МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

СТРУТОВСЬКИЙ МАКСИМ ІГОРОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 20__ р.

АВТОМАТИЧНА ДЕТЕКЦІЯ ТРАНСПОРТНИХ ЗАСОБІВ НА ВІДЕО-СЦЕНАХ, ЩО ОТРИМАНІ ВІД ДРОНІВ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (магістерська) робота

Науковий керівник:
С. Д. Штовба, професор
кафедри інформаційних
технологій, д.т.н., професор
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____

Вінниця — 2024

АНОТАЦІЯ

Струтовський М.І. Автоматична детекція транспортних засобів на відео-сценах, що отримані від дронів. Спеціальність 122 «Комп'ютерні науки», Освітня програма «Комп'ютерні технології обробки даних (Data Science)». Донецький національний університет імені Василя Стуса, Вінниця, 2024.

У кваліфікаційній роботі досліджено розробку сервісу автоматичної детекції транспортних засобів на відео-сценах. Показано процес навчання моделі YOLO на датасеті UAVDT, а також процес розробки Nvidia Deepstream SDK сервісу. Встановлено та обґрунтовано використання Nvidia Deepstream SDK через численні переваги над іншими фреймворками, а також переваги Yolov8 перед іншими нейронними мережами.

Ключові слова: нейронні мережі, машинне навчання, Yolo, UAVDT, Deepstream SDK.

73 с., 1 табл., 35 рис., 51 джерело.

Strutovskyi M. Automatic detection of vehicles in video scenes obtained from drones. Specialty 122 «Computer Science», Programme «Data Science». Vasyl' Stus Donetsk National University, Vinnytsia, 2024.

In the qualification work, the development of the service of automatic detection of vehicles on video scenes was investigated. The training process of the YOLO model on the UAVDT dataset is shown, as well as the development process of the Nvidia Deepstream SDK service. The use of the Nvidia Deepstream SDK is established and justified due to its numerous advantages over other frameworks, as well as Yolov8's advantages over other neural networks.

Keywords: neural networks, machine learning, Yolo, UAVDT, Deepstream SDK.

73 pages, 1 table, 35 images, 51 sources.

ЗМІСТ

ВСТУП 5
РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ДЕТЕКЦІЙ ТА ОГЛЯД ТЕХНОЛОГІЙ 7
1.1 Огляд технологій детекції об’єктів 7
1.1.1 R-CNN, Fast R-CNN, Faster R-CNN 9
1.1.2 SSD 16
1.1.3 YOLO 20
1.2 Вибір моделі для виявлення транспорту в режимі реального часу 23
1.3 Огляд датасетів для навчання розпізнавання транспорту 25
1.3.1 VisDrone Dataset 26
1.3.2 Unmanned Aerial Vehicles (UAVs/UAVDT) 29
1.3.3 Stanford Drone Dataset 30
1.3.4 DOTA (Dataset for Object Detection in Aerial Images) 31
1.3.5 Вибір найкращого датасету для розпізнавання транспорту з дрону .. 33
1.4 Інтеграція моделі у сервіс 33
1.5 Контейнеризація сервісу 39
Висновок до розділу 1 43
РОЗДІЛ 2 РОЗРОБКА СЕРВІСУ ДЕТЕКЦІЙ ТРАНСПОРТНИХ ЗАСОБІВ 45
2.1 Архітектура сервісу 45
2.2 Налаштування та тренування моделі YOLO на UAVDT датасеті 46
2.3 Оптимізація моделі для сервісу 54
2.3.1 Обрізання (Pruning) 54
2.3.2 Квантування (Quantization): 55
2.3.3 Розгортання за допомогою DeepStream SDK: 56

2.4 Інтеграція з Deepstream SDK 57
2.5 Kafka події 63
2.6 Реалізація моніторингу REST API 64
Висновок до розділу 2 66
ВИСНОВОК 68
СПИСОК ДЖЕРЕЛ 70

ВСТУП

Актуальність дослідження. У сфері автономних систем здатність точно виявляти та розпізнавати об’єкти в режимі реального часу є важливою функціональністю, особливо для дронів. Поява компактних, високопродуктивних обчислювальних платформ, таких як NVIDIA Jetson, стала каталізатором революції в мобільній робототехніці, дозволивши розгортати вдосконалені моделі машинного навчання безпосередньо на периферійних пристроях. Цей технологічний стрибок особливо актуальний у контексті виявлення об’єктів транспортного засобу, де здатність обробляти та інтерпретувати візуальні дані на льоту має першочергове значення для виконання завдань, починаючи від моніторингу руху до пошуково-рятувальних операцій. Використання дронів для патрулювання границь дозволяє оперативно виявляти та відслідковувати підозрілі транспортні засоби. Завдяки автоматичному виявленню підозрілих об'єктів, оператори можуть швидко реагувати на можливі порушення та приймати відповідні заходи. У віддалених сільських районах може бути обмежена кількість камер для моніторингу дорожнього руху, що ускладнює збір даних та забезпечення безпеки на дорогах. Дрони, обладнані системами виявлення транспортних засобів, можуть бути використані для пошуку транспортних засобів в важкодоступних місцях.

Ця задача є актуальною та важливою в сучасному світі, оскільки дрони стають все більш популярними і знаходять застосування в різних галузях, включаючи моніторинг дорожнього руху, військову розвідку, пошук і рятування, а також автономну навігацію

Об'єкт дослідження – виявлення транспортних засобів на відеозаписах, отриманих з дронів або безпілотних літальних апаратів (UAVs).

Предмет дослідження – моделі, алгоритми та програмні засоби для автоматичного виявлення транспортних засобів на відеозаписах, отриманих з дронів або безпілотних літальних апаратів.

Метою дослідження є підвищення достовірності виявлення транспортних засобів на відеозаписах, отриманих з використанням дронів та безпілотних літальних апаратів за рахунок вдосконалення методів автоматичного детекції та ефективної їх програмної реалізації.

Завданням дослідження цієї роботи є:

1. Розробка та оптимізація алгоритмів автоматичного виявлення транспортних засобів на відеозаписах, знятих з дронів.

2. Вибір та апробація передових моделей машинного навчання для виявлення об'єктів у реальному часі, зокрема моделі YOLO (You Only Look Once).

3. Порівняння та оцінка ефективності розроблених алгоритмів та моделей в різних сценаріях, включаючи вимірювання точності та швидкості виявлення.

Новизна отриманих результатів – інтеграція моделі, навченої на датасеті UAVDT в середовище Nvidia Deepstream SDK з додатковими мікросервісами, як Apache Kafka.

Публікація та апробація – результати досліджень доповідалися на конференції 2023 IEEE 13th International Conference on Electronics and Information Technologies (ELIT) та опубліковані в збірнику її матеріалів [51].

РОЗДІЛ 1

ТЕОРЕТИЧНІ ОСНОВИ ДЕТЕКЦІЙ ТА ОГЛЯД ТЕХНОЛОГІЙ

1.1 Огляд технологій детекції об’єктів

Виявлення (детекція) об'єктів - це комп'ютерна технологія, пов'язана з комп'ютерним зором і обробкою зображень, яка займається виявленням екземплярів семантичних об'єктів певного класу (таких як люди, будівлі або автомобілі) на цифрових зображеннях і відео. Добре дослідженими областями виявлення об'єктів є виявлення обличчя і виявлення пішоходів. Виявлення об'єктів має застосування в багатьох галузях комп'ютерного зору, включаючи пошук зображень і відеоспостереження.

Детекція та відстеження об'єктів за допомогою глибинного навчання є основою багатьох сучасних додатків комп'ютерного зору. Наприклад, виявлення об'єктів дозволяє створювати системи інтелектуального медичного моніторингу, автономне водіння, розумне відеоспостереження, виявлення аномалій, робототехніку та багато іншого. Кожен додаток штучного інтелекту зазвичай потребує комбінації різних алгоритмів, які формують потік (пайплайн) з кількох етапів обробки.

Технологія зображень штучного інтелекту значно прогресувала останніми роками. Можна використовувати широкий спектр камер, включаючи комерційні охоронні та системи відеоспостереження. Використовуючи сумісну платформу ПЗ для ШІ, таку як Viso Suite, не потрібно купувати камери ШІ з вбудованими можливостями розпізнавання зображень, оскільки цифровий відеопотік практично будь-якої відеокамери можна аналізувати за допомогою моделей детекції об'єктів. Це робить додатки більш гнучкими, оскільки вони більше не залежать від індивідуальних датчиків, дорогих установок та вбудованих апаратних систем, які потрібно замінювати кожні 3-5 років. Тим часом обчислювальна потужність значно збільшилася і стала набагато ефективнішою. В останні роки обчислювальні платформи переходили до

паралелізації за допомогою багатоядерної обробки, графічних процесорів (GPU) та прискорювачів ШІ, таких як тензорні обчислювальні блоки (TPU). Таке обладнання дозволяє застосовувати комп'ютерний зір для детекції та відстеження об'єктів у реальному часі. Отже, швидкий розвиток глибоких згорткових нейронних мереж (CNN) та покращена обчислювальна потужність GPU є основними драйверами значного прогресу в області комп'ютерного зору, заснованого на детекції об'єктів. Ці досягнення зробили можливим ключовий архітектурний концепт, названий Edge AI.

Виявлення різних об’єктів широко використовується в завданнях комп'ютерного зору, таких як анотування зображень, підрахунок транспортних засобів, визначення діяльності, виявлення обличчя, розпізнавання обличчя, спільне виділення об'єктів на відео (рис. 1.1). Також використовується для відстеження об'єктів, наприклад, відстеження м'яча під час футбольного матчу, відстеження руху крикетного битка або відстеження особи на відео.

Рисунок 1.1 – Приклад детекції автотранспорту

Часто тестові зображення вибираються з різних розподілів даних, що робить завдання виявлення об'єктів значно складнішим. Для вирішення викликів, спричинених розривом між навчальними та тестовими даними, було запропоновано багато підходів до безнаглядної адаптації до домена. Простим і прямолінійним рішенням для зменшення розриву між доменами є застосування

підходу перекладу зображення в зображення, такого як cycle-GAN. Одним із застосувань перехресного виявлення об'єктів у різних сферах є автономне водіння, де моделі можуть бути навчені на великій кількості сцен з відеоігор, оскільки мітки можна генерувати без ручної праці.

Для кожного класу об'єктів існують свої особливості, які допомагають у класифікації цього класу - наприклад, всі кола є круглими. Для виявлення класу об'єктів використовуються ці особливості. Наприклад, при пошуку кіл, шукають об'єкти, які знаходяться на певній відстані від точки (тобто центру). Так само, при пошуку квадратів потрібні об'єкти, які перпендикулярні на кутах і мають рівні довжини сторін. Схожий підхід використовується для розпізнавання обличчя, де можна знайти очі, ніс і губи, а також особливості, такі як колір шкіри і відстань між очима.

Методи виявлення об'єктів, як правило, поділяються на два типи: засновані на нейронних мережах і не-нейронні підходи. У не-нейронних підходах необхідно спочатку визначити ознаки за допомогою одного з методів нижче, а потім використовувати техніку, таку як метод опорних векторів (SVM), для класифікації. З іншого боку, нейронні техніки здатні виконувати виявлення об'єктів з початку до кінця, не специфікуючи явно ознаки, і, як правило, базуються на згорткових нейронних мережах (CNN). Також існують такі техніки як R-CNN, SSD та інші.

1.1.1 R-CNN, Fast R-CNN, Faster R-CNN

Архітектура мережі R-CNN (Regions With CNNs) була розроблена командою з UC Berkley для застосування Convolution Neural Networks до задачі виявлення об'єктів. Існуючі на той момент підходи до вирішення таких задач наблизилися до максимуму своїх можливостей і значно покращити їх показники не вдавалося.

R-CNN добре зарекомендували себе у класифікації зображень, і в даній мережі вони були застосовані для того ж самого. Для цього на вхід R-CNN подавалося не все зображення цілком, а попередньо виділені іншим способом

регіони, на яких передбачувано містилися якісь об'єкти. На рисунку 1.2 зображено приклад роботи R-CNN, зображення подається регіонами.

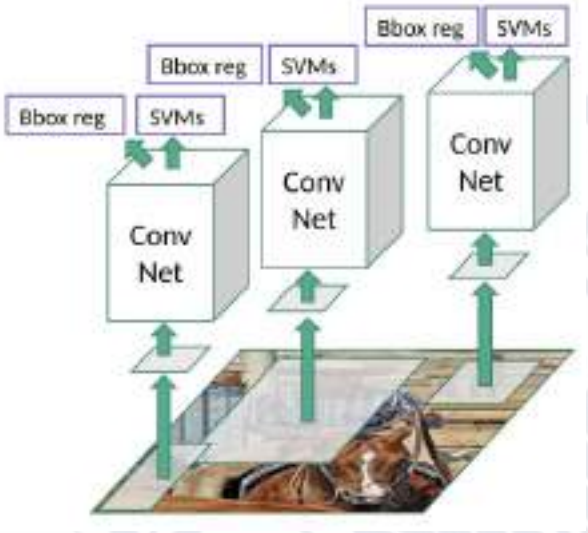


Рисунок 1.2 – Приклад регіонів для використання R-CNN

Архітектура R-CNN використовується для виявлення класів об'єктів на зображеннях та визначення обмежувальних рамок цих об'єктів. Архітектура R-CNN була розроблена, оскільки класифікація не може бути здійснена для більш ніж одного об'єкта з CNN на картинках, що містять більше ніж один об'єкт.

Загальний принцип роботи R-CNN відбувається у два етапи. Спочатку з допомогою селективного пошуку визначаються регіони, де може знаходитися об'єкт на візуалі, потім після визначення регіонів кожен регіон подається як вхід до моделі CNN, і проводиться процес прогнозування для класів та обмежувальних рамок.

У R-CNN використовується селективний пошук для ідентифікації кандидатів на конкретні регіони. Кожен кандидат у регіон подається як вхід до різних мереж CNN. В результаті операцій, виконаних на острівці регіону, отримуються приблизно 2000 різних регіонів, 2000 мереж CNN використовуються для 2000 отриманих регіонів. Класи об'єктів визначаються за допомогою SVM, використовуючи ознаки з цих мереж, а обмежувальні рамки об'єктів визначаються за допомогою регресії.

Оцінка Intersection Over Union (IoU) відноситься до точності передбачених обмежувальних рамок. Отримана обмежувальна рамка та реальна обмежувальна рамка порівнюються, з цього процесу порівняння отримується оцінка IoU (рис. 1.3).

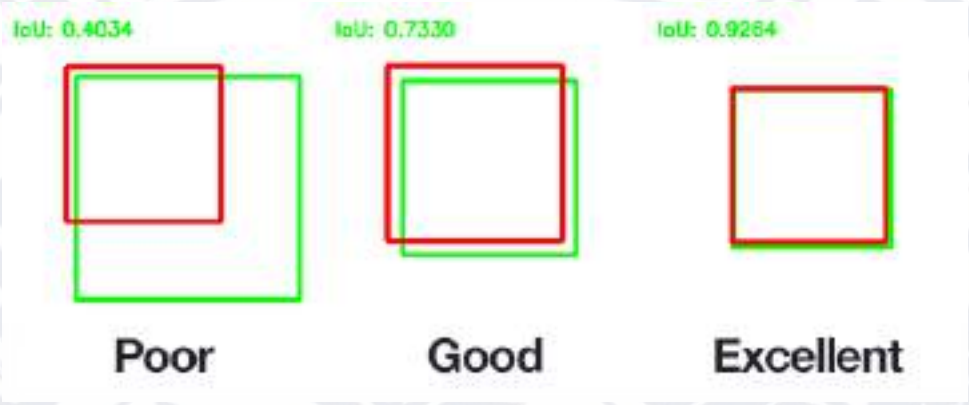


Рисунок 1.3 – Порівняння обмежувальних рамок за допомогою IoU

Не всі регіони, отримані з зображень, використовуються. Для отримання правильних областей використовується техніка Non-Max Suppression. За допомогою цієї техніки зберігаються обмежувальні рамки, отримані з оцінкою Intersection Over Union більше ніж 0.5, а інші обмежувальні рамки ігноруються. Якщо для одного об'єкта отримано більше ніж одну обмежувальну рамку з оцінкою більше ніж 0.5, використовується обмежувальна рамка з найвищою оцінкою IoU.

Вартість моделей R-CNN є досить високою, оскільки для кожного зображення витягуються майже 2000 різних кандидатських регіонів, для кожного регіону використовуються різні мережі CNN. Ці кроки процесу викликають як значні витрати, так і тривалий час навчання. З цієї причини різні моделі CNN, створені для кожного регіону в архітектурі R-CNN, були видалені, і була розроблена архітектура Fast R-CNN, яка використовує одну CNN для регіонів. На відміну від R-CNN, було розроблено використання CNN, SVM та Регресора. Архітектура, створена з комбінації CNN, SVM та Регресора, показала дуже хороші результати з розробленими моделями (рис. 1.4).

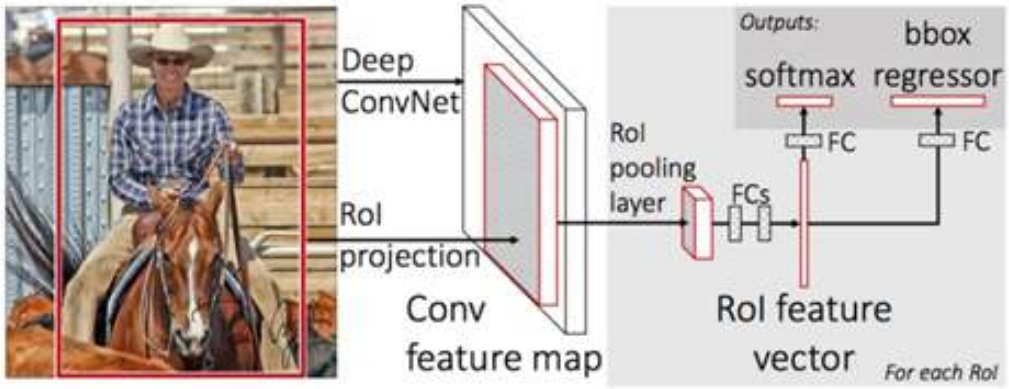


Рисунок 1.4 – Використання SVM, регресора в Fast R-CNN

Все зображення обробляється за допомогою CNN, і отримуються карти ознак. Збираються необхідні ознаки для рекомендацій регіонів (карта ознак пропозиції регіону). Потім до отриманих карт ознак застосовується операція max pooling, і розміри карт ознак зменшуються. Шар, де виконується процес max pooling, називається шаром RoI (Region of Interest) pooling. Карти ознак зменшених розмірів перетворюються в одновимірний вектор і подаються як вхід до моделі CNN. За допомогою Softmax визначається інформація про клас об'єкта в регіоні, тоді як регресор обмежувальної рамки об'єкта визначається. Дана модель Fast-RCNN працює приблизно в 10 разів швидше, ніж R-CNN.

Оскільки селективний пошук, що застосовується в R-CNN та Fast R-CNN, є витратним з точки зору обчислень, у Faster R-CNN використовується мережа пропозицій регіонів (Region Proposal Network, RPN). Ефективність використання RPN була продемонстрована певними дослідженнями.

Будь-яке зображення будь-якого розміру приймається як вхідні дані. Потім зображення подається на вхід моделі CNN. Якщо використовувані моделі є такими, як VGG16, AlexNet, то модель слід реконструювати без використання повнозв'язного шару у моделі. Оскільки для RPN потрібно подати карту ознак як вхід, карти ознак також виробляються згортковими шарами (рис. 1.5).

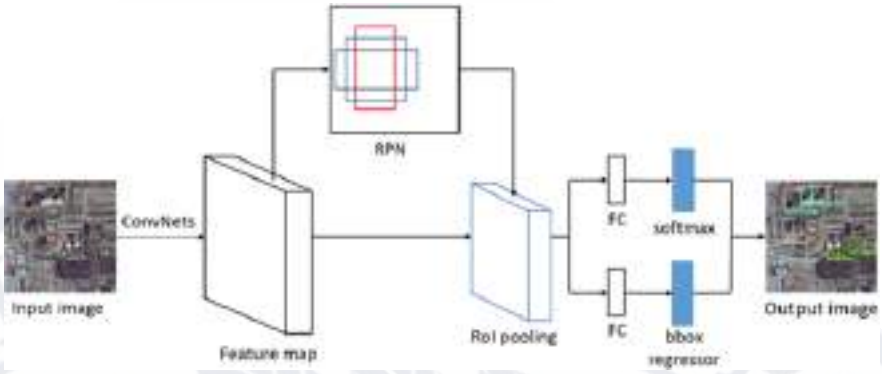


Рисунок 1.5 – Faster R-CNN архітектура

Важливою структурою для RPN є якірні рамки. Якорі - це рамки з різними масштабами і співвідношеннями сторін. Поки створювана маленька мережа ковзає по карті ознак, здійснюється пошук об'єктів у карті ознак відповідно до якоря.

Перший згортковий шар в RPN за замовчуванням застосовує фільтр 3x3, кількість вихідних каналів становить 512. Цей згортковий шар бере карту ознак на вхід. Вихід згорткового шару подається на вхід до двох різних згорткових шарів. Обидва мають розмір фільтра 1x1. Перший шар - cls, тобто шар класифікації. Шар cls інформує нас, чи є об'єкт в місці ковзного вікна чи ні. Тут застосовується бінарний процес класифікації. У випадку об'єкта - 1, у випадку відсутності об'єкта - 0, на етапі моделювання зазвичай використовується сигмоїдна активаційна функція з згортковим шаром. Кількість вихідних каналів цього шару виражається як 2 * 9, за замовчуванням значення k становить 9 (тому розмір фільтра в першому шарі - 3x3), кількість вихідних каналів шару Cls повинна бути 2 * k. Шар Reg працює паралельно шару Cls. Шар Reg відповідальний за малювання обмежувальних рамок об'єктів, виявлених у Cls. Оскільки він містить інформацію про 4 координати, кількість вихідних каналів цього шару виражається як 4 * 9 (4 * k). Для шару Reg у кодуванні моделі віддається перевага лінійній активаційній функції з згортковим шаром.

Підсумовуючи, в результаті процесів фільтрації, вихід RPN - це якірні рамки, позначені як такі, що мають об'єкт. Обмежувальні рамки зі значенням 0 на шарі cls відправляються на задній план.

Шар RoI, який йде після RPN, отримує на вхід виходи шарів cls та reg RPN, а також карту ознак, яку RPN бере на вхід. Шар RoI відповідальний за уніфікацію розміру кожної карти ознак перед повнозв'язним шаром, який є останнім шаром Faster R-CNN. Оскільки рекомендації регіонів від RPN мають різні розміри (тому що якірні рамки мають різні масштаби та співвідношення сторін), необхідно виробляти карти ознак з фіксованим розміром. Шар RoI використовує максимальний пулінг шару 7x7 для розміру. Кількість вихідного шару становить 512.

Значення, отримані після RoI, піддаються процесу класифікації, подаючи їх на вхід до повнозв'язного шару після вирівнювання. У процесі класифікації створюється обмежувальна рамка об'єкта, клас якого передбачається, з допомогою регресора.

Отже, Faster R-CNN є однією з провідних архітектур для задачі виявлення об'єктів (object detection), яка вносить важливі інновації для покращення швидкості та точності порівняно з її попередниками R-CNN та Fast R-CNN. Ось деякі ключові переваги Faster R-CNN:

- Зменшення обчислювальних витрат: Faster R-CNN вводить мережу пропозицій регіонів (Region Proposal Network, RPN), яка значно зменшує кількість регіонів-кандидатів для перевірки, тим самим знижуючи обчислювальні витрати порівняно з селективним пошуком у R-CNN.
- Єдність процесу: RPN інтегрована з основною мережею CNN, що дозволяє спільне навчання і оновлення в реальному часі, відмовляючись від потреби в тренуванні кількох окремих моделей, як у R-CNN.
- Швидкість: завдяки використанню RPN, Faster R-CNN може швидше визначати регіони інтересів, що підвищує швидкість виявлення об'єктів в реальному часі.

- Точність: інтеграція RPN з основною мережею дозволяє точніше визначати регіони інтересів, що покращує загальну точність виявлення об'єктів.
- Гнучкість: Faster R-CNN підтримує різноманітні архітектури основної мережі (backbones), такі як VGG, ResNet, що робить її гнучкою для різних задач і наборів даних.
- Оптимізація використання пам'яті: завдяки спільному використанню карт ознак для RPN і детектора, Faster R-CNN ефективніше використовує пам'ять, що важливо для великих наборів даних.
- Масштабування якорних рамок: завдяки якорним рамкам різного масштабу та співвідношення сторін, Faster R-CNN здатна краще виявляти об'єкти незалежно від їх розміру та форми.
- End-to-end навчання: архітектура дозволяє тренувати всю систему виявлення об'єктів кінця до кінця, що забезпечує більшу консистентність та відповідність між компонентами моделі.

Ці переваги роблять Faster R-CNN одним з найбільш популярних та ефективних рішень для задач виявлення об'єктів у дослідженнях і реальних застосуваннях. Але незважаючи на численні переваги Faster R-CNN у сфері виявлення об'єктів, існують також деякі недоліки, які важливо враховувати:

- Високі обчислювальні вимоги: Faster R-CNN вимагає значної обчислювальної потужності для тренування та висновків, особливо якщо використовуються складні архітектури згорткових нейронних мереж, як-от VGG16 чи ResNet.
- Час навчання: хоча Faster R-CNN швидший за попередні моделі, процес навчання може бути досить тривалим, особливо при великих наборах даних і більш глибоких архітектурах.
- Складність реалізації: архітектура Faster R-CNN вважається більш складною для розуміння та реалізації порівняно з іншими методами виявлення об'єктів, такими як YOLO чи SSD, що може ускладнити використання для новачків у цій сфері.

- Складності з маленькими об'єктами: Faster R-CNN може мати труднощі з виявленням дуже маленьких об'єктів через фіксовані розміри якірних рамок та максимальний пулінг, які можуть втратити важливу інформацію на високих рівнях абстракції.
- Залежність від якорних рамок: ефективність моделі залежить від попередньо визначених розмірів та співвідношень сторін якірних рамок, що може не завжди ідеально підходити під конкретні набори даних.

1.1.2 SSD

Існує компроміс між точністю та швидкістю. Faster R-CNN забезпечує високу точність, але існують інші архітектури виявлення об'єктів, які можуть пропонувати швидші результати з меншою точністю, що може бути переважним у деяких додатках. Розглянемо одну з таких – Single-Shot Detector (SSD).

SSD складається з двох компонентів: основної моделі та голови SSD. Основна модель зазвичай є попередньо навченою мережею класифікації зображень як екстрактором ознак. Це зазвичай така мережа, як ResNet, навчена на ImageNet, від якої видалено останній повнозв'язний класифікаційний шар. Таким чином, ми маємо глибоку нейронну мережу, яка здатна витягувати семантичне значення з вхідного зображення, зберігаючи при цьому просторову структуру зображення, хоча й у меншій роздільності. Для ResNet34 основа дає 256 7x7 карт ознак для вхідного зображення. Ми пояснимо, що таке ознака та карта ознак, пізніше. Голова SSD - це просто один або кілька доданих до цієї основи згорткових шарів, виходи яких тлумачаться як обмежувальні рамки та класи об'єктів у просторовому розташуванні активацій останніх шарів.

На рисунку 1.6 перші кілька шарів (білі прямокутники) є основою, останні кілька шарів (сині прямокутники) представляють голову SSD.

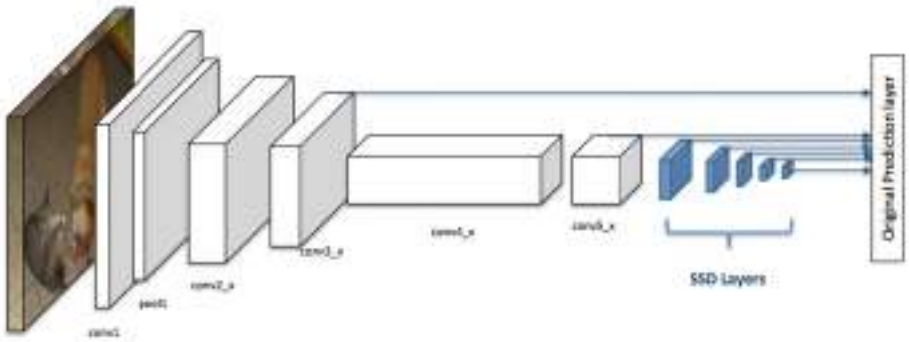


Рисунок 1.6 – Архітектура Single-Shot Detector

Замість використання ковзного вікна, SSD ділить зображення за допомогою сітки і робить кожну сітчасту комірку відповідальною за виявлення об'єктів у цій області зображення. Виявлення об'єктів просто означає прогнозування класу та місцезнаходження об'єкта в цій області. Якщо об'єкт відсутній, ми розглядаємо це як фоновий клас, а місцезнаходження ігнорується. Наприклад, ми могли б використати сітку 4x4 у наведеному нижче прикладі на рисунку 1.7. Кожна сітчаста комірка може виводити положення та форму об'єкта, який вона містить.

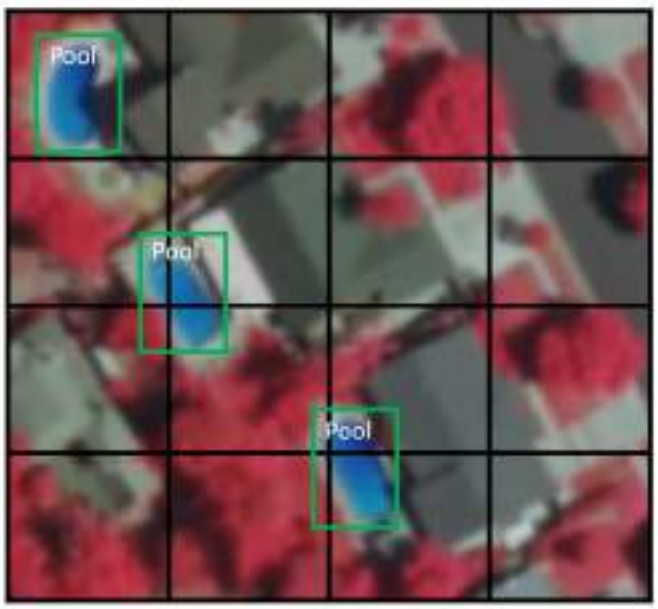


Рисунок 1.7 – Сітка зображення для SSD

Якщо в одній сітчастій комірці знаходиться кілька об’єктів або потрібно виявити кілька об’єктів різних форм, використовуються якірні рамка та рецептивне поле.

Кожній сітчастій комірці в SSD можуть бути призначені кілька якірних/попередніх рамок. Ці якірні рамки є попередньо визначеними, і кожна з них відповідає за розмір і форму в межах сітчастої комірки.

SSD використовує фазу зіставлення під час навчання, щоб зіставити відповідну якірну рамку з обмежувальними рамками кожного об'єкта реального світу всередині зображення. По суті, якірна рамка з найвищим ступенем перекриття з об'єктом відповідає за прогнозування класу цього об'єкта та його розташування. Ця властивість використовується для навчання мережі та для прогнозування виявлених об'єктів та їх розташувань після навчання мережі. На практиці кожна якірна рамка визначається співвідношенням сторін та рівнем збільшення.

Не всі об'єкти мають квадратну форму. Деякі з них довші, а деякі ширші, у різній мірі. Архітектура SSD дозволяє визначати попередньо задані співвідношення сторін якірних рамок, щоб врахувати це. Параметр співвідношень може бути використаний для вказівки різних співвідношень сторін якірних рамок, асоційованих з кожною сітчастою коміркою на кожному рівні збільшення/масштабування.

Розмір якірних рамок не обов'язково має бути таким самим, як розмір сітчастої комірки. Може стояти задача знайти менші або більші об'єкти всередині сітчастої комірки. Для цього параметр збільшення використовується для вказівки на те, наскільки якірні рамки потрібно збільшувати або зменшувати відносно кожної сітчастої комірки.

Рецептивне поле визначається як регіон у вхідному просторі, на який дивиться певна ознака CNN (тобто на який вона впливає). Через операцію згортки ознаки на різних шарах представляють різні розміри регіонів на вхідному зображенні. Чим глибше, тим більший розмір представляє ознака. У цьому прикладі нижче ми починаємо з нижнього шару (5x5) і потім

застосовуємо згортку, яка призводить до середнього шару (3x3), де одна ознака (зелений піксель) представляє 3x3 регіон вхідного шару (нижнього шару). Потім застосовуємо згортку до середнього шару і отримуємо верхній шар (2x2), де кожна ознака відповідає 7x7 регіону на вхідному зображенні. Такі зелені та помаранчеві двовимірні масиви також називають картами ознак, які відносяться до набору ознак, створених шляхом застосування одного і того ж екстрактора ознак у різних місцях вхідної карти у стилі ковзного вікна (рис. 1.8). Ознаки в одній і тій же карті ознак мають одне і те ж рецептивне поле і шукають однаковий патерн, але в різних місцях. Це створює просторову інваріантність ConvNet.

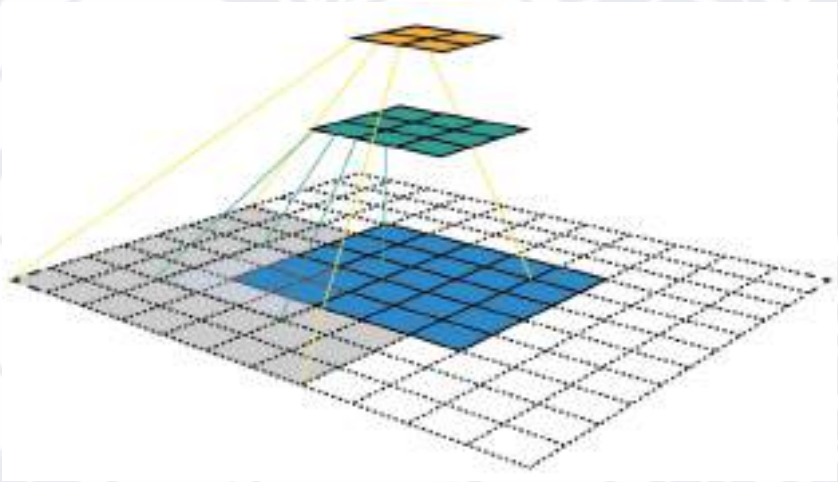


Рисунок 1.8 – Візуалізація рецептивного поля

Порівняємо SSD модель із Faster R-CNN, оглянутою вище в підрозділі 1.2.

Переваги SSD:

- Швидкість: SSD зазвичай швидше працює, ніж Faster R-CNN, оскільки вона виконує виявлення об'єктів за один прохід (single shot), що робить її більш підходящою для реального часу або застосувань з обмеженими ресурсами.
- Простота архітектури: SSD має більш просту архітектуру порівняно з Faster R-CNN, що робить її легшою для реалізації та налаштування.

- Ефективність виявлення малих об'єктів: Завдяки використанню багатьох згорткових шарів для виявлення, SSD може бути більш ефективною у виявленні малих об'єктів.

Недоліки SSD:

- Точність: SSD зазвичай має нижчу точність порівняно з Faster R-CNN, особливо на складних зображеннях з декількома об'єктами або об'єктами, які сильно перекриваються.
- Обмеження в розмірах якірних рамок: SSD використовує фіксований набір якірних рамок, що може обмежувати її здатність точно виявляти об'єкти різних розмірів і форм.

Рецептивне поле є центральною ідеєю архітектури SSD, оскільки воно дозволяє нам виявляти об'єкти різних масштабів та виводити більш точну обмежувальну рамку. Якщо визначається сітка 4x4, найпростіший підхід - це просто застосувати згортку до цієї карти ознак і перетворити її на 4x4. Цей підхід насправді може працювати певною мірою і є саме ідеєю YOLO (You Only Look Once). Додатковий крок, який робить SSD, полягає в тому, що він застосовує більше згорткових шарів до карти ознак основи і дозволяє кожному з цих згорткових шарів виводити результати виявлення об'єктів. Оскільки ранні шари, що мають менше рецептивне поле, можуть представляти об'єкти меншого розміру, прогнози з ранніх шарів допомагають у роботі з об'єктами меншого розміру.

1.1.3 YOLO

You Only Look Once (YOLO) пропонує використання нейронної мережі від кінця до кінця, яка одночасно робить прогнози обмежувальних рамок та ймовірностей класів. Це відрізняється від підходу, який використовували попередні алгоритми виявлення об'єктів, які адаптували класифікатори для виконання виявлення.

YOLO використовує принципово інший підхід до виявлення об'єктів і досягає результатів на рівні найкращих сучасних практик, випереджаючи інші алгоритми виявлення об'єктів у реальному часі на значний відсоток.

Поки алгоритми, подібні до Faster R-CNN, працюють шляхом виявлення можливих регіонів інтересу за допомогою мережі пропозицій регіонів і потім окремого визначення цих регіонів, YOLO виконує всі свої прогнози за допомогою одного повнозв'язного шару.

Методи, які використовують мережі пропозицій регіонів, виконують кілька ітерацій для одного і того ж зображення, тоді як YOLO обходиться однією ітерацією.

З моменту першого випуску YOLO у 2015 році було запропоновано кілька нових версій цієї моделі, кожна з яких покращує і доповнює свого попередника.

Алгоритм YOLO приймає зображення як вхід і потім використовує просту глибоку згорткову нейронну мережу для виявлення об'єктів на зображенні. Перші 20 згорткових шарів моделі попередньо навчаються, використовуючи ImageNet, шляхом підключення тимчасового середнього пулінгу та повнозв'язного шару. Потім ця попередньо навчена модель перетворюється для виконання виявлення, оскільки попередні дослідження показали, що додавання згорткових та пов'язаних шарів до попередньо навченої мережі покращує продуктивність. Останній повнозв'язний шар YOLO прогнозує як ймовірності класів, так і координати обмежувальних рамок.

YOLO ділить вхідне зображення на сітку S × S. Якщо центр об'єкта потрапляє в комірку сітки, ця комірка відповідає за виявлення цього об'єкта. Кожна комірка сітки прогнозує B обмежувальних рамок і бали впевненості для цих рамок. Ці бали впевненості відображають, наскільки модель впевнена, що рамка містить об'єкт, і наскільки точною, на її думку, є прогнозована рамка.

YOLO прогнозує кілька обмежувальних рамок на комірку сітки. Під час навчання ми хочемо, щоб лише один прогнозувач обмежувальної рамки був відповідальний за кожен об'єкт. YOLO призначає один прогнозувач, щоб бути

"відповідальним" за прогнозування об'єкта на основі того, який прогноз має найвищий поточний IOU з істинним значенням. Це призводить до спеціалізації між прогнозувачами обмежувальних рамок. Кожен прогнозувач стає кращим у прогнозуванні певних розмірів, співвідношень сторін або класів об'єктів, покращуючи загальний показник відгуку.

YOLOv8 представляє собою значне вдосконалення порівняно з попередніми версіями YOLO, такими як YOLOv7 та іншими. Ця версія включає в себе нові характеристики та поліпшення, які покращують швидкість, точність та гнучкість моделі.

Однією з ключових відмінностей YOLOv8 є її здатність до "anchor-free" виявлення, що означає безпосереднє прогнозування центру об'єкта замість зсуву від відомого якірного боксу. Це зменшує кількість прогнозованих рамок та прискорює обробку, особливо під час пост-процесингу, такого як "Non-Maximum Suppression" (NMS)

Зміни в архітектурі включають новий тип згорткових блоків та різні покращення в модулях. Такі зміни дозволяють YOLOv8 ефективніше використовувати ресурси та досягати кращих результатів.

Ще одним важливим аспектом є зміна у тренувальній рутині. YOLOv8 використовує різні види аугментацій під час навчання, що допомагає моделі краще адаптуватися до різних умов. Наприклад, мозаїчна аугментація використовується на певному етапі навчання для покращення здатності моделі виявляти об'єкти в різних умовах, але відключається на останніх етапах тренування.

З точки зору точності, YOLOv8 показує вражаючі результати в порівнянні з попередніми версіями, особливо на стандартних бенчмарках, таких як COCO. Також модель була протестована на різних наборах даних, що дозволяє оцінити її здатність до узагальнення.

Що стосується розробки та інтеграції, YOLOv8 включає покращений інтерфейс для розробників, зокрема, CLI (командний рядок інтерфейсу), що робить процес тренування моделі більш інтуїтивним та зручним. На рисунку

1.9 зображено графіки порівняння версій YOLO. На них можна побачити, що остання версія YOLO стала менше попередніх за параметрами, але трохи програє по швидкості.

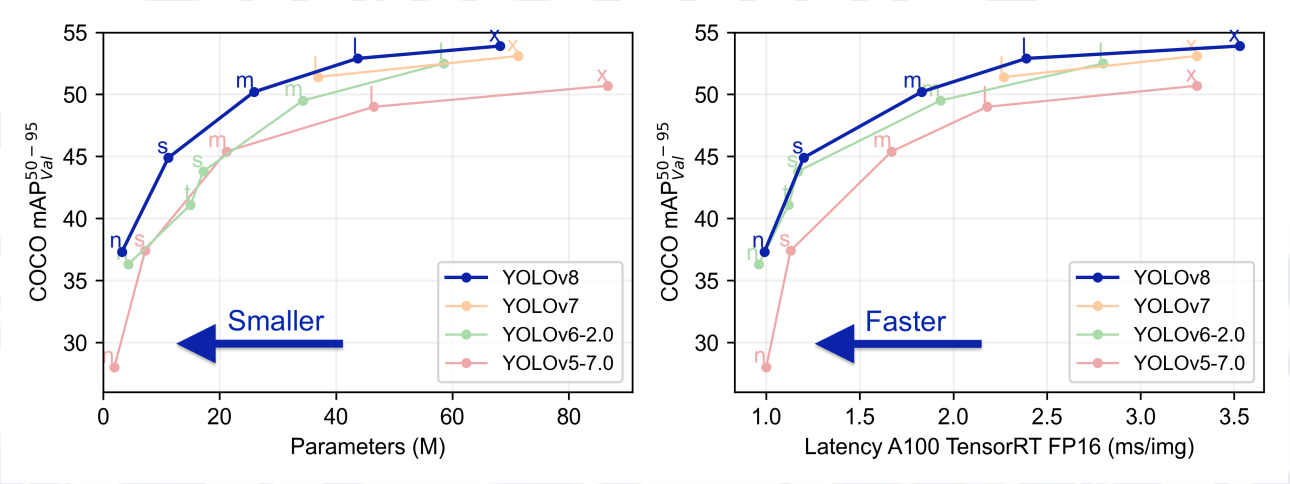


Рисунок 1.9 – Графіки порівняння версій YOLO.

1.2 Вибір моделі для виявлення транспорту в режимі реального часу

Для задачі виявлення транспортних засобів в режимі реального часу, YOLO архітектура, особливо останні версії, як-от YOLOv8, є оптимальним вибором з наступних причин:

- Швидкість та продуктивність у реальному часі.

YOLO відома своєю здатністю до швидкого виявлення об'єктів, що є критично важливим для аналізу зображень з дронів у реальному часі. Оскільки дрони можуть швидко переміщатися та змінювати кути огляду, потрібна модель, яка може миттєво обробляти зображення.

Faster R-CNN хоча і надає високу точність, працює повільніше, що може бути проблематичним для використання в динамічному середовищі.

SSD також швидкий, але може втратити точність при виявленні малих або швидко рухомих об'єктів.

- Точність при різних масштабах.

YOLOv8 ефективно виявляє об'єкти різних розмірів завдяки своїм багатомасштабним згорткам. Це особливо важливо при роботі з зображеннями з дронів, де об'єкти можуть значно відрізнятися за розміром в залежності від висоти польоту.

- Ефективність в різних умовах.

YOLO здатна виявляти об'єкти в різних умовах освітлення та погодних умовах, що є типовим для зйомки з дронів.

- Інтеграція з DeepStream SDK.

YOLO легко інтегрується з NVIDIA DeepStream SDK, що дозволяє ефективно використовувати обчислювальну потужність GPU для швидкої обробки відео в реальному часі.

Вибір YOLO для виявлення транспорту з дрону є оптимальним з точки зору балансу між швидкістю, точністю та гнучкістю. Хоча Faster R-CNN може надати кращу точність, його нижча швидкість робить його менш підходящим для реального часу. SSD, хоча і швидкий, може не забезпечити необхідну точність у виявленні малих або далеких об'єктів. YOLO забезпечує найкращий компроміс для задачі, що вимагає швидкого реагування та високої точності. Результати порівняння описані в таблиці 1.1.

Таблиця 1.1 – Порівняння нейронних мереж за характеристиками

Характеристика	SSD	YOLOv8	Faster R-CNN
Швидкість	Висока (зазвичай швидше за Faster R-CNN)	Дуже висока, підходить для реального часу	Помірна до високої (залежить від конфігурації)
Точність	Висока, але може бути нижчою, ніж у YOLOv8 та Faster R-CNN	Дуже висока, особливо у складних сценах	Дуже висока, часто краща за SSD, але порівнянна з YOLOv8

Комплексність	Простіша, ніж Faster R-CNN, але складніша, ніж YOLO	Відносно проста, особливо у порівнянні з Faster R-CNN	Складна архітектура з окремим етапом виявлення регіонів
Універсальність	Добре підходить для різних розмірів об'єктів	Висока універсальність для різноманітних типів об'єктів та сцен	Висока універсальність, особливо для складних сцен
Виявлення малих об’єктів	Добре, але може мати проблеми з дуже малими об'єктами	Висока ефективність для малих об'єктів	Добре, особливо з налаштуваннями RPN
Реальний час	Підходить для реального часу, але залежить від конфігурації	Висока підходящість для реального часу	Можливо, але залежить від конкретної конфігурації та обчислювальної потужності

1.3 Огляд датасетів для навчання розпізнавання транспорту

Існують різні великі набори даних (датасети), які містять різноманітні зображення та відео транспортних засобів у різних умовах знятих з камери дрону. Використання цих датасетів є важливим для розробки та вдосконалення алгоритмів машинного зору, що дозволяють дронам автоматично виявляти та класифікувати транспортні засоби. Такі дані допомагають моделям навчатися впізнавати транспорт у різних сценаріях, включаючи змінні умови освітлення,

різні фони та різні перспективи. Це особливо важливо для додатків, що вимагають високої точності та надійності, таких як моніторинг дорожнього руху, рятувальні операції або безпека.

Багатство та різноманітність даних в цих датасетах забезпечують, що системи розпізнавання можуть ефективно працювати в реальних умовах, де зустрічаються різні типи та моделі транспортних засобів. Крім того, деякі з цих датасетів містять анотації, які були вручну підготовлені експертами, забезпечуючи високу якість даних для навчання. Це дозволяє моделям точно виявляти не тільки присутність транспортного засобу, але й його тип, розмір, а іноді й поведінку. Такий підхід сприяє розвитку систем автоматизованого нагляду та аналітики, що можуть бути застосовані в різних галузях.

Останнім часом з'явились додаткові датасети з дронів, які включають складніші сценарії, такі як великі транспортні розв'язки або вузькі вулички міст, що дозволяє подальше вдосконалення та адаптацію моделей до ще більшої різноманітності умов. Завдяки цьому, системи розпізнавання, навчені на таких датасетах, можуть бути ефективно використані для моніторингу дорожнього руху, планування містобудування, екологічного нагляду та багатьох інших застосувань.

1.3.1 VisDrone Dataset

Дрони, або загальні БПЛА, оснащені камерами, швидко знаходять застосування у широкому спектрі областей, включаючи аграрний сектор, аерофотозйомку, швидку доставку та спостереження. Відповідно, автоматичне розуміння візуальних даних, зібраних з цих платформ, стає все більш важливим, що все більше зближує комп'ютерний зір з дронами. VisDrone - це масштабний датасет з ретельно анотованою правдивістю (ground-truth) для різних важливих завдань комп'ютерного зору. Набір даних VisDrone2019 зібрано командою AISKYEYE у лабораторії машинного навчання та дата-майнінгу, Тяньцзінського університету, Китай. Датасет складається з 288 відеокліпів, утворених 261 908 кадрами та 10 209 статичними зображеннями,

знятими різними камерами, встановленими на дронах, і охоплює широкий спектр аспектів, включаючи місцезнаходження (зняте у 14 різних містах, розділених тисячами кілометрів у Китаї), середовище (міське і сільське), об'єкти (пішоходи, транспортні засоби, велосипеди тощо) та щільність (рідкісні та переповнені сцени). Зазначимо, що набір даних збирався за допомогою різних платформ дронів (тобто дронів різних моделей), у різних сценаріях та за різних погодних та освітлювальних умовах. Ці кадри вручну анотовані більш ніж 2,6 мільйонами обмежувальних рамок цілей частих інтересів, таких як пішоходи, автомобілі, велосипеди. Деякі важливі атрибути, включаючи видимість сцени, клас об'єкту та перекриття, також надаються для кращого використання даних. На рисунку 1.10 зображено один із кадрів датасету VisDrone2019. На рисунку 1.11 – частина анотацій до цього кадру.

Рисунок 1.10 – Приклад зображення з датасету VisDrone2019

```
499,405,70,100,1,4,0,0
618,296,54,66,1,4,0,0
459,186,50,49,1,4,0,0
443,127,40,40,1,4,0,0
402,112,76,30,1,4,0,1
606,154,38,36,1,4,0,0
765,511,138,142,1,4,0,0
840,416,134,161,1,6,0,0
770,281,56,59,1,4,0,1
770,248,48,36,1,4,0,1
781,217,100,44,1,4,0,0
700,247,10,28,1,1,0,0
710,239,9,38,1,1,0,0
723,234,12,38,1,1,0,0
735,241,9,37,1,1,0,0
790,291,98,130,1,6,0,0
```

Рисунок 1.11 – Приклад файлу анотацій до рисунку 1.10

Датасет VisDrone вирізняється кількома унікальними особливостями, які роблять його важливим ресурсом у галузі комп'ютерного зору та аналізу аерофотознімків:

- Великий Обсяг та Різноманітність Даних: VisDrone містить тисячі зображень та відео, знятих з дронів, що охоплюють широкий спектр сценаріїв, включаючи міські, сільські, промислові райони та інші. Це дозволяє моделям машинного навчання навчатися в різноманітних умовах.
- Високоякісні Анотації: Датасет містить детально анотовані дані, включаючи мільйони обмежувальних рамок для різних об'єктів, таких як пішоходи, транспортні засоби, велосипеди. Ці анотації забезпечують точність навчання та оцінки моделей.
- Виклики Реального Світу: Завдяки реальним умовам зйомки, VisDrone представляє сценарії, які часто зустрічаються в реальному житті, такі як різні погодні умови, різноманітність освітлення та різні рівні щільності об'єктів.

- Географічна Різноманітність: Зібрані з різних місць по всьому Китаю, зображення охоплюють широкий спектр географічних умов, що забезпечує велику різноманітність даних для навчання моделей.
- Підвищення Точності та Надійності Систем: Використання VisDrone сприяє розвитку більш точних та надійних систем розпізнавання та аналізу зображень, особливо для додатків, пов'язаних з дронами.

Завдяки цим особливостям, VisDrone Dataset є одним з найбільш цінних та корисних ресурсів у галузі аерофотознімків, автоматизованого моніторингу та комп'ютерного зору.

1.3.2 Unmanned Aerial Vehicles (UAVs/UAVDT)

UAVDT - це масштабний та складний датасет для виявлення та відстеження за допомогою безпілотних літальних апаратів (БПЛА), що включає близько 80 000 репрезентативних кадрів з 10 годин сирого відео для трьох важливих основних завдань, а саме: виявлення об'єктів (DET), відстеження одного об'єкта (SOT) та відстеження багатьох об'єктів (MOT).

Набір даних знято БПЛА у різних складних сценаріях. Об'єкти інтересу у цьому бенчмарку - транспортні засоби. Кадри вручну були анотовані обмежувальними рамками та деякими корисними атрибутами, наприклад, категорією транспортного засобу та перекриттям.

Бенчмарк UAVDT складається зі 100 відеосеквенцій, які були відібрані з понад 10 годин відео, знятого на платформі БПЛА в ряді локацій у міських районах, представляючи різні звичайні сцени, включаючи площі, артеріальні вулиці, платні станції, автостради, перехрестя та Т-подібні перехрестя (рис. 1.12). Відео записано з частотою 30 кадрів в секунду (к/с), з роздільною здатністю зображення JPEG 1080 × 540 пікселів.

Рисунок 1.12 – Приклад зображень у датасеті UAVDT

1.3.3 Stanford Drone Dataset

Великий набір даних, який містить зображення з висоти пташиного польоту, зібрані на кампусі Стенфордського університету. Він включає анотації різних об'єктів, включаючи пішоходів, велосипедистів, транспортні засоби тощо (рис. 1.13).

Рисунок 1.13 – Приклад зображень у Stanford Drone Dataset

Stanford Drone Dataset (SDD) є цінним ресурсом у галузі комп'ютерного зору, зокрема для аналізу поведінки пішоходів та транспортних засобів у міських умовах. Однак, щодо специфічної задачі навчання детекції транспортних засобів з дрона, SDD може мати певні обмеження.

Основна проблема з SDD полягає у куті зйомки. Зображення в цьому наборі даних були отримані з висоти пташиного польоту, з прямого, зверхнього кута. Натомість, дрони часто використовують кут нахилу або більш скісний кут для виявлення об'єктів, що відрізняється від прямого зверху вниз погляду, притаманного SDD. Зображення, зроблені з дронів під косим кутом, можуть мати перспективні деформації, які не присутні в SDD. Це означає, що моделі, навчені на SDD, можуть не ефективно виявляти транспортні засоби в умовах, типових для зйомки з дронів. Для розробки ефективних систем детекції з дронів, необхідно навчати моделі на даних, що охоплюють широкий спектр кутів зйомки. SDD не надає цієї варіативності, що обмежує його використання для реальних сценаріїв використання дронів. SDD орієнтований на урбаністичні сцени, такі як університетські кампуси або міські вулиці. Це може не відображати різноманітність сцен, які дрони можуть зустрічати в реальному світі, включаючи сільські дороги, промислові зони чи природні ландшафти.

1.3.4 DOTA (Dataset for Object Detection in Aerial Images)

DOTA - це масштабний набір даних для виявлення об'єктів у аерофотознімках. Він може використовуватися для розробки та оцінки детекторів об'єктів у аерофотознімках. Зображення зібрані з різних датчиків та платформ. Кожне зображення має розміри від 800 × 800 до 20,000 × 20,000 пікселів і містить об'єкти, які представляють широкий спектр масштабів, орієнтацій та форм. Екземпляри на зображеннях DOTA анотовані експертами з інтерпретації аерофотознімків за допомогою довільного (8 ступенів свободи) чотирикутника.

Зараз датасет має три версії:

- DOTA-v1.0 містить 15 загальних категорій, 2,806 зображень та 188,282 екземплярів. Пропорції навчального набору, набору для валідації та тестового набору у DOTA-v1.0 становлять 1/2, 1/6 і 1/3 відповідно.
- DOTA-v1.5 використовує ті ж зображення, що й DOTA-v1.0, але також анотовані надзвичайно малі екземпляри (менше 10 пікселів). Крім

того, додано нову категорію "портальний кран". Всього він містить 403,318 екземплярів. Кількість зображень та розділення набору даних такі ж, як у DOTA-v1.0. Ця версія була випущена для DOAI Challenge 2019 з виявлення об'єктів у аерофотознімках на конференції IEEE CVPR 2019.

- DOTA-v2.0 збирає більше зображень з Google Earth, супутника GF-2 та аерофотознімків (на рис. 1.14 представлені анотовані зображення датасету). У DOTA-v2.0 є 18 загальних категорій, 11,268 зображень та 1,793,658 екземплярів. Порівняно з DOTA-v1.5, він додатково включає нові категорії "аеропорт" та "вертолітний майданчик".

Рисунок 1.14 – Приклад анотованих зображень з датасету DOTA

DOTA датасет є одним із найбільших і найбільш різноманітних наборів даних для виявлення об'єктів у аерофотознімках, існують певні аспекти, які обмежують його придатність для тренування моделей детекції транспортних засобів з дронів, зокрема кут зйомки. Як і в Stanford Drone Dataset, зображення в DOTA зазвичай зроблені з прямого, зверхнього кута. Це значно відрізняється від перспектив, які часто використовуються у дронів, що літають на меншій

висоті та забезпечують більш скісний кут зйомки. В результаті, моделі, навчені на DOTA, можуть мати обмежену здатність до адаптації до кутів зйомки, типових для дронів. Аерофотознімки у DOTA часто зроблені з великої відстані, що призводить до маленьких розмірів об'єктів на зображеннях. У контексті детекції транспортних засобів з дронів, це може стати проблемою, оскільки дрони часто використовуються для зйомки з меншої відстані, що надає більший розмір та деталізацію об'єктів.

1.3.5 Вибір найкращого датасету для розпізнавання транспорту з дрону

Оскільки поставлена задача є розпізнавання транспорту з дронів, можемо виключити DOTA та Stanford Drone Dataset з порівняння. Ці датасети не підходять за рахунок прямого, зверхнього кута зйомки. При порівнянні VisDrone Dataset і UAVDT Dataset (Unmanned Aerial Vehicle Dataset), обидва набори даних мають свої унікальні переваги для досліджень у галузі комп'ютерного зору, особливо в контексті аналізу зображень з дронів. Але у UAVDT є переваги: більший фокус на транспортних засобах, складні сценарії, описані категорії транспортних засобів та ступінь перекриття, а також реалістичні умови, які можуть зустрічатися при використанні дронів.

1.4 Інтеграція моделі у сервіс

Перед появою NVIDIA Deepstream SDK основна інтеграція моделей було впровадження моделі за допомогою хмарного рішення, щоб легко інтегрувати модель з REST API. Однак, коли потрібно надавати онлайн-прогнози на основі відео або аудіопотоків, ситуація змінюється. Додавання середовища edge до використання ще більше ускладнює ситуацію. Якщо потрібно зробити систему готовою до виробництва за цих обставин – найкраще рішення є NVIDIA Deepstream SDK.

Сприйняття моделей глибокого навчання як елементів у сервісі або в ланцюзі обробки даних допомагає зрозуміти, як можна їх розгортати. Переважно, розгортання є частиною процесу обробки пакетів, що повторно обмінюється вхідними даними та результатами інференсу в пам'яті. Більше того, коли модель потрібно відкрити для клієнтів, вона, як правило, інтегрується за допомогою REST API.

Тут доступно кілька варіантів. Наприклад, можна використовувати класичний шаблон ЗАПИТ-ВІДПОВІДЬ; альтернативно, можна просто подати запит до черги, припускаючи, що клієнт буде перевіряти його статус до завершення обробки. Одним з факторів, що сприяють складності, є те, що моделі глибокого навчання можуть обробляти різні типи даних. Тому проблема насправді визначається контекстом використання моделі. Зокрема, детекції транспортних засобів в режимі реального часу, потрібен інструмент, здатний ефективно обробляти відеопотоки та розміщувати модель.

Основи обробки відео необхідні для створення інструменту, який розміщуватиме моделі глибокого навчання та надаватиме висновки на основі відеопотоку. На щастя, стабільне рішення вже існує. Це Gstreamer. Розробники Nvidia побудували своє рішення на його основі.

Gstreamer має безліч плагінів, які можна розподілити на джерела (випромінювачі потоку), фільтри (відповідальні за операції над потоком) та стоки (кінцеві елементи потоку). Крім того, плагіни можуть бути організовані у пайплайни для задоволення різних потреб (рис. 1.15).

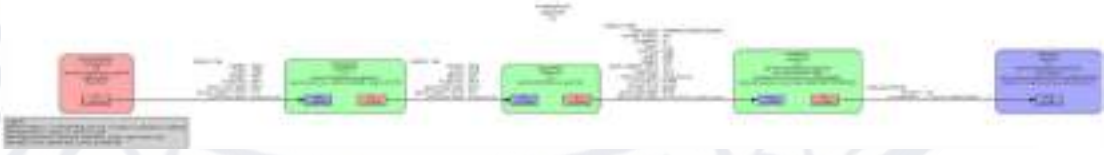


Рисунок 1.15 – Приклад Gstreamer пайплайну

Вихідний код Gstreamer написаний на мові C, також доступні прив'язки до інших мов. Це порівняно низькорівневий фреймворк, інструмент дуже добре задокументований.

Оскільки синхронізації між Gstreamer та Deepstream є нативною, варто розглянути деякі плагіни Gstreamer, розроблені спеціально для використання Deepstream.

NvVideo4Linux2 - це елемент апаратно прискореного декодера відео для GStreamer, який надається NVIDIA як частина DeepStream SDK. Він використовує API Video4Linux2 (V4L2) з пропрієтарними апаратно прискореними кодеками NVIDIA для більш ефективного декодування відеопотоків, переносячи навантаження на обробку з CPU на GPU. Nvvideo4linux2 підтримує різні відеоформати і може значно покращити продуктивність завдань декодування відео в пайплайнах GStreamer на підтримуваному обладнанні NVIDIA.

NvVideoConvert - виконує ефективне апаратно прискорене перетворення формату відео, масштабування та перетворення кольорового простору на платформах NVIDIA. Його часто використовують у пайплайнах GStreamer для обробки та перетворення кадрів відео в формати, сумісні з іншими елементами, використовуючи можливості GPU для зняття навантаження на обробку з CPU та покращення загальної продуктивності.

NvStreamMux - використовується для пакетування декількох відеопотоків у єдиний вихідний потік для ефективної паралельної обробки в AI та глибоких навчальних застосунках. Він розроблений для роботи з обладнанням NVIDIA і допомагає оптимізувати продуктивність пайплайнів відеоаналітики шляхом консолідації декількох вхідних потоків для одночасної обробки наступними елементами, такими як нейронні мережі або інші компоненти обробки відео.

NvInfer – один з найголовніших плагінів DeepStream SDK, призначений для виконання апаратно прискореного інференсу за допомогою TensorRT. NvInfer дозволяє безперешкодно інтегрувати AI моделі, такі як нейронні мережі

для виявлення об'єктів або класифікації зображень, в пайплайни GStreamer, дозволяючи ефективну відеоаналітику та обробку в реальному часі на платформах NVIDIA. NvInfer, важливий компонент Nvidia DeepStream SDK, є ключем до інтеграції глибокого навчання в обробку відео. NvInfer дозволяє інтегрувати попередньо навчені моделі глибокого навчання, такі як нейронні мережі для виявлення об'єктів, класифікації зображень чи аналізу поведінки, безпосередньо в пайплайни GStreamer. Використання TensorRT дозволяє NvInfer оптимізувати моделі для виконання на GPU, значно підвищуючи швидкість обробки та ефективність інференсу. NvInfer підтримує налаштування через конфігураційні файли, дозволяючи користувачам легко змінювати параметри моделі, включаючи вхідні розміри, класи об'єктів, пороги впевненості тощо. Інтеграція з іншими плагінами GStreamer, такими як NvDsOSD для наекранного відображення, дозволяє візуалізувати результати інференсу безпосередньо на відео.

NvDsAnalytics - призначений для виконання аналітики регіонів інтересу (ROI) та генерації метаданих на основі користувацьки визначених правил. Він дозволяє обробляти та аналізувати відеодані в реальному часі, дозволяючи застосуванням, таким як підрахунок об'єктів, виявлення перетину лінії або виявлення присутності в певних регіонах відеокадру.

NvDsOSD - використовується для рендерингу тексту, обмежувальних рамок та інших графічних накладень на кадрах відео. Зазвичай використовується у відеоаналітиці та AI застосунках, NvDsOSD може відображати інформацію, таку як мітки об'єктів, оцінки виявлення або користувацькі метадані на відеовиводі, допомагаючи візуалізувати результати обробки, виконаної іншими елементами в пайплайні, такими як NvInfer або NvDsAnalytics.

На рисунку 1.16 зображено приклад пайплайну Deepstream, який використовує вище перечислені плагіни для обробки та детекції кадрів з відео.

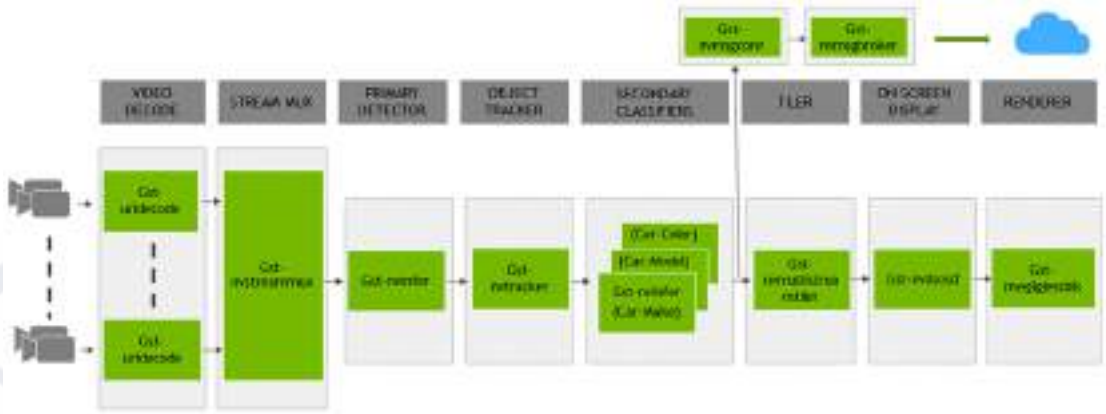


Рисунок 1.16 – Приклад Deepstream пайплайну

У сфері комп’ютерного зору та відеоаналітики розробники та інженери часто стикаються з вибором між DeepStream і OpenCV. Обидва є потужними інструментами з власними перевагами.

OpenCV, скорочення від Open Source Computer Vision Library, є широко використовуваною бібліотекою комп’ютерного зору та машинного навчання з відкритим кодом. Він надає величезну колекцію інструментів і функцій для аналізу зображень і відео, включаючи виявлення об’єктів, обробку зображень, виділення функцій тощо. OpenCV популярний серед дослідників комп’ютерного зору та розробників завдяки своїй універсальності та широкій підтримці спільноти.

Однією з найважливіших переваг DeepStream перед OpenCV є його вбудована підтримка прискорення GPU. DeepStream оптимізовано для використання потужності графічних процесорів NVIDIA, що забезпечує значно вищу швидкість обробки завдань аналізу відео. Це прискорення графічного процесора має важливе значення для додатків у реальному часі, де критично важливі низька затримка та висока пропускна здатність.

Графічні процесори NVIDIA відомі своїми можливостями паралельної обробки, і DeepStream повною мірою використовує цю перевагу, перекладаючи інтенсивні обчислювальні завдання на GPU. Це включає в себе висновок про глибоке навчання, який можна виконувати з надзвичайною швидкістю на

графічних процесорах NVIDIA, що дає змогу запускати складні моделі в режимі реального часу.

Навпаки, OpenCV, будучи універсальною бібліотекою, не має вбудованої підтримки GPU для багатьох своїх функцій. Користувачам часто доводиться вдаватися до додаткових бібліотек або власних реалізацій GPU, щоб досягти прискорення GPU за допомогою OpenCV. Це може бути трудомістким і складним процесом, особливо для розробників, які не знайомі з програмуванням GPU.

Для додатків, які вимагають аналізу відео в реальному часі, таких як виявлення об’єктів у системах спостереження або автономних транспортних засобах, прискорення GPU, яке забезпечує DeepStream, кардинально змінює правила. Це дозволяє обробляти кілька відеопотоків одночасно без шкоди для продуктивності.

DeepStream вирізняється бездоганною інтеграцією з популярними фреймворками глибокого навчання, включаючи TensorFlow, PyTorch і ONNX. Ця інтеграція полегшує розгортання та запуск моделей глибокого навчання для таких завдань, як виявлення об’єктів, класифікація та сегментація.

Глибоке навчання революціонізувало комп’ютерне зір, забезпечивши високоточний і контекстно-залежний аналіз візуальних даних. DeepStream використовує потужність глибокого навчання, дозволяючи розробникам легко включати попередньо підготовлені моделі або навчати власні моделі, використовуючи бажану структуру глибокого навчання.

І хоча OpenCV забезпечує підтримку фреймворків глибокого навчання, його інтеграція не така глибока та бездоганна, як у DeepStream. Використання моделей глибокого навчання в OpenCV часто вимагає додаткових зусиль і коду, а продуктивність може бути не такою оптимізованою, як при використанні вбудованої підтримки глибокого навчання DeepStream.

Ця перевага DeepStream особливо важлива в додатках, де дуже важлива висока точність виявлення об’єктів або розпізнавання зображень.

DeepStream розроблено для одночасної обробки кількох відеопотоків, що робить його ідеальним для додатків, які вимагають одночасної обробки каналів з кількох камер або датчиків. Ця багатопотокова здатність має вирішальне значення для таких сценаріїв, як відеоспостереження на великій території, де потрібно аналізувати декілька камер паралельно.

1.5 Контейнеризація сервісу

Контейнеризація - це процес розгортання програмного забезпечення, який упаковує код додатку разом з усіма файлами та бібліотеками, необхідними для його запуску на будь-якій інфраструктурі.

Docker - це широко використовувана платформа для контейнеризації. Вона дозволяє створювати, розгортати та керувати додатками в легких, портативних контейнерах. Ці контейнери упаковують код додатку, його залежності та конфігурацію, забезпечуючи стабільну продуктивність у різних середовищах. Docker став невід'ємною частиною сучасної розробки програмного забезпечення, забезпечуючи ефективні, масштабовані та ізольовані середовища для додатків, що робить їх легшими у розробці, розгортанні та підтримці.

Архітектура Docker складається з декількох компонентів, які спільно працюють, щоб забезпечити контейнеризацію та управління додатками. На рисунку 1.17 зображена архітектура Docker, яка ділиться на три основні частини: клієнт (Client), хост Docker (Docker host) і реєстр Docker (Registry).

- Клієнт (Client):
    - Docker build – команда, що створює зображення Docker з Dockerfile.
    - Docker pull – команда, яка завантажує зображення Docker з реєстру.
    - Docker run – команда, яка запускає контейнер з зображення Docker.

- Хост Docker (Docker host):
    - Docker Daemon – фоновий сервіс, який керує життєвим циклом контейнерів Docker, включаючи створення, запуск та розподіл.
    - Containers – контейнери, які є запущеними інстанціями зображень Docker.
    - Images – зображення Docker, які є шаблонами для створення контейнерів.
- Реєстр Docker (Registry):
    - Сховище, де зберігаються зображення Docker

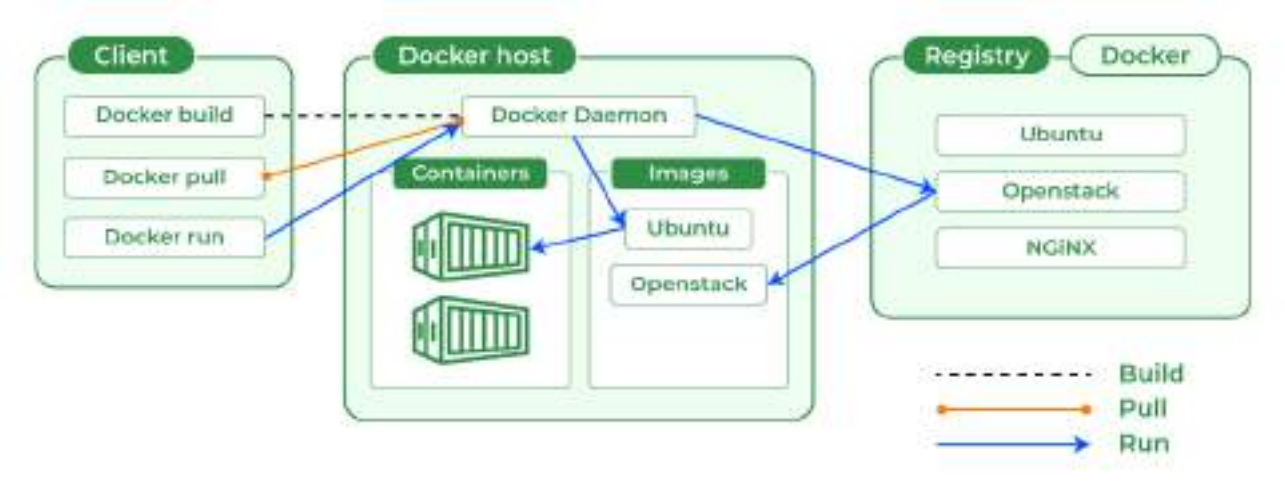


Рисунок 1.17 – Архітектура Docker

Життєвий цикл контейнера Docker складається з кількох етапів, які визначають, як контейнер створюється, виконується, управляється та в кінцевому підсумку припиняє своє існування. На рисунку 1.18 зображено основний цикл контейнера.

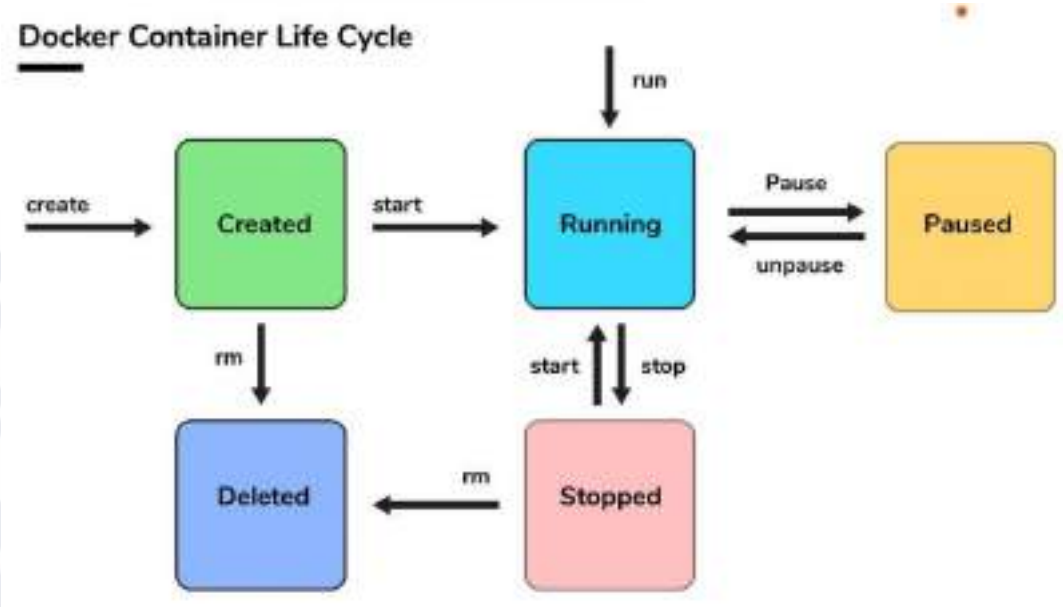


Рисунок 1.18 – Основний цикл життя docker контейнеру

Життєвий цикл починається із створення зображення Docker. Зображення — це шаблон тільки для читання, який містить код програми, бібліотеки, залежності та конфігурації. Зображення зазвичай створюються з Dockerfile — набору інструкцій, який визначає вміст зображення.

Контейнер є інстанцією зображення Docker. Для створення контейнера ви використовуєте команду docker run. На цьому етапі Docker бере зображення та ініціалізує середовище контейнера, налаштовує файлову систему контейнера, мережу та будь-яку зазначену конфігурацію.

Як тільки контейнер ініціалізовано, він переходить до фази виконання. Програма всередині контейнера починає працювати, і контейнер може виконувати свої передбачені функції. Він може взаємодіяти з іншими контейнерами, системою хоста та зовнішніми мережами.

Під час фази виконання можна моніторити та керувати продуктивністю контейнера, логами та використанням ресурсів. Docker надає різні команди та інструменти для інспектування та контролю запущених контейнерів, таких як docker ps, docker logs та docker stats.

Контейнери часто є безстанними, і якщо потрібно внести зміни до програми або її конфігурації, можна створити нове зображення. Потім варто

зупинити та видалити існуючий контейнер та створити новий із оновленого зображення. Це підтримує практики безперервної інтеграції та розгортання.

Контейнери можна призупинити та відновити, використовуючи команди docker pause та docker unpause, які тимчасово заморожують та відновлюють процеси контейнера без його знищення. Це може бути корисним для технічного обслуговування або налагодження.

Коли є потреба зупинити контейнер, варто використоти команду docker stop. Вона надсилає сигнал головному процесу контейнера, щоб ініціювати граціозне завершення. Потім контейнер переходить у зупинений стан, але його дані та конфігурація зберігаються.

Якщо більше не потрібен контейнер, можна видалити його за допомогою команди docker rm. Ця команда видаляє контейнер, звільняючи системні ресурси. Але варто бути обережним при видаленні контейнерів, оскільки будь-які дані або зміни, зроблені в контейнері, втрачаються.

Docker дозволяє перезапускати контейнери за допомогою команди docker restart. Це зручний спосіб перезапустити контейнер з тією самою конфігурацією після його зупинки.

Варто також зазначити можливість використання Docker-compose. Концептуально Docker-compose базується на ідеї, що кожен сервіс у додатку може бути розгорнутий у власному контейнері. Це створює чисте розділення відповідальності та сприяє більшій модульності та масштабованості. Замість використання складних скриптів або ручного керування кожним контейнером індивідуально, Docker-compose дозволяє визначити всю інфраструктуру як код у файлі YAML. Цей файл описує сервіси, мережі та томи, необхідні для вашого додатку, та як вони мають взаємодіяти.

Перевага такого підходу в тому, що він забезпечує однорідність та передбачуваність розгортання. Незалежно від того, чи розгортається додаток на локальній машині, тестовому сервері чи в продакшн-середовищі, можна бути впевнені, що він працюватиме однаково.

Використання Docker-compose також вносить значні спрощення у процес управління контейнерами. Через набір простих команд можна побудувати всі docker зображення, запустити або зупинити всі сервіси, переглянути статус та логи кожного контейнера. Це робить Docker-compose ідеальним інструментом для розробки, тестування та навіть для маломасштабних продакшн-розгортань.

Ключовою характеристикою Docker-compose є його здатність до швидкого масштабування. Він дозволяє масштабувати певний сервіс, збільшуючи або зменшуючи кількість реплік контейнера, що виконується, однією командою. Це особливо корисно у випадках, коли потрібно швидко розширити ресурси для обробки збільшеного навантаження або, навпаки, зменшити їх під час зниження навантаження.

Безпека є ще одним аспектом, де Docker-compose внесок є значним. Він дозволяє визначати та автоматизувати безпекові політики на рівні контейнера, забезпечуючи ізоляцію сервісів та мінімізуючи ризики. Мережеві налаштування, такі як закриті та відкриті порти, контролюються через конфігураційний файл, даючи розробникам детальний контроль над тим, як сервіси комунікують між собою та із зовнішнім світом.

У контексті неперервної інтеграції та доставки (CI/CD), Docker-compose виступає як важливий інструмент для автоматизації та оптимізації процесів розгортання. Його інтеграція з популярними CI/CD платформами, такими як Jenkins, GitLab CI та іншими, є досить простою. Це дозволяє автоматично розгортати та тестувати додатки в контейнеризованому середовищі.

Docker-compose не лише полегшує розгортання та управління мікросервісними додатками, але й сприяє більш ефективному та безпечному розвитку, випробуванню та розгортанню додатків.

Висновок до розділу 1

Завершуючи розділ 1, хотілося б підкреслити важливість і актуальність вивченого матеріалу, а також вибору конкретних технологій та набору даних

для досягнення поставленої мети. Цей розділ надає глибокий огляд технологій, пов'язаних із виявленням транспортних засобів на відео, отриманих з дронів, та обгрунтовує вибір моделі YOLO і набору даних UAVDT для подальших досліджень.

Оглянувши еволюцію технологій виявлення об'єктів, відзначено, що протягом останніх десятиліть спостерігається значний прогрес у цій галузі. Від архітектур R-CNN до більш ефективних та швидких архітектур, таких як YOLO, сучасні технології дають можливість розвивати системи реального часу для виявлення об'єктів, включаючи транспортні засоби. Модель YOLO відзначається високою швидкістю роботи та точністю, що робить її ідеальним вибором для обробки великого обсягу відеоданих, отриманих з дронів.

Підбір відповідного набору даних для тренування моделі має вирішальне значення для досягнення високої точності та надійності. Набір даних UAVDT, спеціально створений для роботи з аерофотознімками, надає важливий інструмент для розробки та навчання моделі YOLO. Він містить різноманітні сценарії та умови, що ідеально відображають виклики, з якими стикаються дрони в умовах відкритого повітряного простору. Такий набір даних дозволяє покращити роботу моделі в умовах змінних масштабів, кутів зйому та погодних умов.

Вибір технологій та набору даних, які найкраще відповідають поставленій меті, є критичним етапом в будь-якому науковому дослідженні. Модель YOLO та набір даних UAVDT обрані з урахуванням їхньої ефективності та відповідності завданню виявлення транспортних засобів на відео, знятому з висоти. Цей вибір покладає фундамент для подальших досліджень та застосувань у сферах, де точність та швидкість виявлення транспортних засобів мають вирішальне значення.

РОЗДІЛ 2

РОЗРОБКА СЕРВІСУ ДЕТЕКЦІЙ ТРАНСПОРТНИХ ЗАСОБІВ

2.1 Архітектура сервісу

Автором була складена архітектура сервісу, що зображена на рисунку 2.1. Зображення з камери дрону зчитується за допомогою Deepstream пайплайну. GStreamer: Основою Deepstream є GStreamer, потужний фреймворк для створення медіапайплайнів. Він дозволяє гнучко налаштувати пайплайн обробки зображень та відео. На даному етапі відбувається зчитування, детекція, трекінг, кодування та запис кадрів. Далі інформація про детекцію йтиме в Kafka, та дублюватиметься записом в NoSQL базу даних MongoDB. MongoDB використовується як основне сховище для зберігання інформації про виявлені транспортні засоби. Кожен об'єкт може бути збережений як документ, що містить метадані (час виявлення, тип транспортного засобу, координати обмежувальної рамки тощо). Також база даних використовується для зберігання подій виявлення транспортних засобів, дозволяючи користувачам аналізувати події в часі та визначати патерни трафіку. Kafka буде виступати як система обробки повідомлень для асинхронного обміну даними між компонентами системи. Вона може використовуватися для трансляції подій виявлення в реальному часі. Інтеграція з FastAPI Backend - Kafka може бути налаштована для відправки повідомлень до FastAPI, де події можуть оброблятися для актуалізації стану бази даних, виклику зовнішніх API або тригера інших бізнес-процесів. Backend FastAPI служить для обробки запитів від frontend і керування взаємодією з MongoDB. За допомогою FastAPI можна створювати RESTful endpoints, які дозволяють користувачу фронтенду отримувати дані та виконувати операції, такі як отримання статистики, перегляд виявлених об'єктів та управління користувачами. Фронтенд в свою чергу забезпечує інтерактивний інтерфейс для кінцевих користувачів, дозволяючи їм візуалізувати результати виявлення, управляти параметрами

системи та переглядати аналітичні звіти. Варто також зазначити, що база даних MongoDB, що розгорнута на Nvidia Jetson на дроні, синхронізована з базою MongoDB, яка розгорнута в хмарному середовищі.

.

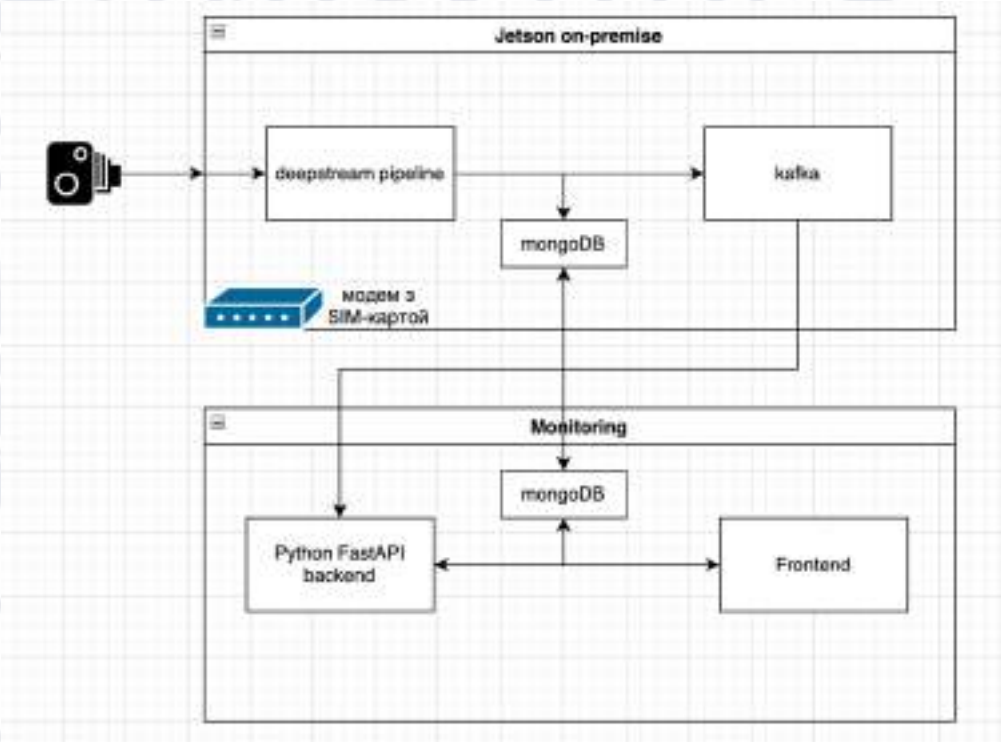


Рисунок 2.1 – Архітектура сервісу

2.2 Налаштування та тренування моделі YOLO на UAVDT датасеті

Оскільки тренування моделі займає багато ресурсів, рекомендується використовувати віртуальні машини або сервери із відеокартами з чіпом Nvidia. Для початку потрібно встановити мову програмування Python та налаштувати віртуальне середовище. Після цього можна перейти до наступного кроку – встановлення Ultralytics Yolo. Для цього завантажимо модуль Ultralytics за допомогою команди pip install.

pip install ultralytics

Важливим етапом, яким не можна нехтувати є обробка даних (датасету UAVDT). Дані з UAVDT можуть мати різні формати та розміри, які не відповідають стандартам, необхідним для тренування моделей глибокого навчання. Обробка даних дозволяє перетворити ці зображення та мітки у стандартизований формат, що сприятиме ефективному навчанню. Нерідко знімки, зроблені з БПЛА, мають варіації у якості через погодні умови, освітлення, кути зйомки тощо. Попередня обробка може включати поліпшення якості зображення, корекцію освітлення, видалення шуму, що підвищує точність навчання моделі. Зображення з різних джерел можуть мати різні розміри та пропорції. Масштабування зображень до єдиного розміру допомагає моделі краще вчитися на даних, а нормалізація піксельних значень сприяє швидшому та стабільнішому навчанню. Для тренування моделей виявлення об'єктів мітки повинні бути відформатовані в специфічний для моделі формат (формат YOLO). Це може включати перетворення координат об'єктів та їх категорій. Також для ефективного навчання та оцінки моделі дані зазвичай ділять на тренувальні, валідаційні та тестові набори. Це дозволяє моделі вчитися на одному наборі даних, а потім перевіряти її ефективність на невідомих даних. Обробка даних може включати техніки аугментації, які збільшують різноманітність даних шляхом внесення змін, таких як обертання, масштабування, або зміна перспективи. Це допомагає уникнути перенавчання моделі на конкретні характеристики тренувального набору даних.

Для датасету UAVDT потрібно зробити три речі, перед початком навчання: скопіювати всі зображення в одну папку, оскільки вихідні зображення зберігаються в кількох папках, перетворити формат міток, а також зберегти всі файли текстових міток в одній папці та розділити дані на тренувальні та валідаційні.

Для того, щоб перетворити формат міток, потрібно зчитати кожен файл міток в датасеті, перетворити формат обмежувальної рамки з [x1, y1, w, h] на [xc, yc, w, h] (рис. 2.2)

```
org_xc = int(line_ls[2]) + int(line_ls[4])/2
org_yc = int(line_ls[3]) + int(line_ls[5])/2
org_w = int(line_ls[4])
org_h = int(line_ls[5])
```

Рисунок 2.2 – Перетворення координат обмежувальної рамки на yolo формат

Також необхідним етапом буде конвертування масштабу координат з розміру зображення до [0, 1], тобто нормалізувати координати до відносної величини. Для цього кожна точка координати ділиться на кількість пікселей ширини або довжини зображення (наприклад, у / 1080). Останнім етапом в перетворенні міток буде видалення неправильних розмічених міток. Деякі з міток в датасеті є надто великими, тож якщо мітка має розмір більше ніж 16.6% (1/6 зображення), можна вважати цю мітку неправильно розміченою, оскільки кут зображень та масштаб не дозволяє об’єкту бути 1/6 та більше зображення. Після даних кроків, можна записувати мітки в новий файл, який буде використовуватись YOLO (рис. 2.3).

```
0 0.23681640625 0.4537037037037037 0.0419921875 0.04814814814814815
0 0.376953125 0.4638888888888889 0.037109375 0.05740740740740741
0 0.56689453125 0.30648148148148147 0.0302734375 0.04259259259259259
0 0.68701171875 0.30925925925925923 0.0283203125 0.03333333333333333
0 0.75439453125 0.26944444444444443 0.0419921875 0.05740740740740741
0 0.443359375 0.5342592592592592 0.03515625 0.06481481481481481
0 0.52880859375 0.5722222222222222 0.0361328125 0.07777777777777778
0 0.5458984375 0.5453703703703704 0.033203125 0.07222222222222222
```

Рисунок 2.3 – Новий формат міток для навчання YOLOv8

Оскільки у датасеті 40 тисяч зображень, потрібно поділити їх на тренувальну та валідаційну вибірку. Була обрана оптимальна пропорція 35 тисяч тренувальних та 5 тисяч валідаційних.

Перед тренуванням обов’язковим є створення файлів конфігурацій для даних та моделі. Файли YML для YOLO моделі використовуються для

конфігурації та налаштування різних аспектів навчання та виявлення об'єктів. YML файли можуть містити деталі конфігурації моделі, такі як архітектура мережі, розміри шарів, фільтри тощо. Це дає змогу змінювати структуру моделі без необхідності зміни коду. Також вони можуть містити інформацію про гіперпараметри для навчання, як-от швидкість навчання, розмір пакету (batch size), кількість епох, втрати (loss functions), оптимізатор тощо. Це дозволяє легко експериментувати з різними налаштуваннями для оптимізації процесу навчання. Налаштування, які впливають на процес виявлення об'єктів, також можуть бути визначені в YML файлах. Ці налаштування включають в себе пороги впевненості (confidence thresholds), пороги перекриття (IoU thresholds) та інші параметри, які впливають на точність та швидкість виявлення.Також конфігураційні файли можуть визначати шляхи до тренувальних, валідаційних та тестових даних, шляхи до файлів міток. Це забезпечує організацію та легкий доступ до датасетів.

Для датасету UAVDT достатньо буде створити конфігураційний файл із даними про шлях до датасету, шлях до тренувальної та валідаційної вибірки, кількість класів та назву класу. На рисунку 2.4 зображено приклад конфігурації датасету для навчання YOLO моделі.

```
1  path: ../dataset
2  train: images/train
3  val: images/val
4
5  nc: 1
6  names: [ 'car' ]
```

Рисунок 2.4 – Приклад конфігурації датасету

Для конфігурації моделі знадобиться більше параметрів. Даний файл, визначає архітектуру та параметри моделі для виявлення об'єктів.

- nc: 80: Кількість класів для виявлення. У цьому випадку, 80 класів.

- depth_multiple: 0.67: Множник глибини моделі. Використовується для масштабування глибини (кількості шарів) моделі.
- width_multiple: 0.75: Множник ширини шарів. Використовується для масштабування кількості каналів у шарах моделі.
- anchors: Якорі, використовувані для виявлення об'єктів. Задаються різні розміри анкерів для різних рівнів вирізання (P3/8, P4/16, P5/32).

Параметри хребта моделі:

- Визначають послідовність шарів, що формують основу моделі.
- Кожен елемент масиву містить [від, кількість, модуль, аргументи].
- Focus, Conv, C3, SPP - різні типи шарів.

Наприклад, [-1, 1, Focus, [64, 3]] означає шар Focus з 64 каналами та розміром ядра 3, що приймає дані від останнього шару (-1).

Голова моделі:

- Визначає шари для виявлення та класифікації об'єктів.
- Включає конволюційні шари, шари об'єднання, та шари для підвищення роздільної здатності.
- Шари, як nn.Upsample, збільшують роздільну здатність карти ознак.
- Шари C3 - це модулі з кількома конволюційними шарами для формування вирізань різних розмірів.
- Detect є шаром, що відповідає за виявлення об'єктів з заданими анкерами та кількістю класів.

Кожен шар або блок у файлі YAML має специфічні параметри та налаштування, які визначають, як дані будуть оброблятися та які ознаки будуть екстрагуватися для виявлення об'єктів. Файл налаштований для моделі середнього розміру (yolov8m), збалансованої для ефективності та точності (рис. 2.5).

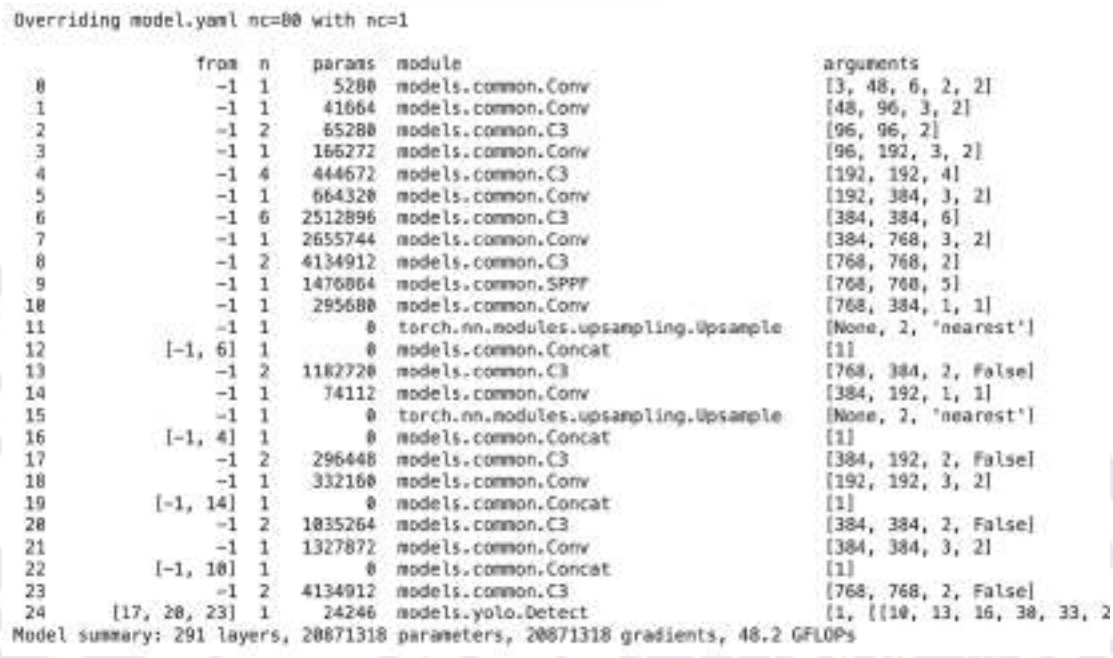
```yaml
nc: 80
depth_multiple: 0.67
width_multiple: 0.75
anchors:
  - [10,13, 16,30, 33,23]
  - [30,61, 62,45, 59,119]
  - [116,90, 156,198, 373,326]

backbone:
  [[-1, 1, Focus, [64, 3]],
   [-1, 1, Conv, [128, 3, 2]],
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],
   [-1, 9, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, C3, [1024, False]],
  ]
```

Рисунок 2.5 – Конфігурація моделі yolov8m

Після створення параметрів конфігурації можна розпочинати тренування моделі. Вкажемо в аргументах кількість епох (epoch) 5 та кількість батчів (batch) 16. За допомогою команди нижче відбувається запуск

python train.py --img 640 --batch 16 --epochs 5 --data UAVDT.yaml --weights yolov5m.pt

Після запуску команди, модель почала тренування на UAVDT датасеті. Нижче на рисунку 2.6 можна побачити параметри моделі, які використовувались для тренування.



```
Overriding model.yaml nc=80 with nc=1

                 from  n    params  module                                  arguments
  0                -1  1      5280  models.common.Conv                      [3, 48, 6, 2, 2]
  1                -1  1     41664  models.common.Conv                      [48, 96, 3, 2]
  2                -1  2     65280  models.common.C3                        [96, 96, 2]
  3                -1  1    166272  models.common.Conv                      [96, 192, 3, 2]
  4                -1  4    444672  models.common.C3                        [192, 192, 4]
  5                -1  1    664320  models.common.Conv                      [192, 384, 3, 2]
  6                -1  6   2512896  models.common.C3                        [384, 384, 6]
  7                -1  1   2655744  models.common.Conv                      [384, 768, 3, 2]
  8                -1  2   4134912  models.common.C3                        [768, 768, 2]
  9                -1  1   1476864  models.common.SPPF                      [768, 768, 5]
 10                -1  1    295680  models.common.Conv                      [768, 384, 1, 1]
 11                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 12          [-1, 6]  1         0  models.common.Concat                    [1]
 13                -1  2   1182720  models.common.C3                        [768, 384, 2, False]
 14                -1  1     74112  models.common.Conv                      [384, 192, 1, 1]
 15                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 16          [-1, 4]  1         0  models.common.Concat                    [1]
 17                -1  2    296448  models.common.C3                        [384, 192, 2, False]
 18                -1  1    332160  models.common.Conv                      [192, 192, 3, 2]
 19          [-1, 14]  1         0  models.common.Concat                    [1]
 20                -1  2   1035264  models.common.C3                        [384, 384, 2, False]
 21                -1  1   1327872  models.common.Conv                      [384, 384, 3, 2]
 22          [-1, 10]  1         0  models.common.Concat                    [1]
 23                -1  2   4134912  models.common.C3                        [768, 768, 2, False]
 24      [17, 20, 23]  1     24246  models.yolo.Detect                      [1, [[10, 13, 16, 38, 33, 2
Model summary: 291 layers, 20871318 parameters, 20871318 gradients, 48.2 GFLOPs
```

Рисунок 2.6 – Параметри моделі під час тренування

На рисунку 2.7 відображено графіки, які ілюструють результати тренування моделі YOLO протягом певної кількості ітерацій (епох).

Тренувальні метрики (верхній ряд):

- Box: Значення помилки прямокутників об'єктів зменшується з часом, що свідчить про те, що модель стає кращою у прогнозуванні точних координат прямокутників.
- Objectness: Показник, який вимірює, наскільки добре модель впевнена у тому, що в прямокутнику є об'єкт, також покращується, оскільки помилка зменшується.
- Classification: Помилка класифікації, яка вимірює, наскільки добре модель класифікує об'єкти, зменшується, свідчить про покращення точності класифікації.
- Precision: Точність моделі збільшується протягом тренування, що означає вищу ймовірність того, що виявлені об'єкти є коректними.
- Recall: Покриття моделі також покращується, оскільки більше реальних об'єктів виявляються правильно.

Валідаційні метрики (нижній ряд):

- val Box: Помилка прямокутників на валідаційному наборі даних також зменшується, хоча і з певною мірою шуму, свідчить про загальну стабільність моделі.
- val Objectness: Валідаційна помилка впевненості в об'єкті зменшується, що вказує на здатність моделі впевнено виявляти об'єкти.
- val Classification: Валідаційна помилка класифікації також зменшується, хоча має деякі сплески, що можуть вказувати на варіації у валідаційному наборі даних.
- mAP@0.5: Середня точність прогнозування (mAP) при порозі перекриття 0.5 значно покращується на валідаційних даних, що є позитивним показником.
- mAP@0.5:0.95: Середня точність прогнозування при порозі перекриття від 0.5 до 0.95 (середнє по діапазону порогів) також покращується, хоча і з більшим шумом, що свідчить про загальну надійність моделі.

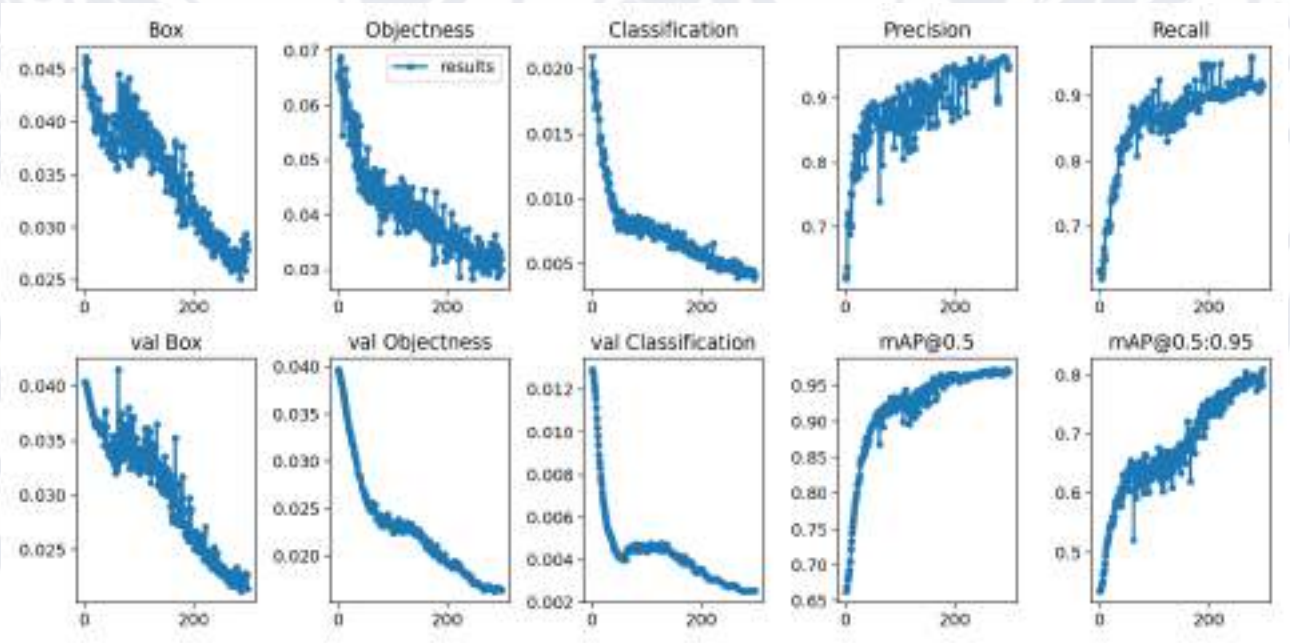


Рисунок 2.7 – Графіки тренування моделі протягом епох

Загалом, графіки вказують на успішне тренування моделі з поступовим покращенням основних показників, як на тренувальному, так і на валідаційному наборах даних. Незважаючи на деякий шум, який є звичайним для реальних даних, тенденції є позитивними.

Для тестової вибірки оберемо зображення з датасету VisDrone. На рисунку 2.8 зображено результат розпізнавання моделі.

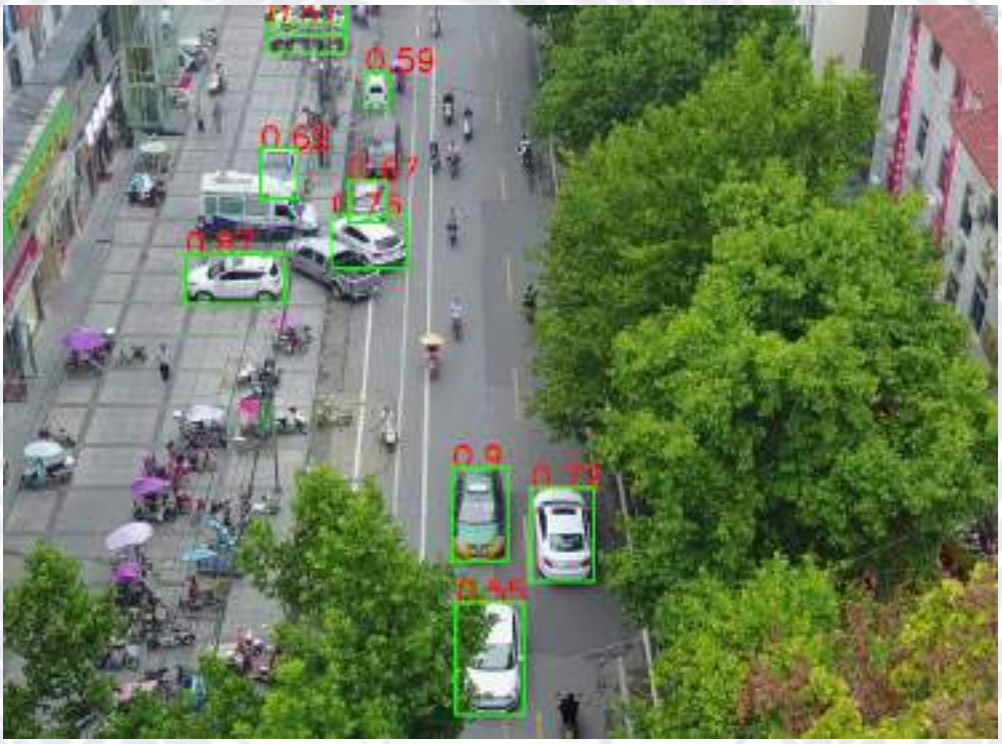


Рисунок 2.8 – Результат розпізнавання навченої моделі

2.3 Оптимізація моделі для сервісу

Оптимізація нейронних мереж, таких як YOLO (You Only Look Once), за допомогою обрізання та квантування, є необхідною для впровадження в умовах обмежених ресурсів, наприклад у вбудованих системах або для реалізації в реальному часі за допомогою NVIDIA DeepStream SDK.

2.3.1 Обрізання (Pruning)

Обрізання полягає у зменшенні кількості параметрів у нейронній мережі, які не мають значного впливу на результат. Це робиться шляхом послідовного

видалення менш важливих нейронів або зв'язків у мережі за певними критеріями, наприклад, абсолютним значенням ваг. Обрізання поділяється на структуроване, неструктуроване та ітеративне.

Структуроване Обрізання - цей метод видаляє цілі канали або фільтри, що може призвести до значного зниження вимог до обчислень і часто більш зручно для обладнання.

Неструктуроване Обрізання - також відоме як обрізання ваг, це видаляє окремі ваги. Хоча це може привести до створення розрідженої моделі, це корисно для зберігання та пам'яті.

Ітеративне Обрізання: Поступовий підхід до обрізання, де мережа обрізається та перенавчається кілька разів для підтримання точності.

Обрізання зменшує розмір моделі та кількість обчислень, що може допомогти досягти реального часу продуктивності під час виконання на NVIDIA's DeepStream SDK, який використовується для потокового аналізу.

2.3.2 Квантування (Quantization):

Квантування зменшує точність чисел, що використовуються для представлення ваг моделі, та, за бажанням, її активацій. Це знижує як вимоги до пам'яті, так і обчислювальні витрати на запуск нейронної мережі.

Квантування після навчання: застосовується після того, як модель була навчена. Воно перетворює ваги та функції активації з чисел з плаваючою комою на цілі числа з нижчою точністю, такі як int8.

Тренування з Урахуванням Квантування. Полягає в навчанні мережі з симульованими вагами низької точності, що може привести до вищої точності, ніж квантування після навчання, оскільки процес навчання адаптується до зменшеної точності.

Квантування особливо ефективно для розгортання на NVIDIA GPU з Tensor Cores, які можуть виконувати операції int8 та int16 набагато швидше, ніж операції з плаваючою комою.

2.3.3 Розгортання за допомогою DeepStream SDK:

Для розгортання обрізаної та квантованої моделі YOLO на DeepStream SDK для інференсу в реальному часі, потрібно виконати такі кроки:

1. Обрізати та квантувати Модель. Застосувати техніки обрізання та квантування для зменшення розміру та обчислювальної складності моделі.

2. Перетворення в TensorRT. Перетворити обрізану та квантовану модель в двигун TensorRT. TensorRT - це SDK для високопродуктивного інференсу глибокого навчання, який включає підтримку int8 та змішаної точності, що робить його ідеальним для запуску оптимізованих моделей.

3. Інтеграція в DeepStream: Інтегрувати двигун TensorRT в додаток DeepStream. DeepStream забезпечує фреймворк для створення та розгортання масштабованих та високопродуктивних AI-додатків для відеоаналітики.

Використовуємо інструмент конвертації, щоб перетворити модель у формат ONNX, який може бути імпортований в TensorRT за допомогою команди:

python3 export.py --weights ./weights/yolov8m.pt --img 640 --batch 1 –simplify

Наступним кроком запустимо TensorRT для оптимізації та квантування моделі. Можна використовувати trtexec, який є утилітою командного рядка для TensorRT.

trtexec --onnx=yolov8m.onnx --saveEngine=yolov8m.engine --explicitBatch --int8

Ця команда перетворює ONNX файл yolov8m.onnx у TensorRT двигун yolov8m.engine з використанням INT8 квантування.

Поєднання обрізання, квантування та розгортання через екосистему NVIDIA, що включає TensorRT та DeepStream SDK, може дозволити YOLO

ефективно працювати в реальному часі на відповідних NVIDIA GPU, роблячи його придатним для таких застосувань, як спостереження, моніторинг трафіку та інших сценаріїв, де критично важливий аналіз в реальному часі.

2.4 Інтеграція з Deepstream SDK

Для розробки сервісу інференсу на базі Deepstream SDK, потрібно для початку скласти пайплайн, що складається з різних допоміжних плагінів. На рисунку 2.9 зображена схему пайплайну NVIDIA Deepstream SDK, яка показує послідовність обробки відео потоку.

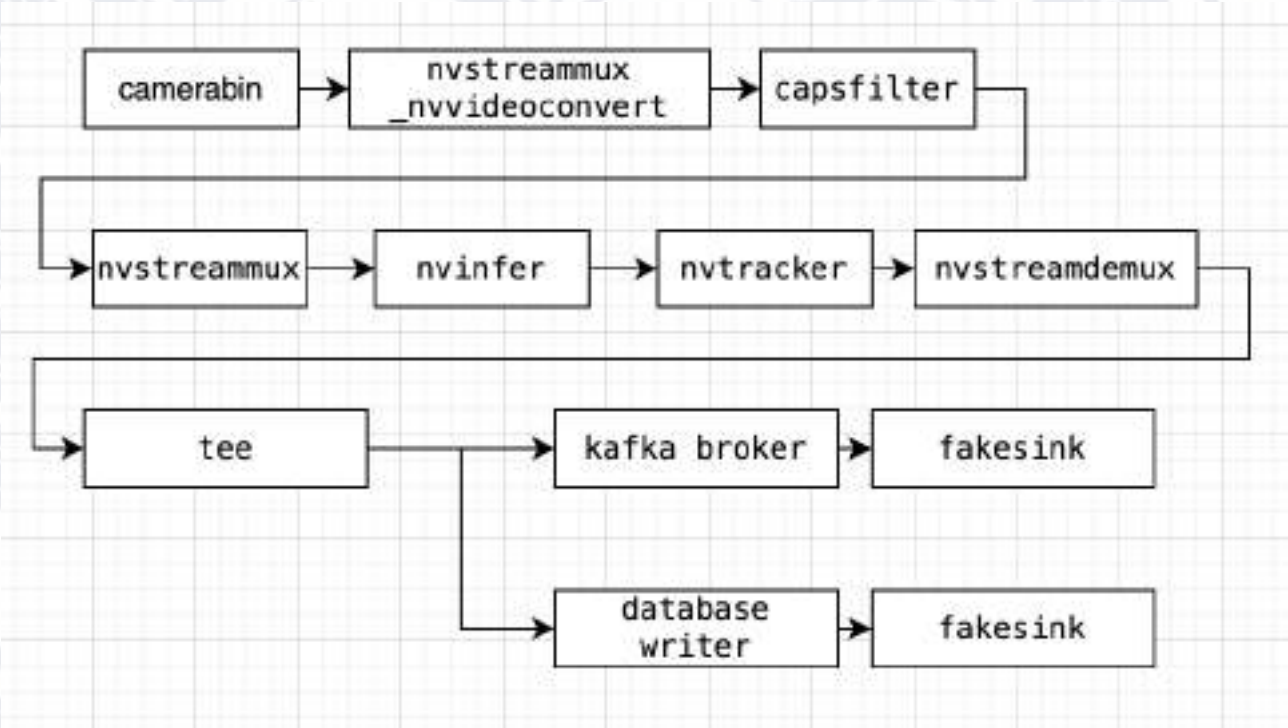


Рисунок 2.9 – Схема пайплайну Deepstream SDK

Розглянемо кожен плагін в даному пайплайні.

camerabin: Цей плагін захоплює відеопотік із камери відеодрону. Оскільки камер може бути декілька, є сенс додати стільки паралельних плагінів camerabin, скільки камер розміщено на дроні.

nvstreammux (nvv4l2decoder може бути використаний для декодування): nvstreammux об'єднує кілька відеопотоків у єдиний батч для одночасної обробки. У пайплайні він використовується для синхронізації та мультиплексування відеоданих перед подачею їх до nvinfer.

nvvideoconvert: Конвертує формати відео до форматів, які прийнятні для обробки іншими плагінами в пайплайні, змінюючи колірну схему або розмір кадру.

capsfilter: Використовується для встановлення або фільтрації специфічних параметрів відеопотоку, як-от розмір кадру, формат пікселів, кадрову частоту тощо.

nvinfer: Це плагін глибокого навчання, який виконує інференс на відеопотоці, використовуючи попередньо навчену модель – YOLOv8m, яка була квантизована та конвертована до формату TensorRT. Загалом, даний плагін – серце пайплайну, оскільки саме він використовується для детекцій транспорту.

nvtracker: Відстежує об'єкти між кадрами. Після того, як об'єкти ідентифіковані за допомогою nvinfer, nvtracker дозволяє відстежувати їх переміщення в часі.

nvstreamdemux: Розділяє об'єднаний потік назад на окремі потоки після обробки, для незалежного відстеження або подальшої обробки.

tee: Розгалужує потік на кілька шляхів, дозволяючи передавати дані до різних плагінів паралельно.

kafka broker: Цей плагін інтегрує вивід пайплайну з Apache Kafka, системою для обробки потокових даних, що дозволяє надсилати оброблені відеодані до сервера Kafka для подальшої обробки або зберігання.

database writer: Користувацький плагін, для запису метаданих детекцій (інференсу) до локальної бази даних.

fakesink: Це "порожній" плагін, який використовується як заглушка. Він споживає дані, але не виводить їх нікуди, тим самим закриваючи пайплайн.

Кожен елемент пайплайну відповідає за певний етап обробки відео, і весь пайплайн спроектований для обробки відеопотоку в реальному часі з високою продуктивністю.

Deepstream SDK та супутній фреймворк GStreamer підтримує мову програмування Python, тому задля простішого читання та використання автор використав дану мову та python-bindings для Gstreamer пайплайну.

Для написання пайплайну, спочатку потрібно ініціалізувати GStreamer за допомогою методу Gst.init(). Наступним етапом буде створення елементів або так званих плагінів (рис. 2.10). Плагіни створюються за допомогою підкласу ElementFactory, який має ключове значення у GStreamer, оскільки він дозволяє гнучко створювати та керувати медіа-елементами, необхідними для будь-якого медіа-пайплайну. Це дозволяє легко інтегрувати різноманітні медіа-операції в пайплайн, навіть з власноруч написаними плагінами.

```
4   camera_bin = Gst.ElementFactory.make("camerabin", "camera_bin")
5   nvstreammux = Gst.ElementFactory.make("nvstreammux", "nvstreammux")
6   capsfilter = Gst.ElementFactory.make("capsfilter", "capsfilter")
7   nvinfer = Gst.ElementFactory.make("nvinfer", "nvinfer")
8   nvtracker = Gst.ElementFactory.make("nvtracker", "nvtracker")
9   nvstreamdemux = Gst.ElementFactory.make("nvstreamdemux", "nvstreamdemux")
10  tee = Gst.ElementFactory.make("tee", "tee")
11  kafka_sink = Gst.ElementFactory.make("kafkasink", "kafka_sink")
12  db_writer = Gst.ElementFactory.make("dbwritersink", "db_writer")
13  fakesink = Gst.ElementFactory.make("fakesink", "fakesink")
```

Рисунок 2.10 – Створення необхідних плагінів

Важливим етапом є налаштування та встановлення властивостей для кожного плагіну (рис. 2.11). Наприклад, плагін camerabin буде очікувати властивість «індекс камери», поставивши який, ми зможемо отримати доступ до цієї камери для зчитування. Плагін nvinfer обов'язково містить встастивості, такі як, шлях до конфігу для моделі.

```
capsfilter.set_property("caps", Gst.Caps.from_string("video/x-raw(memory:NVMM), format=(string)NV12"))
nvstreammux_nvvideoconvert.set_property("interpolation-method", 5)
nvstreammux_nvvideoconvert.set_property("gpu-id", 0)
nvstreammux.set_property("batch-size", 16)
nvinfer.set_property("config-file-path", "config_infer_primary_yoloV8.txt")
```

Рисунок 2.11 – Встановлення властивостей для плагінів

Останнім кроком є створення самого об'єкту пайплайну та зв'язування усіх елементів почергово, як на схемі на рисунку 2.9. Елементи можуть зв'язатись автоматично, якщо передати їх у метод link_many (рис. 2.12).

```
Gst.Element.link_many(camera_bin, *args: nvstreammux, capsfilter,
                      nvinfer, nvtracker, nvstreamdemux, tee)
```

Рисунок 2.12 – Мультиз'єднання елементів

Однак, для з'єднання розгалужених елементів, таких як tee з kafka broker, або елементів з декількаами входами/виходами, потрібно викликати метод link кожного разу, для з'єднання виходу одного елемента з входом іншого (рис. 2.13).

```
tee.link(kafka_sink)
tee.link(db_writer)

fakesink.link(db_writer)
```

Рисунок 2.13 – Окреме з'єднання елементів

Можна вважати, що пайплайн готовий, але одним із завершальних елементів є додавання шини (bus). В GStreamer, шина є важливим компонентом для обробки повідомлень, що генеруються всередині медіа-пайплайну. Шина дозволяє розробникам стежити за подіями, які відбуваються в елементах пайплайну, і реагувати на них відповідно. Шину потрібно обов'язково налаштовувати, аби мати доступ до обробки повідомлень, синхронізувати зовнішні дії та для режиму відладки. Шина перехоплює різні типи повідомлень, такі як помилки, стан готовності до відтворення, кінець потоку (EOS) та інші. Це дозволяє програмі адекватно реагувати на зміни стану або помилки в пайплайні. Через шину можна отримати повідомлення про події, які потребують зовнішньої взаємодії, наприклад, завантаження наступного медіа-файлу після завершення поточного. Шина є корисною для відлагодження, оскільки вона може надавати інформацію про те, що відбувається всередині пайплайну, таку як помилки кодування/декодування або проблеми з мережевими потоками.

Шина створюється автоматично при створенні нового екземпляра пайплайну. На рисунку 2.14 зображена схема спілкування пайплайну та зовнішнього додатку за допомогою шини.

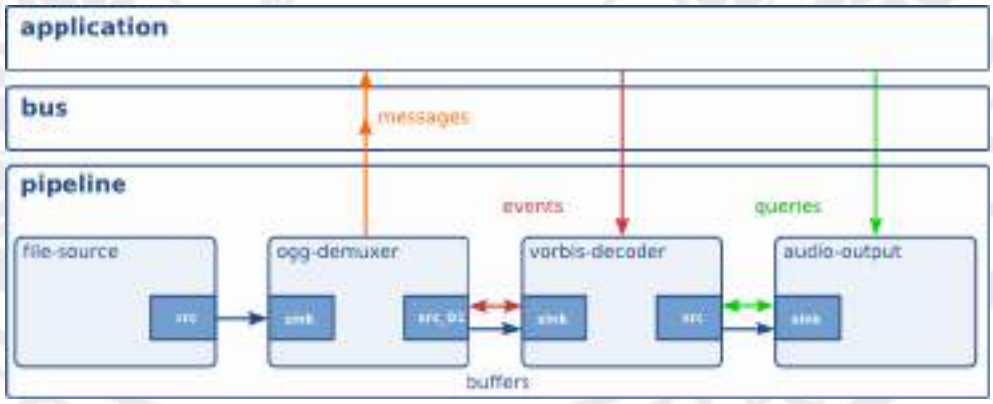


Рисунок 2.14 – Схема взаємодії пайплайну із зовнішним середовищем за допомогою шини

Після того, як шина буде налаштована, а сигнал обробника доданий до шини, пайплайн може бути запущений перемиканням стану пайплайну у режим

PLAYING (програвання). Оскільки GStreamer використовує циклову систему, потрібно запустити також головний цикл для чекання повідомлень. Головний цикл ініціалізований методом MainLoop бібліотеки Glib (рис. 2.15).

```
# Додавання обробника повідомлень до шини
bus.add_signal_watch()
bus.connect("message", bus_call, loop)

# Початок відтворення пайплайну
pipeline.set_state(Gst.State.PLAYING)

# Запуск головного циклу для чекання повідомлень
loop = GLib.MainLoop()
try:
    loop.run()
except KeyboardInterrupt:
    loop.quit()
finally:
    pipeline.set_state(Gst.State.NULL)
```

Рисунок 2.15 – Головний цикл та режими пайплайну

Останніми кроками є налаштування конфігів для nvinfer, nvtracker та інших необхідних плагінів. Для цього потрібно створити конфіг та задати шлях до нього в властивості плагіну в пайплайні. Конфіг nvinfer для моделі yolo може містити такі параметри:

gpu-id: Вказує GPU, яка буде використовуватись. 0 означає перший GPU.

net-scale-factor: Масштабний коефіцієнт для нормалізації вхідних кадрів. 0.0078431372 часто використовується для моделей, натренованих на зображеннях, нормалізованих у діапазоні [0,255].

offsets: Значення віднімання середнього (для кожного RGB каналу), щоб нормалізувати вхідні дані. 127.5;127.5;127.5 свідчить про поширену практику переведення вхідних даних з діапазону [0, 255] у [-1, 1].

model-engine-file: Шлях до оптимізованого для TensorRT файлу моделі.

labelfile-path: Шлях до файлу з мітками для моделі.

batch-size: Кількість кадрів, які обробляються в одній партії. Оптимальним числом є 16 кадрів в партії.

2.5 Kafka події

Kafka є чудовим рішенням у поєднанні з Deepstream пайплайном, оскільки за допомогою цієї системи обміну повідомленнями, можна надсилати результати детекцій в будь-який сервіс. Оскільки дрон має приєднану обчислювальну машину Nvidia Jetson, доцільним буде поєднання Kafka, що розгорнута у хмарному середовищі та брокера-плагіну.

Перед початком надсилання, потрібно налаштувати Kafka. Для цього потрібно встановити сервер у будь-якому хмарному середовищі або використовувати уже існуючі сервери-додатки. Також потрібно скласти шаблон повідомлення, яке буде надсилатись пайплайном. В даній задачі, очікується надсилання координатів обмежувальної рамки, клас об’єкту, дата та час детекції.

Після обробки відеопотоку та отримання результатів детекції, дані надсилаються до Kafka. Kafka брокер, розміщений у хмарному середовищі, отримує дані та розподіляє їх між підписниками. Це можуть бути хмарні аналітичні сервіси, бази даних або інші споживачі, які обробляють ці дані для подальшого аналізу. Kafka забезпечує високу пропускну спроможність та надійність у передачі даних, що є важливим для критичних застосувань. Можливість обробки та передачі даних в реальному часі є ключовою для швидкої реакції на події та прийняття рішень, а легкість інтеграції з різноманітними хмарними сервісами та додатками, дозволяє широко використовувати дані для різних цілей. Для цілей задачі, дані повідомлення з Kafka, будуть оброблені моніторинговими системами backend-frontend.

Повідомлення Kafka, яке надсилається з Deepstream після детекції, зазвичай формату JSON, що робить їх легко читаними та інтегрованими з різними системами (рис. 2.16).

Рисунок 2.16 – Приклад повідомлення від Kafka broker

До Nvidia Jetson також можливо під’єднати GPS та інші датчики, для зчитування даних та передачі на моніторингові сервіси температуру навколишнього середовища, координати локації та ін.

2.6 Реалізація моніторингу REST API

Моніторингова система є важливим сервісом для отримання та обробки даних з сервісу детекцій. Користувачам простіше взаємодіяти з API або користувацьким інтерфейсом, ніж під’єднуватись безпосередньо до логів детекцій. Така система не тільки спрощує процес взаємодії з даними для кінцевих користувачів через API або користувацький інтерфейс, але й значно підвищує ефективність та точність обробки інформації. Простота взаємодії є ключовим фактором успіху будь-якої технологічної системи. API (Application Programming Interface) дозволяє інтегрувати функціонал моніторингової системи в інші додатки, що розширює її використання та доступність.

Для сервісу моніторингу автором був обраний фреймворк FastAPI, для мови програмування Python. Поєднання FastAPI з Kafka для потокової передачі подій і базою даних для зберігання даних робить FastAPI потужним бекендом для базового моніторінгу, а в подальшому для новий задач.

Читання подій з Kafka у додатку FastAPI зазвичай використовується бібліотека, aiokafka, яка надає підтримку асинхронності, сумісної з FastAPI. Для початку потрібно ініціалізувати проєкт FastAPI, разом із додаванням базових точок – ендпойнтів, для отримання повідомлень. Посилаючи GET запит на ендпойнт, додаток буде повертати останні повідомлення з детектора та аналізувати їх. Для повернення повідомлень використовується отримувач (consumer), приклад коду якого зображено на рисунку 2.17.

```python
async def consume():
    consumer = AIOKafkaConsumer(
        'my_topic',
        bootstrap_servers='localhost:9092',
        group_id="my-group"
    )
    # Отримання конфігурації кластера і приєднання до групи
    await consumer.start()
    try:
        # Прийом повідомлень
        async for msg in consumer:
            print("отримано: ", msg.topic, msg.partition, msg.offset,
                  msg.key, msg.value, msg.timestamp)
    finally:
        # Вихід з групи споживачів; автоматичне підтвердження, якщо включено.
        await consumer.stop()
```

Рисунок 2.17 – Приклад коду отримувача повідомлень

Також в сервісі моніторингу можна додати обробку даних та їх аналіз, як от фільтрування обмежувальних рамок за певним критерієм або регіоном, аналіз локацій за кількістю об’єктів та складання таблиць кореляцій. Фільтрування рамок за певними критеріями, такими як розмір, форма, кольорова характеристика або рух, може значно підвищити корисність моніторингової системи. За допомогою алгоритму, фільтрування було

налаштовано таким чином, щоб моніторингова система ігнорувала дрібні об'єкти або об'єкти, які залишаються статичними протягом тривалого часу.

Збір та аналіз даних про кількість об'єктів у різних локаціях може бути важливим для розуміння патернів руху, забруднення, безпеки тощо. Виявлення несподівано високої кількості транспорту у певній зоні може бути сигналом для подальшого аналізу даної зони на забруднення повітря. Такий аналіз може також виявити тенденції, які можуть бути використані для планування міських послуг, наприклад, у сфері транспорту або громадської безпеки.

Висновок до розділу 2

Архітектура сервісу, представлена на рисунку 2.1, є комплексним рішенням для ефективного спостереження та аналізу відеоданих з дронів. Використання DeepStream пайплайну, заснованого на потужному фреймворку GStreamer, дозволяє гнучко налаштовувати процеси зчитування, детекції, трекінгу, кодування та запису кадрів. Інтеграція з Kafka та MongoDB надає можливість асинхронного обміну даними та надійного зберігання інформації про виявлені об'єкти, зокрема транспортні засоби, у вигляді метаданих.

Тренування моделі YOLO на датасеті UAVDT вимагає використання віртуальних машин або серверів із GPU Nvidia, що гарантує достатній рівень обчислювальних ресурсів. Процес тренування включає етапи встановлення Python, налаштування віртуального середовища, інсталяції Ultralytics Yolo та ретельної попередньої обробки даних з датасету UAVDT. Такий підхід забезпечує високу якість навчання моделі та ефективність детекції.

З огляду на значний ресурсний вимогу процесу тренування моделі YOLO на датасеті UAVDT, існує потенціал для оптимізації за допомогою використання хмарних обчислень. Використання хмарних платформ з автоматизованими інструментами для масштабування та управління ресурсами може забезпечити більшу гнучкість та ефективність у процесі тренування.

Оптимізація моделі для сервісу здійснюється шляхом обрізання та квантування, що є критично важливим для реалізації в умовах обмежених

ресурсів, наприклад, на вбудованих системах або для роботи в реальному часі в середовищі NVIDIA DeepStream SDK. Така оптимізація дозволяє підвищити ефективність мережі без значної втрати точності.

Додавання більш складних алгоритмів обробки даних, таких як машинне навчання для прогнозування патернів або розпізнавання аномальних поведінкових моделей, може збільшити цінність моніторингового сервісу. Це також включає вдосконалення інтерфейсу користувача для кращого візуального представлення даних.

Покращення продуктивності системи може бути досягнуте за допомогою використання більш потужних обчислювальних платформ, таких як більш продуктивні GPU або обчислювальні кластери. Це дозволить обробляти більший обсяг відеоданих та застосовувати більш складні моделі машинного навчання для покращення точності виявлення.

Загалом, розроблена архітектура сервісу представляє собою ефективне, багатофункціональне рішення, здатне забезпечити високу точність детекції та аналізу даних.

ВИСНОВОК

У даній роботі було досягнуто значних успіхів у розробці та вдосконаленні системи для автоматичного виявлення транспортних засобів на відеозаписах, отриманих з дронів та безпілотних літальних апаратів (UAVs).

Під час дослідження були вирішені важливі завдання, які спрямовані на оптимізацію та покращення ефективності системи.

Одною із головних поставлених задач дослідження була розробка та налаштування системи, яка здатна виявляти транспортні засоби на відеозаписах у реальному часі. Для досягнення цієї мети було використано модель YOLO (You Only Look Once), яка показала високу точність та швидкість виявлення. Використання навченої моделі на наборі даних UAVDT дозволило адаптувати систему до реальних умов та сценаріїв.

Вибір DeepStream як основи для розробки системи був обґрунтований його високою продуктивністю та можливістю реалізації алгоритмів машинного навчання в реальному часі. Контейнеризація системи у Docker дозволила забезпечити її переносимість та легкість розгортання в різних середовищах. Конфігурація системи була оптимізована для досягнення максимальної швидкості та точності виявлення.

У цій роботі успішно розроблено та оптимізовано систему для автоматичного виявлення транспортних засобів на відеозаписах, отриманих з дронів та UAVs. Використання моделі YOLO, навченої на наборі даних UAVDT, дозволило досягти високої точності та швидкості виявлення. Обрані технології, такі як DeepStream, Docker та оптимізована конфігурація, забезпечили стабільну та ефективну роботу системи.

Подальший розвиток та оптимізація алгоритмів автоматичного виявлення може покращити якість роботи системи. Використання більш ефективних алгоритмів для виявлення та відстеження транспортних засобів може зменшити кількість помилкових сигналів та підвищити точність.

Розширення наборів даних для навчання моделей є важливим кроком для покращення системи. Використання різних датасетів та включення

різноманітних сценаріїв може покращити здатність системи до адаптації до різних умов.

СПИСОК ДЖЕРЕЛ

1. 3 Fundamental Reasons why Quantization is important for tinyML. URL: https://towardsdatascience.com/3-fundamental-reasons-why-quantization-is-important-for-tinyml-92df82234b91
2. Artificial neural network. URL: https://en.wikipedia.org/wiki/Artificial_neural_network
3. Apache Kafka. URL: https://uk.wikipedia.org/wiki/Apache_Kafka
4. A Gentle Introduction to Object Recognition with Deep Learning. URL: https://machinelearningmastery.com/object-recognition-with-deep-learning/
5. A Survey on Evaluation Metrics for Object Detection Algorithms. URL: https://arxiv.org/abs/1602.08486
6. Apache Kafka Tutorial for Beginners. URL: https://www.tutorialspoint.com/apache_kafka/apache_kafka_introduction.htm
7. A Comprehensive Guide to RESTful API Design. URL: https://restfulapi.net/
8. DOTA Dataset. URL: https://captain-whu.github.io/DOTA/
9. DeepStream vs OpenCV: Which Video Loader is Faster? URL: https://henrynavarro.org/deepstream-vs-opencv-which-video-loader-is-faster-part-iii-8812348c1e70
10. Docker. URL: https://uk.wikipedia.org/wiki/Docker
11. Docker Compose overview. URL: https://docs.docker.com/compose/
12. Decision tree pruning. URL: https://en.wikipedia.org/wiki/Decision_tree_pruning
13. Docker Compose: A Beginner's Guide. URL: https://www.docker.com/blog/docker-compose-a-beginners-guide/
14. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. URL: https://arxiv.org/abs/1506.01497
15. From R-CNN to Mask R-CNN. URL: https://medium.com/@umerfarooq_26378/from-r-cnn-to-mask-r-cnn-d6367b196cfd
16. FastAPI reference. URL: https://fastapi.tiangolo.com/reference/

17. GStreamer. URL: https://uk.wikipedia.org/wiki/GStreamer
18. Getting Started with FastAPI: A Modern Python Web Framework. URL: https://realpython.com/fastapi-python-web-apis/
19. How single-shot detector (SSD) works? URL: https://developers.arcgis.com/python/guide/how-ssd-works/
20. Introduction to Convolutional Neural Networks (CNNs) for Visual Recognition. URL: https://towardsdatascience.com/introduction-to-convolutions-using-python-and-opencv-5b0908634a47
21. Introduction to Docker for Data Science. URL: https://towardsdatascience.com/introduction-to-docker-for-data-science-195c61ff6c59
22. Introduction to Quantization in Deep Learning. URL: https://www.tensorflow.org/model_optimization/guide/quantization
23. Introduction to Region-based Convolutional Neural Networks (R-CNNs). URL: https://neptune.ai/blog/region-based-convolutional-neural-networks
24. Introduction to NVIDIA Jetson: A Platform for Edge AI. URL: https://www.nvidia.com/en-us/autonomous-machines/jetson-embedded-platforms/
25. Metrics to Evaluate your Machine Learning Algorithm. URL: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234
26. NVIDIA DeepStream SDK. URL: https://developer.nvidia.com/deepstream-sdk
27. Nvidia Jetson. URL: https://en.wikipedia.org/wiki/Nvidia_Jetson
28. OpenCV documentation. URL: https://docs.opencv.org/4.x/d9/df8/tutorial_root.html
29. OpenCV. URL: https://en.wikipedia.org/wiki/OpenCV
30. Object Detection with Deep Learning: A Review. URL: https://towardsdatascience.com/object-detection-with-deep-learning-a-review-517fbe76f4d1

31. Object Detection using Python OpenCV. URL: https://www.analyticsvidhya.com/blog/2021/03/object-detection-using-python-opencv/
32. Object Detection in Images and Videos Using the YOLO Algorithm. URL: https://nanonets.com/blog/object-detection-yolov3-using-python-tensorflow/
33. Pruning in Deep Learning: Overview and Techniques. URL: https://towardsdatascience.com/pruning-in-deep-learning-overview-and-techniques-5253b3e5c84d
34. Quantization in Deep Learning. URL: https://medium.com/@joel_34050/quantization-in-deep-learning-478417eab72b
35. Region Based Convolutional Neural Networks. URL: https://en.wikipedia.org/wiki/Region_Based_Convolutional_Neural_Networks
36. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms. URL: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
37. REST. URL: https://uk.wikipedia.org/wiki/REST
38. Stanford Drone Dataset. URL: https://cvgl.stanford.edu/projects/uav_data/
39. Transfer Learning for Computer Vision Tutorial. URL: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
40. TensorFlow Object Detection API. URL: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/
41. The OpenCV Tutorials. URL: https://docs.opencv.org/master/d9/df8/tutorial_root.html
42. UAV Project. URL: https://sites.google.com/view/grli-uavdt/%E9%A6%96%E9%A1%B5
43. Understanding GStreamer: A Comprehensive Tutorial. URL: https://gstreamer.freedesktop.org/documentation/tutorials/basic/quick-start.html

44. VisDrone-Dataset. URL: https://github.com/VisDrone/VisDrone-Dataset?tab=readme-ov-file
45. What Is Region Of Interest Pooling? URL: https://analyticsindiamag.com/what-is-region-of-interest-pooling/
46. What is Containerization? URL: https://www.checkpoint.com/cyber-hub/cloud-security/what-is-container-security/what-is-containerization/
47. What is Pruning in Machine Learning? URL: https://opendatascience.com/what-is-pruning-in-machine-learning/
48. YOLO — You only look once, real time object detection explained. URL: https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006
49. Виявляння об'єктів. URL: https://uk.wikipedia.org/wiki/Виявляння_об%27єктів
50. Чому я обираю FastAPI: основні можливості та переваги фреймворку. URL: https://dou.ua/forums/topic/37547/
51. Anastasiia Neskorodieva, Maksym Strutovskyi, Artem Baiev, Oleh Vietrov. Real-time Classification, Localization and Tracking System (Based on Rhythmic Gymnastics). URL: https://ieeexplore.ieee.org/document/10310664

Додаток 2 до наказу
від «31» березня 2023 року
№119/05

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)