МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ЗАСТОСУНКУ ДЕТЕКЦІЇ ТА КЛАСИФІКАЦІЇ ОБ'ЄКТІВ»**

Виконала: студентка 4 курсу, групи 6.1211-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
І.А. Катана
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Мухін В.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Катані Ірині Артемівні
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка застосунку детекції та класифікації об’єктів

керівник роботи Мухін Віталій Вікторович, к.т.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз сучасних методів розпізнавання об’єктів.
2. Розробка та навчання моделі YOLO11.
3. Розробка застосунку для використання розробленої моделі.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 26.12.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	13.01.2025	
2.	Збір вихідних даних.	03.02.2025	
3.	Обробка методичних та теоретичних	21.02.2025	
	джерел.		
4.	Розробка першого та другого розділу.	26.03.2025	
5.	Розробка третього розділу.	19.05.2025	
6.	Оформлення та нормоконтроль	26.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	16.06.2025	

Студент _____ І.А. Катана
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.В. Мухін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка застосунку детекції та класифікації об’єктів»: 75 с., 26 рис., 1 табл., 20 джерел, 2 додатка.

ВІЙСЬКОВА ТЕХНІКА, ДЕТЕКЦІЯ ОБ’ЄКТІВ, КЛАСИФІКАЦІЯ ОБ’ЄКТІВ, КОМП’ЮТЕРНИЙ ЗІР, МАШИННЕ НАВЧАННЯ, STREAMLIT, YOLO.

Об’єкт дослідження – процес тренування моделі машинного навчання для розпізнавання військової техніки та інтеграції її у вебзастосунок.

Мета роботи: розробка вебзастосунку для використання моделі машинного навчання для розпізнавання військової техніки.

Методи дослідження – методи аналізу, емпіричні методи, методи програмної інженерії.

У кваліфікаційній роботі було проведено аналіз сучасних методів розпізнавання об’єктів. Розглянуто різні архітектури моделей та обґрунтовано вибір YOLOv11, як оптимальної. Було сформовано набір даних, що включає чотири класи військової техніки: бронетехніка (AFV), бронетранспортери (APC), артилерійські системи та системи протиповітряної оборони. Після чого, модель YOLOv11 було натреновано на створеному наборі даних. Також, було розроблено вебзастосунок на основі фреймворку Streamlit, що надає можливість розпізнавання військової техніки на зображеннях, відеофайлах, YouTube-відео та RTSP потоках.

Результати роботи можуть бути використані для автоматизації процесів моніторингу та розпізнавання військової техніки в різних умовах спостереження. Подальші дослідження можуть включати розширення набору даних більш детальними класами військової техніки та підвищення точності моделі.

SUMMARY

Bachelor’s qualifying paper “Development of an Application for the Object Detection and Classification”: 75 pages, 26 figures, 1 table, 20 references, 2 supplements.

COMPUTER VISION, MACHINE LEARNING, MILITARY EQUIPMENT, OBJECT CLASSIFICATION, OBJECT DETECTION, STREAMLIT, YOLO.

The object of the study is the process of development a machine learning model for the detection and classification of military equipment and its integration into a web application.

The aim of the study is development of a web application utilizing a machine learning model for the detection and classification of military equipment.

The methods of research are methods of analysis, empirical methods, and the development of software.

In this bachelor’s work, an analysis of modern object detection methods was conducted. Different model architectures were examined, and YOLOv11 was justified as the optimal choice. A dataset was created that includes four classes of military equipment: armored fighting vehicles (AFV), armored personnel carriers (APC), artillery systems, and air defense systems. Subsequently, the YOLOv11 model was trained on the created dataset. Additionally, a web application was developed, using Streamlit framework, which provides the capability to recognize military equipment on images, video files, YouTube videos, and RTSP streams.

The results of this work can be used to automate the monitoring and identification of military equipment under various observation conditions. Further research may include expanding the dataset with more detailed classes of military equipment and improving the model’s accuracy.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Аналіз сучасних методів детекції та класифікації об’єктів 10
1.1 Аналіз предметної області 10
1.2 Огляд еволюції підходів до розпізнавання об’єктів..... 12
1.2.1 Історичний розвиток підходів 12
1.2.2 Виникнення та концепція R-CNN 14
1.2.3 Поява одноетапних моделей..... 15
1.2.4 Сучасні підходи на основі трансформерів 15
1.2.5 Зміна фокусу: від точності до балансу ефективності..... 16
1.3 Переваги та недоліки сучасних моделей розпізнавання об’єктів..... 17
1.3.1 YOLO..... 19
1.3.2 EfficientDet..... 20
1.3.3 RetinaNet 21
1.3.4 Faster R-CNN 22
1.3.5 Deformable DETR..... 23
1.3.6 Вибір оптимальної моделі..... 24
1.4 Архітектура YOLOv11..... 25
1.4.1 Базова мережа (backbone)..... 27
1.4.2 Шия моделі (neck)..... 28
1.4.3 Головна частина (head)..... 28
1.4.4 Додаткові особливості..... 29
2 Розробка та навчання моделі розпізнавння об’єктів 30
2.1 Підготовка тренувального набору даних 30
2.1.1 Вибір класів об’єктів та джерел даних 30
2.1.2 Процес формування набору даних..... 31

2.1.3 Аналіз та експорт набору даних 32
2.2 Навчання моделі YOLOv11 та оцінка точності 33
3 Розробка застосунку для використання моделлі розпізнавання об’єктів 40
3.1 Огляд Streamlit як інструменту для створення вебзастосунків 40
3.2 Реалізація функціоналу додатку 41
3.2.1 Обробка зображень 42
3.2.2 Обробка відеофайлів 43
3.2.3 Обробка відео з YouTube 44
3.2.4 Обробка RTSP 46
Висновки 48
Перелік посилань 50
Додаток А Функції формування набору даних 53
А.1 Process_roboflow_dataset та пов’язані функції 53
А.2 Analyze_dataset_classes 57
А.3 Split_and_export_to_yolo 58
Додаток Б Додаток Streamlit 64
Б.1 Код файлу app.py 64
Б.2 Код файлу worker.py 65

ВСТУП

Російська збройна агресія проти України, що переросла у повномасштабне вторгнення 24 лютого 2022 року, створила нові виклики для національної безпеки та оборони. Використання сучасних технологій машинного навчання та комп'ютерного зору дозволяє автоматизувати процеси розвідки та спостереження. Це, в свою чергу, може значно підвищити ефективність обробки великих обсягів візуальних даних з різних джерел – від безпілотних літальних апаратів до камер спостереження.

Таким чином, метою моєї роботи є розробка вебзастосунку для використання моделі машинного навчання для розпізнавання військової техніки.

Задачі, які необхідно розв'язати для досягнення поставленої мети:

– провести аналіз існуючих методів розпізнавання об'єктів;
– дослідити та порівняти ефективність різних архітектур нейронних мереж для розпізнавання об'єктів;
– сформувати набір даних для навчання моделі, що міститиме зображення військової техніки різних типів;
– навчити модель YOLOv11 на підготовленому наборі даних;
– розробити вебзастосунок для взаємодії з моделлю;
– реалізувати функціонал для обробки різних типів вхідних даних;
– провести тестування та оцінку ефективності розробленого рішення.

Об'єкт дослідження – процес тренування моделі машинного навчання для розпізнавання військової техніки та інтеграції її у вебзастосунок.

Предмет дослідження – методи та алгоритми машинного навчання для розпізнавання військової техніки.

Методи дослідження – метод аналізу, емпіричні методи, методи програмної інженерії.

Структурно робота складається з 3 розділів. У першому розділі проведено аналіз сучасних методів розпізнавання об'єктів, досліджено

еволюцію підходів та обґрунтовано вибір архітектури YOLOv11, як оптимальної для поставленої задачі. Другий розділ містить опис процесу формування та підготовки набору даних, навчання моделі YOLOv11 та оцінку її ефективності. Третій розділ присвячено огляду розробленого вебзастосунку для зручної взаємодії з моделлю.

1 АНАЛІЗ СУЧАСНИХ МЕТОДІВ ДЕТЕКЦІЇ ТА КЛАСИФІКАЦІЇ ОБ'ЄКТІВ

1.1 Аналіз предметної області

Комп'ютерний зір – це галузь штучного інтелекту, яка займається аналізом візуальних даних за допомогою різноманітних алгоритмів машинного навчання. Основна мета цієї технології – навчити комп'ютери сприймати та розуміти зображення або відео подібно до людини.

Завдяки розвитку глибокого навчання та, зокрема, згорткових нейронних мереж (Convolutional Neural Networks, CNN), сучасні системи комп'ютерного зору досягли доволі високого рівня точності. Архітектура CNN була натхнена зоровою системою живих істот, і такий підхід виявився ефективним для застосування на просторових даних, таких як зображення [1]. «CNN здатні вивчати абстраговані ознаки об'єктів на різних рівнях ієрархії, що робить їх ефективнішими в порівнянні з класичними нейронними мережами в завданнях комп'ютерного зору» [1].

Типова архітектура CNN (див. рис. 1.1) складається зі: згорткових шарів, що виділяють характерні ознаки зображення; операцій активації (після кожного згорткового шару); шарів підвибірки, що зменшують розмірність даних; повноз'єднаних шарів, які відповідають за остаточну класифікацію.

У контексті даної роботи комп'ютерний зір використовується для розпізнавання військової техніки на зображеннях та відео.

Можна виділити три основних підходи в завданнях комп'ютерного зору: класифікація, детекція та сегментація об'єктів. Класифікація визначається як «завдання категоризації зображень у одну з кількох наперед визначених категорій» [2]. Такий підхід є базовим і дозволяє ефективно аналізувати великі обсяги даних, але не дає можливості виявити місцезнаходження об'єктів на сцені.

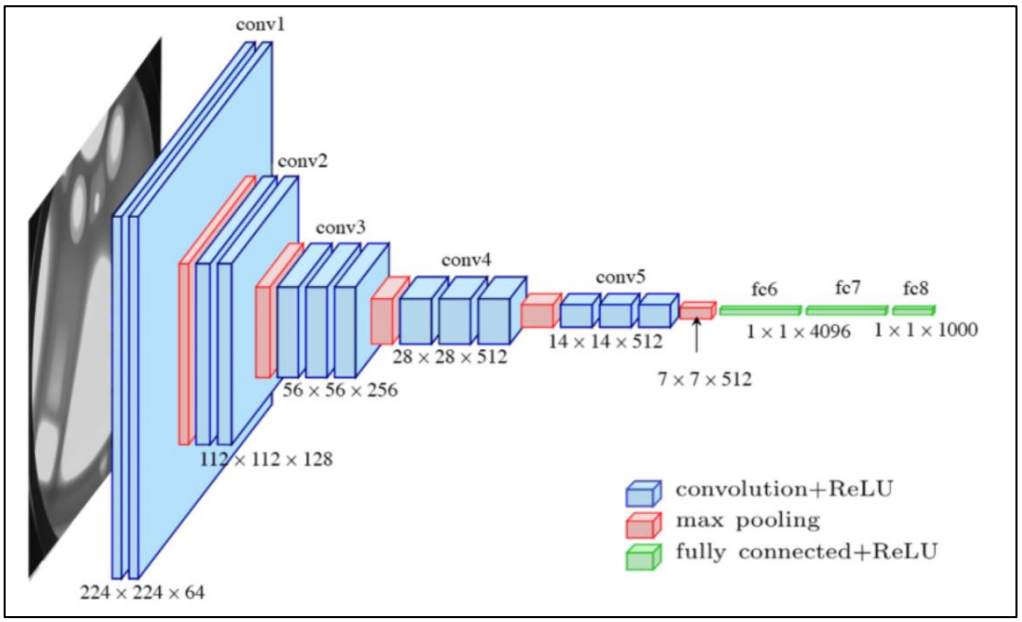


Рисунок 1.1 – Типова архітектура CNN [14]

Задача детекції в свою чергу «полягає у виявленні об’єктів певного класу (наприклад люди, тварини чи автомобілі) на цифрових зображеннях, з метою відповісти на ключове запитання: які об’єкти і де саме вони розташовані» [3].

Сегментація є процесом, «що дозволяє локалізувати об’єкти та визначити їхні межі на зображенні, а також присвоїти кожному пікселю відповідну мітку, що відображає спільні характеристики» [4]. Сегментація дозволяє досягти найвищої точності в аналізі візуальних даних, однак вимагає значних обчислювальних ресурсів. В реальних умовах, детекція об’єктів найчастіше є оптимальним вибором, бо отримується компроміс між продуктивністю та точністю, тож цей підхід було обрано для цієї роботи (див. рис. 1.2).

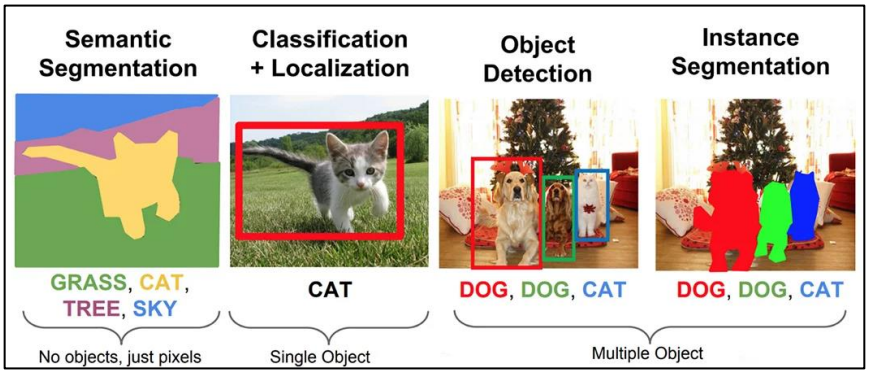


Рисунок 1.2 – Візуалізація типів завдань комп’ютерного зору [15]

Ефективність моделі машинного навчання значною мірою залежить від якості набору даних, використаного для її тренування. У військовій сфері доступність таких даних є проблемою, а існуючі дані часто мають низьку якість [5].

Крім того, труднощі з використанням моделі в реальних умовах пов’язані з якістю вхідних даних. Наприклад відео з дронів або знімки супутників часто містять шум або розмитість. Це може знизити ефективність моделі, особливо, якщо її було навчено на синтетичних даних. Для вирішення цієї проблеми застосовують методи попередньої обробки зображень, наприклад покращення контрастності та нормалізацію.

Крім якості даних, велику роль грає продуктивність моделі та її спроможність приймати рішення в режимі реального часу. Тому існує вимога до високої швидкості обробки зображень моделлю.

Використання заскладних архітектур, таких як Detection Transformers (DETR), може забезпечити високу точність, але вони потребують немалих ресурсів.

Альтернативою є такі моделі, як YOLO, SSD та Faster R-CNN. Вони досить точні і в той же час не вимагають таких великих обчислювальних ресурсів. Крім того, для підвищення продуктивності застосовуються методи подальшої оптимізації нейронних мереж, включно з квантуванням або спеціальні бібліотеки для прискорення обчислень, такі як TensorRT.

1.2 Огляд еволюції підходів до розпізнавання об’єктів

1.2.1 Історичний розвиток підходів

На початку історії виникнення комп’ютерного зору, розпізнавання об’єктів значною мірою залежало від алгоритмів з вручну спроєктованими ознаками. Наприклад, метод Віоли-Джонса використовував ковзаючі вінка,

каскади простих ознак Хаара та адаптивне підсилення (AdaBoost) для виявлення об’єктів на зображеннях. Цей алгоритм першим дозволив здійснювати розпізнавання обличчя в реальному часі [3].

Наступним кроком було використання HOG (Histogram of Oriented Gradients) разом з лінійним класифікатором SVM (Support Vector Machines). Перший з них давав можливість виділяти характеристики різних об’єктів через аналіз градієнтів інтенсивності пікселів зображення. А другий, в свою чергу, визначав границю рішень, представлену лінією (у 2D), площиною (у 3D) та гіперплощиною у вищих вимірах. Цей підхід вважається основою для багатьох наступних методів детекції об’єктів.

Метод DPM (Deformable Part-based Model) запропонував новий підхід – розгляд об’єктів як складну композицію простіших частин. Наприклад людина є комбінацією голови, рук, ніг, тулуба та інших частин (див. рис. 1.3).

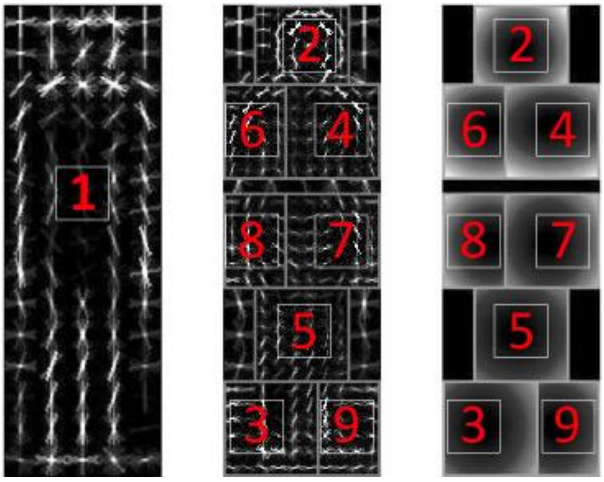


Рисунок 1.3 – Приклад принципу «поділяй і володарюй» [16]

Цей підхід підвищував стійкість до різних положень частин об’єкта або його форм. Проте точність детекторів на основі ручих ознак досягла певної межі, бо її підвищення вимагало б занадто складних обчислень та моделей. На заміну ним прийшли згорткові нейронні мереж.

Глибокі нейронні мережі мають можливість самостійно навчатись визначати потрібні характеристики напряму з даних. А якість виявлення об’єктів, особливо у складних сценах з багатьма об’єктами значно виросла.

1.2.2 Виникнення та концепція R-CNN

Концепція підходу R-CNN (Regions with CNN features) полягає у виділені регіон-кандидатів, тобто потенційних областей об’єктів, та аналізі кожного такого регіону глибокою згортковою мережею [6]. Точність таких моделей була вища за рахунок добре навчених нейромереж, з іншого боку вони були повільні, бо вимагали запуску CNN окремо для кожного із сотень регіонів зображення.

Цей недолік було усунуто в покращеній версії Fast R-CNN, швидкість якої зросла в понад 200 разів при збережені точності. Відмінність принципів було в тому, що зображення пропускалось крізь CNN один раз, формуючи карту ознак; після чого для кожного регіону виконувалось ROI Pooling для витягування ознак відповідної області карти. Проте, дана модель все ще залежала від зовнішнього алгоритму генерування регіонів, який обмежував загальну продуктивність.

Через це наступним кроком стало об’єднання всіх процесів в одну модель – Faster R-CNN. У цій архітектурі безпосередньо в згорткову мережу було включено окремий модуль RPN (Region Proposal Network), який генерував кандидати-області (потенційне розташування об’єктів) з виходів мережі (див. рис. 1.4).

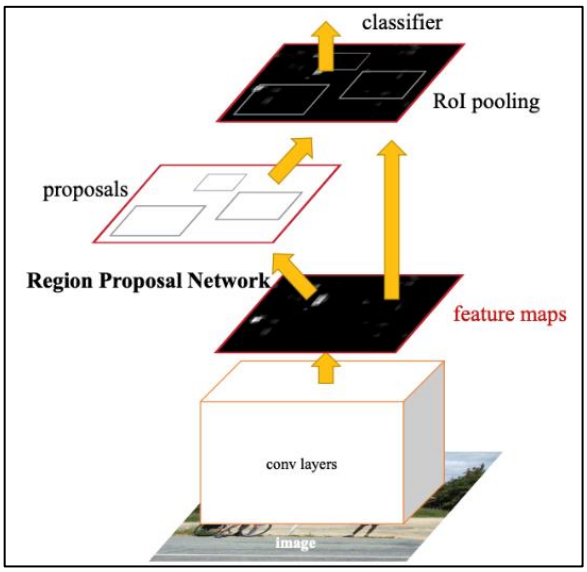


Рисунок 1.4 – Принцип роботи Faster R-CNN [17]

Ця архітектура стала першим детектором з використанням глибокого навчання який був близький до роботи в реальному часі [3].

1.2.3 Поява одноетапних моделей

Вище були розглянуті двоетапні детектори, які виконують детекцію послідовно окремими кроками (генерування регіонів та класфікація). Одноетапні детектори, в свою чергу, намагаються виконати все за один прохід мережі, а саме прогнозування координат та класифікацію об’єктів. Раніше вважалось, що скорочення кроків до одного неможливо без зниження точності, проте архітектури нового покоління спростували цю думку.

YOLO стала першою архітектурою, яка відмовилась від окремого етапу визначення регіон-пропозицій. Цей детектор розбиває зображення на сітку областей та одночасно передбачає межі обрамлення та класи для кожної з них. Такий підхід значно збільшив швидкість роботи моделі, що стало значним прогресом. Проте на початковому етапі точність локалізації, особливо для дрібних об’єктів, була нижчою в порівнянні з двоетапними детекторами.

Приблизно одночасно з YOLO було представлено SSD (Single Shot MultiBox Detector). Ця архітектура використовувала кілька рівнів ознак роздільної здатності для детекції об’єктів різного масштабу в межах однієї CNN, що дозволило значно підвищити якість виявлення дрібних об’єктів.

RetinaNet в свою чергу запропонувала певну інновацію, яка полягає в її функції втрати focal loss, яка вирішує проблему дисбалансу класів. Це дозволило досягти точності, співставної з двоетапними моделями.

1.2.4 Сучасні підходи на основі трансформерів

Останніми роками для задачі детекції почали використовувати трансформери, що було пов’язано з їхнім успіхом у сфері розпізнавання

мови [7]. Трансформери – це моделі глибокого машинного навчання, які засновані на механізмі self-attention і здатні враховувати взаємодії між елементами зображення. Дослідники з Facebook AI запропонували перший підхід до детекції об’єкта на основі чистого трансформера – DETR (DEtection TRansformer). Ця модель використовує абсолютно інший підхід – «прямого прогнозування для обходу сурогатних завдань» [8].

З точки зору архітектури DETR поєднує CNN для визначення ознак та енкодер-декодер трансформер, який обробляє ці ознаки вищезгаданим механізмом self-attention (див. рис. 1.5). На виході модель одночасно прогнозує фіксовану кількість об’єктів, координати меж та їхні класи. Для навчання використовувався метод угорського зіставлення (Hungarian matching), це дало гарантію, що кожен об’єкт буде виявлений рівно один раз. Це звільняє від необхідності використовувати процедуру NMS (Non-Maximum Suppression) та інші правила фільтрації вихідних даних.

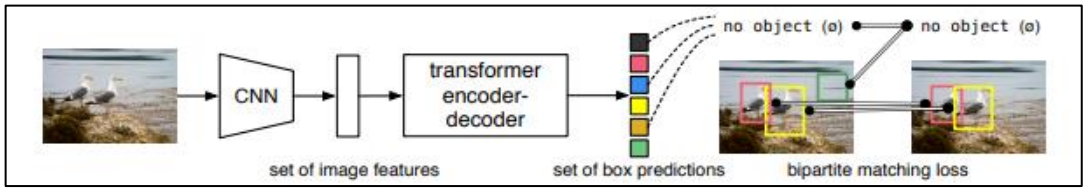


Рисунок 1.5 – Архітектура DETR [18]

Існувало декілька технічних недоліків: початкову версію потрібно було тренувати дуже довго і для малих об’єктів точність була невеликою. Проте, ці недоліки було зменшено в наступних версіях моделі, таких як Deformable DETR.

1.2.5 Зміна фокусу: від точності до балансу ефективності

На початку розвитку детекторів, основною метою було підвищення точності виявлення об’єктів без урахування обсягу обчислювальних витрат. У дослідницьких змаганнях традиційно оцінювались тільки метрики точності,

наприклад mAP, тож лідери часто використовували великі моделі, застосування яких не є доцільним в реальних умовах.

На сьогоднішній день вимоги змінились – одночасно з точністю висуваються обмеження на швидкість роботи моделі та кількість ресурсів, яку вона споживає. Вибір оптимальної моделі залежить від платформи: для мобільного застосування краще трохи менш точна, але значно швидша модель, а для серверних середовищ можна дозволити збільшити складність моделі задля отримання максимальної точності. Тому сучасні дослідження зосереджені на пошуку компромісу «точність проти швидкості».

Особливо важливим є питання продуктивності для пристроїв з обмеженими ресурсами – наприклад, на безпілотних літальних апаратах. У таких ситуаціях пріоритетом буде збереження балансу достатньої точності за умови забезпечення високої швидкості та низького енергоспоживання. Для вирішення цієї проблеми з’являються спеціалізовані полегшені моделі. Наприклад, для аналізу зображень з дронів (UAV) було запропоновано спрощену архітектуру детектора, заснованого на YOLOv5s [9]. Також моделі типу EfficientDet були розроблені саме з метою одночасної оптимізації показників точності та ефективності через масштабування глибини, ширини та роздільної здатності моделі [10].

1.3 Переваги та недоліки сучасних моделей розпізнавання об’єктів

Сучасні підходи та моделі розпізнавання об’єктів відрізняються за архітектурою, продуктивністю та вимогами до ресурсів. У таблиці 1.1 наведено порівняння ключових параметрів кожної архітектури:

- кількість параметрів (у мільйонах): кількість вагових коефіцієнтів у моделі, що впливає на здатність до вивчення складних залежностей та вимоги до пам’яті;
- розмір моделі в MB: обсяг файлу з вагами моделі, що залежить від кількості параметрів і типу збереження (float32, float16 тощо);

- розмір вхідного зображення (в пікселях);
- обчислювальна складність (FLOPs): кількість операцій з плаваючою комою (Floating Point Operations), необхідна для обробки одного зображення, що впливає на продуктивність і споживання ресурсів під час інференсу;
- середня точність на COCO наборі даних (mAP, Mean Avarage Precision): метрика якості, що оцінює правильність класифікації та точність локалізації;
- швидкість інференсу (FPS): кількість окремих зображень (або кадрів відео), які модель обробляє за секунду на певному апаратному забезпеченні, що залежить від FLOPs, оптимізації моделі та розміру вхідних даних;
- приблизний час тренування до збіжності: кількість годин які займе тренування моделі на наборі даних, необхідних для досягнення стабільної якості та продуктивності (час вказано з урахуванням повного проходу COCO train2017 (118000 зображень) при batch size ~ 16-32).

Таблиця 1.1 – Порівняння ключових параметрів різних архітектур

Модель	**Кількість параметрів (млн)**	**Розмір вхідного зображення**	**Розмір (MB)**	**FLOPs (B)**	**COCO mAP**	**FPS (RTX 3090)**	**Час навчання (год)**
YOLOv8n	3,2	640x640	~6,9	8,7	37,3	~160-200	~10-12
YOLOv11n	2,6	640x640	~6	6,5	39,5	~180-220	~8-10
EfficientDet-D0	3,9	512x512	~16	2,5	34,3	~100	~10
EfficientDet-D7	52,0	1536x1536	~208	325,0	53,4	~8-10	~35-45
RetinaNet (ResNet-50)	34,0	640x640 або 800x1333	~136	97,0	39,2	~50-80	~6-10
Faster R-CNN (R50-FPN)	~42	640x640 або 800x1333	~160	~180	~37-40	~30-40	~8-12
Deformable DETR (R50)	~40	800x800	~152	~173	43,8	~25	~12-14

1.3.1 YOLO

YOLO (You Only Look Once) належить до групи одноетапних моделей розпізнавання об’єктів. Основна концепція цієї архітектури полягає в тому, що зображення проходить через мережу тільки один раз. За цей прохід одночасно визначаються координати об’єктів та їх класи.

YOLOv5 стала одним з першим загальнодоступних варіантів, який має гнучку архітектуру та легко налаштовується. Ця модель повністю реалізована на PyTorch, що спрощує її адаптацію для різних завдань. Існує декілька варіантів конфігурацій: v5n, v5s, v5m, v5l, v5x; це дозволяє обрати оптимальний варіант компромісу між точністю та продуктивністю. Порівняно з попередніми варіантами, дана версія показує покращення за рахунок нової структури “neck” (використання PANet), застосування механізмів оптимізації та регулярізації.

YOLOv11 була реалізована як більш узагальнена та оптимізована модульна система. Основна відмінність та інновація полягає у використанні архітектури “Dynamic Scale Prediction”. Це дозволило змінювати розміри об’єктів під час інференсу, що підвищує точність виявлення дрібних об’єктів. Крім того було використано “Efficient Attention Blocks”, що знизило затрати обчислень при збережені продуктивності.

До переваг YOLOv11 можна віднести наступне:

– висока швидкість роботи навіть у більших версіях (до 150 FPS у YOLOv11m);
– легко адаптувати модель під різні задачі завдяки модульній архітектурі;
– завдяки підтримці трансферного навчання модель легко адаптувати до нових класів;
– підтримка багатьох форматів експорту (ONNX, CoreML, TensorRT);
– покращена автоматична аугментація даних (нові стратегії просторових і колінарних трансформацій);

– динамічне масштабування об’єктів під час прогнозування для підвищення точність на об’єктах різного розміру.

«Однак YOLO має певні обмеження, особливо при виявленні дрібних об’єктів або тих, що розташовані близько один до одного. Підхід, заснований на розбитті зображення на сітку, може мати труднощі з точним визначенням розташування об’єктів, які знаходяться біля меж сітки, що призводить до менш точних передбачень» [11].

Навіть враховуючи недоліки цієї архітектури та певні обмеження щодо точності в складних сценах, вона залишається одним з найефективніших рішень для розпізнавання об’єктів в реальному часі.

1.3.2 EfficientDet

EfficientDet є однією з найпомітніших сучасних архітектур для розпізнавання об’єктів, яка була розроблена командою Google Research. Основна ідея моделі – це використання принципу складного масштабування (compound scaling). Модель заснована на покращеній структурі EfficientNet для основної мережі ознак (backbone), а бінаправлена мережа пірамід ознак (BiFPN) використана для ефективнішого злиття інформації з різних рівнів глибини.

Однією з найпотужніших конфігурацій у серії вважається EfficientDet-D7, яка показує високі результати за точністю на стандартних наборах даних, при цьому вимагаючи середніх обчислювальних ресурсів в порівнянні з традиційними архітектурами. Ця модель пропонує кращий компроміс між якістю виявлення дрібних об’єктів та швидкістю обробки [11].

Серед основних переваг архітектури EfficientDet можна також виділити наступне:

– ефективне використання багаторівневої інформації завдяки BiFPN, що покращує виявлення об’єктів різного розміру;

– гнучкість масштабування моделі під конкретні обмеження або задачі, від дуже легких до високоточних варіантів (D0-D7);
– хороша узагальненість на нових даних та відносно краща стабільність під час тренування.

Водночас є певні недоліки:

– більш складна архітектура у порівнянні з моделями як YOLO чи SSD, що може ускладнювати реалізацію без готових фреймворків;
– довший час навчання через більшу кількість параметрів;
– вища чутливість до правильного налаштування гіперпараметрів може вимагати додаткових експериментів на етапі тренування.

EfficientDet залишається одним з найбільш збалансованих підходів до розпізнавання об’єктів, проте ефективність її використання найбільше розкривається в проєктах, де можливе точне налаштування гіперпараметрів та де є ресурси, що дозволяють підтримувати складніші структури.

1.3.3 RetinaNet

RetinaNet одна з перших одноетапних моделей виявлення об’єктів, що досягла рівня точності, наближеного до двоетапних архітектур. Її основною особливістю є використання функції втрат Focal Loss, яка дає можливість зосередити навчання на складних прикладах, що зменшує вплив великої кількості легких негативних прикладів. Цей підхід значно підвищив здатність моделі справлятись з незбалансованими наборами даних для тренування, де кількість прикладів для одного класу значно перевищує кількість прикладів іншого.

Архітектура RetinaNet заснована на мережі ознак ResNet та використовує піраміду ознак (Feature Pyramid Network) для виявлення об’єктів різного розміру. Також така архітектура показує високу точність при обробці складних сцен з великою кількістю об’єктів та є універсальною для задач, де важлива деталізація дрібних об’єктів. Значною перевагою є передбачуваність

поведінки моделі при зміні типу вхідних даних порівняно з деякими іншими одноетапними альтернативами.

Проте у порівнянні з найшвидшими сучасними моделями (наприклад нові версії YOLO), RetinaNet поступається швидкістю обробки, що може обмежувати її застосування в задачах, де критично важливий аналіз у реальному часі. Крім того, для ефективної роботи часто потрібно точно налаштувати параметри оптимізації під конкретний набір даних.

Загалом RetinaNet довела, що одноетапні моделі можуть забезпечити високу точність навіть для складних сцен. Ця архітектура залишається хорошим рішенням для задач, де пріоритетом є надійність розпізнавання об’єктів в різних умовах і сценах. Хоча обмеження у швидкості можуть ускладнювати її використання в системах реального часу.

1.3.4 Faster R-CNN

Однією з найбільш значущих подій у розвитку підходів до розпізнавання об’єктів стало створення Faster R-CNN. Це була перша архітектура, що об’єднала в одну модель генерацію пропозицій регіонів і класифікацію об’єктів. На відміну від попередніх версій (R-CNN та Fast R-CNN), які використовували зовнішні алгоритми для виділення регіонів інтересу, Faster R-CNN створила регіональну пропозиційну мережу (Region Proposal Network) [1].

Ключова новизна цієї архітектури полягала у спільному використанні ознак, які виділялись базовою згортковою мережею, як для формування гіпотез об’єктів, так і для їхньої подальшої класифікації. Це дозволило уникнути дублювання обчислень та підвищити загальну продуктивність системи.

До переваг Faster R-CNN належать:

– висока точність виявлення навіть у складних сценах та для об’єктів різного розміру;

– можливість роботи з великою кількістю класів без суттєвого погіршення якості;
– архітектура є доволі гнучкою та дозволяє адаптувати модель для різних задач.

Попри ці переваги, Faster R-CNN має ряд обмежень:

– модель є доволі важкою в плані обчислювальних витрат, що обмежує можливості її використання на пристроях із низькою продуктивністю;
– швидкість обробки нижча ніж у сучасніших одноетапних детекторів, таких як YOLO або SSD;
– все ще наявні два логічних етапи (генерація пропозиції та класифікація), що ускладнює оптимізацію всієї системи як єдиного блоку.

Архітектура Faster R-CNN зіграла велику роль для подальших розробок у галузі виявлення об’єктів. Багато сучасних гібридних або вдосконалених методів засновано саме на ідеях, закладених у цій архітектурі. А сама Faster R-CNN залишається актуальним вибором для задач, де важливими є точність і можливість роботи з складними сценами та об’єктами.

1.3.5 Deformable DETR

Розвиток використання трансформерів для задачі виявлення об’єктів розпочався у 2020 році із запропонованої Facebook AI Research моделі DETR (Detection Transformer). Ця архітектура вперше використала глобальний механізм самоуваги, що дозволило значно спростити архітектуру та зменшити кількість спеціально налаштованих гіперпараметрів. Але, незважаючи на спрощену архітектуру, оригінальний DETR мав суттєві обмеження: дуже довгий час навчання, труднощі з виявленням дрібних об’єктів та обмежену ефективність у сценах з багатьма об’єктами.

Для вирішення цих проблем була запропонована модель Deformable DETR, основна відмінність якої полягала у використанні деформованого механізму самоуваги, який обмежує розрахунок уваги не на всьому зображенні, а лише на невеликій кількості релевантних точок. Цей підхід дозволив прискорити процес навчання та зменшити обчислювальні витрати на роботу моделі.

Серед переваг Deformable DETR можна виділити наступне:

– швидке навчання (стабільно хороший результат досягається за меншу кількість епох) навіть на великих наборах даних;
– проста та узагальнена архітектура без необхідності у складних постобробках;
– доволі висока якість виявлення дрібних та перекритих об’єктів;
– завдяки локальній увазі краще масштабування на різні розміри зображень.

Все ще залишаються деякі недоліки:

– швидкість роботи не є найоптимальнішою серед альтернативних одноетапих моделей;
– обробка сцен вимагає якісного налаштування гіперпараметрів кількості точок уваги для отримання оптимальної продуктивності.

У підсумку, ця архітектура підходить для задач, де важливими є глобальне розуміння сцени та висока якість локалізації.

1.3.6 Вибір оптимальної моделі

Вибір моделі для системи розпізнавання об’єктів залежить від багатьох факторів, таких як точність, швидкість інференсу та обчислювальні ресурси. З розглянутих архітектур, найбільш оптимальним рішенням для цієї роботи є використання YOLO11n. Ця модель показує значний прогрес у порівнянні з попередніми версіями YOLO та іншими архітектурами за співвідношенням

точності до обчислювальних витрат. Також при порівнянні з альтернативними моделями, YOLO11n має більш оптимальний баланс характеристик:

– EfficientDet-D7 має дещо вищу точність на COCO наборі даних, але значно нижчу швидкість та вимагає більше ресурсів;
– Deformable DETR має схожі результати за точністю, проте поступається за швидкістю;
– Faster R-CNN показує нижчу точність mAP при вищих обчислювальних вимогах.

Тож для розгортання в реальній системі YOLO11n пропонує оптимальне поєднання точності та швидкості. А завдяки модульній архітектурі та підтримці різних форматів експорту ця модель також забезпечує гнучкість при адаптації для конкретних задач.

1.4 Архітектура YOLOv11

З моменту першої моделі YOLO, архітектура зазнала значної еволюції, що дозволило їй стати одним з найпопулярніших алгоритмів розпізнавання об’єктів в реальному часі. Кожна ітерація вносила зміни (див. рис. 1.6), які покращували точність та швидкість роботи.

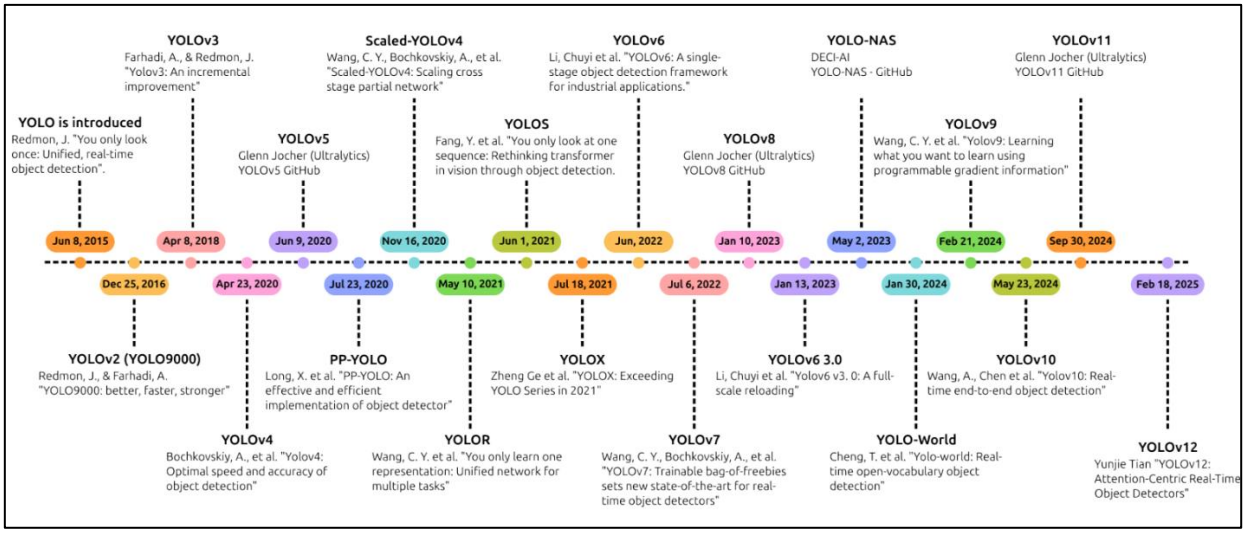


Рисунок 1.6 – Еволюція архітектури YOLO [19]

У 2015 році YOLOv1 представила концепцію одноетапної детекції, що суттєво прискорило розпізнавання об’єктів в порівнянні з двоетапними методами.

YOLOv2 покращила точність завдяки впровадженню якірних рамок та Batch Normalization.

YOLOv3 використала багаторівневе розпізнавання з Feature Pyramid Network, що покращило виявлення об’єктів різного масштабу.

YOLOv4 впровадило нові техніки, такі як CSPDarknet53 в якості базової мережі та різні методи аугментації даних.

YOLOv5 представила модульну архітектуру, а покращення YOLOv6 та YOLOv7 були зосереджені на підвищені ефективності для використання на мобільних пристроях.

YOLOv8 впровадила нову структуру шиї (neck) моделі для покращеного виділення ознак.

YOLOv9 в свою чергу запровадила технологію Programmable Gradient Information (PGI) для покращення навчання.

YOLOv10 запропонувала вдосконалену систему втрат (loss) та новий механізм уваги.

YOLOv11 запровадила інноваційну гібридну архітектуру з використанням трансформерів та адаптивної активації, що значно покращило якість розпізнавання об’єктів різних розмірів.

Найновіша ітерація YOLOv12 розширила можливості моделі завдяки впровадженню динамічної оптимізації пам’яті та самоадаптивних блоків навчання.

Архітектура YOLOv11 складається з трьох ключових компонентів (див. рис. 1.7):

– базова мережа (backbone);
– шия моделі (neck);
– головна частина (head).

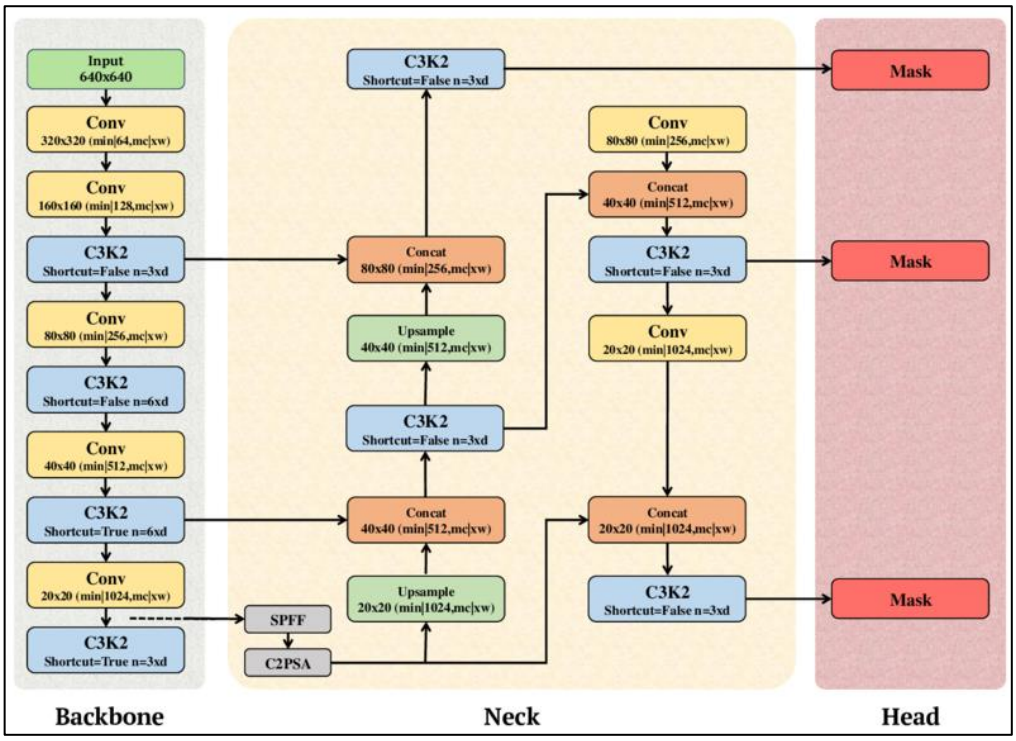


Рисунок 1.7 – Архітектура YOLOv11 [20]

1.4.1 Базова мережа (backbone)

Базова мережа YOLOv11 заснована на архітектурі під назвою HybridFormer. Вона об’єднує найкращі риси нейронний мереж (CNN) та трансформерів. Цей підхід дозволяє захоплювати локальні і глобальні особливості зображень.

У основі HybridFormer лежить покращений CSPDarknet, що включив нові блоки C3-SwinT (Convolutional Swin Transformer). Ці блоки поєднують звичайні згорткові шари з віконним механізмом уваги. Зокрема, використовується нова техніка Cross-Window Attention, яка забезпечує більш цілісне розуміння контексту зображення завдяки можливості обміну інформацією між різними вікнами.

Важливим оновленням також є використання адаптивної активаційної функції DyReLU (Dynamic ReLU), яка автоматично налаштовує свої параметри в залежності від вхідних даних. Це дозволяє моделі бути більш гнучкою при обробці різних патернів.

1.4.2 Шия моделі (neck)

Шия YOLOv11 використовує архітектуру BiPAFPN+ (Bidirectional Path Aggregation Feature Pyramid Network Plus), яка дозволяє об’єднувати інформацію з різних рівнів абстракції. Ключовою особливістю є Cross-Scale Fusion (CSF) блоки. Вони об’єднують ознаки з різних масштабів за допомогою днамічних вагових коефіцієнтів.

Додатково в цій частині архітектури реалізовано механізм Channel-Spatial Attention Module (CSAM), який вибірково фокусується на релевантних каналах ознак та регіонах. Цей механізм працює за принципом «вузьких місць» уваги, стискаючи інформацію для отримання карти значимості, після чого використовуючи її для повторного зважування ознак.

1.4.3 Головна частина (head)

Головна частина YOLOv11 зазнала змін, що забезпечило точніше передбачення обмежувальних рамок та класифікацію об’єктів. Замість традиційного одноетапного детектора, було використано Decoupled Detection Head. Ця технологія розділяє завдання класифікації та регресії обмежувальних рамок, дозволяючи кожній підмережі фокусуватись на своєму завданні.

Крім цього, головна частина включає механізм Oriented Object Detection, який дозволяє виявити об’єкти з довільною орієнтацією. Це особливо корисно для завдань аерофотозйомки та подібних. Механізм засновується а представленні обмежувальних рамок у вигляді прямокутників, які визначаються наступними параметрами: координатами центру (x, y), шириною, висотою та кутом повороту.

Додатково було додано нову функцію витрат – Distribution Focal Loss (DFL), яка замість прямого передбачення координат рамок моделює їх розподіл ймовірностей. Це забезпечує більш стабільне тренування та зменшує вплив викидів даних.

1.4.4 Додаткові особливості

YOLOv11 також включає ряд інших змін в архітектурі, які покращують продуктивність моделі. Серед них:

- Adaptive Feature Selection (AFS) – механізм, який динамічно обирає найбільш релевантні ознаки для кожного зображення;
- Contextual Instance Relation Encoder (CIRE) – моделює взаємозв’язки між різними об’єктами на зображенні, покращуючи розуміння сцени;
- Multi-Scale Gradient Harmonization (MSGH) – стабілізує градієнти під час навчання при роботі з об’єктами різного масштабу;
- Temporal Vision Transformer (TVT) – функція, що дозволяє моделі ефективно працювати не тільки із зображеннями, а й з відеопотоками;
- Quantization-Aware Training (QAT) – вбудована підтримка квантизації моделі з 32-бітних чисел з плаваючою точкою до 8-бітних цілих чисел, що значно зменшує розмір моделі та прискорює інтерфейс без значних втрат точності;
- новий механізм активації ACON (Active or Not) – автоматично визначає, чи слід активувати певні нейрони для конкретного вхідного сигналу.

2 РОЗРОБКА ТА НАВЧАННЯ МОДЕЛІ РОЗПІЗНАВННЯ ОБ'ЄКТІВ

2.1 Підготовка тренувального набору даних

2.1.1 Вибір класів об'єктів та джерел даних

Підготовка якісного набору даних є важливим етапом у процесі тренування моделі машинного навчання. Для задачі розпізнавання військової техніки було визначено чотири класи:

- AFV (Armored Fighting Vehicle) – бойові броньовані машини, бойові танки (Т-64, Т-72, Т-80, Т-90);
- APC (Armored Personnel Carrier) – бронетранспортери та бойові машини піхоти (БТР-70, БТР-80, БМП-1, БМП-2, БМД-2, МТ-ЛБ);
- Artillery – артилерійські системи, включно з самохідними артилерійськими установками та реактивними системами залпового вогню (БМ-21 «Град», «Смерч», «Ураган»);
- Air-Defense – системи протиповітряної оборони (ЗРК «БУК», С-300, «Тор», «Панцир»).

Основним джерелом даних для створення тренувального набору даних є платформа Roboflow [12]. Цей сервіс надає можливість швидко підготувати, анотувати набір даних з зображеннями, навчити нейромережу через зручний вебінтерфейс та інтегрувати отриману модель різними засобами. Крім цього, на платформі є доступ до публічних наборів даних, створених іншими користувачами.

На початку розглядалась можливість включення даних з ImageNet – масштабної бази даних зображень. Було проаналізовано класи ImageNet, пов'язані з військовою технікою, але вирішено не включати їх у фінальний набір даних з наступних причин:

- достатня кількість якісних зображень у сформованому наборі даних;

– зображення є дуже різнотипними в межах одного класу, що може негативно вплинути на тренування моделі.

2.1.2 Процес формування набору даних

Створення збалансованого набору даних включало кроки завантаження наборів даних з Roboflow, обробку та перетворення міток класів, фільтрацію та об’єднання наборів даних. Також було проведено аналіз розподілу класів та експортовано набір даних у формат YOLO.

Для автоматизації було створено набір функцій (повний код надалі згаданих функцій наведено в Додатку А), ключовою з яких є process_roboflow_dataset, яка обробляє кожен набір даних та інтегрує його у загальний набір даних (див. рис. 2.1). Ця функція завантажує датасет зі вказаного посилання, перейменовує мітки класів, фільтрує зображення без міток або з мітками, що відрізняються від обраних класів та об’єднує з існуючим набором даних.

```
roboflow_dir = base_dir / "roboflow1"
roboflow_url = "https://universe.roboflow.com/ds/SU46w8zITA?key=WYPC3O9Qzo"
label_mapping = {
    "bm-21": "Artillery",
    "t-80": "AFV",
    "t-64": "AFV",
    "t-72": "AFV",
    "bmp-1": "APC",
    "bmp-2": "APC",
    "bmd-2": "APC",
    "btr-70": "APC",
    "btr-80": "APC",
    "mt-lb": "APC",
}

dataset = process_roboflow_dataset(roboflow_url, roboflow_dir, label_mapping, delete_empty=False)
```

```
2025-05-11 03:01:14,837 - INFO - Завантажуємо D:\Projects\military_detection_dataset\roboflow1\dataset_rf.zip з https://u
2025-05-11 03:01:21,426 - INFO - Завантаження завершено.
2025-05-11 03:01:22,608 - INFO - Розпаковано у D:\Projects\military_detection_dataset\roboflow1
2025-05-11 03:01:44,030 - INFO - Директорія датасету успішно реструктуризована для підмножин: ['test', 'train', 'valid'].
 100% |████████████| 1042/1042 [5.3s elapsed, 0s remaining, 190.7 samples/s]
Доступні класи перед обробкою: {'bmp-1', 'mt-lb', 'btr-70', 't-80', 'bmd-2', 'bmp-2', 't-64', 't-72', 'bm-21', 'btr-80'}
Кількість класів після об'єднання датасетів:
APC: 595 зображень
Artillery: 100 зображень
AFV: 298 зображень
```

Рисунок 2.1 – Приклад використання функції process_roboflow_dataset

Для забезпечення репрезентативної вибірки було обрано та оброблено 26 різних наборів даних, щоб охопити зображення з різних ракурсів, умов спостереження та моделей військової техніки.

2.1.3 Аналіз та експорт набору даних

Після обробки всіх наборів даних було проведено аналіз отриманого набору даних за допомогою функції analyze_dataset_classes. Результати наведено на рисунку 2.2: загальний набір даних включає 43845 зображень, які містять 66487 анотованих об’єктів. Розподіл між класами є нерівномірним, з найменшою представленістю класу “Air-Defense” (11,1% зображень). Цей розподіл обумовлений нестачею доступних даних деяких класів і буде врахована при навчанні моделі.

```
analyze_dataset_classes(dataset)


=== Dataset Class Distribution ===
Total samples: 43845

Samples per class:
  AFV: 14321 samples (32.7%)
  APC: 16392 samples (37.4%)
  Air-Defense: 4872 samples (11.1%)
  Artillery: 8972 samples (20.5%)

Objects per class:
  AFV: 22070 objects (33.2%)
  APC: 21279 objects (32.0%)
  Air-Defense: 8332 objects (12.5%)
  Artillery: 14806 objects (22.3%)
```

Рисунок 2.2 – Розподіл зображень та об’єктів класів

Для фінального етапу підготовки набору даних була створена функція split_and_export_to_yolo, яка виконала поділ датасету на навчальну, валідаційну та тестову вибірки з урахуванням розподілу класів. Ця функція забезпечила пропорційне представлення класів в усіх вибірках у співвідношенні 85% для навчання, 10% для валідації та 5% для тестування. Для класів з меншою представленістю (“Air-Defense” та “Artillery”) було

застосовано розподіл з перевагою на користь тренувальної вибірки (92% та 90% відповідно). Результати експорту наведено на рисунку 2.3: загальна кількість зображень у тренувальній вибірці 38065 зображень.

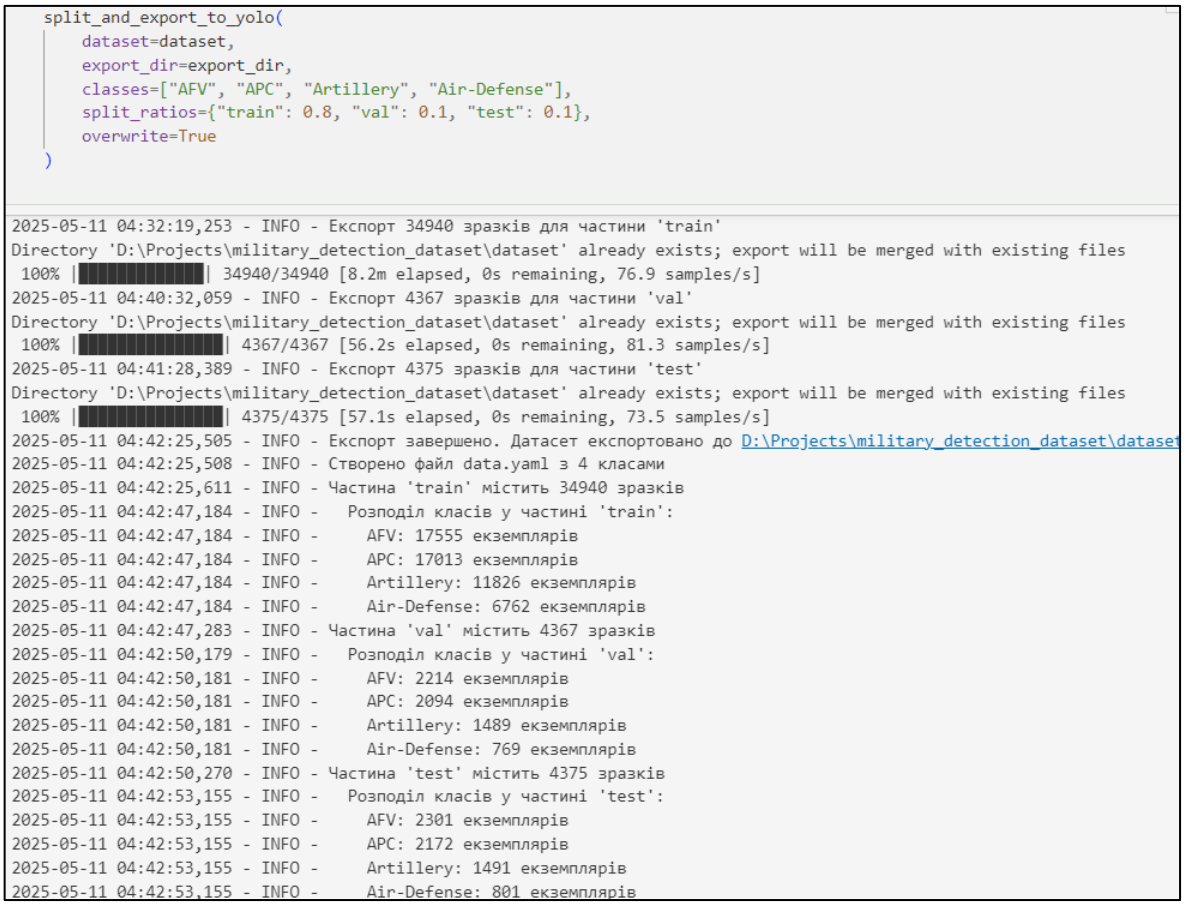


Рисунок 2.3 – Експорт набору даних у форматі YOLO

Отриманий датасет було збережено у форматі YOLO з відповідним конфігураційним файлом dataset.yaml, що містить інформацію про класи та шляхи до вибірок.

2.2 Навчання моделі YOLOv11 та оцінка точності

Вибір оптимальних параметрів навчання YOLO є іншим важливим етапом для досягнення високої точності моделі. Базові параметри навчання визначають загальну конфігурацію процесу навчання:

– epochs – кількість повних проходів через тренувальний набір даних, було обрано 110 епох для забезпечення достатнього часу навчання;
– batch – кількість зображень, що обробляються за один крок оптимізації, значення 128 зберігає баланс між стабільним градієнтним спуском та ефективним використанням GPU для невеликої моделі YOLO11n;
– imgsz=512 – розмір вхідних зображень для моделі, це стандартне значення для моделей YOLO, що також забезпечує баланс між точністю виявлення та обчислювальними витратами;
– amp=True – автоматична змішана точність обчислень, дозволяє пришвидшити навчання та зменшити використання GPU без значної втрати точності.

Параметри оптимізації регулюють, як модель навчається і знаходить оптимальні ваги:

– optimizer=SGD – оптимізатор, який має кращу здатність до генералізації та меншій схильності до перенавчання, зазвичай забезпечує ефективне подолання локальних мінімумів та менше споживання пам’яті;
– lr0=0.01, lrf=0.001 – початкова та фінальна швидкість навчання, цей параметр є одним з найголовніших гіперпараметрів та визначає наскільки великі кроки робить алгоритм оптимізації у напрямку мінімальних втрат (високе значення на початку дозволяє швидко наближатись до оптимальних ваг на початку, а низьке кінцеве значення надає тонке налаштування під кінець навчання);
– momentum=0.937 – коефіцієнт моменту для оптимізатора, який допомагає подолати локальні мінімуми та пришвидшити збіжність, зберігаючи інформацію про напрямок попередніх градієнтів;
– weight_decay=0.0005 – коефіцієнт регуляризації L2, що запобігає перенавчанню моделі шляхом штрафування великих ваг (значення 0.0005 є рекомендованим для YOLOv11 і забезпечує баланс між

точністю на тренувальному наборі даних та здатністю до узагальнення);

– patience=20 – кількість епох без покращення для раннього зупинення навчання для запобігання перенавчання (зупиняє процес навчання, коли модель перестає покращуватись на валідаційній вибірці).

Параметри розігріву забезпечують стабільність навчання на початкових етапах:

– warmup_epochs=3.0 – кількість епох початкового розігріву, під час цих епох швидкість навчання поступово збільшується від низького значення до заданого lr0 (дозволяє стабілізувати початковий процес навчання);

– warmup_momentum=0.8 – значення моменту під час епох розігріву, менше ніж основне значення для забезпечення більш плавного оновлення на початку;

– warmup_bias_lr=0.1 – спеціальна швидкість навчання для bias параметрів під час розігріву.

Параметри аугментації даних також є критично важливими для підвищення різноманітності тренувальних зразків та запобігання перенавчанню:

– mosaic=1.0 – коефіцієнт застосування мозаїчної аугментації, яка комбінує чотири різних зображення в одне (значення 1 означає що усі зображення будуть проходити через цю аугментацію);

– close_mosaic=20 – епохі, після якої вимикається мозаїчна аугментація, це дозволяє на пізніших етапах фокусуватись на оригінальних зображеннях для кращого тонкого налаштування;

– hsv_h=0.015, hsv_s=0.7, hsv_v=0.4 – параметри аугментації кольорового простору HSV (відтінок, насиченість та яскравість), помірне значення для відтінку зберігає оригінальні кольори зображення, а більші значення для насиченості та яскравості дозволяють моделі адаптуватись до різних умов освітлення;

– fliplr=0.5 – ймовірність горизонтального віддзеркалення зображення;
– degrees=5.0 – максимальний кут повороту зображення (значення 5 дозволяє імітувати різні ракурси спостереження без спотворення об’єктів);
– scale=0.5 – коефіцієнт випадкового масштабування;
– erasing=0.4 – випадкове стирання ділянок зображення (підвищує стійкість моделі до часткового перекриття об’єкта та різних умов видимості).

Фрагмент коду, який було використано для тренування моделі показано на рисунку 2.4, а на рисунку 2.5 зображення логування процесу тренування.

```
results_train = model.train(
    data=dataset_dir / "dataset.yaml",  # Конфігурація даних
    epochs=110,                         # Кількість епох
    batch=128,                          # Розмір батчу
    imgsz=512,                          # Розмір зображення
    device=[0, 1],                      # Пристрій
    amp=True,                           # Автоматична змішана точність
    optimizer="SGD",                    # Оптимізатор
    lr0=0.01,                           # Початкова швидкість навчання
    lrf=0.001,                          # Фінальна швидкість навчання
    weight_decay=0.0005,                # Коефіцієнт регуляризації L2
    momentum=0.937,                     # Коефіцієнт моменту для оптимізатора
    patience=20,                        # Кількість епох без покращення для раннього зупинення
    warmup_epochs=3.0,                  # Кількість епох початкового розігріву
    warmup_momentum=0.8,                # Значення моменту під час епох розігріву
    warmup_bias_lr=0.1,                 # Швидкість навчання для bias параметрів під час розігріву
    mosaic=1.0,                         # Коефіцієнт застосування мозаїчної аугментації
    close_mosaic=20,                    # Епоха, після якої вимикається мозаїчна аугментація
    hsv_h=0.015,                        # Аугментація відтінку
    hsv_s=0.7,                          # Аугментація насиченості
    hsv_v=0.4,                          # Аугментація яскравості
    fliplr=0.5,                         # Ймовірність горизонтального віддзеркалення
    degrees=5.0,                        # Максимальний кут повороту зображення
    scale=0.5,                          # Коефіцієнт випадкового масштабування
    erasing=0.4,                        # Випадкове стирання ділянок зображення
    workers=2,                          # Кількість паралельних процесів
    cache=True                          # Кешування для пришвидшення
)
```

Рисунок 2.4 – Код тренування моделі YOLOv11

Тренування моделі відбувалось у середовищі Kaggle, адже воно надає безкоштовний доступ до потужних графічних прискорювачів (NVIDIA Tesla P100 та T4), які дозволяють значно прискорити процес навчання глибоких нейронних мереж з великими наборами даних.

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
5/110      5.64G      1.182      1.448      1.303         76        512: 100%|██████████| 298/298 [03:36<00:00,  1.37it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 31/31 [00:19<00:00,  1.58it/s]
             all       3862       5779      0.619      0.492       0.53      0.354

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
6/110      5.64G      1.148      1.357      1.274         58        512: 100%|██████████| 298/298 [03:35<00:00,  1.38it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 31/31 [00:19<00:00,  1.59it/s]
             all       3862       5779      0.625      0.511      0.538      0.366

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
7/110      5.64G      1.123      1.296      1.262        108        512: 100%|██████████| 298/298 [03:35<00:00,  1.38it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 31/31 [00:19<00:00,  1.60it/s]
             all       3862       5779      0.648      0.584      0.614      0.419

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
8/110      5.65G      1.109      1.249      1.247         59        512: 100%|██████████| 298/298 [03:35<00:00,  1.38it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 31/31 [00:19<00:00,  1.61it/s]
             all       3862       5779      0.607      0.587      0.626      0.445
```

Рисунок 2.5 – Логування процесу тренування YOLOv11

Після завершення навчання моделі було проведено оцінку її точності на тестовій вибірці даних. Для оцінювання використовувались метрики mAP (mean Average Precision), precision, recall та деякі інші. Для передбачення було використано функцію predict бібліотеки Ultralytics (див. рис. 2.6).

```python
model = YOLO(model_path)
source = dataset_dir / "images/test"
results = model.predict(source, stream=False, save_txt=True, save_conf=True)
```

Рисунок 2.6 – Код передбачення для тестової вибірки

Після проведення передбачень для тестової вибірки були отримані метрики ефективності навченої моделі. Як видно з матриці помилок (див. рис. 2.7), модель показує високу точність класифікації для всіх цільових класів, проте є відносно висока ймовірність помилкової класифікації об’єктів класів “AFV” та “APC” як фону (background). Це може бути пов’язано з подібністю візуальних характеристик цих об’єктів до елементів навколишнього середовища при певних умовах.

На графіках навчання (див. рис. 2.8) можна спостерігати поступове зменшення функції втрат як для тренувальної, так і для валідаційної вибірок, що свідчить про відсутність перенавчання. Метрики точності, в свою чергу, зростають швидко на початку навчання та досягають високих значень вже після 50 епох.

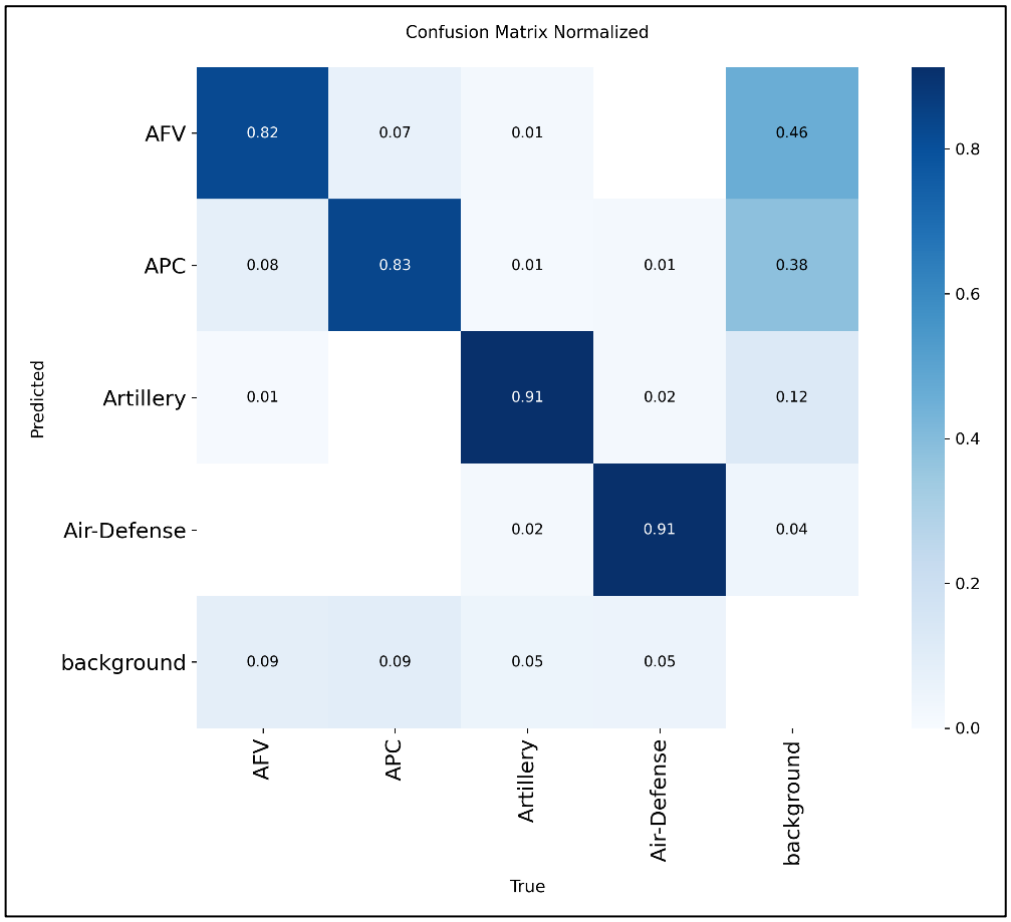


Рисунок 2.7 – Матриця помилок

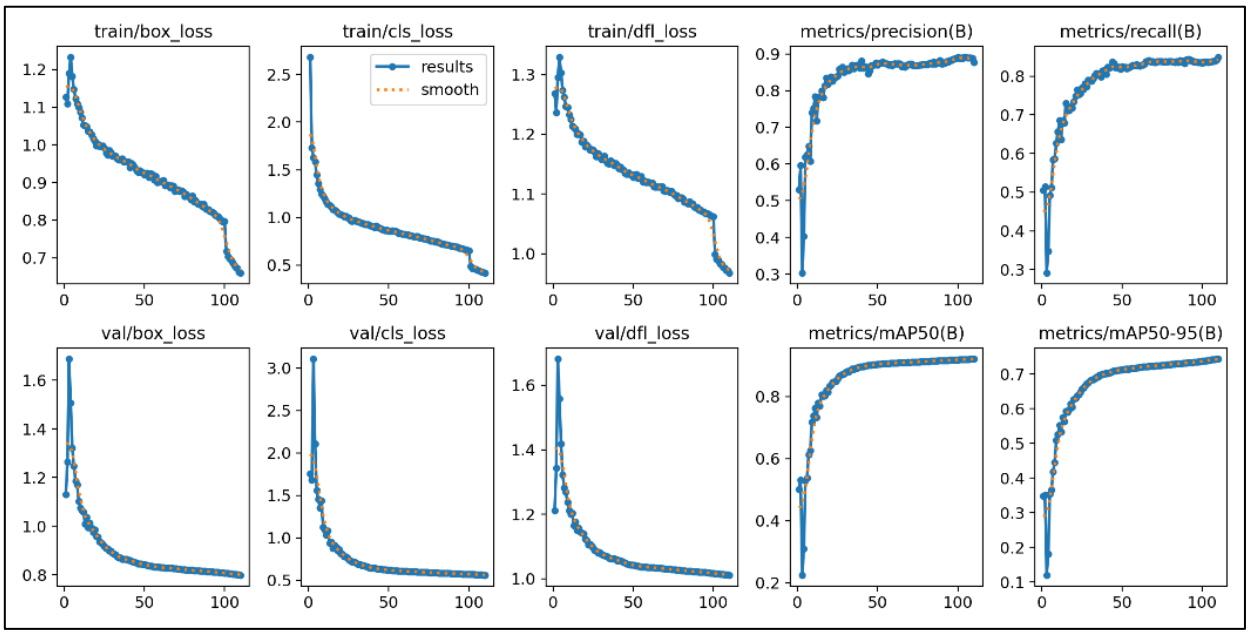


Рисунок 2.8 – Графіки процесу навчання

Більш детальний аналіз метрик по класам (див. рис. 2.9) підтверджує хорошу ефективність розробленої моделі.

Class	Precision	Recall	F1-score	Support	AP
AFV	0.85	0.82	0.83	1073	0.79
APC	0.81	0.87	0.84	1051	0.79
Artillery	0.89	0.91	0.90	434	0.90
Air-Defense	0.88	0.95	0.91	247	0.90
micro avg	0.84	0.86	0.85	2805	
macro avg	1.07	1.10	1.08	2805	
weighted avg	2.75	2.83	2.79	2805	

mAP@0.5: 0.8454

Рисунок 2.9 – Таблиця детальних метрик оцінки точності моделі

Незважаючи на те, що зразків у наборі даних для класів “Air-Defense” та “Artillery” було менше, вони демонструють найвищі показники F1-score. Загальний показник mAP@0.5 для моделі становить 0.8454, що є хорошим результатом для поставленої задачі.

3 РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ВИКОРИСТАННЯ МОДЕЛЛІ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ

Для зручної взаємодії з моделлю машинного навчання було вирішено створити вебзастосунок, використовуючи бібліотеку Streamlit.

3.1 Огляд Streamlit як інструменту для створення вебзастосунків

Streamlit – це відкрита бібліотека Python, яка була спеціально розроблена для створення вебзастосунків з інтерактивними інтерфейсами для проєктів у сферах машинного навчання та аналізу даних [13]. Цей інструмент є зручним, адже на відміну від традиційних фреймворків, таких як Flask або Django, він не вимагає знання HTML, CSS або JavaScript. Замість цього, весь інтерфейс розробляється на Python.

Основна архітектура Streamlit заснована на принципі реактивності: кожного разу, як користувач взаємодіє з елементами інтерфейсу, код додатку перезапускається повністю.

Переваги використання Streamlit для даного проєкту:

- можливість створювати вебдодатки дуже швидко та без глибокого розуміння вебтехнологій;
- безпроблемна взаємодія з бібліотеками машинного навчання (TensorFlow, PyTorch, scikit-learn, Keras, Numpy, OpenCV), що дозволяє інтегрувати різні моделі та алгоритми безпосередньо у застосунок;
- широкі можливості візуалізації даних;
- існування вбудованих можливостей для відображення та взаємодії із зображеннями та відео;
- кешування обчислень.

Тож, в контексті розробки застосунку для роботи з моделлю розпізнавання об’єктів, Streamlit надає всі необхідні інструменти для створення ефективного та зручного інтерфейсу.

3.2 Реалізація функціоналу додатку

Розроблений додаток являє собою вебінтерфейс на базі Streamlit для розпізнавання військової техніки на зображеннях, відео та потоки. Додаток підтримує різні джерела вхідних даних: статичні зображення, відеофайли, YouTube відео та RTSP-потоки.

Архітектурно додаток складається з трьох основних компонентів:

– app.py – головний модуль, що містить конфігурацію вебінтерфейсу;
– config.py – файл з налаштуваннями та константами проєкту (див. рис. 3.1);
– worker.py – модуль, що реалізує функції обробки зображень та відео.

```python
from pathlib import Path
import sys

FILE = Path(__file__).resolve()
ROOT = FILE.parent
if ROOT not in sys.path:
    sys.path.append(str(ROOT))
ROOT = ROOT.relative_to(Path.cwd())

IMAGE = "Image"
VIDEO = "Video"
RTSP = "RTSP"
YOUTUBE = "YouTube"

CLASSES = ['']
SOURCES_LIST = [IMAGE, VIDEO, RTSP, YOUTUBE]

IMAGES_DIR = ROOT / "images"
DEFAULT_IMAGE = IMAGES_DIR / "image.jpg"

VIDEO_DIR = ROOT / "videos"
VIDEOS_DICT = {
    "video_1": VIDEO_DIR / "afv_apc.mp4",
}

MODEL_DIR = ROOT / "weights"
DETECTION_MODEL = MODEL_DIR / "best.pt"
```

Рисунок 3.1 – Зміст файлу config.py

Додатково також існують каталоги для зберігання відео та моделей (див. рис. 3.2). Повний код програми наведено в Додатку Б.

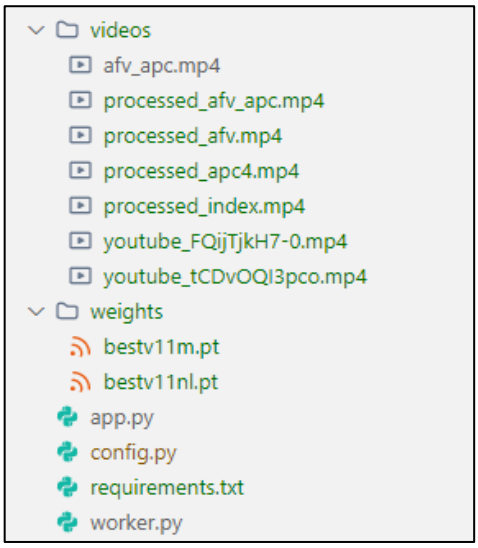


Рисунок 3.2 – Структура проєкту

3.2.1 Обробка зображень

Функціонал обробки зображень реалізований у функції detect_on_image() модуля worker.py (див. рис. 3.3).

```python
def detect_on_image(conf, model):
    """
    Виконує детекцію на зображеннях.
    """
    source_imgs = st.sidebar.file_uploader(
        "Завантаження зображень...",
        type=("jpg", "jpeg", "png"),
        accept_multiple_files=True
    )

    if source_imgs:
        for i, source_img in enumerate(source_imgs):
            col1, col2 = st.columns(2)

            with col1:
                original = Image.open(source_img)
                st.image(original, use_column_width=True)

            with col2:
                img_array = np.array(original)
                res = model.predict(img_array, conf=conf)
                annotated_img = res[0].plot()
                st.image(annotated_img, use_column_width=True)

            st.write("")
            with st.expander(f"Результати обробки для зображення {i+1}"):
                for i, box in enumerate(res[0].boxes):
                    data = box.data[0]
                    st.write(f"Об'єкт #{i+1}:")
                    st.write(f"- Клас: {data[5]}")
                    st.write(f"- Впевненість: {data[4]*100:.2f}%")
                    st.write(f"- Координати: x1={data[0]:.1f}, y1={data[1]:.1f}, x2={data[2]:.1f}, y2={data[3]:.1f}")
            st.write("")
```

Рисунок 3.3 – Реалізація функції detect_on_image

Користувач має можливість завантажити одне або декілька зображень у форматах JPG, JPEG або PNG. Завантажені зображення обробляються моделлю з заданим користувачем рівнем впевненості. Після обробки зображення моделлю, результати будуть показані в додатку наступним чином: оригінальне зображення у лівій колонці, оброблене зображення у правій колонці та опис результатів розпізнавання під ними (див. рис. 3.4).

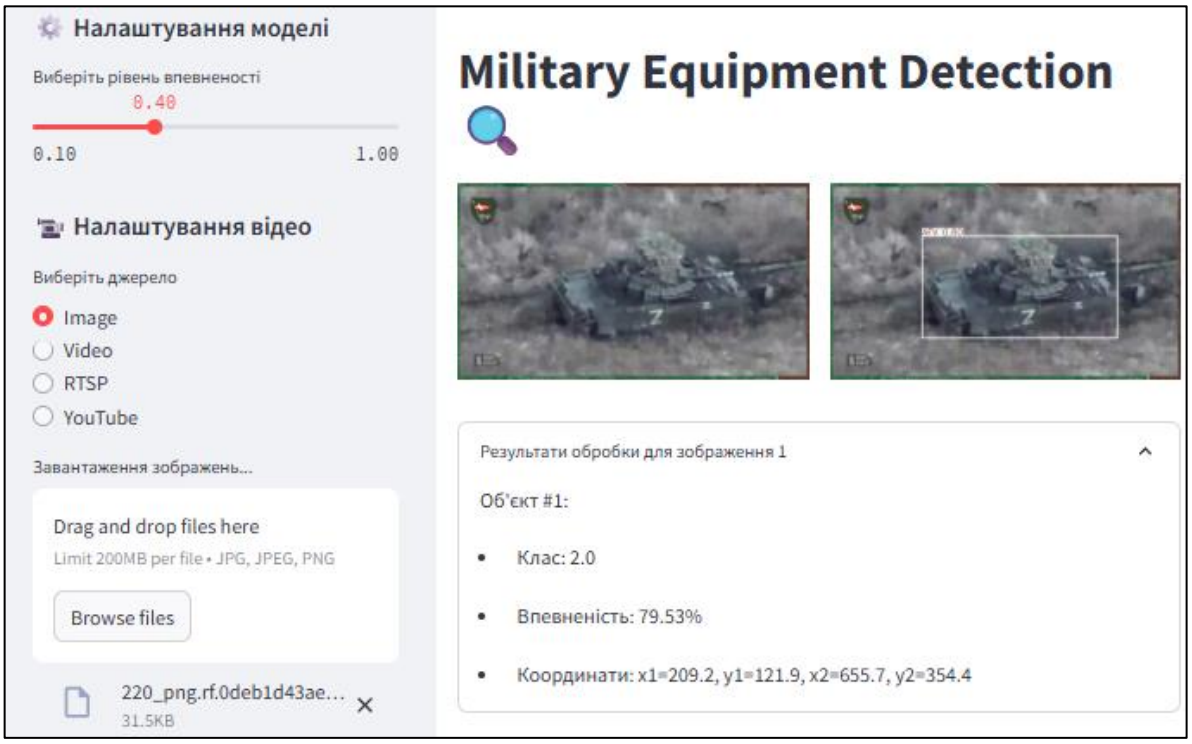


Рисунок 3.4 – Результати обробки зображення

3.2.2 Обробка відеофайлів

Для обробки відеофайлів реалізовано функцію play_stored_video(). Користувач має можливість обрати відео з попередньо визначеного списку або завантажити власний відеофайл. Обробка відео здійснюється покадрово та з використанням трекера об’єктів.

Після завантаження та запуску обробки відео відображається в реальному часі, після чого з оброблених кадрів сформовано відеофайл, який можна переглянути та завантажити (див. рис. 3.5).

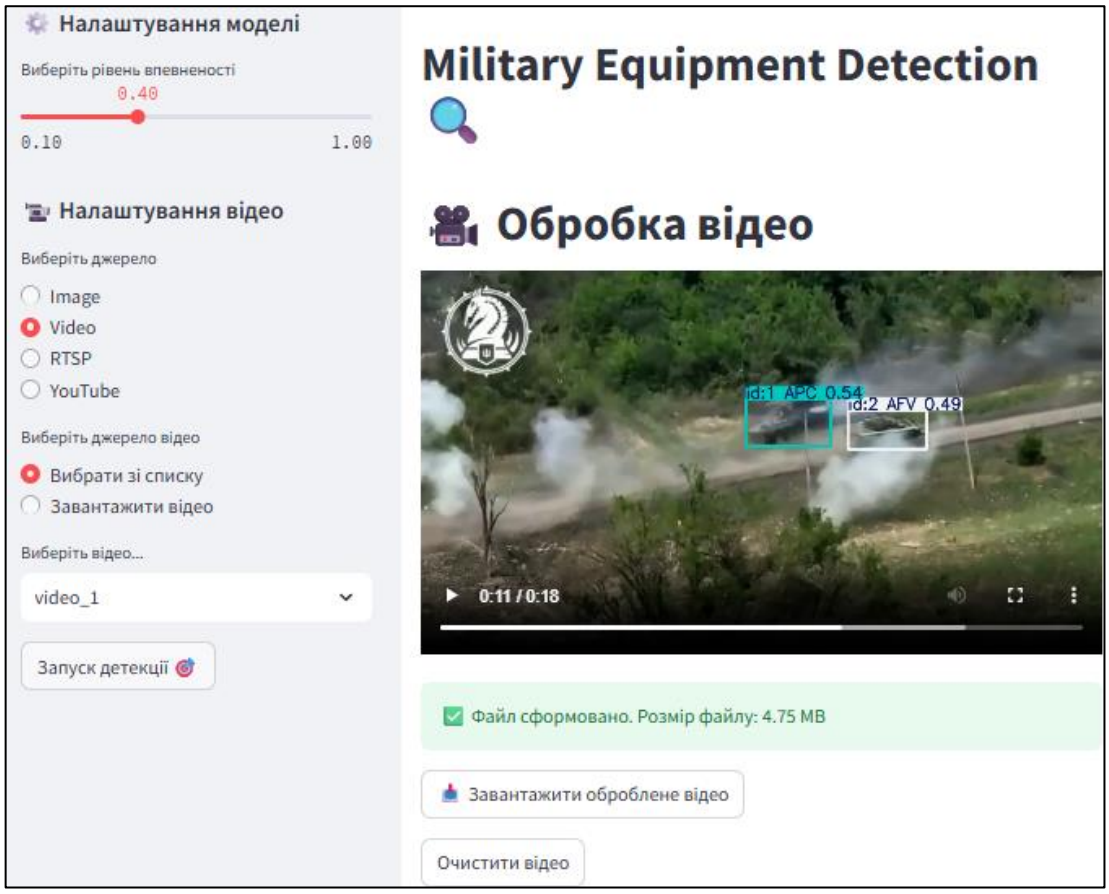


Рисунок 3.5 – Результат обробки відеофайлу

Для відстеження об’єктів між кадрами використовується алгоритм ByteTrack. Також було реалізовано допоміжну функцію get_frames_and_detect(), яка відповідає за зчитування кадрів з джерел, їх обробку та збереження. Функція є універсальною для всіх типів відеоджерел і забезпечує уніфікований підхід до обробки.

3.2.3 Обробка відео з YouTube

Функція play_youtube_video() забезпечує можливість обробки відео з YouTube по URL-адресі. Отримання прмого посилання на відеопотік виконується функцією get_youtube_stream_url() за допомогою бібліотеки yt_dlp (див. рис. 3.6). Принцип обробки самого відео аналогічний до обробки звичайних відеофайлів, а результати показано на рис. 3.7.

```python
def get_youtube_stream_url(youtube_url):
    """
    За допомогою yt_dlp витягує пряме посилання на відео-потік YouTube.
    Уникає формати AV1, які можуть викликати проблеми з декодуванням.
    """
    if not youtube_url:
        raise ValueError("URL відео не може бути порожнім")

    ydl_opts = {
        'format': 'bestvideo[ext=mp4][vcodec!*=av01][vcodec^=avc]+bestaudio[ext=m4a]/best[ext=mp4][vcodec!*=av01]/best[vcodec!*=av01]/best',
        'quiet': True,
        'no_warnings': True
    }

    try:
        with yt_dlp.YoutubeDL(ydl_opts) as ydl:
            info = ydl.extract_info(youtube_url, download=False)

            if 'url' in info:
                return info['url']

            if 'requested_formats' in info:
                for fmt in info['requested_formats']:
                    if fmt.get('vcodec', '').startswith('avc'):
                        return fmt['url']

            if 'formats' in info:
                for fmt in info['formats']:
                    vcodec = fmt.get('vcodec', 'none')
                    if (vcodec != 'none' and 'av01' not in vcodec and
                        fmt.get('height', 0) >= 360):
                        return fmt['url']

            if 'formats' in info and info['formats']:
                for fmt in sorted(info['formats'], key=lambda x: x.get('height', 0), reverse=True):
                    if fmt.get('vcodec', 'none') != 'none' and fmt.get('height', 0) > 0:
                        return fmt['url']

            raise ValueError("Не вдалося отримати відповідний URL потоку")

    except Exception as e:
        raise Exception(f"Помилка при обробці відео: {str(e)}")
```

Рисунок 3.6 – Реалізація функції get_youtube_stream_url

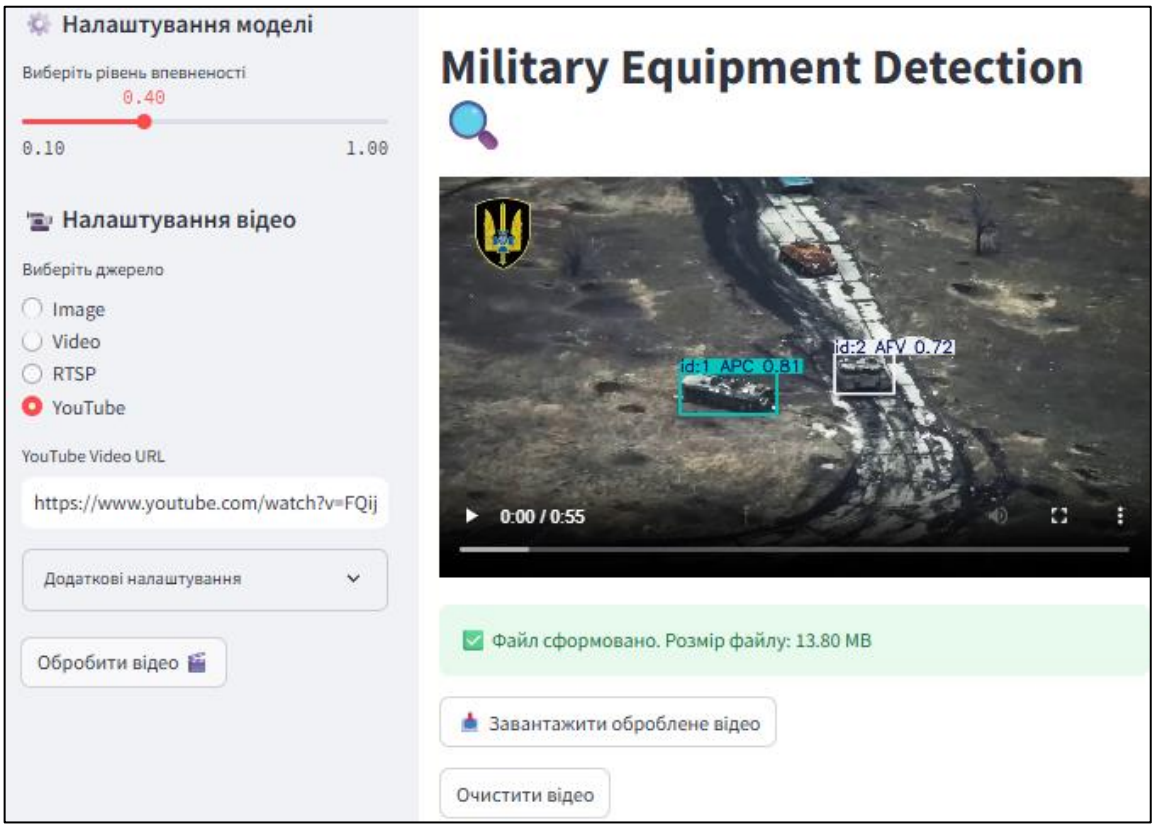


Рисунок 3.7 – Результат обробки YouTube відео

3.2.4 Обробка RTSP

Для обробки відеопотоків з RTSP джерел було реалізовано функцію play_rtsp_stream(). Ця функціональність забезпечує зручну роботу з IP-камерами та системами відеоспостереження.

Відображення оброблених кадрів також відбувається в реальному часі (див. рис. 3.8). Для роботи з RTSP використовується функція get_frames_and_detect, але з урахуванням особливостей потокового відео (див. рис. 3.9 та 3.10 відповідно).



Рисунок 3.8 – Результат обробки RTSP

```python
if source.startswith('rtsp://'):
    while vid_cap.isOpened():
        success, frame = vid_cap.read()
        if not success:
            break

        if tracker:
            res = model.track(frame, conf=conf, tracker=tracker)
        else:
            res = model.predict(frame, conf=conf, stream=True)
```

Рисунок 3.9

```
            res = model.predict(frame, conf=conf, stream=True)

        processed_frame = res[0].plot()

        st_frame.image(
            processed_frame,
            caption="Processing...",
            use_column_width=True,
            channels="BGR"
        )

    vid_cap.release()
    clean_temp_files()
    return None
```

Рисунок 3.10

З’єднання з RTSP потоками здійснюється а допомогою бібліотеки OpenCV, обробка відбувається в реальному часі, без збереження у відео.

ВИСНОВКИ

Під час виконання роботи було розроблено вебзастосунок для розпізнавання військової техніки з використанням моделі машинного навчання YOLO11.

У першому розділі роботи було проведено аналіз сучасних методів розпізнавання об’єктів. Дослідження показало, що підходи до розпізнавання об’єктів можна поділити на три основні категорії: двоетапні (R-CNN архітектури), одноетапні моделі (YOLO, SSD та RetinaNet) та моделі на основі трансформерів (DETR та його модифікації). При порівнянні різних архітектур за ключовими параметрами було визначено, що YOLO11 є оптимальним вибором та надає необхідний баланс між точністю та швидкістю обробки.

Другий розділ було присвячено процесу формування набору даних для навчання моделі, навчанню моделі та оцінці результатів. Було сформовано набір даних, що включає зображення чотирьох класів: бойові броньовані машини (AFV), бронетранспортери (APC), артилерійські системи та системи протиповітряної оборони. Загальний обсяг набору даних склав 43845 зображень і 66487 анотованих об’єктів. Результати оцінки навченої моделі на тестовій вибірці є достатньо високими, наприклад загальний показник mAP@0.5 становить 0.8454.

У третьому розділі було розроблено вебзастосунок за допомогою бібліотеки Streamlit для використання навченої моделі. Застосунок надає можливості для використання моделі для різних вхідних даних (зображення, відеофайли, YouTube відео та RTSP-потоки).

Подальший розвиток роботи може включати розширення набору даних більш детальними класами військової техніки, вдосконалення точності роботи моделі, її оптимізацію для роботи на мобільних та вбудованих пристроях (наприклад безпілотних літальних системах) та інтеграцію з геоінформаційними системами (відображення виявлених об’єктів на карті місцевості).

Таким чином, результатом моєї роботи є вебзастосунок для розпізнавання військової техніки з використанням попередньо натренованої моделі. Розроблений застосунок може бути використаний для автоматизації процесів моніторингу та розвідки, підвищуючи швидкість обробки візуальної інформації. Висока точність моделі та практична цінність отриманих результатів підтверджують успішне виконання завдань та досягнення мети кваліфікаційної роботи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Fundamental Concepts of Convolutional Neural Network / Ghosh A. and etc. *Recent Trends and Advances in Artificial Intelligence and Internet of Things*. 2020. Vol. 172. P. 519-567. URL: https://link.springer.com/chapter/10.1007/978-3-030-32644-9_36 (дата звернення: 15.01.2025).

2. Rawat W., Wang Z. Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Computation*. 2017. Vol. 29. P. 2352-2449. URL: https://direct.mit.edu/neco/article-abstract/29/9/2352/8292/Deep-Convolutional-Neural-Networks-for-Image?redirectedFrom=fulltext (дата звернення: 18.01.2025).

3. Zou Z., Shi Z., Guo Y., Ye J. Object Detection in 20 Years: A Survey. *arXiv*. 2019. Vol. 1905.05055. P. 1-32. URL: https://arxiv.org/abs/1905.05055 (дата звернення: 12.02.2025).

4. Prabu S., Gnanasekar J. M. A Study on Image Segmentation Method for Image Processing. *Recent Trends in Intensive Computing*. 2021. Vol. 39. P. 419-424. URL: https://ebooks.iospress.nl/doi/10.3233/APC210223 (дата звернення: 25.02.2025).

5. Zdimalova M., Vrabel M., Fedorin M., Toth N. Image Processing in Military Security and Defense Problems. *National and International Security: 15th International Scientific Conference.* 2024. P. 344-358. URL: https://aos.sk/www/data/uploads/files/Katedry/KBO/NMB/zbornik-nmb-2024.pdf (дата звернення: 14.03.2025).

6. Relief R-CNN: Utilizing Convolutional Features for Fast Object Detection / Li G. and etc. *Advances in Neural Networks – ISNN 2017.* 2017. Vol. 10261. P. 386-394. URL: https://link.springer.com/chapter/10.1007/978-3-319-59072-1_46 (дата звернення: 29.03.2025).

7. 2D Object Detection with Transformers: A Review / Shehzadi T. and etc. *arXiv*. 2023. Vol. 2306.04670. P. 1-26. URL: https://arxiv.org/abs/2306.04670 (дата звернення: 05.04.2025).

8. End-to-End Object Detection with Transformers / Carion N. and etc. *Computer Vision – ECCV 2020*. 2020. Vol. 12346. P. 213-229. URL: https://link.springer.com/chapter/10.1007/978-3-030-58452-8_13 (дата звернення: 16.04.2025).

9. Lightweight Object Detection Algorithm for UAV Aerial Imagery / Wang J. and etc. *Sensors*. 2023. Vol. 23. P. 1-21. URL: https://www.mdpi.com/1424-8220/23/13/5786 (дата звернення: 24.04.2025).

10. Tan M., Pang R., Le Q. EfficientDet: Scalable and Efficient Object Detection. *arXiv*. 2019. Vol. 1911.09070. P. 1-10. URL: https://arxiv.org/abs/1911.09070 (дата звернення: 02.05.2025).

11. Katkade S., Manza R., Shinde P., Balande B. Advances in Object Detection Algorithms Using Deep Learning. *GRENZE International Journal of Engineering and Technology*. 2025. Vol. 11. P. 4131-4138. URL: https://thegrenze.com/index.php?display=page&view=journalabstract&absid=4128&id=8 (дата звернення: 11.05.2025).

12. Roboflow – офіційна документація. URL: https://docs.roboflow.com/ (дата звернення: 13.05.2025).

13. Streamlit – офіційна документація. URL: https://docs.streamlit.io/ (дата звернення: 14.05.2025).

14. Automatic localization of casting defects with convolutional neural networks / Ferguson M. and etc. *Big Data* : International Conference on Big Data. 2017. P. 1726-1735. URL: https://ieeexplore.ieee.org/document/8258115 (дата звернення: 20.05.2025).

15. Що таке машинне навчання? *Oksim.ua*. URL: https://www.oksim.ua/2023/07/29/shho-take-mashynne-navchannya/ (дата звернення: 20.05.2025).

16. Pedestrian detection in highly crowded scenes using “online” dictionary learning for occlusion handling / Wang B. and etc. *ICIP* : International Conference on Image Processing. 2014. P. 2418-2422. URL: https://ieeexplore.ieee.org/document/7025489 (дата звернення: 20.05.2025).

17. Towards Real-Time Smile Detection Based on Faster Region Convolutional Neural Network / Nguyen C. C. and etc. *MAPR* : 1st International Conference on Multimedia Analysis and Pattern Recognition. 2018. P. 1-6. URL: https://ieeexplore.ieee.org/document/8337524 (дата звернення: 20.05.2025).

18. Understanding “DETR: End-to-End Object Detection with Transformers”. URL: https://medium.com/@shashankag14/understanding-detr-end-to-end-object-detection-with-transformers-ea7e845fad4f (дата звернення: 20.05.2025).

19. Jeghama N., Koh C. Y., Abdelatti M., Hendawi A. YOLO Evolution: A Comprehensive Benchmark and Architectural Review of YOLOv12, YOLO11, and Their Previous Versions. *arXiv*. 2024. Vol. 2411.00201. P. 1-33. URL: https://arxiv.org/abs/2411.00201 (дата звернення: 20.05.2025).

20. Khan A.T., Jensen S.M. LEAF-Net: A Unified Framework for Leaf Extraction and Analysis in Multi-Crop Phenotyping Using YOLOv11. *Agriculture 2025*. 2025. Vol. 15. P. 1-17. URL: https://www.mdpi.com/2077-0472/15/2/196 (дата звернення: 20.05.2025).

ДОДАТОК А

Функції формування набору даних

А.1 Process_roboflow_dataset та пов’язані функції

```
def download_and_extract(url: str, filename: str, save_dir: Path):
    """
    Завантажує .tar або .zip файл та розпаковує його в задану директорію.

    Args:
        url (str): URL файлу
        filename (str): ім'я файлу для збереження
        save_dir (Path): директорія призначення
    """
    save_dir.mkdir(exist_ok=True)
    dest_file = save_dir / filename

    if download_file(url, dest_file):
        if dest_file.suffix == ".zip":
            with zipfile.ZipFile(dest_file, "r") as zip:
                zip.extractall(save_dir)
        else:
            with tarfile.open(dest_file) as tf:
                tf.extractall(save_dir)

        dest_file.unlink()
        logger.info(f"Розпаковано у {save_dir}")

def restructure_dataset(roboflow_dir: Path, subsets=None):
    """
    Реструктуризує датасет, копіюючи jpg зображення в директорію 'data',
    а xml файли в директорію 'labels' з вихідної директорії.

    Args:
        roboflow_dir (Path): Вихідна директорія, що містить датасет.
        subsets (list, optional): Список підмножин для включення (наприклад, ["train"] або
["train", "valid"]).
                                  За замовчуванням ["test", "train", "valid"].
    """
    if subsets is None:
        subsets = ["test", "train", "valid"]
```

```
    # Створюємо цільові піддиректорії, якщо вони не існують
    (roboflow_dir / "data").mkdir(parents=True, exist_ok=True)
    (roboflow_dir / "labels").mkdir(parents=True, exist_ok=True)

    # Копіюємо jpg та xml файли з вказаних підмножин до цільових директорій
    for subset in subsets:
        source_path = roboflow_dir / subset

        if source_path.exists():
            # Копіюємо jpg файли до 'data'
            for jpg_file in source_path.glob("*.jpg"):
                shutil.copy(jpg_file, roboflow_dir / "data")

            # Копіюємо xml файли до 'labels'
            for xml_file in source_path.glob("*.xml"):
                shutil.copy(xml_file, roboflow_dir / "labels")

            # Видаляємо оригінальну піддиректорію
            shutil.rmtree(source_path)

    logger.info(f"Директорія датасету успішно реструктуризована для підмножин: {subsets}.")

def check_available_classes(dataset_rf):
    available_classes = set()

    for sample in dataset_rf:
        labels = sample.ground_truth.detections
        for label in labels:
            available_classes.add(label.label)

    return available_classes

def count_classes(dataset):
    """
    Підрахунок кількості зображень для кожного класу у датасеті.

    Args:
        dataset: Датасет FiftyOne

    Returns:
        None: Виводить на екран кількість зображень для кожного класу
    """
    # Ініціалізуємо словник для підрахунку класів
    class_counts = {}
```

```
    # Проходимо через всі зразки в об'єднаному датасеті
    for sample in dataset:
        # Отримуємо унікальні класи (мітки) у цьому зразку
        labels = set([detection.label for detection in sample.ground_truth.detections])

        # Оновлюємо лічильник для кожного класу
        for label in labels:
            if label in class_counts:
                class_counts[label] += 1
            else:
                class_counts[label] = 1

    # Виводимо класи та кількість зображень, що містять кожен клас
    print("Кількість класів після об'єднання датасетів:")
    for label, count in class_counts.items():
        print(f"{label}: {count} зображень")

def process_roboflow_dataset(roboflow_url, roboflow_dir, label_mapping, subsets=None, delete_empty=True, target_dataset=None):
    """
    Повний процес обробки датасету Roboflow: завантаження, реструктуризація, перетворення міток, фільтрація і злиття.

Args:
        roboflow_url (str): URL для завантаження датасету Roboflow
        roboflow_dir (Path): Директорія для збереження датасету
        label_mapping (dict): Відображення міток з оригінального датасету у потрібний формат
        target_dataset: Цільовий датасет FiftyOne, до якого буде додано зразки. Якщо None, буде повернуто новий датасет.
        subsets (list, optional): Список підмножин для включення
        delete_empty (bool, optional): Чи видаляти зображення без міток

    Returns:
        fo.Dataset: Датасет FiftyOne з обробленими зразками
    """
    # Завантаження та розпакування датасету
    download_and_extract(roboflow_url, "dataset_rf.zip", roboflow_dir)
    restructure_dataset(roboflow_dir, subsets=subsets)

    # Імпорт датасету Roboflow
    dataset_rf = fo.Dataset.from_dir(
        dataset_dir=roboflow_dir,
        dataset_type=fo.types.VOCDetectionDataset,
    )

    available_classes = check_available_classes(dataset_rf)
```

```
print("Доступні класи перед обробкою:", available_classes)
dataset_rf.map_labels(
    "ground_truth",
    label_mapping
).save()

# Дозволені мітки
allowed_labels = ["Air-Defense", "Artillery", "AFV", "APC"]

# Зберігаємо інформацію які зразки були спочатку порожніми
initially_empty_samples = []
for sample in dataset_rf:
    if not sample.ground_truth.detections:
        initially_empty_samples.append(sample.id)

# Фільтруємо мітки та видаляємо зразки, які втратили всі мітки після фільтрації
samples_to_delete = []

for sample in dataset_rf:
    original_labels = sample.ground_truth.detections
    filtered_labels = [label for label in original_labels if label.label in allowed_labels]

    if original_labels and not filtered_labels:
        samples_to_delete.append(sample.id)
    else:
        sample["ground_truth"].detections = filtered_labels
        sample.save()

# Видаляємо зразки, які втратили всі мітки після фільтрації
if samples_to_delete:
    dataset_rf.delete_samples(samples_to_delete)

if delete_empty:
    remaining_empty = [sample_id for sample_id in initially_empty_samples
                       if sample_id in dataset_rf.values("id")]

    if remaining_empty:
        dataset_rf.delete_samples(remaining_empty)

dataset_rf.save()

if target_dataset is not None:
    target_dataset.merge_samples(dataset_rf)
    result_dataset = target_dataset
else:
    result_dataset = dataset_rf
```

```
    # Підрахунок класів у датасеті
    count_classes(result_dataset)

    return result_dataset
```

A.2 Analyze_dataset_classes

```
def analyze_dataset_classes(dataset, label_field="ground_truth"):
    """
    Аналізує класовий розподіл у датасеті.

    Args:
        dataset (fo.Dataset): Датасет FiftyOne для аналізу
        label_field (str): Назва поля з мітками у датасеті

    Returns:
        dict: Словник з кількістю зразків та об'єктів для кожного класу
    """

    samples_with_class = collections.defaultdict(set)
    objects_per_class = collections.Counter()

    for sample in dataset:
        for detection in sample[label_field].detections:
            label = detection.label
            samples_with_class[label].add(sample.id)
            objects_per_class[label] += 1

    print("\n=== Dataset Class Distribution ===")
    print(f"Total samples: {len(dataset)}")

    print("\nSamples per class:")
    for label, sample_ids in sorted(samples_with_class.items()):
        print(f"  {label}: {len(sample_ids)} samples ({len(sample_ids)/len(dataset)*100:.1f}%)")

    print("\nObjects per class:")
    total_objects = sum(objects_per_class.values())
    for label, count in sorted(objects_per_class.items()):
        print(f"  {label}: {count} objects ({count/total_objects*100:.1f}%)")

    return {
        "samples_with_class": {k: len(v) for k, v in samples_with_class.items()},
```

```
        "objects_per_class": dict(objects_per_class)
    }
```

A.3 Split_and_export_to_yolo

```
def split_and_export_to_yolo(
    dataset,
    export_dir,
    classes,
    split_ratios={"train": 0.8, "val": 0.1, "test": 0.1},
    rare_classes_split_ratios=None,  # Новий параметр для окремого розподілу рідкісних класів
    label_field="ground_truth",
    overwrite=False
):
    """
    Виконує стратифікований поділ датасету на основі розподілу класів
    та експортує у формат YOLO. Забезпечує пропорційне представлення
    кожного класу у вибірках train, validation та test, з можливістю
    спеціального розподілу для рідкісних класів.

    Args:
        dataset (fo.Dataset): FiftyOne датасет для експорту
        export_dir (Path): Директорія для експорту
        classes (List[str]): Список класів для експорту
        split_ratios (Dict[str, float]): Співвідношення для поділу train/val/test
        rare_classes_split_ratios (Dict[str, Dict[str, float]]): Словник з окремими
            співвідношеннями для рідкісних класів, наприклад:
            {"Air-Defense": {"train": 0.9, "val": 0.07, "test": 0.03}}
        label_field (str): Назва поля, що містить мітки
        overwrite (bool): Чи перезаписувати існуючу директорію експорту
    """
    export_dir = Path(export_dir)
    if export_dir.exists() and overwrite:
        logger.info(f"Видалення існуючої директорії експорту: {export_dir}")
        shutil.rmtree(export_dir)

    export_dir.mkdir(exist_ok=True, parents=True)

    if rare_classes_split_ratios is None:
        rare_classes_split_ratios = {}
```

```
    for class_name, ratios in rare_classes_split_ratios.items():
        if sum(ratios.values()) != 1.0:
            logger.warning(f"Сума співвідношень для класу {class_name} не дорівнює 1.0. Масштабуємо значення.")
            total = sum(ratios.values())
            rare_classes_split_ratios[class_name] = {k: v/total for k, v in ratios.items()}

    dataset.untag_samples(dataset.distinct("tags"))

    # ЕТАП 1: Ідентифікація нерозмічених зразків
    unlabeled_samples = []
    labeled_samples = []

    class_specific_samples = {class_name: [] for class_name in classes}

    for sample in dataset:
        has_valid_labels = False
        if hasattr(sample, label_field) and sample[label_field] is not None:
            detections = sample[label_field].detections
            if detections:
                # Перевіряємо кожну детекцію
                for det in detections:
                    if det.label in classes:
                        has_valid_labels = True
                        # Додаємо зразок до відповідного списку класу
                        class_specific_samples[det.label].append(sample.id)

                if has_valid_labels:
                    labeled_samples.append(sample.id)

        if not has_valid_labels:
            unlabeled_samples.append(sample.id)

    for class_name in classes:
        class_specific_samples[class_name] = list(set(class_specific_samples[class_name]))

    logger.info(f"Знайдено {len(unlabeled_samples)} зразків без міток")
    logger.info(f"Знайдено {len(labeled_samples)} зразків з мітками")

    for class_name in classes:
        logger.info(f"Клас {class_name}: {len(class_specific_samples[class_name])} зразків")

    # ЕТАП 2: Групування розмічених зразків за їх класовою композицією
    class_sample_map = collections.defaultdict(list)

    for sample_id in labeled_samples:
```

```
    sample = dataset[sample_id]
    sample_classes = set()
    for detection in sample[label_field].detections:
        if detection.label in classes:
            sample_classes.add(detection.label)

    if not sample_classes:
        continue

    class_key = frozenset(sample_classes)
    class_sample_map[class_key].append(sample.id)

logger.info("Розподіл зразків за комбінаціями класів:")
for class_key, sample_ids in class_sample_map.items():
    class_str = ", ".join(sorted(class_key))
    logger.info(f"  {class_str}: {len(sample_ids)} зразків")

# ЕТАП 3: Застосування стратифікованого поділу для розмічених зразків
split_keys = list(split_ratios.keys())

for class_key, sample_ids in class_sample_map.items():
    contains_rare_class = False
    current_rare_class = None

    for rare_class in rare_classes_split_ratios:
        if rare_class in class_key:
            contains_rare_class = True
            current_rare_class = rare_class
            break

    if contains_rare_class and len(class_key) == 1 and current_rare_class in class_key:
        specific_ratios = rare_classes_split_ratios[current_rare_class]

        np.random.shuffle(sample_ids)

        n_samples = len(sample_ids)
        indices = [0]
        cumulative = 0

        specific_split_keys = list(specific_ratios.keys())

        for i in range(len(specific_ratios) - 1):
            cumulative += specific_ratios[specific_split_keys[i]]
            indices.append(int(n_samples * cumulative))
        indices.append(n_samples)
```

```
            for i, split_key in enumerate(specific_split_keys):
                if indices[i] == indices[i+1]:
                    continue

                split_sample_ids = sample_ids[indices[i]:indices[i+1]]
                logger.info(f"  Маркування {len(split_sample_ids)} зразків як '{split_key}' для класу {current_rare_class} (рідкісний клас, спеціальний розподіл)")

                dataset.select(split_sample_ids).tag_samples(split_key)
        else:
            np.random.shuffle(sample_ids)

            n_samples = len(sample_ids)
            indices = [0]
            cumulative = 0

            for i in range(len(split_ratios) - 1):
                cumulative += split_ratios[split_keys[i]]
                indices.append(int(n_samples * cumulative))
            indices.append(n_samples)

            for i, split_key in enumerate(split_keys):
                if indices[i] == indices[i+1]:
                    continue

                split_sample_ids = sample_ids[indices[i]:indices[i+1]]
                logger.info(f"  Маркування {len(split_sample_ids)} зразків як '{split_key}' для класів {', '.join(sorted(class_key))}")

                dataset.select(split_sample_ids).tag_samples(split_key)

    # ЕТАП 4: Обробка нерозмічених зразків
    if unlabeled_samples:
        logger.info(f'Додавання {len(unlabeled_samples)} нерозмічених зразків до навчальної вибірки 'train'")
        dataset.select(unlabeled_samples).tag_samples("train")

    # ЕТАП 5: Експорт кожної частини у формат YOLO
    for split_key in split_ratios.keys():
        split_view = dataset.match_tags(split_key)
        if len(split_view) == 0:
            logger.warning(f'Не знайдено зразків для частини '{split_key}', пропускаємо експорт")
            continue

        logger.info(f'Експорт {len(split_view)} зразків для частини '{split_key}'")
```

```
        if split_key == "train":
            labeled_in_split = 0
            unlabeled_in_split = 0
            for sample in split_view:
                if hasattr(sample, label_field) and sample[label_field] is not None:
                    if sample[label_field].detections and any(det.label in classes for det in sample[label_field].detections):
                        labeled_in_split += 1
                    else:
                        unlabeled_in_split += 1
                else:
                    unlabeled_in_split += 1

            logger.info(f"  У вибірці '{split_key}': {labeled_in_split} розмічених та {unlabeled_in_split} нерозмічених зразків")

        # Експорт у формат YOLO
        split_view.export(
            export_dir=str(export_dir),
            dataset_type=fo.types.YOLOv5Dataset,
            label_field=label_field,
            classes=classes,
            split=split_key
        )

    # ЕТАП 6: Створення файлу конфігурації для YOLOv11
    data_yaml_content = f"""
# YOLOv11 dataset config
path: {export_dir}
train: train/images
val: val/images
test: test/images

nc: {len(classes)}
names: {classes}
    """

    with open(export_dir / 'data.yaml', 'w') as f:
        f.write(data_yaml_content.strip())

    logger.info(f"Експорт завершено. Датасет експортовано до {export_dir} у формат YOLOv5/YOLOv11")
    logger.info(f"Створено файл data.yaml з {len(classes)} класами")

    # ЕТАП 7: Аналіз статистики розподілу
    for split_key in split_ratios.keys():
```

```
split_view = dataset.match_tags(split_key)
if len(split_view) > 0:
    logger.info(f"Частина '{split_key}' містить {len(split_view)} зразків")

    class_counts = collections.Counter()
    for sample in split_view:
        if hasattr(sample, label_field) and sample[label_field] is not None:
            for detection in sample[label_field].detections:
                if detection.label in classes:
                    class_counts[detection.label] += 1

    logger.info(f"  Розподіл класів у частині '{split_key}':")
    for class_name in classes:
        count = class_counts.get(class_name, 0)
        logger.info(f"    {class_name}: {count} екземплярів")
```

ДОДАТОК Б

Додаток Streamlit

Б.1 Код файлу app.py

```
from pathlib import Path
import PIL
import streamlit as st
import config
import worker
import cv2
import numpy as np
from PIL import Image

# Налаштування сторінки
st.set_page_config(
    page_title="Military Equipment Detection",
    page_icon="📡",
    layout="wide",
    initial_sidebar_state="expanded"
)

st.title("Military Equipment Detection 🔍")

# Бічна панель
st.sidebar.header("⚙️ Налаштування моделі")

confidence = st.sidebar.slider("Виберіть рівень впевненості", 0.1, 1.0, 0.4)

# Завантаження моделі
model_path = Path(config.DETECTION_MODEL)
try:
    model = worker.load_model(model_path)
except Exception as ex:
    st.error(f"❌ Неможливо завантажити модель: {model_path}")
    st.error(ex)

st.sidebar.header("📞 Налаштування відео")
source_radio = st.sidebar.radio("Виберіть джерело", config.SOURCES_LIST)

if source_radio == config.IMAGE:
```

```
    worker.detect_on_image(confidence, model)

elif source_radio == config.VIDEO:
    worker.play_stored_video(confidence, model)

elif source_radio == config.RTSP:
    worker.play_rtsp_stream(confidence, model)

elif source_radio == config.YOUTUBE:
    worker.play_youtube_video(confidence, model)
else:
    st.error("⚠ Будь ласка, оберіть коректний тип джерела!")
```

Б.2 Код файлу worker.py

```
import streamlit as st
import cv2
from ultralytics import YOLO
import yt_dlp
import config
from pathlib import Path
import PIL
import streamlit as st
import config
import cv2
import numpy as np
from PIL import Image
import os
import requests
import subprocess
import shutil
import traceback
import re
import time
import tempfile


def load_model(model_path):
    """Завантажує модель YOLO."""
    return YOLO(model_path)
```

```
def detect_on_image(conf, model):
    """
    Виконує детекцію на зображеннях.
    """
    source_imgs = st.sidebar.file_uploader(
        "Завантаження зображень...",
        type=("jpg", "jpeg", "png"),
        accept_multiple_files=True
    )

    if source_imgs:
        for i, source_img in enumerate(source_imgs):
            col1, col2 = st.columns(2)

            with col1:
                original = Image.open(source_img)
                st.image(original, use_column_width=True)

            with col2:
                img_array = np.array(original)
                res = model.predict(img_array, conf=conf)
                annotated_img = res[0].plot()
                st.image(annotated_img, use_column_width=True)

            st.write("")
            with st.expander(f"Результати обробки для зображення {i+1}"):
                for i, box in enumerate(res[0].boxes):
                    data = box.data[0]
                    st.write(f"Об'єкт #{i+1}:")
                    st.write(f"- Клас: {data[5]}")
                    st.write(f"- Впевненість: {data[4]*100:.2f}%")
                    st.write(f"- Координати: x1={data[0]:.1f}, y1={data[1]:.1f}, x2={data[2]:.1f},
y2={data[3]:.1f}")
            st.write("")


def get_frames_and_detect(conf, model, source, tracker="bytetrack.yaml"):
    """
    Допоміжна функція: зчитує кадри із source та виконує детекцію об'єктів.
    Відображає результати детекції в реальному часі.
    Зберігає оброблене відео у стандартний MP4 файл для відео.
    """
    try:
        vid_cap = cv2.VideoCapture(source)
        st_frame = st.empty()
```

```
if not vid_cap.isOpened():
    st.error(" ✕ Не вдається відкрити потік/відео.")
    return None

fps = vid_cap.get(cv2.CAP_PROP_FPS)

if source.startswith('rtsp://'):
    while vid_cap.isOpened():
        success, frame = vid_cap.read()
        if not success:
            break

        if tracker:
            res = model.track(frame, conf=conf, tracker=tracker)
        else:
            res = model.predict(frame, conf=conf, stream=True)

        processed_frame = res[0].plot()

        st_frame.image(
            processed_frame,
            caption="Processing...",
            use_column_width=True,
            channels="BGR"
        )

    vid_cap.release()
    clean_temp_files()
    return None

temp_frames_dir = "temp_frames"
os.makedirs(temp_frames_dir, exist_ok=True)

frame_count = int(vid_cap.get(cv2.CAP_PROP_FRAME_COUNT))
progress_bar = st.progress(0)
frame_idx = 0

while vid_cap.isOpened():
    success, frame = vid_cap.read()
    if not success:
        break

    if frame_count > 0:
        progress_bar.progress(min(frame_idx / frame_count, 1.0))
```

```
        if tracker:
            res = model.track(frame, conf=conf, tracker=tracker)
        else:
            res = model.predict(frame, conf=conf, stream=True)

        processed_frame = res[0].plot()

        st_frame.image(
            processed_frame,
            caption="Processing...",
            use_column_width=True,
            channels="BGR"
        )

        frame_path = os.path.join(temp_frames_dir, f"frame_{frame_idx:06d}.jpg")
        cv2.imwrite(frame_path, processed_frame)

        frame_idx += 1

    vid_cap.release()

    videos_dir = "videos"
    os.makedirs(videos_dir, exist_ok=True)
    output_path = os.path.join(videos_dir, f"processed_{Path(source).stem}.mp4")

    try:
        cmd = [
            "ffmpeg", "-y",
            "-framerate", str(fps),
            "-i", os.path.join(temp_frames_dir, "frame_%06d.jpg"),
            "-c:v", "libx264",
            "-profile:v", "main",
            "-preset", "medium",
            "-r", str(fps),
            "-tune", "zerolatency",
            "-crf", "23",
            "-pix_fmt", "yuv420p",
            "-movflags", "+faststart",
            output_path
        ]

        save_msg = st.empty()
        save_msg.info("⏳ Збереження відео...")
        process = subprocess.Popen(cmd, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        stdout, stderr = process.communicate()
```

```
            if process.returncode != 0:
                st.error(f"❌ Помилка при створенні відео: {stderr.decode()}")
                return None
        except Exception as e:
            st.error(f"❌ Помилка при створенні відео: {str(e)}")
            return None
        finally:
            for file in os.listdir(temp_frames_dir):
                file_path = os.path.join(temp_frames_dir, file)
                try:
                    if os.path.isfile(file_path):
                        os.unlink(file_path)
                except Exception as e:
                    st.warning(f"Не вдалося видалити тимчасовий файл {file_path}: {str(e)}")

            if os.path.exists(temp_frames_dir) and not os.listdir(temp_frames_dir):
                os.rmdir(temp_frames_dir)

        progress_bar.empty()
        st_frame.empty()
        save_msg.empty()

        clean_temp_files()

        return output_path

    except Exception as e:
        st.error(f"Помилка обробки відео: {str(e)}")
        clean_temp_files()
        return None


def play_stored_video(conf, model, tracker="bytetrack.yaml"):
    """
    Функція для обробки та відтворення відео:
    1. Користувач вибирає відео зі списку або завантажує своє
    2. Після натискання кнопки "Запуск детекції" відео обробляється та показується в реальному часі з детекцією
    3. Після закінчення обробки, показується оброблене відео у звичайному форматі та надається можливість завантажити його

    Args:
        conf: Рівень впевненості
        model: Завантажена модель YOLO
        tracker: Конфігурація трекера (default: "bytetrack.yaml")
```

```
"""
st.title("📹 Обробка відео")

# Створюємо тимчасову директорію для збереження кадрів
temp_dir = "temp_frames"
if not os.path.exists(temp_dir):
    os.makedirs(temp_dir)

video_option = st.sidebar.radio(
    "Виберіть джерело відео",
    ("Вибрати зі списку", "Завантажити відео")
)

if video_option == "Вибрати зі списку":
    source_vid = st.sidebar.selectbox(
        "Виберіть відео...",
        list(config.VIDEOS_DICT.keys())
    )
    video_path = str(config.VIDEOS_DICT[source_vid])
else:
    uploaded_file = st.sidebar.file_uploader("Оберіть відео файл", type=["mp4", "avi", "mov"])
    if uploaded_file is not None:
        temp_dir = "temp_uploads"
        os.makedirs(temp_dir, exist_ok=True)
        video_path = os.path.join(temp_dir, uploaded_file.name)
        with open(video_path, "wb") as f:
            f.write(uploaded_file.getbuffer())
    else:
        st.warning("Будь ласка, завантажте відео файл")
        return

video_container = st.empty()

if st.sidebar.button("Запуск детекції 🎯"):
    try:
        processed_video_path = get_frames_and_detect(conf, model, video_path, tracker)

        if processed_video_path and os.path.exists(processed_video_path):
            file_size = os.path.getsize(processed_video_path) / (1024 * 1024)
            st.success(f"✅ Файл сформовано. Розмір файлу: {file_size:.2f} MB")

            video_container.video(processed_video_path)

            with open(processed_video_path, "rb") as file:
                video_bytes = file.read()
```

```
                st.download_button(
                    label="📥 Завантажити оброблене відео",
                    data=video_bytes,
                    file_name=Path(processed_video_path).name,
                    mime="video/mp4"
                )

            if st.button("Очистити відео"):
                video_container.empty()
                clean_temp_files()
        except Exception as e:
            st.error(f"Помилка: {str(e)}")
            clean_temp_files()


def play_youtube_video(conf, model, tracker="bytetrack.yaml"):
    """
    Відтворення YouTube-відео за посиланням у реальному часі.
    Детекція + трекінг на кожному кадрі.

    Args:
        conf: Рівень впевненості
        model: Завантажена модель YOLO
        tracker: Конфігурація трекера (default: "bytetrack.yaml")
    """
    youtube_url = st.sidebar.text_input("YouTube Video URL",
"https://www.youtube.com/watch?v=FQijTjkH7-0")

    with st.sidebar.expander("Додаткові налаштування"):
        debug_mode = st.checkbox("Режим діагностики", value=False)

    video_container = st.empty()

    if st.sidebar.button("Обробити відео 🎬"):
        if not youtube_url:
            st.error("❌ Введіть посилання на YouTube-відео.")
            return

        try:
            videos_dir = "videos"
            os.makedirs(videos_dir, exist_ok=True)
            video_id = extract_youtube_id(youtube_url)
            output_path = os.path.join(videos_dir, f"youtube_{video_id}.mp4")

            connection_msg = st.empty()
```

```
            connection_msg.info("Отримання відео з YouTube...")

            stream_url = get_youtube_stream_url(youtube_url)

            connection_msg.success("✅ Відео отримано. Обробка...")

            temp_file = download_youtube_to_temp(stream_url)

            processed_video_path = get_frames_and_detect(conf, model, temp_file, tracker)

            if processed_video_path and os.path.exists(processed_video_path):
                shutil.copy(processed_video_path, output_path)
                processed_video_path = output_path

            connection_msg.empty()

            if processed_video_path and os.path.exists(processed_video_path):
                file_size = os.path.getsize(processed_video_path) / (1024 * 1024)
                st.success(f"✅ Файл сформовано. Розмір файлу: {file_size:.2f} MB")

                video_container.video(processed_video_path)

                with open(processed_video_path, "rb") as file:
                    video_bytes = file.read()
                    st.download_button(
                        label="📥 Завантажити оброблене відео",
                        data=video_bytes,
                        file_name=f"youtube_video_{video_id}.mp4",
                        mime="video/mp4"
                    )

                clean_temp_files()

                if st.button("Очистити відео"):
                    video_container.empty()
                    clean_temp_files()

        except Exception as e:
            st.error(f"Помилка: {str(e)}")
            if debug_mode:
                st.code(traceback.format_exc())
            clean_temp_files()


def extract_youtube_id(youtube_url):
```

```
    """
    Витягує ID відео з YouTube URL.
    """
    patterns = [
        r'(?:v=|\/)([0-9A-Za-z_-]{11}).*',
        r'(?:embed\/|v\/|youtu\.be\/)([0-9A-Za-z_-]{11})',
        r'(?:watch\?v=)([0-9A-Za-z_-]{11})'
    ]

    for pattern in patterns:
        match = re.search(pattern, youtube_url)
        if match:
            return match.group(1)

    return f"video_{int(time.time())}"


def download_youtube_to_temp(stream_url):
    """
    Завантажує відео з потоку у тимчасовий файл і повертає шлях до нього.
    """

    temp_dir = "temp_youtube"
    os.makedirs(temp_dir, exist_ok=True)
    temp_file = os.path.join(temp_dir, f"temp_video_{int(time.time())}.mp4")

    try:
        response = requests.get(stream_url, stream=True)
        response.raise_for_status()

        with open(temp_file, 'wb') as f:
            for chunk in response.iter_content(chunk_size=8192):
                f.write(chunk)

        return temp_file
    except Exception as e:
        try:
            ydl_opts = {
                'format': 'bestvideo[height<=720][vcodec!*=av01]+bestaudio/best[height<=720]',
                'outtmpl': temp_file,
                'quiet': True
            }
            with yt_dlp.YoutubeDL(ydl_opts) as ydl:
                stream_info = {'url': stream_url}
                ydl.process_ie_result(stream_info, download=True)
```

```
            return temp_file
        except Exception:
            raise Exception(f"Не вдалося завантажити відео: {str(e)}")


def clean_temp_files():
    """
    Очищає всі тимчасові файли.
    """
    temp_dirs = ["temp_youtube", "temp_frames", "temp_uploads"]
    for temp_dir in temp_dirs:
        if os.path.exists(temp_dir):
            try:
                for file in os.listdir(temp_dir):
                    file_path = os.path.join(temp_dir, file)
                    if os.path.isfile(file_path):
                        os.unlink(file_path)

                if not os.listdir(temp_dir):
                    os.rmdir(temp_dir)
            except Exception as e:
                print(f"Помилка при очистці {temp_dir}: {str(e)}")
                continue


def get_youtube_stream_url(youtube_url):
    """
    За допомогою yt_dlp витягує пряме посилання на відео-потік YouTube.
    Уникає формати AV1, які можуть викликати проблеми з декодуванням.
    """
    if not youtube_url:
        raise ValueError("URL відео не може бути порожнім")

    ydl_opts = {
        'format':
'bestvideo[ext=mp4][vcodec!*=av01][vcodec^=avc]+bestaudio[ext=m4a]/best[ext=mp4][vcodec
!*=av01]/best[vcodec!*=av01]/best',
        'quiet': True,
        'no_warnings': True
    }

    try:
        with yt_dlp.YoutubeDL(ydl_opts) as ydl:
            info = ydl.extract_info(youtube_url, download=False)

            if 'url' in info:
```

```
            return info['url']

        if 'requested_formats' in info:
            for fmt in info['requested_formats']:
                if fmt.get('vcodec', '').startswith('avc'):
                    return fmt['url']

        if 'formats' in info:
            for fmt in info['formats']:
                vcodec = fmt.get('vcodec', 'none')
                if (vcodec != 'none' and 'av01' not in vcodec and
                    fmt.get('height', 0) >= 360):
                    return fmt['url']

        if 'formats' in info and info['formats']:
            for fmt in sorted(info['formats'], key=lambda x: x.get('height', 0), reverse=True):
                if fmt.get('vcodec', 'none') != 'none' and fmt.get('height', 0) > 0:
                    return fmt['url']

        raise ValueError("Не вдалося отримати відповідний URL потоку")

    except Exception as e:
        raise Exception(f"Помилка при обробці відео: {str(e)}")


def play_rtsp_stream(conf, model, tracker="bytetrack.yaml"):
    """
    Відтворення RTSP стріму: користувач вводить URL,
    далі кожен кадр обробляється та показується.

    Args:
        conf: Рівень впевненості
        model: Завантажена модель YOLO
        tracker: Конфігурація трекера (default: "bytetrack.yaml")
    """
    source_rtsp = st.sidebar.text_input("RTSP stream URL:")
    st.sidebar.caption("Приклад:
rtsp://rtspstream:NuNGxzjfxj6QeLHwbJ9us@zephyr.rtsp.stream/traffic")

    if st.sidebar.button("Start RTSP 🚀"):
        if not source_rtsp:
            st.error("❌ Будь ласка, введіть коректну RTSP-адресу.")
            return
        get_frames_and_detect(conf, model, source_rtsp, tracker)
```