Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

ДОСЛІДЖЕННЯ МЕТОДУ РОЗПІЗНАВАННЯ ВІЙСЬКОВОГО ТА ЦИВІЛЬНОГО ТРАНСПОРТУ У НЕПЕРЕРВНОМУ ПОТОЦІ ДАНИХ

(тема)

Виконав:
студент 2 курсу, групи ІНФм-22-2

Скударнов М.Д.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Машталір С.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Скударнову Михайлу Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методу розпізнавання військового та цивільного транспорту у неперервному потоці даних

затверджена наказом по університету від 3 листопада 2023 року № 1280Ст

2. Термін подання студентом роботи до екзаменаційної комісії 23 грудня 2023 р.

3. Вихідні дані до роботи науково-методична література, зображення цивільного транспорту, середа для створення штучно згенерованих зображень, нейронна мережа YOLO, використані програмні засоби: мова програмування Python, середовища розробки PyCharm та Google Colab.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд та порівняння методів розпізнавання об'єктів у реальному часі.
2. Створення датасету для навчання.
3. Навчання декількох моделей нейронних мереж на створеному датасеті.
4. Порівняльний аналіз навчених моделей.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми, об'єкт та мета дослідження, постановка задачі, аналіз предметної області, вихідні дані для дослідження, етапи реалізації поставленої задачі, аналіз результатів.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	03.11.2023	
2	Аналіз завдання, підбір літератури	03.11.23-06.11.23	
3	Аналіз літератури з досліджуваної проблеми	06.11.23-07.11.23	
4	Огляд матеріалів для створення датасету	07.11.23-09.11.23	
5	Формування датасету з анотаціями	09.11.23-14.11.23	
6	Програмна реалізація	14.11.23-17.11.23	
7	Оформлення пояснювальної записки	17.11.23-23.11.23	
8	Перевірка на плагіат	05.12.2023	
9	Рецензування	09.12.2023	
10	Підготовка презентації та доповіді	15.12.2023	
11	Занесення роботи в електронний архів	02.01.2024	
12	Попередній захист кваліфікаційної роботи	02.01.2024	

Дата видачі завдання 3 листопада 2023 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Машталір С.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 65 с., 1 табл., 45 рис., 41 джерело.

YOLO, YOU LOOK ONLY ONCE, ВІЙСЬКОВА ТЕХНІКА, ЦИВІЛЬНА ТЕХНІКА, НЕПЕРЕРВНИЙ ПОТІК ДАНИХ, 3D МОДЕЛІ, РОЗПІЗНАВАННЯ ОБ'ЄКТІВ.

Об'єктом дослідження є набір зображень автомобілів та 3D моделей військової техніки.

Метою дослідження є визначення ефективності моделі YOLO в задачі розпізнавання різних видів техніки. А також визначити чи можливо використовувати модель на реальних зображеннях або відео, якщо вона була навчена на 3D моделях з відео гри, тобто штучно згенерованих.

Створено власний датасет для навчання. Було навчено декілька різних моделей YOLO різного об'єму (в даному випадку мова йде про нейрони/параметри нейронної мережі). Досліджено ефективність кожної з них та знайдено оптимальна модель для вирішення задачі.

У результаті дослідження була отримана програма для відстежування техніки у неперервному потоці даних.

YOLO, YOU LOOK ONLY ONCE, MILITARY VEHICLES, CIVIL VEHICLES, CONTINUOUS DATA FLOW, 3D MODELS, OBJECT RECOGNITION.

The object of the research is a set of images of cars and 3D models of military equipment.

The purpose of the research is to determine the effectiveness of the YOLO model in the task of recognizing various types of equipment. And also determine whether it is possible to use the model on real images or videos, if it was trained on 3D models from video games, i.e. artificially generated ones.

Created own dataset for training. Several different YOLO models of varying size (in this case neurons/neural network parameters) were trained. The effectiveness of each of them was studied and the optimal model for solving the problem was found.

As a result of the research, a program for tracking equipment in a continuous data stream was obtained.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 7
Вступ 8
1 Аналіз предметної сфери та постановка задачі 9
1.1 Опис предметної сфери 9
1.2 Огляд та аналіз наявних аналогів та публікацій 12
1.3 Метод розпізнавання: YOLO (You Only Look Once) 13
1.3.1 Основні особливості YOLO 14
1.3.2 Застосування в дослідженні 14
1.4 Порівняння моделі з конкурентами 15
1.4.1 Порівняння з SSD (Single Shot MultiBox Detector) 15
1.4.2 Порівняння з Faster R-CNN (Region-based Convolutional Neural Network) 15
1.4.3 Порівняння з RetinaNet 16
1.4.4 Порівняння різних версій YOLO 16
1.5 Ресурси дослідження 18
1.6 Постановка задачі дослідження 21
2 Огляд моделі нейронної мережі YOLO 23
2.1 Етичні питання розпізнавання військової та цивільної техніки 23
2.2 Виявлення об’єктів 25
2.2.1 Одно етапне виявлення об’єктів 25
2.2.2 Двоетапне виявлення об’єктів 26
2.3 Як працює YOLO 26
2.4 Структура моделей нейронної мережі YOLO 29
2.4.1 Модуль ultralytics.nn.modules.conv.Conv 31
2.4.2 Модуль ultralytics.nn.modules.block.C2f 32
2.4.3 Модуль ultralytics.nn.modules.block.SPPF 33
2.4.4 Модуль torch.nn.modules.upsampling.Upsample 34
2.4.5 Модуль ultralytics.nn.modules.conv.Concat 35

2.4.6 Модуль ultralytics.nn.modules.head.Detect 35
2.5 Оптимізатори навчання 36
3 Програмна реалізація системи для розпізнавання військової та цивільної техніки 39
3.1 Обґрунтування вибору середовища програмної реалізації 39
3.2 Збір даних 42
3.3 Попередня обробка даних 44
3.4 Навчання та відстежування об’єктів 48
3.5 Метрики порівняння моделей 49
3.6 Тестування та порівняння моделей 50
3.7 Аналіз результатів експериментів 56
Висновки 60
Перелік джерел посилання 61

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

YOLO – You Only Look Once (ви дивитеся лише раз)

R-CNN – Region-based Convolutional Neural Network (регіональна згорткова нейронної мережі)

VGG – Visual Geometry Group (група візуальної геометрії)

SSD – Single Shot MultiBox Detector

UAV – Unmanned Aerial Vehicle (безпілотний літальний апарат)

IoU – Intersection over Union (перетин через союз)

SPPF – Spatial Pyramid Pooling Fusion

DFL – Distribution Focal Loss

Adam – Adaptive Moment Estimation (оцінка адаптивного моменту)

SGD – Stochastic Gradient Descent (стохастичний градієнтний спуск)

RMSprop – Root Mean Square Propagation (середньоквадратичний розповсюдження)

AdaGrad – Adaptive Gradient Algorithm (адаптивний градієнтний алгоритм)

Nadam – Nesterov-accelerated Adaptive Moment Estimation (оцінка адаптивного моменту за допомогою Нестерова)

L-BFGS – Limited-memory Broyden-Fletcher-Goldfarb-Shanno (обмежена пам’ять Бройдена-Флетчера-Голдфарба-Шенно)

GPU – Graphics Processing Unit (графічний процесор)

cuDNN – Deep Neural Network library (бібліотека глибокої нейронної мережі)

CUDA – Compute Unified Device Architecture (обчислення уніфікованої архітектури пристрою)

MLOps – Machine Learning Operations (операції машинного навчання)

mAP – mean Average Precision (середня середня точність)

CLI – Command Line Interface (інтерфейс командного рядка)

ВСТУП

В сучасному світі зростає значущість використання технологій машинного навчання та розпізнавання об’єктів в різних сферах життя, включаючи безпеку та транспорт. З одного боку, поширення військового і цивільного транспорту вимагає надійних систем контролю та безпеки. З іншого боку, інновації в галузі розпізнавання об’єктів і потоків даних стають ключовими для удосконалення та оптимізації руху транспорту, забезпечуючи його ефективність та безпеку.

Методи розпізнавання військового та цивільного транспорту в неперервному потоці даних стають актуальним завданням для вчених та інженерів. Ця проблема включає в себе виявлення та ідентифікацію різних видів транспорту, аналіз їх руху, визначення стану та безпеки дорожнього руху. Для цього потрібні надійні алгоритми та системи, які здатні працювати в реальному часі та надавати точні результати.

Актуальність дослідження полягає у сьогоденних реаліях нашого життя. Через російську агресія гостро стало питання у пошуку цивільної техніки для порятунку життів та відстежуванні військової для формування безпечного маршруту тощо. Тому необхідно дослідити сучасний метод розпізнавання та відстежування об’єктів у даній сфері. Результати цього дослідження можуть мати значущий вплив на покращення безпеки, ефективності та комфорту в транспортному секторі, а також знайти застосування в інших галузях, де потрібно виявляти та відстежувати об’єкти у реальному часі.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної сфери

Дослідження методу розпізнавання військового та цивільного транспорту у неперервному потоці даних є актуальною та важливою предметною сферою, яка об’єднує технології розпізнавання об’єктів, аналізу великих обсягів даних і військово-цивільних досліджень. Ця область має на меті розробку та вдосконалення алгоритмів та систем, які дозволяють автоматично відокремлювати та класифікувати транспортні засоби на основі їх характеристик, що може бути корисним в різних контекстах, включаючи:

– моніторинг безпеки: автоматичне розпізнавання військового транспорту може бути важливим для виявлення можливих загроз безпеці. Наприклад, це може використовуватися на кордонах для виявлення незаконного вторгнення або пересування військової техніки;

– цивільний контроль: в цивільних сферах, таких як логістика, транспортні операції та міське планування, розпізнавання цивільного транспорту може допомогти в поліпшенні організації дорожнього руху, контролі навантаження на дороги та покращенні безпеки;

– військові дослідження: дослідженням методів розпізнавання військового транспорту може бути цікаво вивчати тактичні рухи та дислокації військових сил, а також оцінювати їхню потужність та наміри на основі аналізу руху військової техніки;

– транспортна логістика: у сферах, де важливо відстежувати переміщення великих обсягів товарів, таких як сировина або логістичні вантажі, розпізнавання транспорту може поліпшити управління логістичними мережами;

– захист інфраструктури: виявлення незаконних або потенційно небезпечних транспортних засобів може бути важливим для захисту

критичної інфраструктури, такої як аеропорти, залізничні станції або електростанції.

Дослідження в цій сфері передбачає використання передових методів машинного навчання, комп'ютерного зору, обробки сигналів та аналізу даних для розпізнавання різних видів транспорту з великою точністю. Вона також має потенціал вдосконалити різні аспекти цивільного та військового секторів, поліпшити безпеку і допомогти в управлінні ресурсами.

Щоб зменшити навантаження на фахівців з охорони та стеження, необхідно побудувати автоматичну систему виявлення військової техніки. Виявлення транспортних засобів з дрона чи будь якої іншої камери на території бойових дій є життєво важливим як для цивільних, так і для військових. Оскільки дрони стали невід'ємною частиною конфліктів, то треба використовувати їх по максимуму.

Тому необхідно дослідити можливості моделі глибокого навчання для ідентифікації та точної класифікації різних типів військових транспортних засобів, які можуть бути знайдені на полі бою. Кількість камер в зоні бойових дій зростає щодня, й тому потрібен швидкий спосіб аналізу цієї візуальної інформації. В подальшому цю систему можна буде інтегрувати в ройових алгоритм дронів або розвинути та вносити позначки одразу на мапу із зазначенням сектора вогню кожної техніки тощо.

Безпілотні дрони (БПЛА – безпілотні літальні апарати) виявилися дуже важливими у конфліктах і війнах завдяки своїм унікальним можливостям і можливостям. Ось деякі з аспектів, що підкреслюють важливість безпілотних дронів у конфліктах:

– збільшення розвідувальної здатності: дрони дозволяють здійснювати розвідку в реальному часі над великими областями і отримувати важливу інформацію про рухи ворожих військ, локації та інфраструктуру;

– удосконалення артилерійської точності: дрони можуть допомагати точно націлювати артилерійські вогонь на ворожі цілі, що зменшує ризик

нанесення шкоди цивільному населенню і має потенціал скоротити кількість цивільних жертв;

– забезпечення комунікації: дрони можуть бути використані для встановлення мережі зв’язку в зоні бойових дій, де існує ризик зрушення інфраструктури та обмеження зв’язку;

– зниження ризику для військовослужбовців: використання дронів дозволяє здійснювати важливі завдання безпеки, не виставляючи військовослужбовців на прямий вогонь і ризик;

– зменшення витрат і ресурсів: дрони можуть бути більш економічно ефективними в порівнянні з традиційними літальними апаратами і дозволяють зекономити ресурси;

– психологічний тиск: постійна присутність дронів над територією ворога може створювати психологічний тиск на противника і обмежувати його рухи та можливості;

– гуманітарні операції: дрони можуть бути використані для пошуку і рятування під час природних катастроф або гуманітарних криз.

Варто зазначити, що загальне завдання класифікації та відстежування є важливою і найскладнішою частиною загальної проблеми інтелектуального аналізу даних, оскільки воно базується на парадигмі самонавчання, тобто передбачає відсутність попередньо позначеної навчальної вибірки. У реальних умовах це завдання ускладняється тим, що при наявності масивів даних частина спостережень може бути зіпсована аномальними викидами, а частина – містити відсутні дані, тобто таблиця «об’єкт-властивість» має «порожні» чарунки. Крім того, дані можуть надходити в онлайн-режимі під час обробки, особливо для завдань, пов’язаних із Data Stream Mining і Big Data [1, 2]. Так як в даній роботі планується створити власний датасет, усі моменти з аномальними чи відсутніми даними будуть відсіяні ще на початкових етапах.

1.2 Огляд та аналіз наявних аналогів та публікацій

В ході виконання кваліфікаційної роботи було досліджено та проаналізовано існуючі рішення та публікації для розпізнавання військової техніки як на безпілотника так і на супутникових знімках.

У дослідженні [3] автори порівняли найсучасніший алгоритм глибокого навчання, а саме Faster R-CNN, і YOLOv3 для виявлення автомобілів за аерофотознімками. Дослідники також вивчали вплив гіперпараметрів на різні алгоритми та прийшли до висновку, що на конкретному наборі даних Faster R-CNN дав кращі результати щодо швидкості висновку, тоді як YOLOv3 показав кращі результати на наборі даних PSU із вхідними зображеннями розміром 320×320.

В роботі [4] були проведені тести різних структур нейронних мереж, такі як багатошаровий перцептрон, XCeption, VGG та інші авторські моделі. Як показали тести найкращого результату вдалося добитися на авторській моделі.

Також варто зазначити, що модель нейронної мережі VGG (а саме VGG19) показала не аби яку точність у розпізнаванні об’єктів з малими деталями. Як показано в дослідженні [5] ця модель досягла точності у 94,5% на літерах з арабського алфавіту. Але ця модель не розглядалася в задачі розпізнавання в реальному часі, а тому може не підійти для цього дослідження.

В роботі [6] розглядалася ефективність алгоритму YOLOv3. Однак як було згадано автором: однією з головних проблем для навчання мережі на даних, що мають військову важливість, є доступність інформації. Це ж стосується й візуальних даних. У відкритому доступі із очевидних причин та з міркувань безпеки знаходиться дуже обмежена кількість зображень військової техніки та інших стратегічних технічних об’єктів, особливо таких, що були б хоч частково наближені до ситуації їх виявлення на полі бою. Для

вирішення цієї проблеми було запропоновано розшири датасет на основі 3D моделей з відео гри, що дозволить змоделювати будь які ситуації.

В статті [7] розглядалася технологія сканування місцевості за допомогою радара, який створює дво- або тревимірну проекцію місцевості. Враховуючи ціну та мало поширеність таких приладів, даний метод мало чим може бути корисним в реальному конфлікті.

Стаття [8] присвячена розпізнаванню зруйнованих будинків з безпілотного дрону, але вже за допомогою більш сучасної моделі YOLOv8. Їм вдалося досягти точності в 85 відсотків, що є безсумнівно гарним результатом, враховуючи малі розміри будинків відносно розміру зображення.

В статті [9] було модифіковано модель YOLOv8 для розпізнавання крихітних представників транспорту. Автори використали третю версію функції втрат Wise-IoU, а також додали два додаткові блоки: BiFormer та Focal FasterNet block. В результаті дана модель краще, на відміну від базової, розпізнавала маленькі об’єкти, такі як велосипеди чи скутери.

1.3 Метод розпізнавання: YOLO (You Only Look Once)

Метод розпізнавання, обраний для дослідження в даному контексті це YOLO (You Only Look Once), який є одним з перших та найвідоміших алгоритмів для реального часу розпізнавання об’єктів на зображеннях та відео. YOLO відзначається своєю здатністю визначати об’єкти на зображенні та вказувати їхні рамки та класи одночасно в одному проході через мережу. Перша версія YOLO була представлена в 2016 році. Однак архітектура YOLO відтоді пройшла кілька ітерацій і покращень. Перш ніж порівняємо версії цієї моделі та порівняємо з її конкурентами, розглянемо особливості.

1.3.1 Основні особливості YOLO

Швидкість: однією з основних переваг YOLO є його висока швидкість обробки зображень. Алгоритм розроблений таким чином, щоб забезпечити реальний час роботи на відеопотоку з високою швидкістю кадрів.

Одноразовий прохід (Single Shot): у випадку YOLO, аналізується весь образ одразу. Це відрізняє його від підходів, які спершу використовують виявлення рамок, а потім класифікацію об’єктів.

Доступність для реалізації: YOLO має відкритий вихідний код та доступні реалізації на різних платформах та фреймворках (наприклад, Darknet, TensorFlow, PyTorch).

Здатність розпізнавати багато класів об’єктів: YOLO зазвичай навчається розпізнавати сотні різних класів об’єктів, що робить його універсальним для багатьох застосувань.

1.3.2 Застосування в дослідженні

У контексті «Дослідження методу розпізнавання військового та цивільного транспорту у неперервному потоці даних», YOLO може бути використаний для виявлення і класифікації різних видів транспорту в реальному часі. Його висока швидкість та точність можуть сприяти вдосконаленню системи моніторингу та безпеки на дорогах або на військових об’єктах [10].

YOLO дозволяє автоматизовано визначати та слідкувати за рухом транспортних засобів, що полегшує завдання військовим та цивільним організаціям у забезпеченні безпеки та ефективного управління транспортною інфраструктурою.

1.4 Порівняння моделі з конкурентами

Порівняння моделі YOLO (You Only Look Once) з іншими моделями розпізнавання об’єктів в реальному часі може бути корисним для розуміння його переваг і обмежень. Ось порівняння YOLO з деякими іншими відомими моделями.

1.4.1 Порівняння з SSD (Single Shot MultiBox Detector)

Однією з ключових особливостей SSD є використання різних масштабів об’єктів на різних шарах мережі. Вона використовує множину шарів (multi-scale feature maps) з різних рівнів мережі для виявлення об’єктів різних розмірів.

В порівнянні з YOLO маємо наступну картину:

– швидкість: як і YOLO, SSD спроектовано для роботи в реальному часі і володіє високою швидкістю виявлення об’єктів;

– точність: в питанні точності YOLO і SSD зазвичай показують подібні результати, але точність може залежати від конкретної архітектури та набору даних;

– архітектура: SSD використовує більше шарів та механізм anchor boxes для розпізнавання об’єктів на різних масштабах.

1.4.2 Порівняння з Faster R-CNN (Region-based Convolutional Neural Network)

Однією з ключових особливостей Faster R-CNN є використання мережі для визначення пропозицій областей (регіонів), де можливо знаходяться об’єкти. Зазвичай використовуються методи, такі як Region Proposal Network (RPN), щоб генерувати пропозиції областей [11].

В порівнянні з YOLO маємо наступну картину:

– швидкість: YOLO зазвичай працює швидше ніж Faster R-CNN, оскільки у нього менше проміжних обчислень;

– точність: Faster R-CNN, зазвичай, має вищу точність в порівнянні з YOLO, особливо на великих і складних наборах даних;

– архітектура: Faster R-CNN використовує два окремі етапи: виявлення областей і класифікацію об’єктів, тоді як YOLO виконує ці завдання одночасно [12, 13].

1.4.3 Порівняння з RetinaNet

RetinaNet використовує мережу «backbone» (наприклад, ResNet або другі популярні архітектури) для витягування ознак з зображення. Після цього використовується FPN для створення піраміди ознак різних масштабів, що дозволяє розпізнавати об’єкти різних розмірів на зображеннях [14, 15].

В порівнянні з YOLO маємо наступну картину:

– швидкість: RetinaNet являє собою швидку модель, але її швидкість трохи нижча, ніж у YOLO;

– точність: RetinaNet відома своєю доброю точністю та здатністю розпізнавати невеликі об’єкти;

– архітектура: RetinaNet використовує спеціалізовані «фокусні модулі» для виявлення об’єктів на різних масштабах.

1.4.4 Порівняння різних версій YOLO

Послідовні версії YOLO (v2-v8) принесли поліпшення як у швидкості, так і у точності, роблячи їх більш конкурентоспроможними порівняно зі своєю оригінальною версією.

У другій версії вперше була введена ідея anchor boxes для поліпшення визначення положення об’єктів. Також у цій версії додано підтримку багатьох класів об’єктів. В третій версії було значно покращено точність. В четвертій, окрім збільшення точності також були використанні нові архітектурних елементів, такі як CSPDarknet53, PANet, SAM та інші.

В п’ятій версії було змінено архітектуру нейронної мережі, тепер вона базувалась на CSPDarknet53, але мала меншу кількість шарів та параметрів. Тим самим з’явилась можливість використовувати її на GPU з невеликим обсягом пам’яті.

На відміну від попередніх архітектур YOLO, які використовують методи на основі прив’язки для виявлення об’єктів, YOLOv6 обирає метод без прив’язки. Це робить YOLOv6 на 51% швидшим порівняно з більшістю детекторів об’єктів на основі прив’язки.

YOLOv7 в свою чергу забезпечує значно покращену точність виявлення об’єктів у реальному часі без збільшення витрат на логічні висновки. В порівнянні з попередніми ця модель може ефективно зменшити приблизно на 40% параметри та 50% обчислення найсучасніших виявлень об’єктів у реальному часі, а також швидше досягти висновку та вищої точності виявлення.

В YOLOv8 можна виділити наступні ключові особливості:

– зручний API (командний рядок + Python);

– швидше та точніше;

– підтримує: виявлення об’єктів, сегментацію екземпляра та класифікацію зображень;

– розширюється до всіх попередніх версій;

– нова магістральна мережа;

– нова голівка без анкерів;

– нова функція втрати.

Також для дослідження доступно декілька моделей YOLOv8, а саме YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l, YOLOv8x – нано-, маленька, середня, велика та дуже велика моделі відповідно.

1.5 Ресурси дослідження

Для навчання розпізнавання автомобілів було обрано датасет UAV car dataset. Цей набір даних було експортовано через roboflow.ai 3 липня 2022 р. (рис. 1.1) [16].

Рисунок 1.1 – Приклад зображень з датасету автомобілів

Щодо військової техніки, то таких датасетів немає в відкритому доступі. Воно й не дивно через секретність. Тому було створено власний датасет зображень з відео гри. Для моделювання було обрано гру War

Thunder. Вона вже має моделі сучасної техніки та різні мапи для імітації реальної бойової ситуації, як в зелені так і в міській забудові. Використовуючи 3D моделі було зімітувано ситуації наближені до реальних (рис. 1.2 – рис. 1.5).

Рисунок 1.2 – 3D модель двох танків леопард

Рисунок 1.3 – 3D модель танка леопард в кущах

Рисунок 1.4 – 3D модель двох танків Т-72 та Т-64

Рисунок 1.5 – 3D модель трьох танків Т-72

Для виділення об’єктів було використано онлайн сервіс makesense.ai. Він надає дуже зручний інструментарій для виділення об’єктів декількох класів на одному зображені (рис. 1.6, рис. 1.7).

Рисунок 1.6 – Виділені 3D моделі в сервісі makesense.ai

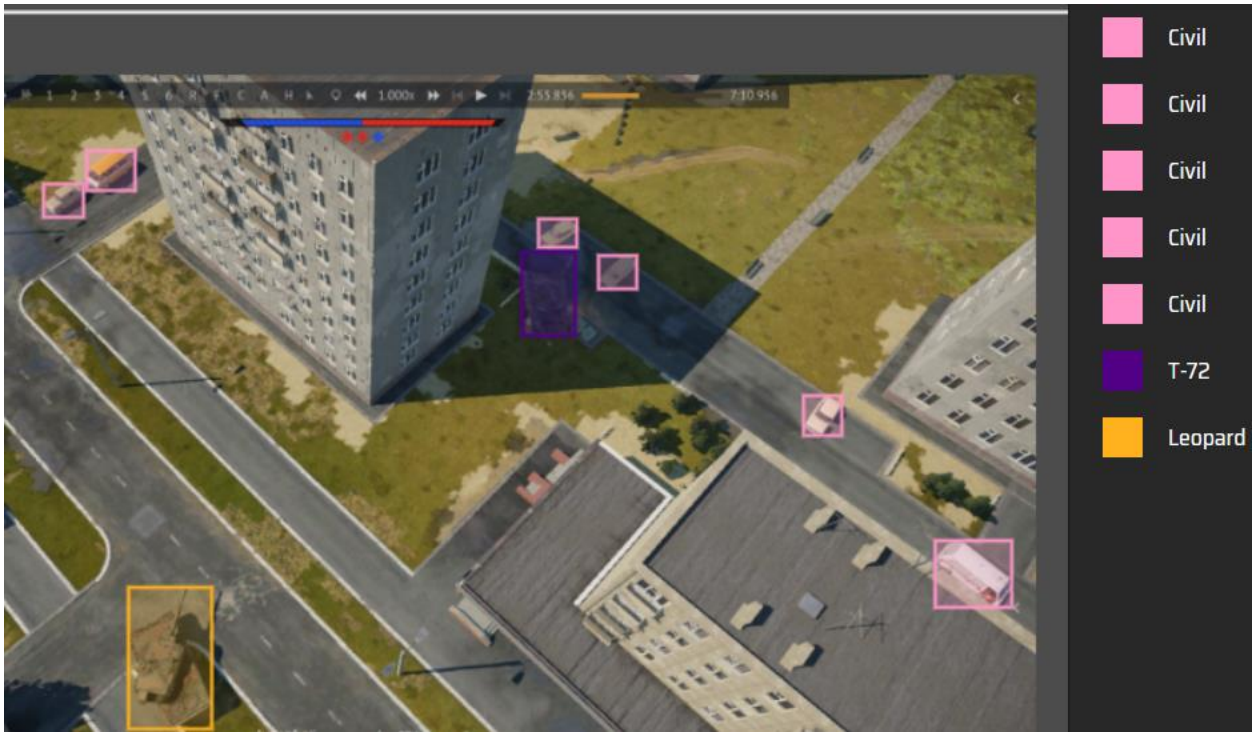

Рисунок 1.7 – Приклад виділення декількох об’єктів різних класів

1.6 Постановка задачі дослідження

Таким чином, розпізнавання військової техніки та можливість відрізнити її від цивільної є актуальним завданням для обробки і

розпізнавання зображень. Тому ставиться завдання дослідити ефективність нової версії моделі YOLOv8 в цій сфері. Та порівняти ефективності декількох представлених моделей восьмої версії. А також дослідити, чи ефективно навчати нейрону мережу на 3D моделях, а використовувати в реальному житті.

Об’єктом дослідження є набір зображень автомобілів та 3D моделей військової техніки.

Метою дослідження є визначення ефективності моделі YOLO в задачі розпізнавання різних видів техніки. А також визначити чи можливо використовувати модель на реальних зображеннях або відео, якщо вона була навчена на 3D моделях з відео гри, тобто штучно згенерованих.

Основні задачі роботи наступні:

– дослідити основні архітектури та особливості будови штучних нейронних мереж що використовуються в задачах розпізнавання об’єктів;

– дослідити основні проблеми та особливості застосування апарату згорткових нейронних мереж для розпізнавання військово та цивільного транспорту;

– обрати або створити набори вхідних даних для експериментів, провести обробку вхідних даних;

– розробити програмне забезпечення для розпізнавання військово та цивільного транспорту;

– провести експерименти та проаналізувати отримані результати.

2 ОГЛЯД МОДЕЛІ НЕЙРОННОЇ МЕРЕЖІ YOLO

2.1 Етичні питання розпізнавання військової та цивільної техніки

Розпізнавання військової та цивільної техніки за допомогою камер та відеоспостереження може породжувати етичні питання, особливо коли воно відбувається без належної контролі та дотримання правил. Використання камер для розпізнавання техніки може ставити під загрозу приватність громадян. Це особливо важливо в контексті відеоспостереження в громадських місцях, офісах, житлових будинках тощо. Порушення приватності може виникнути, якщо недостатньо регулювати, коли і де може проводитися розпізнавання техніки. Розпізнавання техніки може бути використано для спостереження за поведінкою споживачів, зокрема для рекламних цілей. Це може породжувати питання щодо того, наскільки це етично і чи зберігається конфіденційність даних.

Використання камер для розпізнавання військової техніки може бути корисним для забезпечення безпеки і військового контролю. Проте це може також підвищувати загрозу приватності і безпеки військових даних, якщо системи розпізнавання потрапляють в руки недобросовісних суб’єктів. Розпізнавання техніки може бути використано для здійснення недобросовісних дій, таких як слідкування за особами, викрадення транспорту, тероризм, шпигунство тощо. Це створює питання про те, як забезпечити безпеку та контроль над такими системами.

Використання штучного інтелекту і алгоритмів для розпізнавання техніки може вести до помилок і неправильних висновків, що може мати серйозні наслідки для осіб, які стають об’єктом розпізнавання.

Але в той самий час розпізнавання військової та цивільної техніки за допомогою камер може принести дуже велику користь, а саме підвищити загальну безпеку в суспільстві. Вони можуть використовуватися для виявлення та запобігання злочинам, таким як викрадення автомобілів або

несанкціоноване використання військової техніки. Системи розпізнавання техніки можуть бути використані для пошуку зниклих осіб, зокрема в автомобілях чи на інших видів транспорту. Це може допомогти в рятувальних операціях та пошуках загиблих чи зниклих людей.

У військовому контексті розпізнавання військової техніки може допомогти в забезпеченні національної безпеки. Воєнні сили можуть використовувати ці системи для відслідковування та ідентифікації військової техніки противника. Розпізнавання техніки може допомогти в уникненні непорозумінь та непорозумінь, особливо в зоні конфлікту. Це може сприяти розробці діалогу та вирішенню конфліктів шляхом підтримки відомого розташування та ідентифікації сторін.

У надзвичайних ситуаціях, таких як природні катастрофи чи аварії, розпізнавання техніки може допомогти координувати рятувальні операції та розподіл ресурсів ефективніше. Велика частина злочинів включає в себе використання транспортних засобів. Розпізнавання техніки може допомогти у виявленні та розслідуванні злочинів, зменшити кримінальність і підвищити відчуття безпеки у суспільстві.

Однак, слід зазначити, що, окрім безпосередньо військових застосувань, такі технології можуть дуже допомогти для своєчасного застереження цивільних громадян про можливу небезпеку, а також для отримання інформації та висвітлення подій журналістами. В деяких випадках, як от перед початком російського вторгнення в Україну 2022 року, супутникові знімки та відео з дронів на яких було виявлено скупчення техніки біля державного кордону стали для багатьох сигналом та дали підстави та інформацію про те, що потрібно готуватися до можливих провокацій або реального збройного конфлікту. Розповсюдження цих світлин в пресі також допомогло звернути увагу світової громадськості та лідерів інших держав до проблеми навіть до початку повномасштабного вторгнення. Окрім цього, такі технології можуть застосовуватись для того, щоб рятувати життя поранених військових або людей у небезпечній ситуації [6].

2.2 Виявлення об’єктів

Виявлення об’єктів – це завдання комп’ютерного зору, яке передбачає ідентифікацію та визначення місцезнаходження об’єктів на зображеннях або відео. Це важлива частина багатьох додатків, таких як спостереження, безпілотні автомобілі чи робототехніка. Алгоритми виявлення об’єктів можна розділити на дві основні категорії: детектори одноразові та двоступеневі детектори [17].

Однією з перших успішних спроб вирішення проблеми виявлення об’єктів за допомогою глибокого навчання була модель R-CNN (Regions with CNN features), розроблена Россом Гіршиком та його командою з Microsoft Research у 2014 році. Ця модель використовувала комбінацію алгоритмів пропозиції регіонів і згорткові нейронні мережі (CNN) для виявлення та локалізації об’єктів на зображеннях [18].

Алгоритми виявлення об’єктів загалом класифікуються на дві категорії залежно від того, скільки разів те саме вхідне зображення передається через мережу. Варто одразу згадати, що модель YOLO відноситься до першої категорії.

2.2.1 Одно етапне виявлення об’єктів

Одно етапне виявлення об’єктів використовує один прохід вхідного зображення для прогнозування наявності та розташування об’єктів на зображенні. Він обробляє все зображення за один прохід, що робить їх обчислювально ефективними [19].

Однак одно етапне виявлення об’єктів зазвичай менш точне, ніж інші методи, і менш ефективно для виявлення невеликих об’єктів. Такі алгоритми можна використовувати для виявлення об’єктів у реальному часі в середовищах з обмеженими ресурсами. Саме через це для YOLO обрали саме такий підхід.

2.2.2 Двоетапне виявлення об’єктів

Двоетапне виявлення об’єктів використовує два проходи вхідного зображення, щоб зробити передбачення щодо присутності та розташування об’єктів. Перший прохід використовується для створення набору пропозицій або потенційних місць розташування об’єктів, а другий прохід використовується для уточнення цих пропозицій і створення остаточних прогнозів. Цей підхід більш точний, ніж одноразове виявлення об’єкта, але також дорожчий з точки зору обчислень. Загалом, вибір між виявленням об’єктів одним або двома проходами залежить від конкретних вимог і обмежень програми [20].

2.3 Як працює YOLO

В основі YOLO лежить регресія, а не пошук цікавих частин зображення. На виході ми маємо отримати рамку з об’єктом на зображенні, тобто результат передбачення представлено векторами зі значеннями: центр рамки (координати x та y), ширина та висота рамки, значення класу та ймовірність належності об’єкта цьому класу.

Першим кроком моделі буде розбиття зображення на сітку (наприклад 19×19), потім по цій сітці йде передбачення полів (рис. 2.1) та передбачення ймовірності для кожного чарунку (рис. 2.2 [21]): вважається, що об’єкт знаходиться у певному осередку тільки в тому випадку, якщо координати центру поля прив’язки лежать у цьому осередку. Завдяки цій властивості координати центру завжди розраховуються щодо чарунку, тоді як висота та ширина розраховуються щодо всього розміру зображення.

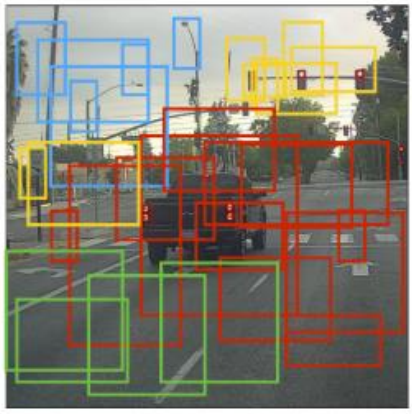

Рисунок 2.1 – Знайдені поля на яких ймовірно присутні об’єкти

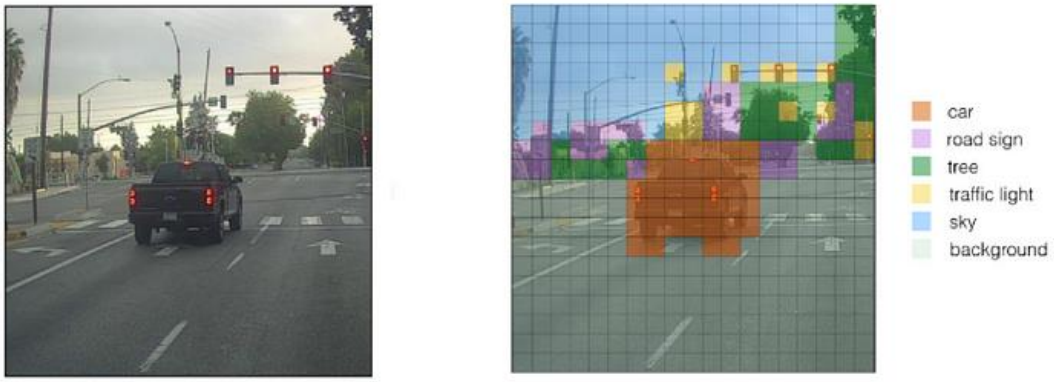

Рисунок 2.2 – Розбите зображення на сітку 19×19 з передбаченням який об’єкт знаходиться у кожному чарунку

Після першого передбачення отримуємо дуже багато полів. Для вирішення цієї проблеми алгоритм використовує немаксимальне подавлення, яке попередньо формує перетин над об’єднанням (IoU) полів з тим яке має найбільшу ймовірність належності класу (рис. 2.3 [22]).

Це допомагає нам видалити повторювані обмежувальні рамки для того самого об’єкта. Для цього ми сортуємо всі передбачення/об’єкти в порядку спадання їх достовірності. Якщо дві обмежувальні рамки вказують на один і той самий об’єкт, їх IoU буде дуже високим. У цьому випадку ми вибираємо коробку з більшою достовірністю (тобто першу коробку) і відхиляємо другу. Якщо IoU дуже низький, це, можливо, означає, що два поля вказують на різні об’єкти одного класу.

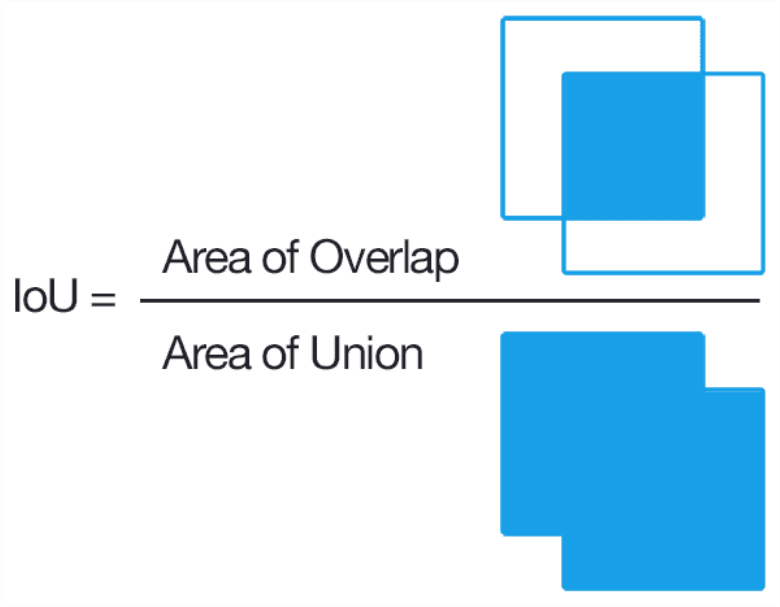

Рисунок 2.3 – Візуальний приклад розрахунку немаксимального подавлення

Поля які мають значення IoU вище деякого порога будуть викинуті. Потім береться наступне поле з найвищою ймовірністю. Процес повторюється доки не зашиться лише одне поле для кожного класу. На рисунку 2.4 [21] наведено приклад останньої ітерації з класом автомобіль [23].

Рисунок 2.4 – Остання ітерація немаксимального подавлення

2.4 Структура моделей нейронної мережі YOLO

Як вже було згадано в першому розділі, у восьмій версії представлено п’ять моделей: YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l, YOLOv8x. Нижче наведені структури кожної з них (рис. 2.5 – рис. 2.9).

	from	n	params	module	arguments
0	-1	1	464	ultralytics.nn.modules.conv.Conv	[3, 16, 3, 2]
1	-1	1	4672	ultralytics.nn.modules.conv.Conv	[16, 32, 3, 2]
2	-1	1	7360	ultralytics.nn.modules.block.C2f	[32, 32, 1, True]
3	-1	1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
4	-1	2	49664	ultralytics.nn.modules.block.C2f	[64, 64, 2, True]
5	-1	1	73984	ultralytics.nn.modules.conv.Conv	[64, 128, 3, 2]
6	-1	2	197632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
7	-1	1	295424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
8	-1	1	460288	ultralytics.nn.modules.block.C2f	[256, 256, 1, True]
9	-1	1	164608	ultralytics.nn.modules.block.SPPF	[256, 256, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
12	-1	1	148224	ultralytics.nn.modules.block.C2f	[384, 128, 1]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
15	-1	1	37248	ultralytics.nn.modules.block.C2f	[192, 64, 1]
16	-1	1	36992	ultralytics.nn.modules.conv.Conv	[64, 64, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
18	-1	1	123648	ultralytics.nn.modules.block.C2f	[192, 128, 1]
19	-1	1	147712	ultralytics.nn.modules.conv.Conv	[128, 128, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]
21	-1	1	493056	ultralytics.nn.modules.block.C2f	[384, 256, 1]
22	[15, 18, 21]	1	897664	ultralytics.nn.modules.head.Detect	[80, [64, 128, 256]]

Model summary: 225 layers, 3157200 parameters, 3157184 gradients

Рисунок 2.5 – Структура моделі yolov8n.pt

	from	n	params	module	arguments
0	-1	1	928	ultralytics.nn.modules.conv.Conv	[3, 32, 3, 2]
1	-1	1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
2	-1	1	29056	ultralytics.nn.modules.block.C2f	[64, 64, 1, True]
3	-1	1	73984	ultralytics.nn.modules.conv.Conv	[64, 128, 3, 2]
4	-1	2	197632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
5	-1	1	295424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
6	-1	2	788480	ultralytics.nn.modules.block.C2f	[256, 256, 2, True]
7	-1	1	1180672	ultralytics.nn.modules.conv.Conv	[256, 512, 3, 2]
8	-1	1	1838080	ultralytics.nn.modules.block.C2f	[512, 512, 1, True]
9	-1	1	656896	ultralytics.nn.modules.block.SPPF	[512, 512, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
12	-1	1	591360	ultralytics.nn.modules.block.C2f	[768, 256, 1]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
15	-1	1	148224	ultralytics.nn.modules.block.C2f	[384, 128, 1]
16	-1	1	147712	ultralytics.nn.modules.conv.Conv	[128, 128, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
18	-1	1	493056	ultralytics.nn.modules.block.C2f	[384, 256, 1]
19	-1	1	590336	ultralytics.nn.modules.conv.Conv	[256, 256, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]
21	-1	1	1969152	ultralytics.nn.modules.block.C2f	[768, 512, 1]
22	[15, 18, 21]	1	2147008	ultralytics.nn.modules.head.Detect	[80, [128, 256, 512]]

Model summary: 225 layers, 11166560 parameters, 11166544 gradients, 28.8 GFLOPs

Рисунок 2.6 – Структура моделі yolov8s.pt

```
                 from  n    params  module                                       arguments
  0                -1  1      1392  ultralytics.nn.modules.conv.Conv             [3, 48, 3, 2]
  1                -1  1     41664  ultralytics.nn.modules.conv.Conv             [48, 96, 3, 2]
  2                -1  2    111360  ultralytics.nn.modules.block.C2f             [96, 96, 2, True]
  3                -1  1    166272  ultralytics.nn.modules.conv.Conv             [96, 192, 3, 2]
  4                -1  4    813312  ultralytics.nn.modules.block.C2f             [192, 192, 4, True]
  5                -1  1    664320  ultralytics.nn.modules.conv.Conv             [192, 384, 3, 2]
  6                -1  4   3248640  ultralytics.nn.modules.block.C2f             [384, 384, 4, True]
  7                -1  1   1991808  ultralytics.nn.modules.conv.Conv             [384, 576, 3, 2]
  8                -1  2   3985920  ultralytics.nn.modules.block.C2f             [576, 576, 2, True]
  9                -1  1    831168  ultralytics.nn.modules.block.SPPF            [576, 576, 5]
 10                -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11            [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                -1  2   1993728  ultralytics.nn.modules.block.C2f             [960, 384, 2]
 13                -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14            [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                -1  2    517632  ultralytics.nn.modules.block.C2f             [576, 192, 2]
 16                -1  1    332160  ultralytics.nn.modules.conv.Conv             [192, 192, 3, 2]
 17           [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                -1  2   1846272  ultralytics.nn.modules.block.C2f             [576, 384, 2]
 19                -1  1   1327872  ultralytics.nn.modules.conv.Conv             [384, 384, 3, 2]
 20            [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                -1  2   4207104  ultralytics.nn.modules.block.C2f             [960, 576, 2]
 22        [15, 18, 21]  1   3822016  ultralytics.nn.modules.head.Detect           [80, [192, 384, 576]]
Model summary: 295 layers, 25902640 parameters, 25902624 gradients, 79.3 GFLOPs
```

Рисунок 2.7 – Структура моделі yolov8m.pt

```
                 from  n    params  module                                       arguments
  0                -1  1      1856  ultralytics.nn.modules.conv.Conv             [3, 64, 3, 2]
  1                -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  2                -1  3    279808  ultralytics.nn.modules.block.C2f             [128, 128, 3, True]
  3                -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  4                -1  6   2101248  ultralytics.nn.modules.block.C2f             [256, 256, 6, True]
  5                -1  1   1180672  ultralytics.nn.modules.conv.Conv             [256, 512, 3, 2]
  6                -1  6   8396800  ultralytics.nn.modules.block.C2f             [512, 512, 6, True]
  7                -1  1   2360320  ultralytics.nn.modules.conv.Conv             [512, 512, 3, 2]
  8                -1  3   4461568  ultralytics.nn.modules.block.C2f             [512, 512, 3, True]
  9                -1  1    656896  ultralytics.nn.modules.block.SPPF            [512, 512, 5]
 10                -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11            [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                -1  3   4723712  ultralytics.nn.modules.block.C2f             [1024, 512, 3]
 13                -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14            [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                -1  3   1247744  ultralytics.nn.modules.block.C2f             [768, 256, 3]
 16                -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 17           [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                -1  3   4592640  ultralytics.nn.modules.block.C2f             [768, 512, 3]
 19                -1  1   2360320  ultralytics.nn.modules.conv.Conv             [512, 512, 3, 2]
 20            [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                -1  3   4723712  ultralytics.nn.modules.block.C2f             [1024, 512, 3]
 22        [15, 18, 21]  1   5644480  ultralytics.nn.modules.head.Detect           [80, [256, 512, 512]]
Model summary: 365 layers, 43691520 parameters, 43691504 gradients, 165.7 GFLOPs
```

Рисунок 2.8 – Структура моделі yolov8l.pt

	from	n	params	module	arguments
0	-1	1	2320	ultralytics.nn.modules.conv.Conv	[3, 80, 3, 2]
1	-1	1	115520	ultralytics.nn.modules.conv.Conv	[80, 160, 3, 2]
2	-1	3	436800	ultralytics.nn.modules.block.C2f	[160, 160, 3, True]
3	-1	1	461440	ultralytics.nn.modules.conv.Conv	[160, 320, 3, 2]
4	-1	6	3281920	ultralytics.nn.modules.block.C2f	[320, 320, 6, True]
5	-1	1	1844480	ultralytics.nn.modules.conv.Conv	[320, 640, 3, 2]
6	-1	6	13117440	ultralytics.nn.modules.block.C2f	[640, 640, 6, True]
7	-1	1	3687680	ultralytics.nn.modules.conv.Conv	[640, 640, 3, 2]
8	-1	3	6969600	ultralytics.nn.modules.block.C2f	[640, 640, 3, True]
9	-1	1	1025920	ultralytics.nn.modules.block.SPPF	[640, 640, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
12	-1	3	7379200	ultralytics.nn.modules.block.C2f	[1280, 640, 3]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
15	-1	3	1948800	ultralytics.nn.modules.block.C2f	[960, 320, 3]
16	-1	1	922240	ultralytics.nn.modules.conv.Conv	[320, 320, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
18	-1	3	7174400	ultralytics.nn.modules.block.C2f	[960, 640, 3]
19	-1	1	3687680	ultralytics.nn.modules.conv.Conv	[640, 640, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]
21	-1	3	7379200	ultralytics.nn.modules.block.C2f	[1280, 640, 3]
22	[15, 18, 21]	1	8795008	ultralytics.nn.modules.head.Detect	[80, [320, 640, 640]]

Model summary: 365 layers, 68229648 parameters, 68229632 gradients, 258.5 GFLOPs

Рисунок 2.9 – Структура моделі yolov8x.pt

Не важко помітити, що послідовність шарів та блоків в цих моделях ідентична, змінюються лише кількість нейронів в шарах та кількість шарів в блоках. Для кращого розуміння розглянемо кожен модуль окремо.

2.4.1 Модуль ultralytics.nn.modules.conv.Conv

Цей модуль являє собою звичайний шар згортки. Він є одним з ключових компонентів у глибокому навчанні для обробки зображень і відео. Він використовує операцію згортки для виявлення фільтрами різних ознак на вхідних даних, таких як пікселі зображення. У шарі згортки використовуються фільтри, які є малими матрицями. Ці фільтри рухаються по вхідному зображенні, виконуючи операцію згортки для виділення певних ознак, таких як ребра, кути, текстура тощо. Шляхом навчання мережі, фільтри автоматично визначаються. Згорткова операція включає переміщення фільтра по вхідному зображенню та обчислення скалярного добутку між фільтром та відповідними пікселями на вхідному зображенні.

Результат цієї операції створює карти ознак (feature maps), які показують, де знайдені різні ознаки. Окрім виявлення ознак, шари згортки також можуть включати операції зменшення розмірності (пулінг), які допомагають зменшити обчислювальну складність мережі та зменшити кількість параметрів. Фільтри у шарі згортки навчаються в процесі зворотного поширення помилки (backpropagation) під час тренування нейронної мережі. Мережа навчається розпізнавати певні ознаки в зображеннях, що допомагає здійснювати класифікацію, виявлення об’єктів, сегментацію тощо. Шари згортки широко використовуються в задачах комп’ютерного зору, включаючи розпізнавання об’єктів, відомості про об’єкти, розпізнавання тексту, сегментацію зображень, відеоаналіз, медичну діагностику та багато інших [24].

2.4.2 Модуль ultralytics.nn.modules.block.C2f

В свою чергу цей модуль представляє блок шарів, а саме швидке впровадження вузького місця (Bottleneck) у згортковому шарі з 2 фільтрами. Це один зі способів оптимізації обчислень та кількості параметрів у глибоких нейронних мережах, особливо в глибоких згорткових нейронних мережах (CNNs). Цей підхід найчастіше використовується в мережах, які розроблені для завдань комп’ютерного зору, таких як розпізнавання об’єктів та сегментація.

Вузьке місце (Bottleneck) зазвичай включає три шари:

– згортковий шар з меншою кількістю фільтрів: у цьому шарі використовуються менше фільтрів, ніж в звичайному згортковому шарі. Наприклад, це можу бути всього 3 або 2(як в нашому випадку) фільтри. Це зменшує обчислювальну складність та кількість параметрів;

– згортковий шар із збільшеною кількістю фільтрів: після згорткового шару з меншою кількістю фільтрів додається ще один згортковий шар з

більшою кількістю фільтрів. Ці фільтри можуть бути використані для виявлення складніших ознак на вищому рівні;

– згортковий шар для зменшення кількості фільтрів: після шару з більшою кількістю фільтрів використовується згортковий шар знову для зменшення кількості фільтрів, подібно до шару вузького місця на початку.

Вузьке місце допомагає зменшити кількість параметрів та обчислень, при цьому зберігаючи здатність моделі до виявлення складних ознак. Це особливо корисно в задачах, де ресурси обмежені, або коли важливо знизити вимоги до обчислювальної потужності [25].

2.4.3 Модуль ultralytics.nn.modules.block.SPPF

Це шар об’єднання просторових пірамід який було написано ще для YOLOv5. Об’єднання просторових пірамід (Spatial Pyramid Pooling, SPP) це метод обробки зображень у глибоких згорткових нейронних мережах, який дозволяє нейронним мережам приймати зображення різних розмірів та вирівнювати їх до фіксованого розміру, щоб використовувати в одній і тій же мережі. У випадку з YOLO, SPPF використовується для розпізнавання об’єктів у реальному часі.

Основна ідея SPPF в YOLO полягає в тому, що перед остаточною класифікацією об’єктів і обчисленням розташування об’єктів, зображення розділяється на кілька сітчастих областей різного розміру та вирівнюється до фіксованого розміру, щоб використовувати однаковий розмір для всіх областей. Це дозволяє моделі працювати з зображеннями різних розмірів та виявляти об’єкти в них.

Одна з основних переваг SPPF в YOLO полягає в тому, що вона дозволяє зберігати інформацію про об’єкти різного розміру на зображенні, а також підвищує точність розпізнавання об’єктів у реальному часі.

SPPF – це важливий компонент архітектури, який сприяє здатності мережі розпізнавати об’єкти на зображеннях з різними розмірами та роздільністю, що робить її ефективною для використання у різних сценаріях, включаючи реальний час та обробку зображень з різницею в розмірі [26].

2.4.4 Модуль torch.nn.modules.upsampling.Upsample

Шар збільшення дискретизації (Upsampling Layer) в нейронних мережах використовується для збільшення розміру вхідних даних, зазвичай з метою відновлення образу або збільшення роздільності зображення. Це може бути корисним у таких завданнях, як сегментація зображень, генерація зображень високої роздільності або відновлення зображень після зменшення їх розміру (пулінгу).

Основні аспекти шару збільшення дискретизації:

– функція збільшення дискретизації, яка виконує процес збільшення розміру вхідних даних. Одним з найпоширеніших методів є білінійна або біквадратна інтерполяція, яка використовує оточуючі пікселі для розрахунку нових значень;

– фактор збільшення: цей шар може збільшувати вхідні дані в задану кількість разів. Цей фактор визначається конфігурацією мережі і вказується як параметр шару;

– завершення інформації: під час збільшення розміру вхідних даних виникає питання про те, як обробляти пропуски в інформації, які виникають внаслідок збільшення розміру. Це може бути реалізовано шляхом додавання просторової інформації з вихідного розміру до розширеного розміру або шляхом заповнення пропусків;

– застосування: шари збільшення дискретизації використовуються в різних типах нейронних мереж, таких як згорткові мережі (Convolutional Neural Networks, CNNs), рекурентні мережі (Recurrent Neural Networks,

RNNs) та глибокі мережі для обробки послідовностей та зображень. Вони можуть бути важливими в завданнях, де робота з високою роздільністю даних важлива, наприклад, у медичному зображенні, сегментації зображень, або у генерації зображень.

В архітектурі YOLO ці шари зазвичай використовуються для збільшення розмірності вихідної карти перед подальшим об’єднанням інформації від різних шарів. Це допомагає покращити точність локалізації об’єктів на зображенні [27].

2.4.5 Модуль ultralytics.nn.modules.conv.Concat

Цей модуль представляє шар об’єднання двох тензорів в заданому вимірі. Важливо, щоб виміри тензорів, які намагаються об’єднатися, були однаковими в усіх інших вимірах, окрім того виміру, який безпосередньо об’єднується. Часто цей шар використовується для об’єднання виходів із кількох шарів на різній глибині. Це дозволяє мережі об’єднувати інформацію з різних масштабів для підвищення точності виявлення об’єктів. Але також його використовують для об’єднання виходів з різних гілок нейронної мережі, так і в нашому прикладі – моделі YOLOv8 [28].

2.4.6 Модуль ultralytics.nn.modules.head.Detect

Головка (блок) виявлення (Detection Head) є однією з ключових складових моделей для завдань виявлення об’єктів на зображеннях або відео. Вона відповідає за прогнозування розташування та класифікацію об’єктів на зображенні.

В нашому випадку, блок виявлення складається зі згорткових шарів та модуля фокусних втрат (Focal Loss Distribution Integral Module, DFL) (рис. 2.10).

```
self.cv2 = nn.ModuleList(
    nn.Sequential(Conv(x, c2, 3), Conv(c2, c2, 3), nn.Conv2d(c2, 4 * self.reg_max, 1)) for x in ch)
self.cv3 = nn.ModuleList(nn.Sequential(Conv(x, c3, 3), Conv(c3, c3, 3), nn.Conv2d(c3, self.nc, 1)) for x in ch)
self.dfl = DFL(self.reg_max) if self.reg_max > 1 else nn.Identity()
```

Рисунок 2.10 – Лістинг коду, ініціалізація шарів класу DFL

Основна ідея інтегрального модуля розподілу фокусних втрат полягає в тому, щоб покращити роботу моделі в умовах розпізнавання об’єктів, коли об’єкти різного розміру зустрічаються в різних частинах зображення. Цей модуль допомагає моделі краще робити прогнози для об’єктів різного розміру та покращує точність розпізнавання.

Інтегральний модуль розподілу фокусних втрат включає в себе покращені механізми обчислення фокусної втрати та покращені метрики для оцінки роботи моделі. Він допомагає збалансувати модель, зробити її стабільнішою та ефективнішою в умовах реального світу [29].

2.5 Оптимізатори навчання

Вибір оптимізатора для навчання нейронної мережі є важливим кроком у процесі розробки моделі глибокого навчання. Різні оптимізатори можуть вести себе по-різному під час навчання і мати вплив на швидкість навчання, стійкість до перенавчання та загальну ефективність моделі [30]. Вибір конкретного оптимізатора залежить від завдання, даних і архітектури моделі. Ось кілька популярних оптимізаторів та поради щодо їхнього вибору:

– Adam (Adaptive Moment Estimation): Adam це часто використовуваний оптимізатор за замовчуванням у багатьох бібліотеках

глибокого навчання, таких як TensorFlow і PyTorch. Він зазвичай добре працює для багатьох завдань і вимагає менше налаштувань гіперпараметрів. Він рекомендується для багатьох задач, особливо при використанні згорткових нейронних мереж [31];

– SGD (Stochastic Gradient Descent): SGD є класичним методом оптимізації. Хоча він може бути менш ефективним за Adam, він добре підходить для деяких задач та може працювати добре з додатковими техніками, такими як інерція та оптимізація навчання швидкістю (learning rate scheduling) [32];

– RMSprop (Root Mean Square Propagation): RMSprop це інший популярний оптимізатор, який може бути ефективним в деяких задачах. Він добре підходить для завдань зі змінними швидкостями навчання і може допомогти покращити стабільність навчання [33];

– AdaGrad (Adaptive Gradient Algorithm): AdaGrad пристосовує швидкість навчання для кожного параметра на основі історії градієнтів. Він може бути корисним в задачах з рідкими градієнтами [34];

– Nadam (Nesterov-accelerated Adaptive Moment Estimation): Nadam поєднує інерцію з методом оптимізації Adam, що допомагає прискорити навчання і зберегти стійкість до перенавчання [35];

– L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno): L-BFGS це метод оптимізації, який може бути ефективним для навчання невеликих моделей, особливо в задачах з рідкими градієнтами [36].

Отже, в даній роботі буде використовуватися саме оптимізатор Adam через його універсальність. Розглянемо його особливості детальніше.

Він є поєднанням методів адаптивного градієнтного спуску та експоненціального згладжування. Adam дозволяє ефективно навчати моделі, вирішує проблему зневаження градієнта (vanishing gradient), і надає декілька переваг порівняно з класичними методами оптимізації, такими як SGD (Stochastic Gradient Descent).

Основні компоненти методу оптимізації Adam включають наступне:

– швидкість навчання (Learning Rate): як і в інших методах оптимізації, встановлює швидкість зміни параметрів під час навчання. Adam автоматично адаптує швидкість навчання для кожного параметра на основі історії градієнтів;

– інерція першого моменту (First Moment): це величина, яка відстежує середній градієнт параметрів моделі. Вона допомагає регулювати швидкість навчання в залежності від градієнтів;

– інерція другого моменту (Second Moment): це величина, яка відстежує середній квадрат градієнту параметрів моделі. Вона допомагає нормалізувати швидкість навчання в залежності від дисперсії градієнтів;

– згладження (Smoothing): Adam використовує експоненціальне згладжування для підрахунку обидві інерції (першого і другого моменту) з градієнтів;

– корекція на зміщення (Bias Correction): Adam використовує корекцію на зміщення, оскільки початкові оцінки інерцій можуть бути зміщеними;

– регуляризація (Regularization): Adam також може містити елементи регуляризації для запобігання перенавчанню (overfitting).

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ДЛЯ РОЗПІЗНАВАННЯ ВІЙСЬКОВОЇ ТА ЦИВІЛЬНОЇ ТЕХНІКИ

3.1 Обґрунтування вибору середовища програмної реалізації

У 2023 році нікого не здивує вибір мови Python для вирішення задачі машинного навчання. Бо вона має значущі переваги в порівнянні з іншими. Python відомий своєю простотою та зрозумілістю синтаксису. Це робить його ідеальним вибором для початківців і професіоналів, які працюють у сферах машинного навчання та комп'ютерного зору. Python має велику кількість бібліотек та інструментів, які розроблені спеціально для машинного навчання та комп'ютерного зору. Бібліотеки, такі як TensorFlow, PyTorch, scikit-learn, OpenCV, і багато інших, надають реалізації різних алгоритмів та моделей, що спрощує розробку та дослідження. Python має велику та активну спільноту користувачів, що робить його ресурсом для отримання підтримки, допомоги та вирішення проблем. Python є крос платформною мовою, що дозволяє виконувати свій код на різних операційних системах без змін. Це полегшує розробку та роботу з різними пристроями. Python легко інтегрується з іншими інструментами, зокрема базами даних, бібліотеками візуалізації, фреймворками для роботи з вебсервісами та багатьма іншими. Це дозволяє зручно обробляти дані та реалізовувати різні компоненти системи. Python підходить для різних видів завдань у машинному навчанні та розпізнаванні об'єктів, від класифікації та сегментації зображень до обробки природної мови та аналізу даних. Python дозволяє створювати власні бібліотеки та фреймворки, а також використовувати мову для розширення функціональності існуючих рішень [37].

Як вже було написано вище, Python має безліч бібліотек, для комп'ютерного зору найпопулярнішими є OpenCV, TensorFlow, PyTorch. Усі вони надають інструменти для виявлення, вимірювань, відстеження об'єктів

та багато іншого. А також містять набір моделей для виявлення об’єктів, включаючи SSD (Single Shot MultiBox Detector), Faster R-CNN та інші [38].

У рамках кваліфікаційної роботи буде використана бібліотека ultralytics [39]. Вона базується на бібліотеці PyTorch. Вона містить інструменти для використання (класифікації, виявлення, сегментації, відстежування та виявлення положення об’єкта рисунок 3.1 [40]), навчання та модифікації моделі YOLO.

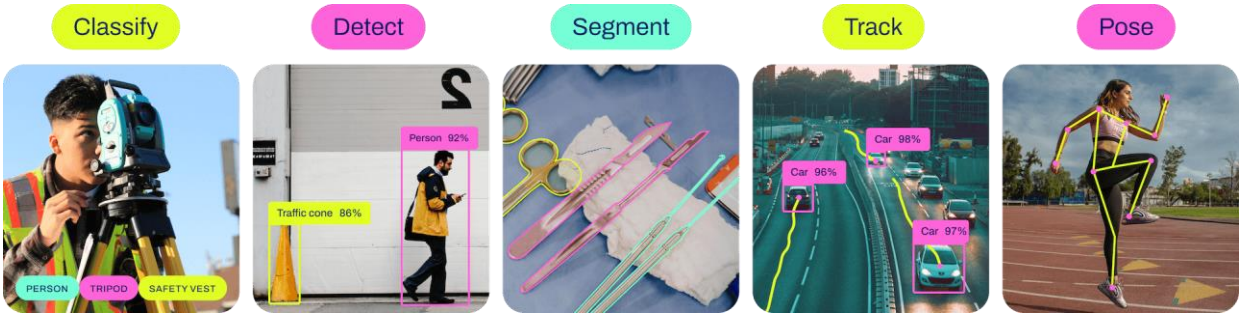

Рисунок 3.1 – Результати роботи моделі YOLO

Розглянемо апаратне середовище використане в даному досліджені. Навчання відбувалося на віртуальній машині, представленій сервісом Google Colab або Google Colaboratory, це безкоштовна платформа для виконання коду Python у хмарі, яка має численні переваги для розробників та дослідників у галузі машинного навчання та інших областях. Ось деякі з основних переваг Google Colab:

– безкоштовність: Google Colab безкоштовний для використання. Ви можете виконувати код Python без необхідності інсталювати або налаштовувати середовище на власному комп’ютері;

– готові середовище з попередньо встановленими бібліотеками: Google Colab включає популярні бібліотеки для машинного навчання, такі як TensorFlow, PyTorch, scikit-learn, і багато інших. Ви можете почати працювати без необхідності встановлення цих бібліотек вручну;

– доступ до обчислювальних ресурсів: Google Colab надає доступ до потужних графічних процесорів (GPU) та процесорів для прискорення обчислень, що особливо корисно для навчання глибоких нейронних мереж та обробки великих обсягів даних (рис. 3.2);

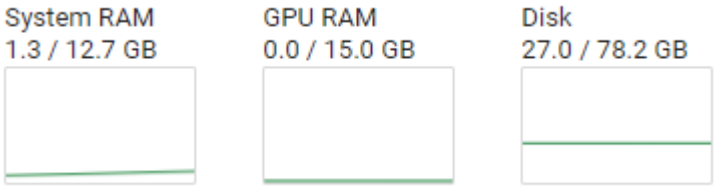

Рисунок 3.2 – Характеристики віртуальної машини предоставленої Google Colab

– зберігання в Google Drive: платформа дозволяє зберігати файли з кодом та дані в Google Drive, що дозволяє доступатися до них з будь-якого пристрою, підключеного до Інтернету.

Безпосередньо виявлення відбувалося на персональному ноутбуці з відеокартою GTX 1650, яка має 4 гігабайти відео пам’яті. Також варто зазначити, що для використання графічного ядра на персональному комп’ютері необхідно встановити додаткові інструменти/бібліотеки: набір інструментів CUDA, NVIDIA cuDNN (Deep Neural Network library) та дистрибутив Anaconda (в даній роботі було використано його мінімальну версію miniconda).

Так як навчання та безпосередньо виявлення виконуються на різних машинах, доцільно використовувати інструмент для зберігання виводів та результатів, у разі навчання це будуть ваги нейронної мережі, в разі відстежування це відеофайл або зображення з рамками. Для цього ідеально підходить сервіс ClearML.

ClearML – це платформа MLOPS (ML Ops – це парадигма, яка спрямована на надійне та ефективне розгортання та підтримку моделей машинного навчання у виробництві) з відкритим кодом [41]. По суті, це набір

інструментів, наповнений усім необхідним для переходу від експериментів до виробництва, ось два найважливіші з них:

– повноцінний менеджер експериментів, який може реєструвати, порівнювати та автоматично відтворювати будь-який експеримент, який ви проводите;

– інструмент керування версіями даних, який може відстежувати зміни у ваших наборах даних і робить їх легко доступними для будь-кого на будь-якій машині.

3.2 Збір даних

Як вже було згадано в першому розділі, більшу частину даних для навчання було взято з відео гри War Thunder. В ній є моделі сучасної військової техніки, а також тієї яка використовується в збройному конфлікті між Україною та Росією. Перш за все цікавими будуть дві моделі: танка леопарда та Т-72.

Завдяки можливості перегляду зіграних матчів, є змога поставити на паузу та облетіти техніку з різних боків, тим самим імітуючи роботу безпілотного дрону (рис. 3.3).

Рисунок 3.3 – Вибірка зі створеного датасету

Окрім танків необхідні також зображення цивільного транспорту (рис 3.4, рис. 3.5). В грі представлено декілька моделей: легкові автомобілі, автобуси, міні буси, вантажні автомобілі.

Рисунок 3.4 – Приклад моделей легкового транспорту

Рисунок 3.5 – Приклад моделей вантажного транспорту

В рамках даного дослідження усі перечисленні автомобілі будуть мати спільний клас, а саме цивільний транспорт. Створений датасет налічує 1006 зображень.

3.3 Попередня обробка даних

Нагадаю з першого розділу, для створення анотації було використано онлайн інструмент makesense.ai. Від дозволяє дуже зручно виділяти велику кількість об’єктів на одному зображені з необмеженою кількістю класів. Ось для прикладу збільшене зображення в інтерфейсі makesense.ai (рис. 3.6).

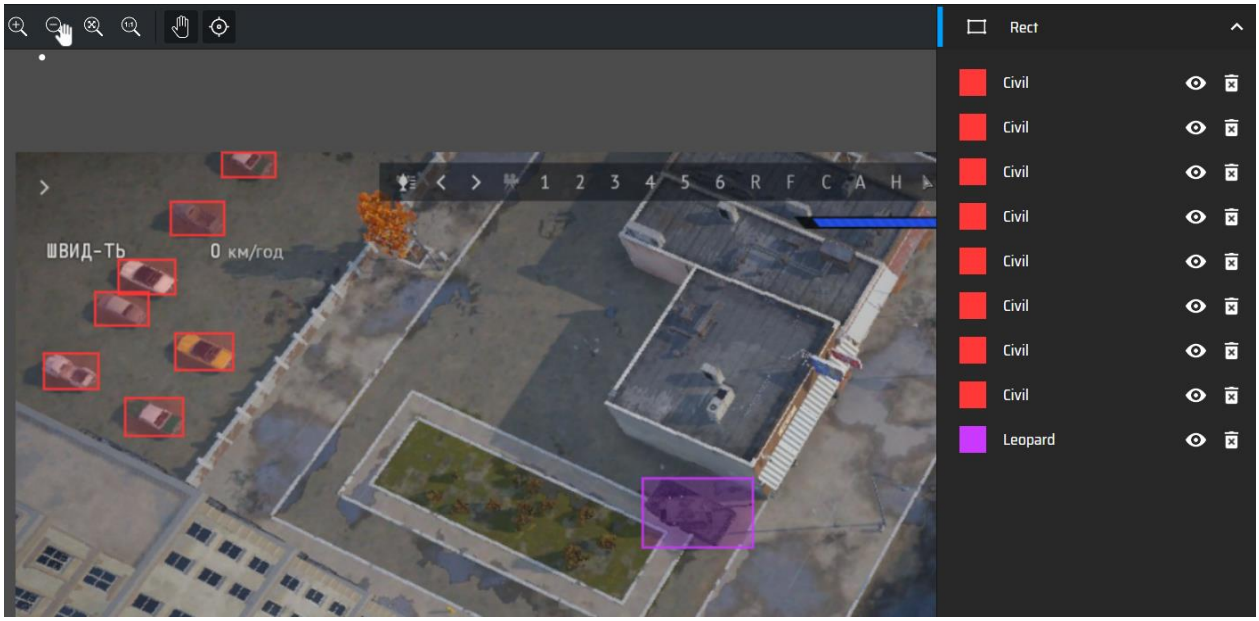

Рисунок 3.6 – Пиклад виділення великої кількості дрібних об’єктів

В результаті отримуємо архів тектових файлів. З номером класу та координатами прямокутника в якому знаходиться предстаник класу відповідно (рис. 3.7).

shot 2023.10.17 18.44.54.txt - Notepad

File Edit Format View Help

```
2 0.642803 0.643564 0.017136 0.054159
2 0.027989 0.196186 0.027291 0.032721
2 0.069878 0.237088 0.029195 0.034414
2 0.095583 0.178979 0.029195 0.033285
2 0.053694 0.140617 0.026657 0.029900
2 0.066387 0.111280 0.029195 0.031029
2 0.092092 0.059096 0.027291 0.029336
2 0.117796 0.010931 0.026657 0.021861
1 0.345011 0.323235 0.055852 0.062057
```

Рисунок 3.7 – Анотація до зображення предстевленого на рисунку 3.6

Як можна помітити, на зображеннях присутній не бажаний ігровий інтерфейс (рис. 3.8), який може непередбачувано вплинути на навчання, тому було прийнято рішення закрити його заглушкою (рис. 3.9).

Рисунок 3.8 – Не бажаний ігровий інтерфейс

```python
import os
from PIL import Image


images_path = "D:\\MyProjs\\ultralytics\\datasets\\goal\\train\\images"
path_to_save = "D:\\MyProjs\\ultralytics\\datasets\\goal\\train\\images_pasted"
dummy_path = "D:\\MyProjs\\ultralytics\\datasets\\goal\\dummy.jpg"

images = os.listdir(images_path)
dummy = Image.open(dummy_path)

for i, image in enumerate(images):
    img = Image.open(f"{images_path}\\{image}")
    img.paste(dummy, (300, 905))
    img.save(f"{path_to_save}\\{image}")
    print(f"{i} images done.")
```

Рисунок 3.9 – Код для перекриття інтерфейсу зображенням заглушкою

Лишилося лише розбити датасет на навчальний та валідаційний. Для цього скористаємося функцією sklearn.model_selection.train_test_split (рис. 3.10). В цьому коді йде поділ як файлів зображень так і анотацій до них. Змінні path_images та path_labels зберігають шлях до всього датасету. В свою чергу path_images_to_val та path_labels_to_val визначають куди зберігати зображення та анотації елементів датасету які тепер вважаємо валідаційними. Розмір валідаційного датасету визначається вхідним параметром test_size. В даному випадку це 20% від усього датасету.

Бібліотека ultralytics змінює форму зображень відповідно до вхідного параметра imgsz. Тому розмір вхідних зображень не обов’язково має бути однаковим між собою. Це дозволяє пропустити шаг нормалізації зображень по їх розміру. Також це дає не аби яку гнучкість: можна комбінувати декілька датасетів й дивитися в оригінальні зображення без їхнього спотворення.

```python
import os
from sklearn.model_selection import train_test_split
import shutil


path_images = "D:\\MyProjs\\ultralytics\\datasets\\goal\\train\\images_pasted"
path_labels = "D:\\MyProjs\\ultralytics\\datasets\\goal\\train\\labels"

path_images_to_val = "D:\\MyProjs\\ultralytics\\datasets\\goal\\val\\images"
path_labels_to_val = "D:\\MyProjs\\ultralytics\\datasets\\goal\\val\\labels"


images = os.listdir(path_images)
labels = os.listdir(path_labels)
labels.pop(0)

images_train, images_val, labels_train, labels_val = \
    train_test_split(images, labels, test_size=0.2, random_state=42)

for i in range(len(images_val)):
    image_val = images_val[i]
    label_val = labels_val[i]
    shutil.copyfile(f"{path_images}\\{image_val}", f"{path_images_to_val}\\{image_val}")
    shutil.copyfile(f"{path_labels}\\{label_val}", f"{path_labels_to_val}\\{label_val}")
    os.remove(f"{path_images}\\{image_val}")
    os.remove(f"{path_labels}\\{label_val}")
```

Рисунок 3.10 – Код для розбиття на навчальний та валідаційні датасети

Отже було отримано навчальний та перевірочний датасети, необхідно створити yaml файл, в якому буде зазначено де брати навчальний та валідаційний датасети та список імен класів. Очікується що файли анотацій лежать в сусідній директорії відносно зображень. Тобто якщо зображення для тренування знаходяться в ../train/images/, то анотації повинні бути в ../train/labels/ (рис. 3.11).

```yaml
train: /content/drive/MyDrive/YOLOv8/datasets/goal/train/images
val: /content/drive/MyDrive/YOLOv8/datasets/goal/val/images

names:
  0: T-72
  1: Leopard
  2: Civil
```

Рисунок 3.11 – Приклад .yaml файлу

3.4 Навчання та відстежування об’єктів

Тепер коли налаштовано середовище та підготовлено датасет з анотаціями, необхідно запустити навчання. Бібліотека ultralytics дозволяє зробити це двома способами: через написання скрипту на мові Python (рис. 3.12) або через інтерфейс командного рядка (Command Line Interface, CLI).

```python
from ultralytics import YOLO


if __name__ == "__main__":
    model = YOLO("yolov8x.pt")
    results = model.train(
        data='coco128.yaml',
        project='training_results',
        epochs=50,
        batch=8,
        imgsz=1024,
        device="cuda:0",
        name="goal_WT",
        amp=False
    )
```

Рисунок 3.12 – Скрипт на мові Python для початку навчання нейронної мережі

Лістинг 3.1 CLI команда:

```
!yolo task=detect mode=train model=yolov8x.pt
data=/content/drive/MyDrive/YOLOv8/yaml/coco128.yaml batch=8
project=/content/drive/MyDrive/YOLOv8/training_results epochs=50
imgsz=1024 name=goal_WT
```

де data – це yaml файл в якому міститься інформація про датасет;

project – директорія куди зберігати результати навчання/відстежування;

epochs – кількість епох навчання;

batch – розмір партії навчання;

imgsz – розмір зображень, до якого будуть вирівнюватися усі картинки в датасеті під час навчання;

name – ім’я запуску, назва папки яка буде створена в директорії project;

task – визначає тип задачі для виконання, можливі значення: detect, segment, classify, pose;

mode – визначає режим в якому модель буде працювати, можливі значення: train, val, predict, export, track, benchmark.

3.5 Метрики порівняння моделей

Після закінчення навчання, в папці з результатами будуть знадитися звіти у вигляді зображень з графіками. На них будуть присутні графіки наступних величин: box_loss, cls_loss, dfl_loss, precision, recall, mAP50, mAP50-95. Розглянемо кожну з них:

– box_loss: втрата, яка вимірює, наскільки щільно передбачені обмежувальні прямокутники розташовані до правдивого об’єкта землі;

– cls_loss: втрата, яка вимірює правильність класифікації кожної передбаченої обмежувальної рамки: кожна рамка може містити клас об’єкта або фон;

– dfl_loss: втрата яка враховує проблему дисбалансу класів під час навчання;

– precision: метрика, яка вимірює, як часто модель машинного навчання правильно прогнозує позитивний клас;

– recall: показник, який вимірює, як часто модель машинного навчання правильно визначає позитивні екземпляри (справжні позитивні результати) з усіх фактичних позитивних зразків у наборі даних;

– mAP50: показник, який оцінює продуктивність моделі, враховуючи як точність, так і запам’ятовування для кількох класів об’єктів на пороговому значенні IoU 0,5, вимірюючи, наскільки добре модель ідентифікує об’єкти з розумним перекриттям;

– mAP50-95: розширює оцінку до діапазону порогових значень IoU від 0,5 до 0,95. Цей показник особливо цінний для завдань, що вимагають точної локалізації та тонкого виявлення об’єктів.

На практиці mAP50 і mAP50-95 допомагають оцінити продуктивність моделі в різних класах і умовах, пропонуючи уявлення про точність виявлення об’єктів, враховуючи компроміс між точністю і запам’ятовуванням. Моделі з вищими балами mAP50 і mAP50-95 надійніші та підходять для таких вимогливих додатків, як автономне водіння та спостереження за системою безпеки.

3.6 Тестування та порівняння моделей

Отже переходимо до навчання. Почнемо з нано моделі, 50 епох, розмір партії 8, розмір зображення 640×640 пікселів. Отримуємо наступні результати (рис. 3.13). Також поглянемо на матрицю плутанини (рис. 3.14). Метрики протягом навчання на всіх епохах зображено на рисунку 3.15.

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
46/50      1.35G      1.014     0.7624      1.033         63        544: 100% 131/131 [00:50<00:00,  2.58it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 17/17 [00:07<00:00,  2.19it/s]
             all        261       1543      0.817      0.689       0.78      0.549

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
47/50      1.44G      1.011     0.7522      1.019         70        544: 100% 131/131 [00:51<00:00,  2.55it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 17/17 [00:06<00:00,  2.62it/s]
             all        261       1543      0.827      0.708      0.794      0.562

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
48/50      1.37G     0.9924     0.7514      1.037         65        544: 100% 131/131 [00:49<00:00,  2.64it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 17/17 [00:06<00:00,  2.44it/s]
             all        261       1543       0.79      0.709      0.783      0.558

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
49/50      1.29G      1.002     0.7418      1.015         69        544: 100% 131/131 [00:50<00:00,  2.62it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 17/17 [00:11<00:00,  1.53it/s]
             all        261       1543      0.829      0.701      0.798      0.564

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
50/50      1.32G      1.001     0.7291      1.023         74        544: 100% 131/131 [00:47<00:00,  2.77it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 17/17 [00:09<00:00,  1.72it/s]
             all        261       1543      0.821      0.715        0.8      0.573
```

Рисунок 3.13 – Виведення останніх епох навчання YOLOv8n

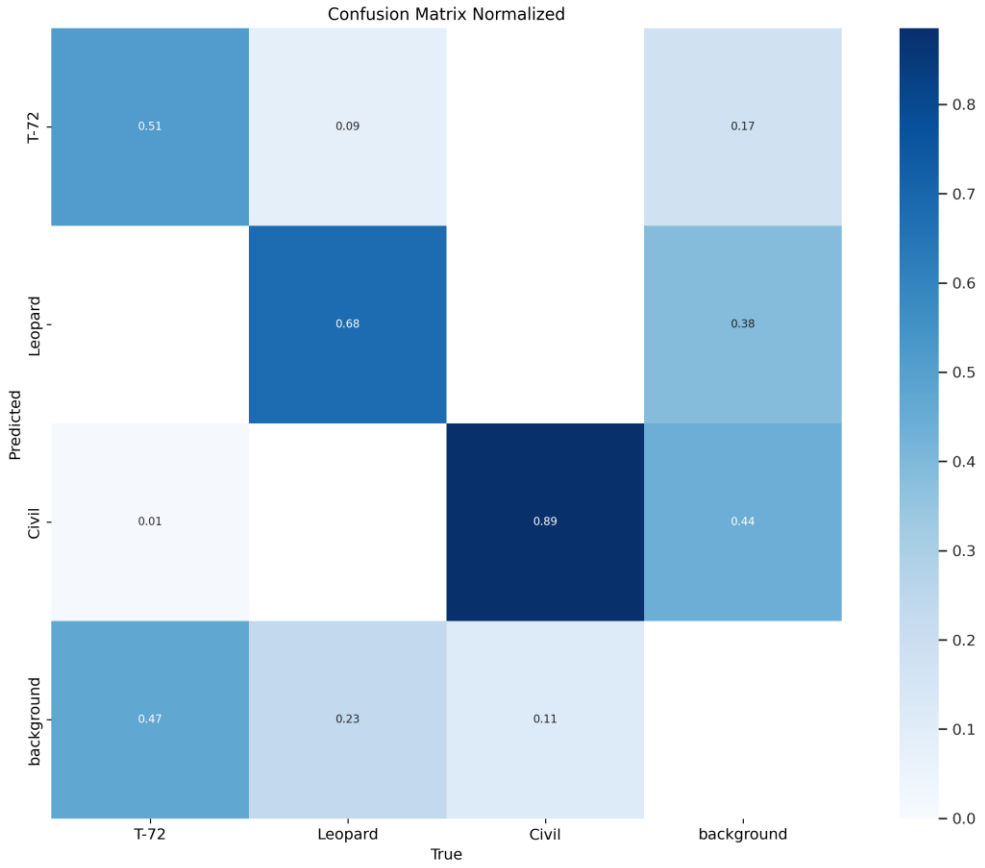

Рисунок 3.14 – Матриця плутанини YOLOv8n

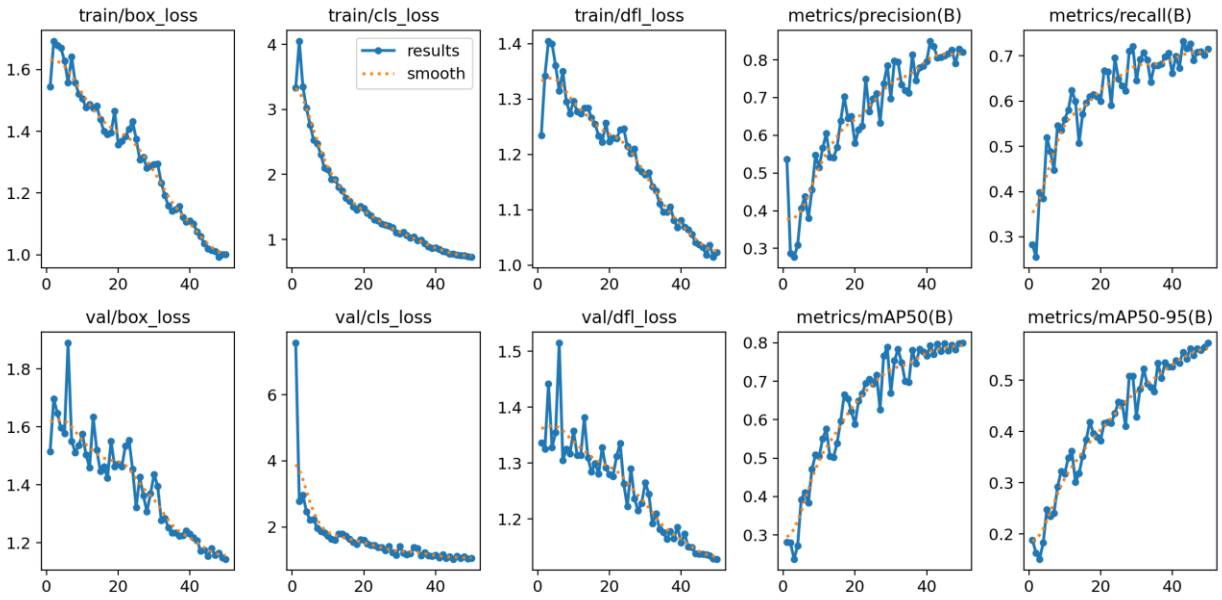

Рисунок 3.15 – Метрики YOLOv8n

Використаємо такі самі параметри навчання для інших моделей: YOLOv8s, YOLOv8m, YOLOv8l, YOLOv8x (рис. 3.16 – рис. 3.23).

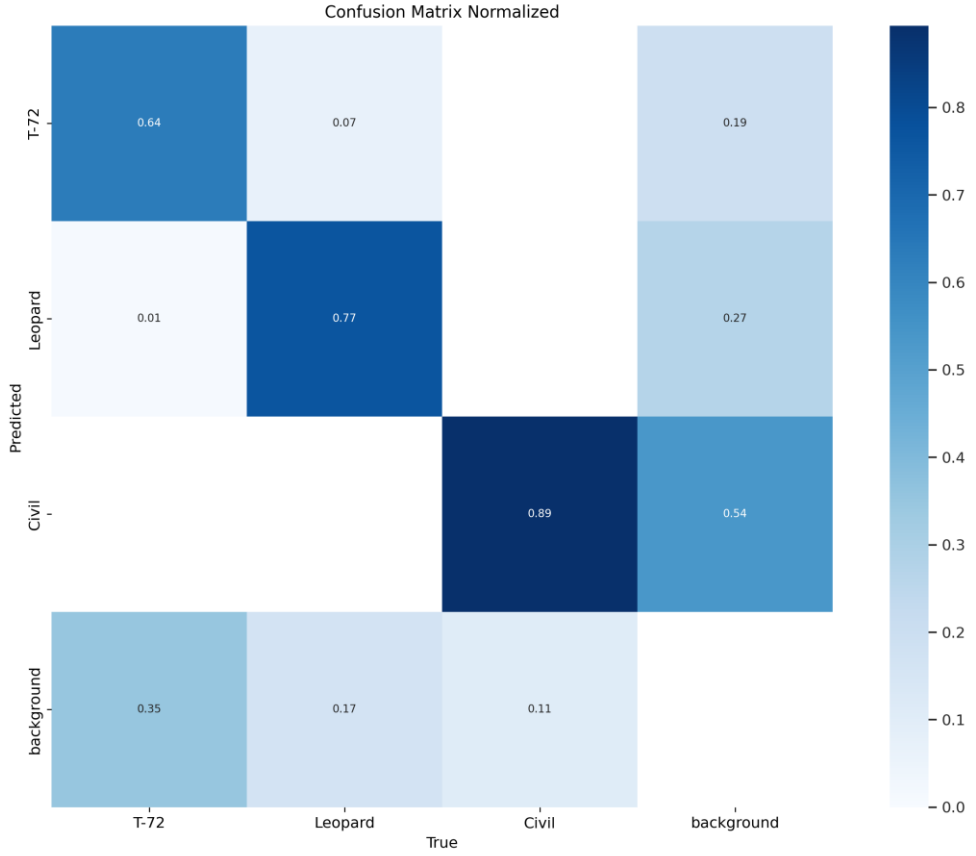

Рисунок 3.16 – Матриця плутанини YOLOv8s

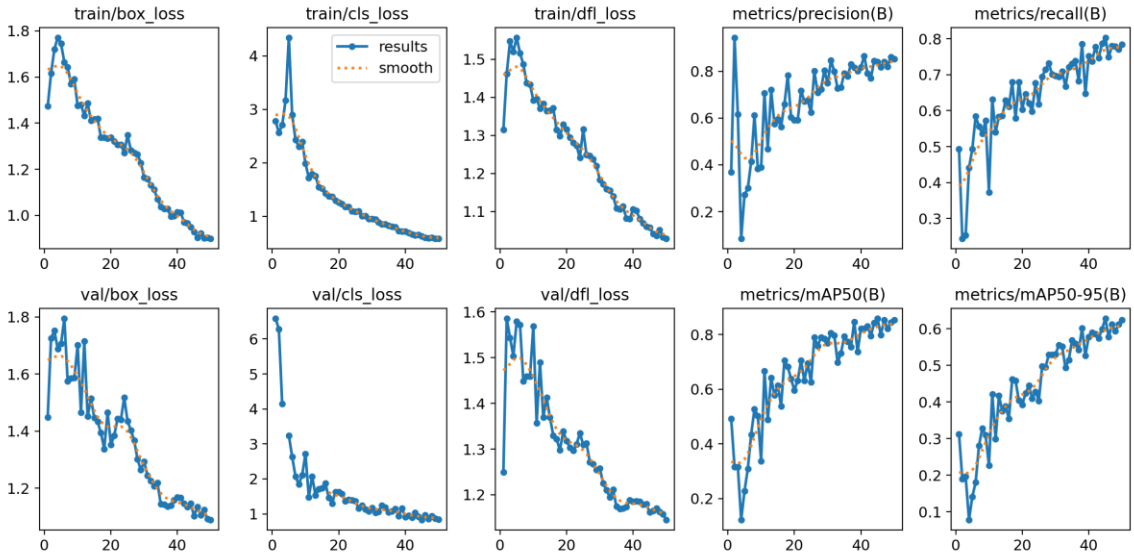

Рисунок 3.17 – Метрики YOLOv8s

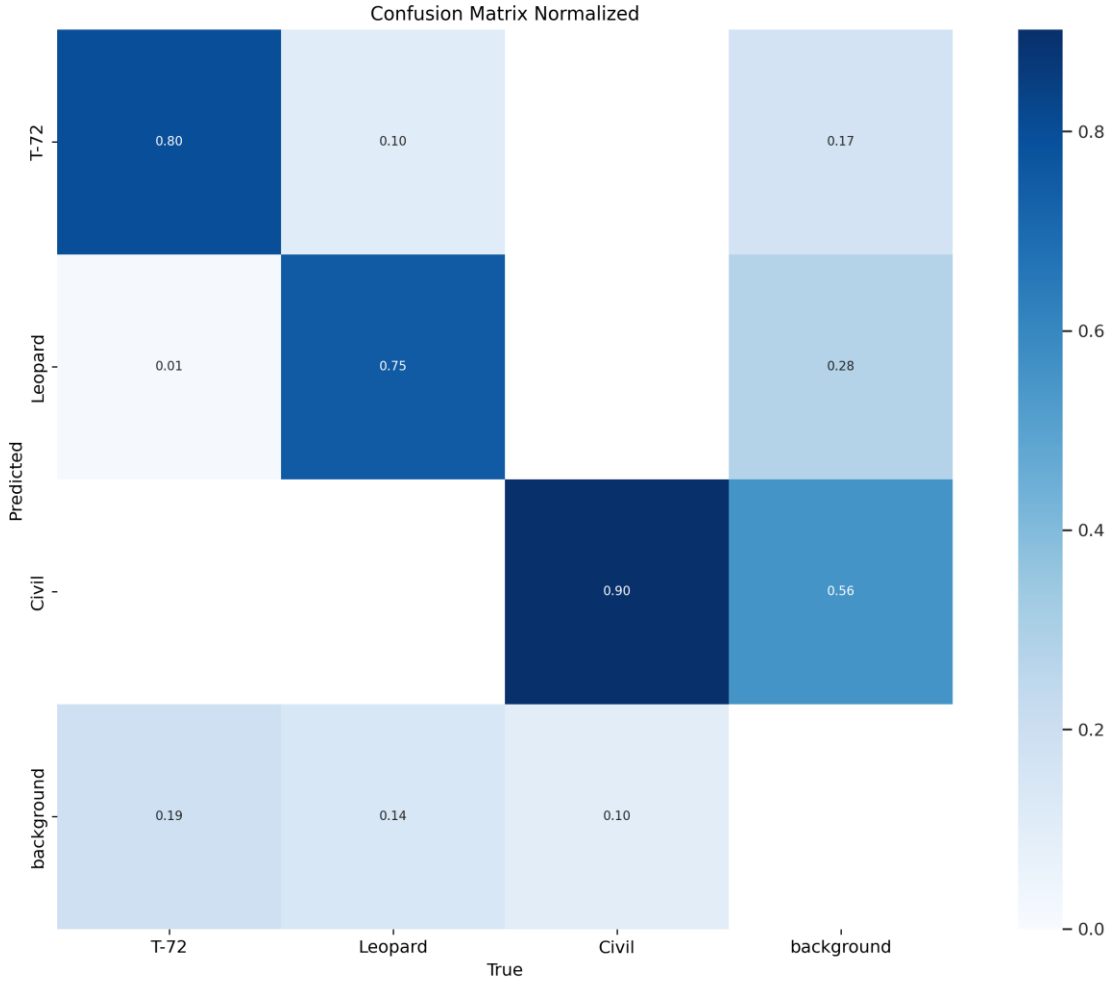

Рисунок 3.18 – Матриця плутанини YOLOv8m

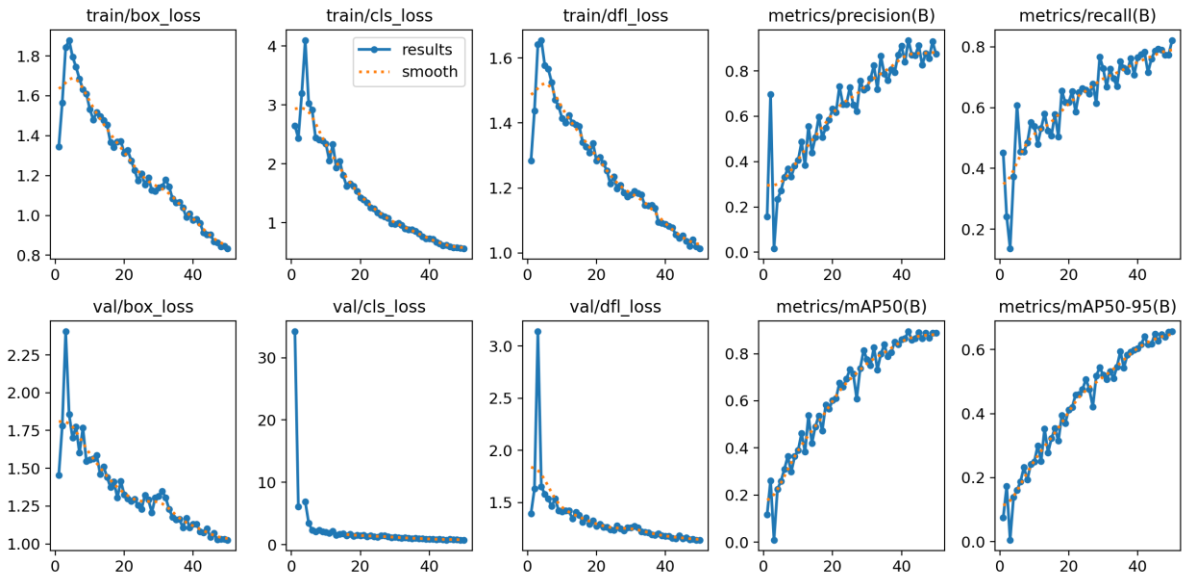

Рисунок 3.19 – Метрики YOLOv8m

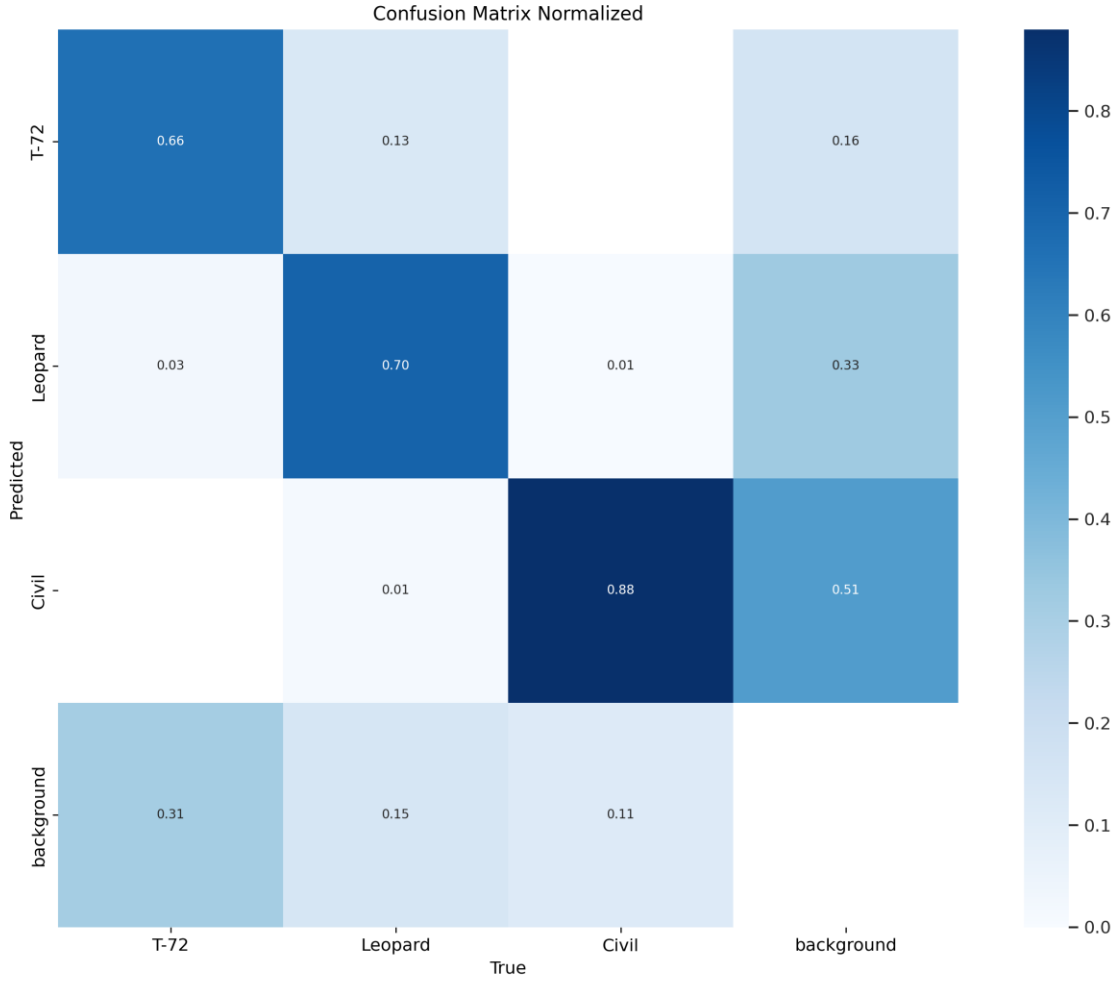

Рисунок 3.20 – Матриця плутанини YOLOv8l

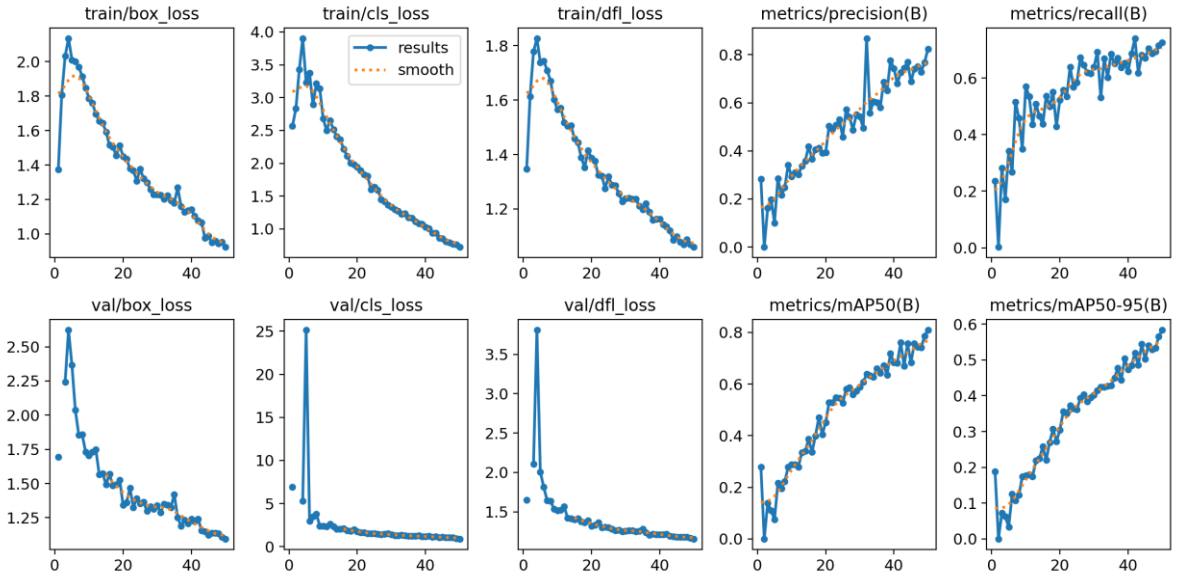

Рисунок 3.21 – Метрики YOLOv8l

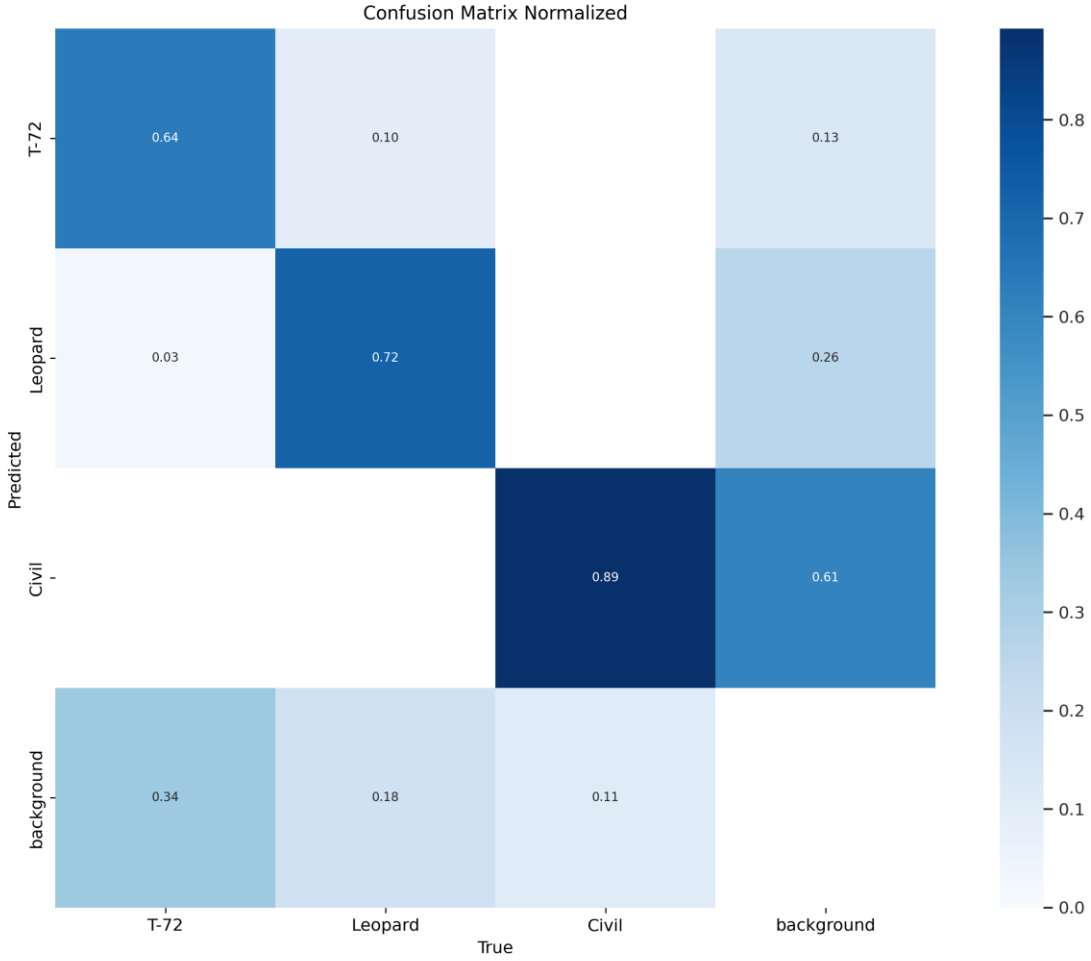

Рисунок 3.22 – Матриця плутанини YOLOv8x

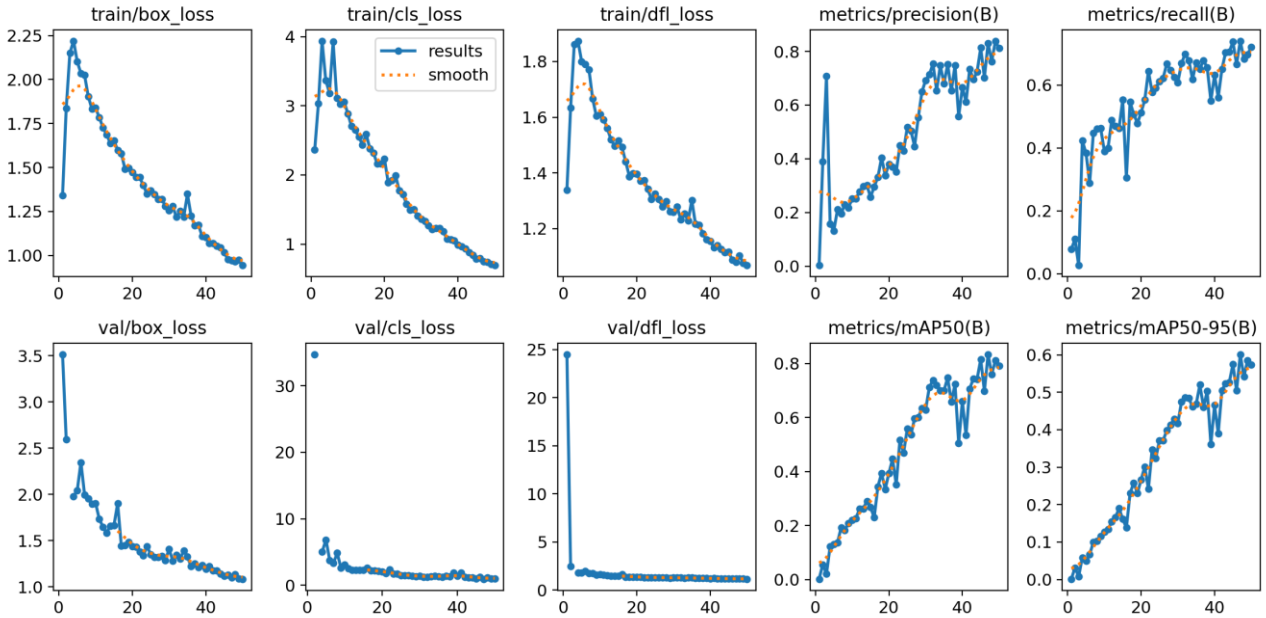

Рисунок 3.23 – Метрики YOLOv8x

3.7 Аналіз результатів експериментів

По матрицям плутанини можна побачити, що модель жодного разу не визначила цивільний транспорт як військовий, а тому в подальшому це допоможе вберегти життя.

В таблиці 3.1 представлено значення метрик точності для всіх моделей на останній 50-ій епосі навчання. Тестування швидкості обробки проводилося на відео з розширенням 1137×640 пікселів.

Таблиця 3.1 – Результати порівнянь

Модель	AP	mAP50	mAP50-95	Speed, ms	FPS
YOLOv8n	0,82071	0,7996	0,57286	21,8	45,9
YOLOv8s	0,85302	0,85328	0,62354	22,3	44,8
YOLOv8m	0,87437	0,88886	0,6566	26,5	37,7
YOLOv8l	0,82447	0,80957	0,58361	46,1	21,7
YOLOv8x	0,81121	0,79049	0,57388	67,2	14,9

В цілому значення вийшли доволі високі. Показники mAP50 більше 0,8 та mAP50-95 більше за 0,50, а подекуди й більше 0,60 вказують, що модель доволі точно визначає місцезнаходження об’єктів на зображенні. Беручи до уваги середню швидкість обробки кадру, деякі з моделей можуть цілком використовуватися на мобільному пристрої.

Враховуючи отримані результати, найкращим вибором для розпізнавання техніки на даному апаратному середовищі буде модель medium. Варто зазначити, що для моделей large та extra large можна отримати кращі значення точності, але враховуючи специфіку можливого використання не завжди знайдеться відеокарта обробляти відео в реальному часі, хоча б в 24 кадри на секунду на даних моделях. Тому представлені нижче результати розпізнавання виконувалися саме на моделі YOLOv8m (рис. 3.24 – рис. 3.28).

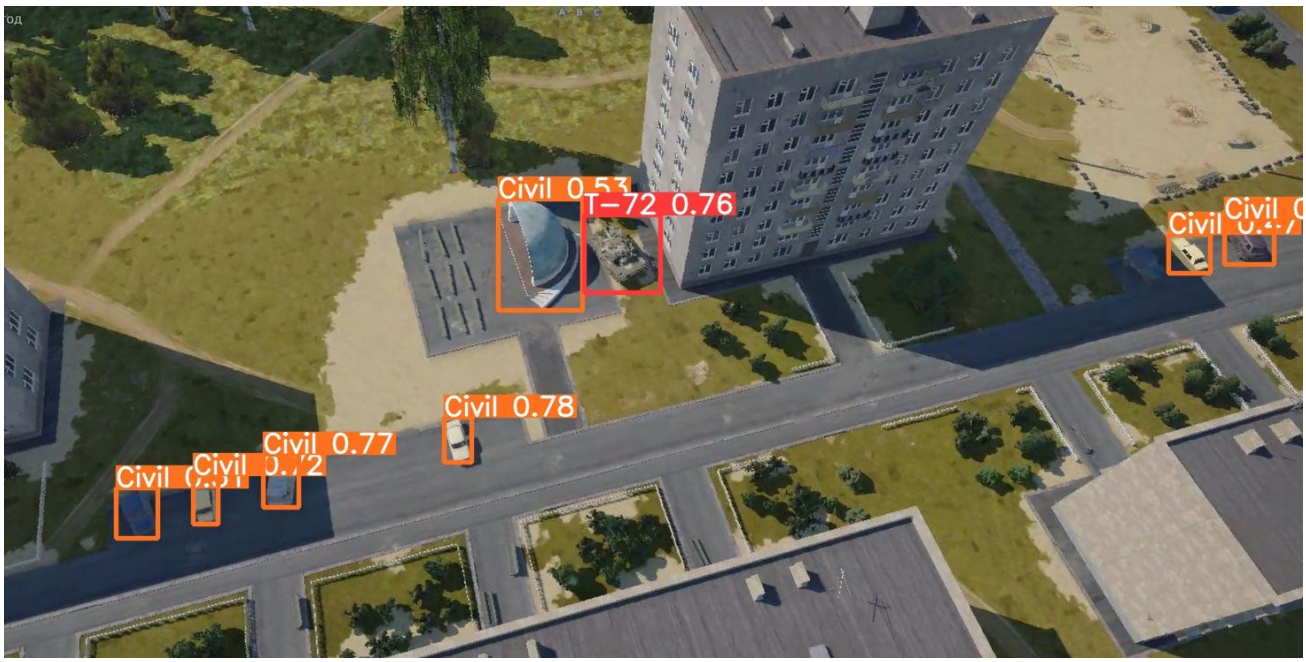

Рисунок 3.24 – Візуальні результати роботи моделі YOLOv8m на штучно згенерованих зображеннях у міському середовищу

Рисунок 3.25 – Візуальні результати роботи моделі YOLOv8m на штучно згенерованих зображеннях у степному середовищі

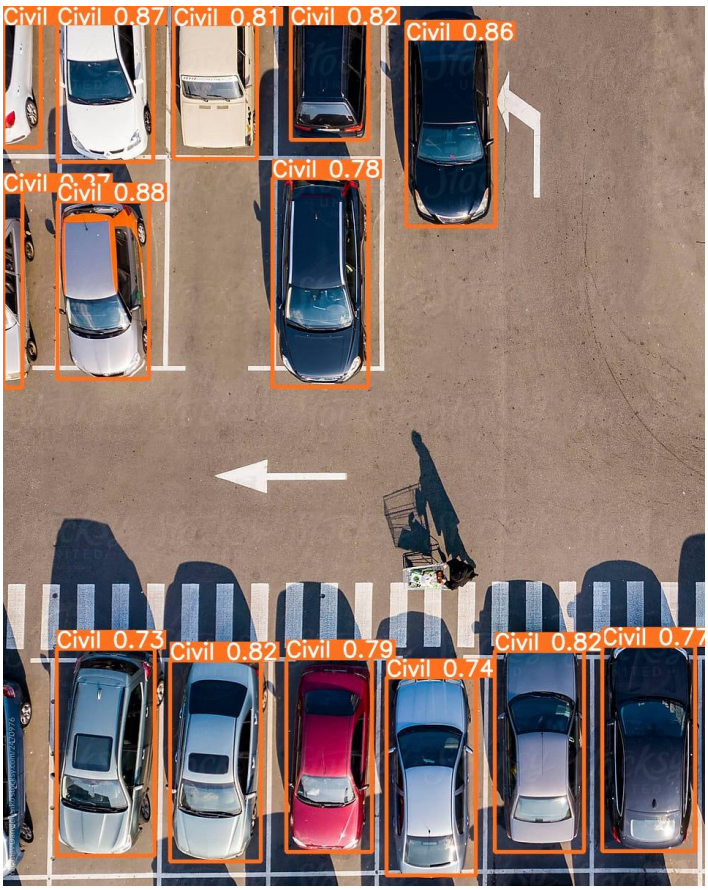

Рисунок 3.26 – Візуальні результати роботи моделі YOLOv8m на реальному зображенні авто на стоянці

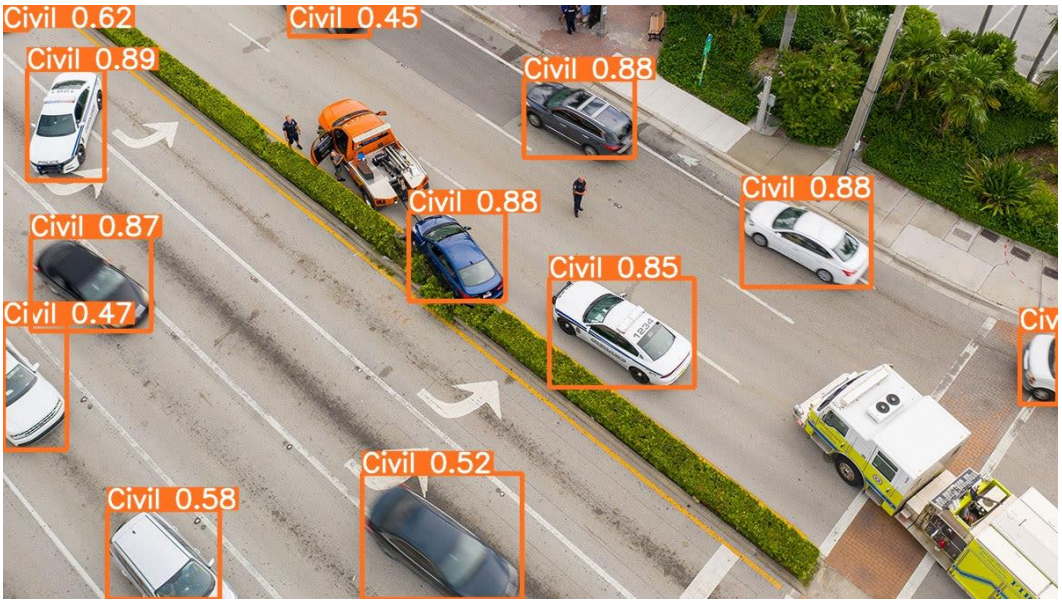

Рисунок 3.27 – Візуальні результати роботи моделі YOLOv8m на реальному зображенні авто у заторі

Рисунок 3.28 – Візуальні результати роботи моделі YOLOv8m на реальному зображенні танка у засідці

Судячи з тестів на зображеннях, які не приймали участі у навчанні, модель доволі точно визначає цивільну техніку як на штучно згенерованих зображеннях, так і на реальних. Щодо військової, то тут було отримано гарні результати на згенерованих та погані на реальних.

ВИСНОВКИ

У рамках кваліфікаційної роботи був створений навчальний датасет с військовою та цивільною технікою, а також навчено декілька моделей нейронної мережі YOLO.

У висновку цієї роботи слід визначити, що дослідження методів розпізнавання військового та цивільного транспорту у неперервному потоці даних має велике значення в контексті вдосконалення систем транспортного контролю та безпеки. Застосування сучасних технологій машинного навчання та комп’ютерного зору в цій області дозволяє розробити надійні та ефективні рішення, які відповідають сучасним викликам та вимогам.

Результати проведеного дослідження свідчать про те, що нейронну мережу можна навчити за допомогою штучно згенерованих зображень, але в тоді датасет має містити й реальні зображення. Лише тоді точність визначення буде прийнятною. Важливо зазначити, що такий підхід дозволить значно пришвидшити збір навчальних. Враховуючи, що за допомогою 3D моделювання можна зімітувати будь яку ситуацію, можна зробити акцент на певні ситуації, які складно, або взагалі не можливо сфотографувати. Тим самим модель буде більш підготовленою ніж та яка не навчалась на 3D моделях.

Загалом, проведене дослідження вносить вагомий внесок у розвиток сучасних технологій транспортного контролю та відкриває нові можливості для створення інтелектуальних систем управління та безпеки, що сприяють вдосконаленню та оптимізації транспортної інфраструктури.

Результати дослідження апробовано у вигляді тез доповідей під час Міжнародного молодіжного форуму «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ» [23].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Mashtalir, S. V., Stolbovyi, M. I., & Yakovlev, S. V. (2019). Clustering video sequences by the method of harmonic k-means. *Cybernetics and Systems Analysis, 55*, 200-206.

2. Bodyanskiy, Y., Shafronenko, A., & Mashtalir, S. (2020). Online robust fuzzy clustering of data with omissions using similarity measure of special type. In *Lecture Notes in Computational Intelligence and Decision Making: Proceedings of the XV International Scientific Conference "Intellectual Systems of Decision Making and Problems of Computational Intelligence"(ISDMCI'2019), Ukraine, May 21–25, 2019 15* (pp. 637-646). Springer International Publishing.

3. Ammar, A., Koubaa, A., Ahmed, M., Saad, A., & Benjdira, B. (2019). Aerial images processing for car detection using convolutional neural networks: Comparison between faster r-cnn and yolov3. arXiv preprint arXiv:1910.07234.

4. Ковалів, О. П. (2023). Дослідження нейронних мереж для розпізнавання військової техніки на супутникових знімках.

5. Pomazan V., Tvoroshenko I., and Gorokhovatskyi V. (2023) Handwritten character recognition models based on convolutional neural networks, International Journal of Academic Engineering Research, 7(9), pp. 64-72.

6. Іванов, С. І. (2022). *Розпізнавання стратегічних технічних об'єктів за допомогою згорткових нейронних мереж* (Master's thesis, КПІ ім. Ігоря Сікорського).

7. Sun, Y., Wang, W., Zhang, Q., Ni, H., & Zhang, X. (2022, July). Improved YOLOv5 with transformer for large scene military vehicle detection on SAR image. In *2022 7th International Conference on Image, Vision and Computing (ICIVC)* (pp. 87-93). IEEE.

8. Azizi, A., Yaghoobi, M., & Kamel, S. R. (2023). Intelligent detection and assessment of damaged buildings using UAV imagery and YOLOv8.

9. Wang, G., Chen, Y., An, P., Hong, H., Hu, J., & Huang, T. (2023). UAV-YOLOv8: A Small-Object-Detection Model Based on Improved YOLOv8 for

UAV Aerial Photography Scenarios. *Sensors*, *23*(16), 7190.

10. Hu, M., Li, Z., Yu, J., Wan, X., Tan, H., & Lin, Z. (2023). Efficient-Lightweight YOLO: Improving Small Object Detection in YOLO for Aerial Images. *Sensors*, *23*(14), 6423.

11. Xie, X., Cheng, G., Wang, J., Yao, X., & Han, J. (2021). Oriented R-CNN for object detection. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 3520-3529).

12. Song, P., Li, P., Dai, L., Wang, T., & Chen, Z. (2023). Boosting R-CNN: Reweighting R-CNN samples by RPN's error for underwater object detection. *Neurocomputing*, *530*, 150-164.

13. Chen, M., Yu, L., Zhi, C., Sun, R., Zhu, S., Gao, Z., ... & Zhang, Y. (2022). Improved faster R-CNN for fabric defect detection based on Gabor filter with Genetic Algorithm optimization. *Computers in Industry*, *134*, 103551.

14. Vecvanags, A., Aktas, K., Pavlovs, I., Avots, E., Filipovs, J., Brauns, A., ... & Anbarjafari, G. (2022). Ungulate detection and species classification from camera trap images using RetinaNet and faster R-CNN. *Entropy*, *24*(3), 353.

15. Harsono, I. W., Liawatimena, S., & Cenggoro, T. W. (2022). Lung nodule detection and classification from Thorax CT-scan using RetinaNet with transfer learning. *Journal of King Saud University-Computer and Information Sciences*, *34*(3), 567-577.

16. Roboflow: Give your software the power to see objects in images and video. URL: https://roboflow.com/ (дата звернення 08.10.2023).

17. Xiao, Y., Tian, Z., Yu, J., Zhang, Y., Liu, S., Du, S., & Lan, X. (2020). A review of object detection based on deep learning. *Multimedia Tools and Applications*, *79*, 23729-23791.

18. Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 580-587).

19. Shugurov, I., Li, F., Busam, B., & Ilic, S. (2022). Osop: A multi-stage

one shot object pose estimation framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 6835-6844).

20. Han, G., Ma, J., Huang, S., Chen, L., & Chang, S. F. (2022). Few-shot object detection with fully cross-transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 5321-5330).

21. Medium.com: A guide to the object detection exercise using YOLO model. URL: https://medium.com/analytics-vidhya/a-guide-to-the-object-detection-exercise-using-yolo-model-c551f65df637 (дата звернення 20.10.2023).

22. Intersection over Union (IoU) for object detection. URL: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/ (дата звернення 21.10. 2023).

23. Скударнов, М. Д. АЛГОРИТМ YOLO ДЛЯ РОЗПІЗНАВАННЯ ВІЙСЬКОВОГО ТА ЦИВІЛЬНОГО ТРАНСПОРТУ У НЕПЕРИВНОМУ ПОТОЦІ ДАНИХ. *27-й Міжнародний молодіжний форум «Радіоелектроніка і молодь у XXI столітті». Зб. матеріалів форуму. Т. 7. Харків: ХНУРЕ. 2023. 243 с.*, 10.

24. Diwan, T., Anirudh, G., & Tembhurne, J. V. (2023). Object detection using YOLO: Challenges, architectural successors, datasets and applications. *multimedia Tools and Applications*, *82*(6), 9243-9275.

25. Zhang, S., & Ge, J. (2023, October). Vehicle detection algorithm for highway driving scenarios. In *Fifth International Conference on Artificial Intelligence and Computer Science (AICS 2023)* (Vol. 12803, pp. 711-715). SPIE.

26. Wang, Y., Zou, H., Yin, M., & Zhang, X. (2023). SMFF-YOLO: A Scale-Adaptive YOLO Algorithm with Multi-Level Feature Fusion for Object Detection in UAV Scenes. *Remote Sensing*, *15*(18), 4580.

27. Al Muksit, A., Hasan, F., Emon, M. F. H. B., Haque, M. R., Anwary, A. R., & Shatabda, S. (2022). YOLO-Fish: A robust fish detection model to detect fish in realistic underwater environment. *Ecological Informatics*, *72*, 101847.

28. Ni, N., & Dong, S. (2023). Numerical computation of partial differential equations by hidden-layer concatenated extreme learning

machine. *Journal of Scientific Computing*, *95*(2), 35.

29. Li, X., Wang, W., Wu, L., Chen, S., Hu, X., Li, J., ... & Yang, J. (2020). Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection. *Advances in Neural Information Processing Systems*, *33*, 21002-21012.

30. Chandra, K., Xie, A., Ragan-Kelley, J., & Meijer, E. (2022). Gradient descent: The ultimate optimizer. *Advances in Neural Information Processing Systems*, *35*, 8214-8225.

31. Dubey, S. R., Singh, S. K., & Chaudhuri, B. B. (2023). AdaNorm: Adaptive Gradient Norm Correction based Optimizer for CNNs. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision* (pp. 5284-5293).

32. Bisla, D., Wang, J., & Choromanska, A. (2022, May). Low-pass filtering sgd for recovering flat optima in the deep learning optimization landscape. In *International Conference on Artificial Intelligence and Statistics* (pp. 8299-8339). PMLR.

33. Elshamy, R., Abu-Elnasr, O., Elhoseny, M., & Elmougy, S. (2023). Improving the efficiency of RMSProp optimizer by utilizing Nestrove in deep learning. *Scientific Reports*, *13*(1), 8814.

34. Chakrabarti, K., & Chopra, N. (2022). A control theoretic framework for adaptive gradient optimizers in machine learning. *arXiv preprint arXiv:2206.02034*.

35. Sharma, J., Soni, S., Paliwal, P., Saboor, S., Chaurasiya, P. K., Sharifpur, M., ... & Afzal, A. (2022). A novel long term solar photovoltaic power forecasting approach using LSTM with Nadam optimizer: A case study of India. *Energy Science & Engineering*, *10*(8), 2909-2929.

36. Pratama, D. A., Abo-Alsabeh, R. R., Bakar, M. A., Salhi, A., & Ibrahim, N. F. (2023). Solving partial differential equations with hybridized physic-informed neural network and optimization approach: Incorporating genetic algorithms and L-BFGS for improved accuracy. *Alexandria Engineering*

Journal, *77*, 205-226.

37. Raschka, S., Liu, Y. H., Mirjalili, V., & Dzhulgakov, D. (2022). *Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python*. Packt Publishing Ltd.

38. Best Python Libraries for Machine Learning. URL: https://www.coursera.org/articles/python-machine-learning-library (дата звернення 09.11. 2023).

39. Ultralytics. URL: https://www.ultralytics.com/ (дата звернення 09.11. 2023).

40. Ultralytics models, github repository. URL: https://github.com/ultralytics/ultralytics (дата звернення 09.11. 2023).

41. ClearML. URL: https://clear.ml/ (дата звернення 09.11. 2023).