МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ “КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ”
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Юрій ІСКРЕНКО
“_____”_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ “МАГІСТР”
ЗА СПЕЦІАЛЬНІСТЮ 123 “КОМП'ЮТЕРНА ІНЖЕНЕРІЯ”

Тема: Автоматизована комп’ютерна система розпізнавання об’єктів з бортової камери БПЛА

Виконавець: Андрій ПІД’ЯПОЛЬСЬКИЙ

Керівник: Юрій ІСКРЕНКО

Нормоконтролер: Наталія ФОМІНА

Київ 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ “КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ”
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

ЗАТВЕРДЖУЮ
Завідувач кафедри КСМ
_____ Юрій ІСКРЕНКО
“____” _____2025 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Під’япольського Андрія Вікторовича
(прізвище ім’я по батькові здобувача вищої освіти в родовому відмінку)

1. Тема роботи: “Автоматизована комп’ютерна система розпізнавання об’єктів з бортової камери БПЛА”
Затверджена наказом ректора від “ 24 ” _жовтня 2025 р. №2344/ст
2. Термін виконання роботи: з _29.09.2025_ по _31.12.2025
3. Вихідні дані до роботи: теоретичне дослідження та реалізації автоматизованої комп’ютерної системи розпізнавання об’єктів з бортової камери БПЛА.
4. Зміст пояснювальної записки: вступ, теоретичні основи та сфери застосування систем розпізнавання об’єктів з бортової камери БПЛА, методи та технології комп’ютерного зору й глибинного навчання, підготовка набору даних та навчання нейромережевих моделей, розробка автоматизованої системи розпізнавання, оцінка якості та швидкодії системи, висновки.
5. Перелік обов'язкового графічного (ілюстративного) матеріалу: результати роботи представлені в презентації.

6. Календарний план

№ п/п	Завдання	Термін виконання етапів	Підпис керівника
1.	Затвердження теми роботи	29.09.2025	
2.	Зібрати та проаналізувати науково-технічну літературу за темою	30.09.2025	
3.	Систематизувати теоретичний матеріал	12.10.2025	
4.	Проаналізувати наявні технології пов'язані з темою дослідження	19.10.2025	
5.	Підготувати набори даних та провести навчання моделей	30.10.2025	
6.	Розробка автоматизованої системи розпізнавання об'єктів з бортової камери БПЛА	14.11.2025	
7.	Провести тестування системи	30.11.2025	
8.	Оформити пояснювальну записку	13.12.2025	
9.	Проходження нормоконтролю та розробка тексту доповіді	18.12.2025	
10.	Отримання відгуку керівника та рецензії	22.12.2025	
11.	Захист кваліфікаційної роботи	23.12.2025–31.12.2025	

7. Дата видачі завдання: “29” __вересня__ 2025 р.

Керівник кваліфікаційної роботи: Юрій ІСКРЕНКО _____
(підпис керівника)

Завдання прийняв до виконання: Андрій ПІД’ЯПОЛЬСЬКИЙ _____
(підпис виконавця)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи “Автоматизована комп’ютерна система розпізнавання об’єктів з бортової камери БПЛА”: 100 сторінок, 37 рисунків, 6 таблиць, 36 інформаційних джерел.

Об’єкт дослідження – процес автоматизованого розпізнавання об’єктів з зображення та відеопотоку отриманих з камери БПЛА.

Предметом дослідження – автоматизована комп’ютерна мережа розпізнавання об’єктів з камери БПЛА.

Мета роботи – розробка автоматизованої комп’ютерної системи розпізнавання об’єктів з камери БПЛА.

Технічні та програмні засоби – мова програмування *Python* та набір бібліотек для комп’ютерного зору і глибинного навчання: *Ultralytics YOLOv*8, *OpenCV*, *NumPy*; сучасні методи розробки вебдодатків: *HTML, CSS, JavaScript.*

Основні характеристики та показники – система містить серверний модуль детекції з двома моделями YOLOv8, веб-інтерфейс оператора, модуль обробки відео з формуванням анотованих матеріалів.

Отримані результати та їх новизна – у роботі побудовано узагальнений *UAV*-датасет на основі власних кадрів з БПЛА та відкритих наборів даних, виконано навчання моделей *YOLOv*8 для умов аеророзвідки, розроблено веб-сервіс для взаємодії оператора з системою. Наукова новизна полягає в адаптації сучасних архітектур детекції до задачі виявлення дрібних військових цілей на складному фоні, а також у реалізації двомодельного підходу розпізнавання.

Рекомендації щодо використання результатів – розроблену систему можна використовувати для підтримки роботи операторів під час аналізу кадрів з БПЛА, а також як базу для подальшої інтеграції на борт БПЛА.

БПЛА, РОЗПІЗНАВАННЯ ОБ’ЄКТІВ, НЕЙРОННІ МЕРЕЖІ, *YOLOv*8, КОМП’ЮТЕРНИЙ ЗІР, *FASTAPI, PYTHON, OPENCV*

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

HCI – human and computer interaction

БПЛА – безпілотний літальний апарат

NMS – non-maximum suppression

YOLO – you only look once

SSD – single shot detector

RPN – region proposal network

ResNet – residual network

VGGNet – visual geometry group network

FPN – feature pyramid networks

MOT – multi object tracking

SORT – simple online and realtime tracking

RGB – red, green, blue

CCD-камера – *charge-coupled device camera*

LiDAR – light detection and ranging

NDVI – normalized difference vegetation index

VGG-16 – *visual geometry group* 16-*layer network*

R-CNN – regions with convolutional neural networks

RoI – region of interest

FPS – frames per second

mAP – mean average precision

ВСТУП

За сучасних умов територіальних конфліктів БПЛА стали ключовим засобом розвідки та моніторингу. Бортові камери БПЛА забезпечують безперервний потік відео на якому можна відслідковувати зміни території, появу важкої техніки та людей. Обсяги таких даних зростають, а час на їх опрацювання лишається обмеженим. При таких умовах переглядати відеоматеріали вручну являється потенційно небезпечною задачею. Оскільки оператор переглядає відео з низькою швидкістю і якраз через це вчасно не виявляються сторонні об'єкти.

Особливу популярність набувають автоматизовані системи розпізнавання об'єктів з бортових камер БПЛА. Відповідно до задач, відбувається ідентифікація різних об'єктів (людей, тварин, знаків, транспорту і тому подібне), але наразі з'являється гостра необхідність розпізнавати техніку, яка в змозі спричинити катастрофу. Сучасні методи глибинного навчання та комп'ютерного зору дозволяють створювати моделі, які можуть виявляти об'єкти за складних умов: дим, зміна кута кадру, рухаючи цілі, дрібні та замасковані об'єкти. Для зручного практичного використання необхідна інтеграція нейромережевих детекторів у зручні для оператора інформаційні системи з інтуїтивним веб-інтерфейсом.

Актуальність теми магістерської роботи “Автоматизована комп’ютерна система розпізнавання об’єктів з бортової камери БПЛА” зумовлена потребою підвищення оперативності й достовірності аналізу розвідувальних даних з безпілотних платформ. Автоматизація процесу виявлення техніки та інших військових об’єктів сприяє зменшенню навантаження на оператора, скороченню часу на реагування та підвищенню точності оцінювання територій. Окрім цього, дану систему можна використовувати у цивільних задачах таких, як: контроль транспортних потоків, наслідки катастроф, виявлення аварійних ситуацій та подібне.

Метою науково-дослідної роботи є розробка та експериментальна оцінка автоматизованої системи, яка забезпечує розпізнавання об’єктів на кадрах та

відеоматеріалах отриманих з бортової камери БПЛА за допомогою двох навчених моделей. Для досягнення поставленої мети необхідно розв’язати такі основні завдання:

1) проаналізувати сучасні підходи до розпізнавання об’єктів з камери БпЛА;

2) визначити вимоги до автоматизованої системи розпізнавання;

3) підготувати набори даних для навчання двох моделей;

4) навчити нейромережеву модель розпізнавати танки та інші військові об'єкти;

5) розробити модуль детектора;

6) реалізувати веб-сервіс та клієнтську частину для взаємодії оператора з системою;

7) провести експериментальну оцінку для двох моделей та порівняти результати.

Під час виконання роботи застосовувались методи комп’ютерного зору та глибинного навчання. За основу було взято модель сімейства *YOLOv*8, яка реалізована в середовищі *Python* з використанням бібліотек *Ultralytics, OpenCV* та *NumPy*. Для написання серверної логіки було використано фреймворк *FastAPI*, а для інтерактивного інтерфейсу основні веб-технології *HTML, CSS* та *JavaScript*.

Очікується, що результати магістерської роботи сприятимуть підвищенню ефективності застосування БПЛА для розвідки та моніторингу, а також дозволять зменшити час на аналіз кадрів низької якості.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ТА СФЕРИ ЗАСТОСУВАННЯ СИСТЕМИ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ З БОРТОВОЇ КАМЕРИ БПЛА

1.1 Розвиток та поняття комп'ютерного зору

Наразі комп'ютерний зір розвинувся у величезну галузь, що охоплює весь спектр — від збору необробленої інформації до вилучення образів та шаблонів. Він поєднує в собі концепції, методи та ідеї з цифрової обробки зображень, розпізнавання образів, штучного інтелекту та комп'ютерної графіки. Велика кількість завдань у сфері комп'ютерного зору пов'язані з процесом отримання інформації про випадки або описи з цифрових зображень та виділення певних ознак. Методи, що застосовуються для розв'язання задач комп'ютерного зору, залежать від області застосування та природи даних, які аналізуються.

Комп'ютерний зір є поєднанням обробки зображень і розпізнавання образів. Результатом процесу комп'ютерного зору є розуміння змісту зображення. Розвиток цієї галузі базується на спробі відтворити здатність людського зору сприймати та інтерпретувати інформацію. Комп'ютерний зір — це наука про отримання інформації із зображень, на відміну від комп'ютерної графіки, яка займається створенням зображень. Його розвиток напряму залежить від рівня комп'ютерних технологій — як щодо підвищення якості зображень, так і щодо розпізнавання [1].

Основна мета комп'ютерного зору — створення моделей, виділення даних і вилучення інформації із зображень, тоді як обробка зображень зосереджується на комп'ютерних перетвореннях самих зображень (різкості, контрастності тощо).

Дана галузь частково перетинається з людино-комп’ютерною взаємодією (*HCI*). *HCI* зосереджується на дизайні, візуалізації та всіх аспектах технологій, пов’язаних із взаємодією між людиною та комп’ютером. Згодом вона переросла на окрему міждисциплінарну науку, яка досліджує взаємозв’язки між людиною і комп’ютером, включно з людським фактором [2]. Функціонально комп’ютерний та людський зір мають спільну мету — інтерпретувати дані, що мають більш ніж один вимір. Однак розраховувати, що комп’ютерний зір повністю відтворить роботу людського ока, неможливо, оскільки його продуктивність і функціональність є обмеженими порівняно з біологічним зором.

Однією з головних проблем технологій комп’ютерного зору є висока чутливість параметрів, обмежена стійкість алгоритмів і неточність результатів, що ускладнює оцінювання ефективності таких систем. Зазвичай оцінювання включає вимірювання основних характеристик алгоритму, таких як точність, надійність і масштабованість, з метою контролю та моніторингу продуктивності системи.

Комп’ютерний зір працює за допомогою алгоритмів і оптичних сенсорів, імітуючи людське сприйняття для автоматичного вилучення необхідної інформації з об’єкта. Порівняно з традиційними методами, які потребують багато часу та складного лабораторного аналізу, комп’ютерний зір перетворився на окрему галузь штучного інтелекту, що відтворює принципи людського зору.

Ця технологія також поєднується з системами висвітленням, що полегшує процес отримання зображень і подальший їх аналіз. Процес аналізу зображень можна поділити на такі етапи:

1) фіксація зображення об’єкта та збереження на комп’ютері;
2) підвищення якості зображення;
3) виокремлення об’єкта з фону;
4) кількісне визначення значущих характеристик;
5) інтерпретація зображення.

У системах відеоспостереження з функцією комп’ютерного зору використовуються наступні технології:

1) відеоаналітіка — програмний алгоритм, який дозволяє швидко і якісно обробляти відеодані і звільнити оператора від звичної роботи спостерігання за великою кількістю камер; розвивається за двома основними технологіями трекінг і ідентифікація; на базі правил, закладених в алгоритм відеоаналізу, формується весь функціонал системи, який важливий для побудови сучасних систем відео нагляду;

2) трекінг — програмний алгоритм обробки вишукує в кадрі рух, визначає і класифікує об’єкт, що пересувається, визначає його характеристики (розмір, колір, швидкість); трекінг включає інтелектуальний пошук в архівах; трекінг дає змогу оператору швидко знаходити необхідний матеріал після спрацювання детектора у випадках, коли точний час події невідомий;

3) ідентифікація — програмний алгоритм розпізнавання образів за відеозображенням, який передбачає класифікацію об’єктів за певними класами або шаблонами та їх порівняння із заздалегідь підготовленою базою еталонних зображень; до основних напрямів ідентифікації належать розпізнавання об’єктів (людей, тварин, транспортних засобів) і розпізнавання державних номерних знаків автомобілів; остання функція є однією з найпоширеніших і найбільш затребуваних у сучасних системах відеоспостереження, при цьому точність розпізнавання може досягати до 95% [3].

1.2 Основні етапи та методи обробки зображень

Обробка зображень є одним із основних етапів функціонування систем комп’ютерного зору, що забезпечують попередню підготовку візуальної інформації перед її аналізом і розпізнаванням. Метою цього процесу є підвищення якості вхідного зображення, виділення важливих деталей, усунення шумів та поліпшення точності подальшої ідентифікації об’єктів. Наведу графічну схему алгоритму попередньої обробки зображення (рис. 1.1).

Початковим етапом є отримання вхідного зображення із камери, встановленої на борту БПЛА. Отримане зображення може бути низької якості: містить шуми, засвітлення, викривлення та тому подібне.

Також варто зауважити, що якість вхідних даних залежить від характеристик камери (частота кадрів, динамічний діапазон, роздільна здатність) та зовнішніх умов зйомки (освітлення, погодні умови).

Рисунок.1.1 – Графічна схема алгоритму попередньої обробки зображення

Початковим етапом є отримання вхідного зображення із камери, встановленої на борту БПЛА. Отримане зображення може бути низької якості: містить шуми, засвітлення, викривлення та тому подібне. Також варто зауважити, що якість вхідних даних залежить від характеристик камери (частота кадрів, динамічний діапазон, роздільна здатність) та зовнішніх умов зйомки (освітлення, погодні умови).

Етап покращення зображення спрямований на підвищення його якості та придатності для подальшого аналізу. На даному етапі проводяться операції фільтрації, нормалізації яскравості та контрасту, усунення шумів і корекції спотворень. Основною метою є збереження важливих структурних особливостей об’єктів, водночас зменшуючи вплив випадкових перешкод і недоліків зйомки. Результатом є поліпшене зображення з підвищеною інформативністю та чіткішими межами об’єктів.

Сегментація (рис. 1.2) — це процес поділу зображення на логічно пов’язані ділянки відповідно до їхніх візуальних характеристик. На цьому етапі система визначає межі між об’єктами та фоном, що дозволяє виділити області, які становлять інтерес для подальшого розпізнавання. Результатом є розподіл сцени на окремі сегменти для більш детального аналізу [4].

Рисунок 1.2 – Сегментація зображення

Зазвичай після сегментації лишаються області, що не маються аналітичної цінності. На даному етапі відбувається очищення зображення від таких ділянок шляхом відсікання або фільтрації. Це дозволяє акцентувати увагу лише на тих об’єктах, які є цільовими для задачі розпізнавання.

Кінцевим етапом є отримання обробленого зображення, яке готове для подальшої роботи з ним з метою розпізнавання або класифікації шуканого об’єкту.

Фільтрація зображень являється значущою частиною попередньої обробки для покращення їх якості перед майбутньою обробкою або класифікацією за допомогою нейронних мереж.

Логарифмічне підсилення застосовується для підвищення контрасту зображень, особливо у випадках, коли вони характеризуються значним діапазоном яскравості. Суть методу полягає у використанні логарифмічного перетворення до значень пікселів, що забезпечує стискання динамічного діапазону яскравості та зменшення впливу яскравих ділянок. Завдяки цьому покращується візуалізація деталей у темних частинах зображення, що сприяє підвищенню його інформативності для подальшої обробки та аналізу.

Формула 1.1 для логарифмічного перетворення:

$$S = c * \log(1 + r) \tag{1.1}$$

де S — це перетворене значення пікселя; r — початкове значення пікселя; c — константа для нормалізації результату.

Дана формула використовується для зменшення впливу яскравих ділянок зображення та покращення видимості темних областей.

Логарифмічне підсилення широко застосовується під час обробки медичних зображень, зокрема рентгенограм, де необхідно підвищити деталізацію у затемнених областях.

Гаусівський фільтр (рис. 1.3) – це один із лінійних фільтрів, який використовується для згладжування та усунення шуму на зображенні.

Гаусівський фільтр функціонує на основі принципу конволюції, відповідно до якого для кожного пікселя вхідного зображення обчислюється нове значення як зважене середнє інтенсивностей сусідніх пікселів.

Вагові коефіцієнти визначаються гаусовим розподілом, що описує внесок кожного пікселя у формування результату згладжування залежно від його відстані до центрального пікселя [5]. Розподіл Гауса є неперервним розподілом імовірностей із функцією щільності (див. формула 1.2):

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \quad (1.2)$$
$$(\sigma < 0, -\infty < x < \infty)$$

де μ - середнє значення, а σ - стандартне відхилення розподілу.

Функція розподілу ймовірностей *F(x)* представлена інтегралом [6]:

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{1}{2}(\frac{x-\mu}{\sigma})^2} du$$
$$(\sigma < 0, -\infty < x < \infty)$$

Результат обчислення за Гаусівським фільтром заміщує початкове значення відповідного пікселя, після чого процедура повторюється для всіх пікселів зображення.

Рисунок 1.3 – Застосування фільтра Гауса

У процесі фільтрації Гаусовий фільтр виконує розмиття області зображення, пригнічуючи високочастотні шуми та дрібні коливання яскравості. Це лінійний фільтр, який згладжує зображення, зменшуючи шум, але водночас частково розмиває краї об’єктів. У цифровій обробці зображень фільтр реалізується у вигляді ядра, що послідовно застосовується до кожного пікселя з урахуванням його локального оточення. Центральний елемент ядра має найбільшу вагу, тоді як ваги сусідніх пікселів з часом зменшуються зі збільшенням відстані від центру. Такий розподіл ваг, що відповідає нормальному розподілу, забезпечує плавне згладжування зображення та зниження впливу шумів, водночас зберігаючи основну структуру об’єктів.

Фільтр Лапласа (рис. 1.4) застосовується для виявлення контурів і меж на зображеннях. Це другий порядок диференціального оператора, який використовується для знаходження місць різкої зміни інтенсивності, що вказує на присутність контурів об'єктів. Даний фільтр є доволі чутливим до шуму, тому зачасту застосовується після попереднього згладжування зображення за допомогою Гаусового фільтра.

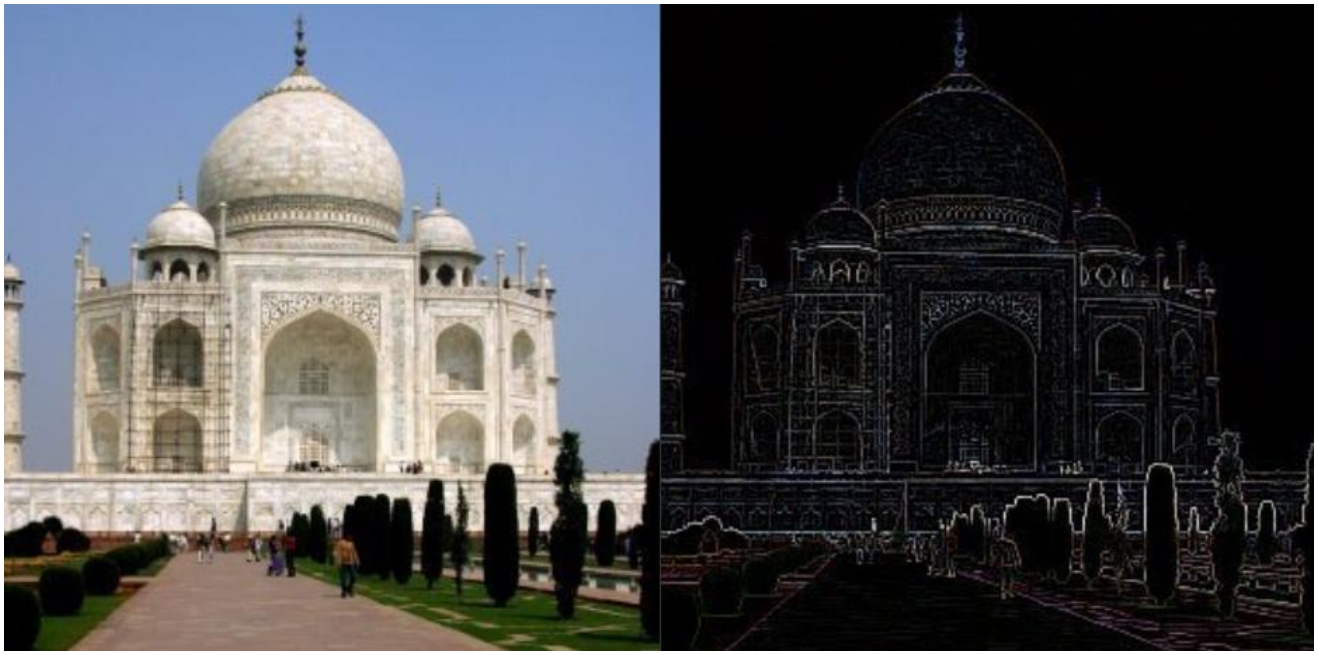


Рисунок 1.4 – Застосування фільтра Лапласа

Дискретна форма оператора Лапласа для двовимірного зображення визначається за формулою:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

де x та y — координати пікселя; f — функція інтенсивності зображення; $\nabla^2 f$ — оператор Лапласа, який виявляє контури у f; $\frac{\partial^2 f}{\partial x^2}$ — друга часткова похідна функції f по змінній x; $\frac{\partial^2 f}{\partial y^2}$ — це друга часткова похідна функції f по змінній y [7].

Фільтр Собеля (рис. 1.5) – дискретний диференціальний оператор, який використовується для наближення градієнта яскравості зображення. Для обробки зображень він застосовується для виявлення контурів. Принцип роботи фільтра базується на обчисленні градієнта яскравості у двох напрямках: горизонтальному та вертикальному. Для цього застосовуються дві матриці згортки (ядра), які визначають різницю інтенсивностей між пікселями в даних напрямках [8].

Рисунок 1.5 – Застосування фільтра Собеля [9]

Після застосування обох фільтрів одержуються дві градієнтні складові G_x та G_y, на основі яких розраховується загальна величина градієнта за формулою:

$$|G| = \sqrt{G_x^2 + G_y^2}$$

Обчислення амплітуди градієнта дозволяє оцінити швидкість зміни інтенсивності пікселів та визначити наявність контурів по всьому зображенню. Орієнтація виявлених меж визначається відносно напряму максимального контрасту — проти годинникової стрілки від темних до світлих ділянок.

Використання лише фільтра Собеля не забезпечує бажаного результату, оскільки на зображенні можуть бути присутні шуми. Для ліпшого результату можна встановити порогове значення, що дозволяє відсікати слабкі градієнти.

1.3 Класифікація безпілотних літальних апаратів

Безпілотні літальні апарати — це літальні апарати, які функціонують без людини-пілота на борту, що керуються на відстані за допомогою оператора або автономно за допомогою вбудованих комп’ютерів та систем.

БПЛА класифікуються за широким спектром різноманітних критеріїв, що описують їхні технічні, конструктивні та функціональні особливості. Єдиного загальноприйнятого стандарту класифікації БПЛА наразі не існує. Оборонно-військові відомства використовують власні системи класифікації, утворені на тактико-технічних характеристиках, таких як маса, дальність та тривалість польоту, рівень автономності. Якщо мова йдеться про цивільний сектор, то застосовуються більш узагальнені та гнучкі категорії, які постійно оновлюються та покращуються відповідно до розвитку технологій і розширення сфер застосування літальних апаратів.

Відомою системою класифікації БПЛА є класифікація, запропонована НАТО. Доречно її використовувати у військовій практиці для визначення характеристик літальних апаратів відповідно до злітної маси, висотою

застосування, радіусом дії, рівнем автономності та категорії військових операцій. Згідно з класифікацією НАТО, безпілотники поділяються на три головні класи залежно від загальної злітної маси. Класифікація БПЛА за стандартами НАТО наведена в таблиці 1.1.

Таблиця 1.1 – Класифікація БПЛА за стандартами НАТО [10]

Клас	Категорія	Призначення	Робоча висота	Радіус польоту	Основний командир	Приклад платформи
1	2	3	4	5	6	7
Клас III (> 600 кг)	Ударні / бойові	Стратегічне / національне	До 20 000 м	Необмежений	Стратегічна область	*Reaper*
	Висотний з великою витривалістю	Стратегічне / національне	До 20 000 м	Необмежений	Стратегічна область	*Global Hawk*
	Середньої висоти з великою витривалістю	Оперативний / стратегічна область	До 14 000 м	Необмежений	Оперативне угруповання	*Heron*
Клас II (150–600 кг)	Тактичні	Тактична підготовка	До 5 500 м	До 200 км	Бригада	*Hermes* 450

Продовження таблиці 1.1.

1	2	3	4	5	6	7
Клас I (<150 кг)	Малий (>15 кг)	Тактична одиниця	До 1 500 м	До 50 км	Батальйон, полк	*Scan Eagle, PD*-2
	Міні (>15 кг)	Міні (<15 кг)	Тактична підодиниця	До 900 м	До 25 км	Рота, взвод, відділення
	Мікро (<66 Дж енергії зльоту)	Тактична підодиниця	До 60 м	До 5 км	Взвод, відділення	*Black Widow*

У ході дослідження систем розпізнавання об’єктів важливо звернути увагу на різноманіття типів безпілотних літальних апаратів, на базі яких може здійснюватися відеоспостереження та фіксація цільової інформації. Це пояснюється тим, що характеристики конкретного типу БПЛА мають значний вплив на точність розпізнавання, якість зображення, тривалість місії та умови її виконання. Тому вибір відповідного літального апарата має ґрунтуватися на аналізі його технічних можливостей та призначення.

З метою систематизації та полегшення вибору безпілотні літальні апарати класифікують за низкою організаційних та технічних ознак. Така класифікація дозволяє виділити ключові параметри, за якими здійснюється розмежування між окремими типами апаратів. До таких параметрів належать масштаби застосування, можливість повторного використання, спосіб управління та відомча належність. Кожен із цих критеріїв має практичне значення при визначенні сфери використання БПЛА у межах конкретної операції або завдання.

На рисунку 1.6 наведено класифікацію за організаційними ознаками.

Класифікацію БПЛА за технічними ознаками подано в додатку А.

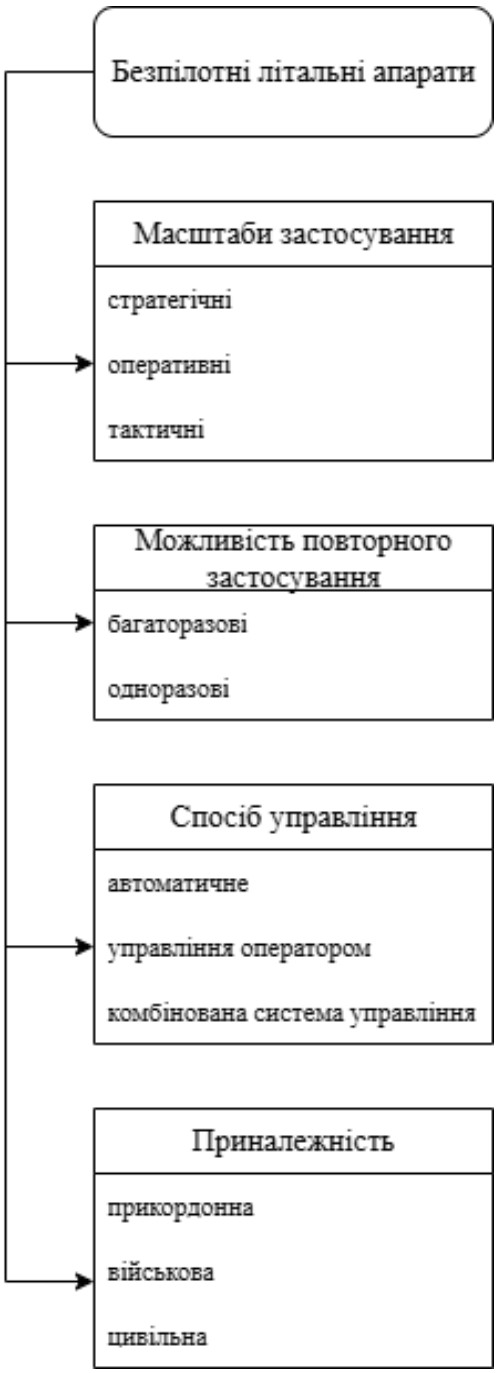


Рисунок 1.6 – Класифікація за організаційними ознаками [11]

1.4 Датчики на борту БПЛА

Безпілотні літальні апарати можуть бути оснащеними різноманітними типами датчиків, які використовуються для отримання інформації про навколишнє середовище та стан дрону. Конфігурація бортових сенсорів залежить на пряму від

умов експлуатації, цільового призначення, вартості апарату та захищеності від надзвичайних ситуацій (втрата сигналу, погодні умови та подібне).

Кожен тип датчика має свої технічні особливості, переваги й обмеження, що визначають сферу його використання. У системах розпізнавання об’єктів на базі БПЛА застосовуються пасивні та активні сенсори, що забезпечують комплексне сприйняття навколишнього простору. У даному підрозділі проведу аналіз найбільш розповсюджених датчиків.

Камери *RGB* являються пасивними сенсорами, які реєструють інтенсивність світла у трьох спектральних діапазонах — червоному, зеленому та синьому. Вони є базовим інструментом збору візуальної інформації для комп’ютерного зору. Основним недоліком використання *RGB*-камер на борту БПЛА є розмиття та шуми зображення через вібрацію, різкі рухи та високу швидкість польоту. Для мінімізації дефектів на знімках застосовують КОМП-камери, які забезпечують зчитування всієї площини кадру, на відміну від *CCD*-камер, де сканування здійснюється стрічково. Для моніторингу та розпізнавання об’єкту з БПЛА використовують відеокамери, що мають роздільну здатність не менше 12 Мп, підтримують запис відео у форматі 4К із частотою понад 30 кадрів/с, широкий динамічний діапазон і можливість роботи в умовах недостатнього освітлення [12].

Камери у градаціях сірого фіксують тільки інтенсивність світлового потоку, забезпечуючи один канал зображення. Вони є доцільними для завдань, де важливим є виявлення контурів або текстур, а не інформації у кольорі.

Камери подій фіксують не повні кадри, а послідовність змін інтенсивності освітлення в окремих пікселях. Такий підхід дозволяє отримувати надзвичайно високу часову роздільну здатність і мінімізувати затримку. Такі сенсори використовують для аналізу руху в реальному часі. Однак їхнє застосування потребує розробки спеціалізованих алгоритмів обробки подій тому, що вихідні дані не є класичними зображеннями.

Тепловізори являються пасивними сенсорами, які фіксують інфрачервоне випромінювання, що випромінюють усі об’єкти з температурою вище абсолютного нуля. Вони дають змогу фіксувати живі об’єкти в умовах повної темряви, диму,

туману й інших оптичних перешкод. Раніше тепловізори використовувались переважно для військових цілей, але при зниженні собівартості вони почали широко застосовуватись у цивільних задачах.

3*D*-камери дозволяють отримувати не лише кольорове, а й глибинне зображення сцени. Основними технологіями для створення таких сенсорів є:

1) стереобачення;
2) *structured light*;
3) *time-of-flight*.

LiDAR — це активний датчик, що з'ясовує відстань до об'єктів шляхом визначення часу за який лазерний імпульс відбивається від поверхні. Такі системи створюють трьохвимірну мапу місцевості з гарною точністю [13].

Раніше *LiDAR*-системи були масивними та дороговартісними, але розвиток твердотільних технологій усунув дані недоліки. Наприклад, *Ouster LiDAR* все частіше використовують у малих та середніх БПЛА для точної просторової орієнтації.

Порівняльна характеристика камер, які зазвичай використовуються для розпізнавання об'єктів з БПЛА наведена в таблиці 1.2.

Таблиця 1.2 – Порівняльна характеристика камер

Тип камери	Принцип роботи	Переваги	Недоліки	Сфери застосування
1	2	3	4	5
RGB-камера	реєстрація інтенсивності видимого світла у трьох спектральних каналах	висока деталізація, невелика вага, доступна вартість, фіксація у реальному часі	залежність від освітлення, спотворення при русі та вібраціях	розпізнавання об'єктів, моніторинг територій, інспекції

Продовження таблиці 1.2

1	2	3	4	5
Камера у градаціях сірого	фіксація інтенсивності світла без кольорових каналів	висока контрастність, низьке енергоспоживання	відсутність кольору, обмежена область застосування	виявлення контурів і структур, навігаційні системи
Камера подій	реєструє зміни яскравості	висока швидкість реакції, низька затримка	не формує класичні зображення, складна подальша обробка	керування польотом у реальному часі, виявлення рухомих об’єктів
Інфрачервона камера	вимірювання інфрачервоного випромінювання від об’єктів	робота у повній темряві, виявлення теплових джерел	низька роздільна здатність, висока ціна	пошуково-рятувальні операції, спостереження, військове застосування
3*D*-камера	визначення глибини за допомогою тріангуляції, часу проходження сигналу або структурованого світла	отримання глибинної інформації, побудова 3D-моделей	залежність від умов освітлення, низька частота кадрів, чутливість до перешкод	навігація, уникнення перешкод, 3*D*-картографування

Закінчення таблиці 1.2

LiDAR	лазерне сканування, вимірювання часу повернення імпульсу	висока точність, незалежність від освітлення, побудова точних 3*D*-карт	висока собівартість, чутливі до дощу/туману.	топографічна зйомка, навігація, автономне керування

1.5 Сфери практичного застосування систем розпізнавання об’єктів з БПЛА

Сучасні безпілотні літальні апарати все частіше комплектуються системами автоматичного розпізнавання об’єктів, що грунтуються на технологіях комп’ютерного зору, машинного навчання та штучного інтелекту. Такі системи в змозі фіксувати зображення, ідентифікувати, класифікувати та відстежувати цільовий об’єкт на місцевості в реальному часі.

Завдяки вище згаданому БПЛА почали широко застосовувати у багатьох сферах.

1.5.1 Сільське господарство

У сільському господарстві системи розпізнавання об’єктів з бортових камер БПЛА використовуються для відслідковування стану посівів, оцінки врожайності та контролю за зрошенням. Завдяки високій просторовій роздільній здатності зображень дрони можуть автоматично визначати стадії росту рослин, визначати частини поля з ознаками посухи чи ураження.

Застосування мультиспектральних і тепловізійних камер дозволяє аналізувати індекс *NDVI* (рис. 1.8), що є важливим показником для оцінки стану вегетації. Отримані значення застосовуються для оптимізації добрив, води та пестицидів.

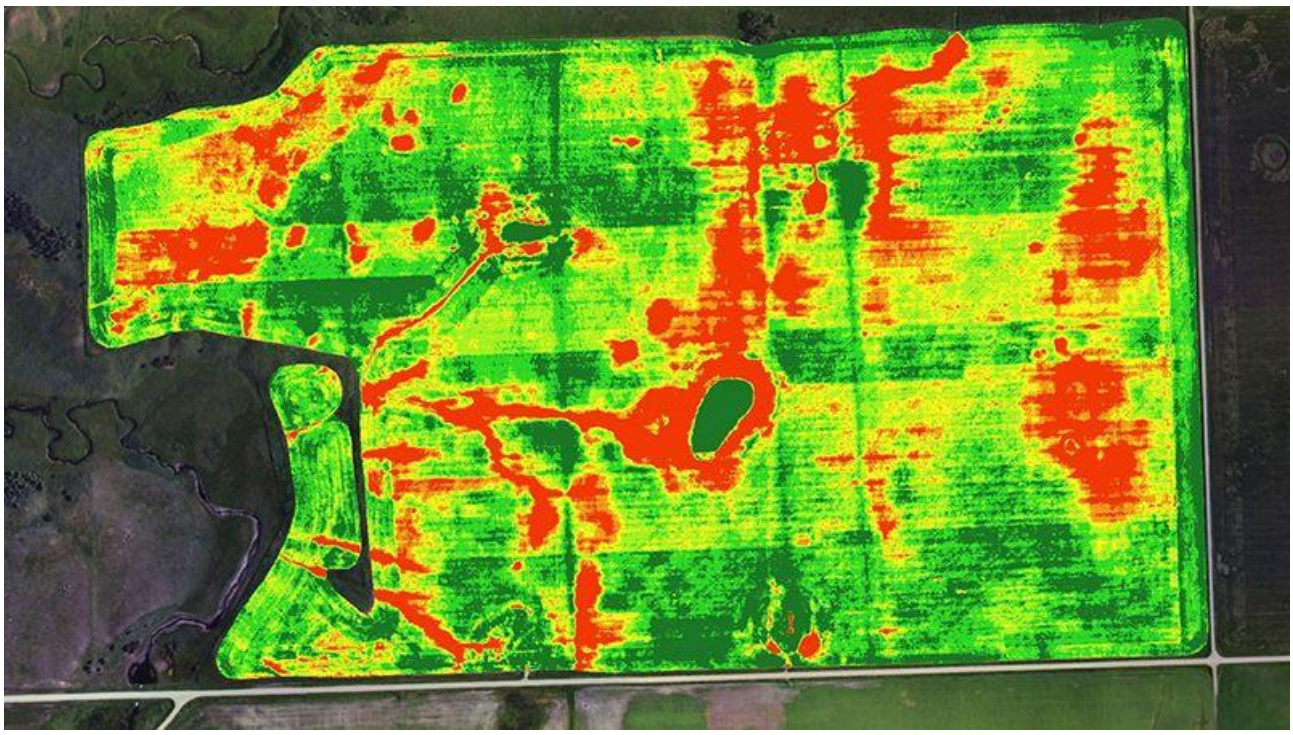


Рисунок 1.8 – Застосування мультиспектральних і тепловізійних камер з БПЛА для визначення індексу *NDVI*

1.5.2 Контроль транспортних потоків і міської інфраструктури

У сучасному світі почали тестувати системи комп’ютерного зору на базі БПЛА для моніторингу дорожнього руху (рис. 1.9), управління транспортними потоками та контролю стану інфраструктури.

Рисунок 1.9 – Приклад моніторингу дорожнього руху з БПЛА

Дрони можуть розпізнавати автомобілі, пішоходів, затори, аварійні ситуації, а також відслідковувати їх динаміку у реальному часі. Дана інформація може інтегруватися в автоматизовані системи керування трафіком, що зменшує ймовірність дорожніх інцидентів. Також системи розпізнавання дозволяють автоматизовано та своєчасно оцінювати стан мостів, будівель та доріг, що підвищує безпеку експлуатації об’єктів.

1.5.3 Використання у сфері безпеки та оборони

Сфера безпеки та оборони є однією з найважливіших для використання систем розпізнавання об’єктів із БПЛА. Такі системи можуть автоматично виявляти техніку (рис. 1.10), військові позиції або переміщення супротивника. Дані системи відіграють ключову роль у сьогоденних бойових інформаційно-аналітичних комплексах [14].

Рисунок 1.10 – Виявлення ворожої техніки за допомогою БПЛА

Висновок до розділу

У ході написання першого розділу було розглянуто поняття комп’ютерного зору, методи обробки зображень, класифікації безпілотних апаратів, типи датчиків, які використовуються для розпізнавання об’єктів з БПЛА, а також сфери практичного застосування таких систем.

У результаті теоретичного аналізу встановлено, що комп'ютерний зір є ключовою технологією автоматизованих систем для реалізації задач виявлення, класифікації та відстеження об'єктів. Його основною метою являється навчання машини людському баченню та розумінню.

Розглянуто основні етапи обробки зображень, які охоплюють попередню обробку, покращення якості зображення, сегментацію, виявлення ознак та класифікацію. Попередня обробка дозволяє усунути шум, покращити контраст і підвищити точність подальшого аналізу.

Також опрацьовано основні класифікації безпілотних літальних апаратів за міжнародними стандартами НАТО, організаційними та технічними ознаками. Розгляд цих класифікацій має важливе значення для розроблення автоматизованої системи розпізнавання об'єктів із камер БПЛА тому, що знання типів, параметрів і особливостей конструкції апаратів надає змогу точніше зрозуміти напрямок застосування таких систем та в кінцевому результаті отримати високу точність вихідної інформації.

Особлива увага приділена дослідженню датчиків, які використовуються на борту БПЛА. Проаналізовано різні типи датчиків, які збирають візуальну, теплову та просторову інформацію. Зокрема, розглянуто особливості *RGB,* інфрачервоної та 3*D* камер, а також систем *LiDAR*. Визначено переваги та недоліки кожної з камер та проведено їх порівняння.

Окремо проаналізовано сфери практичного застосування систем розпізнавання об'єктів з БПЛА. У сільському господарстві такі системи використовують для відслідковування стану посівів, оцінки врожайності та оптимізації використаних ресурсів. У міській інфраструктурі БПЛА застосовуються для контролю транспортних потоків, моніторингу стану будівель та доріг. Досліджувана система активно застосовується у сфері безпеки та оборони для виявлення потенційних загроз або техніки та проведення розвідки.

Таким чином, проведені у першому розділі аналізи створили теоретичну основу для подальших досліджень у наступних розділах роботи.

РОЗДІЛ 2
МЕТОДИ ТА ТЕХНОЛОГІЇ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ З БОРТОВОЇ КАМЕРИ БПЛА

2.1 Сучасні детектори об'єктів

Виявлення об'єктів є одним із найпоширенішим завданням комп'ютерного зору, яке поєднує локалізацію та класифікацію цільового об'єкту на зображеннях. Впродовж останнього десятиліття поширення згорткових нейронних мереж значно збільшило рівень розвитку у цій сфері, забезпечивши надійну роботу в специфічних умовах.

Сучасні алгоритми детекції розділяють на одноступеневі та двоступеневі. Кожен з них має унікальний підхід до виявлення об'єктів з різними перевагами та недоліками. Тому варто розглянути популярні детектори та провести їх порівняння для вибору кращого у обраній сфері.

2.1.1 Одноступеневі детектори

До найпопулярніших одноступеневих детекторів відносять методи *SSD* та *YOLO*.

YOLO був представлений у 2016 році Р. Гіршиком. Фреймворк був поданий для переосмислення виявлення об'єктів як єдиної регресійної задачі, що дозволяє знаходити об'єкти в реальному часі шляхом безпосереднього прогнозування обмежувальних рамок і ймовірностей класів з усього зображення за один прохід згорткової нейронної мережі. Дана архітектура об'єднує локалізацію та класифікацію об'єктів в одній моделі, завдяки чому збільшується швидкість виявлення.

YOLO містить три основні компоненти: вхідний шар, шар вилучення ознак та вихідний шар. Вони призначені для точного визначення розташування, категорій та положень об’єктів [16].

Початковим етапом методу *YOLO* являється поділ вхідного зображення на сітку з комірок однакового розміру. Підхід на основі сітки дозволяє знаходити декілька об’єктів в одному кадрі. Комірка сітки відповідає за прогнозування об'єкта, але за умови коли центр обмежувальної рамки об'єкта знаходиться всередині неї. Це усуває непотрібні обчислення та забезпечує якісну обробку об'єктів, що перекриваються. На рисунку 2.1 продемонстровано яким чином створюється обмежувальна рамка для однієї комірки сітки.

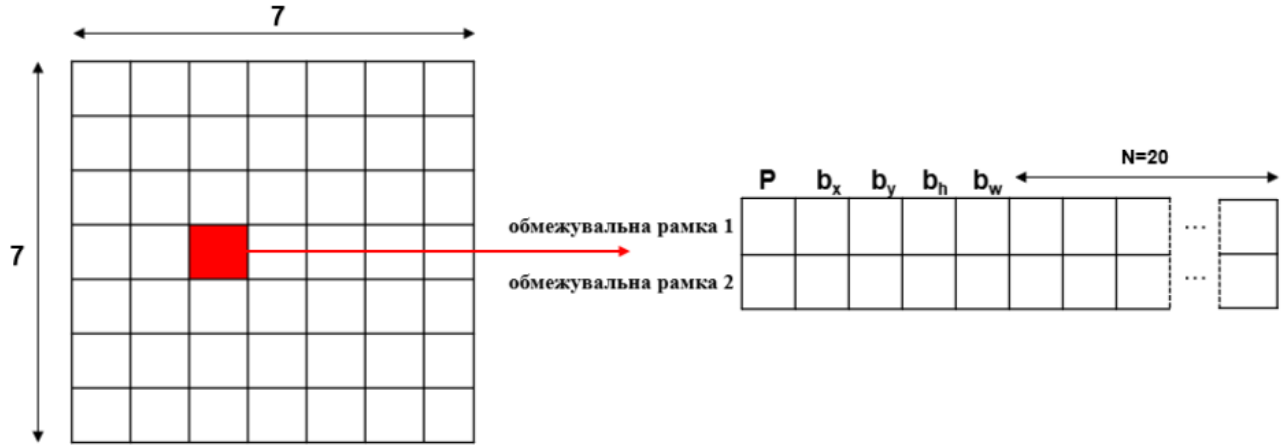


Рисунок 2.1 – Створення обмежувальних рамок для однієї комірки сітки

Алгоритм *YOLO* розділяє вхідне зображення на сітку розміром *S* на *S* (зазвичай обирається 7 на 7), причому кожна комірка відповідає за виявлення об'єктів і містить *B* обмежувальних рамок. В свою чергу обмежувальна рамка має наступні параметри:

1) b_x, b_y – координати центру;
2) b_w – ширину;
3) b_h – висоту;
4) *P* – ймовірність того, що рамка містить об’єкт.

Комірка сітки передбачає N ймовірностей класів, що відповідають можливим категоріям об'єктів. В цілому вона прогнозує $B \times 5 + N$ значень. Якщо говорити про всю сітку, тоді отримаємо $S \times S \times (B \times 5 + N)$. Для прикладу наведеного на рисунку 2.1 з сіткою 7 на 7, двома обмежувальними рамками на комірку та $N = 20$ отримуємо, що кожна комірка сітки передбачає 30 значень, а загальна кількість рівна 1470 передбачень для всього зображення. Отримані прогнозування формують вихідний тензор. Оцінка достовірності для кожної обмежувальної рамки обраховується за наступною формулою:

$$C = \Pr(obj) * IoU_{truth}^{pred}$$

де IoU_{truth}^{pred} – площа перекриття поділена на площу об’єднання, $\Pr(obj)$ – ймовірність того, що комірка сітки містить об'єкт.

Поширеним явищем при виявленні об’єктів є перекриваючі обмежувальні рамки. Особливо спостерігається у випадках коли великі об’єкти охоплюють кілька комір сітки. Для усунення даної проблеми *YOLO* використовує оцінку достовірності класу до кожної обмежувальної рамки для кількісної оцінки ймовірності присутності об'єкта. Поріг достовірності встановлюється для відсіювання обмежувальних рамок з низьким показником. Навіть при застосуванні порогу, перекриваючі обмежувальні рамки можуть залишатися. Для повного усунення цього недоліку *YOLO* використовує *NMS*. Основні кроки *NMS*:

1) ідентифікація обмежувальної рамки з найбільшим балом достовірності та визначенням її як максимальної обмежувальної рамки;

2) обчислення *IoU* між максимальною обмежувальною рамкою та всіма іншими;

3) видалення всіх обмежувальних рамок, що перевищують заздалегідь визначений поріг;

4) повторення процесу, доки не залишаться лише неперекриваючі обмежувальні рамки з найвищою достовірністю [17].

Кінцевий результат поєднує інформацію про комірки сітки з типом і положенням обмежувальної рамки з найвищою достовірністю, що забезпечує точне розпізнавання та локалізацію об'єктів.

Архітектура першої версії *YOLO,* яка являється початком розвитку даної моделі наведена на рисунку 2.2.

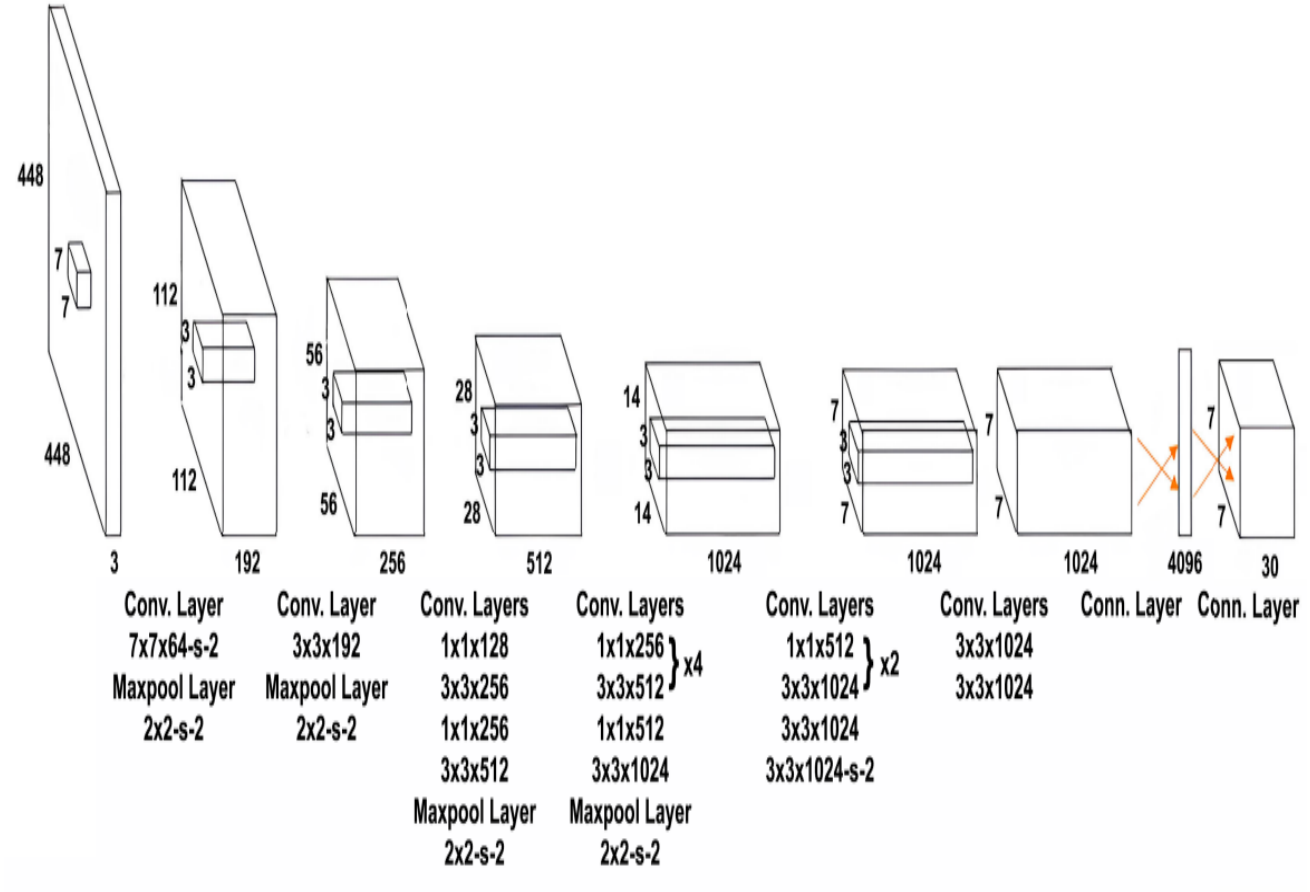


Рисунок 2.2 – Архітектура *YOLO* [18]

Ще однією популярною системою детекції є *SSD*. Мета полягає в тому, щоб реалізовувати класифікацію та локалізацію об’єктів за один прямий прохід нейронної мережі. І тим самим має пришвидшитися час виконання без вагомої втрати точності.

Модель накладає заздалегідь визначений набір рамок за замовченням у кожній точці карт ознак.

Для всіх карт заплановано рамки з п’ятьма відмінними співвідношеннями сторін {1, 2, 0.5, 3, 0.33} та п’ятьма масштабами {0.2, 0.375, 0.55, 0.725, 0.9}. В ході навчання мережа змінює дані рамки під конкретні об’єкти.

Архітектура *SSD* (рис. 2.3) використовує за основу *VGG*-16 та чотири додаткові згорткові шари, щоб формувати карти ознак різного масштабу. Таким чином можна визначати об’єкти будь-якого розміру.

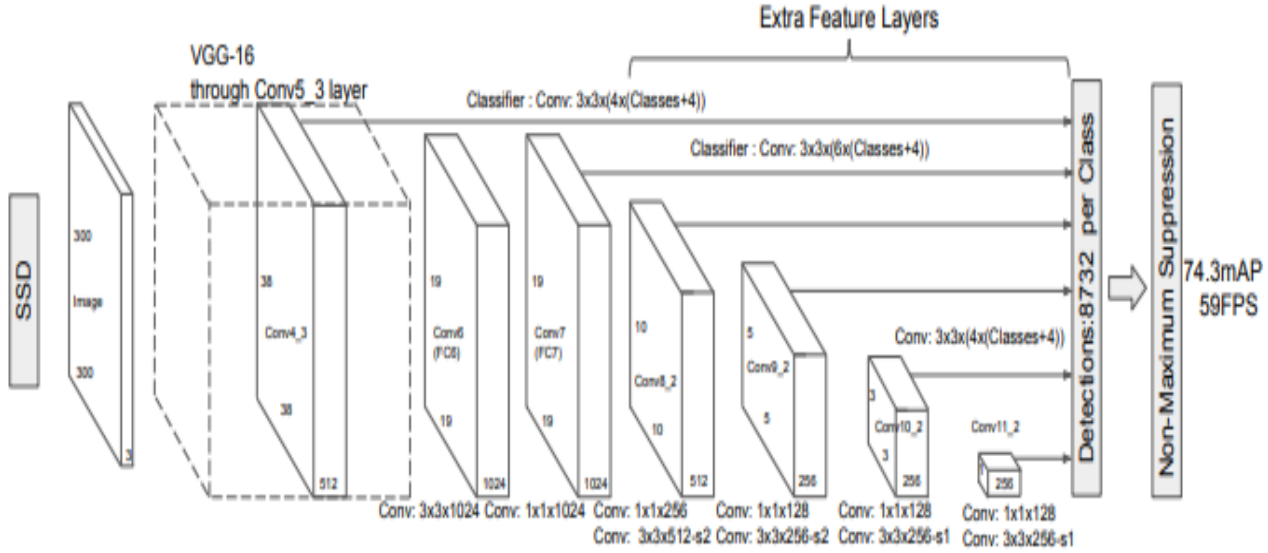


Рисунок 2.3 – Архітектура *SSD*

VGG-16 дуже гарно себе зарекомендувала у класифікації зображень, які мають високу роздільну здатність. Повнозв'язні шари *VGG* були замінені на допоміжні згорткові шари для виявлення цільових об'єктів з різними масштабами та скорочення розміру даних, які надходять до слідуючих шарів. Як і у *YOLO* використовується метод немаксимального придушення.

Під час навчання *SSD* застосовує багатозадачну втрату, яка обчислюється за формулою:

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

Варто зауважити ряд обмежень *SSD*:

1) низька продуктивність на малих об'єктах;

2) нерухомі анкерні рамки;

3) відсутня чітка оцінка об’єктності [19].

2.1.2 Двоступеневі детектори

Виявлення об’єктів за допомогою двоступеневих детекторів відбувається у два етапи. На початку формується обмежена кількість регіонів з об’єктами, а потім для кожного з регіонів виконується точна класифікація та уточнення меж. Якщо порівняти з одноступеневими моделями, то вагомою відмінністю являється те, що двоступеневі не прогнозують класи та межі для всього зображення. Вони, насамперед, визначають місце де варто шукати, а потім визначають об’єкти, що знаходяться у цих частинах картини. Такий підхід покращує стійкість до складних фонів та близько розташованих об’єктів.

У 2014 році Гіршиком та іншими була запропонована згорткова нейронна мережа на основі регіонів (*R-CNN*), яка являється поштовхом до прогресу виявлення об’єктів з використанням механізмів пропозицій області з глибоким згортковим виокремленням ознак. *R-CNN* використовує багатоступеневий конвеєр виявлення (рис. 2.4).

В загальному етапи виявлення об’єктів за допомогою *R-CNN* можна описати наступним чином:

1) формування навчальної вибірки виділення, класифікації та позиціювання об’єктів. За допомогою неї та допоміжних алгоритмів відбувається тренування моделі згорткової нейромережі.

2) Пошук вибіркових регіонів інтересу, що можуть містити цільові об’єкти.

3) Уніфікація розмірності регіонів інтересу згідно параметрів нейромережевої архітектури. Видалення областей, що не містять ніяких об’єктів.

4) Регулювання параметрів нейромережі, спираючись на параметри регіонів, що містять цільові об’єкти.

5) Визначення набору ознак для кожного з регіонів інтересу. Застосування регресійної моделі для виявлення нових областей інтересу [20].

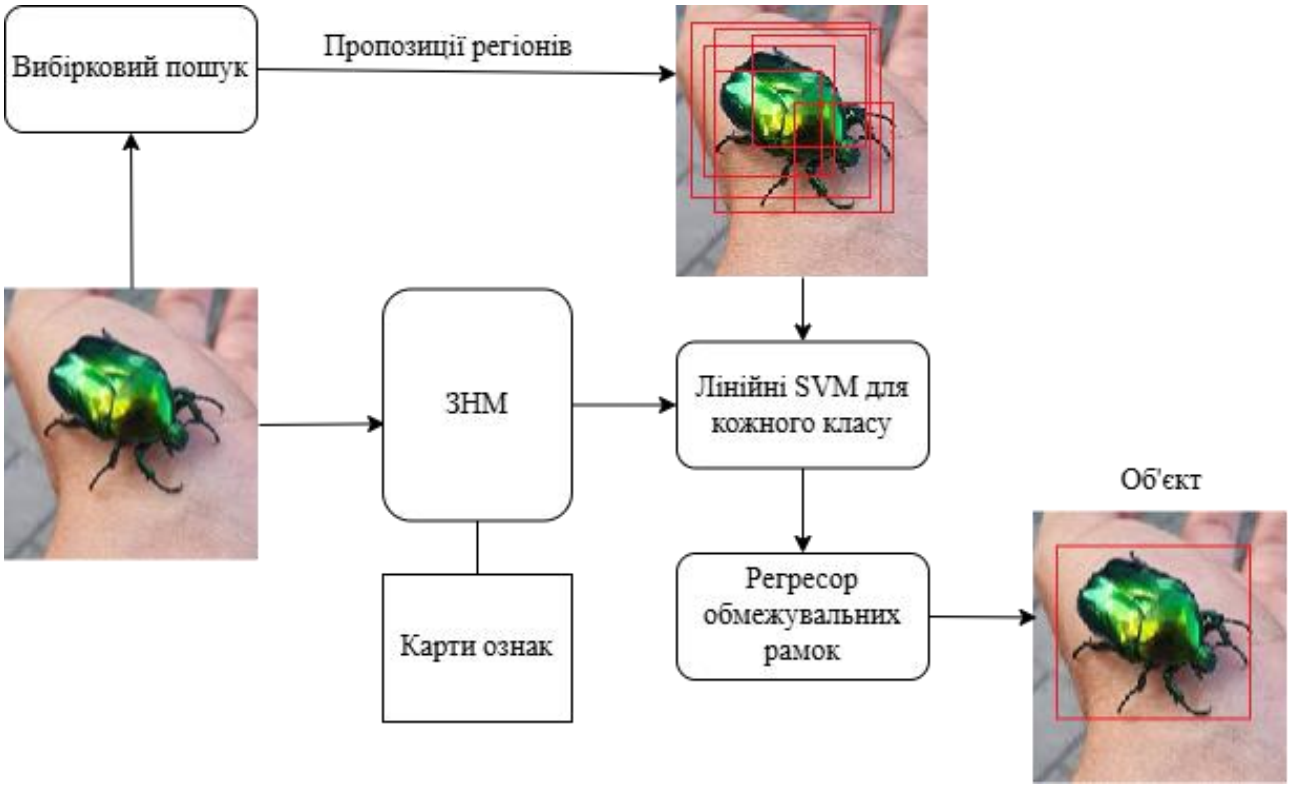


Рисунок 2.4 – Діаграма роботи *R-CNN*

Згодом розробили покращену версію вище розглянутого детектора. *Faster R-CNN* є одним із найрозповсюдженим представником двоступеневої моделі виявлення, яка вміщує в собі *RPN* та мережеву модель *Fast R-CNN*. Частини пришвидшеного детектора працюють паралельно та навчаються незалежно одна від одної для того, щоб була змога знаходити категорію класифікації та регресійний блок локалізації.

Головним нововведенням *Faster R-CNN* являється впровадження *RPN* (рис. 2.5). Вона створює припущення щодо регіонів з карти ознак. *RPN* рухає згорткове вікно розмірністю 3 на 3 над колективною картою ознак. У всіх просторових місцях вона керує:

1) *k* показників об’єктності;
2) 4*k* зміни положення обмежувальної рамки відповідно до *k* якорів.

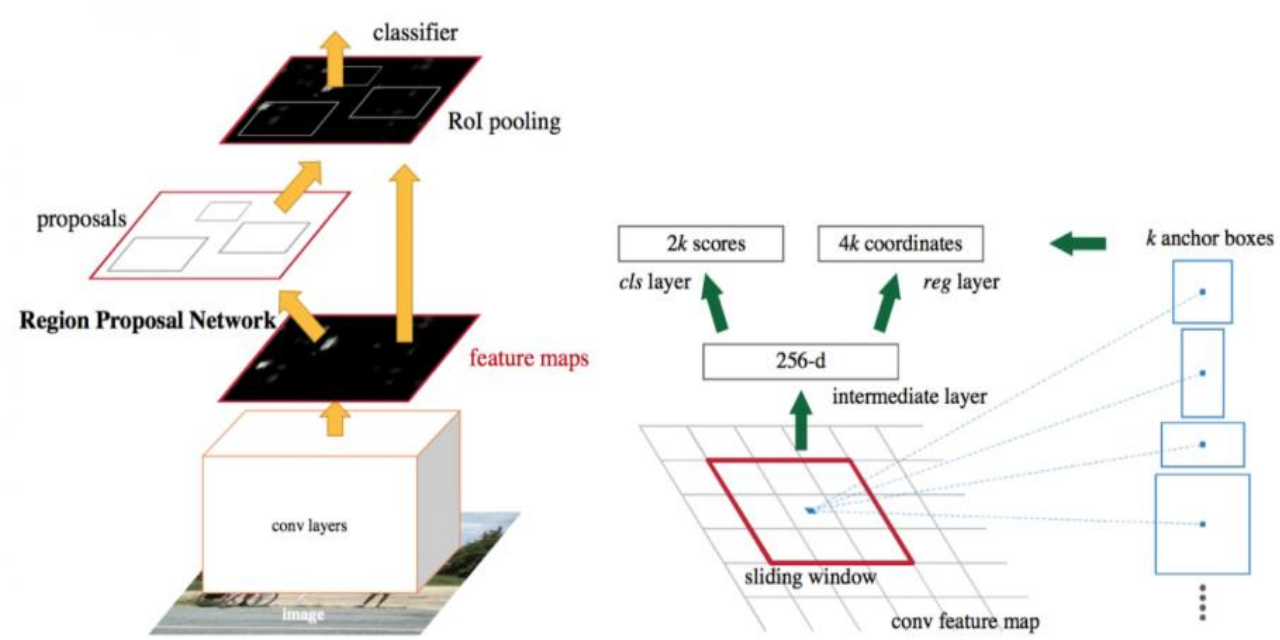


Рисунок 2.5 – Архітектура *RPN*

Розберемо детально роботу *Faster R-CNN* (рис. 2.6).

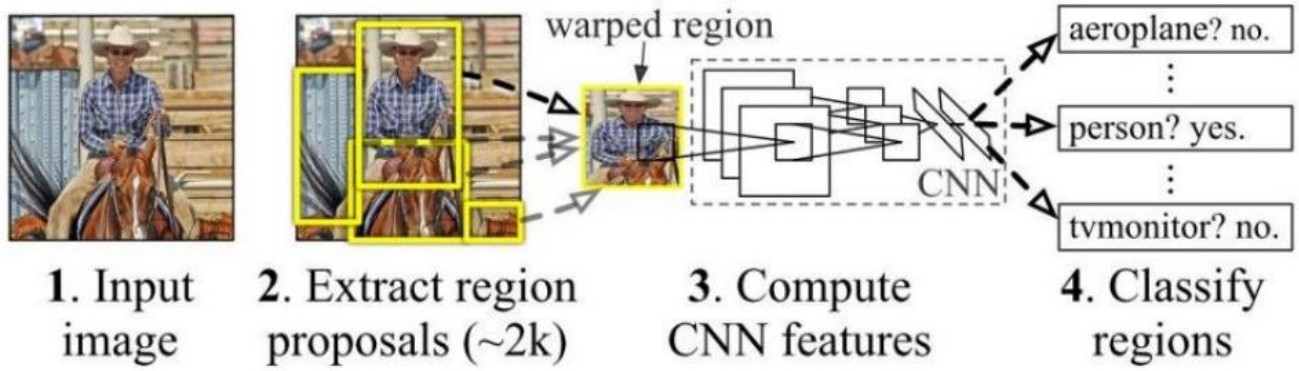


Рисунок 2.6 – Архітектура *Faster R-CNN* [21]

На початковому етапі використовується *ResNet* або *VGGNet*, які завчасно були навчені на заваданнях класифікації зображень з великим масштабом. В результаті вхідні зображення перетворюються на серію карт ознак. Отримані дані застосовує *RPN* для генерації пропозиції регіонів, які цілком ймовірно містять об’єкти інтересу.

Важливим пунктом являється об’єднання та вирівнювання *RoI*. Необхідний він для вилучення карт ознак сталого розміру з карт ознак, що отримали після

роботи *CNN* мережі. Це гарантує однакове представлення кожної області інтересу, незалежно від її масштабу чи співвідношення сторін.

Вилучені ознаки надходять в блок детекції, який застосовується для прогнозування ймовірностей класів та конкретизації координат обмежувальних рамок для кожного з регіонів. Даний етап конкретизує запропоновані регіони і підвищує точність локалізації та класифікації об’єктів.

В ході навчання *Faster R-CNN* оптимізує свої параметри за допомогою функції втрат. Вона містить в собі комбінацію втрат класифікації та локалізації для того, щоб точніше визначати локалізацію об'єктів у запропонованих обмежувальних рамках.

Кінцевим етапом являється застосування *NMS* для фільтрації непотрібних обмежувальних рамок, які не містять цільових об’єктів.

2.1.3 Порівняльний аналіз

Для вибору архітектури детектору для подальшого розроблення системи розпізнавання об’єктів з камери БПЛА гострою необхідністю являється проведення якісного порівняльного аналізу. Спираючись на загальні теоретичні відомості, можна сформувати порівняльну характеристику моделей, яку навів у таблиці 2.1.

Таблиця 2.1 – Порівняльна характеристика моделей

Модель	*FPS*	Складність	Переваги	Недоліки	Застосування з БПЛА
1	2	3	4	5	6
R-CNN	<3	висока	висока локалізація	повільний, висока вартість	не використовується для розпізнавання у реальному часі через низькі показники

Закінчення таблиці 2.1

1	2	3	4	5	6
Faster R-CNN	5-15	висока	точні рамки	низький *FPS*, слкадний	може застосовуватись, але офлайн
SSD	40-90	низька	простота, швидкість	погано розпізнає малі цілі	можна застосовувати на борту, але якість бажає кращого
YOLO	30-90	низька	гарне співвідношення швидкості та якості	необхідно збільшувати зображення при малих об’єктах	гарний варіант для застосування на борту в реальному часі

Так як БПЛА з системами автоматичного розпізнавання об’єктів використовується для забезпечення безпеки та завчасного виявлення військової техніки, то проведемо аналіз роботи кожної з моделей з відповідним набором даних.

Приклад розпізнавання важкої техніки наведено на рисунку 2.7.

Згідно попередньої таблиці можна відразу виключити *R-CNN*. Оскільки дана модель не може застосовуватись у системі автоматизованого розпізнавання об’єктів з БПЛА. Звертати увагу буду на наступні метрики:

1) *precision* – відсоткове відношення правильного спрацювання моделі;
2) *recall* – відсоткове відношення знайдених об’єктів;
3) *F1-Score* – середнє між precision та recall;
4) *mAP* – середня точність по всіх класах, усередненну за різними порогами накладання рамок;
5) *FPS* - кількість кадрів оброблених за секунду [22,23].

Рисунок 2.7 – Виявлення важкої техніки

Даний набір показників дозволяє провести детальний аналіз продуктивності моделей, порівнюючи їхню ефективність роботи.

Порівняльна характеристика моделей для розпізнавання важкої техніки наведена в таблиці 2.2.

Таблиця 2.2 – Порівняльна характеристика моделей для розпізнавання важкої техніки [24]

Модель	*Precision*, %	*Recall*, %	*F1-Score*, %	*mAP*, %	*FPS*
Faster R-CNN	88,7	90,1	89,4	89,2	7
SSD	83,5	82,6	83,4	81	36
YOLO	91,4	84,8	87,97	91,8	55

Модель *Faster R-CNN* продемонструвала значення показників *mAP* – 89,2 % та *F1-Score* – 89,4 %, що демонструє високу точність та влучність моделі. Однак її швидкодія обмежується низьким *FPS* – 7, що негативно впливає при використанні в реальних сценаріях. Хоч і точність виявлення об’єктів знаходиться на високому

рівні модель не відповідає вимогам застосування у режимі реального часу через низьку швидкість обробки вхідних зображень.

В свою чергу *SSD* продемонструвала збалансований результат *mAP* – 81 %, *F1-Score* – 83,4 % та *FPS* – 36. Така модель забезпечує непоганий компроміс між точністю та швидкістю, але має нижчі показники загальної точності розпізнавання порівняно з іншими моделями.

Одні із кращих результатів продемонструвала модель *YOLO*, яка має найвищі показники точності *mAP* – 91,8 % та *FPS* – 55. Швидкість оброблення вхідних кадрів та точність отриманих даних роблять дану модель придатною для використання в реальному часі.

2.2 Специфіка розпізнавання малих об’єктів

Одною із особливостей набору даних зібраних з камери БПЛА є наявність великої кількості малих об’єктів в кадрі. Через це виникає проблема з точним розпізнаванням об’єктів, яку вирішити простим збільшенням глибини мережі не вийде. Надалі буде розглянуто декілька напрямків завдяки яким можна покращити розпізнавання малих деталей.

2.2.1 *Feature Pyramid Networks*

FPN – це спеціалізована архітектура нейронної мережі, розроблена для кращого виявлення та сегментації об’єктів різних розмірів. Традиційні *CNN* мають обмеження у вигляді втрати просторової роздільної здатності, що робить складнішим виявлення малих об’єктів з вхідного кадру. В свою чергу *FPN* навмисно розроблена з ціллю агрегації як високорівневої семантичної, так і низькорівневої детальної просторової інформації в різних масштабах. Вона формує надійні багатомасштабні карти ознак для подальшого виконання поставленого завдання [25]. Якщо мова йдеться про розпізнавання об’єктів з камер БПЛА, то варто зауважити, що транспортний засіб може виглядати величезним на одному знімку та крихітним на іншому. У такому випадку базові згорткові мережі стикаються з проблемою розпізнавання. Оскільки вони застосовують карту ознак

лише одного масштабу, нехтуючи величезними та малими об’єктами. Тому використовують *FPN*, яка формує набір карт ознак у різних масштабах та поєднує контекстну і локальну інформації. Архітектура *FPN* зображена на рисунку 2.8.

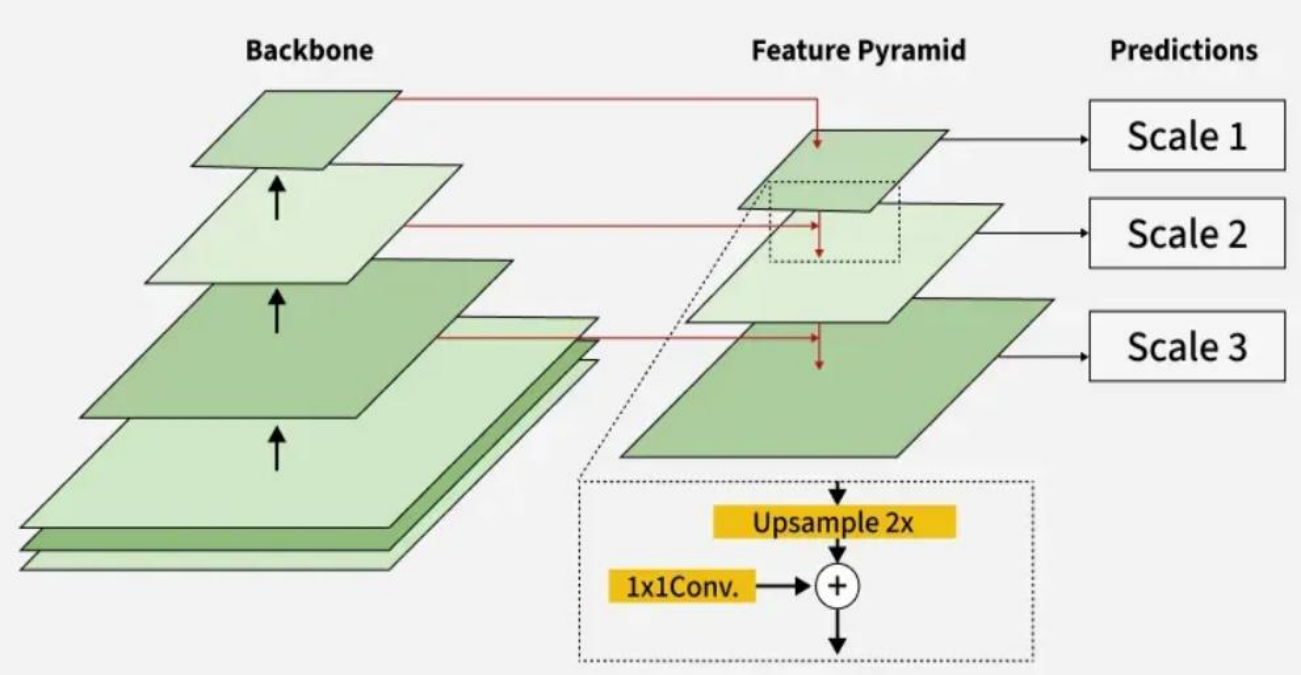


Рисунок 2.8 – Архітектура *FPN*

Робота FPN базується на поєднанні семантично сильних ознак малої роздільної здатності з семантично слабкими ознаками високої роздільної здатності. Спочатку відбувається стандартний прохід знизу догори. У ході проходження кадру через послідовні шари результуючі карти ознак зменшуються в просторовому розмірі, але при цьому збільшуються в семантичній глибині. Потім мережа підвищує карту ознак найглибшого шару. В ході відновлення більших карт ознак низхідним шляхом відбувається поєднання їх з відповідними картами ознак з висхідного шляху. На виході отримується піраміда карт ознак, яка багата на семантику та просторові деталі.

Припустимо, що необхідно виявити танк на вхідному кадрі. Тоді FPN працюватиме наступним чином:

1) найвищий рівень визначає форму транспорту;
2) середній рівень акцентує увагу на башту та гусениці;

3) найнижчий рівень зосереджується на дуло, маркування і тому подібне.

Поєднавши всі ці рівні мережа точно знайде шуканий об’єкт, не зважаючи на його розміри .

До переваг можна віднести:

1) використання функції високої роздільної здатності для виявлення малих об’єктів;

2) обробляє великі та малі об’єкти на одному кадрі;

3) використовує основні функції повторно для зменшення обчислювального навантаження;

4) сумісність з відомими детекторами.

2.2.2 *Context Modeling*

Контекстна інформація може являтися будь-якими візуальними або невізуальними даними. Ці дані можуть містити інформацію про зовнішній вигляд об’єкта (колір, розмір, текстура та подібне), місце розташування, зв’язок між різними об’єктами та будь-яка інформація, що допоможе ліпше зрозуміти середовище [26]. Загальну класифікацію контексту наведено на рисунку 2.9.

Визначення патернів та значущих структур на основі контексту це цілодобова робота людського зору. Незважаючи на перешкоди, розмиття, зміни освітлення чи погоди, люди можуть швидко та вправно визначати візуальне середовище, спираючись на контекстну інформацію. Наприклад, якщо видно частково затемнений об’єкт, то визначити його можна за допомогою аналізу фону, взаємодії з іншими об’єктами та за подібними ознаками.

Комп'ютерний зір намагається відтворити людський зір, щоб використовувати контекстну інформацію для кращого розуміння вхідного кадру. Коли комп’ютер робить спробу виявити об’єкт на зображенні чи відео, то він має провести пошук у кожній частині кадру та у різних масштабах. Такий підхід займає багато обчислювальних ресурсів. Якщо мати уявлення про зовнішній вигляд та можливе місцезнаходження об’єкта, то машина може скоротити область пошуку, ігнорувати не потрібні об’єкти та сконцентрувати увагу на областях, де очікується шуканий об’єкт. Таким чином процес розпізнавання значно пришвидшується та

покращується. Під час визначення техніки з камер БПЛА часто частини шуканих об’єктів затемнені або невидимі через погодні умови, погану якість зйомки і так далі. Однак, враховуючи інші частини шуканої цілі (дуло, загальна форма, позначення на кузові) можна ідентифікувати і сам об’єкт [27].

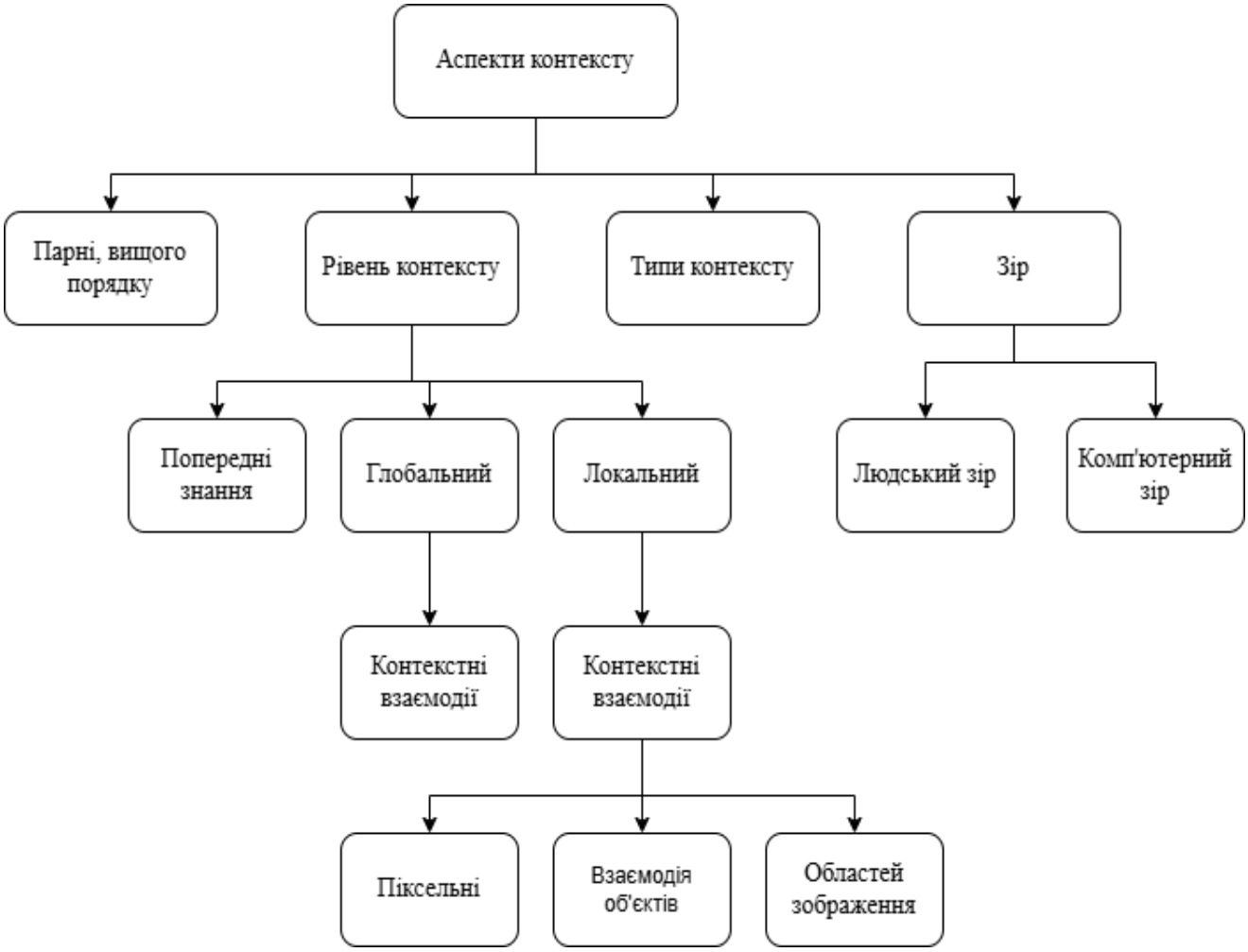


Рисунок 2.9 – Загальна класифікація контексту

Контекст можна класифікувати на три основні рівні: попередні знання, локальний та глобальний. Попередні знання це та інформація, яку можна отримати до спостереженням за середовищем. Такі дані дозволяють спрогнозувати настання певних подій або появу певного об’єкту. Наприклад, під час спостереження польових територій можна побачити людей, техніку, але ймовірність побачити жирафа близька до нуля. На рисунку 2.10 зображено прикладне застосування глобального та локального контекстів.

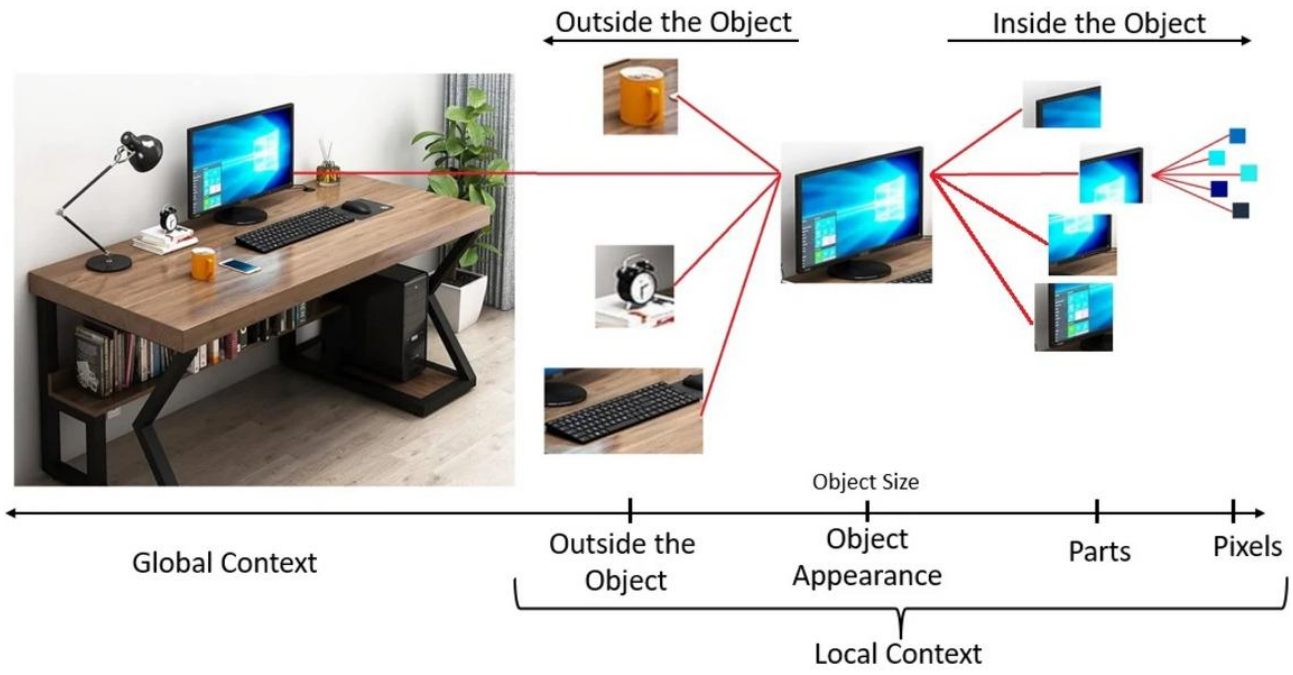


Рисунок 2.10 – Глобальний та локальний контексти [28]

Локальний контекст охоплює інформацію про цільовий об’єкт та область, що його щільно оточує. Він використовується для розпізнавання об’єктів, які мають значну візуальну та структурну схожість.

Використання локального контексту може бути неоптимальним для розпізнавання малих за розміром об’єктів. В такій ситуації глобальний контекст буде доречнішим. Він охоплює більшу область зображення або навіть весь кадр для отримання цінної інформації.

При виявленні важкої техніки в межах кадру глобальний контекст може аналізувати усе зображення для виявлення іншої техніки, слідів від гусеничної стрічки, будь-якої інформації, що може допомогти ідентифікувати об’єкт на зображенні.

Варто зауважити, що використовувати лише глобальний контекст для виявлення об’єкту може призвести до плутанини. Тому гарним варіантом буде об’єднання локального та глобального контекстів для покращення знаходження цільового об’єкту з кадру.

2.2.3 *Multi-Scale training*

Людський зір має можливість фокусувати світло за допомогою війкових м’язів. Для того, щоб мати дану функцію в камері, у ній є фокусний об’єктив за допомогою якого можна фокусуватись на частині кадру. У випадку використання кадрів для машинного навчання дана функція практично недоступна. І це знижує точність алгоритму навчання. Для того, щоб подолати такий недолік застосовується багато масштабне навчання.

Багато масштабне навчання – це метод, який застосовується при розпізнаванні об'єктів для підвищення стійкості та узагальнення навченої моделі. Воно передбачає навчання моделі на кадрах різних розмірів, що дозволяє їй вірно виявляти об'єкти в різних масштабах. Вхідне зображення змінюється у розмірах до певного діапазону вимірів (рис. 2.11). З отриманого набору змінених кадрів обрізається картинка фіксованого розміру. Якщо на одному зображенні є багато ключових ознак, то обрізається декілька зображень, що фокусуються на вагомих ознаках. Таким чином отримується новий набір даних, який містить в собі дрібні деталі цільового об’єкта. Завдяки якому модель не буде пропускати об’єкти малих розмірів [29].

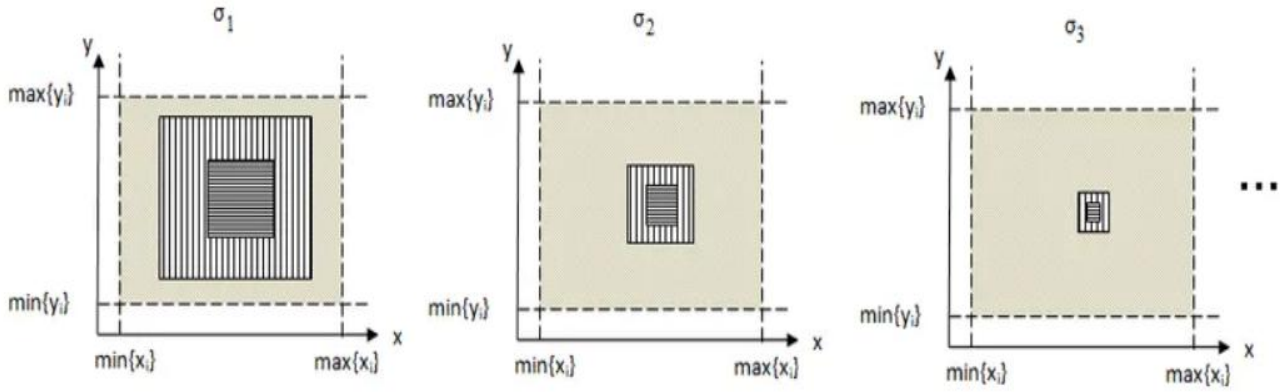


Рисунок 2.11 – Багаторазове масштабування зображення [30]

За замовчуванням у *YOLOv*8 використовується одно масштабне навчання 640 на 640. Однак, існує декілька варіантів реалізації багато масштабного навчання.

Розглянемо найвідоміші підходи:

1) вихідне зображення має мінливий масштаб, а різні масштаби вхідних зображень поповнюються тим самим масштабом;

2) вихідне зображення має фіксований масштаб і виконується вибірка вгору та вниз пакетів зображень для безпосереднього багато масштабного навчання.

Перший підхід може згенерувати багатші масштаби, але саме навчання проходить не ефективно. Другий підхід має більш ефективне навчання через незалежне доповнення одного зображення. Тому часто застосовується останній підхід для реалізації багато масштабного навчання.

2.3 Супровід об’єктів у відео

Відстеження декількох об’єктів (*MOT*) – це одне з завдань комп’ютерного зору, яке містить виявлення та відстеження цільових об’єктів з відеопотоку. Даний процес відбувається без попереднього знання про зовнішній вигляд об’єктів та їх розташування [31]. Під час відстеження необхідно не тільки ідентифікувати, а й пов’язувати виявлення одного об’єкта між кадрами. Результатом роботи *MOT* є утворений набір даних, який складається з треків.

Трек вміщає в собі усі виявлення, що стосуються певного об’єкта. Завдання відстеження стає все складнішим, коли необхідно знайти загальне рішення. Головною проблемою можна відзначити зміну ідентифікатора та втрату об’єкта через часті перекриття.

Окрім того, цілі з подібним зовнішнім виглядом значно ускладнюють їх розрізнення, а непередбачуваний динамічний рух додає складності під час відстеження. Навіть при такій непростій задачі *MOT* являється важливим у багатьох галузях, які потребують аналізу відео в режимі реального часу.

SORT – простий алгоритм для відстеження об’єктів у реальному часі. Він містить модель виявлення, метод асоціацій та обробку відстеження. Для підвищення продуктивності можна додатково використати моделі руху та

зовнішнього вигляду. *SORT* може використовувати будь-які сучасні детектори, наприклад *R-CNN*, *YOLO*. Сам трекер містить важливий гіперпараметр $det_{\square}$. Даний параметр врівноважує хибнопозитивні та хибнонегативні результати. Вищий поріг зменшує кількість хибнопозитивнтх результатів, але збільшує кількість хибнонегативних, і навпаки [32].

SORT використовує модель оцінки фільтра Калмана для прогнозування руху обмежувальної рамки та для фільтрації шуму.

Також застосовується лінійна модель постійної швидкості для прогнозування руху об’єкта. Дане прогнозування використовується під час асоціації для пов’язання треку з виявленням.

Створення та видалення треків повинно відстежувати об’єкти, які з’являються, зникають та повертаються на кадрі. Усі неасоційовані виявлення являються потенційно новими треками з індивідуальним ідентифікатором. Кожен новий трек повинен пройти випробувальний період. Якщо протягом випробувального періоду було пропущено один кадр, то новий трек видаляється. Існуючі треки завершуються, якщо вони не виявляються протягом невеликого періоду.

Deep SORT являється розширеним алгоритмом *SORT*. Він містить метрику глибокої асоціації на основі ознак зовнішнього вигляду. Це дозволяє алгоритму аналізувати ситуації, коли об’єкти частково затьмарені у відеопотоці. *Deep SORT* спочатку генерує виявлення об’єктів, а потім зв’язує ці виявлення з існуючими треками [33]. Архітектура *Deep SORT* зображена на рисунку 2.12.

До переваг згаданої архітектури можна віднести:

1) здатність повторно ідентифікувати об’єкти, які були тимчасово перекриті;
2) здатність розрізняти схожі об’єкти;
3) робота у реальному часі;
4) гнучкість.

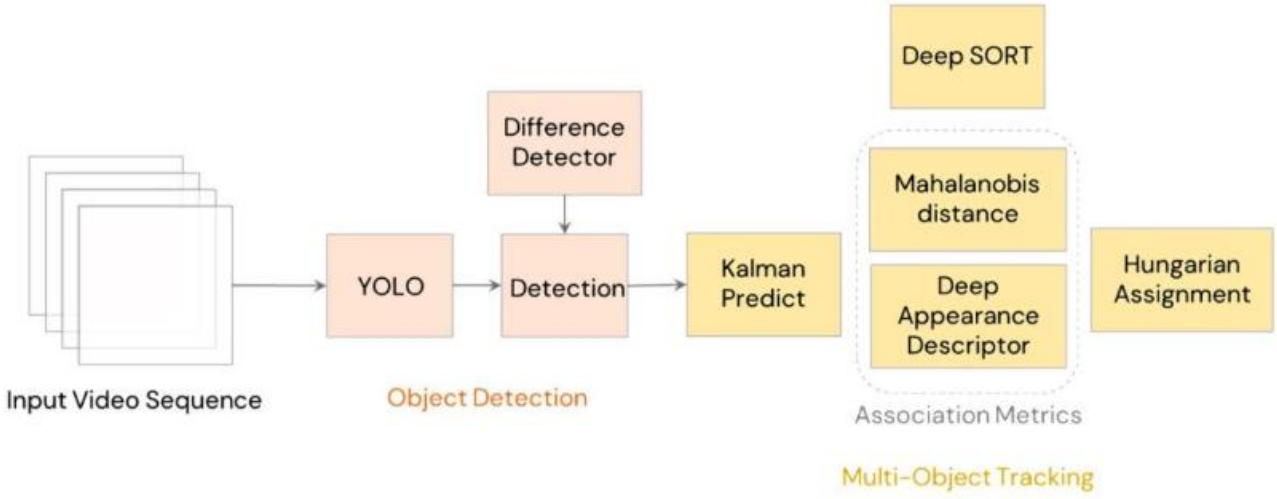


Рисунок 2.12 – Архітектура *Deep SORT*

Ще одним популярним алгоритмом відстеження об’єктів є *ByteTrack*. Фактично даний алгоритм починає свою роботу подібно з вище розглянутими. Однак *ByteTrack* використовує механізм стробування для фільтрації надлишкових даних та дещо покращений процес асоціації даних [34].

Процес асоціації починається зі зіставлення блоків виявлення з великою достовірністю з треками. Це забезпечує правильне поєднання достовірних виявлень з коректними треками, а також зменшується ймовірність плутаниці з ідентифікаторами об’єктів. Потім решту блоків виявлення зіставляє з треками, порівнюючи їх на подібність. Подібність визначається на основі значень перекриття блоків та косинусної подібності ознак зовнішнього вигляду.

ByteTrack має високі показники точності відстеження кількох об’єктів та оцінку ідентифікації *F*1.

OC-SORT доволі популярна вдосконалена версія оригінального алгоритму *SORT*. Нова версія алгоритму ставить виявлене положення об’єктів у кожному кадрі в центр своєї розробки. Також з’явились дві основні ідеї. Коли об’єкт зникає за чимось, а потім знову з’являється, то *OC-SORT* створює уявний шлях. Даний шлях з’єднує місце, де його бачили востаннє, з місцем, де він знову показався. Отриманий напрям використовується для налаштування стану трекера так, ніби об’єкт і не зникав з кадру.

Другою ідеєю являється спостережувано-центричний імпульс. *OC-SORT* розглядає фактичний рух між останнім спостереженням, щоб визначити реальні показники напрямку та швидкості об’єкту [35]. Таким чином трекер може адекватно реагувати на нелінійні рухи.

2.4 Набори даних та анотація

Набори даних відіграють значну роль при навчанні систем розпізнавання. У випадку виявлення дрібних наземних цілей із бортової камери БПЛА потрібні набори даних, що містять змінну висотку, великі кути огляду, різку зміну масштабу та подібні особливості, які властиві знімкам з дрону. Тож розгляну сформовані датасети, які застосовують при навчанні системи для покращення розпізнавання об’єктів з кадру.

Набір даних *VisDrone* зібраний командою *AISKYEYE*. Він містить понад 300 відео, 260 тис. кадрів та 10000 статичних зображень, які були зняті з дронів. Даний датасет охоплює різноманітні аспекти: місцезнаходження, навколишнє середовище, об’єкти, щільність. Важливо зауважити, що набір збирався за допомогою різних платформ БПЛА, погодних умов та освітлення.

UAVDT містить близько 80000 кадрів, які отримали з 10 годинного необробленого відео. Даний набір даних виявився доволі вдалим для навчання систем розпізнавання об’єктів. Оскільки він містить зображення, які були зроблені при різних умовах: погодні, кут нахилу зйомки, освітленість, висоти. Також отриманий набір містить крихітні об’єкти, які займають декілька пікселів.

DOTA – це великомасштабний набір даних для виявлення об'єктів на знімках зроблених з повітря. Такий датасет варто застосовувати для розробки та оцінки детекторів об'єктів на аерофотознімках. Усі кадри були отримані з різноманітних датчиків та платформ. Зображення містять об’єкти, що охоплюють велике різноманіття масштабів, орієнтацій та форм.

xView є ще одним великим та загальнодоступним набором даних, який містить зображення зроблені з висоти. Такий датасет вміщає в себе складні сцени по всьому світу, анотовані за допомогою обмежувальних рамок.

Приклад анотованих зображень можна побачити на рисунку 2.13.

Рисунок 2.13 – Анотовані зображення

Анотування даних – це процес додавання міток до початкової інформації, що робить її придатною для алгоритмів машинного навчання. Даний процес потребує втручання людини для позначення певних особливостей на наборі даних [36].

Зображення анотуються з використанням обмежуючих рамок, сегментаційних масок або міток, що допомагають нейронній мережі розпізнати та класифікувати об’єкти.

Анотовані дані слугують фундаментом для навчання нейронної мережі. В ході навчання алгоритми запам’ятовують закономірності, які використовують в

подальшому розпізнаванні об’єктів. Також такі дані застосовують при оцінюванні продуктивності та точності створеної системи. Якщо розпізнавання виконується не якісно, то за допомогою анотованої інформації можна покращити точність прогнозів.

Висновок до розділу

На початку другого розділу роботи було розглянуто архітектури та проведено порівняльний аналіз сучасних детекторів таких, як: *YOLO, SSD, R-CNN* та *Faster R-CNN*. Виявилось, що одноступеневі детектори є гарним рішенням для бортового застосування завдяки високому *FPS* та енергоефективності.

В свою чергу, двоступеневі детектори не підходять для використання на борту БПЛА через свою складність та низький *FPS*. В рамках розпізнавання важкої техніки з кадру найкращою моделлю виявилась *YOLO* і саме вона обрана для подальшої реалізації системи розпізнавання.

Проаналізовано проблематику та способи розпізнавання дрібних цілей. FPN створює піраміду ознак за якою визначає об’єкти різного розміру з кадру без істотного здорожчання обчислення. *Context Modeling* може визначати малі об’єкти, спираючись на його локальне та глобальне знаходження. *Multi-Scale training* дозволяє мережі фокусуватись на частині зображення, де знаходяться ознаки цільового об’єкту. Тож при застосуванні даних підходів можна якісно проводити розпізнавання малих об’єктів з камер БПЛА.

На останок розглянуто загальнодоступні набори даних для навчання та оцінки працездатності системи розпізнавання. Вони мають покривати варіативність ракурсів, висот, погодних умов, масштабів для створення бажаної системи з мінімальними похибками.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ОЦІНКА АВТОМАТИЗОВАНОЇ СИСТЕМИ РОЗПІЗНАВАННЯ

3.1 Постановка задачі

Попередньо було проаналізовано теоретичну інформацію, яка пов'язана з розпізнаванням та ідентифікацією об'єктів із використанням глибинних нейронних мереж. Також розглянуто особливості застосування методів комп’ютерного зору в аеророзвідці. В даному розділі необхідно реалізувати автоматизовану комп’ютерну систему розпізнавання об’єктів з камери БПЛА, яка забезпечує зручну взаємодію з оператором та працює в режимі обробки зображень і відеоматеріалів. З огляду на заборону використання БПЛА через надзвичайний стан у країні, система буде реалізована як наземний програмний комплекс із веб-інтерфейсом, що відтворює роботу модуля розпізнавання, підключеного до бортової камери безпілотного літального апарата. Очікуються, що вхідними даними будуть зображення та відеофайли отримані з реальних польотів БПЛА. Кінцевим результатом роботи створеної системи є анотовані зображення та відео (виявленні об'єкти будуть позначені рамками із відповідними підписами). Також має формуватися текстовий перелік детекцій.

Розроблювана система повинна забезпечувати можливість завантаження оператором зображень та відеофайлів через веб-інтерфейс. Передбачається підтримка спрощеного режиму роботи в якому головною задачею є виявлення танків, а також розгорнутого режиму у якому виконується детекція різних типів військових об’єктів на основі багатокласової моделі.

Для завантаженого зображення система має використати обрану модель детекції, одержати список ідентифікованих об'єктів, координат обмежувальних рамок та значень довіри. Додатково відбувається формування анотованого зображення з візуальним виділенням об'єктів.

При обробці відеоматеріалів система повинна здійснювати покадрову обробку вхідного відеопотоку, використовуючи обрану модель розпізнавання. Вихідним результатом являється формування анотованого відео у якому відображається знайдені об'єкти з відповідним позначенням. Користувач повинен мати змогу завантажити результат роботи для подальшого аналізу або приєднання до звітних документів.

Важливою вимогою являється виконання базових механізмів логування. Система повинна реєструвати події запуску моделей, відомості про вхідні дані, час обробки та наявні помилки (за їх наявності). Даний підхід дозволяє аналізувати продуктивність, вчасно усувати недоліки та вдосконалювати програмний комплекс.

Так як система розрахована на застосуванні в аеророзвідці, то до неї висувають вимоги не тільки щодо функціональності, а й щодо якості розпізнавання, швидкодії та зручності експлуатації. Якщо звертати увагу на якість детекції, то програмний комплекс повинен забезпечувати прийнятний рівень точності та повноти для ключових класів об'єктів (акцент на танки). Оцінювання виконується за *Precision*, *Recall* і інтегральних показників *mAP@*0.5 та *mAP@*0.5:0.95. Такий метод оцінювання дає змогу порівняти різні конфігурації моделей і режими роботи.

На рахунок швидкодії система повинна обробляти зображення в межах від сотень мілісекунд до кількох секунд залежно від роздільної здатності. Для обробки відео загальний час формування анотованого відеофайлу також має бути адекватним. Для цього застосовуються оптимізації, такі як масштабування кадрів, обробка не кожного кадру, а лише певної їх підмножини.

Програмний комплекс також повинен бути зручним. Зручність використання передбачає інтуїтивно зрозумілий веб-інтерфейс, чітким розділенням режимів роботи, відображенням результатів у вигляді рамок і підписів на зображенні та

доступним текстовим переліченням детекцій. Оператор повинен мати змогу змінювати файли та переключати режими, не перезапускаючи застосунок.

Вагомим є надійність та стійкість до помилок. У разі завантаженні некоректних або пошкоджених файлів система повинна видавати інформативні повідомлення, але ні в якому разі не завершувати роботу аварійно.

При практичній реалізації автоматизованої системи розпізнавання об’єктів з камери БПЛА варто враховувати обмеження, які пов'язані з специфікою отримання та обробки аеророзвідувальних даних. Доволі часто знімки з бортових камер БПЛА містять значні геометричні спотворення через зміни висоти й кута зйомки, розмиття через рух платформи, різкі зміни освітлення, тіні, дим та тому подібне. Усе це суттєво ускладнює задачу детекції та вимагає підбору рхітектур моделей, навчальних вибірок і режимів їх донавчання.

Додатковим фактором є обмеження обчислювальних ресурсів. У реальних умовах модуль розпізнавання використовується на борту БПЛА, де ресурси енергії й обчислень обмежені. Через це необхідно створити два режими роботи: перший буде швидшим, але менш точним, а другий більш точний, але потребує більше обчислювальних ресурсів.

Ключовою особливістю є орієнтація системи на розпізнавання бронетехніки, військових автомобілів та інших елементів військової інфраструктури. Візуальні характеристики зазначених об'єктів значно відрізняються від об’єктів загального призначення, що ускладнює використання універсальних відкритих датасетів. Таким чином виникає потреба у поєднанні власних *UAV*-даних із спеціалізованими наборами та виконанні донавчання нейронних мереж з урахуванням конкретних умов аерозйомки.

3.2 Програмне середовище та інструменти розробки

При написанні автоматизованої системи було використано *Python*, як основну мову програмування. Даний вибір зумовлений широкою підтримкою мови у галузі машинного навчання та комп'ютерного зору, наявністю купи

спеціалізованих бібліотек, а також доволі знайомим синтаксисом. Розроблювана робота виконувалася у межах ізольованого віртуального середовища *venv*, що дає змогу уникати конфліктів версій та забезпечувати відтворюваність поставленого завдання.

Під час реалізації модулів детекції застосовано бібліотеку *Ultralytics YOLOv*8, яка базується на фреймворку *PyTorch* і вміщає в собі високорівневий інтерфейс до сучасних моделей сімейства *YOLO*.

Використовувалась бібліотека *OpenCV* для попередньої підготовки та візуалізації результатів у вигляді зображень та відео. Вона дає змогу зчитувати й записувати медіафайли, коригувати роздільну здатність, робити перетворення кольорових каналів та відображати обмежувальні рамки й підписи. Бібліотека *NumPy* була обрана для роботи з масивами даних, координатами боксів та результатами обчислень. Дана бібліотека чудово виконує операції над багатовимірними масивами та матрицями.

Для серверної частини системи було використано веб-фреймворку *FastAPI*. Він підтримує асинхронну обробку *HTTP*-запитів, автоматично створює документацію *OpenAPI* та легко інтегрується з *Python*-бібліотеками для машинного навчання. Клієнтська частина система реалізована з використанням найпопулярніших веб-технологій: *HTML, CSS* та *JavaScript*. За допомогою *HTML*-розмітки формуємо структуру інтерфейсу. Це стосується до форми завантаження зображень та відео, селектору вибору моделі розпізнавання, блоків для відображення анотованих результатів і списку детекцій. *CSS* використаний для візуального оформлення сторінки. Мова *JavaScript* використовується для переміщення отриманих даних на сервер через *AJAX*-запити, обробки відповідей *FastAPI*, динамічного оновлення зображень без перезапуску сторінки та створення посилання для завантаження анотованого відео.

Безпосередня розробка програмного забезпечення проводилася в операційній системі *Windows* із використанням середовища розробки *Visual Studio Code*. Обране середовище підтримує синтаксис *Python*, роботу з віртуальними середовищами,

інтеграцію з системою контролю версій *Git*, а також містить інтуїтивно зрозумілі засоби перегляду структури проєкту та налагодження коду.

Основна структура програмного проєкту наведена на рисунку 3.1.

У папці *app* міститься основна логіка серверної частини. Файл main.py містить код, який створює об'єкт *FastAPI* та опис кінцевих точок для обробки зображень і відео, а також реалізує вибір моделі та повертає результати детекції. Файл *detector.py* реалізує клас детектора. Він відповідає за завантаження навчених моделей *YOLOv*8, попередню обробку отриманих даних та формування структурованого результату.

Рисунок 3.1 – Основна структура проєкту

Папка *image-innovators* вміщає в собі вихідні дані для навчання спрощеної (акцентована увага на розпізнавання танків) та багатокласової моделів. Якщо переглянути вміст підкаталогу *runs/detect/*, то можна побачити збережені

результати навчання моделей (*train*, *train*2, *train*3, *train*4), зокрема ваги *best.pt*, журнали навчання та сформовані метрики.

Статичні ресурси клієнтської частини розміщено в папці *static*. Файл *styles.css* відповідає за оформлення інтерфейсу, а script.js реалізує взаємодію з *API* серверної частини. Шаблон *HTML*-сторінки index.html знаходиться у окремій папці templates на верхньому рівні проєкту.

3.3 Підготовка даних та навчання моделей детекції

Доволі важливим етапом при розробці автоматизованої системи являється коректна підготовка даних та безпосередній процес навчання моделей детекції. Від різновиду та якості навчальних вибірок залежать здатність моделі працювати у складних умовах аерозйомки та відповідно надійність роботи при реальній експлуатації.

Спочатку було підготовано власний *UAV*-датасет(з відео доступних у мережі) для навчання спрощеної моделі. В ньому містяться кадри танків з бортової камери безпілотного літального апарата (рис. 3.2).

Рисунок 3.2 – Кадр танку отриманий з бортової камери БПЛА

Отримані зображення зберігаються у вигляді послідовності файлів з назвами *frame*_0, *frame*_1, ..., *frame_n* в окремих папках для навчання та тестування. Формат зображень являється стандартним для задач комп'ютерного зору: *JPEG*. Варто зауважити, що роздільна здатність відповідає кадрам отриманих з БПЛА.

До отриманого набору даних було сформовано анотації, які містяться у файлі *annotations.json*. Кожен запис у даному файлі відповідає лише одному зображенню та містить набір полігонів, які описують контури об'єктів класу танк. Ідентифікатор кадру зв'язується з масивом вершин полігона у вигляді пар координат (*x*, *y*) у пікселях. Даний формат доволі зручний для ручної розмітки, але він не підтримується бібліотекою *Ultralytics YOLO*. Дана бібліотека потребує анотації у вигляді прямокутних обмежувальних рамок (рис. 3.3.). Тому отримані анотації варто перетворити у формат, який сумісний з *YOLO*.

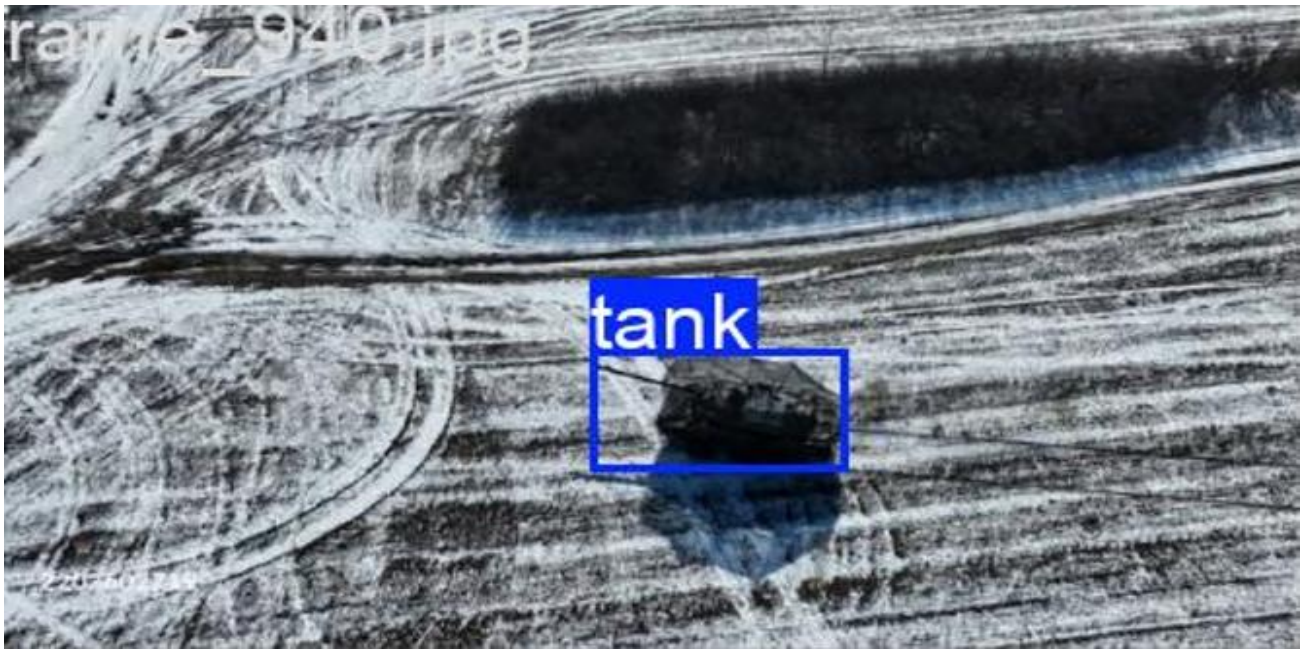


Рисунок 3.3 – Анотація кадру у вигляді прямокутної обмежувальної рамки

З метою підвищення якості розпізнавання та здатності моделі працювати з різними типами військових цілей було використано декілька публічних наборів даних. Новий набір даних охоплює дванадцять класів: солдат (також розпізнає військову особу у камуфляжі), зброя, танк, вантажівка, військовий автомобіль, цивільний, автомобіль, артилерія, окоп, літак, корабель.

Перевагою використання уже створених наборів даних є те, що анотації уже надані у форматі *YOLOv*8. Приклад кадрів з багатокласового набору даних наведено на рисунок 3.4.

Рисунок 3.4 – Приклади кадрів з багатокласового набору даних

Так як власний *UAV*-датасет містив лише один клас, а сформований (з готових наборів даних) багатокласовий датасет дванадцять, то виникла задача узгодження схеми маркування. Для спрощеної моделі було акцентовано увагу на класі танк, тоді як для багатокласової моделі зберігалася повна номенклатура усіх дванадцяти класів. Тож для кожної з моделей було зіставлено ідентифікатори класів з логічними назвами та проведено зміну нумерації таким чином, щоб уникнути конфліктів і забезпечити єдину схему кодування в межах обраного набору.

Після ручної анотації власного *UAV*-датасету було написано скрипт конвертації полігональних анотацій з annotations.json у прямокутні обмежувальні рамки. Для кожного полігона обраховувався мінімальний охоплюючий прямокутник, координати рамки нормалізувалися відносно розмірів кадру і записувалися у вигляді: номер класу, *x*, *y*, ширина, висота.

В результаті конвертації для кожного кадру отримався текстовий файл анотації з розширенням *.txt*, назва якого збігається з назвою відповідного зображення.

Далі необхідно було організувати структуру каталогів відповідно до вимог *Ultralytics*. Тож для кожного набору створювалися підкаталоги *images*/*train*, *images*/*val*, *images*/*test* та відповідні їм *labels*/*train*, *labels*/*val*, *labels*/*test*. Розбиття на *train*, *validation* та *test* виконувалося таким чином, щоб забезпечити репрезентативність валідаційної та тестової вибірок і водночас зберегти достатній обсяг даних для навчання. Для опису набору даних використовувався *YAML*-файл. Він містить шляхи до каталогів із зображеннями, кількість класів та їх назви. Аналогічні дії виконувались і з другим набором даних. Багатокласовий набір уже частково відповідав даному стандарту, але вимагав уточнення шляхів.

Перше навчання проводилось для спрощеної моделі, яка повинна виявляти лише танки. Вихідна архітектура була обрана *YOLOv*8. Вона поєднує високу швидкодію та достатню точність для задачі детекції на аерозображеннях. Для ініціалізації ваг використовувалися заздалегіть навчені ваги базової моделі, а подальше навчання виконувалося в режимі донавчання на підготовленому *UAV*-датасеті. Процес навчання передбачав багатоепохову оптимізацію з поступовим зменшенням функцій втрат для локалізації об’єктів, класифікації та регресії координат.

Для того, щоб оцінити якість моделі були використані стандартні показники задачі детекції: *Precision*, *Recall*, *mAP@*0.5 та *mAP@*0.5–0.95.

В кінці навчання найкращі ваги моделі було збережено у вигляді окремого файлу. Саме даний файл надалі інтегрувалася в програмний модуль детектора та використовувалася в спрощеному режимі для обробки зображень і відео.

Другим етапом було навчено розширену модель, яка використовується для розпізнавання ширшого набору військових об’єктів. Формування навчальної, валідаційної та тестової вибірок, а також опис датасету в *YAML*-конфігурації здійснювалися за аналогічним принципом, як і у першому навчанні. Збережені ваги багатокласової моделі було інтегровано в загальний модуль детекції як додатковий.

У програмній реалізації візуально можна прослідкувати наявність двох режимів роботи системи (спрощена та багатокласова). Даний підхід дозволяє адаптувати систему до конкретних розвідувальних задач.

3.4 Архітектура програмної реалізації системи

Архітектура розробленої системи розпізнавання об’єктів з бортової камери БПЛА побудована за клієнт–серверним принципом. Всі операції детекції виконуються на сервері, де завантажуються та працюють моделі *YOLO*. В свою чергу оператор взаємодіє із системою через веб-інтерфейс. Даний підхід дозволяє спростити розгортання та простіше масштабується. Ще одним важливою перевагою такого підходу є те, що присутня змога замінити чи донавчити модель без зміни клієнтської частини.

В загальному вигляді повний цикл обробки даних розпочинається з того, що оператор через веб-інтерфейс завантажує зображення або відео та обирає режим роботи детектора. Наступним кроком браузер за допомогою *JavaScript* надсилає *HTTP*-запит до серверного застосунку *FastAPI*. Сервер відправляє вхідні дані в модуль детектора, де обрана модель *YOLO* виконує розпізнавання та створює анотований результат. В кінцевому результаті анотовані відео або кадр повертається у браузер.

Рух потоку даних від оператора до моделей *YOLO* та назад наведено на рисунку 3.5.На верхньому рівні система містить дві основні частини: клієнтської та серверної. Клієнтська частина являє собою веб-інтерфейс через який працює оператор. У ньому відображається сторінка з формами завантаження зображень та відео, селектором режиму роботи детектора та блоками для показу результатів.

Клієнтська логіка реалізована у файлі *script.js* і відповідає за отримання даних із форм, надсилання запитів до серверу та оновлення сторінки згідно до отриманої відповіді.

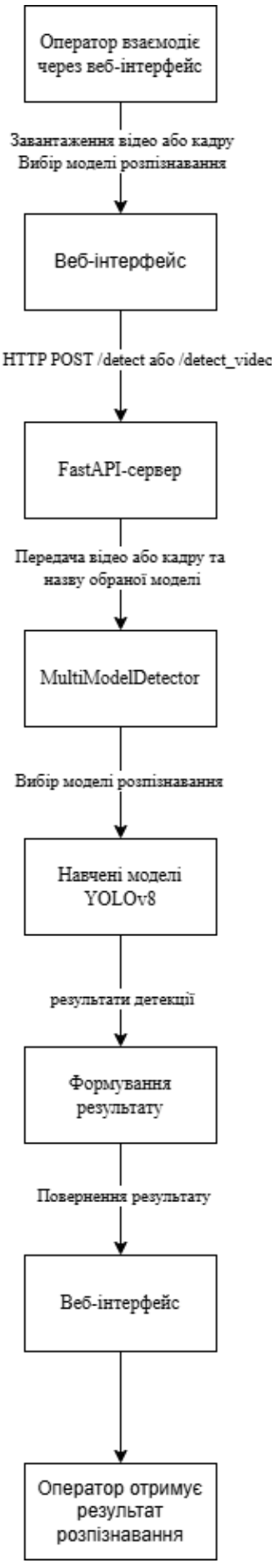


Рисунок 3.5 – Рух потоку даних від оператора до моделей *YOLO* та назад

Серверна частина вміщає в себе *API*-сервіс та модулі детекції. Вона реалізована у модулі main.py. В даному модулі створюється екземпляр застосунку *FastAPI*, описуються *HTTP*-ендпоінти для обробки зображень та відео, а також виконується створення екземпляра модуля детектора, який завантажує навчені моделі *YOLO*. Як тільки подається запит від оператора *FastAPI* приймає файл, зчитує його в пам’ять, отримує значення обраної моделі розпізнавання, а далі передає обробку даних до модуля *MultiModelDetector*.

Модуль детектора містить спрощену та багатокласову моделі *YOLOv*8. Вибір конкретної моделі для розпізнавання об'єктів з кадру здійснюється на основі положення селектору, що надійшов від користувача.

Кінцевим результатом детектора являється структура даних, що містить анотоване або відео та список ідентифікованих класів об'єктів. Безпосередньо отримані результати відображаються у клієнтській частині, а також надається змога завантажити анотоване відео для більш детального аналізу.

Основним елементом серверної логіки є модуль детектора, реалізований у файлі detector.py. У межах цього модуля визначено клас, який інкапсулює завантаження та використання декількох моделей *YOLO*. Принцип роботи класу полягає в тому, що під час ініціалізації завантажуються ваги спрощеної моделі та багатокласової моделі. Додатково задаються загальні параметри детекції, такі як поріг упевненості та поріг *NMS*. Приклад ініціалізації класу *MultiModelDetector*:

class MultiModelDetector:
def __init__(*self, conf_tank: float* = 0.25, *conf_multi: float* = 0.25):
self.conf_tank = conf_tank
self.conf_multi = conf_multi
print(">>> Завантажую спрощену модель:", *TANK_MODEL_PATH*)
self.tank_model = YOLO(str(TANK_MODEL_PATH))
print(">>> Класи спрощеної моделі:", *self.tank_model.names*)
print(">>> Завантажую багатокласову модель:", *MULTI_MODEL_PATH*)
self.multi_model = YOLO(str(MULTI_MODEL_PATH))

print(">>> Класи багатокласової моделі:", *self.multi_model.names*)

Спираючись на це, усі моделі детекції зосереджені в одному місці. Серверна частина звертається до узагальненого інтерфейсу *MultiModelDetector*. При обробці запиту метод детектора отримує назву моделі, яку необхідно використовувати для розпізнавання. У відповідь клас обирає відповідний екземпляр *YOLO* з внутрішнього словника *models*, передає йому кадр або відео та отримує результат детекції.

Даний підхід має декілька переваг:

1) можна легко додати нові моделі;

2) код серверної частини залишається компактним і не залежить від конкретних шляхів до файлів ваг;

3) просто організувати порівняльні експерименти між моделями.

Практична реалізація вибору моделі розпізнавання:

```
def detect_on_image(self, image_bytes: bytes, mode: str = "multi") -> Dict[str, Any]:
    mode = mode.lower()
    if mode == "tank":
        return self._detect(self.tank_model, image_bytes, self.conf_tank)
    else:
        return self._detect(self.multi_model, image_bytes, self.conf_multi)
```

Загальний цикл обробки окремого кадру можна поділити на декілька послідовних етапів:

1) прийом вхідних даних від оператора;

2) конвертація у внутрішній формат;

3) виконання розпізнавання об'єктів;

4) анотація кадру;

5) формування кінцевого результату.

На рівні *FastAPI*-застосунку метод-обробник ендпоінта *detect* приймає файл типу *UploadFile*, зчитує його в байтовий масив та передає отримані байти в необхідний метод класу *MultiModelDetector*.

Обробка запиту *detect* у модулі main.py реалізована наступним чином:

```
@app.post("/detect")
async def detect(
    file: UploadFile = File(...),
    mode: str = Form("multi"),
):
    image_bytes = await file.read()
    result = detector.detect_on_image(image_bytes, mode=mode)
    annotated_image = result["annotated_image"]
    detections = result["detections"]
    jpeg_bytes = detector.encode_image_to_jpeg_bytes(annotated_image)
    image_b64 = base64.b64encode(jpeg_bytes).decode("utf-8")
    return JSONResponse(
        {
            "image_base64": image_b64,
            "detections": detections,
        })
```

У модулі *detector.py* прописаний метод *detect_on_image* в якому зображення перетворюються на масив пікселів у форматі *OpenCV* за допомогою функцій *numpy.frombuffer* та *cv2.imdecode*. Далі проводиться перевірка на успіх декодування кадру. Потім обирається відповідна модель *YOLO* (відповідно параметру *mode*). Результатом розпізнавання є прямокутні координати навколо знайденого об'єкта, значення впевненості та індекс класу.

Наступним кроком відбувається фільтрація об’єктів за порогом впевненості та побудова анотованого зображення. Малюються прямокутні рамки та підписи (назва класу, числове значення впевненості) для усіх об'єктів, які пройшли зазначений поріг. Паралельно формується структурований список розпізнавань, який містить ідентифікатор класу, текстова мітка, значення впевненості та координати рамки у пікселях.

Після завершення процесу анотації зображення результуючий масив пікселів кодується у формат *JPEG* за допомогою *cv*2.*imencode*, а отримані байти перетворюються у рядок *base*64 для передачі через *HTTP* у складі *JSON*-відповіді.

Потім *JavaScript*-код із файлу script.js вставляє отриманий рядок як *src* атрибут *HTML*-елемента <*img*>. Таким чином анотоване зображення миттєво відображається без збереження його на диск. Отриманий список розпізнавання відображається у вигляді текстового переліку класів із зазначенням довіри та координат. Логіка опрацювання відеофайлу організована за аналогічним принципом, але із покадровим читанням та записом анотованих кадрів до нового відеофайлу за допомогою *cv*2.*VideoWriter*.

3.5 Реалізація веб-сервісу розпізнавання

Веб-сервіс розпізнавання являє собою сполучення між оператором та модулями детекції *YOLO*. Саме через нього виконується отримання файлів зображень і відео, вибір моделі розпізнавання, виклик відповідних методів класу *MultiModelDetector* та повернення отриманих результатів.

Серверна частина веб-сервісу була реалізована у файлі *main.py*. Для доступу до функціоналу детекції визначено два основних *REST-endpoint*:

1) *POST /detect* – для обробки одного зображення;
2) *POST /detect_video* – для обробки відео.

Endpoint /detect використовується для детекції об’єктів на статичному зображенні. Він приймає файл зображення та параметр mode, який потрібен для визначення моделі розпізнавання. У середині функції-обробника виконується читання байтів файлу, виклик методу *detect_on_image* класу *MultiModelDetector*, кодування анотованого зображення у формат *JPEG* та перетворення його в *base*64-рядок для передачі у *JSON*-відповіді.

Endpoint /detect_video працює за аналогічною логікою, але для відео. Спираючись на розміри відео та обмеження пам'яті, відео спочатку зберігається у тимчасовий файл, після чого відкривається через *OpenCV*. Дане відео

опрацьовується по кадрах і уже анотовані кадри записуються у новий відеофайл. В кінцевому результаті сервер повертає клієнту відповідь у вигляді *FileResponse*, що дозволяє оператору завантажити уже анотоване відео.

Обробка запиту /*detect_video* (скорочена версія) у модулі main.py:

```
@app.post("/detect_video")
async def detect_video(
    file: UploadFile = File(...),
    mode: str = Form("multi"),
):
    temp_in = tempfile.NamedTemporaryFile(delete=False, suffix=".mp4")
    video_bytes = await file.read()
    temp_in.write(video_bytes)
    temp_in.close()
    cap = cv2.VideoCapture(temp_in.name)
detector.detect_on_frame(...)
    return FileResponse(
        temp_out_name,
        media_type=media_type,
        filename=download_name,
    )
```

Для покращення швидкодії було вирішено не опрацьовувати кожен кадр, а розглядати кожен другий кадр. Таким чином зменшимо час на опрацювання та анотування вхідного відео.

Співдія між веб-інтерфейсом та серверною частиною відбувається через *HTTP*-запити, які формуються засобами *JavaScript*. Для надсилання кадрів та відео використовується формат *FormData*, що дозволяє інкапсулювати бінарний вміст файлу та додаткові текстові параметри в одному запиті типу *POST*. У випадку *Endpoint /detect* у формі передаються два основні поля:

1) *file* – вхідний файл;
2) *mode* – визначення моделі *YOLO*, яка необхідна.

На стороні клієнта це реалізовано через створення об’єкта *FormData* та додавання до нього полів *file* і *mode*, після чого дані відправляються через *fetch* на адресу /*detect*. Формування *HTTP*-запиту з використанням *FormData*:

```
form.addEventListener('submit', async (event) => {
    event.preventDefault();
    const file = imageInput.files[0];
    const mode = modeSelect.value;
    const formData = new FormData();
    formData.append('file', file);
    formData.append('mode', mode);
    const response = await fetch('/detect', {
        method: 'POST',
        body: formData
    });
    const data = await response.json();
    annotatedImage.src = `${data.image_base64}`;
    // ... подальше оновлення списку детекцій
});
```

Форматом відповіді для /*detect* являється *JSON*-об’єкт, який містить два ключі:

1) *image_base*64 – рядок із закодованим у *base*64 анотованим кадром;
2) *detections* – список детекцій.

У веб-браузері *JavaScript*-код декодує отриманий *JSON*, вставляє рядок *image_base*64 у атрибут *src* елемента <*img*>, а список детекцій подається у текстовому вигляді. Даним підходом анотований кадр не потрібно зберігати на диск. Він передається у відповідь безпосередньо в тілі *JSON*.

Важливим етапом є забезпечення базової перевірки вхідних даних і коректна обробка можливих помилкових ситуацій. На рівні *FastAPI* ця валідація виконується через типи параметрів функцій-обробників та через явні перевірки у коді. Для *endpoint /detect* типовою помилкою може бути завантаження пошкодженого

зображення. На рівні модуля detector.py це перевіряється після виклику *cv2.imdecode*.

Також таку помилку можна перехоплювати на рівні *FastAPI* й повертати користувачеві *JSON* з полем *error*.

Навіть якщо в наявній реалізації обробка даного випадку обмежена, самого факту перевірки достатньо, щоб уникнути падіння моделі на некоректних даних.

Для *endpoint /detect_video* реалізовано більш розгалужену валідацію. Після запису байтів у тимчасовий файл сервер виконує спробу відкриття відео за допомогою *cv2.VideoCapture*. За умови не відкриття відео тимчасовий файл видаляється, а оператору відправляється відповідь *JSONResponse* зі статус-кодом 400 та повідомленням про помилку відкриття відео.

Додатковим елементом валідації є робота з кадровою частотою. За умови, якщо прочитане з відео значення *FPS* рівне нулю або від'ємне встановлюється значення за замовчуванням. Такий підхід дозволяє уникнути проблем при записі вихідного відео.

3.6 Реалізація веб-інтерфейсу оператора

Веб-інтерфейс оператора є ключовим засобом взаємодії користувача з системою розпізнавання об'єктів. Завдяки ньому здійснюється завантаження вхідних даних, вибір моделі детекції, запуск процесу розпізнавання та перегляд отриманих результатів. Він розроблений як односторінковий сайт на основі *HTML*, *CSS* та *JavaScript*, який працює у звичному браузері та взаємодіє з сервером через *REST-endpoints*.

Вміст веб-сторінки визначається *HTML*-файлом *index.html*. Розмітка сторінки містить кілька логічних блоків:

1) заголовок сторінки;
2) форма завантаження кадру з БПЛА;
3) форма завантаження відео з БПЛА;
4) селектор вибору моделі розпізнавання;

5) блок для відображення анотованого кадру;

6) блок для текстового виведення списку детекцій;

7) елемент посилання для завантаження анотованого відео.

Форма завантаження кадру містить елемент <*input type="file"*> з атрибутом *accept="image/**", селектор вибору моделі розпізнавання та кнопку для запуску детекції. Форма для відео ідентична з попередньо, але працює з відеофайлами. Застосовується елемент <*img*> з ідентифікатором *annotated-image*, щоб оператор отримував анотований кадр відразу на сайті. В свою чергу для відео використовується елемент <*a id="download-video-link"*>, який після успішного процесу розпізнавання стає активним посиланням для завантаження анотованого файлу.

На рисунку 3.6. відображено головне вікно веб-інтерфейсу оператора.

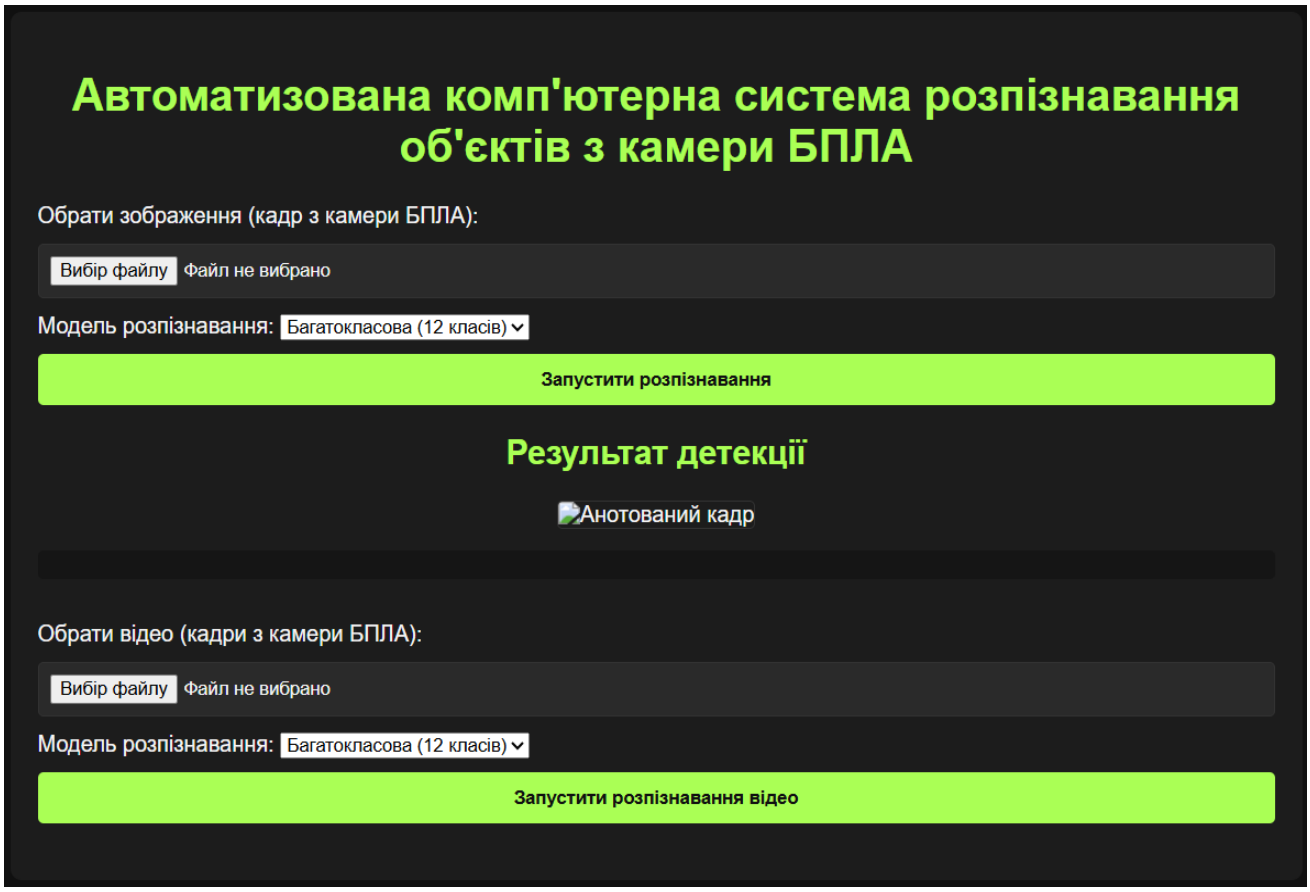


Рисунок 3.6 – Головне вікно веб-інтерфейсу оператора

Селектор вибору моделі розпізнавання реалізований на рівні інтерфейсу за допомогою випадаючих списків для кадрів та відео. Оператор має можливість самостійно обрати одну із наведених моделей: багатокласова та спрощена. Вибір моделі передається на сервер як поле *mode* об’єкта *FormData*. У коді це реалізовано наступним чином:

```
const mode = modeSelect.value;
const formData = new FormData();
formData.append('file', file);
formData.append('mode', mode);
const response = await fetch('/detect', {
   method: 'POST',
   body: formData });
```

Даний простий механізм дає змогу оператору акцентувати увагу системи на шукані об'єкти. У випадках, коли головний інтерес становлять танки, можна використовувати спрощену модель, а при потребі в ширшому аналізі сцени – переключитися на багатокласову детекцію.

Веб-інтерфейс відповідає за візуальне представлення результатів детекції після того, як сервер опрацює вхідні дані. Для кадру сервер повертає *JSON*-структуру, що містить *base64*-кодоване анотоване зображення та список спрацювань.

У клієнтському коді даний результат обробляється в обробнику успішного запиту до *detect*. За умови, якщо поле *image_base*64 присутнє, то воно вставляється як джерело елемента *<img id="annotated-image">*, що дозволяє відобразити зображення без додаткового збереження файлу на диск.

Також оператор отримує список знайдених об'єктів у контейнері *detections-list*. На рисунку 3.7. відображено вивід результату розпізнавання об'єкту з кадру БПЛА. Зображення демонструє приклад обробки кадру в якому зафіксовано ціль та ідентифіковано за допомогою алгоритмів комп’ютерного зору.

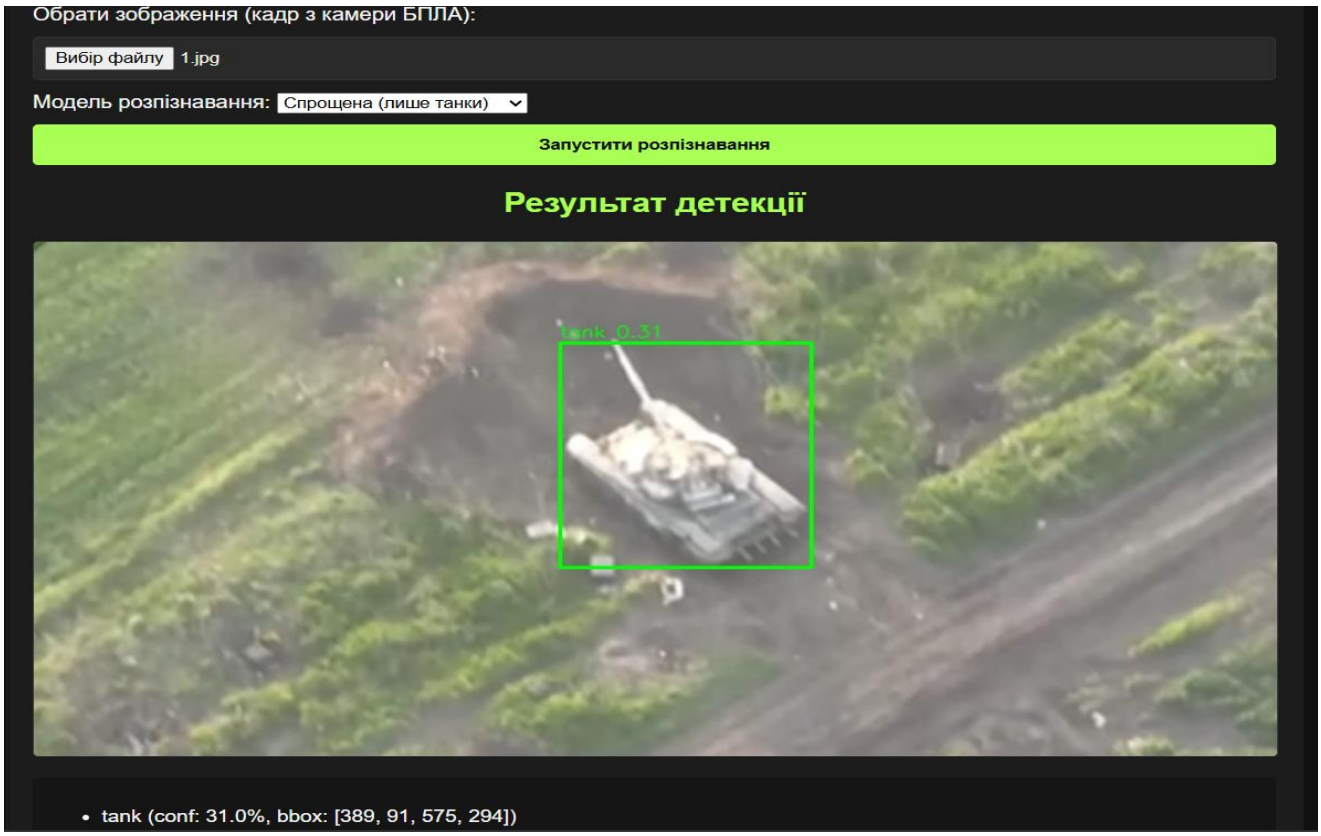


Рисунок 3.7 – Вивід результату розпізнавання об'єкту з кадру БПЛА

Для обробки відеоданих застосовується дещо інша логіка взаємодії клієнта з сервером, ніж у випадку зображень або окремих кадрів. Основна відмінність полягає в тому, що сервер у відповідь на запит не повертає структуровані дані у форматі JSON, а передає анотоване відео у вигляді байтового потоку.

Після отримання потоку на клієнтській стороні виконується його обробка за допомогою інтерфейсу *Blob*. Зокрема, дані відеофайлу інкапсулюються у спеціальний об'єкт типу *Blob*, який слугує контейнером для зберігання бінарних даних із зазначеним типом.

Для забезпечення можливості завантаження відео оператором, з цього об’єкта створюється тимчасове посилання за допомогою вбудованого методу *URL.createObjectURL(blob)*. Це посилання є унікальним *URL*-ідентифікатором, що дозволяє браузеру інтерпретувати локальний *Blob* як повноцінний файл.

Отриманий *URL* динамічно прив'язується до *HTML*-елемента *<a id="download-video-link">* шляхом встановлення його у значення атрибута *href*.

Завдяки цьому користувач має можливість натиснути на посилання і завантажити готове анотоване відео з уже застосованими результатами розпізнавання.

Такий підхід є зручним тому, що дозволяє уникнути проміжного збереження файлу на сервері та забезпечує безпечну передачу персоналізованого результату кінцевому користувачу.

Приклад отримання анотованого відео наведено на рисунку 3.8.

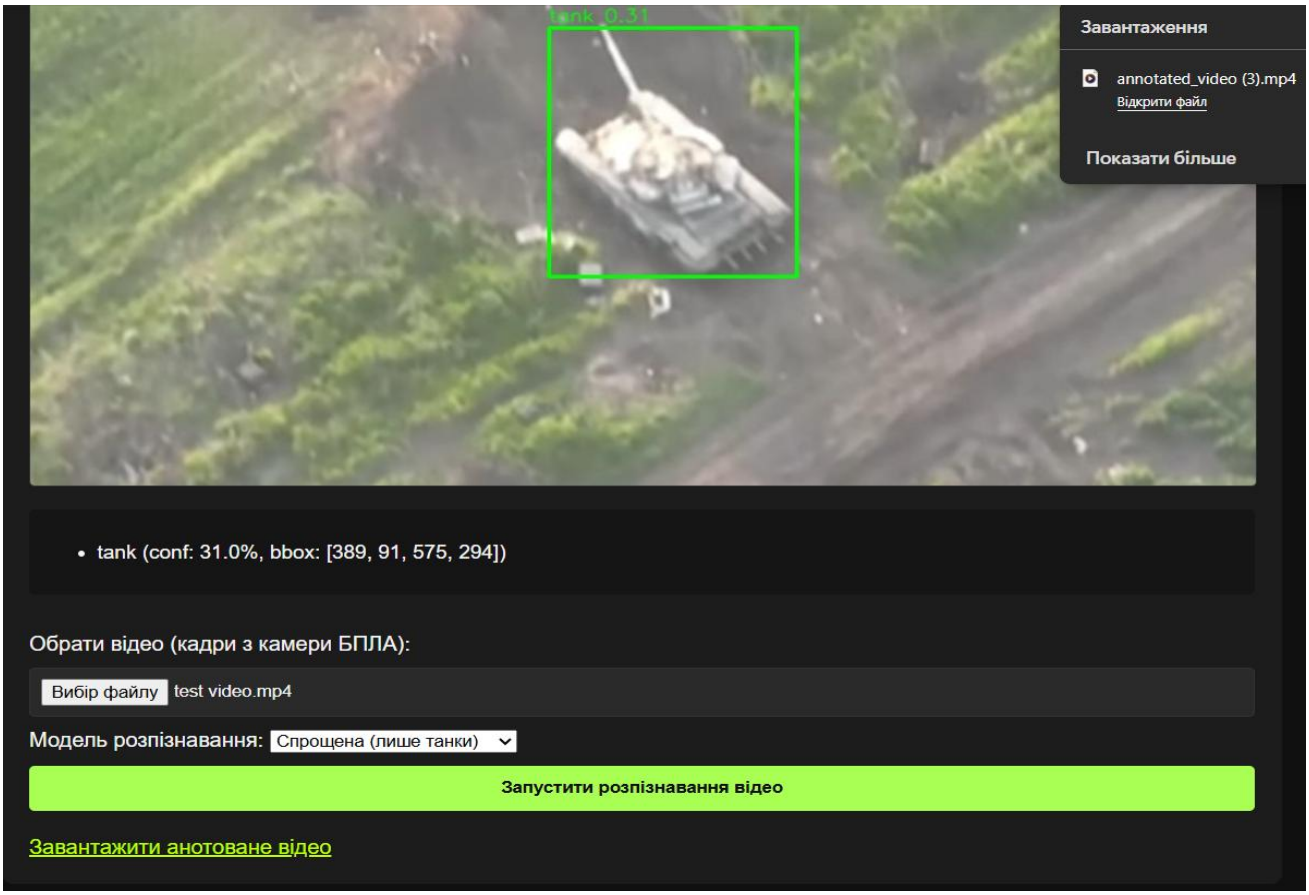


Рисунок 3.8 – Отримання анотованого відео

3.7 Обробка відеоматеріалів з камер БПЛА

Так як реальна робота БПЛА пов’язана з безперервним відеопотоком, то обробка відеоматеріалів є важливим компонентом розробленої системи. У розробленій системі реалізовано офлайн-режим обробки відео. На рисунку 3.9 зображено графічну схему алгоритму обробки відео.

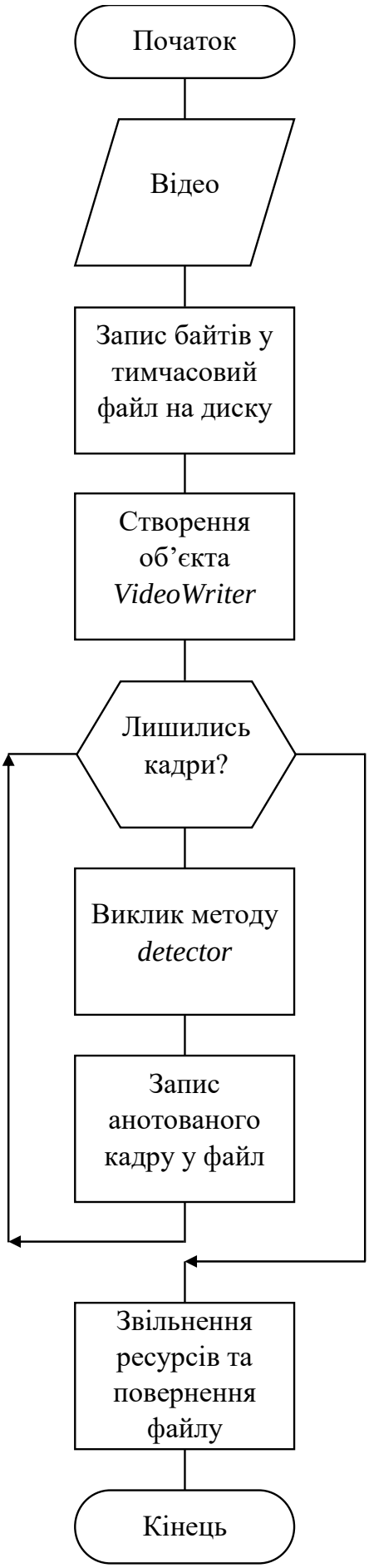


Рисунок 3.9 – Графічна схема алгоритму обробки відео

Оскільки розпізнавання об’єктів на кожному кадрі відео потребує великої кількості обчислювальних ресурсів, у системі передбачено низку простих рішень спрямованих на зменшення часу обробки та навантаження на апаратну платформу.

Першим рішенням являється пропуск частини кадрів. У циклі покадрової обробки створено лічильник кадрів *frame_idx*. Для кожного кадру перевіряється чи є він парним чи непарним. Якщо кадр не підпадає під умову обробки (обробляється кожен другий кадр), то він записується у вихідне відео без запуску детекції. У лістингу коду таке рішення виглядає так:

```
frame_idx = 0
while True:
    ret, frame = cap.read()
    if not ret:
        break
    frame_idx += 1
    if frame_idx % 2 != 0:
        out.write(frame)
        continue
    result = detector.detect_on_frame(frame, mode=mode)
    annotated_frame = result["annotated_image"]
    out.write(annotated_frame)
```

Наступним рішенням стала корекція кадрової частоти. На деяких відео значення *FPS*, прочитане через *cv*2.*CAP_PROP_FPS*, може бути некоректним. У такій ситуації в коді задається значення за замовченням, що дозволяє коректно створити *VideoWriter* і уникнути помилок при записі вихідного відео. Приклад реалізації:

```
fps = cap.get(cv2.CAP_PROP_FPS)
if fps <= 0:
    fps = 25.0
```

У розробленій системі для створення вихідного відео спочатку використовується кодек *mp4v* та формат *mp4*. За умови, якщо *VideoWriter* не відкривається, алгоритм використовує кодек *XVID* та формат *avi*. Таким чином підвищується сумісність із різними системами.

В кінцевому результаті обробки оператор отримує анотоване відео, яке можна взяти за основу для:

1) звіту щодо виявлених об’єктів;
2) подальшого тактичної ситуації;
3) формування розширених навчальних вибірок.

3.8 Оцінка якості системи

Оцінювання проводилося для двох моделей розпізнавання, побудованих на основі *YOLOv*8: спрощеної та багатокласової. Навчання спрощеної моделі було повністю завершене, що дозволило отримати розподіл анотацій у кадрі (рисунок 3.10), графіки зміни *train/val*-втрат (рис. 3.11) та *mAP* під час навчання та матрицю помилок (рис. 3.12).

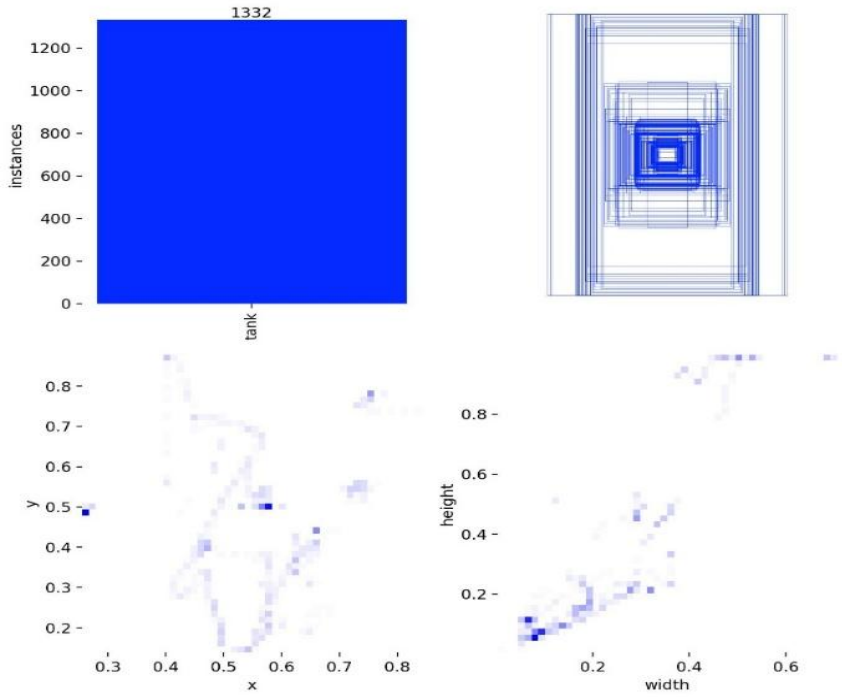


Рисунок 3.10 – Розподіл анотацій у кадрі

У верхньому лівому полі показано гістограму кількості розмічених об’єктів за класами. З нього можна побачити, що набір даних містить 1332 анотації класу tank. У правому верхньому полі продемонстровано всі прямокутні рамки розмічених об'єктів. Така візуалізація допомагає зрозуміти в якій частині кадру найчастіше зустрічається шуканий об'єкт. В даному варіанті танки знаходяться у різних частинах кадру, але найбільше скупчення спостерігається в центральній області з невеликим зміщенням донизу.

Нижнє ліве поле відображає *heatmap* розподілу центрів рамок у нормованих координатах (*x*, *y*). Світлі ділянки відображають зони у яких центри об’єктів зустрічаються найчастіше. Можна зрозуміти, що танки розташовані не тільки по центру і це підвищує різноманітність просторових конфігурацій, стійкість моделі до зміни ракурсу. Нижнє праве поле відповідає *heatmap* пари «ширина–висота» рамок. Відповідно цього можемо зрозуміти, що набір даних містить як компактні цілі так і великі, що займають значну частину кадру. Такий різновид масштабів важливий для навчання детектора, здатного добре розпізнавати цілі на високій та низькій висоті польоту.

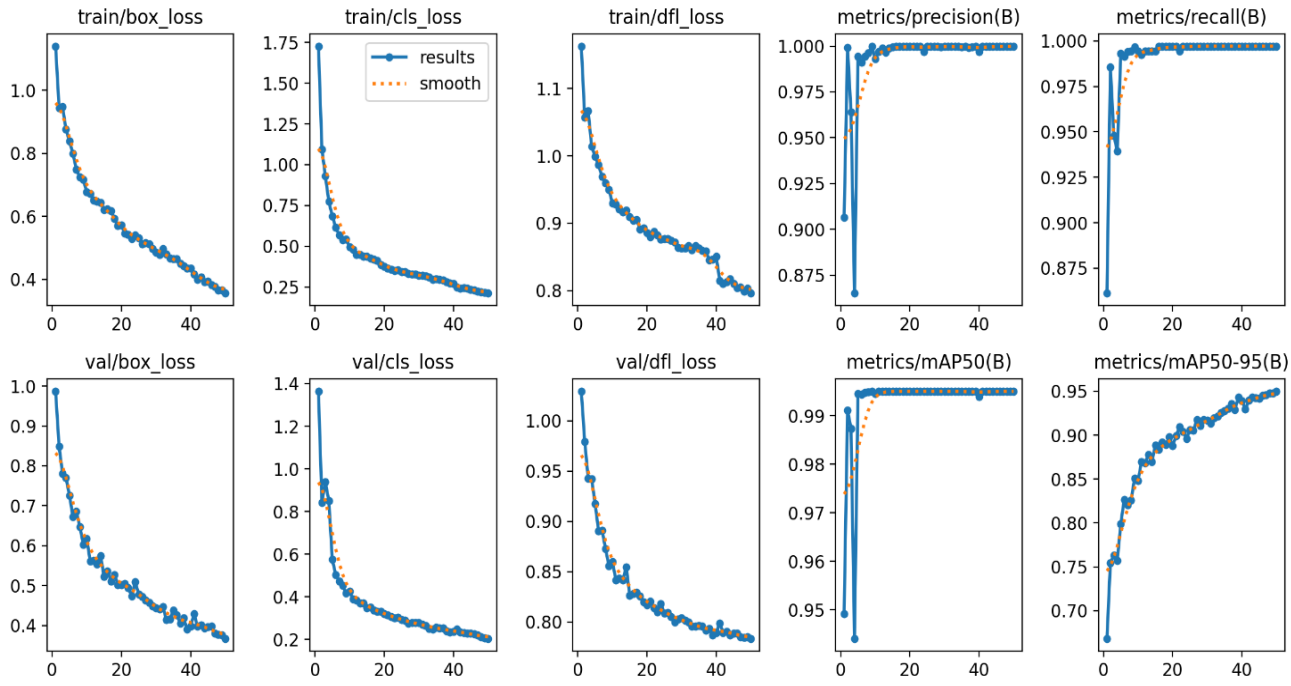


Рисунок 3.11 – Графіки зміни *train/val*-втрат

По горизонталі відкладено істинні класи, по вертикалі спрогнозовані. Темний колір клітинки відповідає більшій кількості прикладів відповідного типу. У верхній лівій клітинці зафіксовано 345 прикладів – це показник правильно виявлених танків. У нижній правій клітинці міститься лише 1 приклад – це випадок, коли на зображенні був танк, але модель не змогла віднести його до правильного класу. Клітинки, які відповідають похибкам порожні. Це означає, що модель не спрацьовує у варіантах, коли шукані об'єкти відсутні. Даний рисунок підтверджує доцільність спеціального навчання на специфічному набору даних кадрів з БПЛА, що забезпечує бажане виявлення бронетехніки в реальних умовах.

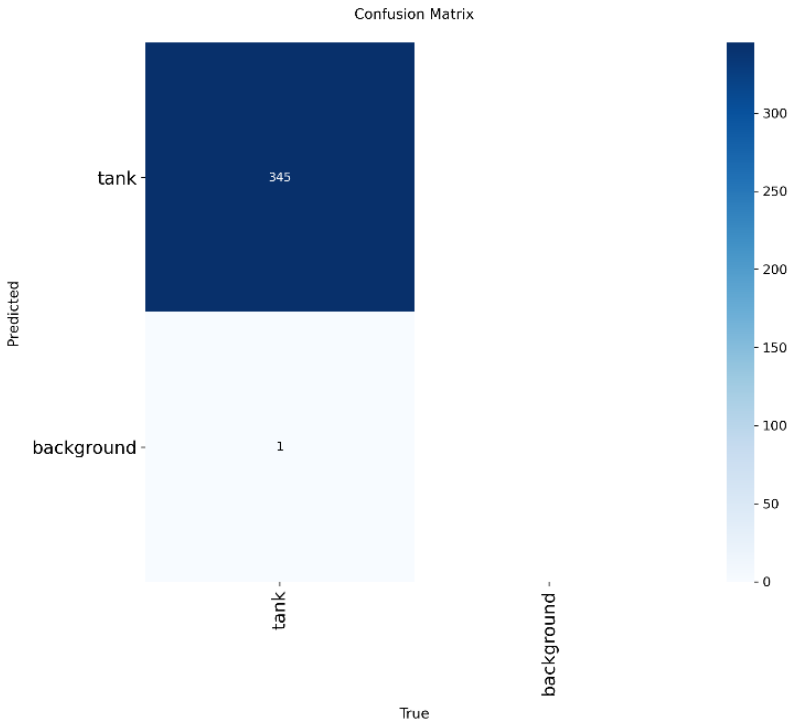


Рисунок 3.12 – Матриця помилок

Навчання багатокласової моделі було свідомо зупинено, тому для неї доступні лише проміжні результати у вигляді числових значень *Precision*, *Recall* та *mAP* з файлу *results.csv*, без повноцінної матриці помилок та фінальної валідації.

На рисунку 3.13 можна побачити приклади анотованих зображень.

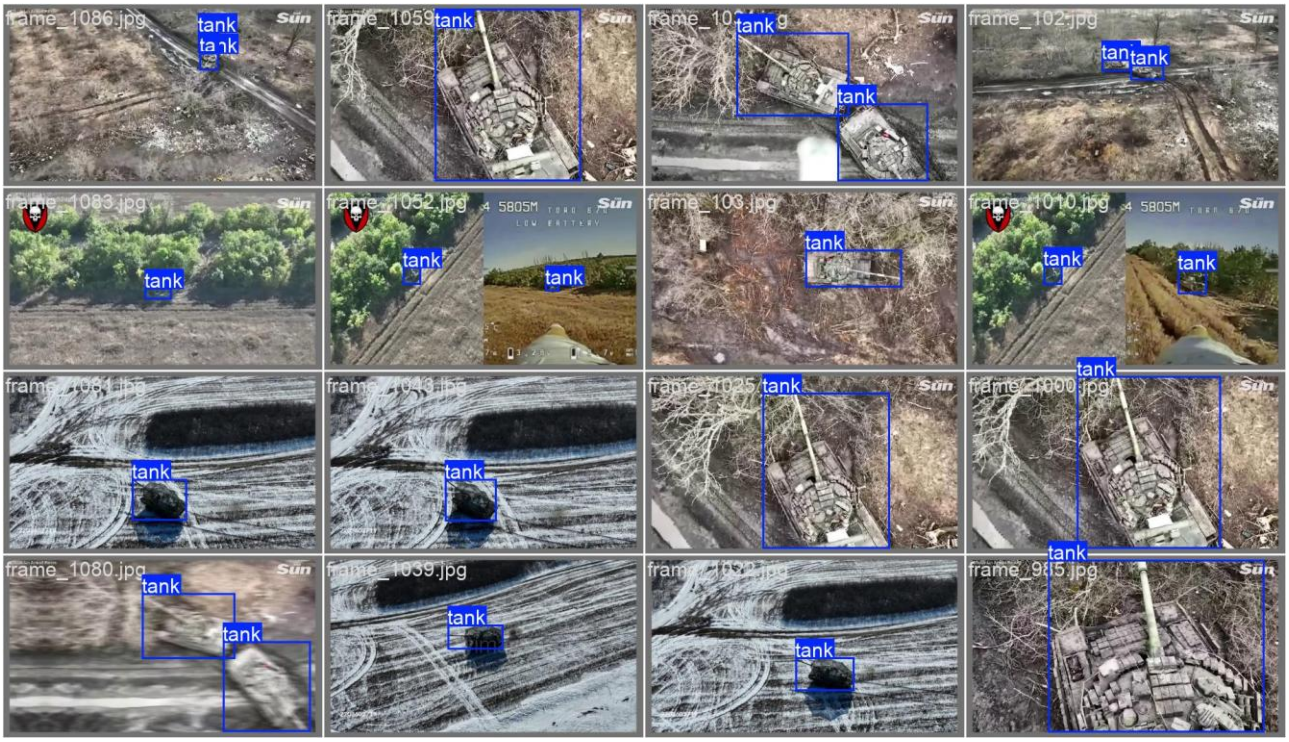


Рисунок 3.13 – Анотовані зображення

Для кількісної оцінки якості розпізнавання використовувались стандартні метрики:

1) *Precision*;
2) *Recall*;
3) *mAP*@0.5;
4) *mAP*@0.5–0.95.

Спираючись на дані метрики сформуємо для моделей таблиці. У таблиці 3.1 відображені метрики спрощеної моделі, а у таблиці 3.2 – багатокласової.

Таблиця 3.1 – Метрики спрощеної моделі

Параметр	Значення
1	2
precision	0,99983
recall	0,99711
mAP50	0,99500

Продовження таблиці 3.1

1	2
box_loss	0,36751
cls_loss	0,20441
dfl_loss	0,78366
mAP50-95	0,94987

З отриманих значень можна сказати, що спрощена модель практично безпомилково виявляє танки на тестових зображеннях. Звертаючи увагу на *Recall* та *Precision*, стає зрозуміло, що майже всі цілі знаходяться, а хибних спрацювань мінімальна кількість. Високий рівень *mAP*@0.5–0.95 означає, що модель не лише знаходить об’єкти, а й досить точно оцінює їхні межі у широкому діапазоні порогів *IoU*. Дані висновки узгоджуються з візуальним аналізом анотованих кадрів та матрицею помилок.

Таблиця 3.2 – Метрики багатокласової моделі

Параметр	Значення
precision	0,57194
recall	0,45746
*mAP*50	0,47148
*mAP*50-95	0,29492
box_loss	1,11756
cls_loss	7,83551
dfl_loss	1,38304

Через те, що модель була перервана при навчанні, то було отримано не зовсім втішні дані. Отримані значення істотно поступаються спрощеній моделі, що пов’язано як із більш складною багатокласовою задачею, так і з фактом незавершеного навчання.

Тепер порівняю швидкодію двох моделей. Для цього використаю 8 кадрів з БПЛА, які не використовувались у навчанні. На рисунку 3.14 відображено консольні значення під час детекції з використанням спрощеної моделі, а на рисунку 3.15 з використанням багатокласової моделі.

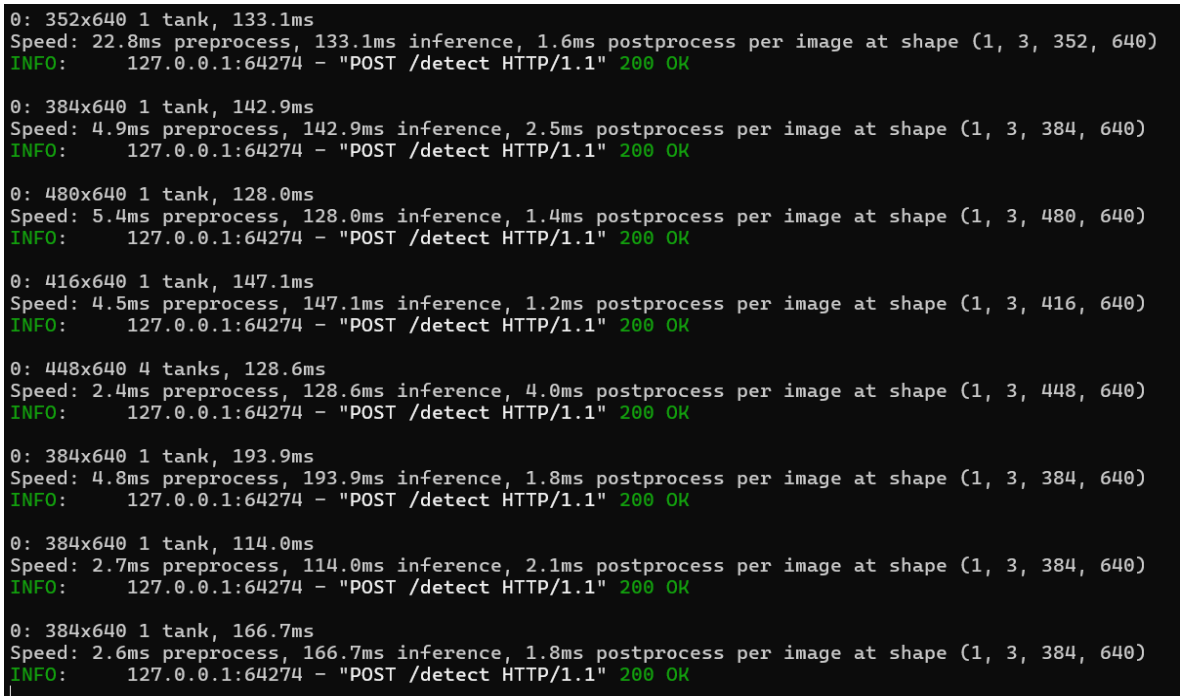


Рисунок 3.14 – Скриншот консолі детекції з використанням спрощеної моделі

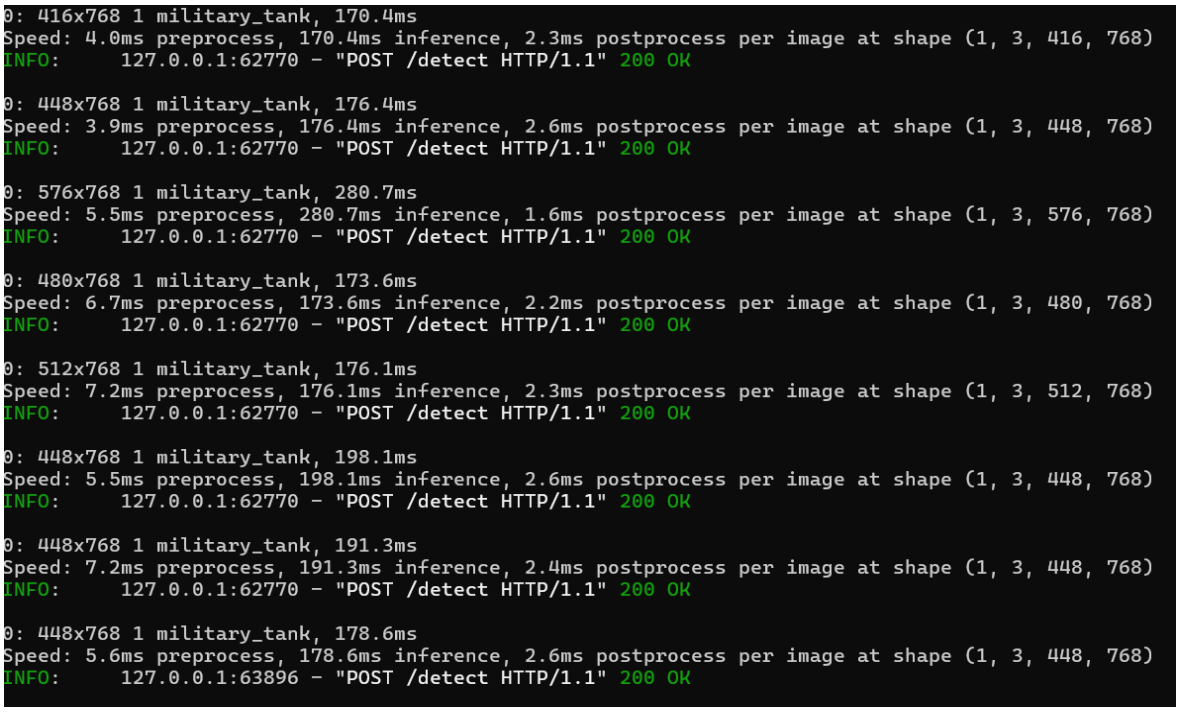


Рисунок 3.15 – Скриншот консолі детекції з використанням багатокласової моделі

Середні значення для спрощеної моделі:

1) середній час 144 *ms*;
2) оцінка *FPS* 6.9 кадр/с.

Середні значення для багатокласової моделі:

1) середній час 180 *ms*;
2) оцінка *FPS* 5.5 кадр/с.

Спрощена модель працює швидше приблизно на 25–30%. Багатокласова модель обробляє кадр довше через більшу кількість класів. В обох випадках *overhead* на *preprocess* та *postprocess* невеликий (кілька мілісекунд), отже різницю в швидкодії в основному створює саме робота нейромережі.

Висновок до розділу

У третьому розділі виконано повну практичну реалізацію та експериментальну перевірку автоматизованої комп'ютерної системи розпізнавання об'єктів з бортової камери БПЛА.

Спочатку сформовано програмне середовище. Було обрано мову *Python*, фреймворк *Ultralytics YOLOv*8, бібліотеки *OpenCV* та *NumPy* для обробки зображень, *FastAPI* як бекенд-фреймворк, а також стек *HTML, CSS, JavaScript* для побудови веб-інтерфейсу оператора. Розроблено структуру програмного проєкту з чітким розподілом на модулі детектування, підготовки даних, веб-сервісу та статичного інтерфейсу.

На основі вихідного *UAV*-датасету та готових наборів даних було проведено два навчання: спрощеної та багатокласової моделів. Згідно стандартних метрик було виявлено, що спрощена модель має високі показники якості. В свою чергу багатокласова модель, через дострокове завершення навчання, демонструє лише проміжні, але перспективні значення метрик.

Проведено експериментальну оцінку системи на тестових кадрах. Виявилось, що спрощена модель має вищу точність та швидкодію порівняно з багатокласовою моделлю. Отримані результати цілком логічні з огляду на більшу складність задачі.

Водночас навіть частково навчена багатокласова модель демонструє непогані результати під час розпізнавання об'єктів з кадру.

Отже, в ході виконання третього розділу роботи було розроблено працездатний веб-сервіс, який інтегрує навчені моделі глибинного навчання з інтерфейсом оператора, підтримує обробку як окремих кадрів, так і відеоматеріалів з бортових камер БПЛА та забезпечує прийнятну якість і швидкодію. Виявлені недоліки багатокласової моделі визначають напрямок подальшого вдосконалення системи.

ВИСНОВКИ

У даній магістерській роботі виконано дослідження та практичну реалізацію автоматизованої комп’ютерної системи розпізнавання об’єктів з бортової камери БПЛА. Робота об'єднує теоретичний аналіз сучасних підходів комп’ютерного зору та глибинного навчання з розробкою програмного прототипу, придатного для застосування оператором у реальних умовах. Дослідження складалося з трьох основних розділів.

У ході написання першого розділу роботи було проаналізовано основні теоретичні засади комп'ютерного зору, обробки зображень, класифікації БПЛА, датчиків, які використовуються на борту безпілотників та розглянуто сфери практичного застосування літальних розвідувальних комплексів. Таким чином, проведені у першому розділі аналізи створили теоретичну основу для подальших досліджень у наступних розділах роботи.

У другому розділі було проаналізовано існуючі підходи до розпізнавання об’єктів та спеціалізовані рішення для обробки кадрів отриманих з камер БПЛА. Виявилось, що для задачі виявлення наземної техніки на знімках найбільш доцільним є використання одноступеневого детектору *YOLOv*8. Він забезпечує прийнятний компроміс між точністю та швидкодією. Окрему увагу приділено проблемі розпізнавання дрібних цілей у складних умовах, особливостям формування карт ознак та труднощам створення набору даних для навчання нейромережі.

Кінцевим етапом являється практична частина роботи. У ній було підготовлено набори даних у форматі *YOLO*, які містять різні перешкоди у вигляді диму, погодних умов та маскування цільових об’єктів. На основі отриманих наборів було навчено спрощену та багатокласову моделі. Розроблено повноцінну систему розпізнавання об'єктів з бортової камери БПЛА, яка містить серверний модуль, відповідальний за розпізнавання та обробку запитів, та клієнтський веб-

інтерфейс для взаємодії оператора із системою в режимі реального часу. Проведено порівняльний аналіз спрощеної та багатокласової моделей.

Спираючись на отримані результату можна сказати, що спрощена модель є оптимальною для точного й швидкого виявлення танків, а багатокласова дозволяє отримати більше інформації про вміст об'єктів на кадрі. В обох випадках досягнуто прийнятних значень часу обробки зображення та відео.

Розроблена система може бути використана як основа для впровадження засобів автоматизованого розпізнавання об'єктів з кадрів літальних апаратів у розвідувальних комплексах. Подальший розвиток роботи може передбачати додаткове навчання багатокласової моделі на розширених наборах даних та безпосередню інтеграцію системи на борт БПЛА.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Лесюк А. М., Яцишин С.П. Комп'ютерний зір та його застосування. 2020. - *URL*: *https://lnk.ua/k4kjDygNL* (дата звернення 12.10.2025).

2. *Victor Wiley, Thomas Lucas. Computer Visionand Image Processing: A Paper Review.* 2018. - *URL*: *https://ijair.id/index.php/ijair/article/view/42/pdf* (дата звернення 12.10.2025).

3. Савкін В. Б., Павлюк Д. С., Мочульська О. М., Козбур І. Р. Комп'ютерний зір в системах відеоспостереження. 2025. - *URL*: *https://sci-conf.com.ua/wp-content/uploads/2025/07/SCIENCE-IN-THE-MODERN-WORLD-INNOVATIONS-AND-CHALLENGES-12-14.06.2025.pdf* (дата звернення 12.10.2025).

4. Соловйов П.С. Застосування нейронних мереж для ідентифікації меж об'єктів зображення. 2023. - *URL*: *https://lnk.ua/ANDZ1g3ex* (дата звернення 15.10.2025).

5. *Sriani, Ilka Zufria, Mhd. Syahnan. Improved digital image quality using the gaussian filter method.* 2022. - *URL*: *https://lnk.ua/mNBrQYjNG* (дата звернення 15.10.2025).

6. *Johannes Maucher. Gaussian Filter and Derivatives of Gaussian*. 2021. - *URL*: *https://hannibunny.github.io/orbook/preprocessing/04gaussianDerivatives.html#dimensional-gaussian-filter-1* (дата звернення 20.10.2025).

7. Клюшин Д. А. Методи розпізнавання контурів зображень ядер клітин. 2014. - *URL*: *http://nbuv.gov.ua/UJRN/VKNU_fiz_mat_2014_3_29* (дата звернення 20.10.2025).

8. *Chethan K. S., Dr. Nataraj K. R., Sinchana G. S., Dr. Choodarathnakara A. L. Analysis of Image Quality using Sobel Filter*. 2019. - *URL*: *https://lnk.ua/R4aMDPy4J* (дата звернення 05.11.2025).

9. *Rupika Rana, Ashish Verma. Comparison and Enhancement of Digital Image by Using Canny Filter and Sobel Filter.* 2014. - *URL*: *https://www.academia.edu/7177881/B016190610* (дата звернення 05.11.2025).

10. *Tachinina O.M., Lysenko A.I., Kutiepov V.O. Classification of modern unmanned aerial vehicles*. 2022. 79-86 с.

11. Сироткіна Н.П., Василів С.С., Музика Л.В. Класифікація безпілотних літальних апаратів та їх реактивних енергетичних установок. 2025. 19-25 с.

12. *Aaron Dennis, James Archibald, Barrett Edwards, D.J. Lee. On-Board Vision-Based Sense-and-Avoid for Small UAVs.* 2012. - *URL*: *https://arc.aiaa.org/doi/epdf/*10.2514/6.2008-7322 (дата звернення 05.11.2025).

13. Глотов В., Петришин І. Аналзі сучасних безпілотних літальних апаратів, оснащених системою лазерного сканування. 2023. - *URL*: *https://zgt.com.ua/wp-content/uploads/2023/05/7.pdf* (дата звернення 08.11.2025).

14. Опанасенко М.О. Сфери застосування безпілотних літальних апаратів. 2022. - *URL*: *https://lnk.ua/94y9DE14M* (дата звернення 13.11.2025).

15. Гозак Ярослав, Палій Сергій. Порівняльний аналіз нейронних мереж розпізнавання об'єктів на зображеннях. 2025. - *URL*: *https://www.researchgate.net/publication/396576312_PORIVNALNIJ_ANALIZ_NEJRONNIH_MEREZ_ROZPIZNAVANNA_OB'EKTIV_NA_ZOBRAZENNAHNEURAL_NETWORKS_FOR_OBJECT_RECOGNITION_IN_IMAGES_COMPARATIVE_ANALYSIS* (дата звернення 13.11.2025).

16. *A Benchmark Review of YOLO Algorithm Developments for Object Detection*. 2025. - *URL*: *https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=11072404* (дата звернення 14.11.2025).

17. *Comprehensive analysis on Ultralytics-supported YOLO models for detection and recognition of large office objects for indoor navigation / R. G. Baldovino et al. Procedia Computer Science.* 2024. *Vol*. 246. 3851–3858 с.

18. *Explore Ultralytics YOLOv*8. - *URL*: *https://docs.ultralytics.com/models/yolov8/* (дата звернення 20.11.2025).

19. *Jia Liu, Shuang Liu, Shujuan Xu, Changjun Zhou. Two-stage underwater object detection network using swin transformer*. 2022. - *URL*: *https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9938441* (дата звернення 20.11.2025).

20. *Explore Ultralytics YOLOv8.* - *URL*: *https://docs.ultralytics.com/models/yolov8/* (дата звернення 22.11.2025).

21. Сизоненко О., Божуха Л. Виявлення об'єктів на зображенні в потоковому режимі при використанні *YOLOv5* і *FASTER R-CNN*. 2024. - *URL*: *https://lnk.ua/AVMm6rvVo* (дата звернення 25.11.2025).

22. Цивадиць Павло, Скрипник Тетяна Вознюклеонід. Порівняння методів виявлення об'єктів в комп'ютерному зорі. 2024. - *URL*: *https://heraldts.khmnu.edu.ua/index.php/heraldts/article/view/145/153* (дата звернення 25.11.2025).

23. *Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, Jiaya Jia. Path Aggregation Network for Instance Segmentation.* 2018. - *URL*: *https://lnk.ua/MVwoBx3Vz* (дата звернення 28.11.2025).

24. Глод С. І., Дорошенко А. В. Вибір ефективної моделі для розпізнавання військових об'єктів у режимі реального часу на спеціалізованих наборах даних / Міністерство освіти і науки України, Національний університет «Львівська політехніка». Електронне мережне навчальне видання. Львів, 2024. 137-148 с.

25.*Feature Pyramid Network (FPN).* 2025. - *URL*: *https://lnk.ua/YNg5DlYeZ* (дата звернення 27.11.2025).

26. *Sara Beery, Guanhang Wu, Vivek Rathod, Ronny Votel, Jonathan Huang. Context R-CNN: Long Term Temporal Context for Per-Camera Object Detection.* 2020. - *URL*: *https://lnk.ua/RVdKDlme3* (дата звернення 27.11.2025).

27. *Mahya Nikouei, Bita Baroutian, Shahabedin Nabavi, Fateme Taraghi, Atefe Aghaei, Ayoob Sajedi, Mohsen Ebrahimi Moghaddam. Small Object Detection: A Comprehensive Survey on Challenges, Techniques and Real-World Applications*. - *URL*: *https://arxiv.org/pdf/2503.20516* (дата звернення 28.11.2025).

28. *Mahtab Jamali, Paul Davidsson, Reza Khoshkangini, Martin Georg Ljungqvist & Radu-Casian Mihailescu. Context in object detection: a systematic literature review*. 2025. - *URL*: *https://link.springer.com/article/10.1007/s10462-025-11186-x* (дата звернення 28.11.2025).

29. *Multi Scale Training in Neural network*. 2020. - *URL*: *https://lnk.ua/R4aMDPd4J* (дата звернення 28.11.2025).

30. *Jumabek Alikhanov, Dilshod Obidov, Mirsaid Abdurasulov, Hakil Kim. Practical Evaluation Framework for Real-Time Multi-Object Tracking: Achieving Optimal and Realistic Performance*. 2025. - *URL*: *https://lnk.ua/Men0DEvNg* (дата звернення 29.11.2025).

31. *Momir Adzemovic. Deep Learning-Based Multi-Object Tracking: A Comprehensive Survey from Foundations to State-of-the-Art.* 2025. - *URL*: *https://arxiv.org/pdf/2506.13457* (дата звернення 02.12.2025).

32. *Observation-Centric SORT: Rethinking SORT for Robust Multi-Object Tracking. 2023. - URL*: *https://lnk.ua/x4LX5zw4n* (дата звернення 02.12.2025).

33. *Gerard Maggiolino, Adnan Ahmad, Jinkun Cao, Kris Kitani. DEEP OC-SORT: MULTI-PEDESTRIAN TRACKING BY ADAPTIVE RE-IDENTIFICATION. 2023. - URL: https://arxiv.org/pdf/2302.11813* (дата звернення 03.12.2025).

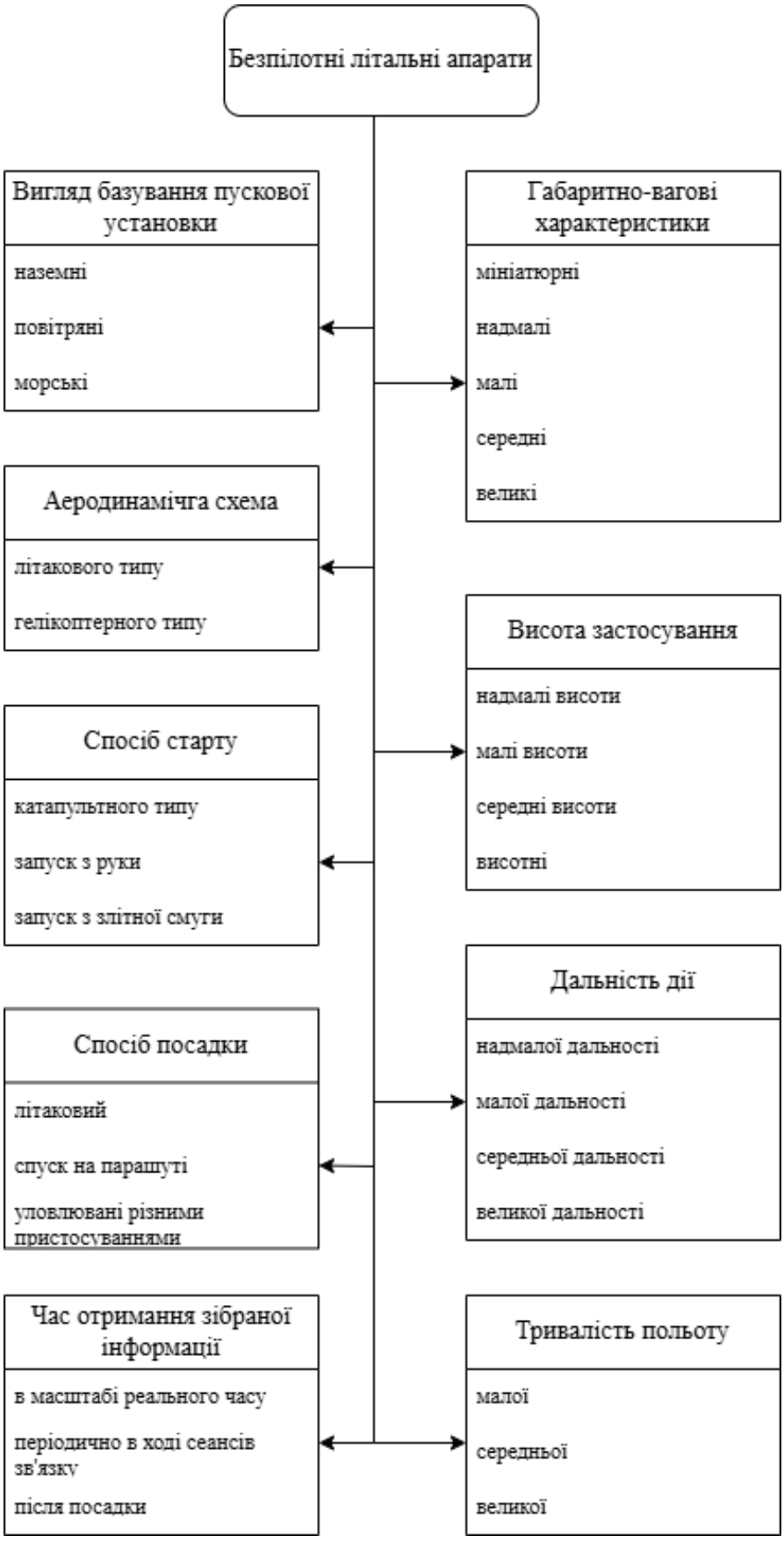
34. *What is ByteTrack? A Deep Dive.* 2024. - *URL*: *https://blog.roboflow.com/what-is-bytetrack-computer-vision/* (дата звернення 04.12.2025).

35. *Szegedy C., Liu W., Jia Y. et al. Going Deeper with Convolutions. In: Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

36. *Bastiaan J. Boom, Phoenix X. Huang, Jiyin He, Robert B. Fisher. Supporting Ground-Truth annotation of image datasets using clustering*. 2012. - *URL*: *https://www.pure.ed.ac.uk/ws/portalfiles/portal/11411403/Boom_Huang_et_al_2913_Approximate_Nearest_Neighbor_Search_to_Support_Manual_Image_Annotation_of_Large_Domain_specific_Datasets.pdf* (дата звернення 04.12.2025).

Додаток А

Класифікація за технічними ознаками



Вміст файлу *main.py*

```
from pathlib import Path
import base64
import tempfile
import os
import cv2
from fastapi import FastAPI, UploadFile, File, Form
from fastapi.responses import HTMLResponse, JSONResponse, FileResponse
from fastapi.staticfiles import StaticFiles
from .detector import MultiModelDetector
BASE_DIR = Path(__file__).resolve().parent.parent
app = FastAPI()
app.mount(
    "/static",
    StaticFiles(directory=str(BASE_DIR / "static")),
    name="static",
)
detector = MultiModelDetector(conf_tank=0.25, conf_multi=0.25)
@app.get("/", response_class=HTMLResponse)
async def index():
    index_path = BASE_DIR / "index.html"
    html = index_path.read_text(encoding="utf-8")
    return HTMLResponse(content=html)
@app.post("/detect")
async def detect(
    file: UploadFile = File(...),
    mode: str = Form("multi"),  # "tank" або "multi"):
```

```
    image_bytes = await file.read()
    result = detector.detect_on_image(image_bytes, mode=mode)
    annotated_image = result["annotated_image"]
    detections = result["detections"]
    jpeg_bytes = detector.encode_image_to_jpeg_bytes(annotated_image)
    image_b64 = base64.b64encode(jpeg_bytes).decode("utf-8")
    return JSONResponse(
        {
            "image_base64": image_b64,
            "detections": detections,
        }
    )
@app.post("/detect_video")
async def detect_video(
    file: UploadFile = File(...),
    mode: str = Form("multi"),
):
    temp_in = tempfile.NamedTemporaryFile(delete=False, suffix=".mp4")
    video_bytes = await file.read()
    temp_in.write(video_bytes)
    temp_in.close()
    cap = cv2.VideoCapture(temp_in.name)
    if not cap.isOpened():
        os.unlink(temp_in.name)
        return JSONResponse(
            {"error": "Не вдалося відкрити відео."},
            status_code=400,
        )
    fps = cap.get(cv2.CAP_PROP_FPS)
```

```
        if fps <= 0:
            fps = 25.0
        width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fourcc = cv2.VideoWriter_fourcc(*"mp4v")
        temp_out = tempfile.NamedTemporaryFile(delete=False, suffix=".mp4")
        temp_out_name = temp_out.name
        temp_out.close()
        out = cv2.VideoWriter(temp_out_name, fourcc, fps, (width, height))
        if not out.isOpened():
            print(">>> [detect_video] Не вдалося відкрити VideoWriter для
MP4/mp4v, пробую AVI/XVID")
            # Пробуємо альтернативу: AVI + XVID
            out.release()
            temp_out = tempfile.NamedTemporaryFile(delete=False, suffix=".avi")
            temp_out_name = temp_out.name
            temp_out.close()
            fourcc = cv2.VideoWriter_fourcc(*"XVID")
            out = cv2.VideoWriter(temp_out_name, fourcc, fps, (width, height))
        if not out.isOpened():
            print(">>> [detect_video] Не вдалося відкрити VideoWriter взагалі :(")
            cap.release()
            os.unlink(temp_in.name)
            os.unlink(temp_out_name)
            return JSONResponse(
                {"error": "Не вдалося відкрити відеозаписувач (VideoWriter)."},
                status_code=500,
            )
        frame_idx = 0
```

```
while True:
    ret, frame = cap.read()
    if not ret:
        break
    frame_idx += 1

    if frame_idx % 2 != 0:
        out.write(frame)
        continue
    result = detector.detect_on_frame(frame, mode=mode)
    annotated_frame = result["annotated_image"]
    out.write(annotated_frame)
cap.release()
out.release()
os.unlink(temp_in.name)

if temp_out_name.endswith(".mp4"):
    media_type = "video/mp4"
    download_name = "annotated_video.mp4"
else:
    media_type = "video/x-msvideo"
    download_name = "annotated_video.avi"
print(f">>> [detect_video] Готовий файл: {temp_out_name}")
return FileResponse(
    temp_out_name,
    media_type=media_type,
    filename=download_name,
)
```

Додаток В

Вміст файлу *detector.py*

```
from pathlib import Path
from typing import List, Dict, Any
import cv2
import numpy as np
from ultralytics import YOLO
BASE_DIR = Path(__file__).resolve().parent.parent
TANK_MODEL_PATH = (
    BASE_DIR
    / "image-innovators"
    / "runs"
    / "detect"
    / "train3"
    / "weights"
    / "best.pt")
MULTI_MODEL_PATH = (
    BASE_DIR
    / "image-innovators"
    / "runs"
    / "detect"
    / "train4"
    / "weights"
    / "best.pt"
)
class MultiModelDetector:
    def __init__(self, conf_tank: float = 0.25, conf_multi: float = 0.25):
        self.conf_tank = conf_tank
```

```
        self.conf_multi = conf_multi
        print(">>> Завантажую tank-модель:", TANK_MODEL_PATH)
        self.tank_model = YOLO(str(TANK_MODEL_PATH))
        print(">>> Класи tank-моделі:", self.tank_model.names)
        print(">>> Завантажую multi-модель:", MULTI_MODEL_PATH)
        self.multi_model = YOLO(str(MULTI_MODEL_PATH))
        print(">>> Класи multi-моделі:", self.multi_model.names)
    def _detect(
        self,
        model: YOLO,
        image_bytes: bytes,
        conf_threshold: float,
    ) -> Dict[str, Any]:
        nparr = np.frombuffer(image_bytes, np.uint8)
        img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
        if img is None:
            raise ValueError("Не вдалося декодувати зображення")
        results = model(img)[0]
        detections: List[Dict[str, Any]] = []
        for box in results.boxes:
            conf = float(box.conf[0])
            if conf < conf_threshold:
                continue
            cls_id = int(box.cls[0])
            label = model.names.get(cls_id, str(cls_id))
            x1, y1, x2, y2 = box.xyxy[0].tolist()
            x1, y1, x2, y2 = map(int, [x1, y1, x2, y2])
            detections.append(
                {
```

```
                "class_id": cls_id,
                "label": label,
                "confidence": conf,
                "bbox": [x1, y1, x2, y2],
            }
        )
        cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
        text = f"{label} {conf:.2f}"
        cv2.putText(
            img,
            text,
            (x1, max(y1 - 5, 0)),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.5,
            (0, 255, 0),
            1,
            cv2.LINE_AA,
        )
    return {
        "annotated_image": img,
        "detections": detections,
    }
def detect_on_image(self, image_bytes: bytes, mode: str = "multi") -> Dict[str,
Any]:

    mode = mode.lower()
    if mode == "tank":
        return self._detect(self.tank_model, image_bytes, self.conf_tank)
    else:
```

```
        return self._detect(self.multi_model, image_bytes, self.conf_multi)
    def detect_on_frame(self, frame: np.ndarray, mode: str = "multi") -> Dict[str,
Any]:

        mode = mode.lower()
        if mode == "tank":
            model = self.tank_model
            conf_threshold = self.conf_tank
        else:
            model = self.multi_model
            conf_threshold = self.conf_multi
        img = frame.copy()
        results = model(img)[0]
        detections: List[Dict[str, Any]] = []
        for box in results.boxes:
            conf = float(box.conf[0])
            if conf < conf_threshold:
                continue
            cls_id = int(box.cls[0])
            label = model.names.get(cls_id, str(cls_id))
            x1, y1, x2, y2 = box.xyxy[0].tolist()
            x1, y1, x2, y2 = map(int, [x1, y1, x2, y2])
            detections.append(
                {
                    "class_id": cls_id,
                    "label": label,
                    "confidence": conf,
                    "bbox": [x1, y1, x2, y2],
                }
```

```
            )
            cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
            text = f"{label} {conf:.2f}"
            cv2.putText(
                img,
                text,
                (x1, max(y1 - 5, 0)),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.5,
                (0, 255, 0),
                1,
                cv2.LINE_AA,
            )
        return {
            "annotated_image": img,
            "detections": detections,
        }
    @staticmethod
    def encode_image_to_jpeg_bytes(image: np.ndarray) -> bytes:

        success, buffer = cv2.imencode(".jpg", image)
        if not success:
            raise ValueError("Не вдалося закодувати зображення у JPEG")
        return buffer.tobytes()
```