

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

**Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту**

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«__» _____ 20__ р.

Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системи і методи штучного
інтелекту»
спеціальності 122 «Комп'ютерні науки»
на тему: «Інтелектуальна веб-система автоматизованого виявлення та
класифікації наземних об'єктів на відео з БПЛА»

Виконав:

студент IV курсу, групи КІ-22

Комісаров Кирило Костянтинівич _____

Керівник:

доцент кафедри штучного інтелекту, к.ф.-м.н., проф.

Тимошенко Юрій Олександрович _____

Консультант з нормоконтролю:

фахівець першої категорії кафедри штучного інтелекту,

к.т.н., доц. Комариста Богдана Миколаївна _____

Рецензент:

доцент кафедри інформаційної безпеки, к.т.н., доцент

Гальчинський Леонід Юрійович _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент Комісаров К.К.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«15» січня 2026 р.

ЗАВДАННЯ
на дипломну роботу студенту
Коміссарову Кирилу Костянтиновичу

1. Тема роботи «Інтелектуальна веб-система автоматизованого виявлення та класифікації наземних об'єктів на відео з БПЛА», керівник роботи Тимошенко Юрій Олександрович професор кафедри штучного інтелекту, д.т.н., проф, затверджені наказом по НН ІПСА від «27» травня 2026 р. № 1944-с.
2. Термін подання студентом роботи «05» червня 2026 року.
3. Вихідні дані до роботи: наукові публікації у галузі комп'ютерного зору та архітектур нейронних мереж; документація до бібліотек і фреймворків FastAPI, Ultralytics, OpenCV; відкриті набори зображень (датасети) для навчання моделей; специфікація технології Server-Sent Events (SSE); середовище розробки Python, Docker.
4. Зміст роботи: вступ; аналіз методів та існуючих систем виявлення об'єктів на відео; обґрунтування вибору технологій та проектування архітектури системи; підготовка датасету та навчання моделі YOLO; реалізація клієнтської та серверної частин веб-сервісу; тестування роботи застосунку та оцінка метрик; висновки.

5. Перелік ілюстративного матеріалу: Рисунок 2.1 - схема нейронної мережі CNN (15 стр); Рисунок 2.2 - Схема алгоритму YOLO (18 стр); Рисунок 3.1 - Структурна схема застосунку (22 стр); Рисунок 3.2 - Блок-схема алгоритму обробки відео-потoku (24 стр); Рисунок 3.3 - Діаграма прецедентів системи (26 стр); Рисунок 4.2.1 - демонстративний приклад роботи моделі (32 стр); Рисунок 4.3.1 - Скріншот головного екрана інтерфейсу UAV-VISION Core (34 стр); Рисунок 4.4.1 - Успішна детекція техніки при $imgsz=832$ (36 стр); Рисунок 4.7.1 - тестування моделі на солдатах з камуфляжем (39 стр); Рисунок 4.7.2 - невдалі кадри з тестування (40 стр); Рисунок 4.7.3 - відео щільного скупчення техніки та осіб (41 стр); Рисунок 4.7.4 - результат тестування відео з танком на ґрунтовій дорозі (42 стр)

6. Дата видачі завдання «02» лютого 2026 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Збір та аналіз літературних джерел, аналіз існуючих аналогів систем детекції об'єктів з БПЛА.	20.04.2026	Виконано
2	Обґрунтування вибору технологій та проектування архітектури системи.	27.04.2026	Виконано
3	Збір, розмітка та підготовка універсального військового датасету, застосування аугментації даних.	04.05.2026	Виконано
4	Донавчання нейромережі YOLOv8 та розв'язання проблеми катастрофічного забування моделі.	11.05.2026	Виконано
5	Розробка серверної частини веб-сервісу на FastAPI та інтеграція технології Server-Sent Events (SSE).	18.05.2026	Виконано
6	Розробка клієнтського веб-інтерфейсу за принципом Single Page Application та тестування OpenCV.	25.05.2026	Виконано
7	Проведення загального тестування системи, оцінка фінальних метрик точності (mAP) та швидкості (FPS).	01.06.2026	Виконано
8	Оформлення тексту пояснювальної записки, підготовка презентації та доповіді до захисту.	05.06.2026	Виконано

Студент

Кирило КОМИССАРОВ

Керівник

Олександр ТИМОШЕНКО

РЕФЕРАТ

Дипломна робота: 50 с., 12 рис., 25 посилань, додаток.

БПЛА, КОМП'ЮТЕРНИЙ ЗІР, ВИЯВЛЕННЯ ОБ'ЄКТІВ, НЕЙРОННІ МЕРЕЖІ, YOLO, FASTAPI, ВЕБ-СЕРВІС, ДАТАСЕТ, ВІДЕОАНАЛІТИКА, DOCKER.

Об'єктом дослідження є процес автоматизованого розпізнавання військової техніки на відеоданих з БПЛА.

Предметом дослідження є методи машинного навчання та програмні засоби для розробки веб-сервісу детекції об'єктів.

Метою роботи є створення швидкого та зручного веб-сервісу для знаходження військової техніки на відео, що знизить когнітивне навантаження операторів.

У роботі проаналізовано існуючі системи детекції та популярні архітектури нейромереж, обґрунтовано вибір моделі YOLOv8-small. Для усунення проблеми катастрофічного забування нейромережі зібрано та підготовлено власний змішаний датасет (близько 7000 фотографій), після чого успішно донавчено фінальну модель.

Розроблено SPA веб-застосунок мовою Python з використанням фреймворку FastAPI та бібліотеки OpenCV для покадрової обробки відео. Запропоновано технологію SSE для відображення прогресу в реальному часі. Проєкт успішно контейнеризовано за допомогою Docker.

В результаті тестування отримано точність детекції за метрикою mAP@0.5 на рівні 48%. Встановлено, що швидкість обробки відеопотоку досягає 5-7 кадрів на секунду, чого достатньо для ефективного покадрового аналізу.

ABSTRACT

Bachelor's thesis: 50 p., 12 figures, 25 references, appendix.

UAV, COMPUTER VISION, OBJECT DETECTION, NEURAL NETWORKS, YOLO, FASTAPI, WEB SERVICE, DATASET, VIDEO ANALYTICS, DOCKER.

The object of the study is the process of automated recognition of military equipment in video data from UAVs.

The subject of research is machine learning methods and software tools for developing an object detection web service.

The purpose of the work is to create a fast and convenient web service for finding military equipment on video, which will reduce the cognitive load on operators.

The paper analyzes existing detection systems and popular neural network architectures, and justifies the choice of the YOLOv8-small model. To eliminate the problem of catastrophic forgetting of the neural network, a custom mixed dataset (about 7000 photos) was collected and prepared, after which the final model was successfully fine-tuned.

A SPA web application was developed in Python using the FastAPI framework and the OpenCV library for frame-by-frame video processing. SSE technology was proposed to display progress in real time. The project was successfully containerized using Docker.

As a result of testing, the detection accuracy according to the mAP@0.5 metric was obtained at 48%. It was established that the video stream processing speed reaches 5-7 frames per second, which is sufficient for effective frame-by-frame analysis and relieving the routine burden on the human operator.

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЇ.....	9
1.1 Аналіз використання БПЛА та систем комп'ютерного зору у сучасних умовах.....	9
1.2 Огляд існуючих рішень та їх недоліки.....	10
1.2.1 Великі комерційні та військові комплекси.....	10
1.2.2 Хмарні сервіси комп'ютерного зору.....	11
1.2.3 Звичайні системи відеоспостереження з функцією детекції руху.....	11
1.3 Порівняльний аналіз алгоритмів розпізнавання об'єктів.....	12
1.3.1 Faster R-CNN (Region-based Convolutional Neural Networks).....	13
1.3.2 SSD (Single Shot MultiBox Detector).....	13
1.3.3 YOLO (You Only Look Once).....	14
1.4 Обґрунтування вибору програмного стеку.....	14
1.4.1 Мова програмування: Python.....	15
1.4.2 Веб-фреймворк: FastAPI.....	15
1.4.3 Технологія контейнеризації: Docker.....	16
1.4.4 Клієнтська частина (Frontend).....	16
Висновок до розділу 1.....	17
РОЗДІЛ 2 ТЕОРЕТИЧНІ ОСНОВИ ТА МАТЕМАТИЧНІ МОДЕЛІ	
АЛГОРИТМІВ КОМП'ЮТЕРНОГО ЗОРУ.....	18
2.1 Основи комп'ютерного зору та згорткових нейронних мереж (CNN).....	18
2.2 Огляд алгоритмів виявлення об'єктів (Object Detection).....	19
2.3 Архітектура та принцип роботи сімейства YOLO.....	20
2.4. Метрики оцінки якості моделей комп'ютерного зору.....	21
Висновок до розділу 2.....	22

РОЗДІЛ 3 ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ ТА

АЛГОРИТМІВ РОБОТИ.....	23
3.1 Загальна архітектура веб-застосунку.....	23
3.2 Алгоритм роботи модуля комп'ютерного зору.....	24
3.3 Моделювання поведінки системи (UML-діаграми).....	26
3.4 Проєктування інтерфейсу користувача (UI/UX).....	27
Висновок до розділу 3.....	28

РОЗДІЛ 4 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

ПРОГРАМНОГО ЗАСТОСУНКУ.....	30
4.1 Етап базового тестування та аналіз проблематики.....	30
4.2 Формування та підготовка спеціалізованого набору даних.....	31
4.3 Програмна реалізація інтерфейсу користувача.....	32
4.4 Вибір архітектури нейронної мережі та оптимізація розміру вхідного тензора.....	34
4.5 Контейнеризація та розгортання програмного комплексу на базі хмарних рішень.....	35
4.6 Аналіз результатів практичного тестування.....	36
Висновок до розділу 4.....	40

ВИСНОВКИ.....	42
----------------------	-----------

ПЕРЕЛІК ПОСИЛАНЬ.....	43
------------------------------	-----------

ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО КОДУ ВЕБ-ЗАСТОСУНКУ ТА КОНФІГУРАЦІЙНИХ ФАЙЛІВ.....	46
---	-----------

ВСТУП

Актуальність теми. Протягом останніх років безпілотні літальні апарати (БПЛА) стали одним із найголовніших інструментів для проведення розвідки, моніторингу територій та пошуково-рятувальних операцій¹. Кількість дронів постійно зростає, а разом із цим зростає і обсяг відеоданих, які вони передають. Зазвичай ці відео в реальному часі або в записі аналізує людина - оператор БПЛА.

Головна проблема полягає в тому, що оператор змушений годинами безперервно дивитися в монітор, намагаючись знайти невеликі об'єкти (наприклад, замасковану наземну техніку або людей) з нестандартного ракурсу та при різному освітленні. Через втому та зниження концентрації уваги різко зростає вплив людського фактора - оператор може просто не помітити важливу ціль.

Саме тому виникає гостра необхідність у системах, які б автоматично аналізували відеопотік та підказували людині, де знаходиться об'єкт. Сучасні технології машинного навчання та комп'ютерного зору здатні виконувати цю рутинну роботу значно швидше за людину і без втрати уваги. Розробка зручного веб-застосунку, який би дозволяв завантажити відео з дрона і на виході отримати готовий матеріал із виділеними об'єктами, є вкрай актуальним завданням, оскільки це безпосередньо знижує когнітивне навантаження на операторів та підвищує загальну ефективність їхньої роботи.

Мета дослідження. Метою дипломної роботи є розробка програмного веб-застосунку для автоматизованого виявлення та класифікації наземних

¹Тут і нижче використані такі інструменти штучного інтелекту як чат-бот з генеративним штучним інтелектом ChatGPT виключно для коригування та редагування тексту, створеного автором цієї дипломної роботи, на основі автоматизованої перевірки граматики, структури та стилю, що відповідає Політиці використання штучного інтелекту для академічної діяльності в КПІ ім. Ігоря Сікорського (протокол №11 Вченої ради КПІ ім. Ігоря Сікорського від 11 грудня 2023 р.).

об'єктів на відеозаписах з БПЛА з використанням алгоритмів комп'ютерного зору.

Хід роботи. Для реалізації поставленої задачі необхідно виконати наступні кроки:

- провести аналіз предметної області та дослідити існуючі підходи до розпізнавання об'єктів на відео з повітря;
- обрати та налаштувати оптимальну модель нейронної мережі для швидкої та точної детекції наземних об'єктів;
- розробити серверну частину, яка буде відповідати за обробку завантаженого відеофайлу та взаємодію з навченною нейро моделлю;
- створити зручний інтерфейс для користувача;
- забезпечити легкість та портативність у розгортанні застосунку за допомогою технології контейнеризації (Docker).

Практичне значення одержаних результатів. Результатом роботи стане готовий до використання програмний продукт. Його можна застосовувати як допоміжний інструмент для фахівців, що працюють з аеророзвідкою або моніторингом місцевості. Використання контейнеризації (Docker) дозволяє легко розгорнути цю систему на будь-якому сервері чи навіть на локальному комп'ютері без складнощів у налаштуванні оточення, що робить збірку максимально підходячою для всіх.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЇ

1.1 Аналіз використання БПЛА та систем комп'ютерного зору у сучасних умовах

В останні роки безпілотні літальні апарати (БПЛА) перетворилися з вузькоспеціалізованої техніки на масовий та незамінний інструмент у багатьох сферах діяльності. Особливого значення вони набули у військовій справі, охоронній діяльності, під час проведення пошуково-рятувальних операцій, а також у моніторингу промислових і сільськогосподарських об'єктів. Головною перевагою БПЛА є здатність швидко та відносно безпечно отримувати візуальні дані з важкодоступних або небезпечних територій у режимі реального часу.

Зі зростанням технічних можливостей дронів, обсяг інформації, що передається на пульт оператора, дуже сильно виріс. Це призвело до виникнення нової проблеми – інформаційного та когнітивного перевантаження людини-оператора.

Традиційний підхід до аналізу відеоданих з БПЛА передбачає безперервне візуальне спостереження з боку фахівця. Оператор змушений годинами аналізувати зображення, намагаючись виявити цільові об'єкти, особливо це важко коли складні погодні умови, погане освітлення, малі розміри об'єктів на екрані і так далі. Внаслідок цього ефективність ручного моніторингу суттєво знижується вже після першої години роботи. Виникає ризик пропуску критично важливих цілей або хибної ідентифікації об'єктів.

Вирішенням цієї проблеми є впровадження системи комп'ютерного зору та алгоритмів штучного інтелекту безпосередньо в процес обробки відеопотоку. Сучасні методи машинного навчання дозволяють автоматизувати процес детекції. Програмний алгоритм здатен покадрово аналізувати відео, виділяти контури об'єктів, класифікувати їх та накладати на екран відповідні маркери ще до того, як людина встигне їх помітити.

Таким чином, розробка програмного застосунку для автоматизованого аналізу відео з БПЛА є великою необхідністю у сьогодення. Використання комп'ютерного зору дозволяє перекласти рутинну роботу машині, залишаючи за людиною лише функцію прийняття остаточних рішень на основі вже оброблених даних.

1.2 Огляд існуючих рішень та їх недоліки

Щоб обґрунтувати необхідність розробки веб-застосунку, необхідно проаналізувати вже існуючі на ринку рішення для аналізу відео з БПЛА. Усі поточні системи можна умовно поділити на три великі категорії, кожна з яких має свої недоліки в контексті завдання.

1.2.1 Великі комерційні та військові комплекси

Це системи рівня Palantir або спеціалізоване закрите ПЗ, яке йде в комплекті з дорогими розвідувальними безпілотниками. Звісно, такі рішення мають величезний функціонал: вони забезпечують високу точність детекції, вміють будувати складні маршрути і можуть аналізувати відео одразу з десятків дронів, зводячи все в єдину картину.

Але в цій масштабності криється їхній головний недолік. По-перше, такі системи є надзвичайно дорогими і вимагають розгортання цілих мобільних серверних центрів для обробки даних. По-друге, інтерфейс там настільки перевантажений, що оператору потрібно проходити довге спеціальне навчання, щоб просто почати працювати. Окремі підрозділи, невеликі мобільні групи або волонтери просто не мають доступу до такого рівня програмного забезпечення.

І навіть якби мали, такий софт фізично неможливо запустити на звичайному ноутбукі де у польових умовах - комп'ютер просто не витягне таких навантажень, або система відмовиться працювати без підключення до закритої військової мережі.

1.2.2 Хмарні сервіси комп'ютерного зору

Прикладами таких систем є Google Cloud Vision або AWS Rekognition. Тобто, це системи, які працюють віддалено на серверах та добре виконують свою роботу, тим паче, що засновані вони від компаній-гігантів. Але головна проблема - критична залежність від швидкого та стабільного інтернету. У реальних умовах інтернет часто відсутній, або працює з перебоями. Крім того, передавати конфіденційні кадри з позиціями чи технікою на комерційні сервери - це відверто погана ідея з точки зору безпеки та витоку інформації. Також не варто забувати про фінансову складову: такі платформи тарифікують кожен секунду або кожен кадр обробленого відео. Якщо дрон літає і стрімить годинами, використання хмарних API швидко вилітєть у величезні рахунки.

1.2.3 Звичайні системи відеоспостереження з функцією детекції руху

Багато стандартних програм для камер вміють реагувати на рух у кадрі. Але вони занадто примітивні, такі алгоритми не вміють класифікувати об'єкти, програма буде сигналізувати про рух навіть якщо пролетів птах, або похитнулася листва на дереві від вітру, або освітлення змінилося і так далі. Такі системи абсолютно не надійні для аналізу відео з дрона.

Аналізуючи існуючі рішення для розпізнавання відео з дронів, можна помітити певну прогалину на ринку програмного забезпечення. Зараз доступні або дуже складні та дорогі корпоративні системи, або ж розрізнені скрипти, для запуску яких користувачеві треба добре розумітися на програмуванні та роботі з консоллю.

Звісно, сама тематика виявлення об'єктів з БПЛА не є абсолютно новою. Вона вже досліджувалася раніше, зокрема і в попередніх дипломних роботах студентів. Усвідомлюючи цей досвід та спираючись на існуючі напрацювання, у даній дипломній роботі пропонується підійти до задачі з боку зручності кінцевого користувача.

Мета проєкту - розробити повноцінний клієнт-серверний веб-застосунок на базі сучасних ML-моделей. Фокус робиться на тому, щоб створити готовий програмний продукт, де будь-який оператор дрона, або будь-який користувач міг би просто відкрити браузер, завантажити відеофайл і автоматично отримати результат з розпізнаними об'єктами у зручному інтерфейсі, без необхідності розбиратися в коді чи налаштуваннях середовища.

1.3 Порівняльний аналіз алгоритмів розпізнавання об'єктів

Для реалізації базового функціоналу веб-застосунку необхідно обрати архітектуру нейронної мережі, яка здатна вирішувати задачу детекції об'єктів, тобто, не лише визначати, що знаходиться в кадрі, а й обводити цей об'єкт рамкою із зазначеними координатами.

На сьогоднішній день існує три основні сімейства алгоритмів: Faster R-CNN, SSD та YOLO [13, 23, 25]. Розглянемо їх детальніше в контексті завдання.

1.3.1 Faster R-CNN (Region-based Convolutional Neural Networks)

Цей алгоритм працює у два етапи. Спочатку спеціальна внутрішня підмережа (її ще називають Region Proposal Network) сканує все зображення та генерує гіпотези - тобто вона просто пропонує прямокутні регіони на картинці, де суто теоретично може знаходитися якийсь об'єкт. Після цього алгоритм ніби "вирізає" ці шматки кадрів, і вже на другому етапі інша частина нейромережі окремо класифікує кожен з цих виділених регіонів та уточнює їхні межі.

Переваги: Дуже висока точність розпізнавання. Оскільки мережа фактично двічі дивиться на об'єкт, вона добре справляється з пошуком навіть тих цілей, які частково перекриті іншими предметами або знаходяться на складному фоні.

Недоліки: Через оцю двоетапність алгоритм є занадто важким і повільним для цієї задачі. Він чудово підходить для детального аналізу статичних фотографій, або коли є час почекати на результат. Але для обробки відеопотоку з дрона в режимі реального часу він не підходить. Щоб відео не гальмувало і видавало хоча б мінімально стабільний FPS, алгоритм вимагає надзвичайно потужних і дорогих серверних відеокарт. В реальних польових умовах оператори зазвичай працюють з портативних станцій або звичайних ноутбуків, які просто фізично не витягнуть таку архітектуру.

1.3.2 SSD (Single Shot MultiBox Detector)

На відміну від Faster R-CNN, цей алгоритм є однопрохідним. Він обробляє все зображення за один раз, що робить його значно швидшим.

Переваги: Висока швидкість роботи, менше навантаження на процесор.

Недоліки: В архітектурі SSD є проблема - вона досить погано розпізнає дрібні

об'єкти. Оскільки камера БПЛА зазвичай знімає з великої висоти, військова техніка або люди в кадрі часто займають лише кілька десятків пікселів. SSD може їх просто проігнорувати, тому для аеророзвідки це не найкращий вибір.

1.3.3 YOLO (You Only Look Once)

Назва алгоритму перекладається як "Ти дивишся лише один раз". Мережа розбиває зображення на сітку і миттєво, за один прохід, передбачає класи об'єктів та їхні координати для кожної клітинки.

Переваги: YOLO забезпечує найкращий компроміс між швидкістю та точністю. Сучасні версії цієї архітектури (YOLOv8, YOLOv10) здатні обробляти відео в реальному часі навіть на відносно слабкому обладнанні або звичайних ноутбуках. До того ж, останні версії YOLO мають спеціальні механізми, які чудово справляються з розпізнаванням дрібних об'єктів під різними кутами, що ідеально підходить для відео з повітря.

Отже, беручи до уваги специфіку задачі - необхідність швидкої, покадрової обробки відеоданих та вимогу щодо точного виявлення відносно невеликих об'єктів з нестандартного ракурсу - для розробки системи було обрано архітектуру YOLO. Вона дозволить застосунку працювати швидко, не перевантажуючи сервер або комп'ютер користувача.

1.4 Обґрунтування вибору програмного стеку

Для створення повноцінного веб-застосунку, який об'єднує в собі роботу неймережі та зручний інтерфейс для користувача, необхідно було обрати

оптимальний набір технологій. Рішення приймалося з огляду на вимоги до швидкодії та простоти розгортання.

1.4.1 Мова програмування: Python

Основою для серверної частини та модулю машинного навчання було обрано мову Python [20]. На сьогоднішній день це безперечний лідер та стандарт у сфері штучного інтелекту та комп'ютерного зору. Більшість сучасних бібліотек, необхідних для роботи (зокрема OpenCV [1] для обробки кадрів та PyTorch [21] для роботи з моделлю YOLO), мають високу стабільну підтримку саме в Python. Використання інших мов для цієї задачі значно ускладнило б розробку та інтеграцію нейромережі.

Крім того, вибір Python сильно зекономив час на розробку. Для реалізації цього проекту дуже важливо мати змогу швидко перевіряти різні ідеї. У Python немає такої складної роботи з пам'яттю чи типами даних, як, наприклад, у C++ або Java. Завдяки цьому можна зосередитися саме на налаштуванні нейромережі YOLOv8 та логіці роботи веб-сервісу, а не витрачати багато часу на виправлення системних помилок у коді.

Ще один плюс - це універсальність. Завдяки Python не доведеться писати модуль нейромережі на одній мові, а бекенд сайту - на іншій. Серверна частина успішно працює на FastAPI, який чудово поєднується з машинним навчанням. Це спростить процес пакування всього готового проекту в Docker-контейнер, оскільки не треба налаштовувати складні зв'язки між різними середовищами розробки.

1.4.2 Веб-фреймворк: FastAPI

Для побудови серверної логіки розглядалися популярні фреймворки Django, Flask та FastAPI [7, 9, 11]. Django є занадто масивним та важким для створення мікросервісного застосунку. Flask є легким, але застарілим у плані роботи з асинхронним кодом. Тому вибір зупинився на FastAPI. Цей фреймворк є одним із найшвидших сучасних інструментів для Python. Він підтримує асинхронність з слабких девайсів, що критично важливо для паралельної обробки потоку відеокадрів, а також автоматично генерує документацію до коду.

1.4.3 Технологія контейнеризації: Docker

Однією з головних проблем при розробці систем з машинним навчанням є конфлікти залежностей: специфічні версії бібліотек, драйверів відеокарт та системних пакетів. Налаштування такого оточення на новому комп'ютері може зайняти багато часу. Щоб уникнути класичної проблеми "на моєму комп'ютері це працювало", було вирішено використати Docker [12]. Цей інструмент дозволяє "запакувати" застосунок разом з усіма бібліотеками та налаштуваннями у єдиний контейнер. Завдяки цьому систему можна миттєво розгорнути на будь-якому сервері чи ноутбучі з будь-якою операційною системою без ручного встановлення Python чи інших компонентів. Таке рішення є вкрай ефективним, оскільки позбавляє користувача від складного налаштування та зберігає його час.

1.4.4 Клієнтська частина (Frontend)

Для реалізації інтерфейсу користувача будуть використані базові веб-технології (HTML, CSS, JavaScript [5, 14, 17]). Це дозволить створити легку сторінку в браузері для завантаження відео та перегляду результатів без необхідності встановлювати додаткове програмне забезпечення на комп'ютер користувача.

Таким чином, було досліджено та обрано технічний стек для подальшої реалізації проекту.

Висновок до розділу 1

У першому розділі було проведено детальний аналіз предметної області та досліджено проблематику обробки відеоданих, отриманих з безпілотних літальних апаратів. Встановлено, що ручний моніторинг відеопотоку оператором має суттєві недоліки через вплив людського фактора, такі як втома, зниження уваги, що призводить до пропуску важливих об'єктів. Аналіз існуючих програмних рішень показав, що на ринку бракує легких, автономних систем, здатних працювати локально без залежності від потужного інтернет-з'єднання. Для вирішення цього завдання було обрано архітектуру згорткової нейронної мережі YOLO, яка забезпечує найкращий баланс між високою швидкістю та точністю детекції об'єктів на відео. Визначено та обґрунтовано програмний стек розробки, основою якого є мова Python та фреймворк FastAPI. Використання технології Docker забезпечить портативність та легкість розгортання майбутньої системи.

РОЗДІЛ 2 ТЕОРЕТИЧНІ ОСНОВИ ТА МАТЕМАТИЧНІ МОДЕЛІ АЛГОРИТМІВ КОМП'ЮТЕРНОГО ЗОРУ

2.1 Основи комп'ютерного зору та згорткових нейронних мереж (CNN)

Для того щоб автоматизувати процес пошуку об'єктів на відео з безпілотників, необхідно розуміти базові принципи роботи комп'ютерного зору. Для комп'ютера будь-яке зображення або кадр з відео – це просто матриця чисел, де кожне число відповідає за яскравість певного пікселя.

Раніше розробникам доводилося вручну писати алгоритми, які шукали на цих матрицях перепади контрастності, щоб знайти краї об'єктів чи специфічні форми. Сьогодні цей підхід майже не використовується, оскільки на зміну йому прийшло глибоке навчання (Deep Learning) [6], а саме - згорткові нейронні мережі (CNN) [4].

Головна перевага CNN полягає в тому, що мережа сама вчиться знаходити важливі ознаки на зображенні під час тренування. Вона пропускає картинку через набір математичних фільтрів (згортки). Перші шари нейромережі зазвичай бачать прості речі: лінії, кути або зміни кольору. Глибші шари комбінують ці лінії у складніші фігури (наприклад, колесо або дах автомобіля), а останні шари вже приймають рішення щодо того, який саме об'єкт знаходиться на картинці. Такий підхід ідеально підходить для відео з БПЛА, оскільки об'єкти можуть бути під різним кутом, частково перекриті або мати різний масштаб.

Схема нейронної мережі CNN зображено на рисунку 2.1.

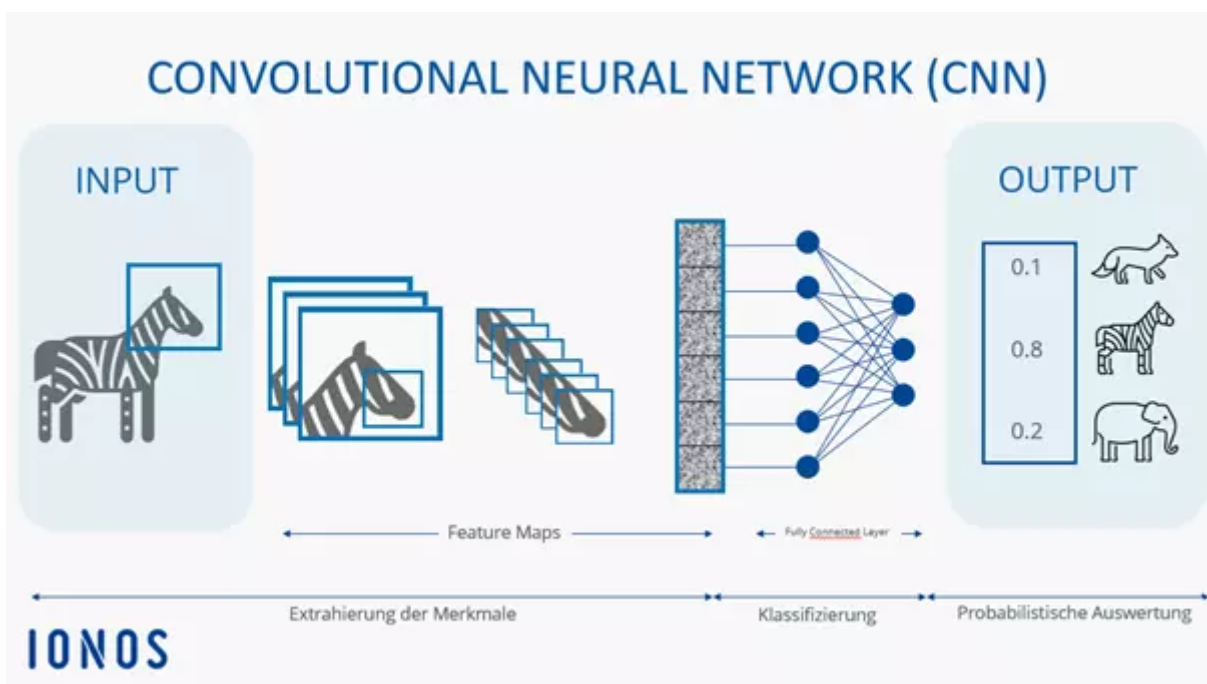


Рисунок 2.1 – Схема нейронної мережі CNN²

2.2 Огляд алгоритмів виявлення об'єктів (Object Detection)

Варто розрізняти дві базові задачі комп'ютерного зору: класифікацію зображень та виявлення об'єктів. Якщо класифікація просто відповідає на питання «Що зображено на фото?», то виявлення об'єктів (Object Detection [19]) має відповісти на питання «Де саме знаходиться об'єкт і що це таке?». Тобто результатом роботи алгоритму є координати обмежувальної рамки (bounding box [24]) та мітка класу.

На сьогоднішній день алгоритми виявлення об'єктів діляться на два основні типи.

Двостадійні (Two-stage detectors). Наприклад, алгоритми сімейства R-CNN. Вони працюють у два етапи: спочатку інший алгоритм пропонує

²Джерело зображення: URL

https://www.ionos.com/digitalguide/fileadmin/_processed_/b/e/csm_schaubild-convolutional-neural-network_e9402d4c16.webp

регіони на зображенні, де може знаходитися об'єкт, а потім неймережа класифікує кожен з цих регіонів. Вони дуже точні, але працюють досить повільно.

Одностадійні (One-stage detectors). Наприклад, SSD та YOLO. Вони аналізують все зображення за один прохід і одночасно передбачають і координати рамок, і класи об'єктів.

Оскільки в нашій задачі відео надходить з дрона (де часто потрібна обробка в реальному часі або з мінімальними затримками), використання двостадійних алгоритмів є недоцільним через їхню повільність. Тому для розробки системи було обрано одностадійний підхід.

2.3 Архітектура та принцип роботи сімейства YOLO

Принцип роботи YOLO кардинально відрізняється від класичних алгоритмів. Неймережа розбиває вхідне зображення на умовну сітку (наприклад, розміром 13x13 клітинок). Якщо центр потрібного нам об'єкта (скажімо, автомобіля) потрапляє в певну клітинку цієї сітки, то саме ця клітинка стає відповідальною за його розпізнавання.

Кожна клітинка сітки намагається передбачити кілька обмежувальних рамок навколо себе та розрахувати ступінь впевненості (Confidence Score) в тому, що всередині рамки є об'єкт. Паралельно мережа рахує ймовірність того, до якого саме класу належить цей об'єкт. Вибір YOLOv8 для цього проєкту пояснюється її архітектурною гнучкістю. Хоча в ідеальних умовах на потужних відеокартах модель здатна видавати дуже високий FPS, для даної роботи важливішою була можливість стабільного запуску на доступному залізі та хмарних сервісах. Навіть при обмежених ресурсах (наприклад, використанні лише CPU), модель забезпечує прийнятну швидкість обробки кадрів, якої достатньо для коректної детекції об'єктів без критичних затримок. Це дозволяє

впевнено знаходити дрібні деталі на відео з дрона, зберігаючи баланс між точністю розпізнавання та навантаженням на систему

Схема алгоритму YOLO зображено на рисунку 2.2.

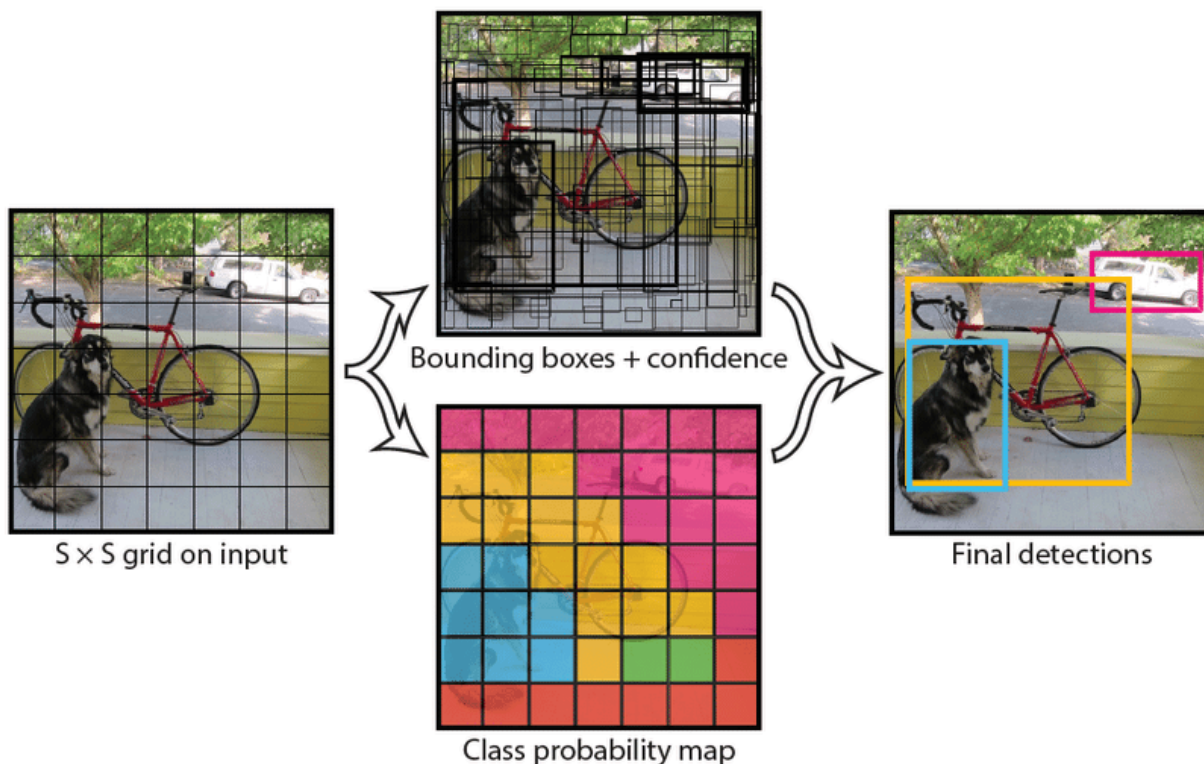


Рисунок 2.2 – Схема алгоритму YOLO³

2.4. Метрики оцінки якості моделей комп'ютерного зору

Щоб об'єктивно оцінити, наскільки добре навчена модель знаходить об'єкти на відео, використовується стандартизований набір метрик.

Базовим поняттям для оцінки є **IoU (Intersection over Union)** - це відношення площі перетину знайденої рамки з реальною рамкою об'єкта до

³Джерело зображення: URL

https://miro.medium.com/v2/resize:fit:720/format:webp/1*15uBgdR3_rNZzx665Leang.jpeg

їхньої загальної площі [16]. Якщо IoU дорівнює 1, це означає ідеальне співпадання рамок. Зазвичай, якщо IoU перевищує 0.5, вважається, що об'єкт знайдено коректно.

На основі IoU розраховуються дві головні метрики:

- **точність (Precision)** показує, яка частка з того, що алгоритм назвав "об'єктом", дійсно є цим об'єктом (дозволяє оцінити кількість хибних спрацювань);

- **повнота (Recall)** показує, яку частку реальних об'єктів на кадрі алгоритм зміг знайти (дозволяє оцінити, чи багато об'єктів модель "пропускає").

Для комплексної оцінки алгоритму використовують метрику **mAP (mean Average Precision)** — середнє значення точності для всіх класів об'єктів при різних порогах IoU [18]. Саме ця метрика є загальноприйнятим стандартом для порівняння різних версій моделей комп'ютерного зору.

Висновок до розділу 2

У другому розділі було розглянуто теоретичні основи розв'язання задачі комп'ютерного зору. Аналіз існуючих підходів показав, що для обробки відеоданих з БПЛА найоптимальнішим вибором є використання одностадійних згорткових нейронних мереж, зокрема архітектури YOLO, завдяки її високій швидкості роботи та здатності знаходити об'єкти за один прохід алгоритму. Також було визначено ключові метрики (IoU, mAP, Precision та Recall), за допомогою яких в подальшому буде оцінюватися ефективність розробленого модуля розпізнавання.

РОЗДІЛ 3 ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ ТА АЛГОРИТМІВ РОБОТИ

3.1 Загальна архітектура веб-застосунку

Для забезпечення надійної та швидкої роботи системи розпізнавання об'єктів на відео з БПЛА, було обрано класичну клієнт-серверну архітектуру. Вона дозволяє чітко розділити зону відповідальності між інтерфейсом користувача, логікою обробки запитів та обчислювальними процесами нейронної мережі.

Архітектурно застосунок можна поділити на три модулі.

Клієнтський модуль(Frontend). Це веб-інтерфейс, з яким буде взаємодіяти користувач/оператор БПЛА. Головна задача - зробити інтерфейс максимально зручним та зрозумілим для використання, відображати результат аналізу.

Серверний модуль (Backend). Серверна частина виступає головним координатором системи. Вона приймає відеодані від клієнта, проводить їх валідацію та тимчасово зберігає для подальшої обробки. Завдяки асинхронній природі фреймворку FastAPI, сервер здатен обробляти кілька запитів одночасно, не блокуючи роботу інтерфейсу. Після завершення аналізу сервер формує відповідь і відправляє готове відео назад користувачу.

Модуль комп'ютерного зору (ML-модуль). Це обчислювальна основа застосунку, побудоване на базі архітектури YOLO. Серверний модуль передає йому завантажене відео, після чого ML-модуль розбиває відеоряд на окремі кадри. Кожен кадр пропускається через нейронну мережу, яка виконує детекцію: знаходить об'єкти, визначає їхній клас (наприклад, людина або транспортний засіб), обчислює координати меж та ймовірність правильного розпізнавання. Опрацьовані кадри знову зшиваються у відеофайл і повертаються на сервер.

Загальний вигляд взаємодії компонентів виглядає так:

- користувач через веб-інтерфейс завантажує відеофайл;
- з клієнтського модуля (Frontend) відправляється відео на серверний модуль (Backend);
- Backend приймає файл і передає його до ML-модуля для покадрового аналізу;
- нейромережа обробляє відео, накладає рамки та повертає на серверний модуль;
- Backend повертає готове відео користувачу у браузері.

Наглядна структурна схема застосунку зображена на рисунку 3.1.

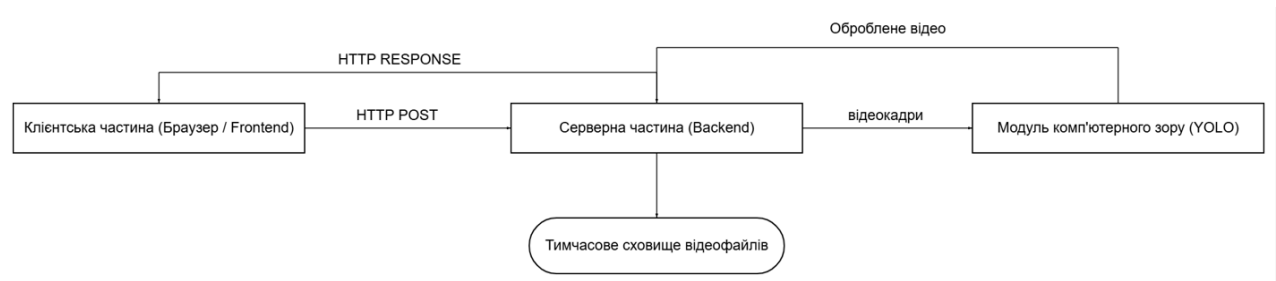


Рисунок 3.1 – Структурна схема застосунку

3.2 Алгоритм роботи модуля комп'ютерного зору

Як було зазначено раніше, основним мозком системи є модель YOLO. Проте сама по собі нейронна мережа не здатна обробляти відеофайли як єдине ціле – вона працює виключно зі статичними зображеннями (матрицями пікселів). Тому для аналізу відеопотоку було розроблено покроковий алгоритм обробки, який реалізується за допомогою бібліотеки комп'ютерного зору OpenCV.

Весь процес обробки одного відеофайлу можна розділити на п'ять логічних кроків.

Декодування та розбиття на кадри. Після отримання відеофайлу програма відкриває його і починає читати послідовно кадр за кадром.

Підготовка зображення. Відео з БПЛА часто має дуже високу роздільну здатність (наприклад, 4K). Обробляти такі великі зображення дуже довго, тому кожен отриманий кадр стискається до стандартного розміру, оптимізованого для нашої моделі YOLO (наприклад, 832x640 пікселів). Це значно пришвидшує роботу системи без критичної втрати якості розпізнавання.

Прохід через нейромережу. Підготовлений кадр передається моделі YOLO. Нейромережа аналізує зображення за один прохід і повертає масив даних про знайдені об'єкти. Для кожного об'єкта алгоритм видає:

- координати меж, щоб зрозуміти, де малювати рамку;
- клас об'єкта (наприклад людина або військова техніка);
- рівень впевненості (наскільки модель впевнена у своєму рішенні).

Фільтрація результатів. Щоб уникнути хибних результатів, модель буде відсіювати всі результати, в яких рівень впевненості менший за поріг (наприклад, менше 40%).

Збирання нового відео. Після нанесення маркерів нові кадри збираються у новий відеофайл.

Такий покадровий підхід дозволяє ефективно обробляти відеофайли будь-якої тривалості, при цьому не перевантажувати оперативну пам'ять сервера, оскільки в один момент часу в пам'яті тримається лише один поточний кадр.

Більш детальна блок-схема алгоритму обробки відео-потoku зображено на рисунку 2.2.

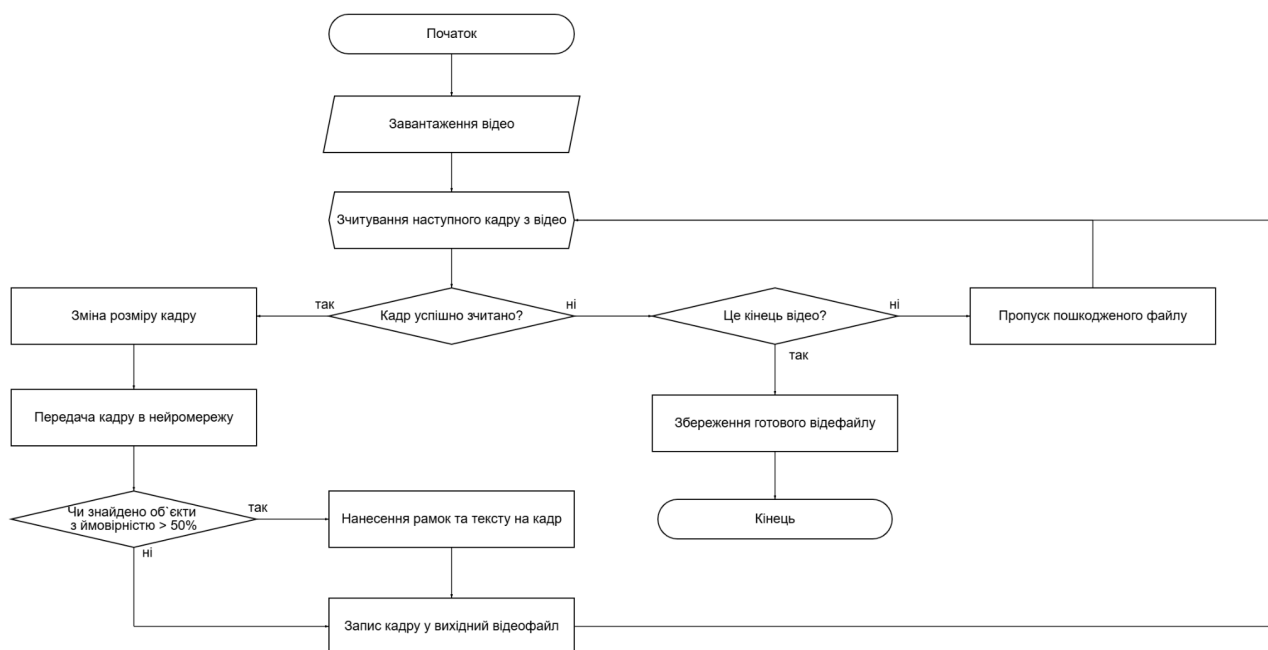


Рисунок 3.2 – Блок-схема алгоритму обробки відео-потoku

3.3 Моделювання поведінки системи (UML-діаграми)

Для проектування логіки взаємодії між користувачем та ПЗ було використано уніфіковану мову моделювання (UML). Щоб наочно продемонструвати, які саме функції будуть доступні в готовому застосунку, побудовано діаграму прецедентів.

Головною дійовою особою у нашій системі є оператор БПЛА, або звичайний користувач. Оскільки система проектується максимально легкою, оператор одразу отримує доступ до основного функціоналу.

Основні прецеденти (дії), які може виконувати користувач:

- завантаження відеофайлу: користувач обирає файл на своєму пристрої та передає його на сервер;
- перегляд результату: після завершення аналізу оператор може візуально оцінити знайдені об'єкти (військову техніку, людей) прямо у веб-плеєрі браузера;

– завантаження обробленого відео: збереження готового файлу на локальний диск для подальшого використання або передачі в штаб.

Кожна дія користувача запускає внутрішні процеси системи, які є обов'язковими, але прихованими від очей користувача. Наприклад, прецедент «Завантаження відеофайлу» описує процес передачі файлу до серверної частини, де він готується для подальшого покадрового аналізу нейромережею. А прецедент «Перегляд результату» стає можливим лише після того, як система ініціалізує модель YOLO та виконає покадровий аналіз.

Таке моделювання дозволяє чітко окреслити межі проекту: створити зручний інструмент для швидкої обробки відео, без перевантаження інтерфейсу зайвими функціями.

UML діаграма функціоналу застосунку зображено на рисунку 3.3.

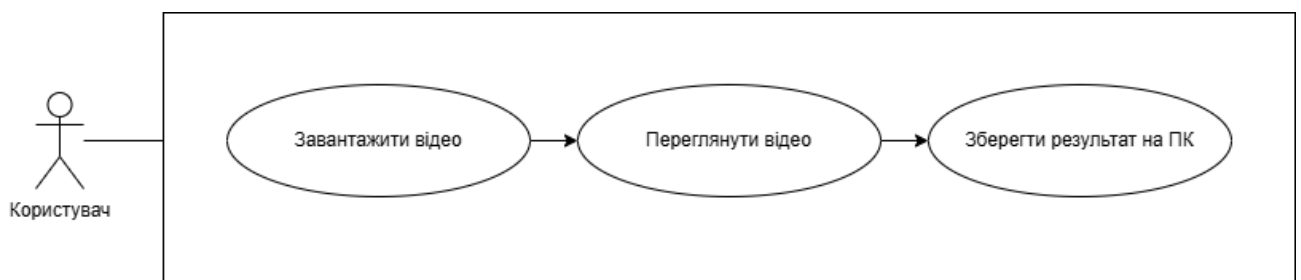


Рисунок 3.3 – Діаграма прецедентів системи

3.4 Проєктування інтерфейсу користувача (UI/UX)

Оскільки система розробляється з урахуванням можливого використання в польових умовах або при дефіциті часу, головною вимогою до інтерфейсу користувача є мінімалізм та зрозумілість. Інтерфейс не повинен бути перевантажений зайвими графічними елементами чи складною навігацією.

Для реалізації візуальної частини обрано підхід односторінкового застосунку, тобто, вся взаємодія відбувається на одному екрані без необхідності переходити за посиланнями.

Основні елементи робочого екрана:

- **зона завантаження файлу:** поле з лівої верхньої сторони екрана, куди користувач може натиснути для вибору файлу через стандартне вікно провідника;

- **індикатор стану:** Обробка відео неймережею займає певний час, тому оператор повинен бачити статус (наприклад, "Завантаження...", "Обробка кадру 45 з 1200...", "Готово"). Це дає розуміння, що система не зависла та працює;

- **візуалізатор результату:** після завершення обробки на місці зони завантаження з'являється стандартний HTML5-плеєр, де можна одразу переглянути відео з накладеними рамками розпізнаних об'єктів;

- **панель управління файлом:** кнопки "Завантажити результат на ПК" та можливість обрати нове відео.

Висновок до розділу 3

У третьому розділі було виконано проектування архітектури веб-застосунку та розроблено алгоритми його роботи. Обґрунтовано використання клієнт-серверної архітектури з розділенням на Frontend, Backend та ML-модуль. Це дозволить системі працювати стабільно та обробляти процеси комп'ютерного зору. Розписано алгоритм покадрової обробки відеопотоку за допомогою неймережі YOLO. За допомогою UML-моделювання візуалізовано основні прецеденти взаємодії користувача з системою. Також сформовано ключові вимоги до інтерфейсу користувача, який має бути мінімалістичним та працювати за принципом єдиного вікна.

Проведене проектування дає чіткий план для переходу до наступного етапу - технічної реалізації.

РОЗДІЛ 4 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

4.1 Етап базового тестування та аналіз проблематики

На початковому етапі дослідження було проведено тестування базових зборок нейронної мережі YOLOv8 (версій Small, Medium та Large), які були попередньо навчені на стандартному загальнопризначеному наборі даних COCO (Common Objects in Context) [3]. Головною метою цього етапу було визначення здатності базових моделей розпізнавати наземні об'єкти на відеоматеріалах з військовою технікою, отриманих з безпілотних літальних апаратів (БПЛА).

Експериментальні запуски виявили ряд недоліків при виконанні завдань.

Нездатність розпізнавати замасковану та військову техніку: Базова модель успішно детектувала стандартні цивільні транспортні засоби (легкові автомобілі, вантажівки), проте виявилася неефективною при знаходженні військової техніки. Наприклад, під час тестування на відеокдрах з близькою дистанцією зйомки, жодна з базових моделей (навіть версія Large) не змогла ідентифікувати бронетранспортер (БТР), оскільки його візуальні ознаки відрізнялися від зразків, на яких навчалася мережа.

Інформаційний шум: Оскільки набір даних COCO містить 80 різних класів об'єктів, модель намагалася класифікувати фонові елементи ландшафту та об'єкти, що не мають жодної цінності для оператора БПЛА. Це призводило до переначислення обробленого відео хибними спрацюваннями.

Отримані результати показали, що використання нейронних мереж із заводськими налаштуваннями є недоцільним для вузькоспеціалізованих завдань аеророзвідки. Аби досягти нормальної точності розпізнавання техніки згори, базову модель довелося додатково тренувати на спеціально підготовлених даних за допомогою трансферного навчання.

4.2 Формування та підготовка спеціалізованого набору даних

Наступним кроком після базового тестування стало застосування методу донавчання моделі на додаткових військових даних. Перша ітерація навчання проводилася на локальному наборі даних, що складався виключно із зображень важкої бронетехніки (танків).

Проте під час тестування оновленої моделі було зафіксовано проблему машинного навчання - явище «катастрофічного забування» [2]. Нейронна мережа досягла гарної точності в ідентифікації танків, але при цьому майже повністю втратила здатність розпізнавати інші класи об'єктів (наприклад, піхоту або легкий транспорт), які вона гарно детектувала до процесу донавчання.

Для усунення цієї проблеми та забезпечення балансу класів було прийнято рішення про створення розширеного набору даних. До фінального датасету увійшли зображення різноманітної військової техніки (бойові машини піхоти, бронетранспортери, артилерійські установки) та військовослужбовці.

Загальний обсяг сформованого набору даних склав понад 7000 зображень. Критично важливою вимогою до відбору фотоматеріалів була наявність специфічних ракурсів зйомки - вигляду зверху та під гострим кутом, що максимально імітують реальні умови та дистанцію з камери безпілотної літального апарата.

Після проведення експериментів із новим набором даних, було зафіксовано суттєве покращення результатів роботи моделі. Якщо раніше виникали труднощі з ідентифікацією військової техніки, то тепер нейромережа почала чітко розрізняти та класифікувати танки, БТР та навіть окремі групи піхоти.

На рисунку 4.1 зображено результат обробки моделі з новим датасетом.



Рисунок 4.1 – Демонстративний приклад роботи моделі

Отже, результат показав, що для задач аеророзвідки неможливо обійтися стандартними моделями. Проблема "катастрофічного забування" змусила повністю переглянути підхід до підготовки даних. Тільки після формування власного датасету, де були враховані реальні ракурси з дронів, вдалося досягти стабільного результату. Це дозволило моделі не просто бачити окремі об'єкти, а ще й навчитися розрізняти під різні класи.

4.3 Програмна реалізація інтерфейсу користувача

Для практичної роботи з моделлю було створено веб-інтерфейс «UAV-VISION Core». Під час його розробки основний акцент робився на простоті, адже в умовах реального моніторингу важливо швидко завантажити дані й отримати результат. Тому застосунок працює як одна сторінка, куди відео

у форматі MP4 можна завантажити для обробки. Одразу після цього запуск нових файлів блокується, щоб уникнути перевантаження серверної частини.

Процес обробки поділено на зрозумілі для користувача етапи, які виводяться на екран. Спочатку індикатор показує етап інференсу (роботу самої нейромережі), а потім - етап FFMPEG, коли відео готується для веб-плеєра [10]. Щоб процес був більш зрозумілим, збоку виводиться інформація з логів: можна в реальному часі спостерігати, скільки кадрів має завантажене відео та яка швидкість обробки за секунду. Фінальний результат завантажуються у звичайний HTML5-плеєр, який дозволяє зупиняти або перемотувати відео для зручного візуального аналізу з можливістю встановити на власний ПК.

При розробці дизайну було обрано темну колірну схему, що відповідає сучасним стандартам програмного забезпечення для систем моніторингу та зменшує навантаження на зір оператора при тривалій роботі. Весь простір організовано так, щоб центральне місце займав відеоплеєр, а решта інформації та результати обробки знаходилися поруч для аналізу у реальному часі. Результат веб-застосунку зображено на рисунку 4.2.

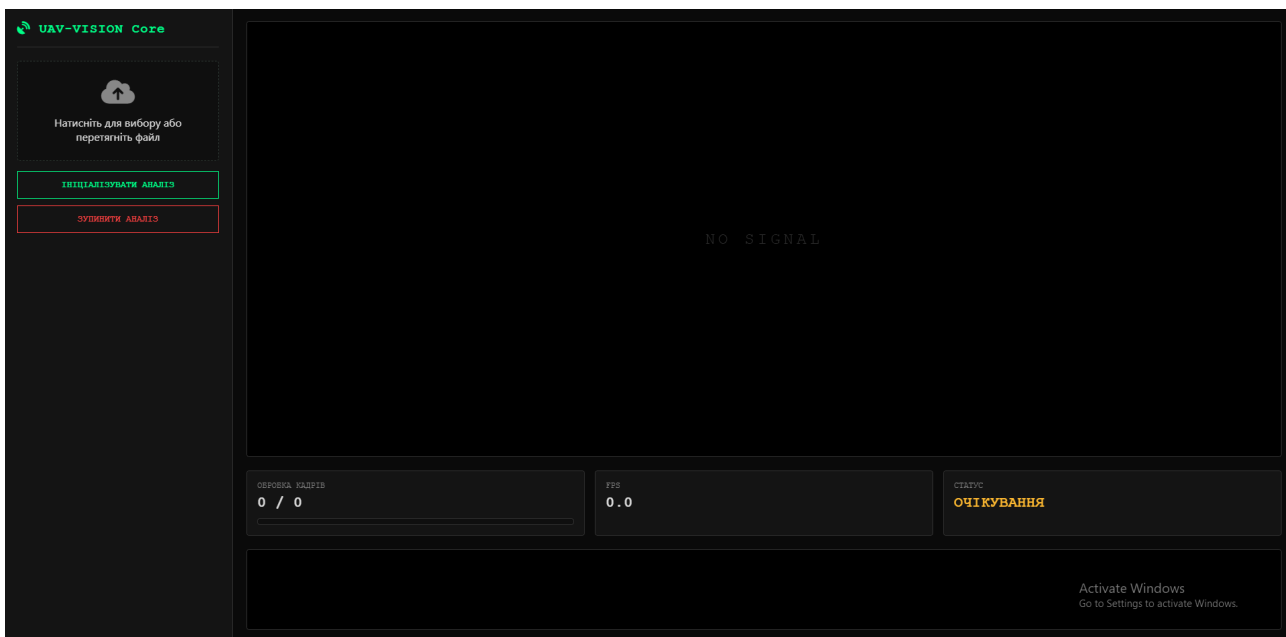


Рисунок 4.2 – Скріншот головного екрана інтерфейсу UAV-VISION Core

4.4 Вибір архітектури нейронної мережі та оптимізація розміру вхідного тензора

Під час реалізації проекту було одне питання: як змусити систему працювати швидко, не втрачаючи при цьому дрібні об'єкти на відео. Оскільки софт для БПЛА має бути ефективним навіть на звичайних комп'ютерах без надпотужних відеокарт, було вирішено зосередитися на пошуку балансу між швидкістю (FPS) та точністю.

Спочатку проводились тестування з "важкою" моделлю YOLOv8-Large. Здавалося, що вона має бути найточнішою, але на практиці результати були незадовільними: швидкість обробки впала майже до нуля, а при стандартному стисненні кадру до 640 пікселів дрібна техніка просто розмивалася.

Щоб вирішити було здійснено заміну моделі на більш легшу - YOLOv8-Small, але натомість збільшено вхідну роздільну здатність до 832 пікселів. Для того, щоб зображення при цьому не розтягувалося і не втрачало пропорції, було застосовано метод Letterboxing (додавання технічних полів по краях) [22]. Це спрацювало набагато краще: нейромережа отримала більше деталей на вхідному етапі, а завдяки легкій архітектурі швидкість залишилася прийнятною для реальних задач. Приклад того, як тепер система бачить об'єкти, наведено на рисунку 4.3.

У результаті такої оптимізації швидкість обробки склала близько 5–6 кадрів на секунду. Оскільки система розрахована на офлайн-аналітику (коли відео завантажується після польоту), цього показника цілком достатньо для комфортної роботи. Звісно, YOLOv8-Small має меншу кількість параметрів порівняно з Large-версією, тому іноді вона може пропустити сильно закамуфльовану техніку або надто дрібні об'єкти. Але це був вимушений крок пожертвувати точністю, щоб програма взагалі могла стабільно працювати на доступному обладнанні, а не зависала на кожному кадрі.



Рисунок 4.3 – Успішна детекція техніки при `imgsz=832`

4.5 Контейнеризація та розгортання програмного комплексу на базі хмарних рішень

Щоб система працювала стабільно на будь-якому обладнанні без налаштувань, на фінальному етапі було впроваджено контейнеризацію через **Docker**. У проєктах із комп'ютерним зором часто виникають конфлікти між версіями бібліотек, тому використання ізольованих контейнерів стало логічним рішенням. Завдяки спеціальному конфігураційному файлу (Dockerfile [8]) середовище збирається автоматично, включаючи в себе не лише Python та базові бібліотеки для роботи нейромереж, а й системну утиліту FFmpeg, без якої неможлива коректна обробка відеофайлів. Такий підхід забезпечує повну ідентичність роботи програми як під час розробки, так і після її перенесення на сторонній сервер.

Для відкритої демонстрації та перевірки інтерфейсу було обрано хмарну платформу Hugging Face Spaces [15]. Це рішення базувалося на практичних

перевагах майданчика: наявності безкоштовного доступу до ресурсів, що дозволило уникнути оренди платних серверів, та нативній підтримці Docker. Це забезпечило швидку синхронізацію коду з репозиторію та не погану роботу веб-сервісу. У підсумку, такий вибір інструментів дозволив перетворити локальну програму на веб-застосунок, доступний для тестування безпосередньо через браузер.

Після розгортання системи на Hugging Face було проведено тестування на її продуктивності, щоб зрозуміти вплив хмарного середовища на швидкість обробки даних. Тестування на ідентичному відео з налаштуваннями моделі YOLOv8-Small (imgsz=832) показало суттєву різницю в швидкодії порівняно з локальним запуском. Якщо на локальній машині вдавалося підтримувати стабільні 5–6 кадрів на секунду, то в хмарі цей показник знизився до 1.5 FPS.

Таке падіння продуктивності пояснюється використанням безкоштовного хостингу, а саме відсутністю доступу до GPU та використанням спільних ресурсів центрального процесора. Хоча така швидкість є недостатньою для аналізу відеопотоку в реальному часі, обрана платформа цілком виконує свою роль як середовище для демонстрації UI/UX інтерфейсу та перевірки загальної логіки роботи алгоритмів. Тому для переходу від прототипу до реального використання наступним кроком має стати розгортання системи на обладнанні з відеокартами.

4.6 Аналіз результатів практичного тестування

Щоб перевірити програму в умовах, наближених до реальних, тестування проходило на кількох відео з різними типами місцевості. Головним завданням було не просто переконатися, що код працює, а знайти слабкі місця алгоритму під час детекції. Для детального розбору обрано три найбільш показові ситуації.

Приклад 1

На першому відео перевірялася робота моделі з піхотою. Солдати були в камуфляжі та просто стояли. Загалом алгоритм впорався добре. Він нормально знаходив людей у центрі кадру і правильно позначав їх рамками.

Але під час перегляду результатів стала помітна одна проблема. Коли люди зміщувалися ближче до країв екрана, рамки іноді просто зникали. Модель переставала їх бачити.

Тут є декілька варіантів. По-перше, військова форма дуже добре зливається з таким фоном. По-друге, по краях будь-якого відео з камери завжди трохи падає різкість. Через цей кут зйомки деталі розмиваються, і неймережі банально не вистачає чіткості, щоб розпізнати силует людини. Те, як це виглядає на практиці, показано на рисунках 4.4 та 4.5.



Рисунок 4.4 – Тестування моделі на солдатах з камуфляжем

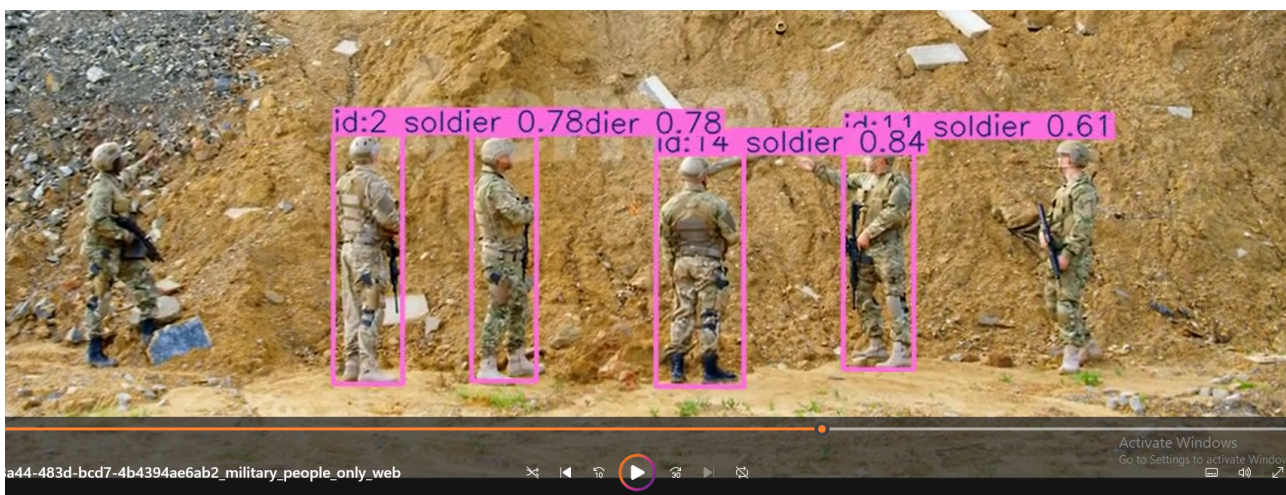


Рисунок 4.5 – Невдалі кадри з тестування

Приклад 2

Наступний приклад з відео, на якому зображено як військову техніку(танки, військові авто, артилерія), так і люди. Аналіз відео показав ще одну важливу деталь. Хоча система чудово впоралася з пошуком техніки, яка стояла щільно одна до одної, вона повністю проігнорувала людей поруч. Насправді це очікувана проблема для легких моделей, коли йдеться про дуже дрібні цілі. З такого ракурсу силует людини займає всього кілька пікселів, і навіть збільшений вхідний кадр не дає мережі достатньо інформації, щоб зачепитися за об'єкт. Цей приклад пояснює те, що легкої архітектури об'єктивно замало, щоб знаходити піхоту з великої висоти. У майбутньому для вирішення цієї проблеми варто спробувати більш складні нейромережі та проводити тести.

Приклад цього дослідження зображений на рисунку 4.6.

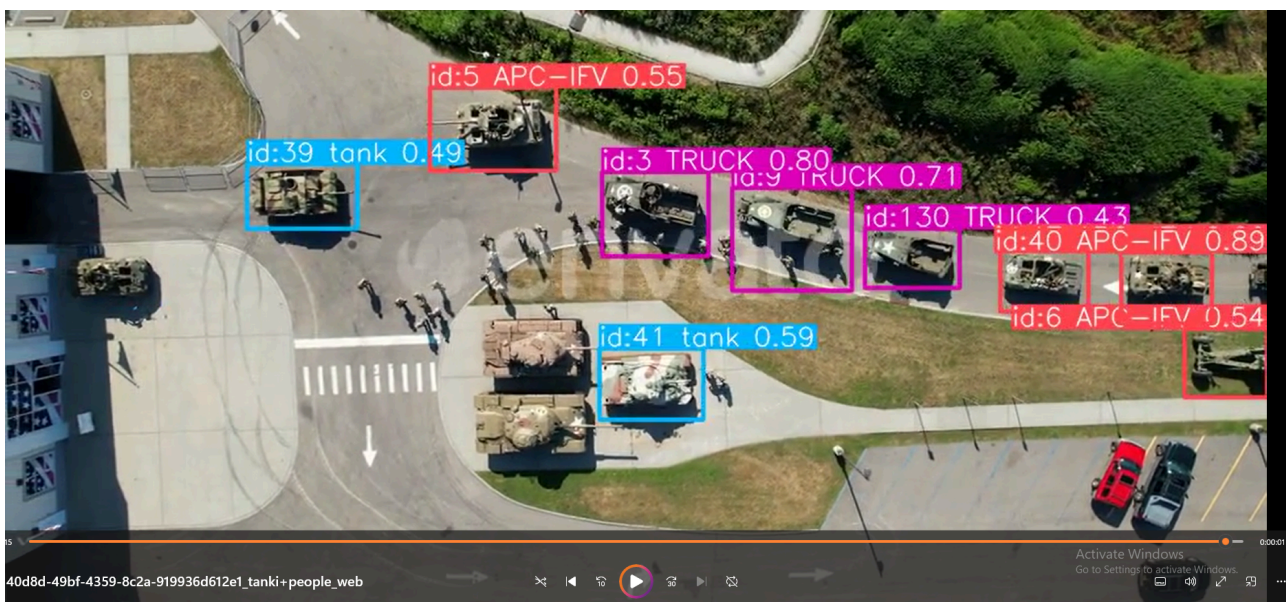


Рисунок 4.6 – Відео щільного скупчення техніки та осіб

Приклад 3

Наступне відео було з танком на ґрунтовій дорозі, де результат був задовільним: модель дуже чітко та якісно обводила бронетехнику під час всього відео. Але так само було помітно недоліки: при зміні кута огляду нейромережа іноді починає плутати класи, змінюючи позначку з танка (tank) на БМП (APC-IFV). З огляду на візуальну схожість цих машин зверху, така поведінка є цілком очікуваною. Інша складність стосується безпосередньо трекінгу: якщо машина хоча б на мить перекривається пилом або деревом, алгоритм губить ціль і згодом присвоює їй новий ID. Також траплялися випадки, коли тіні чи плями бруду на дорозі розпізнавалися як вантажівки (TRUCK). Можливий варіант для вирішення цієї проблеми буде розширення датасету з фоновими зображеннями без техніки, щоб навчити систему ігнорувати подібний шум на дорозі, або використовувати більш потужні моделі та проводити тести. Результати цього прикладу зображений на рисунку 4.7.



Рисунок 4.7 – Результат тестування відео з танком на ґрунтовій дорозі

Висновок до розділу 4

У четвертому розділі підведено підсумки розробки програми «UAV-VISION Core». Головним завданням на старті було адаптувати неймережу до специфіки відео з дронів. Для цього знадобився власний набір даних, який допоміг усунути проблему втрати пам'яті моделі на попередні класи. За основу було взято легку архітектуру YOLOv8-Small. Щоб вона краще розпізнавала деталі з висоти, розмір вхідного кадру розширили до 832 пікселів. Завдяки цьому алгоритм працює досить швидко і добре бачить об'єкти на складному фоні.

Щоб програмою було зручно користуватися, весь код запакували в Docker-контейнер. Це вирішило всі проблеми з конфліктами бібліотек і дозволило запустити веб-інтерфейс на платформі Hugging Face. Тестування в хмарі очікувано показало межі безкоштовних серверів: без нормальної відеокарти швидкість обробки впала до 1.5 кадра на секунду (проти 5-6 кадрів під час локального запуску).

Перевірка на реальних відео підтвердила, що система відмінно знаходить велику техніку навіть у скупченнях. Основні складнощі виникають саме з

піхотою. З великої висоти людина займає на екрані надто мало пікселів, тому легка модель періодично губить такі цілі або плутається через ідеальний камуфляж. Загалом прототип повністю довів свою працездатність. Для його реального використання достатньо перенести систему на потужне обладнання з GPU та додати більше фотографій дрібних цілей у тренувальний датасет.

ВИСНОВКИ

Під час виконання дипломної роботи було вирішено завдання з розробки програмного комплексу для автоматизованого виявлення та класифікації наземних об'єктів на відеопотоках з безпілотних літальних апаратів. У результаті проведеного аналізу сучасних методів комп'ютерного зору було обрано та адаптовано архітектуру YOLOv8-Small, яка забезпечила необхідний баланс між точністю детекції дрібних об'єктів та швидкістю обробки кадрів у реальному часі. Особливу увагу було приділено підготовці спеціалізованого набору даних, що дозволило подолати ефект «катастрофічного забування» та навчити модель розпізнавати нову техніку без втрати раніше засвоєних знань. Програмна реалізація серверної частини на базі фреймворку FastAPI дозволила побудувати асинхронну архітектуру для стабільної передачі даних між нейронною мережею та веб-інтерфейсом. Застосування технологій контейнеризації забезпечило повну автономність програмного продукту від системних залежностей та можливість його успішного розгортання на хмарних платформах для демонстрації результатів через веб-інтерфейс. Підсумком роботи став працездатний прототип інтелектуальної системи моніторингу, який підтвердив свою ефективність у задачах автоматичного аналізу відео та може бути використаний як основа для подальшого впровадження у реальних системах експлуатації БПЛА з метою зниження навантаження на операторів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Amit. (n.d.). *OpenCV: Open Source Computer Vision Library*. Medium. Retrieved June 1, 2026, from <https://medium.com/@amit25173/opencv-short-for-open-source-computer-vision-library-is-an-extensive-open-source-library-that-9e5e6f562c45>
2. Practical DevSecOps. (n.d.). *Catastrophic forgetting*. Retrieved June 1, 2026, from <https://www.practical-devsecops.com/glossary/catastrophic-forgetting/>
3. Ultralytics. (n.d.). *COCO dataset*. Ultralytics Docs. Retrieved June 1, 2026, from <https://docs.ultralytics.com/datasets/detect/coco#coco-pretrained-models>
4. TensorFlow. (n.d.). *Convolutional neural network (CNN)*. TensorFlow Core. Retrieved June 1, 2026, from <https://www.tensorflow.org/tutorials/images/cnn>
5. Mozilla. (n.d.). *CSS: Cascading Style Sheets*. MDN Web Docs. Retrieved June 1, 2026, from <https://developer.mozilla.org/en-US/docs/Web/CSS>
6. Wikipedia contributors. (n.d.). *Deep learning*. In *Wikipedia*. Retrieved June 1, 2026, from https://en.wikipedia.org/wiki/Deep_learning
7. Mozilla. (n.d.). *Django web framework (Python)*. MDN Web Docs. Retrieved June 1, 2026, from https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Django
8. Docker Inc. (n.d.). *Dockerfile reference*. Docker Docs. Retrieved June 1, 2026, from <https://docs.docker.com/reference/dockerfile/>
9. FastAPI. (n.d.). *FastAPI documentation*. Retrieved June 1, 2026, from <https://fastapi.tiangolo.com/>
10. FFmpeg. (n.d.). *FFmpeg documentation*. Retrieved June 1, 2026, from <https://ffmpeg.org/ffmpeg.html#Description>
11. Pallets Projects. (n.d.). *Flask documentation*. Retrieved June 1, 2026, from <https://flask.palletsprojects.com/en/stable/>

12. Docker Inc. (n.d.). *Get started with Docker*. Docker Docs. Retrieved June 1, 2026, from <https://docs.docker.com/get-started/>
13. Esri. (n.d.). *How SSD works*. ArcGIS Developers. Retrieved June 1, 2026, from <https://developers.arcgis.com/python/latest/guide/how-ssd-works/>
14. Mozilla. (n.d.). *HTML: HyperText Markup Language*. MDN Web Docs. Retrieved June 1, 2026, from <https://developer.mozilla.org/en-US/docs/Web/HTML>
15. Hugging Face. (n.d.). *Hugging Face documentation*. Retrieved June 1, 2026, from <https://huggingface.co/docs>
16. Metrics Reloaded. (n.d.). *Intersection over Union*. Retrieved June 1, 2026, from https://metrics-reloaded.dkfz.de/metric-library/intersection_over_union
17. Mozilla. (n.d.). *JavaScript*. MDN Web Docs. Retrieved June 1, 2026, from <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
18. Evidently AI. (n.d.). *Mean Average Precision (mAP) in ranking*. Retrieved June 1, 2026, from <https://www.evidentlyai.com/ranking-metrics/mean-average-precision-map>
19. Wikipedia contributors. (n.d.). *Object detection*. In *Wikipedia*. Retrieved June 1, 2026, from https://en.wikipedia.org/wiki/Object_detection
20. Python Software Foundation. (n.d.). *Python 3 documentation*. Retrieved June 1, 2026, from <https://docs.python.org/3/>
21. PyTorch. (n.d.). *PyTorch documentation*. Retrieved June 1, 2026, from <https://docs.pytorch.org/docs/2.12/index.html>
22. Raktim, R. (n.d.). *Letterboxing in YOLOv5, YOLOv7, YOLOv8: An intuitive explanation with Python code*. Medium. Retrieved June 1, 2026, from <https://medium.com/@reachraktim/letterboxing-in-yolov5-yolov7-yolov8-an-intuitive-explanation-with-python-code-88f7d4323d6c>
23. Robu, R. (n.d.). *Understanding and implementing Faster R-CNN*. Medium. Retrieved June 1, 2026, from <https://medium.com/@RobuRishabh/understanding-and-implementing-faster-r-cnn-248f7b25ff96>

24. LlamaIndex. (n.d.). *What is a bounding box?* LlamaIndex Glossary. Retrieved June 1, 2026, from <https://www.llamaindex.ai/glossary/what-is-a-bounding-box>
25. Encord. (n.d.). *YOLO object detection: The complete guide*. Retrieved June 1, 2026, from <https://encord.com/blog/yolo-object-detection-guide/>

ДОДАТОК А ЛІСТИНГ ПРОГРАМНОГО КОДУ ВЕБ-ЗАСТОСУНКУ ТА КОНФІГУРАЦІЙНИХ ФАЙЛІВ

Файл [main.py](#) (основна логіка API та обробка відео)

```
import asyncio
import os
import shutil
import cv2
import subprocess
import time
import imageio_ffmpeg
from fastapi import FastAPI, File, UploadFile, Request,
BackgroundTasks
from fastapi.responses import HTMLResponse
from fastapi.staticfiles import StaticFiles
from ultralytics import YOLO
from sse_starlette.sse import EventSourceResponse
import uuid
import json

app = FastAPI(title="UAV Object Detection Analytics Dashboard")

# Папки
UPLOAD_FOLDER = "uploads"
STATIC_FOLDER = "static"
RESULT_FOLDER = os.path.join(STATIC_FOLDER, "results")
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
os.makedirs(RESULT_FOLDER, exist_ok=True)

app.mount("/static", StaticFiles(directory=STATIC_FOLDER),
name="static")

processing_tasks = {}
model = YOLO("military_uav_v5.pt")

# Переименовываем классы
target_names = model.names
for key in list(target_names.keys()):
    original_name = target_names[key]
    if 'soldier' in original_name.lower():
        target_names[key] = 'soldier'
```

```

        if 'tank' in original_name.lower():
            target_names[key] = 'tank'

    if hasattr(model, 'model') and hasattr(model.model, 'names'):
        model.model.names = target_names

@app.post("/stop-task/{task_id}")
async def stop_task(task_id: str):
    if task_id in processing_tasks:
        processing_tasks[task_id]["stop_requested"] = True
        return {"status": "stop_requested"}
    return {"status": "nothing_to_stop"}

@app.get("/get-tasks")
async def get_tasks():
    return processing_tasks

@app.get("/")
async def get_index():
    html_content = """
    <!DOCTYPE html>
    <html lang="uk">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,
initial-scale=1.0">
        <title>UAV-VISION | Аналітичний Дашборд</title>
        <style>
            :root {
                --bg-main: #0a0a0a; --bg-panel: #141414; --border:
#333;
                --text-main: #e0e0e0; --text-muted: #888;
                --accent: #00ff88; --accent-dim: rgba(0, 255, 136,
0.2);
                --danger: #ff4444; --warning: #ffbb33;
                --font-mono: 'Courier New', Courier, monospace;
            }
            body { font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
sans-serif; background-color: var(--bg-main); color: var(--text-main);
margin: 0; display: flex; height: 100vh; overflow: hidden; }
            .dashboard { display: flex; width: 100%; height: 100%; }
            .sidebar { width: 300px; background-color:
var(--bg-panel); border-right: 1px solid var(--border); padding: 20px;
display: flex; flex-direction: column; }

```

```

        .logo-area { font-family: var(--font-mono); font-size:
1.2rem; font-weight: bold; color: var(--accent); border-bottom: 1px solid
var(--border); padding-bottom: 15px; margin-bottom: 20px; letter-spacing:
1px; }

        .upload-zone { border: 1px dashed var(--border);
background: #0f0f0f; padding: 25px 15px; text-align: center; border-radius:
4px; margin-bottom: 15px; cursor: pointer; transition: all 0.2s ease; }

        .upload-zone:hover { border-color: var(--accent);
background-color: #141414; }

        .upload-zone i { font-size: 2.5rem; color:
var(--text-muted); margin-bottom: 15px; }

        #file-input { display: none; }

        .btn-submit { background-color: transparent; border: 1px
solid var(--accent); color: var(--accent); width: 100%; padding: 12px;
font-family: var(--font-mono); font-weight: bold; cursor: pointer;
text-transform: uppercase; transition: all 0.2s; }

        .btn-submit:hover { background-color: var(--accent-dim); }

        .btn-submit:disabled { border-color: var(--text-muted);
color: var(--text-muted); cursor: not-allowed; }

        .btn-stop { background-color: transparent; border: 1px
solid var(--danger); color: var(--danger); width: 100%; padding: 12px;
font-family: var(--font-mono); font-weight: bold; cursor: pointer;
text-transform: uppercase; margin-top: 10px; transition: all 0.2s; }

        .btn-stop:hover { background-color: rgba(255, 68, 68,
0.1); }

        .main-panel { flex-grow: 1; display: flex; flex-direction:
column; padding: 20px; background: repeating-linear-gradient(0deg,
transparent, transparent 1px, rgba(255,255,255,0.01) 1px,
rgba(255,255,255,0.01) 2px); }

        .video-container { flex-grow: 1; background-color: #000;
border: 1px solid var(--border); border-radius: 4px; display: flex;
align-items: center; justify-content: center; position: relative; overflow:
hidden; margin-bottom: 20px; }

        #result-video { width: 100%; height: 100%; object-fit:
contain; display: none; }

        .placeholder-text { font-family: var(--font-mono); color:
var(--border); font-size: 1.5rem; text-transform: uppercase; letter-spacing:
5px; }

        .telemetry { display: flex; gap: 15px; font-family:
var(--font-mono); margin-bottom: 20px; }

        .metric-box { flex: 1; background: var(--bg-panel);
border: 1px solid var(--border); padding: 15px; border-radius: 4px; }

        .metric-title { font-size: 0.75rem; color:
var(--text-muted); text-transform: uppercase; margin-bottom: 5px; }

```

```

        .metric-value { font-size: 1.4rem; color:
var(--text-main); font-weight: bold; }
        .progress-wrapper { background: #0f0f0f; border: 1px solid
var(--border); height: 8px; border-radius: 4px; overflow: hidden;
margin-top: 10px; }
        .progress-fill { height: 100%; width: 0%;
background-color: var(--accent); transition: width 0.2s; }
        .log-container { background: #000; border: 1px solid
var(--border); padding: 15px; font-family: var(--font-mono); font-size:
0.8rem; color: var(--text-muted); height: 120px; overflow-y: auto;
border-radius: 4px; width: 100%; box-sizing: border-box; }
        .log-entry { margin-bottom: 6px; }
        .log-success { color: var(--accent); }
        .log-error { color: var(--danger); }
        .log-warning { color: var(--warning); }
    </style>

    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.
css">

</head>
<body>
    <div class="dashboard">
        <aside class="sidebar">
            <div class="logo-area"><i class="fas
fa-satellite-dish"></i> UAV-VISION Core</div>
            <div id="upload-section">
                <div class="upload-zone" id="drop-zone">
                    <i class="fas fa-cloud-upload-alt"></i>
                    <div id="file-name-display">Натисніть для
вибору або перетягніть файл</div>
                    <input type="file" id="file-input"
accept="video/mp4, video/avi">
                </div>
                <button type="button" class="btn-submit"
id="submit-btn">Ініціалізувати аналіз</button>
                <button type="button" class="btn-stop"
id="stop-btn">ЗУПИНИТИ АНАЛІЗ</button>
            </div>
        </aside>
        <main class="main-panel">
            <div class="video-container">
                <div class="placeholder-text"
id="video-placeholder">NO SIGNAL</div>
                <video id="result-video" controls muted></video>
            </div>

```

```

        <div class="telemetry">
            <div class="metric-box">
                <div class="metric-title">Обробка кадрів</div>
                <div class="metric-value" id="stat-frames">0
<span>/ 0</span></div>
                                <div class="progress-wrapper"><div
class="progress-fill" id="progress-bar"></div></div>
                                </div>
                                <div class="metric-box">
                                    <div class="metric-title">FPS</div>
                                    <div class="metric-value"
id="stat-fps">0.0</div>
                                </div>
                                <div class="metric-box">
                                    <div class="metric-title">Статус</div>
                                    <div class="metric-value" id="stat-status"
style="color: var(--warning);">ОЧІКУВАННЯ</div>
                                    </div>
                                </div>
                                <div id="log-container" class="log-container"></div>
        </main>
    </div>

    <script>
        let currentTaskId = null;
        let eventSource = null;

        const fileInput = document.getElementById('file-input');
        const submitBtn = document.getElementById('submit-btn');
        const stopBtn = document.getElementById('stop-btn');
        const dropZone = document.getElementById('drop-zone');
        const fileNameDisplay =
document.getElementById('file-name-display');
        const progressBar =
document.getElementById('progress-bar');
        const statStatus = document.getElementById('stat-status');
        const videoPlaceholder =
document.getElementById('video-placeholder');
        const videoPlayer =
document.getElementById('result-video');

        function addLog(msg, type = 'info') {
            const container =
document.getElementById('log-container');
            const entry = document.createElement('div');

```

```

        entry.className = `log-entry log-${type}`;
        entry.textContent = `[${new
Date().toLocaleTimeString()}] > ${msg}`;
        container.appendChild(entry);
        container.scrollTop = container.scrollHeight;
    }

    // Логіка вибору файлу (блокуємо, якщо вже йде обробка)
    dropZone.addEventListener('click', function() {
        if (currentTaskId) {
            addLog('Обробка вже триває. Спочатку зупиніть
процес або дочекайтеся завершення.', 'warning');
            return;
        }
        fileInput.click();
    });

    fileInput.addEventListener('change', function() {
        if (this.files[0]) {
            const name = this.files[0].name;
            addLog(`Файл готовий: ${name}`, 'success');
            fileNameDisplay.innerHTML = `Вибрано: <br><span
style="color: var(--accent); word-break: break-all;">${name}</span>`;
        }
    });

    // Drag & Drop
    ['dragenter', 'dragover', 'dragleave', 'drop'].forEach(e
=> dropZone.addEventListener(e, (ev) => { ev.preventDefault();
ev.stopPropagation(); }));
    dropZone.addEventListener('drop', (e) => {
        if (currentTaskId) return addLog('Обробка вже
триває!', 'warning');
        fileInput.files = e.dataTransfer.files;
        const event = new Event('change');
        fileInput.dispatchEvent(event);
    });

    function connectToProgress(taskId) {
        currentTaskId = taskId;
        if (eventSource) eventSource.close();
        eventSource = new
EventSource(`/stream-progress/${taskId}`);

        submitBtn.disabled = true;

```

```

submitBtn.textContent = 'ОБРОБКА...';
videoPlaceholder.textContent = 'PROCESSING...';

eventSource.onmessage = function(event) {
    const taskData = JSON.parse(event.data);
    if (taskData.status === 'processing') {
        statStatus.textContent = 'ІНФЕРЕНС';
        progressBar.style.width = taskData.progress +
'%';

document.getElementById('stat-frames').innerHTML =
`${taskData.frames_processed} <span>/ ${taskData.total_frames}</span>`;

document.getElementById('stat-fps').textContent = taskData.fps.toFixed(1);
    } else if (taskData.status === 'converting') {
        statStatus.textContent = 'FFMPEG';
    } else if (taskData.status === 'completed') {
        finishTask(taskData.video_url);
    } else if (taskData.status === 'stopped') {
        stopTaskUI('Обробку зупинено');
    } else if (taskData.status === 'error') {
        stopTaskUI('Помилка: ' + taskData.message,
'error');
    }
};

function finishTask(videoUrl) {
    if(eventSource) eventSource.close();
    statStatus.textContent = 'ГОТОВО';
    statStatus.style.color = 'var(--accent)';
    progressBar.style.width = '100%';
    videoPlayer.src = videoUrl + "?t=" + new
Date().getTime();

    videoPlaceholder.style.display = 'none';
    videoPlayer.style.display = 'block';
    addLog('Аналіз завершено успішно!', 'success');
    resetUI();
}

function stopTaskUI(msg, type = 'error') {
    if(eventSource) eventSource.close();
    statStatus.textContent = 'ЗУПИНЕНО';
    statStatus.style.color = 'var(--danger)';
    addLog(msg, type);
}

```

```

        resetUI();
    }

    function resetUI() {
        submitBtn.disabled = false;
        submitBtn.textContent = 'ІНІЦІАЛІЗУВАТИ АНАЛІЗ';
        currentTaskId = null;
        // Не очищуємо назву файлу, щоб користувач бачив, що
        він обробляв
    }

    async function restoreSession() {
        try {
            const res = await fetch('/get-tasks');
            const tasks = await res.json();
            for (const [id, data] of Object.entries(tasks)) {
                if (['accepted', 'processing',
'converting'].includes(data.status)) {
                    addLog('Відновлення сесії активної
задачі...', 'warning');

                    connectToProgress(id);
                    break;
                }
            }
        } catch (e) {}
    }

    window.onload = restoreSession;

    submitBtn.addEventListener('click', async function() {
        if (!fileInput.files[0]) return alert('Оберіть
файл!');

        const formData = new FormData();
        formData.append('file', fileInput.files[0]);
        addLog('Надсилання файлу на сервер...', 'info');

        try {
            const res = await fetch('/upload-video/', {
method: 'POST', body: formData });
            const data = await res.json();
            connectToProgress(data.task_id);
        } catch (e) { addLog('Помилка завантаження', 'error');
        }

    });

```

```

        stopBtn.addEventListener('click', async function() {
            if (!currentTaskId) return addLog('Немає активних
процесів', 'warning');
            await fetch(`/stop-task/${currentTaskId}`, { method:
'POST' });
            addLog('Запит на зупинку надіслано...', 'warning');
        });
    </script>
</body>
</html>
"""
    return HTMLResponse(content=html_content)

@app.post("/upload-video/")
async def upload_video(background_tasks: BackgroundTasks, file:
UploadFile = File(...)):
    task_id = str(uuid.uuid4())
    safe_filename = f"{task_id}_{file.filename.replace(' ', '_')}"
    file_path = os.path.join(UPLOAD_FOLDER, safe_filename)
    with open(file_path, "wb") as buffer:
        shutil.copyfileobj(file.file, buffer)

    processing_tasks[task_id] = {"status": "accepted", "progress": 0,
"frames_processed": 0, "total_frames": 0, "fps": 0.0, "stop_requested":
False}

    background_tasks.add_task(process_video_yolo, task_id, file_path)
    return {"task_id": task_id}

def process_video_yolo(task_id: str, source_path: str):
    try:
        cap = cv2.VideoCapture(source_path)
        total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
        cap.release()

        processing_tasks[task_id]["total_frames"] = total_frames

        # Інференс
        results_generator = model.track(
            source=source_path, save=True,
project=os.path.abspath(STATIC_FOLDER),
            name="results", exist_ok=True, imgsz=832, conf=0.40,
tracker="bytetrack.yaml", stream=True
        )

        start_time = time.time()

```

```

for i, _ in enumerate(results_generator):
    if processing_tasks[task_id].get("stop_requested"):
        processing_tasks[task_id]["status"] = "stopped"
        break

    frames = i + 1
    if frames % 5 == 0 or frames == total_frames:
        processing_tasks[task_id].update({
            "progress": int((frames / total_frames) * 100),
            "frames_processed": frames,
            "fps": frames / (time.time() - start_time),
            "status": "processing"
        })

    if processing_tasks[task_id]["status"] == "stopped":
        if os.path.exists(source_path): os.remove(source_path)
        return

    processing_tasks[task_id]["status"] = "converting"
    base_name = os.path.splitext(os.path.basename(source_path))[0]
    res_dir = os.path.join(os.path.abspath(STATIC_FOLDER),
"results")

    yolo_file = next(f for f in os.listdir(res_dir) if
f.startswith(base_name) and not f.endswith("_web.mp4"))

    yolo_path = os.path.join(res_dir, yolo_file)
    web_path = os.path.join(res_dir, f"{base_name}_web.mp4")

    subprocess.run([imageio_ffmpeg.get_ffmpeg_exe(), "-i",
yolo_path, "-vcodec", "libx264", "-y", web_path], check=True,
capture_output=True)

    os.remove(yolo_path)
    if os.path.exists(source_path): os.remove(source_path)
    processing_tasks[task_id].update({"video_url":
f"/static/results/{base_name}_web.mp4", "status": "completed", "progress":
100})

except Exception as e:
    processing_tasks[task_id].update({"status": "error",
"message": str(e)})

@app.get("/stream-progress/{task_id}")
async def stream_progress(task_id: str, request: Request):
    async def event_generator():
        while True:

```

```

        if await request.is_disconnected(): break
        task = processing_tasks.get(task_id, {"status": "error"})
        yield {"event": "message", "data": json.dumps(task)}
        if task.get("status") in ["completed", "error",
"stopped"]: break
        await asyncio.sleep(0.5)
    return EventSourceResponse(event_generator())

```

Файл Dockerfile (конфігурація та розгортання відео)

```

FROM python:3.10-slim

RUN apt-get update && apt-get install -y \
    ffmpeg \
    && rm -rf /var/lib/apt/lists/*

RUN useradd -m -u 1000 user
USER user
ENV PATH="/home/user/.local/bin:$PATH"

WORKDIR /app

COPY --chown=user requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY --chown=user . .

RUN mkdir -p uploads static/results

EXPOSE 7860

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "7860"]

```

Файл requirements.txt (залежності проекту)

```

fastapi
uvicorn
python-multipart
ultralytics
opencv-python-headless
imageio[ffmpeg]
sse-starlette
lap

```