 500 епох. Також було наведено приклад залежності середньої точності від кількості епох.

Далі були введені зміни в код запуску нейронної мережі (detect.py) для відстеження транспортних засобів, їх підрахунку та виведення лічильника підрахунку на екран. В результаті, змінена мережа виконує поставлене завдання.

Отримана мережа може використовуватися в різних ситуаціях, одними з яких є підрахунок транспорту на парковкі для транспортних засобів, на дорозі для зручнішого керування рухом та при дослідженні руху транспортних засобів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Chinmoy Borah - Evolution of Object Detection. URL: https://medium.com/analytics-vidhya/evolution-of-object-detection-582259d2aa9b (дата звернення: 1.11.2020);
2. James Skelton - Step-by-step instructions for training YOLOv7 on a Custom Dataset. URL: https://blog.paperspace.com/train-yolov7-custom-data/;
3. Manivannan Murugavel - Object Tracking - Referenced with the previous frame using Euclidean. URL: distance.https://manivannan-ai.medium.com/object-tracking-referenced-with-the-prevIoUs-frame-using-euclidean-distance-49118730051a (дата звернення: 9.04.2019);
4. Aditya Mittal - Haar Cascades, Explained. URL: https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d (дата звернення: 21.12.2020);
5. Rapid Object Detection using a Boosted Cascade of Simple Features
6. BenMauss - The Integral Image Introduction to Computer Vision, Part 3. URL: https://levelup.gitconnected.com/the-integral-image-4df3df5dce35 (дата звернення: 13.02.2021);
7. Introduction to Convolution Neural Network. URL: https://www.geeksforgeeks.org/introduction-convolution-neural-network/ (дата оновлення: 24.03.2023);
8. Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do та Kaori Togashi Convolutional neural networks: an overview and application in radiology. URL: https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9 (дата звернення: 22.06.2018);
9. Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi: You Only Look Once: Unified, Real-Time Object Detection;

10. Sairaj Neelam - YOLO for Object Detection, Architecture Explained!. URL: https://medium.com/analytics-vidhya/understanding-yolo-and-implementing-yolov3-for-object-detection-5f1f748cc63a (дата звернення: 29.08.2021);
11. Chien-Yao Wang, Alexey Bochkovskiy та Hong-Yuan Mark Liao: YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detector.

ДОДАТКИ