Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ ВІДСТЕЖЕННЯ ТА ПІДРАХУНКУ ОБ'ЄКТІВ НА ВІДЕО НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

Виконав: студент 4 курсу, групи 3КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Хоцько Б. В.
(прізвище та ініціали)

Керівник: к.т.н., проф. каф.КН
Колесницький О. К.
(прізвище та ініціали)
«04» червня 2025 р.

Рецензент: к.т.н., проф. каф.КСУ
Биков М. М.
(прізвище та ініціали)
«05» червня 2025 р.

Допущено до захисту
Зав. кафедри КН Ірвин А.А.
«06» червня 2025 р.

Вінниця ВНТУ - 2025 рік

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Колесницький О. К., к.т.н., проф. каф. КН		

7. Дата видачі завдання „05" травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області	05.05.25	07.05.25	
2	Обгрунтування методу розв'язання задачі, проектування модуля відстеження та підрахунку об'єктів на відео на основі згорткової нейронної мережі	08.05.25	15.05.25	
3	Програмна реалізація модуля відстеження та підрахунку об'єктів на відео на основі згорткової нейронної мережі	16.05.25	26.05.25	
4	Аналіз результатів тестування модуля відстеження та підрахунку об'єктів на відео на основі згорткової нейронної мережі	27.05.25	01.06.25	
5	Оформлення матеріалів до захисту БКР	02.06.25	04.06.25	

Студент _____ **Хоцько Б. В.**
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ **Колесницький О. К.**
(підпис) (прізвище та ініціали)

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо - професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«05» травня 2025 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Хоцьку Богдану Володимировичу

1. Тема роботи: Програмний модуль відстеження та підрахунку об'єктів на відео на основі згорткової нейронної мережі.
керівник роботи: Колесницький Олег Костянтинович, к.т.н., проф.
затверджені наказом вищого навчального закладу "20" березня 2025 року № 97

2. Термін подання студентом роботи „30" травня 2025 р.

3. Вихідні дані до роботи :
кількість класів відстежуваних об'єктів – не менше 50, вид класифікатора – нейронна мережа, кількість навчальних прикладів – не менше 4000, кількість тестових прикладів – не менше 1000, кількість зон підрахунку об'єктів – не менше 2, використання об'єктно-орієнтованої мови програмування.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
проаналізувати існуючі методи відстеження та підрахунку об'єктів на відео та вибрати найбільш перспективний;
вибрати нейронну мережу та її структуру для модуля відстеження та підрахунку об'єктів на відео;
розробити алгоритм роботи модуля відстеження та підрахунку об'єктів;
спроектувати та програмно реалізувати модуль відстеження та підрахунку об'єктів на відео за допомогою нейронної мережі;
протестувати роботу програми відстеження та підрахунку об'єктів на відео на основі нейронної мережі.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
схема алгоритму роботи програми;
структура нейронної мережі;
результати роботи програми;
оцінка продуктивності роботи програми.

АНОТАЦІЯ

Хоцько Б. В. Програмний модуль відстеження та підрахунку об’єктів на відео на основі згорткової нейронної мережі. Бакалаврська кваліфікаційна робота скаладається з 77 сторінок формату А4, на яких є 33 рисунки, 5 таблиць, список використаних джерел містить 21 найменувань.

Метою роботи є підвищення точності відстеження та підрахунку об’єктів на відео за рахунок використання згорткової нейронної мережі.

У роботі було обґрунтовано вибір типу нейронної мережі для модуля відстеження та підрахунку об’єктів на відео – згорткова нейронна мережа YOLOv8. Також було обгрунтовано вибір трекера Deep SORT, робота якого заснована на використанні відстані Махалонобіса та фільтра Калмана.. Для програмної реалізації модуля було обґрунтовано вибір мови програмування Python та спеціалізованих бібліотек PyTorch, Ultralytics та OpenCV. Розроблений програмний модуль відстеження та підрахунку об’єктів на відео переважає аналог (на основі нейромережі YOLOv8s і трекера ByteTrack) на 4,5% по влучності відстеження кількох об’єктів (MOTP) та на 2,3% по достовірності відстеження кількох об’єктів (MOTA). Тобто мета роботи досягнута – точність відстеження та підрахунку об’єктів на відео підвищена.

Ключові слова: відстеження об’єктів, виявлення об’єктів, підрахунок об’єктів, відео, згорткова нейронна мережа.

ABSTRACT

Khotsko B. V. Software module for tracking and counting objects in video based on a convolutional neural network. The bachelor's qualification paper consists of 77 pages of A4 format, on which there are 33 figures, 5 tables, the list of used sources contains 21 names.

The purpose of the work is to increase the accuracy of tracking and counting objects in video by using a convolutional neural network.

The work justified the choice of the type of neural network for the module for tracking and counting objects in video - a convolutional neural network YOLOv8. The choice of the Deep SORT tracker, whose work is based on the use of the Mahalonobis distance and the Kalman filter, was also justified. For the software implementation of the module, the choice of the Python programming language and specialized libraries PyTorch, Ultralytics and OpenCV was justified. The developed software module for tracking and counting objects in video outperforms its analogue (based on the YOLOv8s neural network and the ByteTrack tracker) by 4.5% in terms of multiple object tracking accuracy (MOTP) and by 2.3% in terms of multiple object tracking reliability (MOTA). That is, the goal of the work has been achieved - the accuracy of tracking and counting objects in video has been increased.

Keywords: object tracking, object detection, object counting, video, convolutional neural network.

ЗМІСТ

ВСТУП

Актуальність досліджень. У сфері комп’ютерного зору задача відстеження та аналізу рухомих об’єктів у реальному часі є дуже актуальною. Відстеження об’єктів – це розширена частина виявлення об’єктів, що надає можливість визначення розташування та класифікацію об’єктів у кадрі із використанням унікального ідентифікатора для кожного виявленого об’єкта в наступних кадрах відео.

Відстеження об'єктів у сфері відеоаналітики - найважливіше завдання, яке дозволяє не лише визначати місце розташування та клас об'єктів у кадрі, але й зберігати унікальний ідентифікатор кожного виявленого об'єкта у міру просування відео. Сфери застосування цієї технології безмежні – від відеоспостереження та безпеки до спортивної аналітики у реальному часі.

Спостереження за кількома об'єктами має безліч застосувань.

1) Транспорт: Відстеження транспортних засобів для керування рухом та автономного водіння.

2) Роздрібна торгівля: Відстеження людей для аналітики та безпеки у магазині.

3) Аквакультура: Спостереження за рибами для моніторингу водного середовища.

4) Спортивна аналітика: Відстеження гравців та обладнання для аналізу продуктивності.

5) Системи безпеки: Моніторинг підозрілих дій та створення охоронної сигналізації.

Конкретна мета розроблюваної програми – виявляти об’єкти на відео, відстежувати їх переміщення із кадру до кадру та вести підрахунок об’єктів у виділенній зоні кадру.

Метою дослідження є підвищення точності відстеження та підрахунку об’єктів на відео за рахунок використання згорткової нейронної мережі.

Об’єктом дослідження є процес комп’ютеризованого відстеження та підрахунку об’єктів на відео на основі нейромережі.

Предметом дослідження є алгоритми та програмні засоби відстеження та підрахунку об’єктів на відео на основі нейронної мережі та точність їх роботи.

Задачі дослідження:

1. Проаналізувати відомі методи відстеження та підрахунку об’єктів на відео та обрати напрямок досліджень;
2. Обгрунтувати вибір архітектури нейромережі;
3. Розробити алгоритм роботи програмного модуля відстеження та підрахунку об’єктів на відео;
4. Спроектувати модуль відстеження та підрахунку об’єктів на відео на основі нейронної мережі;
5. Здійснити програмну реалізацію програмного модуля відстеження та підрахунку об’єктів на відео;
6. Провести тестування програмного модуля відстеження та підрахунку об’єктів на відео.

Під час підготовки бакалаврської кваліфікаційної роботи досліджувались згорткові нейронні мережі і по цій тематиці були опубліковані тези доповіді [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ВІДСТЕЖЕННЯ ТА ПІДРАХУНКУ ОБ’ЄКТІВ НА ВІДЕО

1.1 Постановка задачі

Відстеження об’єктів (Object Tracking) - дуже цікавий напрямок, який досліджується та еволюціонує не перший десяток років. Зараз багато розробок у цій царині побудовано на глибокому навчанні [2], яке має перевагу над стандартними алгоритмами, оскільки нейронні мережі можуть апроксимувати функції часто краще.

Відразу потрібно зробити дуже важливе пояснення - виявлення об’єктів (Object Detection) це не те ж саме, що відстеження об’єктів (Object Tracking). Часто ці поняття плутають. Тому розглянемо це детальніше.

Виявлення об’єктів (Object Detection) - це просто визначення об'єктів на зображенні/кадрі. Тобто алгоритм або нейронна мережа визначають об'єкт і записують його позицію і обмежувальні рамки (параметри прямокутників навколо об'єктів). Поки що мови про інші кадри не йде, і алгоритм працює тільки з одним кадром. Приклад виявлення об’єктів (Object Detection) показано на рис. 1.1.

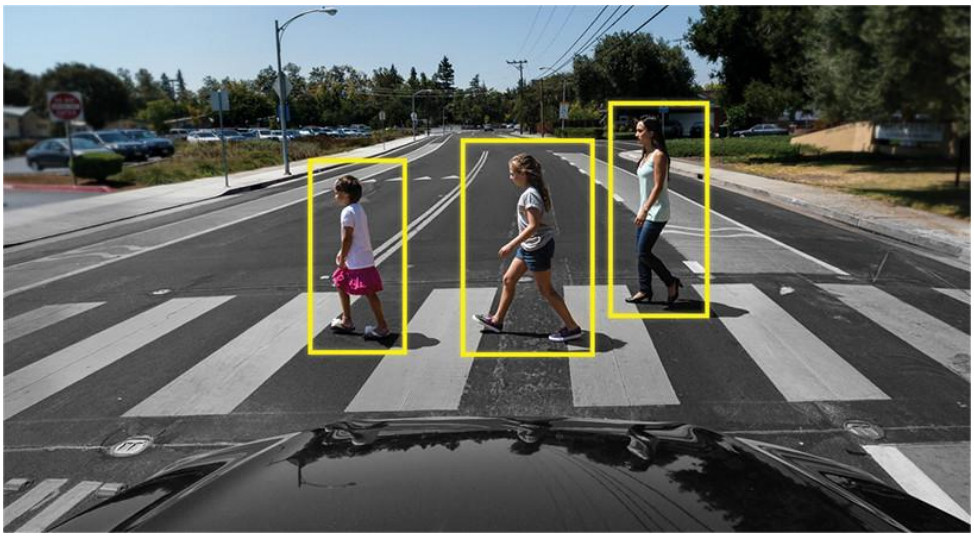


Рисунок 1.1 – Приклад виявлення об’єктів (Object Detection)

Відстеження об’єктів (Object Tracking) - зовсім інша справа. Тут завдання не просто визначити об'єкти на кадрі, а ще й пов'язати інформацію з попередніх кадрів таким чином, щоб не втрачати об'єкт, або зробити його унікальним. Приклад відстеження об’єктів (Object Tracking) показано на рис. 1.2.

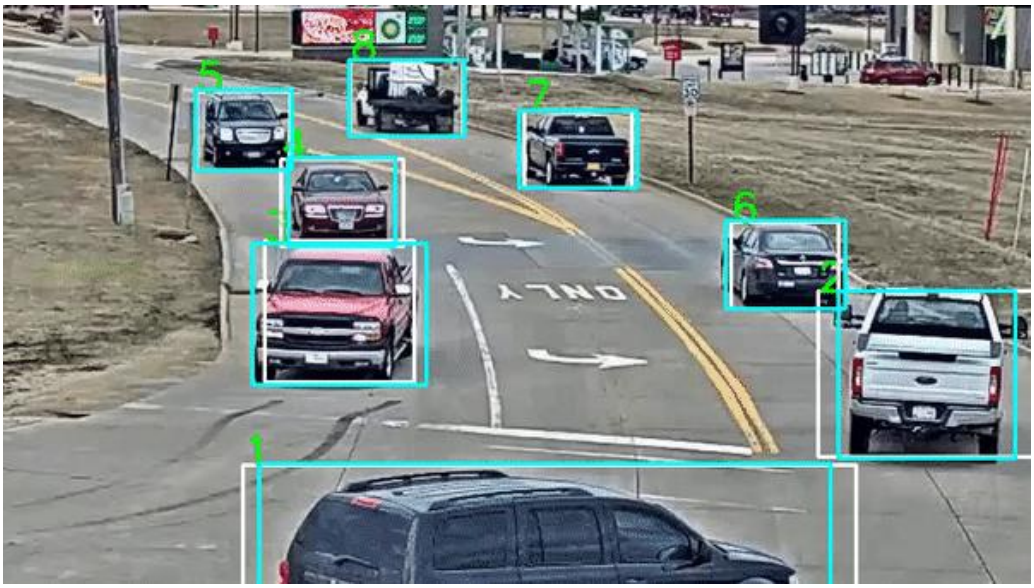


Рисунок 1.2 – Приклад відстеження об’єктів (Object Tracking)

Тобто відстежувач об’єктів (Object Tracker) включає в себе Object Detection для визначення об'єктів, та інші алгоритми для розуміння, який об'єкт на новому кадрі належить якому з попереднього кадру. Тому відстеження об’єктів (Object Tracking) відіграє дуже важливу роль у завданні трекінгу.

1.2 Огляд відомих методів відстеження та підрахунку об’єктів на відео

Виявлення об’єктів – завдання, у якому потрібно локалізувати та класифікувати об’єкти на зображенні або послідовності відеокадрів.

Варто зауважити, що для ефективного відстеження об’єктів необхідні вхідні дані від системи виявлення об’єктів (у даному випадку – YOLOv8).

Розвиток галузі виявлення об’єктів [3] за останні 20 років загалом розділяють на 2 періоди:

– період традиційних методів виявлення об’єктів (до 2014 року);
– період методів на основі глибокого навчання для виявлення об’єктів (після 2014 року).

Також останнім часом для виявлення об’єктів використовують трансформер RT-DETR (Real Time Detection Transformer), що побудований на так званих відеотрансформерах (Vision Transformers) [4] для ефективної роботи в реальному часі в різних масштабах обробки ознак зображення.

Щоб повністю розуміти основи технології відстеження об’єктів, важливо спочатку вивчити різні поточні методи відстеження (рисунок 1.3).

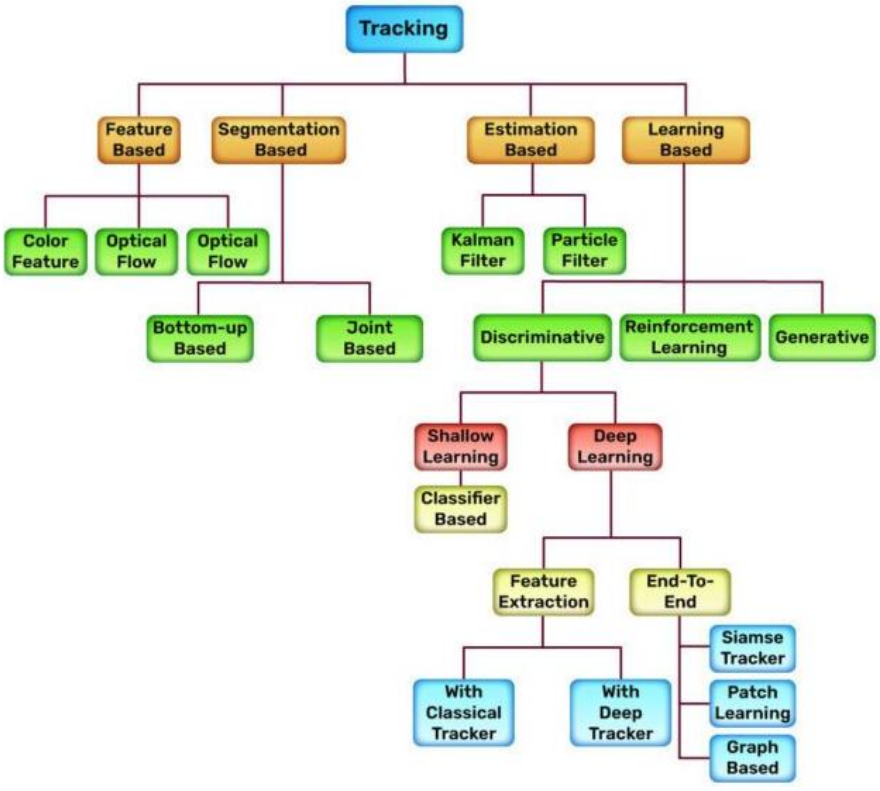


Рисунок 1.3 – Класифікаця методів відстеження

У статті [5] можна дізнатися, що методи відстеження об'єктів мають різні категорії. Наприклад, від методів, заснованих на ознаках, таких як оптичний потік, до фільтра Калмана, який використовує методи на основі оцінки для

відстеження об'єктів, до методів на основі глибокого навчання, таких як GOTURN або SORT [6].

У статті [7] описано новітній підхід до відстеження об'єктів на основі таких нейромереж як трансформери або модулі уваги. У цій статті також розглянуто застосування відстеження об'єктів у режимі реального часу, продемонстровано його практичне використання в цій передовій галузі.

Процес відстеження об'єктів можна класифікувати на дві основні категорії залежно від типу та функціональності доступних трекерів (рисунок 1.4). Це відстежувач одного об’єкта (Single Object Tracker - SOT) та відстежувач багатьох об’єктів (Multiple Object Tracker - MOT).

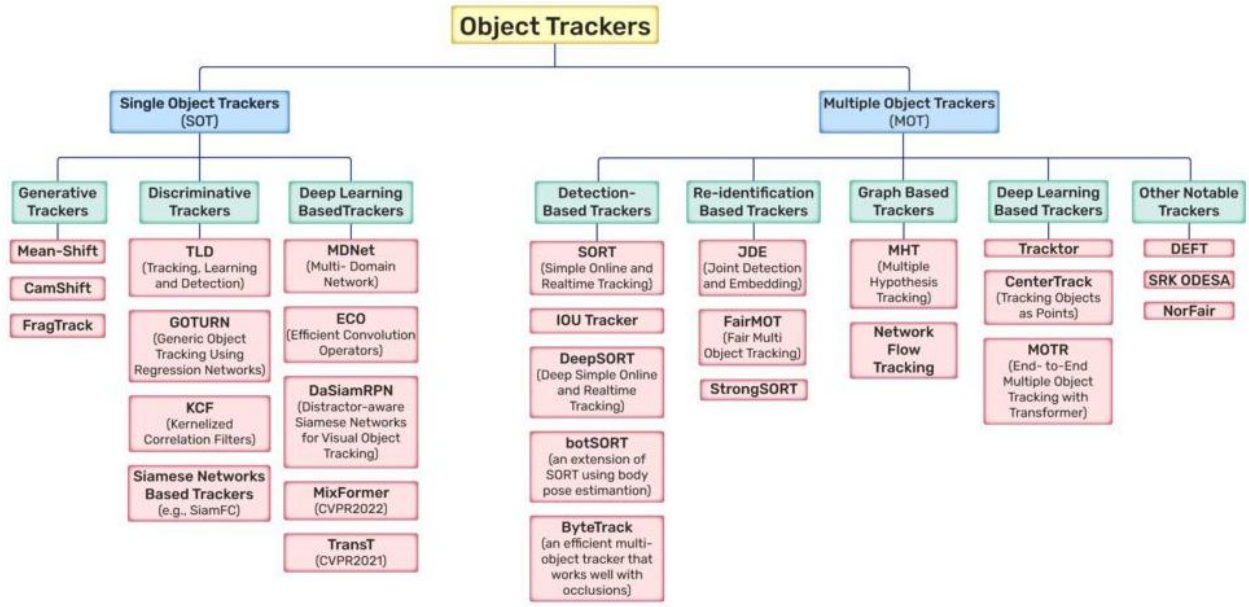


Рисунок 1.4 – Класифікація за трекерами

Відстежувач одного об’єкта (Single Object Tracker - SOT).

Якщо відтстежувати лише один об’єкт, то на першому кадрі необхідний об’єкт позначається за допомогою прямокутника для позначення місця розташування об'єкта, який ми хочемо відстежувати. Потім об'єкт відстежується в наступних кадрах за допомогою алгоритму відстеження. У більшості реальних додатків ці трекери використовуються з детектором об'єктів. OpenCV має вбудовані трекери, включаючи KCF, TLD і GOTURN.

Відстежувач багатьох об’єктів (Multiple Object Tracker - MOT).

За допомогою швидкого детектора об'єктів має сенс виявити кілька об'єктів, а потім запустити трекер для відстеження кількох об'єктів в одному кадрі або послідовності кадрів (рис. 1.5).

Існують різні вдосконалені системи відстеження кількох об'єктів (MOT), такі як DeepSORT, FaitMOT, ByteTrack, botSORT тощо. Ці системи використовують складні алгоритми для точного та ефективного відстеження кількох об'єктів у відеопотоці.

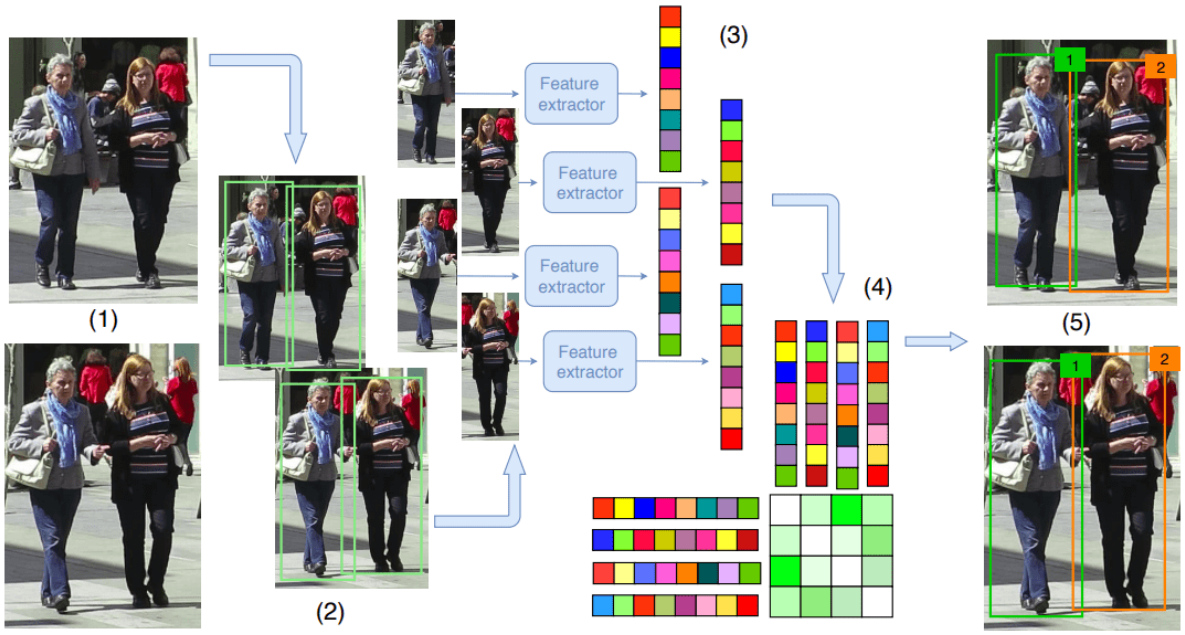


Рисунок 1.5 – Архітектура відстеження багатьох об’єктів (Multiple Object Tracker - MOT) у відеопотоці

У статті [8] стандартним підходом в алгоритмах відстеження кількох об'єктів (MOT) є «відстеження через виявлення», де детектори (обмежувальні рамки, що ідентифікують цілі у відеокадрах) керують процесом відстеження. Ці виявлені об'єкти пов'язані з підтриманням узгоджених ідентифікаторів для одних і тих самих цілей у різних кадрах. Це робить відстеження кількох об'єктів (MOT) в першу чергу проблемою призначення. Завдяки сучасним фреймворкам виявлення, що забезпечують високоякісне виявлення, більшість методів

відстеження кількох об'єктів (MOT) зосереджені на покращенні асоціації даних, а не на виявленні. Багато наборів даних для відстеження кількох об'єктів (MOT) надають заздалегідь визначені виявлення, що дозволяє алгоритмам обійти виявлення та зосередитися виключно на порівнянні якості асоціацій. Цей підхід ізолює вплив роботи детектора на результати відстеження.

1.3 Обгрунтування вибору аналогу до програмного модуля відстеження та підрахунку об’єктів на відео

На теперішній час розроблено багато відомих програмних засобів, що придатні здійснювати відстеження та підрахунок об’єктів на відео. Для цієї задачі використовують різні штучні нейронні мережі для виявлення об’єктів на відео та різні алгоритми їх відстеження та підрахунку.

У даній роботі розробляється програмний модуль на основі нейромережі YOLOv8x (Huge) і трекера DeepSORT. Для доведення переваг розробленого модуля треба порівняти його параметри точності відстеження об’єктів з параметрами точності аналога , тобто програми аналогічного призначення.

У знайдених програмах-аналогах точність оцінюється з використанням різних метрик та різних навчальних та тестових наборів даних. Вибір навчальних та тестових датасетів сильно впливає на кількісні показники метрик оцінки точності розроблених програмних засобів. Тому було прийнято рішення розробити другий варіант модуля відстеження та підрахунку об’єктів на відео на іншій модифікації нейронної мережі та з використанням іншого трекера. Взяти його як аналог і протестувати його характеристики точності на тому самому датасеті, що і розроблений модуль і з використанням такої ж метрики.

Поскільки у даній роботі розробляється програмний модуль на основі нейромережі YOLOv8x (Huge) і трекера DeepSORT, то як аналог було обрано програмний модуль на основі нейромережі YOLOv8s (Small) і трекера ByteTrack.

1.4 Висновок до розділу 1

У розділі описано детальну постановку задачі відстеження та підрахунку об’єктів на відео. Також розглядаються різні методи розв'язання задачі відстеження та підрахунку об’єктів на відео і оцінюються їх переваги і недоліки. Недоліки полягають у неточному відстеженні через наявність оклюзій та перемикання ідентифікаторів об’єктів. На основі цих недоліків сформульовано мету роботи – підвищення точності відстеження та підрахунку об’єктів на відео Для реалізації поставленої задачі для виявлення об’єктів було обрано метод на основі глибокого навчання. Для відстеження об’єктів було обрано метод множинного відстеження на основі асоціацій. Крім цього, було обгрунтовано для визначення переваг розробленого модуля порівнювати його з аналогом на основі іншої модифікації нейронної мережі та іншого трекера.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ ВІДСТЕЖЕННЯ ТА ПІДРАХУНКУ ОБ’ЄКТІВ НА ВІДЕО НА ОСНОВІ НЕЙРОННОЇ МЕРЕЖІ

2.1 Обґрунтування вибору типу нейронної мережі для модуля відстеження та підрахунку об’єктів на відео

Архітектури різних нейронних мереж для визначення об'єктів (Object Detectors).

Існує кілька архітектур нейронних мереж [9], створених для визначення об'єктів. Вони здебільшого поділяються на "дворівневі", як-от RCNN [10], Fast RCNN і Faster RCNN, та "однорівневі", як-от YOLO.

"Дворівневі" нейронні мережі, перераховані вище, використовують так звані регіони на зображенні, щоб визначити, чи знаходиться в цьому регіоні певний об'єкт. Зазвичай це виглядає так (для faster RCNN, яка є найшвидшою з перерахованих дворівневих систем – рис.2.1):

- подається картинка/кадр на вхід,
- кадр проганяється через CNN для формування карт ознак (feature maps),
- окремою нейронною мережею визначаються регіони з високою ймовірністю знаходження в них об'єктів,
- далі ці регіони за допомогою області уваги RoI pooling стискаються і подаються в нейронну мережу, що визначає клас об'єкта в регіонах.

Але в цих нейронних мережах є дві ключові проблеми: вони не дивляться на картинку «повністю», а лише на окремі регіони і вони відносно повільні.

У чому ж переваги YOLO? У тому, що ця архітектура не має двох проблем, зазначених у попередньому абзаці, і вона довела неодноразово свою ефективність.

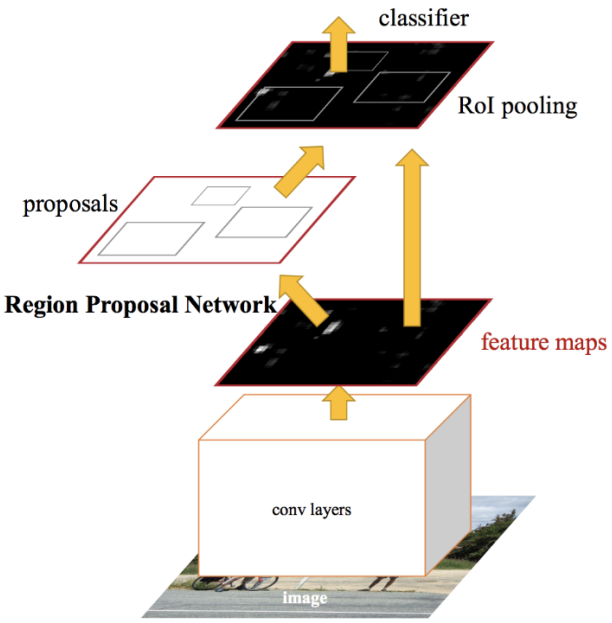


Рисунок 2.1 – Визначення об’єктів за допомогою Faster RCNN

Взагалі, архітектура YOLO в перших версіях не сильно відрізняється за «логікою блоків» від інших детекторів, тобто на вхід подається картинка, далі створюються карти ознак (feature maps) за допомогою згорткової нейронної иережі CNN (в YOLO використовується своя CNN під назвою Darknet-53), потім ці карти ознак (feature maps) певним чином аналізуються, видаючи на виході позиції та розміри обмежувальних рамок (bounding boxes) та класи, яким вони належать (рис. 2.2).

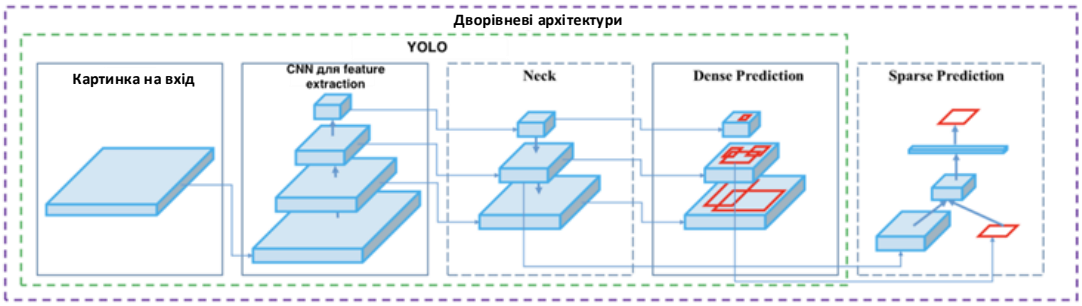


Рисунок 2.2 – Визначення об’єктів за допомогою YOLO

Але що таке шия (Neck), щільний прогноз (Dense Prediction) і розріджений прогноз (Sparse Prediction) на рис. 2.2?

Sparse Prediction - це просто повторення того, як дворівневі алгоритми працюють: визначають окремо регіони і потім класифікують ці регіони.

Neck (або "шия") - це окремий блок, який створений для того, щоб агрегувати інформацію від окремих шарів з попередніх блоків (як показано на рис. 2.2) для збільшення точності передбачення.

І, нарешті, те, що відрізняє YOLO від усіх інших архітектур - блок під назвою (на рис. 2.2) Dense Prediction. Його треба описати докладніше, тому що це дуже цікаве рішення, яке якраз дало змогу YOLO вирватися в лідери за ефективністю визначення об'єктів.

YOLO (You Only Look Once) несе в собі філософію дивитися на картинку один раз, і за цей один перегляд (тобто один прогін картинки через одну нейронну мережу) робити всі необхідні визначення об'єктів.

Отже, на виході від роботи YOLO ми зазвичай хочемо побачити те, що зображено на рис. 2.3.

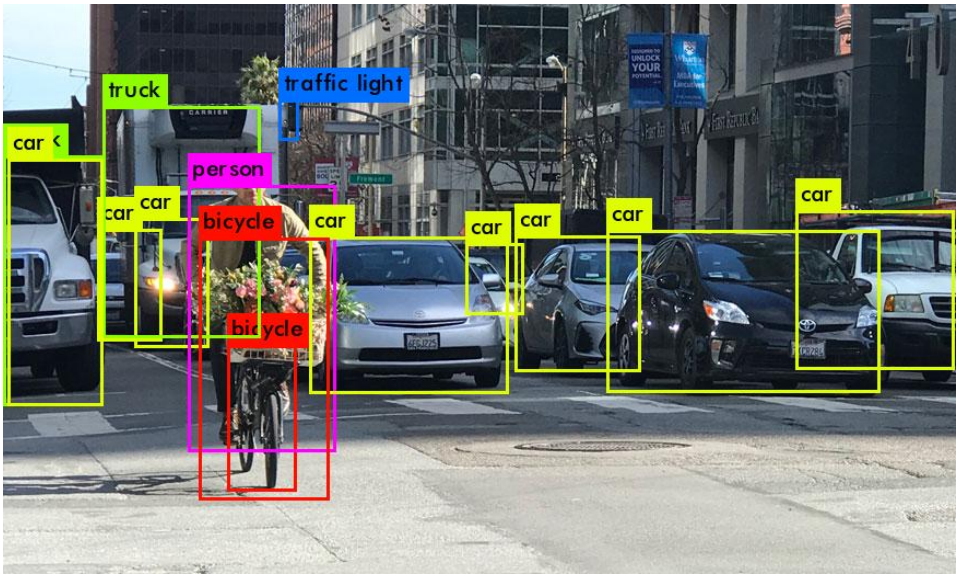


Рисунок 2.3 – Результат визначення об’єктів за допомогою YOLO

YOLOv8 — це одне з найновіших сімейств моделей виявлення об’єктів на основі YOLO від Ultralytics, що забезпечує найсучаснішу продуктивність.

Використовуючи всі переваги попередніх версій YOLO, модель YOLOv8 є швидшою та точнішою, забезпечуючи уніфіковану структуру для тренувальних моделей для виконання таких задач: виявлення об'єктів, сегментація екземплярів, класифікація зображень.

На даний момент до репозиторію Ultralytics YOLOv8 ще належить додати багато функцій. Це включає повний набір функцій експорту для навчених моделей. Крім того, Ultralytics порівнює YOLOv8 з іншими найсучаснішими моделями зору.

Компанія Ultralytics випустила абсолютно новий репозиторій для моделей YOLO. Він створений як уніфікована структура для навчання моделям виявлення об’єктів, сегментації екземплярів і класифікації зображень.

Ось деякі ключові особливості нового випуску:

- зручний API (командний рядок + Python),
- швидше та точніше,
- розширюється до всіх попередніх версій,
- нова структура хребта мережі,
- нова структура голови мережі без анкерів,
- нова функція втрат.

YOLOv8 також високоефективна і гнучка, підтримує численні формати експорту, і модель може працювати на центральних і графічних процесорах.

На рівні архітектури було внесено такі зміни:

- модулі C3 були замінені на модулі C2f.
- першу 6×6 згортку було замінено на 3×3 згортку у хребті мережі.
- використання розділеної голови мережі та видалення гілки Objectness.
- перша 1×1 зготка в хребті мережі була замінена на 3×3 згортку.

Доступні моделі YOLOv8.

У кожній категорії моделей YOLOv8 є п’ять моделей [11] для виявлення, сегментації та класифікації (див. табл. 2.1). YOLOv8 Nano (YOLOv8n) є

найшвидшою і найменшою, тоді як YOLOv8 Extra Large (YOLOv8x) є найточнішою, але найповільнішою серед них.

Таблиця 2.1 – Специфікація різновидів моделей YOLOv8

YOLOv8n	YOLOv8s	YOLOv8m	YOLOv8l	YOLOv8x
Nano (нано)	Small (мала)	Medium (середня)	Large (велика)	Huge (величезна)

YOLOv8 постачається в комплекті з такими попередньо навченими моделями:

- контрольні точки виявлення об’єктів навчені на наборі даних виявлення COCO [12] з роздільною здатністю зображення 640;
- контрольні точки сегментації екземпляра навчені на наборі даних сегментації COCO з роздільною здатністю зображення 640;
- моделі класифікації зображень, попередньо навчені на наборі даних ImageNet із роздільною здатністю зображення 224.

Таким чином, для практичної реалізації програмного модуля відстеження та підрахунку об’єктів на відео обираємо нейронну мережу YOLOv8 як найбільш перспективну.

2.2 Розробка моделі відстеження та підрахунку об’єктів на відео на основі нейронної мережі YOLOv8

YOLOv8 – це одна з найновіших архітектур згорткових нейромереж сімейства YOLO для виявлення об’єктів на зображенні від Ultralytics, яка забезпечує високу продуктивність.

Використовуючи попередні версії YOLO, модель YOLOv8 є швидшою та точнішою, водночас забезпечує уніфіковану структуру моделей для виконання:

- виявлення об’єктів;

– відстеження об’єктів;
– сегментації екземплярів;
– класифікації зображень.

Архітектура YOLOv8 наведена на рис. 2.4.

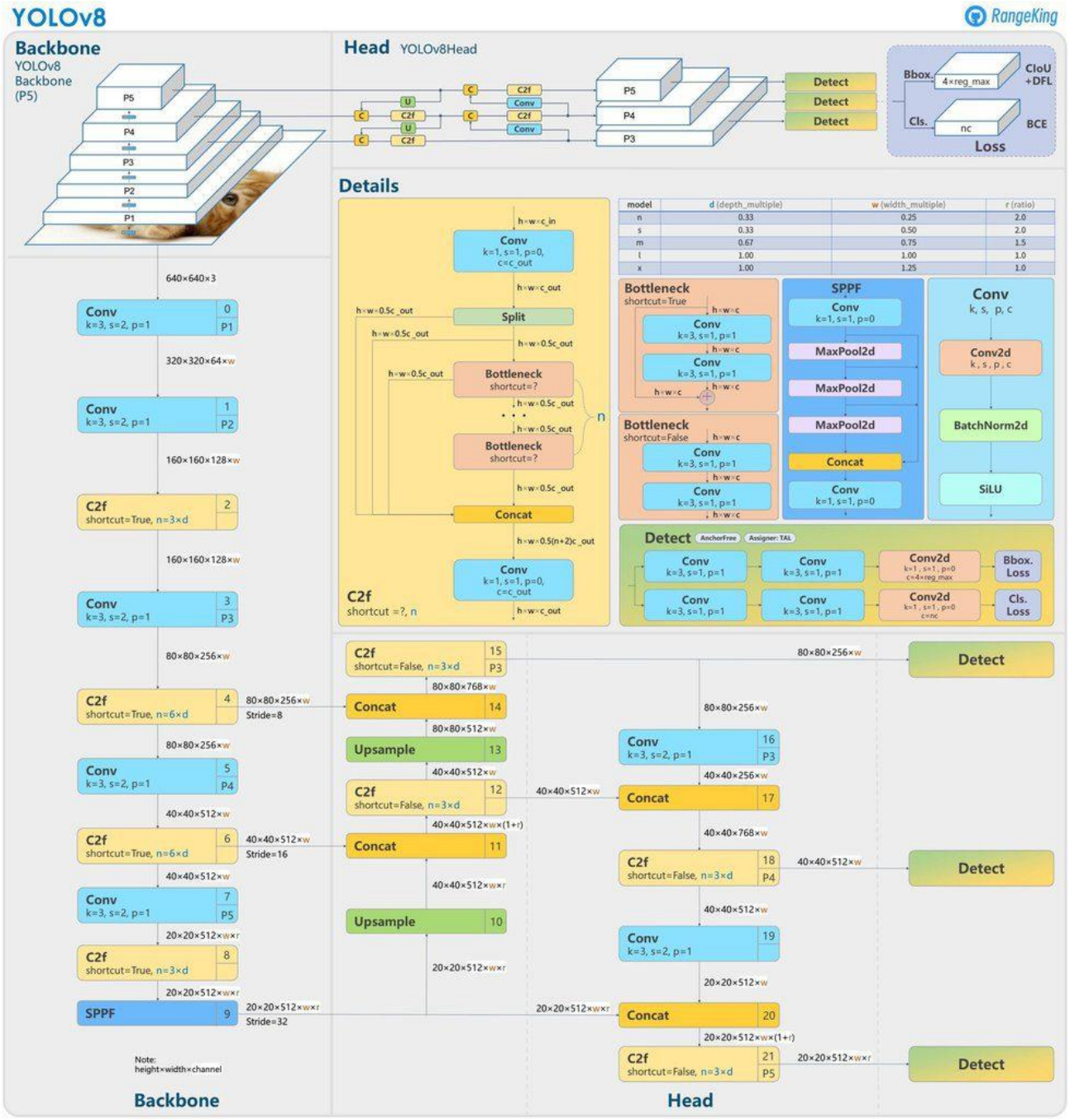


Рисунок 2.4 - Архітектура YOLOv8

Ultralytics випустила повний репозиторій для моделей YOLO. Крім того, Ultralytics у статті [13] порівнює YOLOv8 з іншими найбільш сучасними моделями комп’ютерного зору.

2.2.1 Виявлення об’єктів

Що робить YOLO коли вчиться на даних:

Крок 1: Зазвичай картинки решейплять під розмір 416x416 перед початком навчання нейронної мережі, щоб можна було їх подавати батчами (для прискорення навчання).

Крок 2: Ділимо картинку (поки що подумки) на клітини розміром аха. У YOLOv3-4 прийнято ділити на клітини розміром 13x13 (рис. 2.5).

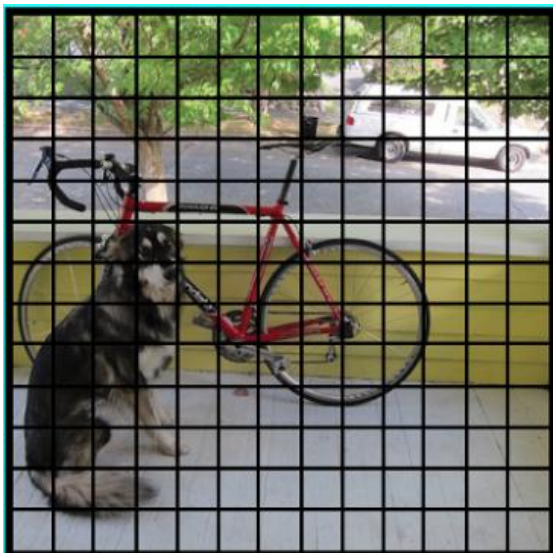


Рисунок 2.5 – Поділ картинки на клітини розміром 13x13

Тепер фокусуємося на цих клітинках, на які ми розділили картинку/кадр. Такі клітинки, які називаються grid cells, лежать в основі ідеї YOLO. Кожна клітинка є "якорем", до якого прикріплюються обмежувальні прямокутники (bounding boxes). Тобто навколо клітинки малюють кілька прямокутників для визначення об'єкта (оскільки незрозуміло, якої форми прямокутник буде найбільш підходящим, їх малюють одразу кілька та різних форм), і їхні позиції, ширину та висоту обчислюють відносно центру цієї клітинки.

Як же малюються ці прямокутники (bounding boxes) навколо клітинки? Як визначається їхній розмір і позиція? Тут у боротьбу вступає техніка anchor boxes (у перекладі - якірні коробки, або "якірні прямокутники"). Їх задає на самому початку або сам користувач, або їхні розміри визначають, виходячи з розмірів обмежувальних прямокутників (bounding boxes), які є в датасеті, на якому тренуватимуть YOLO (використовують K-means clustering і IoU для визначення найбільш підходящих розмірів). Зазвичай задають близько 3 різних якірних прямокутники (anchor boxes), які будуть намальовані навколо (або всередині) однієї клітини (рис. 2.6).

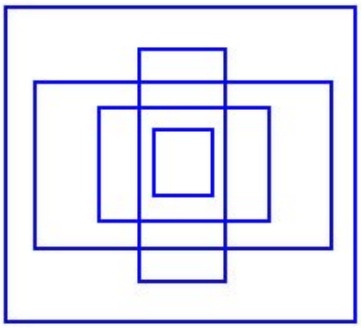


Рисунок 2.6 –Різні якірні прямокутники (anchor boxes), намальовані навколо (або всередині) однієї клітини

Навіщо це зроблено? Зараз усе буде зрозуміло, оскільки ми обговоримо те, як YOLO навчається.

Крок 3. Картинка з датасету проганяється через нейронну мережу (зауважимо, що окрім картинки в тренувальному датасеті в нас мають бути визначені позиції та розміри справжніх обмежувальних прямокутників (bounding boxes) для об'єктів, які є на ній. Це називається "анотація" і робиться це здебільшого вручну).

Що нам потрібно отримати на виході.

Для кожної клітини, нам потрібно зрозуміти дві принципові речі:

- Який з якірних прямокутників (anchor boxes), з 3 намальованих навколо клітини, нам підходить найбільше і як його можна трохи підправити для того, щоб він добре вписував у себе об'єкт,
- Який об'єкт знаходиться всередині цього якірного прямокутника (anchor box) і чи є він взагалі.

Який же повинен бути тоді вихід у YOLO?

1. На виході для кожної клітини ми хочемо отримати (рис. 2.7):

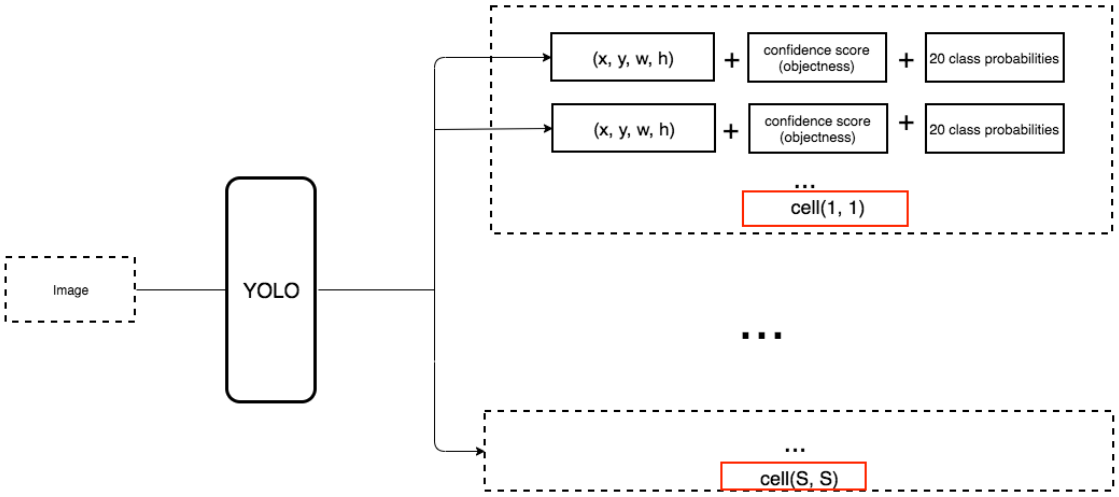


Рисунок 2.7 –Вихід YOLO для кожної клітини зображення (інформація про якірні прямокутники, намальовані навколо (або всередині) клітини)

2. Вихід YOLO має містити ось такі параметри (рис. 2.8).

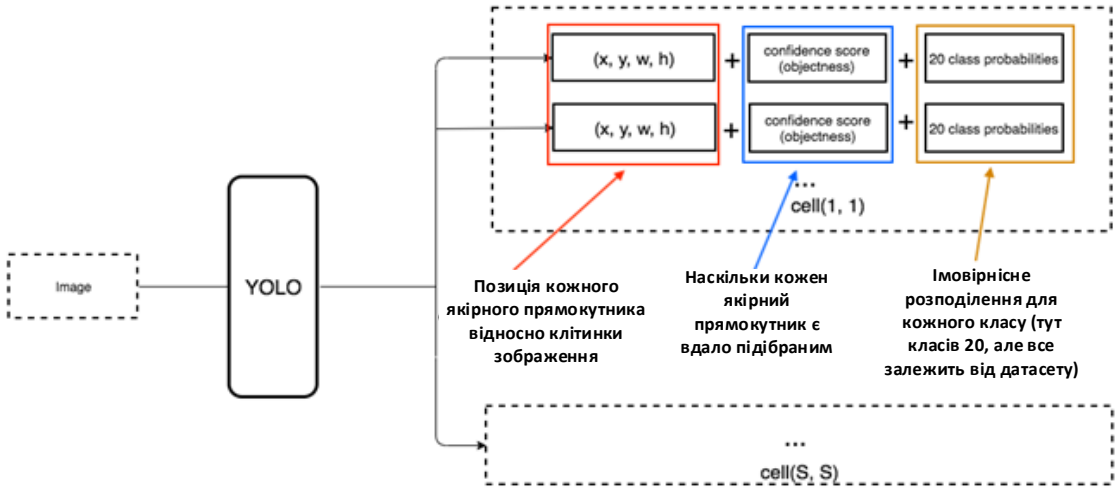


Рисунок 2.8 – Параметри, які має містити вихід YOLO

Як визначається об'єктивність (objectness)? Насправді цей параметр визначається за допомогою метрики IoU під час навчання. Метрика IoU працює як показано на рис. 2.9.

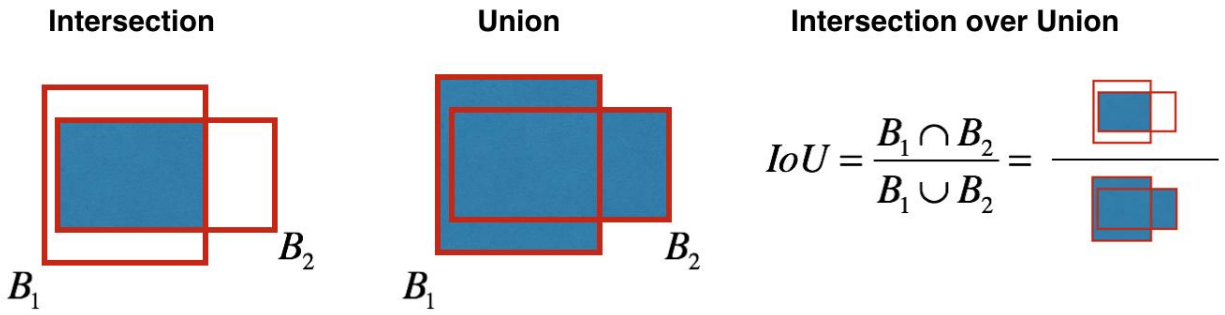


Рисунок 2.9 – Метрика IoU

На початку можна виставити поріг для цієї метрики, і якщо передбачений обмежувальний прямокутник (bounding box) буде вищим за цей поріг, то він матиме objectness, що дорівнюватиме одиниці, а всі інші обмежувальні прямокутники, у яких objectness нижча, будуть виключені. Ця величина objectness знадобиться, коли рахуватимемо загальний confidence score (показник того, на скільки ми впевнені, що це саме потрібний нам об'єкт розташований усередині передбаченого прямокутника) у кожного певного об'єкта.

Для того, щоб натренувати YOLO на те, щоб розпізнавати об’єкти на кадрі/картинці, потрібно робити наступне. Подати картинку з датасету в YOLO, там відбувається виділення ознак (feature extraction) на початку, а наприкінці в нас виходить CNN шар, який розповідає нам про всі клітинки, на які ми "поділили" нашу картинку. І якщо цей шар розповідає нам "неправду" про клітинки на картинці, то в нас мають бути великі втрати/помилки (Loss), щоб потім його зменшувати під час подачі в нейронну мережу наступних картинок.

Для повної ясності на рис. 2.10 наведена проста схема з тим, як YOLO створює цей останній шар.

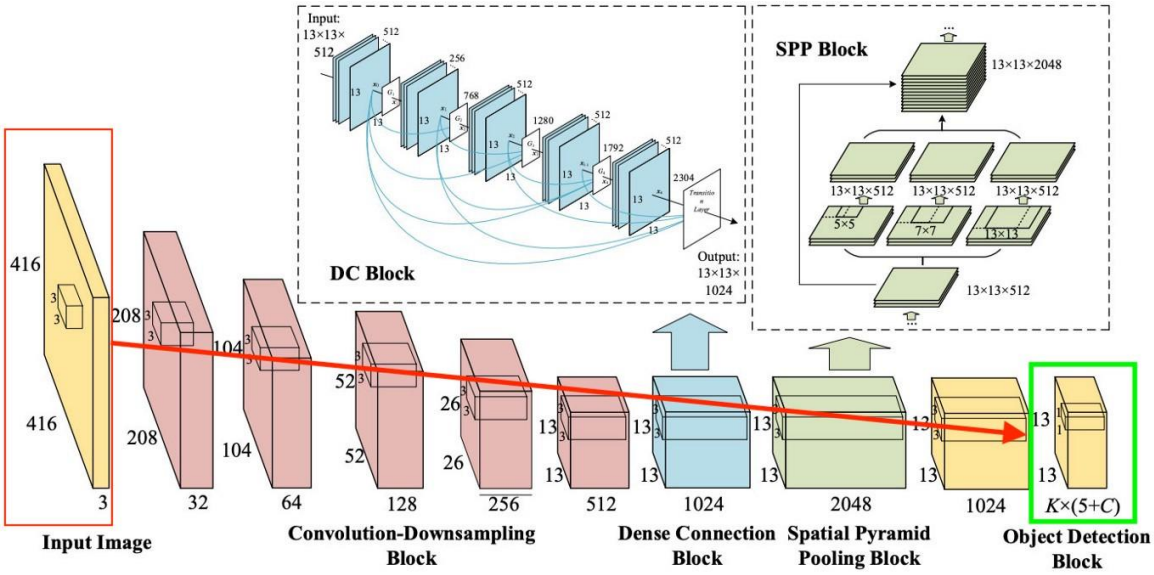


Рисунок 2.10 – Схема YOLO з деталізацією створення останнього шару

Як ми бачимо з рис. 2.10, цей шар має розмір 13x13 (для картинок початкового розміру 416x416) для того, щоб розповідати про "кожну клітину" на картинці. З цього останнього шару і дістається інформація, яку ми хочемо.

YOLO передбачає 5 параметрів (для кожного якірного прямокутника (anchor box) для певної клітинки), як показано на рис. 2.11

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$
$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o)$$

де

t_x, t_y, t_w, t_h **- це те, що передбачила YOLO**

c_x, c_y **- це координата верхньої лівої точки потрібної клітинки**

p_w, p_h **- це ширина та висота визначеного якірного прямокутника**

b_x, b_y, b_w, b_h **- це параметри для передбачення обмежувального прямокутника**

$\sigma(t_o)$ **- це наперед заданий confidence score**

Рисунок 2.11 – 5 параметрів для кожного якірного прямокутника клітинки

Детальне пояснення цього представлено на рис. 2.12.

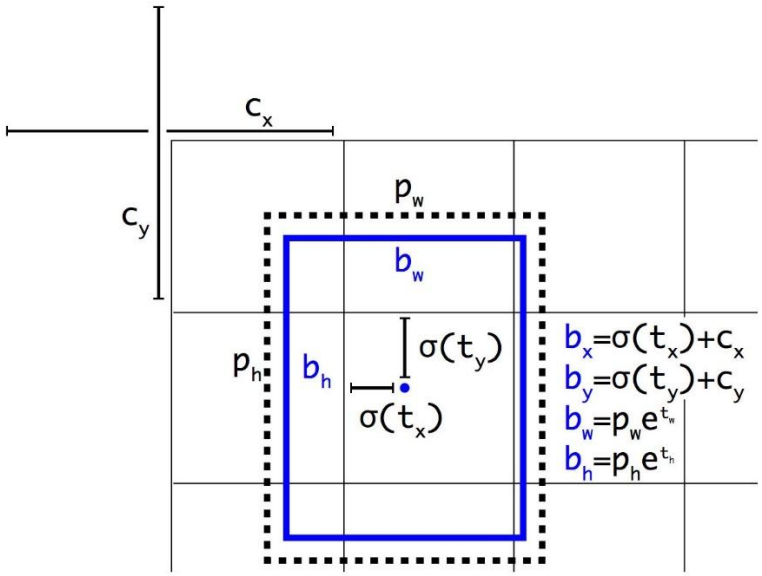


Рисунок 2.12 – Детальне пояснення визначення якірного прямокутника для певної клітинки

Як можна зрозуміти з рис. 2.12, завдання YOLO - максимально точно передбачити ці параметри, щоб максимально точно визначати об'єкт на картинці. А оцінка впевненості (confidence score), яка визначається для кожного передбаченого обмежувального прямокутника, є таким собі фільтром для того, щоб відсіяти зовсім неточні передбачення. Для кожного передбаченого обмежувального прямокутника ми множимо його IoU на ймовірність того, що це певний об'єкт (імовірнісний розподіл розраховують під час навчання нейронної мережі), беремо кращу ймовірність з усіх можливих, і якщо число після множення перевищує певний поріг, то ми можемо залишити цей передбачений обмежувальний прямокутник на картинці.

Далі, коли в нас залишилися тільки передбачені обмежувальні прямокутники із високою оцінкою впевненості (confidence score), наші передбачення (якщо їх візуалізувати) можуть виглядати приблизно так, як показано на рис. 2.13

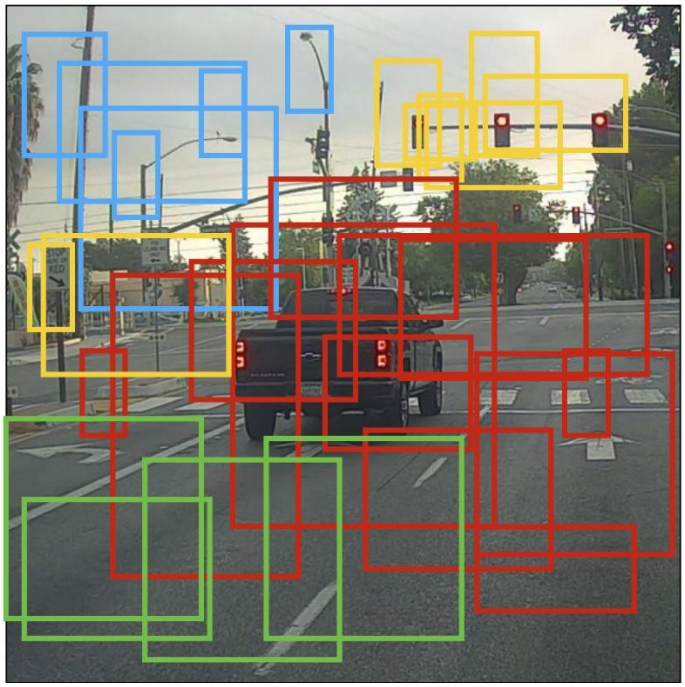


Рисунок 2.13 – Вигляд передбачених обмежувальних прямокутників із високою оцінкою впевненості

Ми можемо тепер використовувати техніку NMS немаксимального придушення (non-max suppression), щоб відфільтрувати обмежувальні прямокутники таким чином, щоб для одного об'єкта був тільки один передбачений обмежувальний прямокутник, як зображено на рис. 2.14.

Рисунок 2.14 – Застосування техніки NMS (non-max suppression), щоб відфільтрувати непотрібні обмежувальні прямокутники і залишити для одного об'єкта тільки один передбачений обмежувальний прямокутник

Потрібно також знати, що YOLO передбачають на 3-х різних шкалах. Тобто картинка ділиться на 64 клітин (grid cells), на 256 клітин і на 1024 клітини, щоб також можна було побачити маленькі об'єкти. Для кожної групи клітин алгоритм повторює необхідні дії під час передбачення/навчання, які були описані вище.

2.2.2 Відстеження об’єктів.

У попередньому пункті описано використання нейронної мережі YOLO для виявлення об’єктів (Object Detection) з найкращим поєднанням швидкість/якість. А тепер перейдемо до того, як пов'язувати інформацію про наші певні YOLO об'єкти між кадрами відео. Як програма може розуміти, що людина на попередньому кадрі - це та сама людина, що й на новому?

Для цього використовується Deep SORT, яке засновано на використанні відстані Махалонобіса [14] та фільтра Калмана [15].

Відстань Махалонобіса.

Розглянемо дуже простий приклад, щоб інтуїтивно зрозуміти, що таке відстань Махолонобіса і навіщо вона потрібна. Широко відомою є евклідова відстань. Зазвичай, це відстань від однієї точки до іншої в евклідовому просторі (рис. 2.15):

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{k=1}^{n} (p_k - q_k)^2} \quad (2.1)$$

де p_i - елементи вектора P, а q_i - елементи вектора Q, які порівнюються.

Припустимо, у нас є дві змінні - X1 і X2. Для кожної з них у нас є багато вимірювань - точок (рис. 2.16а).

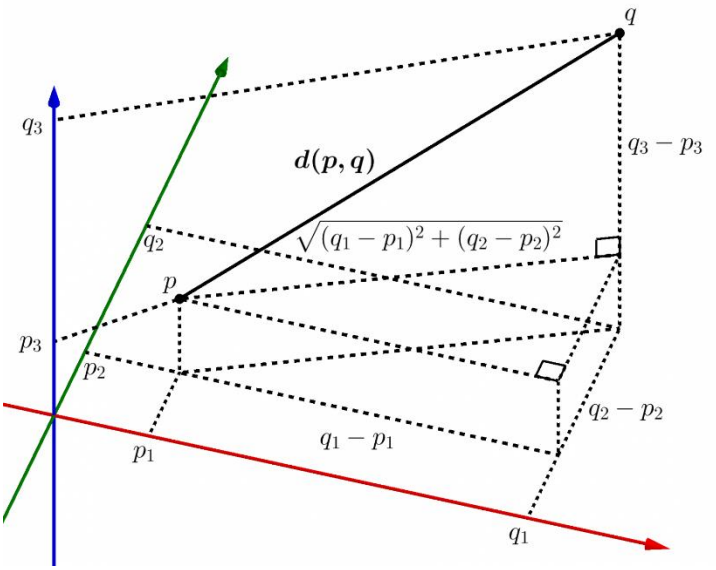
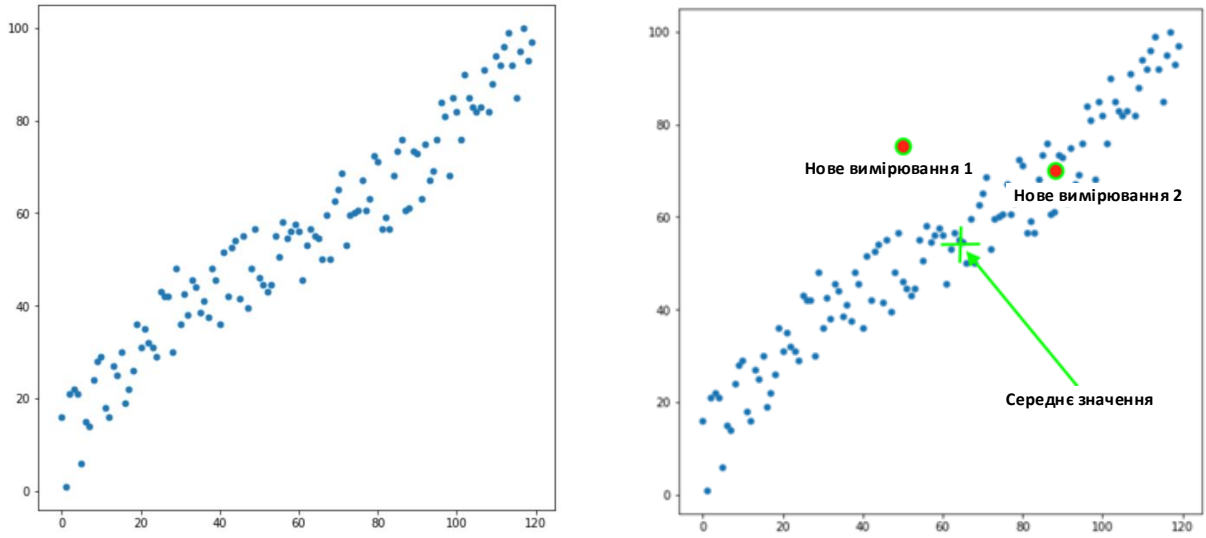


Рисунок 2.15 – Визначення евклідової відстані - це відстань від однієї точки до іншої в евклідовому просторі

Тепер, припустімо, у нас з'явилося 2 нових вимірювання (рис. 2.16б).



а) б)

Рисунок 2.16 – Пояснення застосування евклідової відстані

Як зрозуміти, яке з цих двох значень найбільше підходить для нашого розподілу? На око все очевидно - точка 2 нам підходить. Але ось евклідова відстань до середнього значення з обох точок однакова. Відповідно, проста евклідова відстань до середнього значення нам не підійде.

Як ми бачимо з рис. 2.16, змінні між собою корелюють, і досить сильно. Якби вони не корелювали між собою, або корелювали набагато менше, ми могли б заплющити очі та застосувати евклідову відстань для певних завдань, але тут нам потрібно зробити поправку на кореляцію та взяти її до уваги.

Із цим якраз справляється відстань Махалонобіса. Оскільки зазвичай в датасетах змінних більше ніж дві, замість кореляції ми будемо використовувати коваріаційну матрицю:

$$\begin{array}{c} \\ \boldsymbol{x1} \\ \boldsymbol{x2} \\ \boldsymbol{x3} \end{array} \begin{array}{c} \begin{array}{ccc} \boldsymbol{x1} & \boldsymbol{x2} & \boldsymbol{x3} \end{array} \\ \begin{bmatrix} cov(x1,x1) & cov(x1,x2) & cov(x1,x3) \\ cov(x2,x1) & cov(x2,x2) & cov(x2,x3) \\ cov(x3,x1) & cov(x3,x2) & cov(x3,x3) \end{bmatrix} \end{array} \tag{2.2}$$

Що насправді робить відстань Махалонобіса:

– позбавляється коваріації змінних,
– робить дисперсію (variance) змінних рівною 1,
– після цього використовує звичайну евклідову відстань для трансформованих даних

Подивимося на формулу, як обчислюється відстань Махалонобіса:

$$D_M(\vec{x}) = \sqrt{\underbrace{(\vec{x}-\vec{\mu})^T}_{1}\underbrace{S^{-1}}_{2}(\vec{x}-\vec{\mu})}. \tag{2.3}$$

Давайте розберемося, що означають складові нашої формули:

1) ця різниця - різниця між нашою новою точкою і середніми значеннями для кожної змінної,
2) S - це коваріаційна матриця, формула якої наведена трохи вище.

З формули можна зрозуміти дуже важливу річ. Ми за фактом множимо на перевернуту коваріаційну матрицю. У цьому разі, що вища кореляція між змінними, то найімовірніше ми скоротимо дистанцію, адже домножуватимемо

на обернене більшому - тобто менше число. Тобто ми вимірюємо відстань між точками таким чином, щоб узяти до уваги дисперсію наших змінних і коваріацію між ним.

Фільтр Калмана.

Щоб зрозуміти, що це корисна річ, яку можна застосовувати в дуже багатьох галузях, достатньо знати, що фільтр Калмана винайшли в 1960-х. Фільтр Калмана часто застосовується в аналізі часових рядів на фінансових ринках, в аналізі показників різних датчиків на заводах, підприємствах і багато де ще.

DeepSORT.

Отже, ми тепер знаємо, що таке фільтр Калмана і відстань Махалонобіса. Технологія DeepSORT просто пов'язує ці два поняття між собою для того, щоб переносити інформацію від одного кадру до іншого, і додає нову метрику, під назвою appearance. Спочатку за допомогою «object detection» визначаються позиція, розмір і клас одного обмежувального прямокутника (bounding box). Потім можна в принципі застосувати Угорський алгоритм, щоб пов'язати певні об'єкти з ID об'єктів, що раніше були на кадрі та відстежуються за допомогою фільтрів Калмана, як в оригінальному SORT. Але технологія DeepSORT дає змогу поліпшити точність визначення і зменшити кількість перемикань між об'єктами, коли, припустімо, одна людина на кадрі загороджує ненадовго іншу, і тепер людину, яку загороджували, вважають новим об'єктом.

Технологія DeepSORT додає виграшний елемент у свою роботу - так званий "зовнішній вигляд" людей, які з'являються на кадрі (appearance). Цей зовнішній вигляд був натренований окремою нейронною мережею, яка була створена авторами DeepSORT. Вони використовували близько 1,100,000 картинок із понад 1000 різних людей, щоб нейронна мережа правильно передбачала. В оригінальному SORT є проблема - оскільки там не використовується зовнішній вигляд об'єкта, то за фактом, коли об'єкт чимось закривається на кілька кадрів (наприклад, іншою людиною або колоною всередині будівлі), то алгоритм потім привласнює інший ID цій людині -

унаслідок чого так звана "пам'ять" про об'єкти в оригінального SORT доволі короткострокова.

Отже, тепер об'єкти мають дві властивості - їхня динаміка руху та їхній зовнішній вигляд. Для динаміки в нас є показники, які фільтруються і передбачаються за допомогою фільтра Калмана - (u,v,a,h,u',v',a',h'), де u,v - це позиція передбачуваного прямокутника за X і Y, a - це співвідношення сторін передбачуваного прямокутника (aspect ratio), h - висота прямокутника, ну й похідні за кожною величиною. Для зовнішнього вигляду тренували нейронну мережу, яка мала структуру, наведену в табл. 2.2.

Таблиця 2.2 – Структура нейронної мережі, яку тренували для оцінювання зовнішнього вигляду

Name	**Patch Size/Stride**	**Output Size**
Conv 1	$3 \times 3/1$	$32 \times 128 \times 64$
Conv 2	$3 \times 3/1$	$32 \times 128 \times 64$
Max Pool 3	$3 \times 3/2$	$32 \times 64 \times 32$
Residual 4	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 5	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 6	$3 \times 3/2$	$64 \times 32 \times 16$
Residual 7	$3 \times 3/1$	$64 \times 32 \times 16$
Residual 8	$3 \times 3/2$	$128 \times 16 \times 8$
Residual 9	$3 \times 3/1$	$128 \times 16 \times 8$
Dense 10		128
Batch and ℓ_2 normalization		128

Ця нейронна мережа в кінці видавала вектор ознак (feature vector) розміром 128x1. А далі, замість того, щоб рахувати дистанцію між визначеними об'єктами за допомогою YOLO, та об'єктами, за якими ми вже слідкували на кадрі, а потім приписувати певний ID просто за допомогою відстані Махалонобіса, було створено нову метрику для підрахунку відстані, що містить у собі як передбачення за допомогою фільтрів Калмана, так і "косинусну відстань" (cosine distance), як називають її інакше, коефіцієнт Отіаї.

У підсумку відстань від певного YOLO об'єкта до передбаченого фільтром Калмана об'єкта (або об'єкта, що вже є серед тих, який спостерігався на попередніх кадрах) дорівнює:

$$D = Lambda * D_k + (1 - Lambda) * D_a \quad (2.4)$$

де D_a - це дистанція за зовнішньою схожістю, а D_k - відстань Махалонобіса.

Далі ця гібридна дистанція застосовується в Угорському алгоритмі, щоб якраз правильно відсортувати певні об'єкти з наявними ID.

Таким чином, проста додаткова метрика D_a допомогла створити новий алгоритм DeepSORT, який застосовується в багатьох завданнях і є доволі популярним у завданні «Object Tracking».

2.2.3 Підрахунок об’єктів.

Підрахунок об'єктів YOLOv8 – це розширена частина виявлення об'єктів та відстеження об'єктів. Він починається з відстеження об'єктів для ідентифікації об'єктів у відеокадрах. Потім ці об'єкти відстежуються по кадрах за допомогою таких алгоритмів, як DeepSORT або ByteTrack, підтримуючи узгоджену ідентифікацію. Приклад підрахунку об’єктів представлено на рис. 2.17.

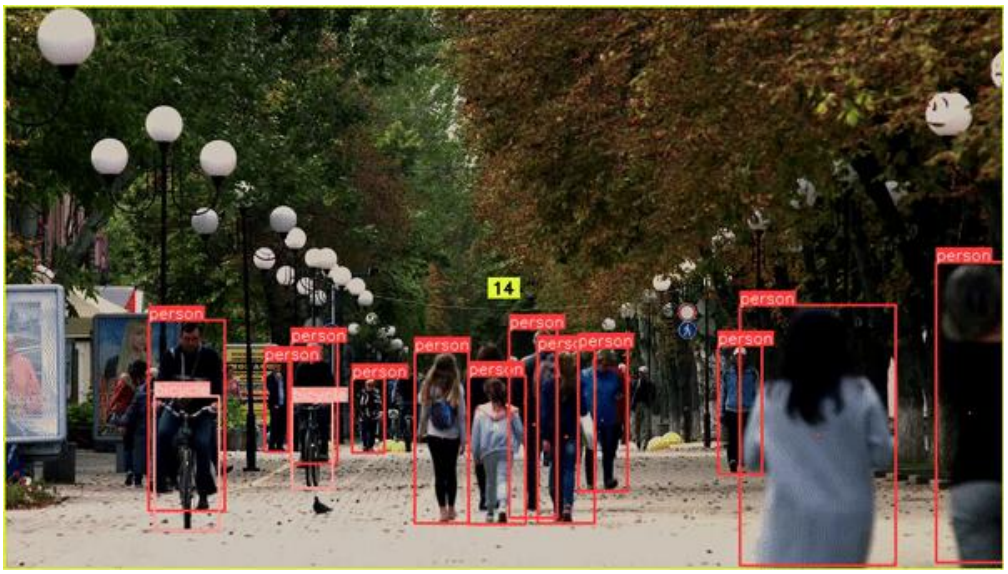


Рисунок 2.17 – Підрахунок об’єктів за допомогою YOLOv8

У підрахунку використовується простий підрахунок на весь кадр, підрахунок рядків (підрахунок об'єктів, що перетинають заздалегідь визначену лінію) і підрахунок на основі зон (підрахунок об'єктів у межах певної області). Підрахунок входів/виходів додатково розрізняє предмети, що входять у визначені зони або виходять з них.

Щоб забезпечити точність, особливо проти подвійного підрахунку, об'єкти відстежуються списками (наприклад), а кількість збільшується (наприклад) за певних умов.

Вирішуючи такі проблеми, як оклюзії та динамічні умови, цей процес дає цінні дані для розуміння в різних галузях, від моніторингу трафіку до аналітики роздрібної торгівлі.

У статті [16] представлено фреймворк глибокого навчання PseCo (рис. 2.18), який поєднує в собі чудові переваги двох базових моделей без шкоди для їх можливості нульового пострілу:

– SAM для сегментації всіх можливих об'єктів як пропозицій масок;
– CLIP для класифікації пропозицій з метою отримання точної кількості об'єктів.

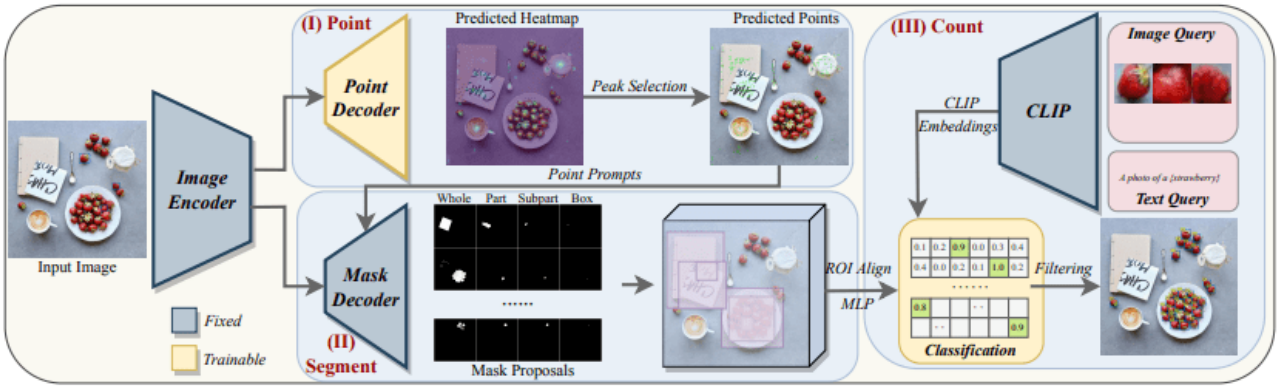


Рисунок 2.18 – Архітектура PesCo

Програмна реалізація процесу підрахунку об’єктів за допомогою відстеження об’єктів YOLOv8 детально описано в п.3.3.

Структура процесів обробки інформації програмного модуля відстеження та підрахунку об’єктів на відео складається з таких етапів:

1. Завантаження фреймворку нейронної мережі YOLOv8.
2. Завантаження набору даних COCO128
3. Навчання нейронної мережі YOLOv8.
4. Тестування нейронної мережі YOLOv8.
5. Виявлення об’єктів на відео за допомогою YOLOv8.
6. Реалізація алгоритму відстеження об’єктів на зазначеному відео.
7. Реалізація алгоритму підрахунку об’єктів на зазначеному відео.
8. Відображення результатів роботи програми.

2.3 Розробка алгоритму роботи програмного модуля відстеження та підрахунку об’єктів на відео

Згідно з метою роботи та постановкою задачі було розроблено алгоритм програмного модуля відстеження та підрахунку об’єктів на відео, представлений на рис. 2.19.

Першим кроком в програмному модулі відстеження та підрахунку об’єктів на відео буде завантаження бібліотек (блок 1). Потім йде завантаження нейронної мережі YOLOv8 (блок 2) та завантаження набору даних COCO (блок 3).

Далі переходимо до навчання нейронної мережі YOLOv8 (блок 4). Потім проводиться тестування нейронної мережі YOLOv8 (блок 5) та визначення її параметрів точності. За тим аналізується запит на обробку відео (блок 6). Якщо запита на обробку відео немає, то програма закінчує свою роботу. Якщо запит на обробку відео є, то відбувається зчитування шляху до вказаного відео (блок 7).

Далі здійснюється виявлення об’єктів на відео (блок 8) за допомогою нейромережі YOLOv8.

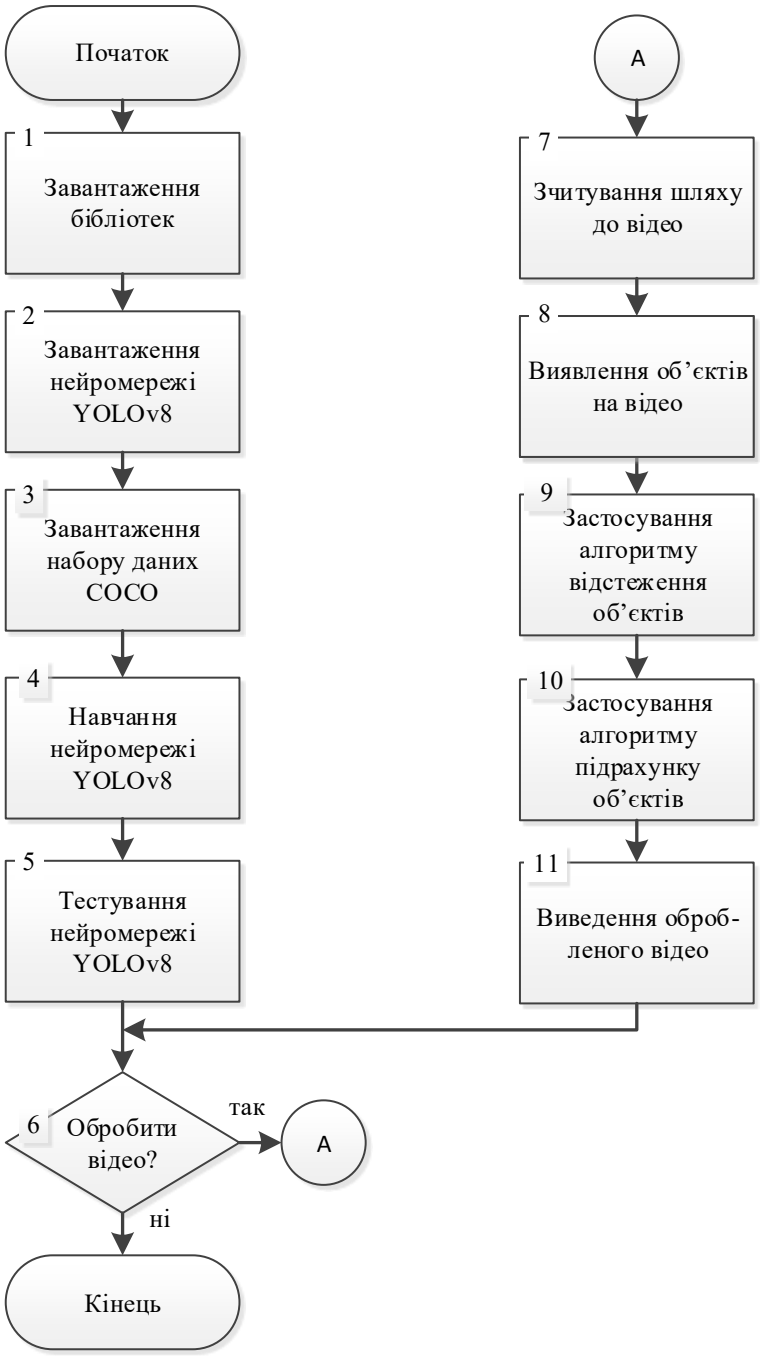


Рисунок 2.19 – Схема алгоритму роботи програмного модуля відстеження та підрахунку об’єктів на відео

Після цього застосовується алгоритм відстеження об’єктів (блок 9), а за тим застосовується алгоритм підрахунку об’єктів у кожному кадрі відео (блок 10). І нарешті, відбувається виведення обробленого відео (блок 11) та інформації про відстежені та підраховані об’єкти на кожному кадрі відео.

2.5 Висновок до розділу 2

У розділі було розглянуто різні типи згорткових нейронних мереж та для практичної реалізації програмного модуля відстеження та підрахунку об’єктів на відео обрано нейронну мережу YOLOv8 як найбільш перспективну, проаналізовано її архітектуру. Було розроблено модель відстеження та підрахунку об’єктів на відео на основі нейронної мережі YOLOv8 та трекера Deep SORT, робота якого заснована на використанні відстані Махалонобіса та фільтра Калмана. Було розроблено структуру процесів обробки інформації програмного модуля відстеження та підрахунку об’єктів на відео та алгоритм його роботи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ ВІДСТЕЖЕННЯ ТА ПІДРАХУНКУ ОБ’ЄКТІВ НА ВІДЕО НА ОСНОВІ НЕЙРОННОЇ МЕРЕЖІ

3.1 Обґрунтування вибору мови та спеціалізованих бібліотек

Для реалізації програмного модуля відстеження та підрахунку об’єктів на відео на основі нейронної мережі YOLOv8 обрано мову програмування Python та програмне середовище Anaconda.

Python [17] (читається — «Пайтон») — це об'єктно-орієнтована мова програмування високого рівня із суворою динамічною типізацією. Наявність у цій мові структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням робить її придатною для швидкої розробки програм. Python підтримує модулі та пакети модулів і сприяє повторному використанню кодів. Інтерпретатор Python має багато стандартних бібліотек, які доступні як у скомпільованій, так і у початковій формі на усіх платформах. Мова Python підтримує декілька парадигм програмування: функціональну, об'єктно-орієнтовану, аспектно-орієнтовану та процедурну.

Переваги мови програмування Python:

- зручний синтаксис (відступи застосовуються для виділення блоків);
- можливість перенесення програм ;
- стандартний дистрибутив має велику кількість корисних модулів (у тому числі - модуль розробки графічного інтерфейсу);
- властивість використовувати мови Python у діалоговому режимі (корисне для простих завдань та експериментів з програмами);
- є зручним для вирішення математичних задач (є засоби роботи з цілими числами, комплексними числами, може використовуватися як потужний калькулятор);
- відкритий код (доступність коду та можливість його редагування іншими користувачами).

Недоліки мови програмування Python.

Невисока швидкодія. Python, як і більшість інших інтерпретованих мов, які не використовують JIT-компілятори, має цей недолік — відносно невисоку швидкість виконання програм. Однак, у Python цей недолік компенсується зменшенням часу розробки програм. У середньому, програма на Python, займає в 2-4 рази менше місця, ніж її аналог на C++ або Java. Збереження байт-коду (файли типу .pyc і .pyo) дозволяє інтерпретатору, на відміну, від мови Perl, не витрачати час на перекомпіляцію коду модулів при кожному запуску. Крім того, є спеціальна JIT-бібліотека psyco, яка на жаль призводить до збільшення потреби в оперативній пам'яті. Ефективність psyco суттєво залежить від архітектури програми.

Оскільки ми використовуємо нейромережі, то для цієї роботи знадобляться бібліотеки PyTorch [18] та Ultralytics [19].

PyTorch — фреймворк машинного навчання мови Python з відкритим вихідним кодом, створений з урахуванням Torch. Використовується на вирішення різних завдань: комп'ютерний зір, обробка природної мови. Розробляється переважно групою штучного інтелекту Facebook. Також навколо цього фреймворку вибудовано екосистему, що складається з різних бібліотек, які розробляють сторонні команди.

Ultralytics — це інтуїтивно зрозуміла платформа штучного інтелекту для створення, навчання та розгортання моделей машинного навчання з безкодовим інтерфейсом і підтримкою глибокого навчання. Вона спрощує процес і надає користувачам наскрізне рішення для розгортання в хмарі, периферії або браузері.

Ultralytics створює найсучасніші моделі (SOTA) YOLO, засновані на роках фундаментальних досліджень комп’ютерного зору та ШІ. Моделі Ultralytics, які постійно оновлюються для забезпечення продуктивності та гнучкості, швидкі, точні та прості у використанні. Вони відмінно справляються з виявленням об’єктів, відстеженням, сегментацією екземплярів, класифікацією зображень і завданнями оцінки поз.

OpenCV [20] — це найбільша бібліотека з відкритим вихідним кодом для комп'ютерного зору, яка містить майже всі можливі алгоритми обробки зображень. Використання OpenCV для відстеження об'єктів YOLOv8 поєднує розширені можливості виявлення YOLOv8 із надійними функціями бібліотеки OpenCV, пропонуючи інноваційне рішення для складного відстеження об'єктів у реальному часі. OpenCV насамперед надає вісім різних трекерів, доступних у OpenCV 4.2 — BOOSTING, MIL, KCF, TLD, MEDIANFLOW, GOTURN, MOSSE та CSRT. Більшість трекерів із відкритим кодом використовують OpenCV для більшості робіт із візуалізації та обробки зображень.

3.2 Опис набору даних для навчання та тестування модуля

Моделі YOLO поставляються вже попередньо навченими на наборі даних COCO та експортовані у файли з розширенням .pt. Існує три типи моделей (класифікація, виявлення, сегментація) та по 5 моделей різного розміру для кожного типу (табл. 3.1).

Таблиця 3.1 - Типи моделей YOLO

Классификація	**Виявлення**	**Сегментація**	**Модель**
yolov8n-cls.pt	yolov8n.pt	yolov8n-seg.pt	Nano
yolov8s-cls.pt	yolov8s.pt	yolov8s-seg.pt	Small
yolov8m-cls.pt	yolov8m.pt	yolov8m-seg.pt	Medium
yolov8l-cls.pt	yolov8l.pt	yolov8l-seg.pt	Large
yolov8x-cls.pt	yolov8x.pt	yolov8x-seg.pt	Huge

Набір даних COCO є величезною колекцією зображень - 80 класів. Тому, якщо немає особливих потреб, можна просто запустити вже готову модель без попереднього навчання і оцінювати результат. Назви класів зображень COCO представлені у табл. 3.2.

Таблиця 3.2 - Назви класів зображень датасету COCO

0: 'person'	0: 'особа',	39: 'bottle'	39: 'пляшка',
1: 'bicycle'	1: "велосипед",	40:'wine glass'	40: 'фужер',
2: 'car'	2: "автомобіль",	41: 'cup'	41: 'чашка',
3: 'motorcycle	3: 'мотоцикл',	42: 'fork'	42: 'вилка',
4: 'airplane'	4: "літак",	43: 'knife'	43: "ніж",
5: 'bus'	5: 'автобус',	44: 'spoon'	44: 'ложка',
6: 'train'	6: "потяг",	45: 'bowl'	45: 'чаша',
7: 'truck'	7: 'вантажівка',	46: 'banana'	46: 'банан',
8: 'boat'	8: "човен",	47: 'apple'	47: 'яблуко',
9: traffic light'	9: "світлофор",	48: 'sandwich'	48: "сендвіч",
10: 'fire hydrant'	10: «пожежний гідрант»,	49: 'orange'	49: 'апельсин',
		50: 'broccoli'	50: "брокколі",
11: 'stop sign'	11: «знак зупинки»,	51: 'carrot'	51: 'морква',
12: 'parking meter'	12: «паркувальний автомат»,	52: 'hot dog'	52: "хот-дог",
		53: 'pizza'	53: "піца",
13: 'bench'	13: «лава»,	54: 'donut'	54: 'пончик',
14: 'bird'	14: 'птах',	55: 'cake'	55: "торт",
15: 'cat'	15: 'кіт',	56: 'chair'	56: 'крісло',
16: 'dog'	16: "собака",	57: 'couch'	57: 'диван',
17: 'horse'	17: 'кінь',	58: 'potted plant'	58: «рослина в горщику»,
18: 'sheep'	18: 'вівця',		
19: 'cow'	19: 'корова',	59: 'bed'	59: 'ліжко',
20: 'elephant'	20: "слон",	60: 'dining table'	60: 'обідній стіл',
21: 'bear'	21: 'ведмідь',		
22: 'zebra'	22: "зебра",	61: 'toilet'	61: 'туалет',
23: 'giraff'	23: "жирафа",	62: 'tv'	62: телевізор,
24: 'backpack'	24: 'рюкзак',	63: 'laptop'	63: 'ноутбук',
25: 'umbrella'	25: "парасолька",	64: 'mouse'	64: 'миша',
26: 'handbag'	26: 'сумочка',	65: 'remote'	65: 'пульт',
27: 'tie'	27: 'краватка',	66: 'keyboard'	66: 'клавіатура',
28: 'suitcase'	28: 'валіза',	67: 'cell phone'	67: 'стільниковий телефон',
29: 'frisbee'	29: "фрісбі",		
30: 'skis'	30: 'лижі',	68: 'microwave'	68: 'мікрохвильова піч',
31:'snowboard'	31: 'сноуборд',		
32: 'sports ball',	32: "спортивний м'яч",	69: 'oven'	69: 'піч',
		70: 'toaster'	70: 'тостер',
33: 'kite'	33:' повітряний змій,	71: 'sink'	71: 'раковина',
34: 'baseball bat'	34: 'бейсбольна біта',	72:'refrigerator'	72: 'холодильник',
		73: 'book'	73: 'книга',
35: 'baseball glove'	35: 'бейсбольна рукавичка',	74: 'clock'	74: 'годинник',
		75: 'vase'	75: 'ваза',
36:'skateboard'	36: 'скейтборд',	76: 'scissors'	76: "ножиці",
37: 'surfboard'	37: 'дошка для серфінгу',	77: 'teddy bear'	77: 'тедді ведмедик',
		78: 'hair drier'	78: 'фен',
38:'tennis racket'	38: 'тенісна ракетка',	79: 'toothbrush'	79: 'зубна щітка'

Будь-яке навчання нейронної мережі починається з підготовки даних для навчання. Тому нам треба дані для навчання представити у тому форматі, який необхідний для навчання моделей мережі YOLO.

Для навчання моделі нам необхідно підготувати датасет зображень та розмічених на ньому об'єктів, а потім розділити ці дані на набори для навчання та тестування.

Структура набору даних для навчання моделей YOLO повинна мати формат, представлений на рис. 3.1

Датасет повинен бути розділений на 2 папки: train (тренувальна) та val (валідаційна). Також може бути необов'язкова папка test (для тестування моделі). У кожній папці лежать ще 2 папки: images (зображення) та labels (мітки) – папка з текстовими файлами анотацій, що містять мітки об'єктів на цих картинках у форматі YOLO.

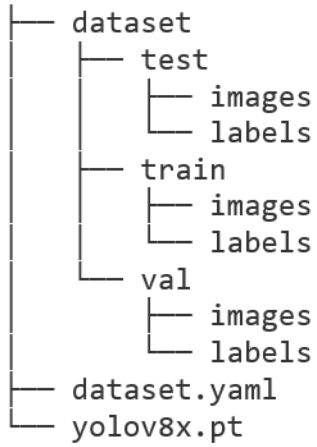


Рисунок 3.1 – Формат структури набору даних для навчання моделей YOLO

Цей формат має на увазі, що кожен рядок файлу представляється у вигляді:

```
{object_class_id} {x_center} {y_center} {width} {height}
```

Усі координати мають бути нормалізовані так, щоб вони поміщалися в діапазон від 0 до 1. Для їх розрахунку використовуються формули

```
x_center = (box_x_left+box_x_width/2)/image_width
y_center = (box_y_top+box_y_height/2)/image_height
width = box_width/image_width
height = box_height/image_height
```

де box - обмежувальна рамка об'єкта того класу, для якого ми проводимо обчислення координат.

Якщо на зображенні кілька об'єктів, текстовий файл анотацій може виглядати так:

```
1 0.589869281 0.490361446 0.326797386 0.527710843
0 0.323529412 0.585542169 0.189542484 0.351807229
```

Перший рядок містить обмежувальну рамку для одного об'єкта, другий рядок містить обмежувальну рамку для другого об'єкта. Звичайно, файл може містити і більше рядків, залежно від кількості об'єктів на зображенні.

Текстові файли анотацій повинні мати ті самі імена, що і файли зображень, і мати розширення .txt.

Далі в нашому датасеті ще обов'язково має бути файл із розширенням .yaml, який містить шляхи до навчальної та валідаційної вибірок, а також кількість класів та їх мітки.

```
train: ../train/images
val: ../val/images
test: ../test/images
nc: 80
names: ['person', … 'toothbrush']
```

У перших двох рядках вказуються шляхи до зображень та перевірочного набору даних. Потім рядок nc вказує кількість класів, які у цих наборах даних, а імена є імена класів у правильному порядку. Індекси цих елементів - це числа, які використовувалися при анотуванні зображень, і ці індекси будуть повернуті моделлю при виявленні об'єктів з використанням методу прогнозування. Файл .yaml передаватиметься під час навчання моделі.

Такий набір даних може бути отриманий під задачу користувача. По-перше, можна взяти готовий датасет у вільному доступі, наприклад, на сайті kaggle.com. Але для того, щоб знайти набір у правильному форматі, доведеться витратити час. По-друге, великий вибір безкоштовних наборів для комп'ютерного зору можна знайти на universe.roboflow.com. Цей сервіс зручний тим, що датасет можна завантажувати у різних форматах.

3.3 Програмна реалізація модуля відстеження та підрахунку об’єктів на відео

Починаємо з того, що по-перше, необхідно створити віртуальне середовище у робочому просторі. Для прикладу використовуємо Miniconda (можна використовувати будь-які інші бібліотеки відповідно до конкретних вимог):

```
!conda create -n your_env python=3.9.0
!conda activate your_env
!conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c
pytorch -c nvidia
!pip install ultralytics
!pip install opencv-python
```

Після цього потрібно імпортувати бібліотеки, з допомогою яких ми зможемо запустити приклад:

```
import cv2
from ultralytics import YOLO
```

У кожній категорії моделей YOLOv8 є 5 моделей для виявлення, сегментації та класифікації. YOLOv8 Nano є найшвидшим і найменшим, тоді як YOLOv8 Extra Large (YOLOv8x) є найточнішим, але найповільнішим серед них. Ми будемо використовувати YOLOv8x для виконання обробки відео. Також

можна точно налаштувати ці моделі відповідно до конкретних сценаріїв використання. Приклад:

```
# Load an official or custom model
model = YOLO('yolov8x.pt')  # Load an official Detect model

# Perform tracking with the model
results = model.track(source="path_to_your_video.mp4", show=False, save=True,
name='test_result', persist=True, classes=18)  # Tracking with default tracker
```

Що ми зробили, так це спочатку створили об'єкт і завантажили модель з директорії YOLO. Потім ми використовували цей метод трекінгу для виконання обробки відео. Результат наведено на рисунку 3.2.

Рисунок 3.2 – Результат відстеження об’єктів

Цей метод трекінгу внутрішньо запускає `tracker.py` скрипт, який підтримує два алгоритми відстеження. Їх можна включити, передавши відповідний файл конфігурації YAML, наприклад: `tracker=tracker_type.yaml`.

Можна використовувати один з двох наступних трекерів

– DeepSORT – новий надійний сучасний трекер, який поєднує в собі переваги інформації про рух і зовнішній вигляд, компенсацію руху камери та більш точний вектор стану фільтра Калмана.

– ByteTrack – простий, ефективний і загальний метод асоціацій, який відстежує шляхом асоціювання кожного поля виявлення, а не лише тих, що отримали високі бали. Для обмежувальних рамок виявлення з низькими оцінками ми використовуємо їх схожість з треклетами, щоб відновити дійсні об'єкти та відфільтрувати фонові виявлення.

Було запущено обробку прикладу (інференс) за допомогою трекера за замовчуванням DeepSORT.

Підрахунок об’єктів за допомогою відстеження об’єктів YOLOv8.

Використовуючи відстеження об’єктів YOLOv8, можна здійснювати облік об’єктів у режимі реального часу. Також для цього знадобиться OpenCV.

Спочатку потрібно клонувати репозиторій Ultralytics на свою локальну машину та перейти до директорії `YOLOv8-Region-Counter`:

```
# Clone Ultralytics repo
!git clone https://github.com/ultralytics/ultralytics

# Navigate to the local directory
!cd ultralytics/examples/YOLOv8-Region-Counter
```

Потім, необхідно внести кілька змін у скрипт `yolov8_region_counter.py`:

```
counting_regions = [

   {
       "name": "YOLOv8 Rectangle Region",
       "polygon": Polygon([(0, 0), (0, 1280), (720, 1280), (720, 0)]),  # Polygon points
(tl,bl,br,tr)
       "counts": 0,
       "dragging": False,
       "region_color": (37, 255, 225),  # BGR Value
       "text_color": (0, 0, 0),  # Region Text Color
   },
]
```

Таким чином ми додали індивідуальну область (прямокутник) і задали 4 кутові точки нашого вхідного відеокадру (рисунок 3.3).

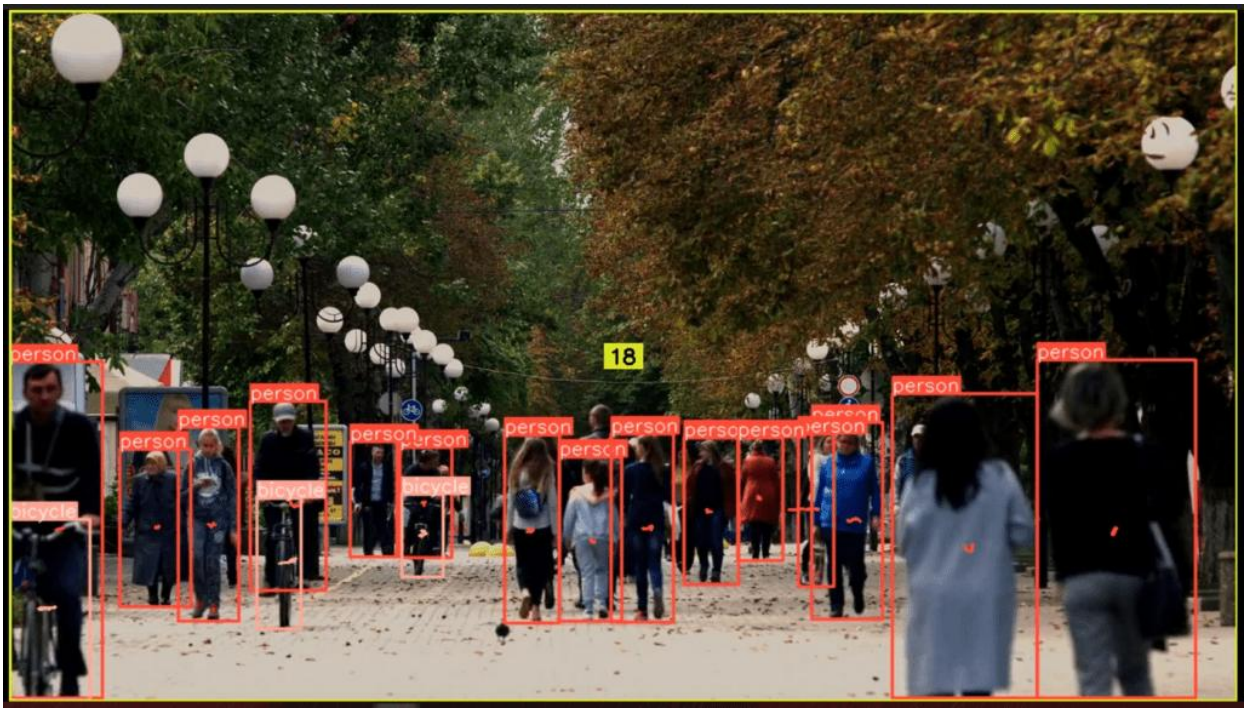


Рисунок 3.3 – Індивідуальна область для підрахунку об’єктів у цілому кадрі

Код нижче показує як вказати зображення для обробки. Тепер все готово, щоб здійснити відстеження та підрахунок об’єктів. Наведена нижче проста команда використовується для того, щоб запустити обробку відео локально:

```
!python ultralytics/examples/YOLOv8-Region-
Counter/yolov8_region_counter.py --source "/path/to/your/video.mp4" --
weights "yolov8x.pt" --view-img --save-img
```

Розглянемо детальніше як працює `yolov8_region_counter.py`.

```
results = model.track(frame, persist=True, classes=classes)
```

По-перше, скрипт використовує YOLOv8 для виявлення подій і алгоритм DeepSORT для призначення ідентифікаторів кожній виявленій людині.

Після цього скрипт проходить через кожний виявлений об’єкт. Він визначає, чи потрапляють об’єкти в попередньо визначений регіон. Якщо об’єкт

перебуває в межах регіону, сценарій збільшує відповідний лічильник щодо цієї області:

```
for region in counting_regions:
    for box in boxes:
        bbox_center = ((box[0] + box[2]) / 2, (box[1] + box[3]) / 2)  # Center of the
bounding box
        if region["polygon"].contains(Point(bbox_center)):
            region["counts"] += 1
```

Для візуалізації підрахунків і регіону використовуються cv2.rectangle, cv2.putText, cv2.polylines:

```
# Draw regions (Polygons/Rectangles)
 for region in counting_regions:
     region_label = str(region["counts"])
     region_color = region["region_color"]
     region_text_color = region["text_color"]

     polygon_coords = np.array(region["polygon"].exterior.coords, dtype=np.int32)
     centroid_x, centroid_y = int(region["polygon"].centroid.x),
int(region["polygon"].centroid.y)

     text_size, _ = cv2.getTextSize(region_label, cv2.FONT_HERSHEY_SIMPLEX,
fontScale=0.7, thickness=line_thickness)
     text_x = centroid_x - text_size[0] // 2
     text_y = centroid_y + text_size[1] // 2
     cv2.rectangle(
                frame,
                (text_x - 5, text_y - text_size[1] - 5),
                (text_x + text_size[0] + 5, text_y + 5),
                region_color,
                -1,)
     cv2.putText(frame, region_label, (text_x, text_y), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
region_text_color, line_thickness)
     cv2.polylines(frame, [polygon_coords], isClosed=True, color=region_color,
thickness=region_thickness)
```

Всі необхідні зміни вносяться у сценарій yolov8_region_counter.py. Все, що потрібно зробити, це замінити оригінальний файл yolov8_region_counter.py у клонованому каталозі новим зміненим сценарієм.

3.4 Висновок до розділу 3

У розділі було обґрунтовано вибір мови програмування Python та спеціалізованих бібліотек PyTorch, Ultralytics та OpenCV для програмної реалізації модуля відстеження та підрахунку об’єктів на відео. Проведено аналіз набору даних зрбражень для навчання нейронної мережі YOLOv8. Проаналізовано структуру набору даних, класи виявлених об’єктів та принципи формування анотацій для обмежувальних прямокутників. Описано основні етапи програмної реалізація та функціонування модуля відстеження та підрахунку об’єктів на відео, що складається із завантаження фреймворку нейронної мережі YOLOv8, завантаження набору даних COCO128, навчання нейронної мережі YOLOv8, тестування нейронної мережі YOLOv8, виявлення об’єктів на відео за допомогою YOLOv8, реалізації алгоритму відстеження об’єктів на зазначеному відео, реалізації алгоритму підрахунку об’єктів на зазначеному відео, відображення результатів роботи програми.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ ПРОГРАМНОГО МОДУЛЯ ВІДСТЕЖЕННЯ ТА ПІДРАХУНКУ ОБ'ЄКТІВ НА ВІДЕО НА ОСНОВІ НЕЙРОННОЇ МЕРЕЖІ

4.1 Тестування програмного модуля відстеження та підрахунку об'єктів на відео на основі нейронної мережі

Тестування проводилось на відео у форматі mp4 . На рис. 4.1 - 4.3 показано окремі кадри відео з результатами роботи програмного модуля.

Рисунок 4.1 – Окремий кадр відео з результатами роботи програмного модуля

На рис. 4.1-4.3 видно, що задано 2 області для підрахунку об'єктів: прямокутник жовтого кольору і п'ятикутник синього кольору. Цифри у центрі прямокутників показують кількість об'єктів, що знаходяться в його зоні. Так, на рис. 4.1 в прямокутнику знаходиться 5 об'єктів, а в п'ятикутнику – 0. На рис. 4.2 в прямокутнику знаходиться 1 об'єкт, а в п'ятикутнику – 6. На рис. 4.3 в

прямокутнику знаходиться 3 об’єкти, а в п’ятикутнику – 1. Кожен об’ється обмежувальною рамкою і пишеться його назва.

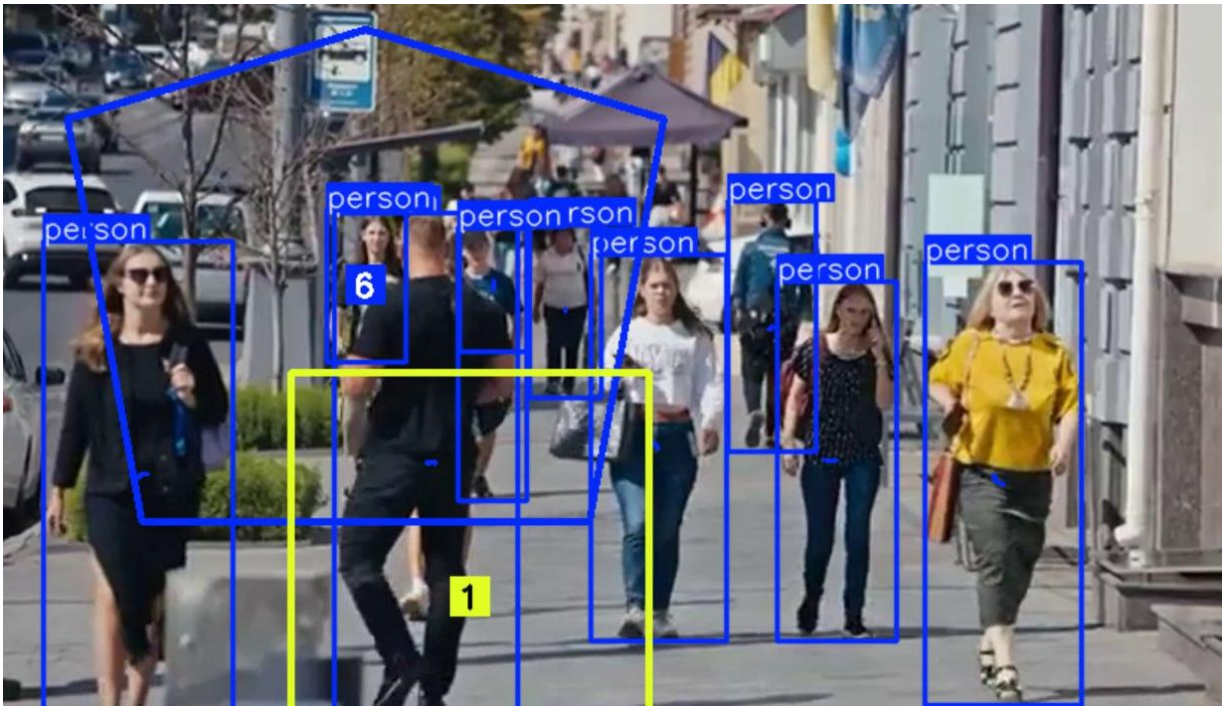


Рисунок 4.2 – Окремий кадр відео з результатами роботи програмного модуля)

Рисунок 4.3 – Окремий кадр відео з результатами роботи програмного модуля

Також програма формує звіт, де по кожному кадру виводить скільки і яких об’єктів було знайдено у визначеній зоні. Приклад звіту покадрового відстеження та підрахунку об’єктів наведено на рис. 4.4

```
video 1/1 (frame 72/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 4 cars, 4 handbags, 587.2ms
video 1/1 (frame 73/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 4 cars, 4 handbags, 584.1ms
video 1/1 (frame 74/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 5 handbags, 587.4ms
video 1/1 (frame 75/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 5 handbags, 627.0ms
video 1/1 (frame 76/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 5 handbags, 612.9ms
video 1/1 (frame 77/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 5 handbags, 600.3ms
video 1/1 (frame 78/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 5 handbags, 586.8ms
video 1/1 (frame 79/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 4 handbags, 588.5ms
video 1/1 (frame 80/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 4 handbags, 588.4ms
video 1/1 (frame 81/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 4 handbags, 590.6ms
video 1/1 (frame 82/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 4 handbags, 586.5ms
video 1/1 (frame 83/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 4 handbags, 584.8ms
video 1/1 (frame 84/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 3 handbags, 579.6ms
video 1/1 (frame 85/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 3 handbags, 589.7ms
video 1/1 (frame 86/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 3 handbags, 592.2ms
video 1/1 (frame 87/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 3 handbags, 591.7ms
video 1/1 (frame 88/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 3 handbags, 587.8ms
video 1/1 (frame 89/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 3 handbags, 607.0ms
video 1/1 (frame 90/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 3 handbags, 594.8ms
video 1/1 (frame 91/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 3 handbags, 604.1ms
video 1/1 (frame 92/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 3 handbags, 589.3ms
video 1/1 (frame 93/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 1 umbrella, 4 handbags, 598.2ms
video 1/1 (frame 94/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 1 umbrella, 4 handbags, 593.9ms
video 1/1 (frame 95/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 1 umbrella, 4 handbags, 589.8ms
video 1/1 (frame 96/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 1 umbrella, 4 handbags, 590.3ms
video 1/1 (frame 97/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 2 cars, 1 truck, 1 umbrella, 4 handbags, 590.2ms
video 1/1 (frame 98/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 2 cars, 1 truck, 1 umbrella, 3 handbags, 593.3ms
video 1/1 (frame 99/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 umbrella, 3 handbags, 591.2ms
video 1/1 (frame 100/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 umbrella, 3 handbags, 588.2ms
video 1/1 (frame 101/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 umbrella, 4 handbags, 583.6ms
video 1/1 (frame 102/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 umbrella, 4 handbags, 596.7ms
video 1/1 (frame 103/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 umbrella, 5 handbags, 644.0ms
video 1/1 (frame 104/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 umbrella, 5 handbags, 588.8ms
video 1/1 (frame 105/149) G:\BKRRRR\video.mp4: 384x640 1 bus, 581.9ms
```

Рисунок 4.4 – Приклад звіту покадрового відстеження та підрахунку об’єктів

Таким чином, в результаті тестування було доведено повну працездатність програми і відповідність поставленому завданню.

4.2 Аналіз результатів роботи програмного модуля відстеження та підрахунку об’єктів на відео на основі нейронної мережі

Оцінка якості роботи програми завжди відіграє велику роль під час експериментів і тестування нових програмних продуктів. Розроблений у даній роботі модуль спочатку виявляє об’єкти, потім їх відстежує і потім підраховує, тобто його робота містить 3 складових частини. Тому його якість можна

оцінювати окремо по кожній із складових. Якщо об’єкти вірно визначені і відстежені, то їх підрахунок є простою задачею, тобто третя складова залежить від перших двох і не представляє складності. Можна було б оцінювати якість роботи програми тільки по першій складовій, тобто по точності виявлення об’єктів. Але точність виявлення залежить від роботи самої нейромережі YOLOv8 і оцінена у багатьох дослідженнях у порівнянні з іншими архітектурами згорткових нейронних мереж. Похибки у роботі розробленої програми можуть виникнути не на етапі виявлення об’єктів, а на етапі відстеження , коли об’єкт ненадовго перекривається іншими об’єктами (так звана оклюзія).

Оскільки основним призначенням розробленої програми є відстеження об’єктів, то потрібно оцінювати якість роботи програми по здатності саме точно відстежувати об’єкти. Для цього є спеціальні метрики та спеціальні тестові датасети (набори даних для порівняння).

Ми будемо використовувати показники CLEARMOT, щоб оцінити ефективність нашого програмного модуля на наборі даних MOT17.

Тест MOT Challenge — це платформа, яка надає велику колекцію наборів даних із складними реальними послідовностями, точними анотаціями та багатьма показниками. MOT Challenge складається з різних наборів даних, таких як люди, об’єкти, 2D, 3D та багато іншого. Зокрема, щороку випускається кілька варіантів набору даних, наприклад MOT15, MOT17 і MOT20, які вводяться для вимірювання продуктивності засобів відстеження кількох об’єктів.

Для нашої оцінки ми будемо використовувати підмножину набору даних MOT17. Набір даних MOT17 має формат, який пропонує відеофайли:

- seqinfo.ini – інформація про відеофайл;
- gt.txt – реальна (правдива) анотація відстеження об’єктів;
- det.txt – реальна (правдива) анотація виявлення об’єктів;
- вихідний формат.

Щоб оцінити продуктивність, вивід трекера повинен мати певний формат, наприклад frame,id,x1,y1,x2,y2,1,-1,-1,-1, де:

- frame: номер кадру,
- id: ідентифікатор відстежуваного об'єкта,
- x1, y1: ліві верхні координати,
- x2, y2: праві нижні координати.

Формат реальної (правдивої) анотації.

Для оцінки нам знадобиться файл gt.txt із zip-файлу анотації. Його можна завантажити з https://motchallenge.net/data/MOT17Labels.zip.

Формат такий: frame, ID, bbox, whether to ignore, classes, occlusion, де

- frame: номер кадру відео,
- ID: ID відстежуваного об'єкта,
- bbox: координата обмежувальної рамки об’єкта,
- whether to ignore: чи ігнорувати об’єкт, 0 означає ігнорувати,
- classes: містить класи пішоходів, автомобілів, статичних осіб тощо,
- occlusion : оклюзія (показує, чи закритий або розділений об’єкт іншими об’єктами).

Метрики ClearMOT.

Це framework для оцінки продуктивності трекера за різними параметрами. Загалом надається 8 різних показників для оцінки виявлення об’єктів, локалізації та ефективності відстеження. Він також надає нам два нові показники:

Влучність відстеження кількох об’єктів (MOTP - Multiple Object Tracking Precision)

Достовірність відстеження кількох об’єктів (MOTA - Multiple Object Tracking Accuracy)

Ці показники допомагають оцінити загальні переваги трекера та оцінити його загальну продуктивність. Інші показникі подані на рис. 4.5.

Evaluation Measures

Lower is better. Higher is better.

Measure	Better	Perfect	Description
Avg Rank	lower	1	This is the rank of each tracker averaged over all present evaluation measures.
MOTA	higher	100 %	Multiple Object Tracking Accuracy [1]. This measure combines three error sources: false positives, missed targets and identity switches.
MOTP	higher	100 %	Multiple Object Tracking Precision [1]. The misalignment between the annotated and the predicted bounding boxes.
IDF1	higher	100 %	ID F1 Score [2]. The ratio of correctly identified detections over the average number of ground-truth and computed detections.
FAF	lower	0	The average number of false alarms per frame.
MT	higher	100 %	Mostly tracked targets. The ratio of ground-truth trajectories that are covered by a track hypothesis for at least 80% of their respective life span.
ML	lower	0 %	Mostly lost targets. The ratio of ground-truth trajectories that are covered by a track hypothesis for at most 20% of their respective life span.
FP	lower	0	The total number of false positives.
FN	lower	0	The total number of false negatives (missed targets).
ID Sw.	lower	0	The total number of identity switches. Please note that we follow the stricter definition of identity switches as described in [3].
Frag	lower	0	The total number of times a trajectory is fragmented (i.e. interrupted during tracking).
Hz	higher	Inf.	Processing speed (in frames per second excluding the detector) on the benchmark.

Рисунок 4.5 – Метрики ClearMOT для оцінки продуктивності трекера

Для відстеження об’єктів ми будемо оцінювати ефективність розробленого модуля на основі показника MOTA, який показує ефективність виявлення, промахів і перемикання ідентифікаторів. Точність трекера MOTA (Multiple Object Tracking Accuracy) обчислюється за формулою:

$$\text{MOTA} = 1 - \frac{\sum_t FN_t + FP_t + IDS_t}{\sum_t GT_t} \tag{4.1}$$

де FN — кількість хибно негативних результатів, FP — кількість хибно позитивних результатів, IDS — кількість перемикань ідентифікатору в момент часу t, а GT (ground truth) — реальний стан. MOTA також може бути негативним.

Для оцінки продуктивності з використанням набору даних MOT17 при тестуванні, потрібно завантажити відеопослідовності з цього набору, потім запустити їх одну за одною. Для кожної буде обчислено ряд параметрів, для яких знаходять середні значення. Приклад оцінки продуктивності при використання 3 відеопослідовностей показано на рис. 4.6.

```
****************************** MOT17-02-DPM.mp4 Evaluation ******************************
IDF1  IDP  IDR  | Rcll  Prcn  FAR   | GT   MT   PT   ML   | FP     FN     IDs   FM    | MOTA  MOTP
 31.5 55.8 22.0 |  30.6  77.9  2.70 |   62    7   19   36 |   1619  12887    87   187 |  21.5  77.9

****************************** MOT17-04-DPM.mp4 Evaluation ******************************
IDF1  IDP  IDR  | Rcll  Prcn  FAR   | GT   MT   PT   ML   | FP     FN     IDs   FM    | MOTA  MOTP
 47.2 73.1 34.9 |  36.5  82.4  3.54 |   83    6   38   39 |   3720  30195    45   239 |  28.6  82.2

****************************** MOT17-10-DPM.mp4 Evaluation ******************************
IDF1  IDP  IDR  | Rcll  Prcn  FAR   | GT   MT   PT   ML   | FP     FN     IDs   FM    | MOTA  MOTP
 42.9 56.9 34.5 |  50.2  82.9  2.03 |   57   11   24   22 |   1329   6392   104   276 |  39.1  74.7

****************************** Summary Evaluation ******************************
IDF1  IDP  IDR  | Rcll  Prcn  FAR   | GT   MT   PT   ML   | FP     FN     IDs   FM    | MOTA  MOTP
 43.0 66.4 31.8 |  37.4  81.6  2.89 |  202   24   81   97 |   6668  49474   236   702 |  28.6  79.7
```

Рисунок 4.6 – Приклад оцінки продуктивності при використання 3 відеопослідовностей

При використанні тих самих відеопослідовностей були розраховані відповідні метрики і для програми-аналога (на основі YOLOv8s і трекера ByteTrack). Результати занесено у табл. 4.1.

Із табл. 4.1 видно, що розроблений модуль (на основі нейромережі YOLOv8x і трекера DeepSORT) переважає аналог (на основі нейромережі YOLOv8s і трекера ByteTrack) за метрикою MOTP на 4,5% (79.7 % проти 75,2%), а за метрикою MOTA на 2,3% (28.6 % проти 26,3%).

Таблиця 4.1 – Середні значення метрик якості роботи для розробленого модуля і аналога

№ п/п	Метрика	Розроблений модуль (YOLOv8x і DeepSORT)	Аналог (YOLOv8s і ByteTrack)
1	**MOTP** Влучність відстеження кількох об’єктів (Multiple Object Tracking Precision)	79,7 %	75,2 %
2	MOTA Достовірність відстеження кількох об’єктів (Multiple Object Tracking Accuracy)	28,6 %	26,3 %

Розроблений програмний модуль показав себе досить добре візуально. Однак метрики показують не надто хороші результати (особливо MOTA далека від 100%). Причиною цього є такі недоліки, як перемикання ідентифікаторів, погана обробка оклюзії, розмиття руху та багато іншого. Середня достовірність відстеження кількох об’єктів (MOTA), яку ми отримали, становить 28,6, що є дуже низьким показником. Але позитивним моментом є швидкодія. Результати можна покращити за допомогою новітніх алгоритмів. Алгоритми, такі як FairMOT і CentreTrack, є дуже просунутими і можуть значно зменшити перемикання ідентифікаторів і дуже добре обробляти оклюзії.

Таким чином, можна зробити висновок, що розроблений програмний модуль відстеження та підрахунку об’єктів на відео на основі нейронної мережі має порівняно з аналогом збільшену на 4,5% влучність відстеження кількох об’єктів (MOTP) та збільшену на 2,3% достовірність відстеження кількох об’єктів (MOTA). Тобто мета роботи досягнута – точність відстеження та підрахунку об’єктів на відео підвищена.

4.3 Висновок до розділу 4

В результаті тестування програми було доведено її повну працездатність та відповідність поставленому завданню. Для оцінки точності роботи модуля використовувався спеціалізований набір даних MOT17. Розроблений програмний модуль відстеження та підрахунку об’єктів на відео (на основі нейромережі YOLOv8x і трекера DeepSORT) переважає аналог (на основі нейромережі YOLOv8s і трекера ByteTrack) на 4,5% по влучності відстеження кількох об’єктів (MOTP) та на 2,3% по достовірності відстеження кількох об’єктів (MOTA). Тобто мета роботи досягнута – точність відстеження та підрахунку об’єктів на відео підвищена.

ВИСНОВКИ

У роботі було розглянуто задачу відстеження та підрахунку об’єктів на відео. Були розглянуті різні методи розв'язання задачі відстеження та підрахунку об’єктів на відео і оцінено їх переваги і недоліки. Недоліки полягають у неточному відстеженні через наявність оклюзій та перемикання ідентифікаторів об’єктів. На основі цих недоліків сформульовано мету роботи – підвищення точності відстеження та підрахунку об’єктів на відео Для реалізації поставленої задачі для виявлення об’єктів було обрано метод на основі глибокого навчання. Для відстеження об’єктів було обрано метод множинного відстеження на основі асоціацій. Крім цього, було обгрунтовано для визначення переваг розробленого модуля порівнювати його з аналогом на основі іншої модифікації нейронної мережі та іншого трекера.

Також було розглянуто різні типи згорткових нейронних мереж та для практичної реалізації програмного модуля відстеження та підрахунку об’єктів на відео обрано нейронну мережу YOLOv8 як найбільш перспективну, проаналізовано її архітектуру. Було розроблено модель відстеження та підрахунку об’єктів на відео на основі нейронної мережі YOLOv8 та трекера Deep SORT, робота якого заснована на використанні відстані Махалонобіса та фільтра Калмана. Було розроблено структуру процесів обробки інформації програмного модуля відстеження та підрахунку об’єктів на відео та алгоритм його роботи.

У практичній частині було обґрунтовано вибір мови програмування Python та спеціалізованих бібліотек PyTorch та Ultralytics для програмної реалізації модуля відстеження та підрахунку об’єктів на відео. Проведено аналіз набору даних зрбражень для навчання нейронної мережі YOLOv8. Проаналізовано структуру набору даних, класи виявлених об’єктів та принципи формування анотацій для обмежувальних прямокутників. Описано основні етапи програмної реалізація та функціонування модуля відстеження та підрахунку об’єктів на відео, що складається із завантаження фреймворку нейронної мережі YOLOv8,

завантаження набору даних COCO128, навчання нейронної мережі YOLOv8, тестування нейронної мережі YOLOv8, виявлення об’єктів на відео за допомогою YOLOv8, реалізації алгоритму відстеження об’єктів на зазначеному відео, реалізації алгоритму підрахунку об’єктів на зазначеному відео, відображення результатів роботи програми.

В результаті тестування програми було доведено її повну працездатність та відповідність поставленому завданню. Для оцінки точності роботи модуля використовувався спеціалізований набір даних MOT17. Розроблений програмний модуль відстеження та підрахунку об’єктів на відео (на основі нейромережі YOLOv8x і трекера DeepSORT) переважає аналог (на основі нейромережі YOLOv8s і трекера ByteTrack) на 4,5% по влучності відстеження кількох об’єктів (MOTP) та на 2,3% по достовірності відстеження кількох об’єктів (MOTA). Тобто, мета роботи досягнута – точність відстеження та підрахунку об’єктів на відео підвищена.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Б. В. Хоцько, О. К. Колесницький. Нейромережевий модуль для рекомендації музичних треків на основі уподобань користувача. Матеріали LIV науково-технічної конференції підрозділів ВНТУ, Вінниця, 20-22 березня 2025 р. Електрон. текст. дані. 2025. URI: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/23738

2 Y. Wei, S. Tran, S. Xu, B. Kang, and M. Springer. Deep learning for retail product recognition: Challenges and techniques. Computational Intelligence and Neuroscience,, page 8875910, 2020.

3 Zou, Zhengxia, et al. “Object detection in 20 years: A survey.” Proceedings of the IEEE (2023).

4 Bichen Wu and Chenfeng Xu and Xiaoliang Dai and Alvin Wan and Peizhao Zhang and Zhicheng Yan and Masayoshi Tomizuka and Joseph Gonzalez and Kurt Keutzer and Peter Vajda, Visual Transformers: Token-based Image Representation and Processing for Computer Vision, 2020, https://arxiv.org/abs/2006.03677.

5 Soleimanitaleb, Zahra, and Mohammad Ali Keyvanrad. “Single object tracking: a survey of methods, datasets, and evaluation metrics.” arXiv preprint arXiv:2201.13066 (2022).

6 Bewley, Alex, et al. “Simple online and realtime tracking.” 2016 IEEE international conference on image processing (ICIP). IEEE, 2016.

7 Bashar, Mk, et al. “Multiple object tracking in recent times: a literature review.” arXiv preprint arXiv:2209.04796 (2022).

8 Ciaparrone, Gioele, et al. “Deep learning in video multi-object tracking: A survey.” Neurocomputing 381 (2020): 61-88.

9 Руденко О.В. Штучні нейронні мережі: Навчальний посібник / О.В.Руденко, Є.В.Бодянський. - Харків : ТОВ «Компанія СМІТ», 2006. — 404 с. - ISBN 966-8630-73-X.

10 R-CNN – Region-Based Convolutional Neural Networks https://www.geeksforgeeks.org/r-cnn-region-based-cnns/

11 Comparing KerasCV YOLOv8 Models on the Global Wheat Data 2020, https://learnopencv.com/comparing-kerascv-yolov8-models/

12 T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick. Microsoft COCO: Common objects in context. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, Computer Vision – ECCV 2014, pages 740–755, Cham, 2014. Springer International Publishing.

13 YOLOv8 : Comprehensive Guide to State Of The Art Object Detection, [Електронний ресурс] – Режим доступу: https://learnopencv.com/ultralytics-yolov8/

14 Відстань Махаланобіса, [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%B4%D1%81%D1%82%D0%B0%D0%BD%D1%8C_%D0%9C%D0%B0%D1%85%D0%B0%D0%BB%D0%B0%D0%BD%D0%BE%D0%B1%D1%96%D1%81%D0%B0

15 Фільтр Калмана, [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%A4%D1%96%D0%BB%D1%8C%D1%82%D1%80_%D0%9A%D0%B0%D0%BB%D0%BC%D0%B0%D0%BD%D0%B0

16 Zhizhong, Huang, et al. “Point, Segment and Count: A Generalized Framework for Object Counting.” arXiv preprint arXiv:2311.12386 (2023)

17 Welcome to Python [Електронний ресурс] – Режим доступу: https://www.python.org

18 PyTorch GET STARTED [Електронний ресурс] – Режим доступу: https://pytorch.org/

19 Ultralytics [Електронний ресурс] – Режим доступу: https://www.ultralytics.com/

20 OpenCV is the world's biggest computer vision library. [Електронний ресурс] – Режим доступу: https://opencv.org/

21 Методичні вказівки до виконання бакалаврської кваліфікаційної роботи для студентів денної та заочної форм навчання спеціальності 122 - «Комп’ютерні науки» / Укладачі А.А.Яровий, О.В.Сілагін, І.В.Богач. – Вінниця: ВНТУ, 2022. – 47 с.

Додаток А (обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Програмний модуль відстеження та підрахунку об'єктів на відео на основі згорткової нейронної мережі

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 21,09 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Колесницький О.К.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Хоцько Б.В.
(підпис) (прізвище, ініціали)

Додаток Б (обов'язковий)

Лістинг програми

Фрагменти лістінгу коду програми (Jupiter notebook)

```
{
  "nbformat": 4,
  "nbformat_minor": 0,
  "metadata": {
    "colab": {
      "provenance": []
    },
    "kernelspec": {
      "name": "python3",
      "display_name": "Python 3"
    },
    "language_info": {
      "name": "python"
    }
  },
  "cells": [
    {
      "cell_type": "markdown",
      "source": [
        "# Object Tracking"
      ],
      "metadata": {
        "id": "jzIZaPRfPSCw"
      }
    },
    {
      "cell_type": "markdown",
      "source": [
        "## Environment Set-UP"
      ],
      "metadata": {
        "id": "IMEpxDu2fZWy"
      }
    },
    {
      "cell_type": "code",
      "source": [
        "!conda create -n ankan python=3.9.0\n",
        "!conda activate ankan\n",
        "!conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c
nvidia\n",
        "!pip install ultralytics\n",
        "!pip install opencv-python\n",
        "!pip install jupyterlab"
      ],
      "metadata": {
        "id": "EoDiNyVEPT-n"
      },
      "execution_count": null,
      "outputs": []
    },
    {
      "cell_type": "markdown",
      "source": [
        "**you have to go to pytorch [website](https://pytorch.org/get-started/locally/)
to install your preferable cuda version.**"
      ],
      "metadata": {
        "id": "eiei34NwfeYl"
      }
    },
```

```
    {
      "cell_type": "markdown",
      "source": [
        "### Library Imports"
      ],
      "metadata": {
        "id": "Eyw7Pbapfh--"
      }
    },
    {
      "cell_type": "code",
      "source": [
        "import cv2\n",
        "from ultralytics import YOLO\n",
        "import torch\n",
        "import numpy as np"
      ],
      "metadata": {
        "id": "XLReHY80fg7l"
      },
      "execution_count": null,
      "outputs": []
    },
    {
      "cell_type": "markdown",
      "source": [
        "## Tracking with Official YOLOv8 Model"
      ],
      "metadata": {
        "id": "mQvN8pfOgS5P"
      }
    },
    {
      "cell_type": "code",
      "source": [
        "# Load an official or custom model\n",
        "model = YOLO('yolov8x.pt')  # Load an official Detect model\n",
        "\n",
        "# Perform tracking with the model\n",
        "results = model.track(source=\"path/to/your/video.mp4\", show=False, save=True,
name='output_dir_name', persist=True, classes=1)  # Tracking with default tracker\n",
        "# results = model.track(source=\"https://youtu.be/LNwODJXcvt4\", show=True,
tracker=\"bytetrack.yaml\")  # Tracking with ByteTrack tracker"
      ],
      "metadata": {
        "id": "mD5S4QQPgWB3"
      },
      "execution_count": null,
      "outputs": []
    },
    {
      "cell_type": "markdown",
      "source": [
        "# Object Counting"
      ],
      "metadata": {
        "id": "R43SY-vsi7Fo"
      }
    },
    {
      "cell_type": "markdown",
      "source": [
        "## Whole Frame based Counting"
      ],
      "metadata": {
        "id": "l4HvKLSui-Qv"
      }
    },
    {
      "cell_type": "markdown",
```

```
      "source": [
        "## Set up the Environment"
      ],
      "metadata": {
        "id": "2f5PGW2hjvFm"
      }
    },
    {
      "cell_type": "code",
      "source": [
        "# Clone Ultralytics repo\n",
        "git clone https://github.com/ultralytics/ultralytics\n",
        "\n",
        "# Navigate to the local directory\n",
        "cd ultralytics/examples/YOLOv8-Region-Counter"
      ],
      "metadata": {
        "id": "c9Tza7E9j8EE"
      },
      "execution_count": null,
      "outputs": []
    },
    {
      "cell_type": "markdown",
      "source": [
        "### Now You have to go to the `ultralytics/examples/YOLOv8-Region-
Counter/yolov8_region_counter.py` and replace the updated `yolov8_region_counter.py` file
provided."
      ],
      "metadata": {
        "id": "f2dOsNnxkPR4"
      }
    },
    {
      "cell_type": "markdown",
      "source": [
        "## Run Inference"
      ],
      "metadata": {
        "id": "-lFSXaBfm45d"
      }
    },
    {
      "cell_type": "code",
      "source": [
        "# Run Inference on Official/Custom Models\n",
        "!python ultralytics/examples/YOLOv8-Region-Counter/yolov8_region_counter.py --
source \"path/to/your/video.mp4\" --weights \"path/to/yout/model.pt\" --view-img --save-
img --classes 2"
      ],
      "metadata": {
        "id": "Fexp-L-IjSg2"
      },
      "execution_count": null,
      "outputs": []
    }
  ]
}
```

Фрагменти лістінгу коду програми yolov8_region_counter.py

```
import argparse
from collections import defaultdict
from pathlib import Path

import cv2
import numpy as np
from shapely.geometry import Polygon
```

```
from shapely.geometry.point import Point

from ultralytics import YOLO
from ultralytics.utils.files import increment_path
from ultralytics.utils.plotting import Annotator, colors

track_history = defaultdict(list)

current_region = None
counting_regions = [

    {
        "name": "YOLOv8 Rectangle Region",
        "polygon": Polygon([(0, 0), (0, 1280), (720, 1280), (720, 0)]),  # Polygon points
(tl,bl,br,tr)
        "counts": 0,
        "dragging": False,
        "region_color": (37, 255, 225),  # BGR Value
        "text_color": (0, 0, 0),  # Region Text Color
    },
]


def mouse_callback(event, x, y, flags, param):
    """
    Handles mouse events for region manipulation.

    Parameters:
        event (int): The mouse event type (e.g., cv2.EVENT_LBUTTONDOWN).
        x (int): The x-coordinate of the mouse pointer.
        y (int): The y-coordinate of the mouse pointer.
        flags (int): Additional flags passed by OpenCV.
        param: Additional parameters passed to the callback (not used in this function).

    Global Variables:
        current_region (dict): A dictionary representing the current selected region.

    Mouse Events:
        - LBUTTONDOWN: Initiates dragging for the region containing the clicked point.
        - MOUSEMOVE: Moves the selected region if dragging is active.
        - LBUTTONUP: Ends dragging for the selected region.

    Notes:
        - This function is intended to be used as a callback for OpenCV mouse events.
        - Requires the existence of the 'counting_regions' list and the 'Polygon' class.

    Example:
        >>> cv2.setMouseCallback(window_name, mouse_callback)
    """
    global current_region

    # Mouse left button down event
    if event == cv2.EVENT_LBUTTONDOWN:
        for region in counting_regions:
            if region["polygon"].contains(Point((x, y))):
                current_region = region
                current_region["dragging"] = True
                current_region["offset_x"] = x
                current_region["offset_y"] = y

    # Mouse move event
    elif event == cv2.EVENT_MOUSEMOVE:
        if current_region is not None and current_region["dragging"]:
            dx = x - current_region["offset_x"]
            dy = y - current_region["offset_y"]
            current_region["polygon"] = Polygon(
                [(p[0] + dx, p[1] + dy) for p in
current_region["polygon"].exterior.coords]
            )
            current_region["offset_x"] = x
```

```
            current_region["offset_y"] = y

    # Mouse left button up event
    elif event == cv2.EVENT_LBUTTONUP:
        if current_region is not None and current_region["dragging"]:
            current_region["dragging"] = False


def run(
    weights="yolov8n.pt",
    source=None,
    device="cpu",
    view_img=False,
    save_img=False,
    exist_ok=False,
    classes=None,
    line_thickness=2,
    track_thickness=2,
    region_thickness=2,
):
    """
    Run Region counting on a video using YOLOv8 and ByteTrack.

    Supports movable region for real time counting inside specific area.
    Supports multiple regions counting.
    Regions can be Polygons or rectangle in shape

    Args:
        weights (str): Model weights path.
        source (str): Video file path.
        device (str): processing device cpu, 0, 1
        view_img (bool): Show results.
        save_img (bool): Save results.
        exist_ok (bool): Overwrite existing files.
        classes (list): classes to detect and track
        line_thickness (int): Bounding box thickness.
        track_thickness (int): Tracking line thickness
        region_thickness (int): Region thickness.
    """
    vid_frame_count = 0

    # Check source path
    if not Path(source).exists():
        raise FileNotFoundError(f"Source path '{source}' does not exist.")

    # Setup Model
    model = YOLO(f"{weights}")
    model.to("cuda") if device == "0" else model.to("cpu")

    # Extract classes names
    names = model.model.names

    # Video setup
    videocapture = cv2.VideoCapture(source)
    frame_width, frame_height = int(videocapture.get(3)), int(videocapture.get(4))
    fps, fourcc = int(videocapture.get(5)), cv2.VideoWriter_fourcc(*"mp4v")

    # Output setup
    save_dir = increment_path(Path("ultralytics_rc_output") / "exp", exist_ok)
    save_dir.mkdir(parents=True, exist_ok=True)
    video_writer = cv2.VideoWriter(str(save_dir / f"{Path(source).stem}.mp4"), fourcc,
fps, (frame_width, frame_height))

    # Iterate over video frames
    while videocapture.isOpened():
        success, frame = videocapture.read()
        if not success:
            break
        vid_frame_count += 1
```

```
        # Extract the results
        results = model.track(frame, persist=True, classes=classes)

        if results[0].boxes.id is not None:
            boxes = results[0].boxes.xyxy.cpu()
            track_ids = results[0].boxes.id.int().cpu().tolist()
            clss = results[0].boxes.cls.cpu().tolist()

            annotator = Annotator(frame, line_width=line_thickness, example=str(names))

            for box, track_id, cls in zip(boxes, track_ids, clss):
                annotator.box_label(box, str(names[cls]), color=colors(cls, True))
                bbox_center = (box[0] + box[2]) / 2, (box[1] + box[3]) / 2  # Bbox center

                track = track_history[track_id]  # Tracking Lines plot
                track.append((float(bbox_center[0]), float(bbox_center[1])))
                if len(track) > 30:
                    track.pop(0)
                points = np.hstack(track).astype(np.int32).reshape((-1, 1, 2))
                cv2.polylines(frame, [points], isClosed=False, color=colors(cls, True),
thickness=track_thickness)

                # Check if detection inside region
                for region in counting_regions:
                    if region["polygon"].contains(Point((bbox_center[0],
bbox_center[1]))):
                        region["counts"] += 1

        # Draw regions (Polygons/Rectangles)
        for region in counting_regions:
            region_label = str(region["counts"])
            region_color = region["region_color"]
            region_text_color = region["text_color"]

            polygon_coords = np.array(region["polygon"].exterior.coords, dtype=np.int32)
            centroid_x, centroid_y = int(region["polygon"].centroid.x),
int(region["polygon"].centroid.y)

            text_size, _ = cv2.getTextSize(
                region_label, cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.7,
thickness=line_thickness
            )
            text_x = centroid_x - text_size[0] // 2
            text_y = centroid_y + text_size[1] // 2
            cv2.rectangle(
                frame,
                (text_x - 5, text_y - text_size[1] - 5),
                (text_x + text_size[0] + 5, text_y + 5),
                region_color,
                -1,
            )
            cv2.putText(
                frame, region_label, (text_x, text_y), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
region_text_color, line_thickness
            )
            cv2.polylines(frame, [polygon_coords], isClosed=True, color=region_color,
thickness=region_thickness)

        if view_img:
            if vid_frame_count == 1:
                cv2.namedWindow("Ultralytics YOLOv8 Region Counter Movable")
                cv2.setMouseCallback("Ultralytics YOLOv8 Region Counter Movable",
mouse_callback)
            cv2.imshow("Ultralytics YOLOv8 Region Counter Movable", frame)

        if save_img:
            video_writer.write(frame)

        for region in counting_regions:  # Reinitialize count for each region
            region["counts"] = 0
```

```
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break

    del vid_frame_count
    video_writer.release()
    videocapture.release()
    cv2.destroyAllWindows()


def parse_opt():
    """Parse command line arguments."""
    parser = argparse.ArgumentParser()
    parser.add_argument("--weights", type=str, default="yolov8n.pt", help="initial
weights path")
    parser.add_argument("--device", default="", help="cuda device, i.e. 0 or 0,1,2,3 or
cpu")
    parser.add_argument("--source", type=str, required=True, help="video file path")
    parser.add_argument("--view-img", action="store_true", help="show results")
    parser.add_argument("--save-img", action="store_true", help="save results")
    parser.add_argument("--exist-ok", action="store_true", help="existing project/name
ok, do not increment")
    parser.add_argument("--classes", nargs="+", type=int, help="filter by class: --
classes 0, or --classes 0 2 3")
    parser.add_argument("--line-thickness", type=int, default=2, help="bounding box
thickness")
    parser.add_argument("--track-thickness", type=int, default=2, help="Tracking line
thickness")
    parser.add_argument("--region-thickness", type=int, default=4, help="Region
thickness")

    return parser.parse_args()


def main(opt):
    """Main function."""
    run(**vars(opt))


if __name__ == "__main__":
    opt = parse_opt()
    main(opt)
```

Фрагменти лістінгу коду програми frame_corner_point.py

```
import cv2

# Function to get the four corner coordinates of a frame
def get_corner_coordinates(frame):
    height, width = frame.shape[:2]
    top_left = (0, 0)
    top_right = (width, 0)
    bottom_left = (0, height)
    bottom_right = (width, height)
    return top_left, top_right, bottom_left, bottom_right


# Path to the video file
video_path = "path/to/your/video.mp4"

# Open the video
cap = cv2.VideoCapture(video_path)

# Check if video opened successfully
if not cap.isOpened():
    print("Error: Could not open video.")
    exit()
```

```
# Read the video frame by frame
while True:
    ret, frame = cap.read()

    # If frame is read correctly ret is True
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break

    # Get corner coordinates
    corners = get_corner_coordinates(frame)
    print("Corner coordinates:", corners)

    # You can also display the frame with corner points
    # for corner in corners:
    #     cv2.circle(frame, corner, 5, (0, 0, 255), -1)
    # cv2.imshow('Frame with Corners', frame)

    # Press 'q' to exit the loop
    if cv2.waitKey(1) == ord("q"):
        break

# When everything done, release the video capture object
cap.release()
cv2.destroyAllWindows()
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

«ПРОГРАМНИЙ МОДУЛЬ ВІДСТЕЖЕННЯ ТА ПІДРАХУНКУ ОБ'ЄКТІВ НА ВІДЕО НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ»

Виконав: студент 4 курсу,групи 3КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Хоцько Б. В.
(прізвище та ініціали)

Керівник: к.т.н., проф. каф.КН
Колесницький О. К.
(прізвище та ініціали)
«03» червня 2025 р.

Вінниця ВНТУ - 2025 рік

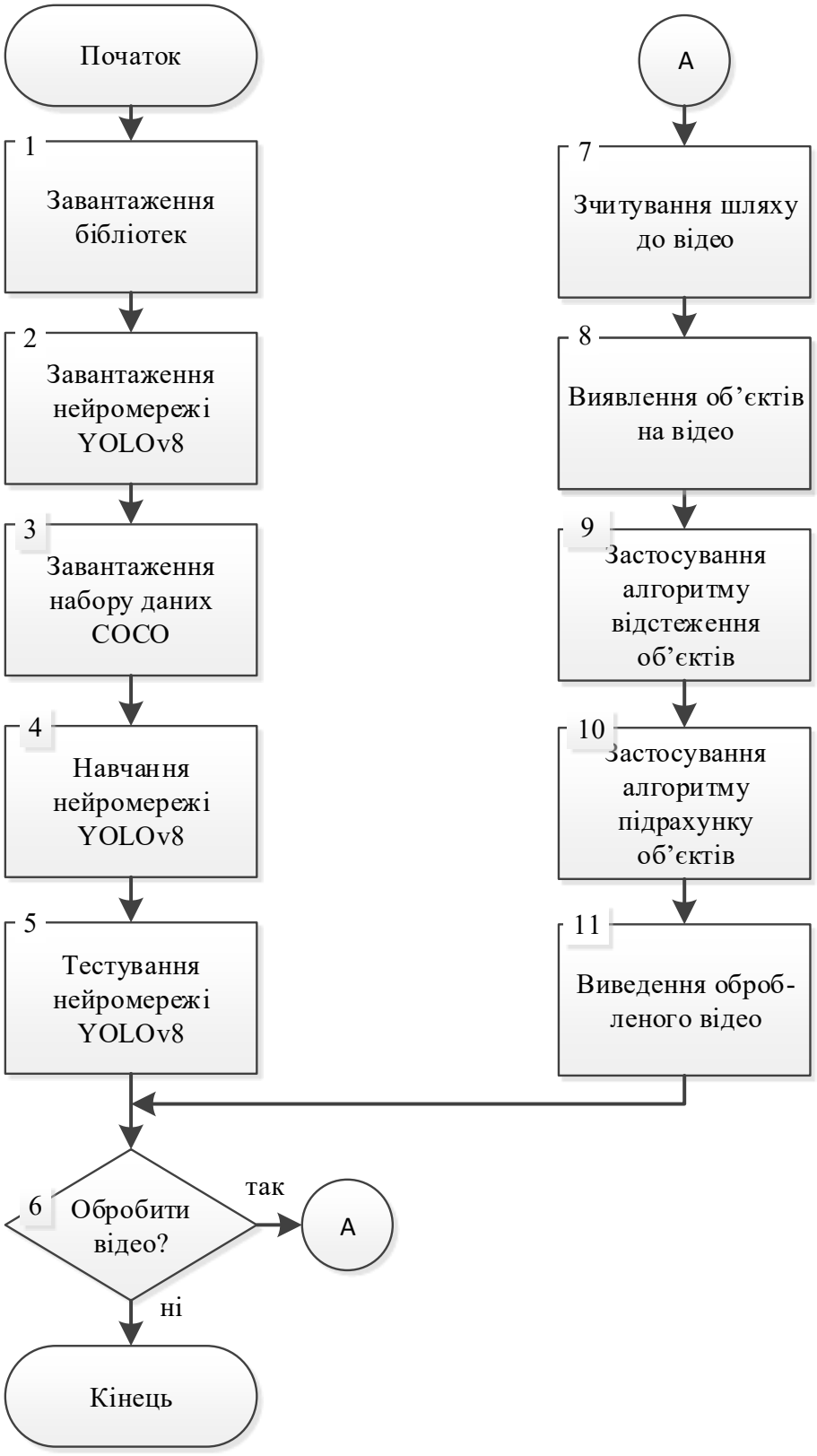


Рисунок В.1 – Схема алгоритму роботи програмного модуля відстеження та підрахунку об’єктів на відео

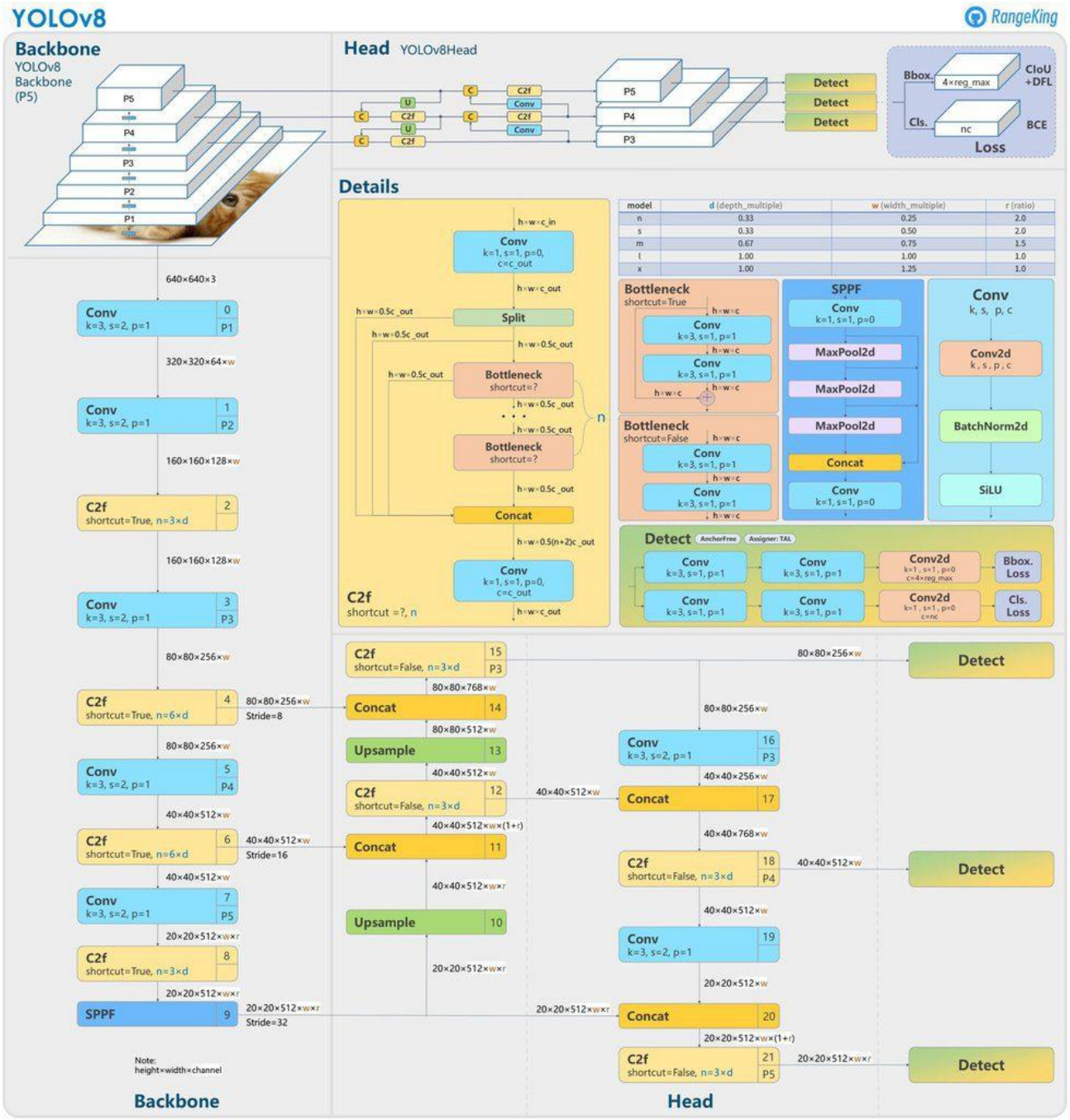


Рисунок В.2 – Архітектура YOLOv8

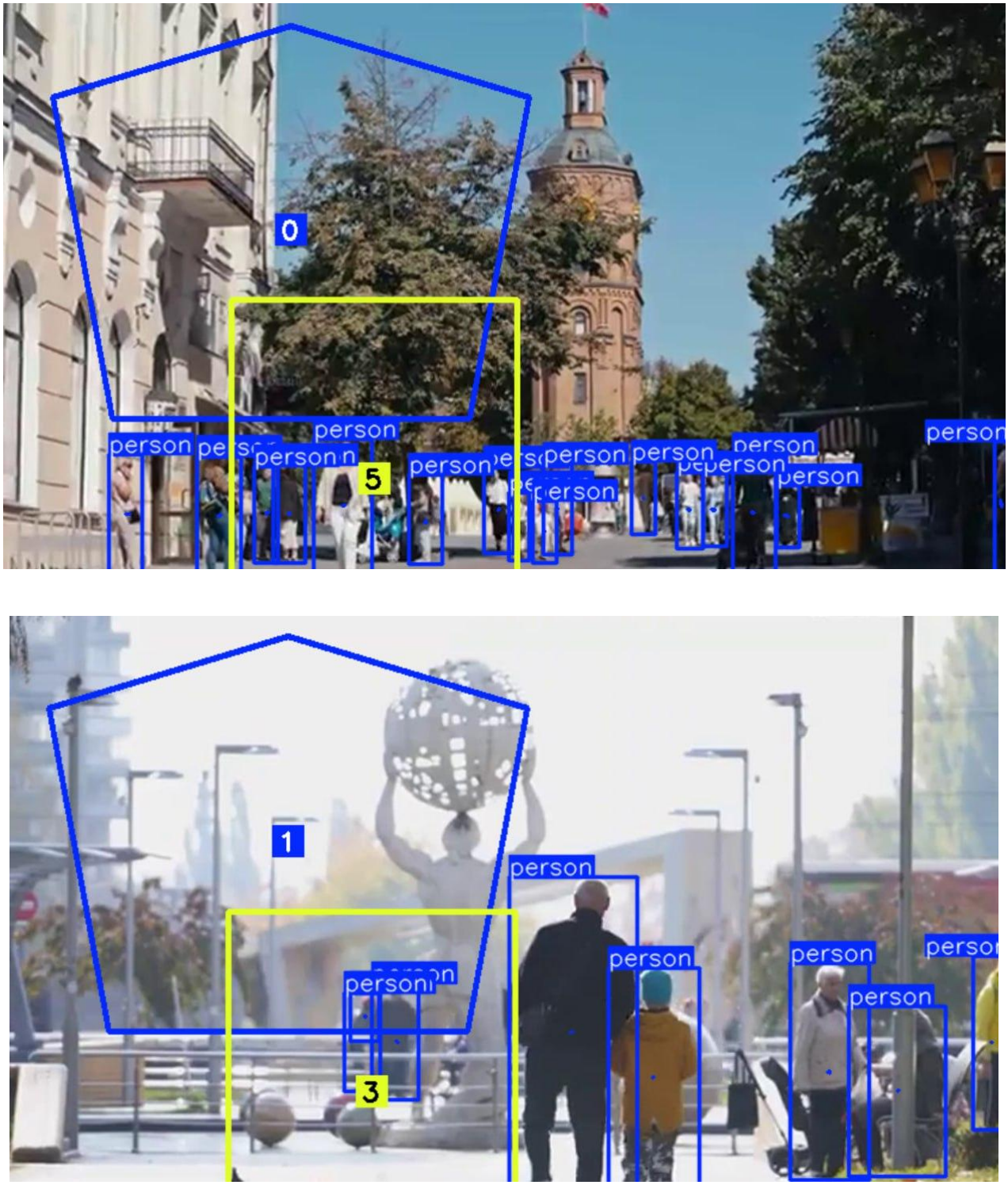


Рисунок В.3 – Окремі кадри відео з результатами роботи програмного модуля

```
video 1/1 (frame 72/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 4 cars, 4 handbags, 587.2ms
video 1/1 (frame 73/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 4 cars, 4 handbags, 584.1ms
video 1/1 (frame 74/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 5 handbags, 587.4ms
video 1/1 (frame 75/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 5 handbags, 627.0ms
video 1/1 (frame 76/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 5 handbags, 612.9ms
video 1/1 (frame 77/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 5 handbags, 600.3ms
video 1/1 (frame 78/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 5 handbags, 586.8ms
video 1/1 (frame 79/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 4 handbags, 588.5ms
video 1/1 (frame 80/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 4 handbags, 588.4ms
video 1/1 (frame 81/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 4 handbags, 590.6ms
video 1/1 (frame 82/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 4 handbags, 586.5ms
video 1/1 (frame 83/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 4 handbags, 584.8ms
video 1/1 (frame 84/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 3 handbags, 579.6ms
video 1/1 (frame 85/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 3 handbags, 589.7ms
video 1/1 (frame 86/149) G:\BKRRRR\video.mp4: 384x640 9 persons, 3 cars, 1 truck, 3 handbags, 592.2ms
video 1/1 (frame 87/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 3 handbags, 591.7ms
video 1/1 (frame 88/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 3 handbags, 587.8ms
video 1/1 (frame 89/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 3 handbags, 607.0ms
video 1/1 (frame 90/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 3 handbags, 594.8ms
video 1/1 (frame 91/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 3 handbags, 604.1ms
video 1/1 (frame 92/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 3 handbags, 589.3ms
video 1/1 (frame 93/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 1 umbrella, 4 handbags, 598.2ms
video 1/1 (frame 94/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 1 umbrella, 4 handbags, 593.9ms
video 1/1 (frame 95/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 1 umbrella, 4 handbags, 589.8ms
video 1/1 (frame 96/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 truck, 1 umbrella, 4 handbags, 590.3ms
video 1/1 (frame 97/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 2 cars, 1 truck, 1 umbrella, 4 handbags, 590.2ms
video 1/1 (frame 98/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 2 cars, 1 truck, 1 umbrella, 3 handbags, 593.3ms
video 1/1 (frame 99/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 umbrella, 3 handbags, 591.2ms
video 1/1 (frame 100/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 umbrella, 3 handbags, 588.2ms
video 1/1 (frame 101/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 umbrella, 4 handbags, 583.6ms
video 1/1 (frame 102/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 umbrella, 4 handbags, 596.7ms
video 1/1 (frame 103/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 umbrella, 5 handbags, 644.0ms
video 1/1 (frame 104/149) G:\BKRRRR\video.mp4: 384x640 10 persons, 3 cars, 1 umbrella, 5 handbags, 588.8ms
video 1/1 (frame 105/149) G:\BKRRRR\video.mp4: 384x640 1 bus, 581.9ms
```

Рисунок В.4 – Приклад звіту покадрового відстеження та підрахунку об’єктів

```
****************************** MOT17-02-DPM.mp4 Evaluation ******************************
IDF1  IDP  IDR  | Rcll  Prcn  FAR   | GT   MT   PT   ML   | FP     FN     IDs   FM    | MOTA  MOTP
 31.5 55.8 22.0 |  30.6  77.9  2.70 |   62    7   19   36 |   1619  12887    87   187 |  21.5  77.9

****************************** MOT17-04-DPM.mp4 Evaluation ******************************
IDF1  IDP  IDR  | Rcll  Prcn  FAR   | GT   MT   PT   ML   | FP     FN     IDs   FM    | MOTA  MOTP
 47.2 73.1 34.9 |  36.5  82.4  3.54 |   83    6   38   39 |   3720  30195    45   239 |  28.6  82.2

****************************** MOT17-10-DPM.mp4 Evaluation ******************************
IDF1  IDP  IDR  | Rcll  Prcn  FAR   | GT   MT   PT   ML   | FP     FN     IDs   FM    | MOTA  MOTP
 42.9 56.9 34.5 |  50.2  82.9  2.03 |   57   11   24   22 |   1329   6392   104   276 |  39.1  74.7

****************************** Summary Evaluation ******************************
IDF1  IDP  IDR  | Rcll  Prcn  FAR   | GT   MT   PT   ML   | FP     FN     IDs   FM    | MOTA  MOTP
 43.0 66.4 31.8 |  37.4  81.6  2.89 |  202   24   81   97 |   6668  49474   236   702 |  28.6  79.7
```

Рисунок В.5 – Приклад оцінки продуктивності при використання 3 відеопослідовностей

Таблиця В.1 - Середні значення метрик якості роботи для розробленого модуля і аналога

№ п/п	Метрика	Розроблений модуль (YOLOv8x і DeepSORT)	Аналог (YOLOv8s і ByteTrack)
1	**MOTP** Влучність відстеження кількох об’єктів (Multiple Object Tracking Precision)	79,7 %	75,2 %
2	MOTA Достовірність відстеження кількох об’єктів (Multiple Object Tracking Accuracy)	28,6 %	26,3 %