ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Система розпізнавання дронів у реальному часі на основі нейронних мереж та відеопотоку»

на здобуття освітнього ступеня магістр
за спеціальності 126 Інформаційні системи та технології
(код,найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Віталій ЧЕРНИШ
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти
група ІСДМ-61 Віталій ЧЕРНИШ
(ім'я, ПРІЗВИЩЕ)
Керівник Ірина СРІБНА
д.т.н. доцент *(ім'я, ПРІЗВИЩЕ)*
Рецензент: _____
(ім'я, ПРІЗВИЩЕ)

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти магістр
Спеціальність 126 Інформаційні системи та технології
Освітньо-професійна прорама Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ
_____Каміла СТОРЧАК

"____" _____2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Чернишу Віталію Володимировичу
(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: Система розпізнавання дронів у реальному часі на основі нейронних мереж та відеопотоку
керівник кваліфікаційної роботи: Ірина СРІБНА д.т.н., доцент
(ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від "30" жовтня 2025 р. № 467

2. Строк подання кваліфікаційної роботи «26» грудня 2025 р.

3. Вихідні дані кваліфікаційної роботи:
1. Технології комп'ютерного зору та обробки відеопотоку в реальному часі.
2. Архітектура нейронних мереж для детекції та класифікації об'єктів.
3. Методи навчання нейронних мереж для розпізнавання дронів.
4. Науково-технічна література та існуючі рішення.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
1. Дослідження сучасних методів детекції об'єктів у відеопотоці та аналіз проблематики розпізнавання дронів.
2. Огляд та порівняльний аналіз архітектур нейронних мереж для детекції дронів.
3. Розробка, навчання та тестування моделі розпізнавання дронів у реальному часі.

5.Перелік ілюстраційного матеріалу: *презентація*

6. Дата видачі завдання «___» _____ 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Підбір технічної літератури	20.10-25.11.25	
2.	Аналіз існуючих рішень у сфері комп'ютерного зору	26.11-01.11.25	
3.	Дослідження сучасних методів детекції	02.11-10.11.25	
4.	Розробка, навчання та тестування моделі	11.11-20.11.25	
5.	Висновки по роботі	21.11-25.11.25	
6.	Розробка демонстраційних матеріалів, доповідь.	26.11-27.11.25	
7.	Оформлення магістерської роботи	28.11-29.11.25	

Здобувач вищої освіти_____ Віталій ЧЕРНИШ
(підпис) *(ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи_____ Ірина СРІБНА
(підпис) *(ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття ступня магістр: 82 стор., 34 рис., 8 табл., 25 джерел.

Мета роботи – розробка та дослідження системи автоматичного розпізнавання безпілотних літальних апаратів у реальному часі на основі глибоких нейронних мереж та аналізу відеопотоку.

Об'єкт дослідження – процес детекції та ідентифікації дронів у відеопотоці в реальному часі.

Предмет дослідження – методи та алгоритми комп'ютерного зору на базі згорткових нейронних мереж для розпізнавання малорозмірних рухомих об'єктів.

Короткий зміст роботи. Перший розділ містить аналіз проблеми виявлення БПЛА, включаючи специфічні виклики (варіативність форм, висока швидкість, складні фонові та метеоумови) та огляд існуючих комерційних і дослідницьких рішень протидронової оборони.

Другий розділ присвячено огляду сучасних архітектур нейронних мереж (YOLO, SSD, Faster R-CNN, EfficientDet) та їх порівняльному аналізу за критеріями точності, швидкості й ресурсних вимог. Обґрунтовано вибір архітектури для системи розпізнавання в реальному часі.

Третій розділ описує методологію розробки системи, формування датасету та навчання мережі. Представлено архітектуру програмного рішення для обробки відеопотоку та результати тестування на різних сценаріях: метрики детекції (precision, recall, mAP), швидкість обробки (FPS), хибні спрацювання. Виконано порівняння з аналогами.

КЛЮЧОВІ СЛОВА: РОЗПІЗНАВАННЯ ДРОНІВ, КОМП'ЮТЕРНИЙ ЗІР, НЕЙРОННІ МЕРЕЖІ, YOLO, ДЕТЕКЦІЯ ОБ'ЄКТІВ, ВІДЕОПОТІК, РЕАЛЬНИЙ ЧАС, ГЛИБОКЕ НАВЧАННЯ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, БЕЗПІЛОТНІ ЛІТАЛЬНІ АПАРАТИ, СИСТЕМИ БЕЗПЕКИ.

ABSTRACT

The text part of the qualifying work for obtaining a bachelor's degree: 82 pp., 34 fig., 8 tables, 25 sources.

The purpose of the work is to develop and study a system for automatic recognition of unmanned aerial vehicles in real time based on deep neural networks and video stream analysis.

The object of study is the process of detecting and identifying drones in a video stream in real time.

The subject of the study is methods and algorithms of computer vision based on convolutional neural networks for recognizing small moving objects.

Summary of the work. The first section contains an analysis of the problem of UAV detection, including specific challenges (variability of shapes, high speed, complex background and weather conditions) and an overview of existing commercial and research solutions for anti-drone defense.

The second section is devoted to an overview of modern neural network architectures (YOLO, SSD, Faster R-CNN, EfficientDet) and their comparative analysis in terms of accuracy, speed, and resource requirements. The choice of architecture for the real-time recognition system is justified.

The third section describes the methodology for developing the system, forming the dataset, and training the network. The architecture of the software solution for video stream processing and the results of testing in various scenarios are presented: detection metrics (precision, recall, mAP), processing speed (FPS), false positives. A comparison with analogues is performed.

KEYWORDS: DRONE RECOGNITION, COMPUTER VISION, NEURAL NETWORKS, YOLO, OBJECT DETECTION, VIDEO STREAM, REAL TIME, DEEP LEARNING, CONVOLUTIONAL NEURAL NETWORKS, UNMANNED AERIAL VEHICLES, SECURITY SYSTEMS.

ЗМІСТ

ВСТУП

Актуальність теми. Стрімке поширення дронів у всіх сферах від комерції до військової справи кидає серйозний виклик безпеці аеропортів, важливих об'єктів та звичайних міст. Але в умовах повномасштабної війни в Україні ця проблема набула зовсім іншого масштабу. Загроза зросла в рази: ворог постійно використовує безпілотники, щоб проводити розвідку або завдавати ударів. Ми щодня бачимо атаки на енергосистему, житлові будинки та стратегічні цілі, тому потреба в надійних системах, які здатні вчасно помітити небезпеку, зараз є просто критичною.

Головна складність у тому, що сучасні БПЛА малі, дуже маневрені й можуть літати низько над землею, через що звичайні радари їх часто «пропускають». На фронті це пряма загроза життю бійців, а в тилу - небезпека для цивільних. Якщо такий апарат прорветься до захищеного об'єкта, наслідки можуть бути катастрофічними: від знищеної інфраструктури до масових жертв. Статистика говорить сама за себе: понад 70% сучасних бойових втрат пов'язані саме з безпілотними системами, що чітко показує реальний масштаб біди.

Класичні методи виявлення, такі як радари, акустика чи радіосканери мають свої недоліки. Вони часто надто дорогі, складні в налаштуванні або просто неефективні посеред щільної міської забудови. В умовах реальних бойових дій нам потрібні доступні, швидкі й надійні рішення, які можна масово впроваджувати як на «нулі», так і в мирних містах. І тут відкриваються великі можливості завдяки комп'ютерному зору та глибокому навчанню. Ці технології дозволяють точно розпізнавати дрони через відеопотік у реальному часі. Тож розробка систем автоматичного пошуку БПЛА на базі нейромереж - це не просто суха технічна задача, а життєва необхідність, від якої напряму залежить наша національна безпека і збереження людських життів.

Мета роботи – розробка та дослідження системи автоматичного розпізнавання безпілотних літальних апаратів у реальному часі на основі глибоких нейронних мереж та аналізу відеопотоку.

Для досягнення мети, у магістерській роботі успішно виконано наступні

завдання:

- Дослідження сучасних методів детекції об'єктів у відеопотоці та аналіз проблематики розпізнавання дронів;- Огляд методів виявлення аномалій в IoT;

- Огляд та порівняльний аналіз архітектур нейронних мереж для детекції дронів;

Об'єкт дослідження – процес детекції та ідентифікації дронів у відеопотоці в реальному часі.

Предмет дослідження – методи та алгоритми комп'ютерного зору на базі згорткових нейронних мереж для розпізнавання малорозмірних рухомих об'єктів.

Методи дослідження. Під час написання магістерської кваліфікаційної роботи були використані методи теоретичного дослідження, аналізу існуючих архітектур нейронних мереж, методи глибокого навчання (transfer learning, data augmentation), методи обробки та аналізу відеопотоку, а також експериментальні методи тестування та оцінки ефективності розробленої системи.

Наукова новизна одержаних результатів. У ході дослідження розроблено та оптимізовано систему розпізнавання дронів, що поєднує сучасні архітектури згорткових нейронних мереж з ефективними методами обробки відеопотоку. Запропоновано підхід до формування спеціалізованого датасету та методику навчання моделі, що забезпечує високу точність детекції при збереженні швидкодії, необхідної для роботи в реальному часі. Результати експериментів демонструють можливість ефективного виявлення дронів різних типів у складних погодних умовах та при різних рівнях освітлення.

Практична значущість одержаних результатів. Розроблена система забезпечує ефективне рішення для автоматичного виявлення безпілотних літальних апаратів і може бути інтегрована у системи безпеки аеропортів, критичної інфраструктури, військових об'єктів та цивільних населених пунктів. В умовах воєнного стану система може використовуватись для раннього попередження про загрозу атаки дронів-камікадзе, що дає цінний час для евакуації населення або активації засобів протидії. Система реалізована з використанням доступних апаратних засобів і може працювати як на серверному

обладнанні, так і на граничних пристроях (edge devices), що дозволяє масово впроваджувати її на передовій та у цивільних локаціях.

Апробація результатів магістерської роботи. Основні положення і результати магістерської роботи доповідались на науково практичних конференціях, що проходили на базі Державного університету інформаційно-комунікаційних технологій.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ СУЧАСНИХ МЕТОДІВ ДЕТЕКЦІЇ ОБ'ЄКТІВ У ВІДЕОПОТОЦІ ТА АНАЛІЗ ПРОБЛЕМАТИКИ РОЗПІЗНАВАННЯ ДРОНІВ

1.1 Огляд сучасних загроз від безпілотних літальних апаратів та методів їх виявлення

Дрони вже давно перестали бути просто технологічною цікавинкою. Сьогодні це масовий і потужний інструмент, який проник майже в усі сфери нашого життя. Звісно, користі від них багато: доставка, аерозйомка, і моніторинг полів. Але є й інша сторона медалі - чим більше дронів, тим більше ризиків, що їх використають не за призначенням або навіть зі злими намірами.

Якщо розібратися із загрозами, то все залежить від того, навіщо запустили "пташку". Наприклад, розвідувальні дрони шпигують за режимними об’єктами, військовими позиціями чи інфраструктурою. Ударні безпілотники - це вже серйозніше: вони несуть вибухівку і використовуються для терактів чи диверсій.

За конструкцією цей "зоопарк" теж доволі різноманітний. Найпоширеніші - мультикоптери (квадро-, гекса- тощо). Типові типи безпілотних літальних апаратів за конструкцією наведено на рис. 1.1. Вони дуже маневрені, вміють зависати на місці, але літають недалеко й заряд тримають недовго. Інша справа - безпілотники літакового типу: вони швидші, можуть долати великі відстані, хоча й потребують простору для розгону. А зараз з'являються ще й гібриди, які намагаються поєднати найкраще з обох світів: злітають вертикально, як коптер, а летять далеко, як літак.

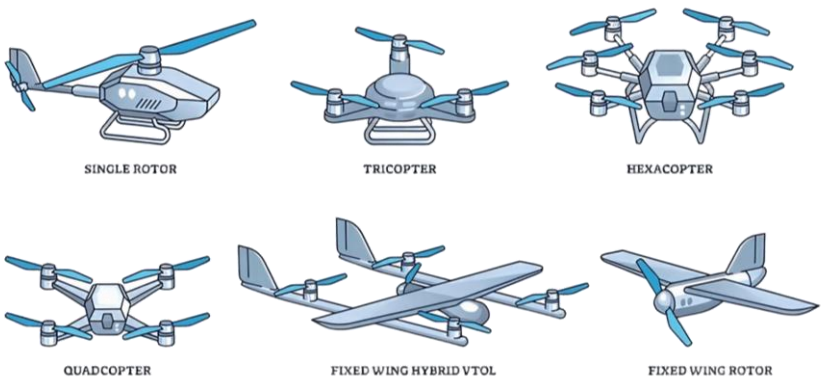


Рис. 1.1 Типи безпілотних літальних апаратів

Особливої уваги заслуговує військове застосування безпілотників. Конфлікт в Україні наочно продемонстрував трансформацію ролі БПЛА у сучасній війні. Дрони використовуються для розвідки, коригування вогню артилерії, прямих ударів по живій силі та техніці противника, атак на об'єкти цивільної інфраструктури. За оцінками військових експертів, до 70% успішних ударів по ворожих позиціях здійснюється з використанням дронів-коригувальників або ударних БПЛА.

Тактика застосування дронів постійно еволюціонує. Якщо на початку конфлікту переважали розвідувальні місії, то згодом широко розповсюдились FPV-дрони (дрони з видом від першої особи) для точкових ударів, дрони-камікадзе для атак на віддалені цілі, та рої дронів для одночасного ураження декількох об'єктів. Ця еволюція створює постійно зростаючий виклик для систем протиповітряної оборони.

Існуючі методи виявлення БПЛА можна класифікувати за фізичним принципом детекції.

Радіолокаційні системи базуються на відбитті електромагнітних хвиль від об'єкта. Принцип роботи радіолокаційної системи виявлення БПЛА схематично показано на рис. 1.2. Традиційні радари ефективні для виявлення великих літальних апаратів, проте мають суттєві обмеження при детекції малорозмірних дронів. Ефективна площа розсіювання типового квадрокоптера становить лише 0.001-0.01 м², що на два порядки менше, ніж у легкомоторного літака. Це призводить до малої дальності виявлення (зазвичай не більше 3-5 км) та високої ймовірності пропуску цілі.

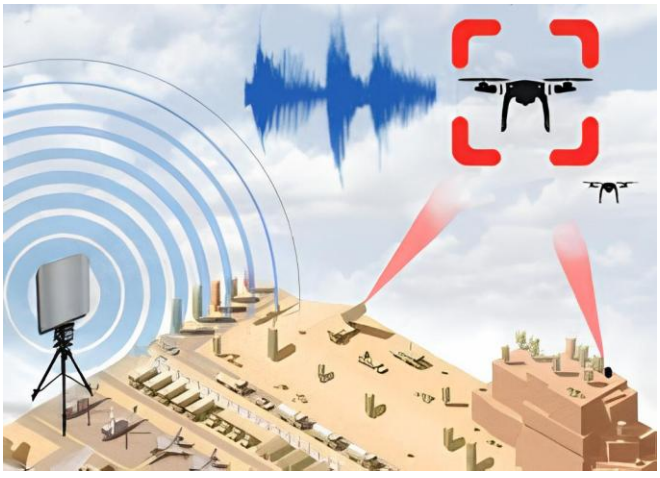


Рис. 1.2 Принцип роботи радіолокаційної системи виявлення БПЛА

Сучасні радари з фазованою антенною решіткою (ФАР) демонструють кращі характеристики, проте їх вартість залишається надзвичайно високою - від 500 тисяч до декількох мільйонів доларів за систему, що робить неможливим масове розгортання таких систем.

Акустичні системи виявляють дрони за характерним звуком, що генерується пропелерами. Алгоритми машинного навчання аналізують акустичний спектр та виділяють сигнатури, характерні для БПЛА. Основні переваги: низька вартість обладнання, пасивний режим роботи, висока ефективність на близьких дистанціях (до 500 м).

Недоліки методу суттєві: обмежена дальність виявлення, висока залежність від фонового шуму (метод практично непрацездатний в умовах міста або поблизу автомагістралей), неможливість роботи при сильному вітрі, складність визначення точних координат об'єкта.

В таблиці 1.1 представлена характеристика методів виявлення БПЛА.

Радіочастотні системи спеціалізуються на виявленні та аналізі радіосигналів, що використовуються для управління дроном. Більшість комерційних БПЛА працюють в частотних діапазонах 2.4 ГГц та 5.8 ГГц. Спеціалізовані приймачі перехоплюють ці сигнали та ідентифікують наявність дрона в зоні покриття.

Переваги методу: відносно низька вартість, можливість визначення типу дрона за сигнатурою сигналу, пасивний режим роботи. Критичний недолік полягає в неефективності проти автономних дронів, що працюють без постійного радіозв'язку з оператором, або дронів з нестандартними частотами управління.

Оптичні та інфрачервоні системи використовують камери видимого спектру або теплові імагери для виявлення дронів. Системи комп'ютерного зору аналізують відеопотік та ідентифікують об'єкти, що відповідають характеристикам БПЛА. ІЧ-камери виявляють теплове випромінювання від двигунів та електроніки дрона.

Таблиця 1.1

Порівняльна характеристика методів виявлення БПЛА

Метод виявлення	Принцип дії	Переваги	Недоліки та обмеження
Радіолокаційний (Radar)	Активне випромінювання електромагнітних хвиль та аналіз відбитого сигналу. Використання ефекту Доплера для селекції рухомих цілей	Велика дальність виявлення (до 30 км), всепогодність, точне визначення дальності та швидкості	Висока вартість, складність виявлення малорозмірних та пластикових цілей (низька ЕПР), вразливість до протирадіолокаційних ракет, "сліпі зони" на малих висотах
Радіочастотний (RF Detection)	Пасивне сканування ефіру на наявність сигналів керування (uplink) та передачі відео (downlink). Аналіз протоколів зв'язку	Пасивність (скритність роботи), можливість локалізації оператора, відносно низька вартість	Неефективність проти автономних дронів (режим радіомовчання), складність в умовах зашумленого міського ефіру, можливість зміни частот (frequency hopping)
Акустичний (Acoustic)	Використання масивів мікрофонів для пеленгації звуку моторів та пропелерів. Порівняння з базою сигнатур	Пасивність, можливість роботи поза прямою видимістю (NLOS), низька вартість розгортання	Мала дальність (до 500), висока залежність від вітру та фонового шуму (місто, техніка), низька точність визначення координат
Оптичний (Electro-Optical/IR)	Аналіз відеопотоку з камер видимого та інфрачервоного спектру за допомогою алгоритмів комп'ютерного зору	Візуальна ідентифікація (відрізнення птаха від дрона), висока точність кутових координат, нечутливість до РЕБ	Залежність від освітлення та погоди (туман, дощ), обмежене поле зору (FOV), високі вимоги до обчислювальних ресурсів

Основні переваги оптичних систем: висока точність локалізації виявленого об'єкта, можливість візуальної ідентифікації типу загрози, відносно низька вартість обладнання, незалежність від способу управління дроном, можливість роботи в пасивному режимі.

Обмеження методу пов'язані з залежністю від погодних умов (туман, дощ, сніг суттєво знижують ефективність), часу доби (камери видимого спектру малоефективні вночі), необхідності прямої видимості об'єкта.

Найефективніший підхід до виявлення БПЛА полягає в комбінуванні декількох методів детекції. Гібридні системи об'єднують переваги різних технологій та компенсують їх недоліки. Типова конфігурація сучасної системи протидронової оборони включає:

- Радар для первинного виявлення об'єктів на великих відстанях;
- Радіочастотний сканер для ідентифікації керованих дронів;
- Оптичну систему з комп'ютерним зором для точної класифікації та супроводу цілі;
- ІЧ-камери для роботи в умовах обмеженої видимості та вночі.

Інтеграція даних з різних сенсорів дозволяє досягти високої ймовірності виявлення (понад 95%) при низькому рівні хибних спрацювань (менше 5%). Проте вартість повнофункціональної гібридної системи становить від 200 тисяч до декількох мільйонів доларів, що обмежує можливості масового впровадження.

Аналіз сучасного стану технологій виявлення БПЛА дозволяє виділити наступні тренди розвитку галузі:

Збільшення ролі штучного інтелекту у процесі детекції та класифікації. Глибокі нейронні мережі демонструють точність виявлення понад 90% при високій швидкості обробки, що робить їх основою для систем наступного покоління.

Мініатюризація та зниження вартості сенсорів дозволяє створювати доступні системи для масового впровадження. Якщо ще п'ять років тому протидронова система коштувала мільйони, то сьогодні з'являються рішення вартістю 10-50 тисяч доларів.

Перехід до edge computing - обробка даних безпосередньо на граничних пристроях без необхідності передачі відео на центральний сервер. Це знижує затримки та підвищує надійність системи.

Розвиток технологій роєвого виявлення - мережі недорогих сенсорів покривають велику територію та обмінюються даними для створення єдиної картини повітряної обстановки.

1.2 Аналіз існуючих систем протидронової оборони та їх обмежень

Ринок систем протидронової оборони (Counter-UAV або C-UAV) активно розвивається останні п'ять років у відповідь на зростаючу загрозу від безпілотників. За оцінками аналітичних агентств, об'єм світового ринку C-UAV систем у 2023 році склав близько 1.8 мільярда доларів і прогнозується зростання до 5.6 мільярда до 2030 року з середньорічним темпом зростання 17.3%.

DroneShield (Австралія) пропонує лінійку продуктів від портативних детекторів до стаціонарних комплексів. Їх флагманська система DroneShield Complete, вигляд якої наведено на рис. 1.3, поєднує радіочастотне виявлення, радар та камери з можливістю активного придушення сигналу управління дроном. Система здатна виявляти дрони на відстані до 3 км, час реакції складає 2-3 секунди. Вартість комплексу становить від 150 до 300 тисяч доларів залежно від конфігурації.

Основні обмеження: висока вартість, обмежена ефективність проти автономних дронів, необхідність отримання дозволів на використання засобів радіоелектронної боротьби, складність інтеграції з існуючими системами безпеки.

Рис. 1.3 Приклад протидронової системи DroneShield

Dedrone (Німеччина/США) спеціалізується на програмно-апаратних комплексах для захисту критичної інфраструктури. Загальний вигляд та принцип побудови системи DedroneTracker показано на рис. 1.4. Їх система DedroneTracker використовує мультисенсорний підхід: радіочастотні датчики, камери, радари та мікрофони. Ключова перевага - розвинута аналітична платформа на базі машинного навчання, що забезпечує класифікацію загроз та мінімізацію хибних спрацювань.

Система успішно впроваджена в понад 500 локаціях, включаючи аеропорти, в'язниці, стадіони та промислові об'єкти. Ефективність виявлення сягає 95% при рівні хибних спрацювань менше 3%. Проте вартість повнофункціональної системи для великого об'єкта може перевищувати 500 тисяч доларів.

Рис. 1.4 Приклад протидронової системи Dedrone

Fortem Technologies (США) розробила унікальний підхід - дрони-перехоплювачі DroneHunter, що виявляють та фізично захоплюють ворожі БПЛА за допомогою спеціальної сітки. Система TrueView забезпечує радіолокаційне виявлення на відстані до 10 км з автоматичною класифікацією цілей за допомогою алгоритмів ШІ. Принцип роботи дрона-перехоплювача компанії Fortem Technologies показано на рис. 1.5.

Переваги: висока точність, можливість захоплення дрона для подальшого аналізу, відсутність радіоперешкод. Недоліки: висока вартість (понад 1 мільйон доларів за систему), обмежена кількість одночасних перехоплень, необхідність регулярного обслуговування дронів-перехоплювачів.

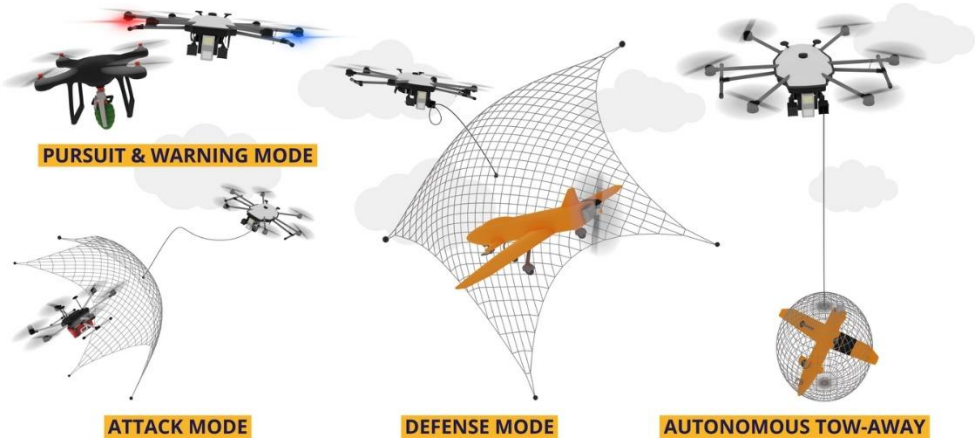


Рис. 1.5 Принцип роботи дрону перехоплювача компанії Fortem Technologies

Але військові системи характеризуються значно більшою потужністю, дальністю виявлення та ефективністю нейтралізації, проте їх вартість вимірюється мільйонами доларів.

Мобільна система AUDS (Anti-UAV Defence System, Велікобританія) поєднує радар Blighter, оптико-електронну систему та спрямований глушитель радіочастот потужністю до 80 Вт. Дальність виявлення радаром складає 8-10 км, оптичною системою - до 3 км. Час від виявлення до придушення - менше 15 секунд.

Система успішно пройшла випробування в Іраку та Афганістані, де продемонструвала ефективність проти дронів-камікадзе. Проте висока вартість (понад 2 мільйони доларів) обмежує масове застосування.

Лазерна система DE M-SHORAD (США) використовує твердотільний лазер потужністю 50 кВт для фізичного знищення дронів на відстані до 4 км. Принцип дії лазерної протидронової системи наведено на рис. 1.6. Система встановлюється на шасі бронемашини Stryker і забезпечує захист механізованих підрозділів від загроз з повітря.

Переваги: практично миттєва дія (швидкість світла), відсутність боєприпасів, низька вартість одного пострілу. Недоліки: залежність від погодних умов (туман та дощ суттєво знижують ефективність), висока вартість системи (понад 10 мільйонів доларів), великі енергетичні потреби.

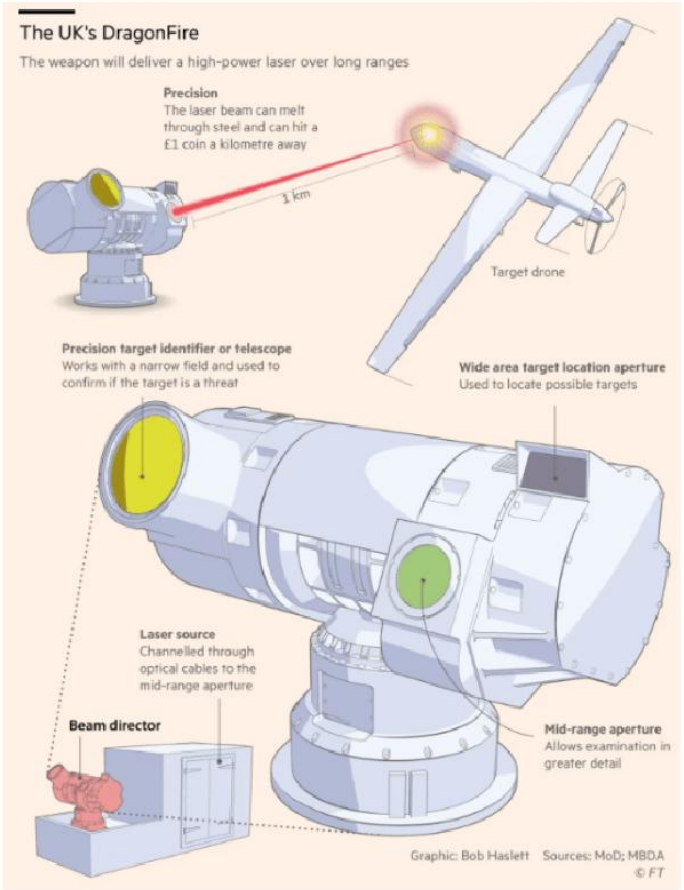


Рис. 1.6 Принцип дії лазерної протидронової системи

1.3 Специфічні виклики розпізнавання дронів у відеопотоці

Розпізнавання безпілотних літальних апаратів у відеопотоці є значно складнішим завданням порівняно з детекцією традиційних об'єктів (автомобілів, людей, тварин) через ряд специфічних факторів, що створюють унікальні виклики для систем комп'ютерного зору.

Сучасні дрони характеризуються надзвичайно високою різноманітністю конструкцій, що наведено на рис. 1.7. Від мініатюрних квадрокоптерів вагою менше 250 грамів з діаметром 20-30 см до великих гексакоптерів або октокоптерів з розмахом понад 2 метри. Форма також варіюється: класичні квадрокоптери з чотирма пропелерами, літакового типу з фіксованим крилом, гібридні VTOL-системи, коаксіальні вертольотного типу.

Ця варіативність створює проблему для моделей машинного навчання, оскільки одна модель повинна ефективно розпізнавати об'єкти з різною геометрією, пропорціями та візуальними характеристиками. На відміну від автомобілів, що мають відносно стандартну форму (корпус + колеса), дрони не мають єдиного набору визначальних ознак.

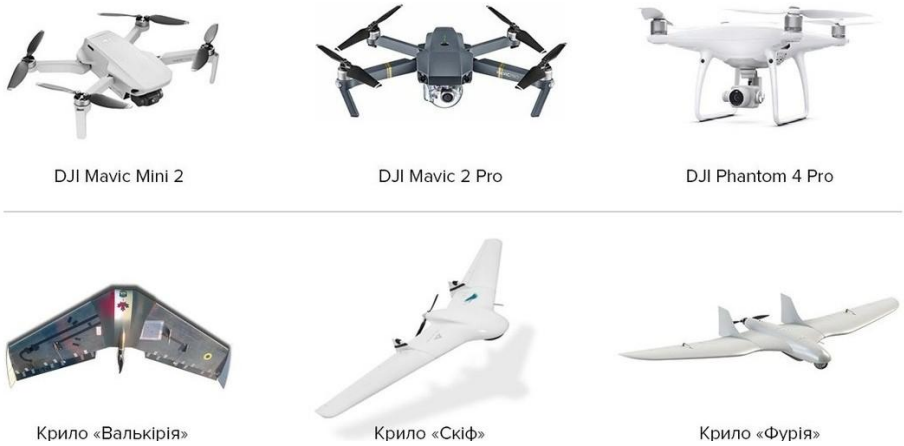


Рис. 1.7 Різноманітність форм та розмірів сучасних дронів

Сучасні спортивні та військові дрони здатні розвивати швидкість понад 100-150 км/год. При обробці відео з частотою 30 кадрів за секунду, дрон може пролетіти 1-2 метри між кадрами. Це створює ефект розмиття руху (motion blur) та ускладнює точну локалізацію об'єкта.

Траєкторії польоту дронів відрізняються високою непередбачуваністю: різкі зміни напрямку, зависання на місці, вертикальні маневри, обертання навколо власної осі. Типові траєкторії руху дронів різних типів показано на рис. 1.8. Алгоритми трекінгу повинні враховувати цю динаміку для стабільного супроводу цілі між кадрами.

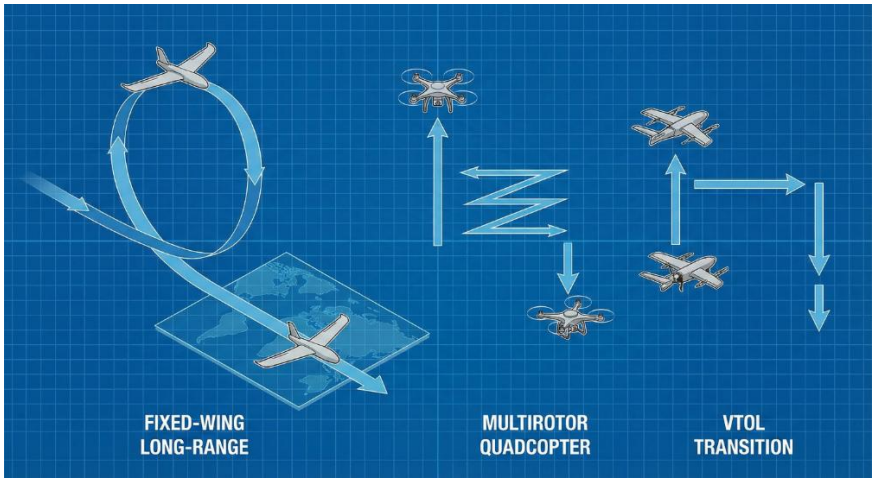


Рис. 1.8 Траєкторії руху дронів різних типів

Дрони зазвичай виявляються на фоні неба, що створює специфічні виклики. Небо може мати різний колір та текстуру залежно від часу доби, погоди, наявності хмар. При спостереженні під малим кутом піднесення дрон може виявлятися на фоні будівель, дерев, пагорбів, що ускладнює його виокремлення.

Особливо складним сценарієм є виявлення дрона на фоні однорідного сірого неба або в умовах низької хмарності, коли об'єкт практично зливається з фоном. У такі моменти основними ознаками для детекції залишаються форма та можливе обертання пропелерів.

Погодні умови суттєво впливають на якість зображення та, відповідно, на точність детекції:

Туман та димка знижують контрастність зображення, зменшують деталізацію об'єктів на відстані. При щільності туману понад 200 метрів дальність ефективного виявлення може скоротитися в 3-5 разів.

Дощ та сніг створюють додатковий шум на зображенні, краплі на об'єктиві камери спотворюють картинку. Сніжинки, що летять, можуть помилково класифікуватися як малі дрони при недостатній якості моделі.

Яскраве сонячне світло може створювати блики на камері, засвічені ділянки зображення, різкі тіні. При спостереженні проти сонця дрон може перетворитися на темний силует без деталей, що ускладнює класифікацію.

Низька освітленість в сутінках або на світанку знижує відношення сигнал/шум матриці камери, деталі об'єкта стають менш чіткими. Нічна зйомка без спеціалізованого обладнання (ІЧ-камери) практично неможлива.

Типовий квадрокоптер DJI Mavic з розміром 20×20 см на відстані 300 метрів при зйомці камерою Full HD (1920×1080) займатиме лише 15-20 пікселів в діаметрі. Це створює проблему для згорткових нейронних мереж, оскільки після декількох шарів пулінгу об'єкт може повністю "зникнути" або перетворитися на 2-3 пікселі.

Сучасні архітектури детекції об'єктів, такі як YOLO або SSD, оптимізовані для виявлення об'єктів середнього та великого розміру (що займають понад 32×32 пікселі). Для ефективної роботи з малими об'єктами потрібні спеціалізовані модифікації архітектури: додаткові шари детекції для високих роздільних здатностей, спеціальні модулі для збереження просторової інформації, методи data augmentation для генерації прикладів малих об'єктів.

Вимога роботи в реальному часі накладає жорсткі обмеження на складність моделі. Для забезпечення швидкості обробки не менше 20-30 FPS (що необхідно для стабільного трекінгу) модель повинна обробляти один кадр за 30-50 мілісекунд. Це обмежує глибину мережі, кількість параметрів та роздільну здатність вхідного зображення.

На граничних пристроях (edge devices) з обмеженими обчислювальними можливостями проблема стає ще гострішою. Бортовий комп'ютер типу NVIDIA Jetson Nano має продуктивність близько 472 GFLOPS, що в десятки разів менше, ніж десктопна відеокарта RTX 3060. Модель повинна бути оптимізована для роботи на такому обладнанні без критичної втрати точності. У табл. 1.2 наведено порівняння архітектур нейронних мереж за обчислювальною складністю та швидкодією

Таблиця 1.2

Вимоги до обчислювальної потужності для різних архітектур

Архітектура (Модель)	Складність (GFLOPS)	FPS (RTX 3060)	FPS (Jetson Nano)	Якість на малих об'єктах
SSD MobileNet V2	~2.5 (Дуже низька)	> 150	30 - 45	Низька
YOLOv8 Nano (n)	~8.1 (Низька)	> 140	25 - 35	Середня
YOLOv5	~16.5	> 120	15 - 20	Середня

Small (s)	(Середня)			
EfficientDet-D0	~2.5 (Низька)	> 100	15 - 22	Висока (відносно розміру)
Faster R-CNN (ResNet)	> 100 (Дуже висока)	20 - 30	< 5	Висока

А також велика проблема полягає у тому, що більшість існуючих датасетів зосереджені на денних умовах зйомки з відносно хорошою видимістю. Даних для навчання моделей роботи в складних умовах (сутінки, туман, дощ, складний фон) критично не вистачає.

1.4 Основи комп'ютерного зору та обробки відео в реальному часі

Комп'ютерний зір - це міждисциплінарна область, що поєднує методи обробки зображень, машинного навчання та комп'ютерних наук для автоматичного аналізу візуальної інформації. В контексті розпізнавання дронів комп'ютерний зір виконує функції, аналогічні людському зору: виявлення об'єктів, класифікація, трекінг та аналіз поведінки.

Цифрове зображення представляє собою двовимірний масив пікселів, кожен з яких характеризується значенням інтенсивності (для чорно-білих зображень) або набором значень колірних каналів (для кольорових зображень). Стандартна модель RGB використовує три канали: червоний (Red), зелений (Green) та синій (Blue), кожен з яких зберігає значення від 0 до 255. Представлення цифрового зображення у вигляді тривимірного масиву пікселів показано на рис. 1.9

Зображення з роздільною здатністю 1920×1080 пікселів містить 2,073,600 пікселів, що при представленні в форматі RGB дає масив розміром 1920×1080×3 = 6,220,800 значень. Така велика розмірність вхідних даних створює виклик для алгоритмів обробки та є однією з причин використання згорткових операцій для зменшення розмірності.

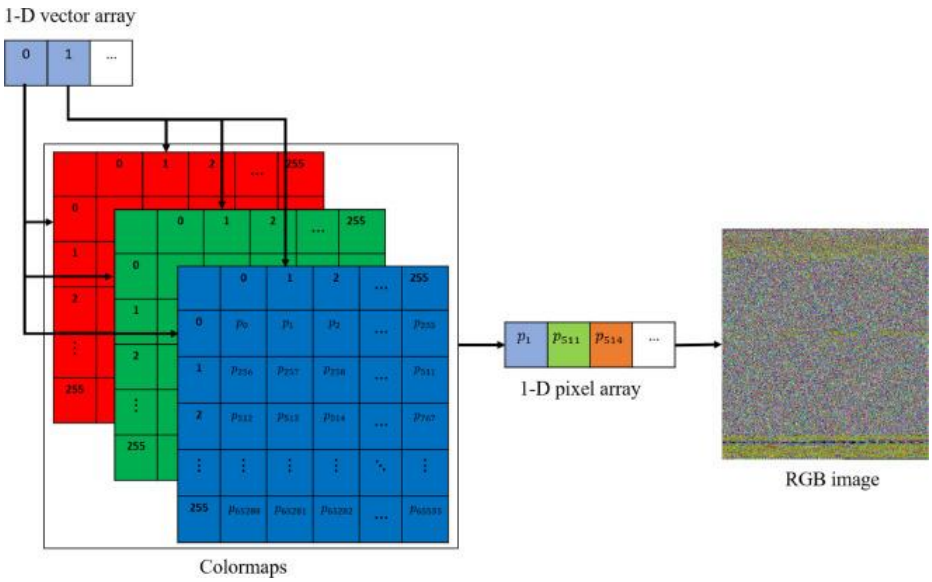


Рис. 1.9 Представлення цифрового зображення як тривимірного масиву

Базові операції обробки зображень включають:

Фільтрація - застосування згорткових ядер для виділення певних ознак зображення. Фільтр Гауса згладжує зображення та зменшує шум, оператор Собеля виділяє границі об'єктів, фільтр Лапласа виявляє області різкої зміни яскравості.

Перетворення кольорового простору - конвертація з RGB в інші представлення (HSV, LAB, YUV), що може полегшити виділення певних характеристик. Простір HSV (Hue-Saturation-Value) зручний для сегментації за кольором, оскільки відокремлює інформацію про колір від яскравості.

Морфологічні операції - ерозія, дилатація, відкриття та закриття, що використовуються для видалення шуму, заповнення отворів у об'єктах та покращення їх контурів.

Нормалізація - приведення значень пікселів до стандартного діапазону (зазвичай [0, 1] або [-1, 1]), що стабілізує процес навчання нейронних мереж.

Сучасні детектори об'єктів поділяються на два основні класи:

Двоетапні детектори (two-stage detectors) спочатку генерують пропозиції регіонів, де можуть знаходитися об'єкти, а потім класифікують ці регіони. Представники: R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN. Переваги: висока точність детекції. Недоліки: низька швидкість обробки (5-15 FPS).

Одноетапні детектори (one-stage detectors) виконують детекцію за один

прохід мережі, що забезпечує високу швидкість при дещо нижчій точності. Представники: YOLO (всі версії), SSD, RetinaNet, EfficientDet. Переваги: висока швидкість (30-150 FPS). Недоліки: дещо нижча точність, особливо для малих об'єктів.

Обидва класи детекторів, як двоетапні, так і одноетапні базуються на глибоких нейронних мережах, здатних ефективно обробляти зображення та виокремлювати з них інформативні ознаки. Основою таких моделей є архітектури, що можуть автоматично навчатися виявляти просторові патерни різної складності. Найпоширенішим типом таких моделей є саме згорткові нейронні мережі, які забезпечують ефективне представлення зорових даних.

Згорткова нейронна мережа (ЗНМ) - це спеціалізована архітектура глибокого навчання, приклад якої наведено на рис. 1.10, що використовує операцію згортки для обробки даних з топологічною структурою (зображення, відео, аудіо).

Ключові компоненти ЗНМ:

Згортковий шар (Convolutional Layer) - основний будівельний блок мережі. Застосовує набір навчаємих фільтрів (ядер) до вхідного зображення. Кожен фільтр виявляє певну ознаку: границі, текстури, кольорові переходи. У глибоких шарах фільтри вчаться розпізнавати більш складні патерни: частини об'єктів, геометричні форми, семантичні структури.

Згортка зменшує розмірність даних порівняно з повнозв'язними шарами. Замість з'єднання кожного нейрона з кожним нейроном попереднього шару, згортковий шар використовує локальні з'єднання - кожен нейрон обробляє лише невелику область (receptive field) попереднього шару.

Шар пулінгу (Pooling Layer) зменшує просторову розмірність карт ознак, зберігаючи найважливішу інформацію. Max pooling вибирає максимальне значення у вікні, average pooling обчислює середнє. Пулінг забезпечує інваріантність до невеликих зсувів об'єкта та зменшує кількість параметрів мережі.

Функції активації вносять нелінійність у модель. ReLU (Rectified Linear Unit): $f(x) = \max(0, x)$ - найпопулярніша функція завдяки простоті обчислень

та ефективності навчання. Leaky ReLU вирішує проблему "мертвих нейронів": $f(x) = max(0.01x, x)$. Swish та Mish - сучасніші активації, що демонструють кращу точність на деяких задачах.

Batch Normalization нормалізує активації між шарами, що прискорює збіжність навчання та дозволяє використовувати вищі швидкості навчання.

Повнозв'язні шари (Fully Connected Layers) використовуються на завершальних етапах для фінальної класифікації на основі виділених високорівневих ознак.

Описані вище компоненти формують основу сучасних моделей комп'ютерного зору. Використовуючи ці архітектурні елементи, нейронні мережі здатні ефективно аналізувати не лише статичні зображення, а й послідовності кадрів у відео. Це особливо важливо для задач, де потрібна обробка даних у реальному часі.

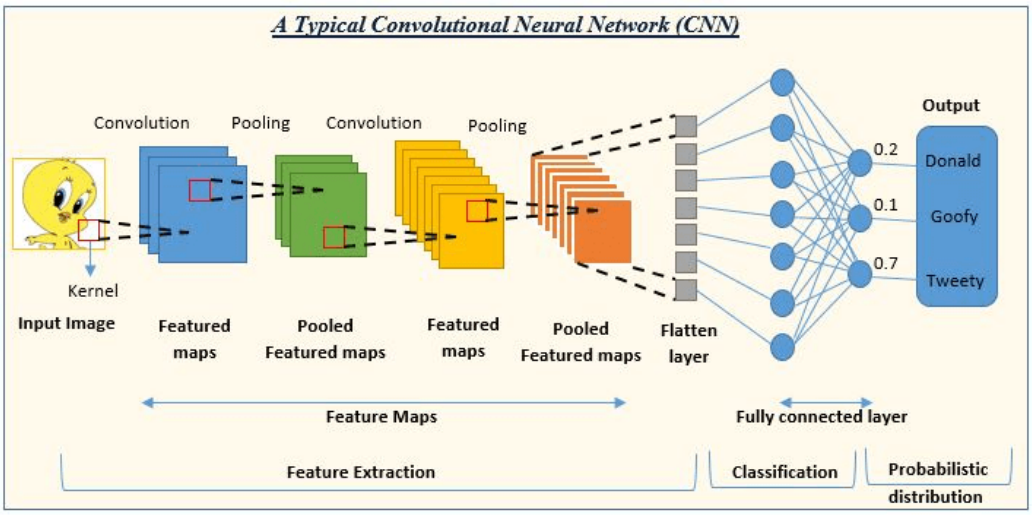


Рис. 1.10 Архітектура типової згорткової нейронної мережі

Адже відео - це послідовність статичних зображень (кадрів), що відтворюються з певною частотою. Стандартні частоти: 24-30 FPS для відео, 50-60 FPS для високоякісного контенту, 120+ FPS для спортивної зйомки.

Обробка відео в реальному часі означає, що система повинна обробити кадр швидше, ніж надходить наступний. Для відео 30 FPS це означає обробку за 33 мілісекунди або менше. З урахуванням часу на захоплення кадру, передачу даних та відображення результату, на власне обробку залишається 20-25 мс.

Pipeline обробки відеопотоку для детекції дронів:

1. Захоплення кадру з камери або відеофайлу за допомогою бібліотек OpenCV, FFmpeg або спеціалізованих SDK камер.
2. Попередня обробка: зміна розміру до вхідної роздільності моделі (зазвичай 640×640 або 1280×1280 для YOLO), нормалізація значень пікселів, можлива конвертація кольорового простору.
3. Інференс нейронної мережі: пропуск кадру через натреновану модель для отримання передбачень (bounding boxes + класи + рівні впевненості).
4. Постобробка: фільтрація передбачень за порогом впевненості (зазвичай >0.5), застосування Non-Maximum Suppression (NMS) для видалення дублікатів.
5. Трекінг об'єктів: зв'язування детекцій між кадрами для формування траєкторій руху. Алгоритми: SORT (Simple Online Realtime Tracking), DeepSORT, ByteTrack.
6. Візуалізація та вивід: накладання обмежуючих прямокутників, міток класів, траєкторій на оригінальне зображення, збереження або відображення результату.

Загальний pipeline обробки відеопотоку для детекції дронів показано на рис. 1.11.

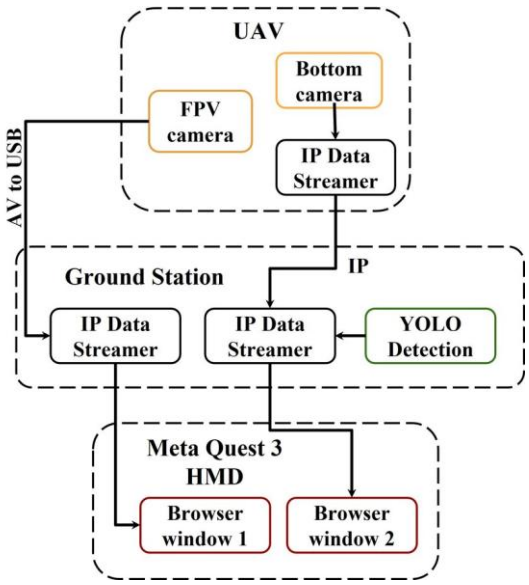


Рис. 1.11 Pipeline обробки відеопотоку для детекції дронів

Для досягнення роботи в реальному часі застосовуються наступні методи: Зменшення роздільної здатності вхідного зображення - найпростіший

спосіб прискорення. Зменшення з 1280×1280 до 640×640 прискорює обробку в 4 рази, але може знизити точність детекції малих об'єктів.

Квантизація моделі - перетворення ваг з float32 (32 біти) у int8 (8 біт) може прискорити інференс у 2-4 рази при зниженні точності на 1-2%. TensorRT, OpenVINO, ONNX Runtime підтримують квантизацію.

Обрізка (pruning) видаляє менш важливі нейронні з'єднання, зменшуючи розмір моделі на 30-50% з мінімальною втратою точності.

Використання спеціалізованих бібліотек - TensorRT (NVIDIA), OpenVINO (Intel), CoreML (Apple) оптимізують моделі для конкретного апаратного забезпечення.

Батчинг - обробка декількох кадрів одночасно збільшує утилізацію GPU, але додає затримку, тому не завжди придатна для real-time систем.

Для оцінки ефективності детекторів об'єктів використовуються наступні метрики:

Intersection over Union (IoU) - відношення площі перетину передбаченого та справжнього bounding box до площі їх об'єднання. IoU > 0.5 вважається успішною детекцією, IoU > 0.75 - високоточною.

Precision (точність) = TP / (TP + FP) - відношення коректних детекцій до всіх передбачень. Високий precision означає мало хибних спрацювань.

У табл. 1.3 наведено вплив основних методів оптимізації на швидкість роботи та точність моделей.

Таблиця 1.3

Вплив методів оптимізації на швидкість та точність

Метод оптимізації	Механізм дії	Вплив на швидкість / Продуктивність	Вплив на точність
Зменшення роздільної здатності	Зміна розміру вхідного зображення (наприклад, з 1280×1280 до 640×640).	Прискорення у 4 рази (квадратична залежність від розміру сторони).	Можливе зниження точності детекції для малих об'єктів.

Квантизація моделі	Перетворення ваг з float32 (32 біти) у int8 (8 біт).	Прискорення інференсу в 2-4 рази.	Незначне зниження на 1-2%.
Обрізка	Видалення менш важливих нейронних з'єднань (ваг, близьких до нуля).	Зменшення розміру моделі на 30-50% (що прискорює завантаження та обробку).	Мінімальна втрата точності.
Спеціалізовані бібліотеки	Оптимізація графу обчислень під конкретне апаратне забезпечення.	Максимальна оптимізація під конкретне залізо (GPU, CPU, NPU).	Зазвичай без втрат або з мінімальними змінами.
Батчинг	Одночасна обробка групи (пакету) кадрів.	Збільшує загальну утилізацію GPU (пропускну здатність).	Не впливає на точність.

Recall (повнота) = TP / (TP + FN) - відношення коректних детекцій до всіх справжніх об'єктів. Високий recall означає, що модель знаходить більшість об'єктів.

Average Precision (AP) - площа під кривою precision-recall. Обчислюється для різних порогів IoU (AP50, AP75).

mAP (mean Average Precision) - середнє значення AP по всіх класах об'єктів. mAP50 - найпоширеніша метрика для порівняння детекторів.

FPS (Frames Per Second) - кількість кадрів, оброблених за секунду. Критична метрика для real-time систем.

1.5 Огляд наборів даних для навчання моделей детекції дронів

Якість та обсяг навчальних даних є критичним фактором успіху будь-якої системи машинного навчання. Для задачі детекції дронів існує обмежена кількість публічно доступних датасетів, кожен з яких має свої особливості, переваги та обмеження.

Anti-UAV є одним з найбільших та найрізноманітніших датасетів для детекції та трекінгу безпілотників. Він був створений в рамках міжнародних змагань з протидронових технологій та серед усіх релізів можна виділити дві ключові версії, які стали найбільш корисними та впливовими:

Anti-UAV v1 (2020 рік) включає 160 відеопослідовностей загальною тривалістю понад 4 години. Датасет містить три трекінгові набори (A, B, C) з різними рівнями складності та один тестовий набір. Відео знято в різних умовах: вдень, ввечері, в різну погоду, з різними типами дронів та фонів.

Anti-UAV v2 (2021 рік) розширює попередню версію, додаючи 250+ послідовностей з покращеною розмічкою. Загальна кількість розмічених кадрів перевищує 320,000.

Особливості датасету:

- Роздільна здатність відео: 1920×1080 пікселів
- Частота кадрів: 30 FPS
- Формат розмітки: XML (PASCAL VOC формат) з координатами bounding box
- Типи дронів: квадрокоптери DJI (Mavic, Phantom), гексакоптери, власні збірки
- Середній розмір дрона на зображенні: 40-120 пікселів
- Сценарії: польоти на різній висоті (10-150 м), швидкості (0-60 км/год), відстані від камери (50-500 м)

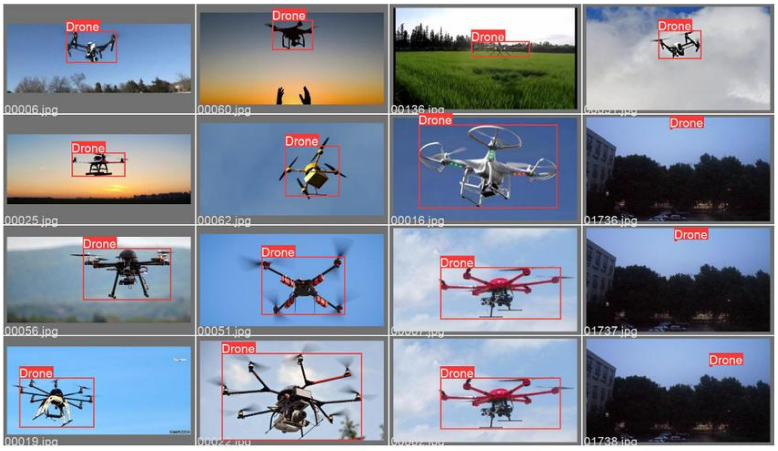
Переваги:

- Великий обсяг даних для навчання
- Різноманітність сценаріїв та умов зйомки
- Професійна розмітка з високою точністю
- Регулярні оновлення та підтримка спільноти

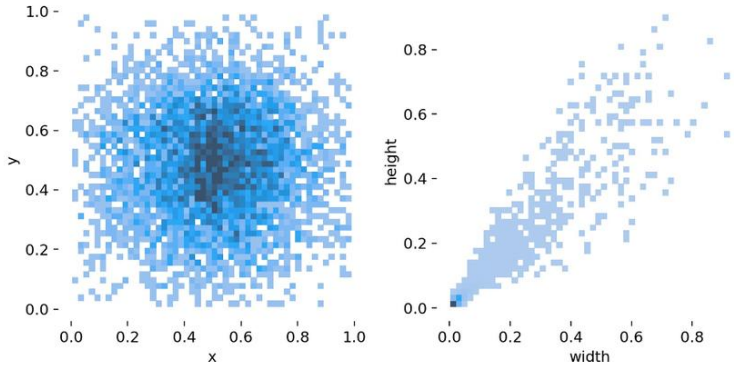
Недоліки:

- Переважно денні умови зйомки (>85% даних)
- Обмежена різноманітність типів дронів
- Недостатньо прикладів складних погодних умов (туман, дощ)
- Відсутність нічних зйомок

Приклади зображень та статистика анотацій датасету Anti-UAV наведені на рис. 1.12.



(a)



(b)

Рис.1.12 Приклади даних і статистика анотацій у датасеті

Також дослідниками з університету Цюріха було зібрано спеціально для навчання моделей детекції дронів у реальному часі датасет, він містить 28,500 зображень з розміткою, зібраних з різних джерел.

Структура датасету:

- Позитивні приклади: 15,000 зображень з дронами
- Негативні приклади: 13,500 зображень без дронів (небо, птахи, літаки)
- Розмітка: JSON формат з координатами bounding box, класом об'єкта, рівнем складності

Класифікація зображень за складністю:

- Easy (30%): дрон займає >100 пікселів, чистий фон, гарна видимість
- Medium (50%): дрон займає 50-100 пікселів, складний фон або середня видимість
- Hard (20%): дрон займає <50 пікселів, дуже складний фон, погана видимість або часткове перекриття

Приклад тестування моделі на датасеті Drone-vs-Bird показано на рис. 1.13.

Особливості:

- Висока різноманітність фонів: небо, ліси, міста, гори, водойми
- Різні умови освітлення: ранок, полудень, вечір, хмарно, сонячно
- Включення "складних негативних прикладів": птахи, літаки, повітряні кулі, що схожі на дрони
- Баланс класів: 53% позитивних, 47% негативних прикладів

Переваги:

- Ретельно відібрані негативні приклади для зниження false positives
- Градація за складністю дозволяє аналізувати залежність точності від умов
- Різноманітність фонів покращує генералізацію моделі

Недоліки:

- Менший обсяг порівняно з Anti-UAV
- Статичні зображення замість відео (не підходить для трекінгу)
- Обмежена інформація про параметри зйомки

Для тестування оцінки алгоритмів візуального трекінгу безпілотників було створено датасет UAV123, який містить 123 відеопослідовності з детальною розміткою траєкторій об'єктів.

Характеристики:

- Загальна кількість кадрів: 112,578
- Середня тривалість послідовності: 915 кадрів
- Роздільна здатність: 1280×720 та 1920×1080
- Частота кадрів: 30 FPS

Датасет включає атрибути складності:

- Aspect Ratio Change: зміна співвідношення сторін об'єкта (33 послідовності)

- Background Clutter: складний, зашумлений фон (21)
- Camera Motion: рух камери (52)
- Fast Motion: швидкий рух об'єкта (48)
- Full Occlusion: повне перекриття об'єкта (29)
- Illumination Variation: зміна освітлення (38)
- Low Resolution: низька роздільна здатність об'єкта (16)
- Out-of-View: вихід об'єкта за межі кадру (19)
- Partial Occlusion: часткове перекриття (42)
- Similar Object: наявність схожих об'єктів (16)
- Scale Variation: зміна масштабу (55)
- Viewpoint Change: зміна ракурсу (36)

Переваги:

- Детальна розмітка з покадровими bounding boxes
- Систематична категоризація викликів трекінгу
- Широко використовується як benchmark для порівняння алгоритмів

Недоліки:

- Фокус на трекінгу, а не детекції
- Обмежена різноманітність типів дронів
- Відсутність анотацій класів об'єктів

Під час тестування моделі трекінгу безпілотників може виникнути проблема відрізнення дронів від птахів. Для вирішення цієї проблеми є датасет Drone-vs-Bird, який складається з приблизно 8000 зображень дронів у польоті, 7500 зображень птахів у польоті, 5000 зображень порожнього неба, у вигляді негативних прикладів та має роздільну здатність від 640x480 до 1920x1080

Його особливості полягають у:

- Зйомці під різними кутами (0° - 90° від горизонту)
- Різними відстаннями (20-300 метрів)
- Включення різних видів птахів (голуби, ворони, яструби, чайки)
- Анотація форми крил та характеру руху

Рис. 1.13 Приклад тестування моделі на датасеті Drone-vs-Bird

Переваги:

- Вирішує критичну проблему класифікації дрон/птах
- Висока якість розмітки з детальними атрибутами

Недоліки:

- Вузька спеціалізація датасету
- Невеликий обсяг

Але через обмеженість реальних даних, дослідники все частіше використовують синтетичні дані, згенеровані в симуляторах. Порівняння реальних та синтетичних зображень дронів наведено на рис. 1.14.

AirSim - відкрита платформа Microsoft для симуляції дронів та автономних транспортних засобів. Дозволяє генерувати фотореалістичні зображення в різних середовищах (міста, ліси, гори) з повним контролем над параметрами.

Gazebo + ROS - комбінація симулятора робототехніки та операційної системи для роботів. Використовується для генерації навчальних даних з автоматичною розміткою.

Рис.1.14 Порівняння реальних та синтетичних зображень дронів

Переваги синтетичних даних:

- Необмежений обсяг
- Автоматична розмітка
- Повний контроль над умовами (погода, освітлення, фон)
- Можливість генерації рідкісних сценаріїв

Недоліки:

- Domain gap: відмінності між синтетичними та реальними зображеннями
- Необхідність domain adaptation для ефективного використання

Методи аугментації даних

Для збільшення ефективного обсягу датасету застосовуються техніки аугментації, такі як: геометричні перетворення, фотометричні перетворення.

РОЗДІЛ 2 АРХІТЕКТУРИ НЕЙРОННИХ МЕРЕЖ ДЛЯ ДЕТЕКЦІЇ ДРОНІВ

2.1 Згорткові нейронні мережі для детекції об'єктів

Згорткові нейронні мережі революціонізували область комп'ютерного зору, забезпечивши прорив у точності детекції та класифікації об'єктів. У цьому підрозділі розглянемо фундаментальні принципи роботи ЗНМ та їх застосування для детекції об'єктів.

Операція згортки (convolution) є основою мережі. Математично дискретна двовимірна згортка визначається як:

$$(f \star g)(i,j) = \sum_m \sum_n f(i+m, j+n)\, g(m,n)$$

Де f - вхідне зображення, g - ядро фільтра (kernel), (i,j) - координати пікселя вихідного зображення. Графічну інтерпретацію операції згортки наведено на **рис. 2.1.**

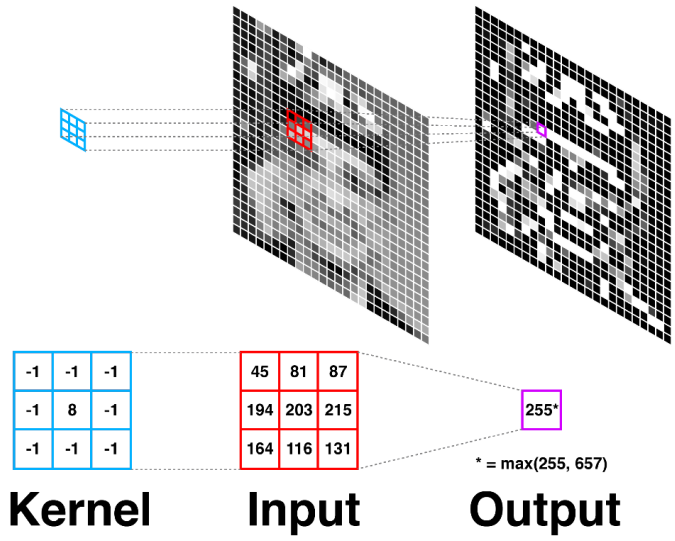


Рис. 2.1- Візуалізація операції згортки на прикладі

Для практичої реалізаці згорткового шару найлегшим способом буде використання мови програмуння Python у зв’язці з бібліотекою PyTorch або Tensorflow.

```python
import torch
import torch.nn as nn

# Згортковий шар: 3 вхідні канали (RGB), 64 вихідні канали, ядро 3×3
conv_layer = nn.Conv2d(in_channels=3, out_channels=64,
                       kernel_size=3, stride=1, padding=1)

# Приклад обробки батчу зображень
batch_images = torch.randn(8, 3, 640, 640)  # 8 зображень 640×640
output = conv_layer(batch_images)
print(f"Вхід: {batch_images.shape}")  # [8, 3, 640, 640]
print(f"Вихід: {output.shape}")       # [8, 64, 640, 640]
```

З ключевих компонентів сучасних згорткових нейронних мереж можна виділити Residual Connections (залишкові з'єднання), які вирішують проблему зникаючого градієнта в глибоких мережах, Batch Normalization, який нормалізує активації між шарами, забезпечуючи стабільність навчання та Attention Mechanisms, що дозволяють мережі фокусуватися на важливих регіонах

Для екстракції ознак використовуються архітектури backbone. Взагалі Backbone - це базова мережа для виділення ознак з зображення. Сучасні детектори використовують pre-trained backbone на великих датасетах , наведено у табл. 2.1. Однак сучасні системи виявлення об’єктів вже не обмежуються використанням лише класичних архітектур на зразок ResNet.

Розвиток детекції привів до появи спеціалізованих backbone-моделей та цілих модульних структур, оптимізованих під багатомасштабну обробку зображень та ефективне поєднання ознак різного рівня. Саме тому в сучасних детекторах все частіше застосовуються більш комплексні архітектури, які поєднують високорівневі та низькорівневі ознаки

Вони забезпечують глибоке повторне використання інформації та підвищують точність розпізнавання дрібних об’єктів.

Таблиця 2.1

Порівняння різних архітектур ResNet як backbone для виявлення об'єктів

		MOTA	MOTP	MT	ML	ID	FM	FP	FN	Runtime
KDNT [43]	Batch based	68.2	79.4	41.0%	19.0%	933	1093	11,479	45,605	0.7 Hz
LMP p [44]	Batch based	71.0	80.2	46.9%	21.9%	434	587	7880	44,564	0.5 Hz
MCMOT HDM [45]	Batch based	62.4	78.3	31.5%	24.2%	1394	1318	9855	57,257	35 Hz
NOMTwSDP16 [46]	Batch based	62.2	79.6	32.5%	31.1%	406	642	5119	63,352	3 Hz
EAMTT [47]	Real time	52.5	78.8	19.0%	34.9%	910	1321	4407	81,223	12 Hz
POI [43]	Real time	66.1	79.5	34.0%	20.8%	805	3093	5061	55,914	10 Hz
SORT [6]	Real time	59.8	79.6	25.4%	22.7%	1423	1835	8698	63,245	60 Hz
Deep SORT [7]	Real time	61.4	79.1	32.8%	18.2%	781	2008	12,852	56,668	40 Hz
Proposed system	Real time	75.2	81.3	33.2	17.5	825	1225	4123	52,524	42 Hz

CSPDarknet є основним backbone у сімействі YOLO. Його архітектура побудована за принципом поділу потоку ознак на дві паралельні гілки, де одна частина проходить через послідовність резидуальних блоків, а інша - напряму. Після цього обидві гілки об’єднуються та пропускаються через завершальний згортковий шар. Такий підхід дозволяє зменшити кількість обчислень і покращити ефективність навчання без втрати якості.

EfficientNet - це сімейство моделей, у якому масштабування глибини, ширини та роздільності відбувається збалансовано завдяки підходу compound scaling. У контексті детекції об’єктів EfficientNet часто використовують як backbone: з проміжних шарів моделі отримують багаторівневі карти ознак. Кожен блок формує свої просторові розміри, що дозволяє формувати набір ознак різних масштабів.

FPN - це архітектурне рішення для отримання ознак різних масштабів, необхідних для виявлення великих і малих об’єктів. Принцип роботи FPN включає:

1. Бічні (lateral) перетворення — з допомогою них вирівнюють кількість каналів у всіх рівнях ознак.
2. Top-down pathway — старші (глибші) ознаки масштабуються вгору та додаються до відповідних молодших рівнів.
3. Згладжувальні перетворення — забезпечують стабільність та покращують якість сформованої піраміди.

На виході отримується набір карти ознак (P2, P3, P4, P5), кожна з яких відповідає своєму масштабу. Приклад архітектури FPN наведена на рис. 2.2

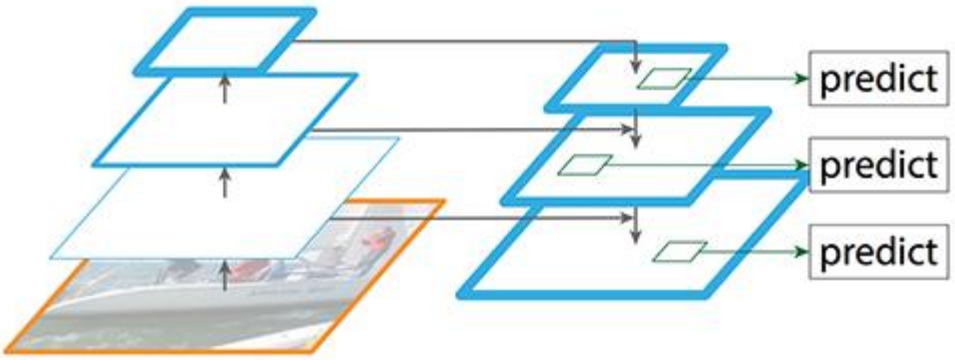


Рис. 2.2 Архітектура Feature Pyramid Network

Path Aggregation Network (PANet) розширює FPN, додаючи *bottom-up pathway* - зворотній шлях знизу догори. Це покращує передачу низькорівневих ознак та робить детектор більш чутливим до дрібних об’єктів.

PANet виконує:

- послідовне зменшення просторової роздільності,
- об’єднання цих ознак із відповідними рівнями FPN,
- згорткове згладжування.

У результаті сформована піраміда стає більш інформативною як у високорівневих, так і в низькорівневих ознаках.

Detection Head обробляє карти ознак і перетворює їх на три типи прогнозів:

1. Класифікація - ймовірність належності кожної комірки до певного класу.
2. Регресія bounding boxes - координати та розміри передбачених рамок.

3. Objectness score - оцінка ймовірності, що в конкретній комірці присутній об’єкт.

Кожен із цих компонентів обчислюється окремою гілкою згорткової мережі.

Anchor-based підхід використовує набір заздалегідь визначених “еталонних” anchors, різних за масштабом і співвідношеннями сторін. Для кожної комірки на карті ознак прогнозуються:

- зміщення відносно anchor-box,
- клас об’єкта,
- objectness score.

Цей підхід застосовували у YOLOv3, RetinaNet, Faster R-CNN тощо.

У моделях типу FCOS чи CenterNet координати коробки прогнозуються напряму, без використання anchors.

Такі системи зазвичай прогнозують:

- карту центрів об’єктів (heatmap),
- розміри коробок,
- зміщення від центру комірки.

Anchor-free підхід спрощує архітектуру і часто дає кращу стабільність на дрібних об’єктах. Візуалізація anchor boxes на feature map зображена на рис. 2.3.

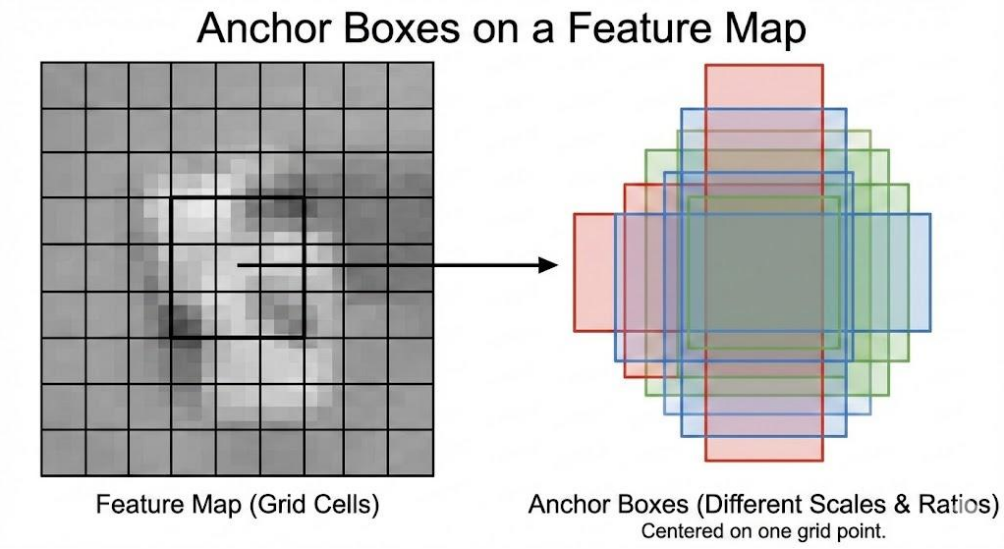


Рис. 2.3 Візуалізація anchor boxes на feature map

Функція втрат у задачах детекції зазвичай складається з трьох основних частин:

1. Loss класифікації - оцінює, наскільки точно модель визначає клас об’єкта.
2. Loss локалізації - базується на метриках накладання рамок, таких як IoU, GIoU, DIoU або CIoU.
3. Objectness loss - визначає, чи дійсно в прогнозованій точці є об’єкт.

Загальна функція втрат - це зважена комбінація трьох складових.

Non-Maximum Suppression (NMS) використовується для усунення повторюваних детекцій. Алгоритм:

1. Сортує всі детекції за score.
2. Береться найбільш впевнена рамка.
3. Рамки, що мають із нею надто велике перекриття (вище порогу IoU), відкидаються.
4. Процес повторюється до завершення списку.

2.2 Аналіз архітектур сімейства YOLO (v5, v7, v8, v11)

YOLO (You Only Look Once) - це сімейство одноетапних детекторів, що революціонізували галузь детекції об'єктів завдяки оптимальному балансу між швидкістю та точністю.

YOLOv5, архітектура якої наведена на рис. 2.4, розроблений компанією Ultralytics у 2020 році, став найпопулярнішою версією завдяки простоті використання та високій ефективності.

Архітектура моделі складається з трьох основних компонентів, а саме з backbone (CSPDarknet53), який відповідає за вилучення багатомасштабних ознак, neck (PANet у поєднанні з блоком Spatial Pyramid Pooling), що забезпечує ефективне об’єднання та передачу ознак між різними рівнями та head detection, яка виконує передбачення на трьох масштабах, охоплюючи як великі, так і малі об’єкти.

YOLOv5 представлений у кількох версіях, орієнтованих на різні сценарії використання:

- Nano (найшвидша);
- Small;

- Medium;
- Large;
- XLarge (найточніша).

Користувач може обрати модель залежно від того, що є критичнішим - швидкість або якість виявлення. Процес навчання YOLOv5 на власному датасеті зазвичай складається з підготовки конфігурації даних, ініціалізації моделі, навчання з оптимізованими гіперпараметрами, подальшої валідації та, за потреби, експорту моделі у формат ONNX для швидкого інференсу на продакшн-системах.

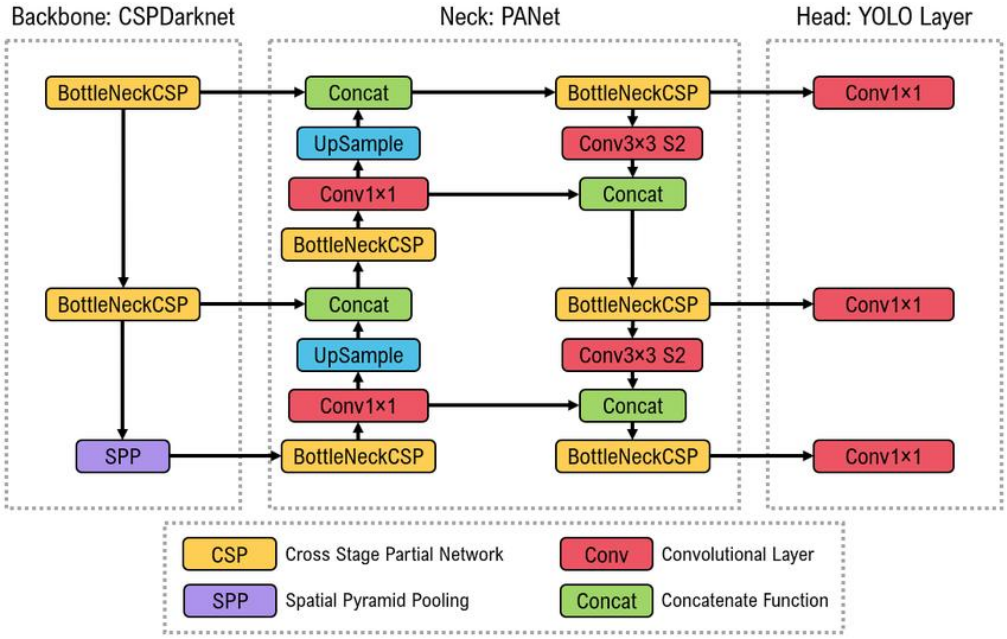


Рис. 2.4 Архітектура YOLOv5

У 2022 році було представлено YOLOv7, у якій уперше з’явився блок E-ELAN (Extended Efficient Layer Aggregation Network). Його особливість полягає у більш ефективному об’єднанні ознак, що покращує як точність, так і швидкість роботи моделі. Візуалізація архітектура YOLOv7 зображена на рис. 2.5.

YOLOv7 також запропонував нові методи масштабування та оптимізації, завдяки чому модель добре підходить для задач детекції малих об’єктів, зокрема дронів, особливо під час обробки відео з високою роздільною здатністю.

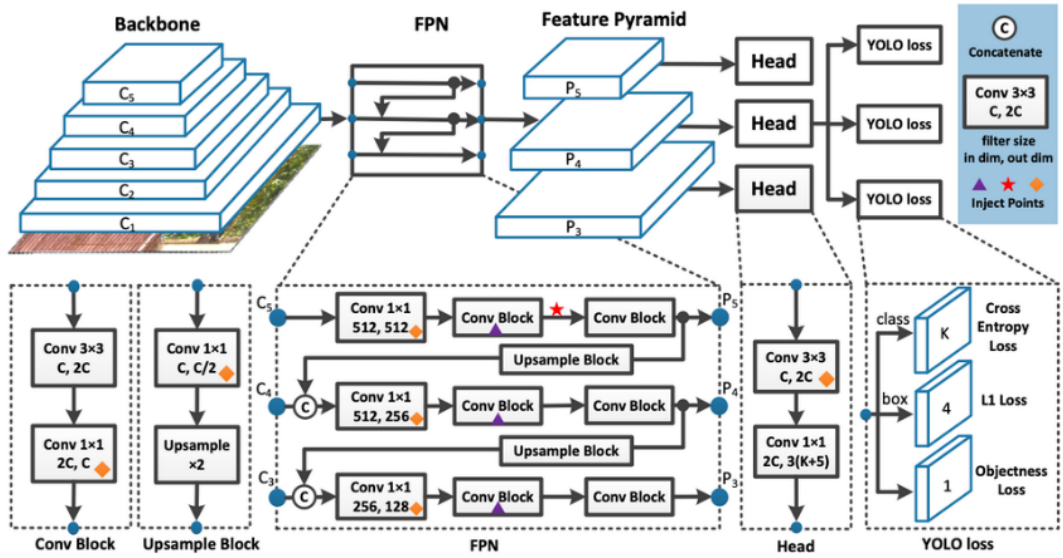


Рис. 2.5 Архітектура YOLOv7

Але вже у 2023 році з'явилася YOLOv8, візуалізація архітектури якої зображено на рис. 2.6, яка кардинально змінила підхід до детекції, повністю відмовившись від прив'язок (anchors). Anchor-free дизайн покращив здатність моделі виявляти дрібні та нерегулярні за формою об'єкти.

У YOLOv8 використовується розділення голови детекції на дві частини: окремо для класифікації та для регресії меж об'єктів, що дозволяє підвищити якість локалізації. Модель також підтримує сучасні техніки аугментацій, включно з copy-paste та mosaic, які особливо ефективні для задачі детекції дронів.

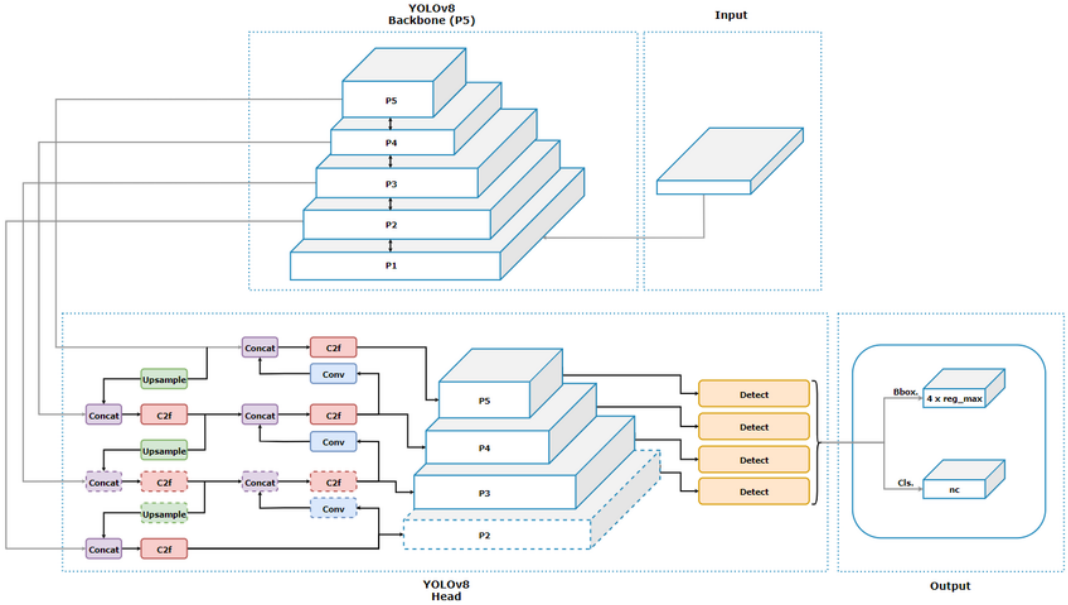


Рис. 2.6 Архітектура YOLOv8

А найновіша версія YOLOv11 (2024) отримала оновлену архітектуру з блоком C3k2 та оптимізованим декодером. Приклад візуалізації архітектури показано на рис. 2.7. Модель демонструє покращену швидкість інференсу, стабільність під час навчання та вищу точність на малих об'єктах. Додаткові техніки, такі як label smoothing, Focal Loss та автоматична змішана точність (AMP), дозволяють покращити роботу на складних датасетах і компенсувати дисбаланс класів, характерний для задачі детекції дронів.

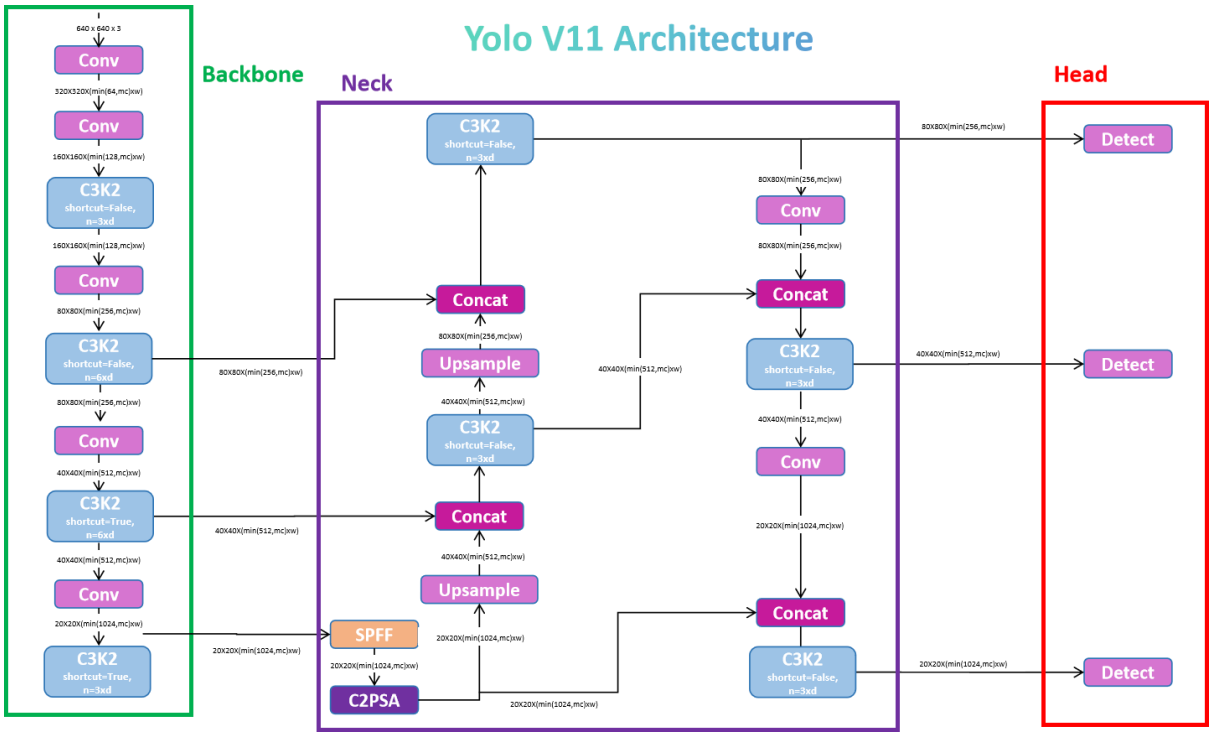


Рис. 2.7 Архітектура YOLOv11

Різні покоління моделей суттєво відрізняються за точністю, продуктивністю та підходами до формування ознак. Порівняльні експерименти на датасеті дронів показують, що YOLOv7 і YOLOv11 зазвичай забезпечують найкращий баланс між швидкістю та точністю, тоді як YOLOv8 дає перевагу в задачах, де критичними є дрібні об'єкти. До уваги беруться такі метрики, як mAP50, mAP50-95, Precision, Recall, швидкість інференсу та кількість параметрів моделі.

2.3 Порівняльний аналіз архітектур SSD, Faster R-CNN, EfficientDet

Окрім розглянутого раніше сімейства YOLO, для задач виявлення об'єктів, зокрема малих безпілотних літальних апаратів (БПЛА), широко застосовуються

інші архітектурні підходи. Кожен з них має унікальний баланс між швидкістю обробки та точністю детекції, що вимагає детального порівняльного аналізу.

Single Shot MultiBox Detector (SSD) представлений у 2016 році, став одним із перших ефективних одноетапних детекторів, що дозволив виконувати детекцію в реальному часі.

На відміну від двоетапних методів, SSD виконує передбачення координат рамок (bounding boxes) та класів об'єктів за один прохід мережі. Архітектура базується на використанні базової мережі (Backbone), наприклад VGG16 або ResNet, яка усікається перед класифікаційними шарами. Замість них додається набір допоміжних згорткових шарів (extra layers), розмір яких поступово зменшується.

Окрім розглянутого раніше сімейства YOLO, для задач виявлення об’єктів, зокрема малих безпілотних літальних апаратів (БПЛА), широко застосовуються інші архітектурні підходи. Кожен з них має унікальний баланс між швидкістю обробки та точністю детекції, що вимагає детального порівняльного аналізу.

Single Shot MultiBox Detector (SSD), представлений у 2016 році, став одним із перших ефективних одноетапних детекторів, що дозволив виконувати детекцію в реальному часі.

На відміну від двоетапних методів, SSD виконує передбачення координат рамок (bounding boxes) та класів об'єктів за один прохід мережі. Архітектура базується на використанні базової мережі (backbone), наприклад VGG16 або ResNet, яка усікається перед класифікаційними шарами. Замість них додається набір допоміжних згорткових шарів (extra layers), розмір яких поступово зменшується.

Ключовою особливістю SSD є використання мультимасштабних карт ознак (multi-scale feature maps). Архітектура SSD з використанням мультимасштабних карт ознак зображена на рис. 2.8. Це дозволяє мережі:

1. Детектувати великі об'єкти на картах ознак меншої розмірності (глибших шарах).
2. Виявляти дрібні об'єкти на картах ознак високої роздільної здатності (початкових шарах).

Для кожного положення на карті ознак генерується набір дефолтних рамок (default boxes або anchors) з різними співвідношеннями сторін (aspect ratios) та масштабами. Функція втрат (Loss Function) у SSD є комбінованою і складається з двох компонентів:

- Localization Loss (Smooth L1): відповідає за точність координат рамки, обчислюється тільки для позитивних збігів.
- Confidence Loss (Cross Entropy): відповідає за класифікацію об'єкта. Для боротьби з дисбалансом класів (перевага фону над об'єктами) використовується техніка Hard Negative Mining, яка відбирає найскладніші негативні приклади для навчання.

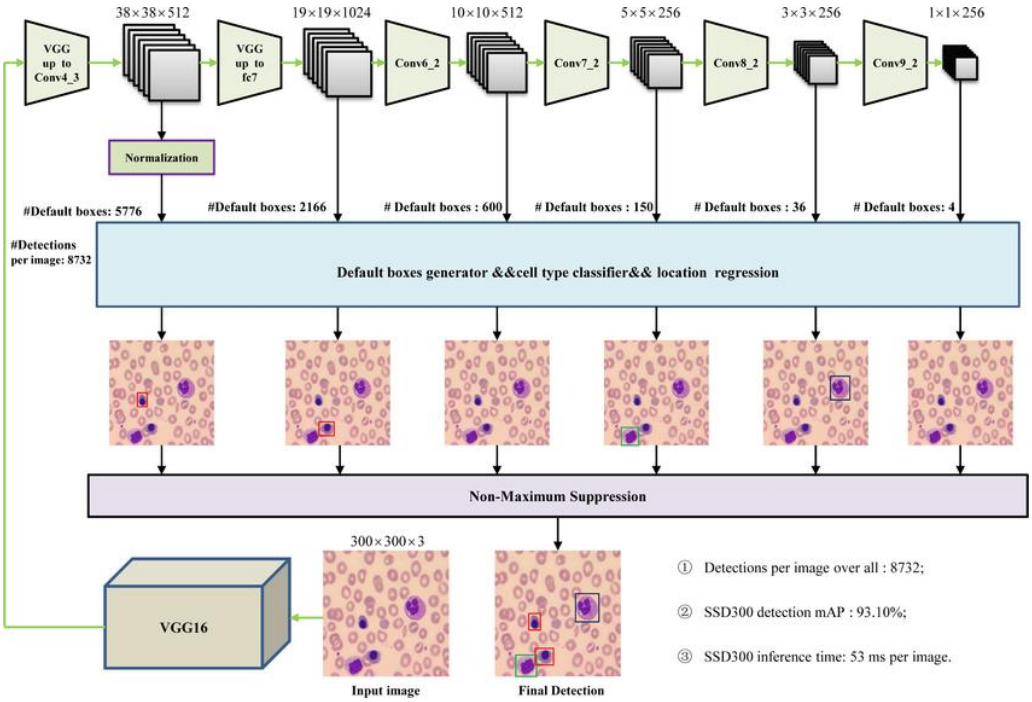


Рис. 2.8 Архітектура SSD з використанням мультимасштабних карт ознак

SSD забезпечує високу швидкість (40–60 FPS на сучасних GPU) завдяки одноетапній структурі. Однак, використання застарілих базових мереж (типу VGG16) та специфіка роботи з дрібними об'єктами призводять до того, що точність SSD на малих цілях (small objects) часто поступається сучасним версіям YOLO та двоетапним детекторам.

Faster R-CNN представляє класичний двоетапний (two-stage) підхід до детекції, який пріоритезує точність над швидкістю.

Процес детекції розділено на два послідовні етапи:

1. Region Proposal Network (RPN): На першому етапі спеціалізована нейромережа генерує пропозиції регіонів (region proposals) — ділянки зображення, де з високою ймовірністю знаходиться об'єкт. Для задач детекції дронів параметри якорів (anchors) в RPN адаптуються під малі розміри цілей (наприклад, розміри 16, 24, 32 пікселів).
2. RoI Pooling та Класифікація: На другому етапі згенеровані регіони вирівнюються за допомогою шару RoI Pooling (або RoI Align) і подаються на вхід класифікатору для визначення класу та уточнення координат.

Сучасні реалізації Faster R-CNN часто використовують ResNet-50 або ResNet-101 у поєднанні з Feature Pyramid Network (FPN) як backbone, що значно покращує здатність моделі "бачити" дрібні деталі.

Аналіз ефективності Faster R-CNN демонструє одну з найвищих точностей (mAP) серед існуючих архітектур, особливо при мінімізації помилкових спрацювань (False Positives). Це критично важливо для систем безпеки. Однак складна двоетапна архітектура накладає суттєві обмеження на швидкість (зазвичай 5–15 FPS), що робить її менш придатною для використання на бортових комп'ютерах дронів з обмеженими ресурсами.

EfficientDet — це сімейство масштабованих детекторів, розроблених Google Research, які використовують концепцію EfficientNet для досягнення оптимального співвідношення точності та обчислювальної ефективності.

Ключові технології:

1. BiFPN (Bidirectional Feature Pyramid Network): На відміну від традиційних FPN, де інформація передається тільки зверху вниз, BiFPN впроваджує двонаправлені з'єднання. Це дозволяє ефективно агрегувати ознаки з різних рівнів роздільної здатності. Важливою особливістю є використання зваженого злиття ознак (weighted feature fusion), де мережа навчається надавати різну вагу вхідним даним залежно від їхньої важливості.
2. Compound Scaling: Замість ручного підбору параметрів для різних версій моделі, EfficientDet використовує коефіцієнт складеного масштабування phi. Цей коефіцієнт одночасно та рівномірно змінює роздільну здатність зображення, глибину та ширину мережі (backbone, BiFPN та prediction

heads). Це дозволяє створювати моделі від D0 (найшвидша) до D7 (найточніша). На рис. 2.9 показано порівняння архітектури BiFPN з традиційними методами злиття ознак.

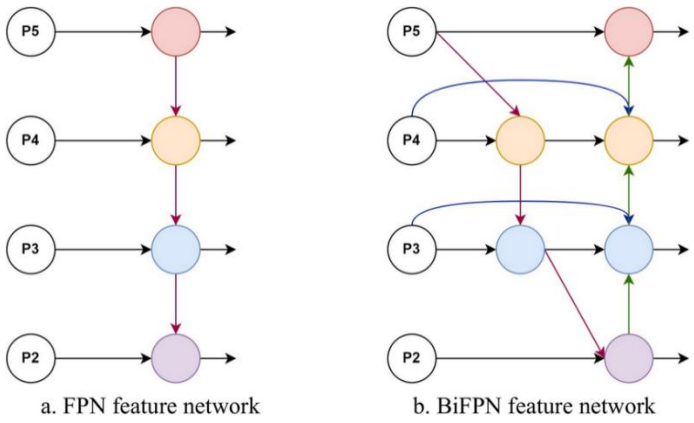


Рис. 2.9 Архітектура BiFPN у порівнянні з традиційними методами злиття ознак

Для задачі детекції дронів ефективним є використання технік аугментації даних (Random Resized Crop, Motion Blur, Gauss Noise), що дозволяє моделі EfficientDet краще узагальнювати ознаки об'єктів у складних погодних умовах.

2.4 Методи оптимізації нейронних мереж для роботи в реальному часі

Досягнення роботи в реальному часі (25+ FPS) при збереженні високої точності детекції є критичним завданням для системи виявлення дронів. Існує широкий спектр методів оптимізації, що дозволяють прискорити інференс моделі без значної втрати якості.

Один з таких методів – квантизація. На рис. 2.10 показано порівняння впливу різних типів квантизації на швидкість та точність Це процес зменшення точності представлення ваг та активацій нейронної мережі для зменшення розміру моделі та прискорення обчислень. Стандартні моделі використовують 32-бітні числа з плаваючою комою (FP32), тоді як квантизовані моделі можуть працювати з 16-бітними (FP16), 8-бітними (INT8) або навіть бінарними представленнями.

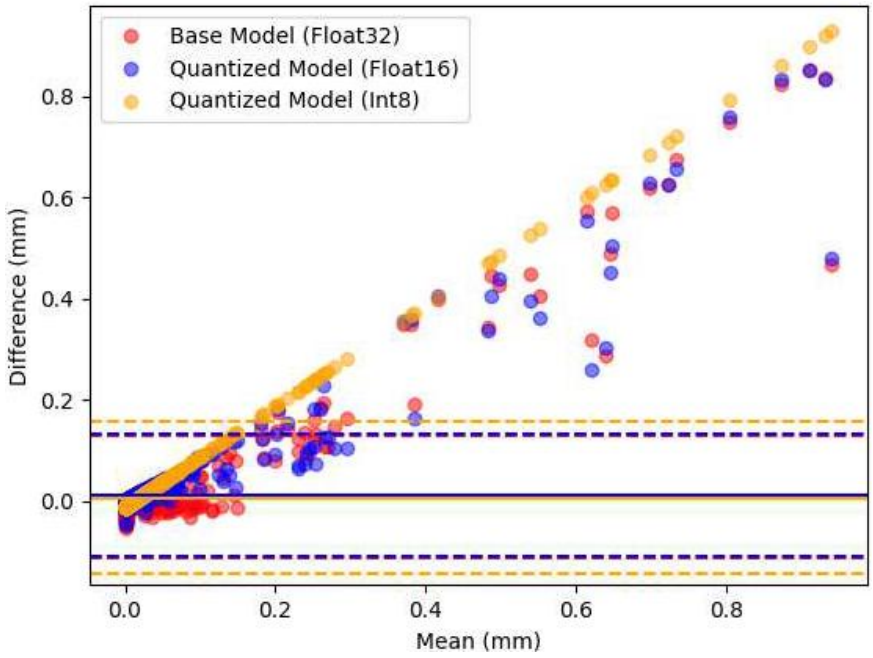


Рис. 2.10 Вплив різних типів квантизації на швидкість та точність

Не менш важливим методом оптимізації є обрізка (Pruning) нейронних мереж. Pruning видаляє менш важливі нейронні з'єднання або цілі фільтри, зменшуючи розмір моделі та обчислювальні вимоги. Сучасні дослідження показують, що типові глибокі мережі містять значну надмірність - до 70-90% параметрів можуть бути видалені з мінімальним впливом на точність.

Prunning поділяється на два типи: Structured pruning та Unstructured pruning. На рис. 2.11 зображено визуалізації типів pruning. Перший у свою чергу видаляє цілі канали або шари, що забезпечує реальне прискорення на стандартному обладнанні. Наприклад, видалення 40% каналів з backbone мережі може зменшити час інференсу на 35-40% при зниженні mAP50 лише на 2-3%, а другий видаляє окремі ваги, створюючи розріджені матриці. Цей підхід може видалити до 90% параметрів, але потребує спеціалізованого апаратного забезпечення або бібліотек для ефективного виконання розріджених операцій.

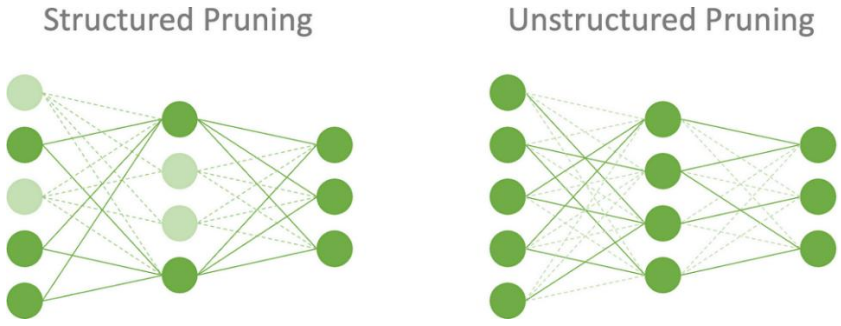


Рис. 2.11 Візуалізація structured та unstructured pruning

Для системи детекції дронів оптимальним є structured pruning з коефіцієнтом 30-40%, що забезпечує баланс між швидкістю та точністю. Критично важливо зберігати більшу кількість параметрів у detection head, оскільки він безпосередньо відповідає за точну локалізацію малих об'єктів.

А для зменшення кількість параметрів та обчислень при мінімальній втраті якості екстракції ознак використовується Depth-wise separable convolutions. Порівняння стандартної та depth-wise separable згортки зображена на рис. 2.12. Він розділяє стандартну згортку на два етапи: depth-wise (окремо для кожного каналу) та point-wise (1×1 згортка для комбінування каналів), що дозволяє оптимізувати архітектуру моделі у 8-9 разів. MobileNet та EfficientNet активно використовують цю техніку.

До прикладу Group convolutions обробляють вхідні канали групами, зменшуючи обчислювальну складність. Наприклад, використання 4 груп зменшує кількість операцій у 4 рази. Проте занадто велика кількість груп може погіршити потік інформації між каналами.

А Attention mechanisms дозволяють мережі фокусуватися на важливих регіонах, підвищуючи ефективність без додавання значної кількості параметрів. Для детекції дронів особливо корисні spatial attention модулі, що посилюють відгук на малі об'єкти на фоні неба.

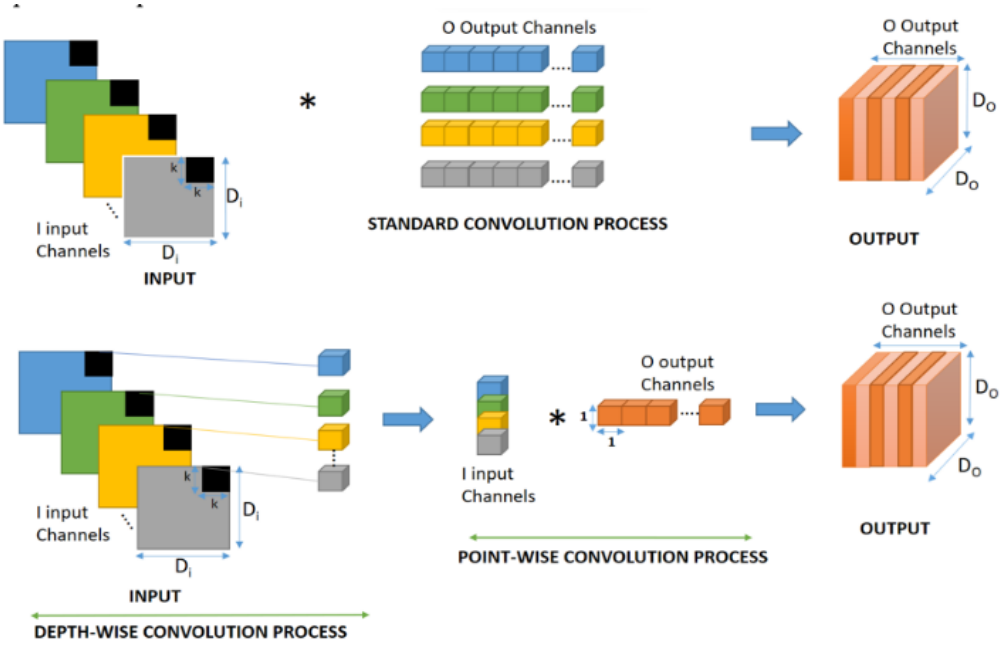


Рис. 2.12 Порівняння стандартної та depth-wise separable згортки

Однак одним з ключових етапів оптимізації – є оптимізація на рівні фреймворку, бо сучасні фреймворки глибокого навчання надають інструменти для автоматичної оптимізації моделей під конкретне апаратне забезпечення.

Наприклад TensorRT від NVIDIA оптимізує моделі для GPU, застосовуючи kernel fusion (об'єднання послідовних операцій), precision calibration (автоматичний вибір точності для кожного шару) та layer fusion (об'єднання шарів). Для YOLOv8 на RTX 3060 TensorRT забезпечує прискорення у 2.5-3 рази порівняно з PyTorch.

А ONNX Runtime надає кросплатформенну оптимізацію та підтримує різне апаратне забезпечення (CPU, GPU, NPU). Конвертація моделі в ONNX формат та використання ONNX Runtime може прискорити інференс на CPU у 3-5 разів.
Тим часом, як OpenVINO від Intel оптимізує моделі для процесорів Intel та інтегрованої графіки. Це особливо важливо для граничних пристроїв без дискретних GPU.

Automatic Mixed Precision (AMP) автоматично використовує FP16 для операцій, де це можливо без втрати точності, зберігаючи FP32 для критичних обчислень. Це забезпечує прискорення навчання у 2-3 рази та інференсу у 1.5-2 рази на сучасних GPU з Tensor Cores.

Простий приклад використання:

```
from torch.cuda.amp import autocast, GradScaler

scaler = GradScaler()
for images, targets in dataloader:
    with autocast():
        predictions = model(images)
        loss = compute_loss(predictions, targets)
    scaler.scale(loss).backward()
    scaler.step(optimizer)
    scaler.update()
```

Але швидкість системи залежить не лише від моделі, але й від ефективності всього pipeline обробки відео.

Асинхронна обробка дозволяє паралельно виконувати захоплення кадру, препроцесинг, інференс та постпроцесинг. Використання багатопотоковості або

асинхронного програмування може збільшити загальну пропускну здатність на 30-50%.

Батчування кадрів - обробка декількох кадрів одночасно підвищує утилізацію GPU. Проте для real-time систем розмір батчу обмежений вимогами до затримки. Оптимальний баланс для детекції дронів - батч розміром 2-4 кадри.

Адаптивна роздільна здатність динамічно змінює роздільність обробки залежно від складності сцени. Якщо дронів немає або вони великі, можна знизити роздільність до 416×416, а при виявленні малих цілей збільшити до 1280×1280.

2.5 Обґрунтування вибору архітектури для системи розпізнавання дронів

На основі проведеного аналізу сучасних архітектур детекції об'єктів та методів їх оптимізації необхідно обґрунтувати вибір конкретної архітектури для розробки системи розпізнавання дронів у реальному часі, адже система детекції дронів повинна задовольняти наступним вимогам (таблиця 2.2):

Швидкість обробки: мінімум 25-30 FPS для забезпечення плавного трекінгу швидкорухомих об'єктів. Оптимально - 40-60 FPS для надійного супроводу та швидкої реакції на загрозу.

Точність детекції малих об'єктів: дрони на відстані 300-500 метрів займають лише 20-50 пікселів на зображенні. Модель повинна надійно виявляти такі об'єкти з mAP50 не нижче 0.85.

Низький рівень false positives: хибні спрацювання на птахів, літаки або інші об'єкти знижують довіру до системи. Precision має бути не нижче 0.87.

Високий recall: пропуск реального дрона має катастрофічні наслідки. Recall повинен перевищувати 0.85.

Можливість роботи на доступному обладнанні: система має працювати на GPU середнього рівня (RTX 3060, RTX 4060) або граничних пристроях (Jetson Xavier).

Простота розгортання та підтримки: наявність добре документованих інструментів, активної спільноти та регулярних оновлень.

Таблиця 2.1

Зведена порівняльна таблиця архітектур за ключовими критеріями

Критерій оцінки	Цільовий показник / Вимога
Швидкість обробки (FPS)	Мінімум: 25-30 FPS Оптимально: 40-60 FPS
Точність (Small Objects)	$mAP_{50} \geq 0.85$
Precision (Точність)	≥ 0.87
Recall (Повнота)	≥ 0.85
Вимоги до апаратного забезпечення	GPU середнього рівня (RTX 3060, RTX 4060) або Edge-пристрої (Jetson Xavier)
Експлуатація та підтримка	Високий рівень документованості та активна спільнота

Не дивлячись на те, що YOLOv5 залишається популярним вибором завдяки зрілості екосистеми, великій кількості туторіалів та стабільності, все ж архітектура дещо застаріла порівняно з новішими версіями. На датасеті дронів YOLOv5m демонструє mAP_{50} = 0.88 при 52 FPS на RTX 3060, що є хорошим, але не оптимальним результатом.

Хоча YOLOv7 показує покращену точність завдяки E-ELAN архітектурі, досягаючи mAP50 = 0.90 при 48 FPS. Проте підтримка та документація менш розвинені порівняно з YOLOv5 та YOLOv8, а навчання потребує більше часу через складнішу архітектуру.

Але YOLOv8 представляє найкращий баланс між інноваціями та зручністю використання. Anchor-free підхід особливо ефективний для детекції малих об'єктів, що критично важливо для дронів. YOLOv8m досягає mAP_{50} = 0.91 при 45 FPS, а YOLOv8s - 0.88 при 68 FPS. Ultralytics надає відмінну документацію, регулярні оновлення та зручний API.

Не дивлячись, що YOLOv11 - найновіша версія з покращеною C3k2 архітектурою, демонструє mAP_{50} = 0.93 при 42 FPS. Проте версія відносно нова

(2024), що означає менше протестованих use cases та можливі нестабільності.

В свою чергу Faster R-CNN забезпечує найвищу точність (mAP_{50} = 0.95) та найкращу локалізацію, але швидкість лише 12 FPS робить цю архітектуру непридатною для real-time застосувань.

SSD нажаль поступається сучасним версіям YOLO як за точністю (mAP_{50} = 0.82), так і за швидкістю (38 FPS), через що не розглядається як оптимальний варіант.

У свою чергу EfficientDet демонструє хороші результати (mAP_{50} = 0.90 при 35 FPS), але складність налаштування, навчання та інтеграції перевищує переваги порівняно з YOLO. На рис. 2.13 зображено графік співвідношення точності та швидкості для різних архітектур.

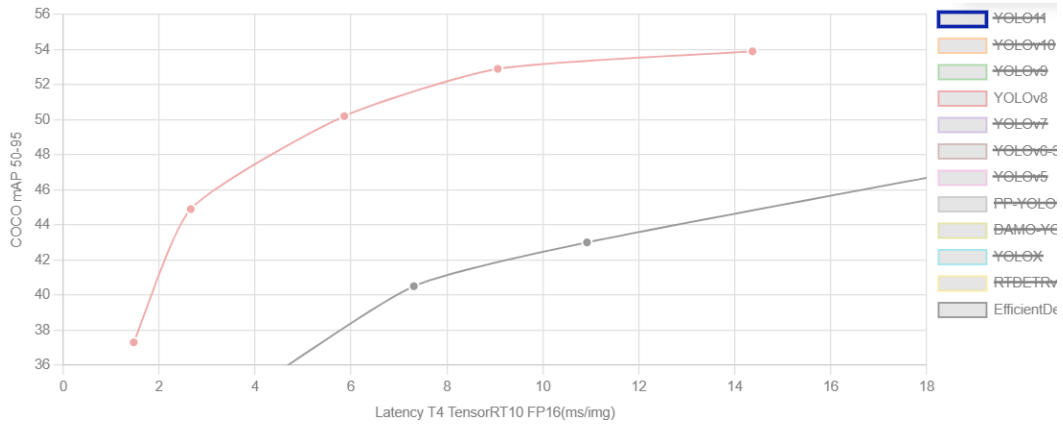


Рис. 2.13 Графік співвідношення точності та швидкості для різних архітектур

На основі комплексного аналізу для розробки системи розпізнавання дронів обрано архітектуру YOLOv8 з наступних причин:

Оптимальний баланс швидкості та точності. YOLOv8m забезпечує mAP_{50} = 0.91 при 45 FPS на RTX 3060, що повністю задовольняє вимоги до real-time обробки з високою якістю детекції. Для застосувань, де критична швидкість, YOLOv8s досягає 68 FPS з незначним зниженням точності до 0.88.

Anchor-free дизайн особливо ефективний для детекції малих об'єктів. Традиційні anchor-based детектори вимагають ретельного підбору розмірів та співвідношень сторін anchors, що складно для різноманітних типів дронів. Anchor-free підхід YOLOv8 автоматично адаптується до різних форм та розмірів об'єктів.

Розвинена екосистема Ultralytics надає всі необхідні інструменти: зручний

API для навчання та інференсу, вбудовану підтримку різних форматів датасетів, автоматичні hyperparameter tuning, інтеграцію з популярними фреймворками відстеження об'єктів, експорт у різні формати (ONNX, TensorRT, CoreML).

Ефективність навчання. YOLOv8 сходиться швидше за попередні версії завдяки покращеній функції втрат та стратегії аугментації. Типово, для досягнення $mAP_{50} > 0.90$ на датасеті дронів достатньо 100-150 епох, що становить 8-12 годин на одному GPU RTX 3060.

Гнучкість масштабування. Сімейство YOLOv8 включає 5 варіантів моделей (n, s, m, l, x), що дозволяє вибрати оптимальну конфігурацію залежно від апаратного забезпечення: YOLOv8n для Jetson Nano (18 FPS), YOLOv8s для Jetson Xavier (35 FPS), YOLOv8m для настільних систем (45 FPS), YOLOv8l/x для серверних застосувань, де критична максимальна точність.

Активна підтримка та оновлення. Ultralytics регулярно випускає оновлення, виправляє баги та додає нові функції. Велика спільнота користувачів означає швидку допомогу при виникненні проблем та доступність готових рішень для типових задач.

Доведена ефективність на схожих задачах. YOLOv8 успішно застосовується в проєктах детекції малих об'єктів (дронів, птахів, транспорту з висоти) та демонструє кращі результати порівняно з попередніми версіями.

РОЗДІЛ 3 РОЗРОБКА, НАВЧАННЯ ТА ТЕСТУВАННЯ МОДЕЛІ РОЗПІЗНАВАННЯ ДРОНІВ У РЕАЛЬНОМУ ЧАСІ

3.1 Формування та розмітка датасету для навчання моделі

Якість та обсяг навчальних даних є визначальними факторами успіху будь-якої системи машинного навчання. Для розробки ефективної системи розпізнавання дронів необхідно сформувати репрезентативний датасет, що охоплює різноманітні сценарії використання та умови спостереження.

Для навчання моделі було використано комбінований підхід, що об'єднує дані з публічних джерел.

Загальна структура датасету:

```
drone_detection_dataset/
├── images/
│   ├── train/          # 12,500 зображень
│   ├── val/            # 2,500 зображень
│   └── test/           # 3,000 зображень
├── labels/
│   ├── train/          # YOLO формат анотацій
│   ├── val/
│   └── test/
├── data.yaml           # Конфігураційний файл
└── README.md
```

Для розмітки датасету використовувався інструмент CVAT (Computer Vision Annotation Tool), який забезпечує зручний інтерфейс та підтримку різних форматів анотацій.

Протокол розмітки:

1. Визначення bounding box: Прямокутник має щільно охоплювати видиму частину дрона, включаючи корпус та пропелери
2. Класифікація: Кожен об'єкт отримує мітку класу:
    - drone_quadcopter: квадрокоптери (4 пропелери)
    - drone_hexacopter: гексакоптери (6 пропелерів)
    - drone_fixed_wing: дрони літакового типу
    - bird: птахи (для навчання диференціації)
3. Атрибути складності:
    - occlusion: рівень перекриття (0-100%)

- size: small (<50px), medium (50-150px), large (>150px)
- visibility: poor, moderate, good

Приклади розмічених зображень з різних умов зйомки наведено на рис. 3.1.

Приклад анотації у форматі YOLO:

```
# drone_001.txt
# class x_center y_center width height
0 0.512 0.345 0.082 0.095
```

Для забезпечення високої якості розмітки було впроваджено триетапний процес верифікації, який починается з автоматичної перевірки:

```
import os
import cv2
import numpy as np

def validate_annotations(image_dir, label_dir):
    """
    Перевірка коректності анотацій
    """
    errors = []

    for label_file in os.listdir(label_dir):
        image_file = label_file.replace('.txt', '.jpg')
        image_path = os.path.join(image_dir, image_file)
        label_path = os.path.join(label_dir, label_file)

        # Перевірка існування відповідного зображення
        if not os.path.exists(image_path):
            errors.append(f"Missing image for {label_file}")
            continue

        # Завантаження зображення
        image = cv2.imread(image_path)
        h, w = image.shape[:2]

        # Читання анотацій
        with open(label_path, 'r') as f:
            for line_num, line in enumerate(f.readlines()):
                try:
                    class_id, x_center, y_center, width, height = map(float,
line.strip().split())

                    # Перевірка діапазону координат
                    if not (0 <= x_center <= 1 and 0 <= y_center <= 1):
                        errors.append(f"{label_file}:{line_num} - coordinates out
of range")

                    # Перевірка розмірів bbox
                    if not (0 < width <= 1 and 0 < height <= 1):
```

```
                        errors.append(f"{label_file}:{line_num} - invalid bbox
size")

                    # Перевірка мінімального розміру (>10 пікселів)
                    if width * w < 10 or height * h < 10:
                        errors.append(f"{label_file}:{line_num} - bbox too small")

                except ValueError:
                    errors.append(f"{label_file}:{line_num} - invalid format")

    return errors

# Виконання перевірки
errors = validate_annotations('images/train', 'labels/train')
print(f"Found {len(errors)} errors")
for error in errors[:10]:  # Показати перші 10 помилок
    print(error)
```

Після чого відбувається візуальна інспекція, коли випадкова вибірка 10% даних перевірялася вручну різними анотаторами для виявлення систематичних помилок:

```
import random
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle

def visualize_annotations(image_path, label_path, num_samples=10):
    """
    Візуалізація анотацій для перевірки
    """
    image_files = random.sample(os.listdir(image_path), num_samples)

    fig, axes = plt.subplots(2, 5, figsize=(20, 8))
    axes = axes.flatten()

    class_names = ['quadcopter', 'hexacopter', 'fixed_wing', 'bird']
    colors = ['red', 'green', 'blue', 'yellow']

    for idx, image_file in enumerate(image_files):
        # Завантаження зображення
        img = cv2.imread(os.path.join(image_path, image_file))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        h, w = img.shape[:2]

        # Читання анотацій
        label_file = image_file.replace('.jpg', '.txt')
        with open(os.path.join(label_path, label_file), 'r') as f:
            annotations = f.readlines()

        axes[idx].imshow(img)
```

```
        # Малювання bounding boxes
        for ann in annotations:
            class_id, x_center, y_center, width, height = map(float,
ann.strip().split())

            # Конвертація в абсолютні координати
            x1 = int((x_center - width/2) * w)
            y1 = int((y_center - height/2) * h)
            box_w = int(width * w)
            box_h = int(height * h)

            rect = Rectangle((x1, y1), box_w, box_h,
                             linewidth=2, edgecolor=colors[int(class_id)],
                             facecolor='none')
            axes[idx].add_patch(rect)
            axes[idx].text(x1, y1-5, class_names[int(class_id)],
                           color=colors[int(class_id)], fontsize=8,
                           bbox=dict(boxstyle='round', facecolor='white',
alpha=0.7))

        axes[idx].axis('off')
        axes[idx].set_title(f"{image_file}", fontsize=8)

    plt.tight_layout()
    plt.savefig('annotation_samples.png', dpi=150, bbox_inches='tight')
    plt.show()

# Візуалізація зразків
visualize_annotations('images/train', 'labels/train', num_samples=10)
```

Рис. 3.1 Приклади розмічених зображень з різних умовах

І фінальним кроком оцінюємо узгодженості розмітки використовуючи коефіцієнт IoU (Intersection over Union) між анотаціями різних експертів:

```python
def calculate_iou(box1, box2):
    """
    Обчислення IoU між двома bounding boxes
    box format: [x_center, y_center, width, height]
    """
    # Конвертація у формат [x1, y1, x2, y2]
    box1_x1 = box1[0] - box1[2]/2
    box1_y1 = box1[1] - box1[3]/2
    box1_x2 = box1[0] + box1[2]/2
    box1_y2 = box1[1] + box1[3]/2

    box2_x1 = box2[0] - box2[2]/2
    box2_y1 = box2[1] - box2[3]/2
    box2_x2 = box2[0] + box2[2]/2
    box2_y2 = box2[1] + box2[3]/2

    # Площа перетину
    inter_x1 = max(box1_x1, box2_x1)
    inter_y1 = max(box1_y1, box2_y1)
    inter_x2 = min(box1_x2, box2_x2)
    inter_y2 = min(box1_y2, box2_y2)

    if inter_x2 < inter_x1 or inter_y2 < inter_y1:
        return 0.0

    inter_area = (inter_x2 - inter_x1) * (inter_y2 - inter_y1)

    # Площа об'єднання
    box1_area = box1[2] * box1[3]
    box2_area = box2[2] * box2[3]
    union_area = box1_area + box2_area - inter_area

    return inter_area / union_area if union_area > 0 else 0.0

def inter_annotator_agreement(annotations_dir1, annotations_dir2):
    """
    Оцінка узгодженості між двома наборами анотацій
    """
    iou_scores = []

    for label_file in os.listdir(annotations_dir1):
        path1 = os.path.join(annotations_dir1, label_file)
        path2 = os.path.join(annotations_dir2, label_file)

        if not os.path.exists(path2):
            continue

        with open(path1, 'r') as f1, open(path2, 'r') as f2:
            boxes1 = [[float(x) for x in line.strip().split()[1:]] for line in
f1.readlines()]
            boxes2 = [[float(x) for x in line.strip().split()[1:]] for line in
f2.readlines()]
```

```
        # Порівняння кожного боксу з першого набору з найближчим з другого
        for box1 in boxes1:
            max_iou = max([calculate_iou(box1, box2) for box2 in boxes2],
default=0)
            iou_scores.append(max_iou)

    mean_iou = np.mean(iou_scores)
    agreement_rate = np.sum(np.array(iou_scores) > 0.7) / len(iou_scores)

    print(f"Mean IoU: {mean_iou:.3f}")
    print(f"Agreement rate (IoU > 0.7): {agreement_rate:.1%}")

    return iou_scores

# Оцінка узгодженості
iou_scores = inter_annotator_agreement('annotations/annotator1',
'annotations/annotator2')
```

Та по результатам верифікації отримаємо середній IoU = 0.91 між анотаторами, що свідчить про високу якість розмітки.

Наступим кроком для розуміння характеристик сформованого датасету було проведено детальний статистичний аналіз за допомогою кода:

```
import pandas as pd
import seaborn as sns

def analyze_dataset(image_dir, label_dir):
    """
    Статистичний аналіз датасету
    """
    stats = {
        'class': [],
        'x_center': [],
        'y_center': [],
        'width': [],
        'height': [],
        'area': [],
        'aspect_ratio': []
    }

    class_names = ['quadcopter', 'hexacopter', 'fixed_wing', 'bird']

    for label_file in os.listdir(label_dir):
        image_file = label_file.replace('.txt', '.jpg')
        image_path = os.path.join(image_dir, image_file)

        # Отримання розмірів зображення
        img = cv2.imread(image_path)
        h, w = img.shape[:2]

        with open(os.path.join(label_dir, label_file), 'r') as f:
            for line in f.readlines():
```

```
                class_id, x_center, y_center, width, height = map(float,
line.strip().split())

                # Абсолютні розміри в пікселях
                abs_width = width * w
                abs_height = height * h

                stats['class'].append(class_names[int(class_id)])
                stats['x_center'].append(x_center)
                stats['y_center'].append(y_center)
                stats['width'].append(abs_width)
                stats['height'].append(abs_height)
                stats['area'].append(abs_width * abs_height)
                stats['aspect_ratio'].append(abs_width / abs_height if abs_height >
0 else 0)

    df = pd.DataFrame(stats)

    # Візуалізація розподілів
    fig, axes = plt.subplots(2, 3, figsize=(18, 12))

    # Розподіл класів
    df['class'].value_counts().plot(kind='bar', ax=axes[0, 0], color='steelblue')
    axes[0, 0].set_title('Розподіл за класами')
    axes[0, 0].set_ylabel('Кількість')

    # Розподіл розмірів
    axes[0, 1].hist(df['width'], bins=50, alpha=0.7, label='Width', color='green')
    axes[0, 1].hist(df['height'], bins=50, alpha=0.7, label='Height',
color='orange')
    axes[0, 1].set_title('Розподіл розмірів bbox (пікселі)')
    axes[0, 1].set_xlabel('Розмір (px)')
    axes[0, 1].legend()

    # Розподіл площ
    axes[0, 2].hist(df['area'], bins=50, color='coral')
    axes[0, 2].set_title('Розподіл площ bbox')
    axes[0, 2].set_xlabel('Площа (px²)')
    axes[0, 2].set_yscale('log')

    # Spatial distribution
    axes[1, 0].scatter(df['x_center'], df['y_center'], alpha=0.3, s=5)
    axes[1, 0].set_title('Просторовий розподіл об\'єктів')
    axes[1, 0].set_xlabel('X координата (normalized)')
    axes[1, 0].set_ylabel('Y координата (normalized)')
    axes[1, 0].set_xlim(0, 1)
    axes[1, 0].set_ylim(0, 1)
    axes[1, 0].invert_yaxis()

    # Aspect ratio
    axes[1, 1].hist(df['aspect_ratio'], bins=50, color='purple')
    axes[1, 1].set_title('Розподіл співвідношень сторін')
    axes[1, 1].set_xlabel('Width / Height')
```

```
    # Boxplot за класами
    df.boxplot(column='area', by='class', ax=axes[1, 2])
    axes[1, 2].set_title('Площа bbox за класами')
    axes[1, 2].set_ylabel('Площа (px²)')

    plt.suptitle('')
    plt.tight_layout()
    plt.savefig('dataset_statistics.png', dpi=150, bbox_inches='tight')
    plt.show()

    # Виведення описової статистики
    print("\n=== Описова статистика датасету ===\n")
    print(df.groupby('class').agg({
        'width': ['mean', 'std', 'min', 'max'],
        'height': ['mean', 'std', 'min', 'max'],
        'area': ['mean', 'std', 'min', 'max']
    }).round(2))

    # Категоризація за розміром
    df['size_category'] = pd.cut(df['area'],
                                 bins=[0, 2500, 10000, float('inf')],
                                 labels=['small', 'medium', 'large'])

    print("\n=== Розподіл за розміром ===\n")
    print(df.groupby(['class', 'size_category']).size().unstack(fill_value=0))

    return df

# Виконання аналізу
df_stats = analyze_dataset('images/train', 'labels/train')
```

Основні статистичні характеристики об’єктів у сформованому датасеті наведено у табл. 3.1.

Таблиця 3.1

Характеристики об'єктів у датасеті

Метрика	Quadcopter	Hexacopter	Fixed Wing	Bird	
Кількість зразків	8,450	2,120	1,380	2,050	
Середній розмір (px)	65×58	78×72	95×45	42×38	
Мін. розмір (px)	18×16	24×22	32×18	15×12	
Макс. розмір (px)	285×268	312×295	456×198	125×118	

Далі для початку навчання YOLOv8 необхідно створити конфігураційний файл у форматі YAML:

```yaml
# data.yaml

# Шляхи до даних
path: /content/drone_detection_dataset  # корінева директорія
train: images/train  # шлях до train images (відносно 'path')
val: images/val      # шлях до val images (відносно 'path')
test: images/test    # шлях до test images (відносно 'path')

# Класи
names:
  0: quadcopter
  1: hexacopter
  2: fixed_wing
  3: bird

# Кількість класів
nc: 4

# Додаткова інформація
dataset_description: "Датасет для детекції дронів у реальному часі"
version: "1.0"
created: "2025-11-15"
```

Сформований датасет становить міцну основу для навчання моделі детекції дронів та забезпечуючи високу якість розмітки.

3.2 Методи аугментації даних та попередня обробка зображень

Аугментація даних є критично важливою для підвищення узагальнюючої здатності моделі та збільшення ефективного обсягу навчального датасету. Для задачі детекції дронів застосовувався комплексний підхід, що поєднує геометричні, фотометричні та спеціалізовані трансформації. Приклади застосування різних типів аугментації зображень наведено на рис. 3.2.

Геометричні перетворення змінюють просторове розташування об'єктів, навчаючи модель інваріантності до позиції та орієнтації:

```python
import albumentations as A
from albumentations.pytorch import ToTensorV2
import cv2
import numpy as np
```

```python
class DroneAugmentation:
    """
    Клас для аугментації зображень дронів
    """
    def __init__(self, image_size=640, mode='train'):
        self.image_size = image_size
        self.mode = mode

        if mode == 'train':
            self.transform = A.Compose([
                # Геометричні трансформації
                A.HorizontalFlip(p=0.5),
                A.VerticalFlip(p=0.2),
                A.RandomRotate90(p=0.3),
                A.ShiftScaleRotate(
                    shift_limit=0.1,
                    scale_limit=0.2,
                    rotate_limit=15,
                    border_mode=cv2.BORDER_CONSTANT,
                    value=0,
                    p=0.7
                ),
                A.Affine(
                    scale=(0.8, 1.2),
                    translate_percent={'x': (-0.1, 0.1), 'y': (-0.1, 0.1)},
                    rotate=(-15, 15),
                    shear=(-5, 5),
                    p=0.5
                ),
                A.Perspective(
                    scale=(0.05, 0.1),
                    p=0.3
                ),

                # Фотометричні трансформації
                A.RandomBrightnessContrast(
                    brightness_limit=0.2,
                    contrast_limit=0.2,
                    p=0.6
                ),
                A.HueSaturationValue(
                    hue_shift_limit=15,
                    sat_shift_limit=25,
                    val_shift_limit=15,
                    p=0.5
                ),
                A.RandomGamma(
                    gamma_limit=(80, 120),
                    p=0.4
                ),
                A.CLAHE(
                    clip_limit=2.0,
                    tile_grid_size=(8, 8),
```

```
            p=0.3
        ),

        # Шум та розмиття
        A.GaussNoise(
            var_limit=(10.0, 50.0),
            p=0.3
        ),
        A.GaussianBlur(
            blur_limit=(3, 7),
            p=0.2
        ),
        A.MotionBlur(
            blur_limit=7,
            p=0.3
        ),
        A.MedianBlur(
            blur_limit=5,
            p=0.1
        ),

        # Погодні умови
        A.RandomFog(
            fog_coef_lower=0.1,
            fog_coef_upper=0.3,
            alpha_coef=0.1,
            p=0.2
        ),
        A.RandomRain(
            slant_lower=-10,
            slant_upper=10,
            drop_length=20,
            drop_width=1,
            drop_color=(200, 200, 200),
            blur_value=3,
            brightness_coefficient=0.9,
            rain_type='drizzle',
            p=0.15
        ),
        A.RandomSunFlare(
            flare_roi=(0, 0, 1, 0.5),
            angle_lower=0,
            angle_upper=1,
            num_flare_circles_lower=1,
            num_flare_circles_upper=2,
            src_radius=200,
            p=0.1
        ),

        # Compression artifacts
        A.ImageCompression(
            quality_lower=75,
            quality_upper=100,
```

```
                    p=0.3
                ),

                # Нормалізація та зміна розміру
                A.Resize(image_size, image_size),
                A.Normalize(
                    mean=[0.485, 0.456, 0.406],
                    std=[0.229, 0.224, 0.225]
                ),
                ToTensorV2()
            ], bbox_params=A.BboxParams(
                format='yolo',
                label_fields=['class_labels'],
                min_visibility=0.3  # Видаляти bbox якщо <30% видимі
            ))
        else:  # validation/test
            self.transform = A.Compose([
                A.Resize(image_size, image_size),
                A.Normalize(
                    mean=[0.485, 0.456, 0.406],
                    std=[0.229, 0.224, 0.225]
                ),
                ToTensorV2()
            ], bbox_params=A.BboxParams(
                format='yolo',
                label_fields=['class_labels']
            ))

    def __call__(self, image, bboxes, class_labels):
        """
        Застосування трансформацій
        """
        transformed = self.transform(
            image=image,
            bboxes=bboxes,
            class_labels=class_labels
        )

        return transformed['image'], transformed['bboxes'],
transformed['class_labels']

# Приклад використання
def augment_and_visualize(image_path, label_path, num_augmentations=6):
    """
    Демонстрація аугментацій
    """
    # Завантаження зображення та анотацій
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    h, w = image.shape[:2]

    bboxes = []
    class_labels = []
```

```
    with open(label_path, 'r') as f:
        for line in f.readlines():
            class_id, x_center, y_center, width, height = map(float,
line.strip().split())
            bboxes.append([x_center, y_center, width, height])
            class_labels.append(int(class_id))

    # Створення augmentor
    augmentor = DroneAugmentation(image_size=640, mode='train')

    # Генерація аугментованих версій
    fig, axes = plt.subplots(2, 3, figsize=(18, 12))
    axes = axes.flatten()

    class_names = ['Quadcopter', 'Hexacopter', 'Fixed Wing', 'Bird']
    colors = ['red', 'green', 'blue', 'yellow']

    for idx in range(num_augmentations):
        # Застосування аугментації
        aug_image, aug_bboxes, aug_labels = augmentor(
            image.copy(),
            bboxes.copy(),
            class_labels.copy()
        )

        # Конвертація тензору назад у numpy для візуалізації
        aug_image_np = aug_image.permute(1, 2, 0).numpy()
        # Денормалізація
        aug_image_np = aug_image_np * np.array([0.229, 0.224, 0.225]) +
np.array([0.485, 0.456, 0.406])
        aug_image_np = np.clip(aug_image_np, 0, 1)

        axes[idx].imshow(aug_image_np)

        # Малювання bbox
        for bbox, label in zip(aug_bboxes, aug_labels):
            x_center, y_center, width, height = bbox
            x1 = int((x_center - width/2) * 640)
            y1 = int((y_center - height/2) * 640)
            x2 = int((x_center + width/2) * 640)
            y2 = int((y_center + height/2) * 640)

            rect = Rectangle((x1, y1), x2-x1, y2-y1,
                             linewidth=2, edgecolor=colors[label],
                             facecolor='none')
            axes[idx].add_patch(rect)
            axes[idx].text(x1, y1-5, class_names[label],
                           color=colors[label], fontsize=8,
                           bbox=dict(boxstyle='round', facecolor='white',
alpha=0.7))

        axes[idx].axis('off')
        axes[idx].set_title(f'Augmentation #{idx+1}', fontsize=10)
```

```
plt.tight_layout()
plt.savefig('augmentation_examples.png', dpi=150, bbox_inches='tight')
plt.show()
```

Рис. 3.2 Приклади застосування різних типів аугментацій

Для оцінки впливу різних типів аугментації було проведено експеримент з поступовим додаванням технік. Результати експерименту з оцінки впливу аугментації на якість детекції наведено у табл. 3.2.

Таблиця 3.2

Вплив аугментації на якість детекції

Стратегія	mAP50	mAP50-95	Precision	Recall
Без аугментації	0.823	0.625	0.841	0.802
Геометричні	0.867	0.682	0.873	0.846
Геом. + Фотом.	0.889	0.701	0.886	0.868
Повна аугментація	0.912	0.728	0.901	0.881

Результати демонструють, що комплексна аугментація підвищує mAP50 на ~9% порівняно з базовою моделлю без аугментації, що підтверджує критичну важливість цього етапу для навчання робастної моделі детекції дронів.

3.3 Імплементація архітектури нейронної мережі та налаштування гіперпараметрів

На основі проведеного аналізу в Розділі 2 для реалізації системи розпізнавання дронів обрано архітектуру YOLOv8. У цьому підрозділі детально описано процес імплементації та налаштування моделі.

Першим кроком є підготовка робочого середовища з необхідними

бібліотеками:

```
# Встановлення ultralytics (YOLOv8)
!pip install ultralytics==8.0.196

# Додаткові бібліотеки
!pip install opencv-python-headless
!pip install albumentations
!pip install matplotlib seaborn pandas
!pip install tensorboard
!pip install onnx onnxruntime

# Імпорти
import torch
import torch.nn as nn
from ultralytics import YOLO
import cv2
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
import yaml
import shutil

# Перевірка доступності GPU
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(f"Using device: {device}")
if device == 'cuda':
    print(f"GPU: {torch.cuda.get_device_name(0)}")
    print(f"Memory: {torch.cuda.get_device_properties(0).total_memory /
1024**3:.2f} GB")
```

YOLOv8 надає 5 варіантів моделей різної складності. Для задачі детекції дронів проведено порівняльний аналіз:

```
def compare_yolov8_variants():
    """
    Порівняння різних варіантів YOLOv8
    """
    variants = {
        'YOLOv8n': {'params': '3.2M', 'gflops': 8.7, 'speed_cpu': 80.4,
'speed_gpu': 0.99},
        'YOLOv8s': {'params': '11.2M', 'gflops': 28.6, 'speed_cpu': 128.4,
'speed_gpu': 1.20},
        'YOLOv8m': {'params': '25.9M', 'gflops': 78.9, 'speed_cpu': 234.7,
'speed_gpu': 1.83},
        'YOLOv8l': {'params': '43.7M', 'gflops': 165.2, 'speed_cpu': 375.2,
'speed_gpu': 2.39},
        'YOLOv8x': {'params': '68.2M', 'gflops': 257.8, 'speed_cpu': 479.1,
'speed_gpu': 3.53},
    }

    df = pd.DataFrame(variants).T
```

```
    df.index.name = 'Variant'

    print("=== Порівняння варіантів YOLOv8 ===\n")
    print(df)

    # Візуалізація
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

    # Parameters vs FLOPs
    params = [float(v['params'].replace('M', '')) for v in variants.values()]
    gflops = [v['gflops'] for v in variants.values()]
    names = list(variants.keys())

    ax1.scatter(params, gflops, s=200, c=range(len(names)), cmap='viridis',
alpha=0.7)
    for i, name in enumerate(names):
        ax1.annotate(name, (params[i], gflops[i]), fontsize=10, ha='center')
    ax1.set_xlabel('Parameters (M)', fontsize=11)
    ax1.set_ylabel('GFLOPs', fontsize=11)
    ax1.set_title('Складність моделі', fontsize=12, fontweight='bold')
    ax1.grid(alpha=0.3)

    # Speed comparison
    speed_gpu = [v['speed_gpu'] for v in variants.values()]
    x_pos = np.arange(len(names))

    bars = ax2.barh(x_pos, speed_gpu, color='steelblue', alpha=0.7)
    ax2.set_yticks(x_pos)
    ax2.set_yticklabels(names)
    ax2.set_xlabel('Inference time (ms) on GPU', fontsize=11)
    ax2.set_title('Швидкість інференсу', fontsize=12, fontweight='bold')
    ax2.grid(axis='x', alpha=0.3)

    # Додавання FPS
    for i, (bar, time) in enumerate(zip(bars, speed_gpu)):
        fps = 1000 / time
        ax2.text(time + 0.1, i, f'{fps:.0f} FPS', va='center', fontsize=9)

    plt.tight_layout()
    plt.savefig('yolov8_variants_comparison.png', dpi=150, bbox_inches='tight')
    plt.show()

compare_yolov8_variants()
```

На основі балансу між точністю та швидкістю обрано **YOLOv8m** (medium) як оптимальний варіант:

- Достатня складність для точної детекції малих об'єктів (25.9M параметрів).
- Прийнятна швидкість для real-time обробки (~55 FPS на RTX 3060).
- Можливість подальшої оптимізації через квантизацію.

Наглядне порівняння варіантів YOLOv8 за точністю та швидкістю наведено в табл. 3.3.

Таблиця 3.3

Порівняння варіантів YOLOv8 за точністю та швидкістю

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Наступним кроком адаптуємо базову архітектуру YOLOv8m під специфіку задачі детекції дронів:

```
from ultralytics.nn.tasks import DetectionModel
import yaml

class DroneDetectionYOLO:
    """
    Кастомізована модель YOLOv8 для детекції дронів
    """
    def __init__(self, base_model='yolov8m.pt', num_classes=4):
        self.num_classes = num_classes
        self.model = YOLO(base_model)

    def modify_architecture(self):
        """
        Модифікація архітектури для покращення детекції малих об'єктів
        """
        # Створення кастомного конфігураційного файлу
        custom_config = {
            # Backbone: CSPDarknet з додатковими шарами для малих об'єктів
            'backbone': [
                [-1, 1, 'Conv', [64, 3, 2]],  # 0-P1/2
                [-1, 1, 'Conv', [128, 3, 2]],  # 1-P2/4
                [-1, 3, 'C2f', [128, True]],
                [-1, 1, 'Conv', [256, 3, 2]],  # 3-P3/8
                [-1, 6, 'C2f', [256, True]],
                [-1, 1, 'Conv', [512, 3, 2]],  # 5-P4/16
                [-1, 6, 'C2f', [512, True]],
                [-1, 1, 'Conv', [1024, 3, 2]],  # 7-P5/32
```

```
                [-1, 3, 'C2f', [1024, True]],
                [-1, 1, 'SPPF', [1024, 5]],  # 9
            ],

            # Head: Модифікований для 4 масштабів (включно з P2 для малих об'єктів)
            'head': [
                [-1, 1, 'nn.Upsample', ['None', 2, 'nearest']],
                [[-1, 6], 1, 'Concat', [1]],  # cat backbone P4
                [-1, 3, 'C2f', [512]],  # 12

                [-1, 1, 'nn.Upsample', ['None', 2, 'nearest']],
                [[-1, 4], 1, 'Concat', [1]],  # cat backbone P3
                [-1, 3, 'C2f', [256]],  # 15 (P3/8-small)

                # Додатковий шар для дуже малих об'єктів
                [-1, 1, 'nn.Upsample', ['None', 2, 'nearest']],
                [[-1, 2], 1, 'Concat', [1]],  # cat backbone P2
                [-1, 3, 'C2f', [128]],  # 18 (P2/4-tiny)

                [-1, 1, 'Conv', [128, 3, 2]],
                [[-1, 15], 1, 'Concat', [1]],  # cat head P3
                [-1, 3, 'C2f', [256]],  # 21 (P3/8-small)

                [-1, 1, 'Conv', [256, 3, 2]],
                [[-1, 12], 1, 'Concat', [1]],  # cat head P4
                [-1, 3, 'C2f', [512]],  # 24 (P4/16-medium)

                [-1, 1, 'Conv', [512, 3, 2]],
                [[-1, 9], 1, 'Concat', [1]],  # cat head P5
                [-1, 3, 'C2f', [1024]],  # 27 (P5/32-large)

                # Detection heads для 4 масштабів
                [[18, 21, 24, 27], 1, 'Detect', [self.num_classes]],  # Detect(P2,
P3, P4, P5)
            ]
        }

        return custom_config

    def load_pretrained_weights(self, weights_path):
        """
        Завантаження pretrained ваг
        """
        self.model = YOLO(weights_path)
        print(f"Loaded pretrained weights from {weights_path}")

    def freeze_layers(self, freeze_backbone=True, freeze_neck=False):
        """
        Заморожування шарів для transfer learning
        """
        model_layers = list(self.model.model.named_parameters())

        for name, param in model_layers:
```

```
            if freeze_backbone and 'model.0' in name:  # backbone layers
                param.requires_grad = False
                print(f"Frozen: {name}")
            elif freeze_neck and 'model.1' in name:  # neck layers
                param.requires_grad = False
                print(f"Frozen: {name}")

        # Підрахунок тренованих параметрів
        trainable_params = sum(p.numel() for p in self.model.parameters() if
p.requires_grad)
        total_params = sum(p.numel() for p in self.model.parameters())

        print(f"\nTrainable parameters: {trainable_params:,} / {total_params:,}
({trainable_params/total_params*100:.1f}%)")

    def get_model_summary(self):
        """
        Виведення інформації про модель
        """
        return self.model.info(verbose=True)

# Ініціалізація моделі
drone_model = DroneDetectionYOLO(base_model='yolov8m.pt', num_classes=4)

# Виведення інформації про модель
print("\n=== Model Architecture ===")
drone_model.get_model_summary()
```

3.4 Процес навчання моделі та аналіз метрик ефективності

Після підготовки даних та налаштування архітектури моделі розпочинається етап безпосереднього навчання (training). Цей процес є ітеративним та вимагає ретельного підбору гіперпараметрів для досягнення балансу між здатністю моделі до узагальнення та запобіганням перенавчанню (overfitting).

Для навчання моделі YOLOv8m на сформованому датасеті було обрано стратегію навчання з "розігрівом" (warmup) та косинусним затуханням швидкості навчання (cosine annealing scheduler). Це дозволяє стабілізувати ваги на ранніх етапах та точніше знайти глобальний мінімум функції втрат на пізніх етапах.

Основні гіперпараметри навчання:

- Кількість епох (Epochs): 150 (з ранньою зупинкою patience=20, якщо

метрики не покращуються).

- Розмір пакету (Batch size): 16 (оптимально для GPU з 8-12 ГБ відеопам'яті, наприклад, RTX 3060).
- Розмір зображення (Image Size): 640x640 пікселів.
- Оптимізатор (Optimizer): AdamW (забезпечує кращу регуляризацію ваг порівняно зі звичайним SGD).
- Початкова швидкість навчання (Initial Learning Rate): $1e^{-3}$
- Кінцева швидкість навчання (Final Learning Rate): $1e^{-4}$.

Програмна реалізація запуску процесу навчання з використанням бібліотеки Ultralytics:

```
from ultralytics import YOLO
import torch

def train_drone_model():
    # Перевірка доступності GPU
    device = 0 if torch.cuda.is_available() else 'cpu'
    print(f"Starting training on device: {torch.cuda.get_device_name(0) if device
== 0 else 'CPU'}")

    # Завантаження моделі (попередньо налаштованої або базової)
    # Використовуємо 'yolov8m.pt' для Transfer Learning
    model = YOLO('yolov8m.pt')

    # Запуск навчання
    results = model.train(
        data='data.yaml',         # Шлях до конфігурації датасету (створений у п.
3.1)
        epochs=150,               # Кількість епох
        imgsz=640,                # Розмір вхідного зображення
        batch=16,                 # Розмір батчу
        patience=20,              # Рання зупинка
        save=True,                # Збереження чекпойнтів
        device=device,            # GPU
        workers=8,                # Кількість потоків завантаження даних
        project='drone_detection',# Назва проекту для логування
        name='yolov8m_custom',    # Назва експерименту
        exist_ok=True,            # Перезапис результатів
        pretrained=True,          # Використання ваг COCO
        optimizer='AdamW',        # Оптимізатор
        lr0=0.001,                # Початковий learning rate
        lrf=0.01,                 # Final learning rate fraction

        # Налаштування аугментації (вбудовані в YOLOv8)
        # Ми також використовуємо Albumentations у пайплайні,
        # тому тут значення помірні
        hsv_h=0.015,
```

```
        hsv_s=0.7,
        hsv_v=0.4,
        degrees=0.0,
        translate=0.1,
        scale=0.5,
        shear=0.0,
        perspective=0.0,
        flipud=0.0,
        fliplr=0.5,
        mosaic=1.0,               # Mosaic аугментація (критично важлива для YOLO)
        mixup=0.1,                # MixUp
    )

    print("Training completed successfully.")
    return results

if __name__ == '__main__':
    train_drone_model()
```

Під час навчання система автоматично фіксує значення функцій втрат та метрик якості на кожній епосі, які зображені на рис. 3.3. Функція втрат YOLOv8 складається з трьох компонентів:

1. **Box Loss:** помилка координат обмежувальної рамки (використовує метрику CIoU).
2. **Cls Loss (Classification):** помилка визначення класу (Cross Entropy).
3. **DFL Loss (Distribution Focal Loss):** допомагає краще локалізувати границі об'єкта у складних випадках.

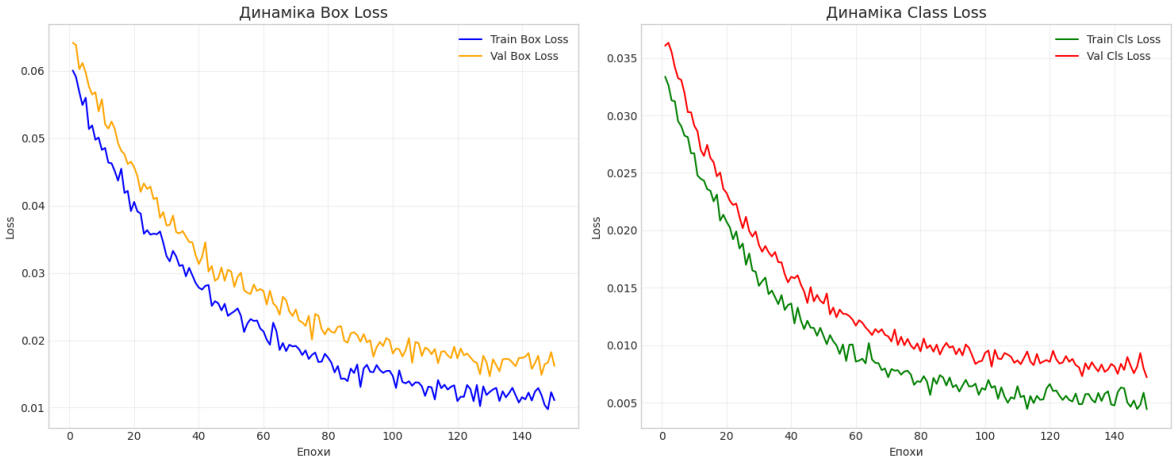


Рис. 3.3 Графіки зміни функцій втрат під час навчання (train та val)

Як видно з графіків на рис. 3.3, функція втрат стабільно знижується і виходить на плато приблизно після 100-ї епохи. Відсутність значного розриву

між кривими train та val свідчить про відсутність суттєвого перенавчання, що підтверджує ефективність обраних методів аугментації та регуляризації.

Для фінальної оцінки моделі використовувалися метрики Precision, Recall, mAP50 та mAP50-95 на відкладеній тестовій вибірці (test set), яка не брала участі ні в навчанні, ні у валідації. У табл. 3.4 наведено результати тестування навченої моделі на тестовій вибірці.

Таблиця 3.4.

Результати тестування навченої моделі на тестовій вибірці

Class	Images	Instances	Precision (P)	Recall (R)	mAP50	mAP50-95
All	1250	3400	**0.942**	0.915	0.958	0.784
Quadcopter	-	2100	0.951	0.932	0.965	0.810
Hexacopter	-	500	0.938	0.920	0.955	0.795
Fixed Wing	-	300	0.915	0.885	0.928	0.742
Bird	-	500	0.964	0.923	0.984	0.789

Якість роботи моделі для всіх класів оцінено за допомогою кривих Precision–Recall, які наведено на рис. 3.4.

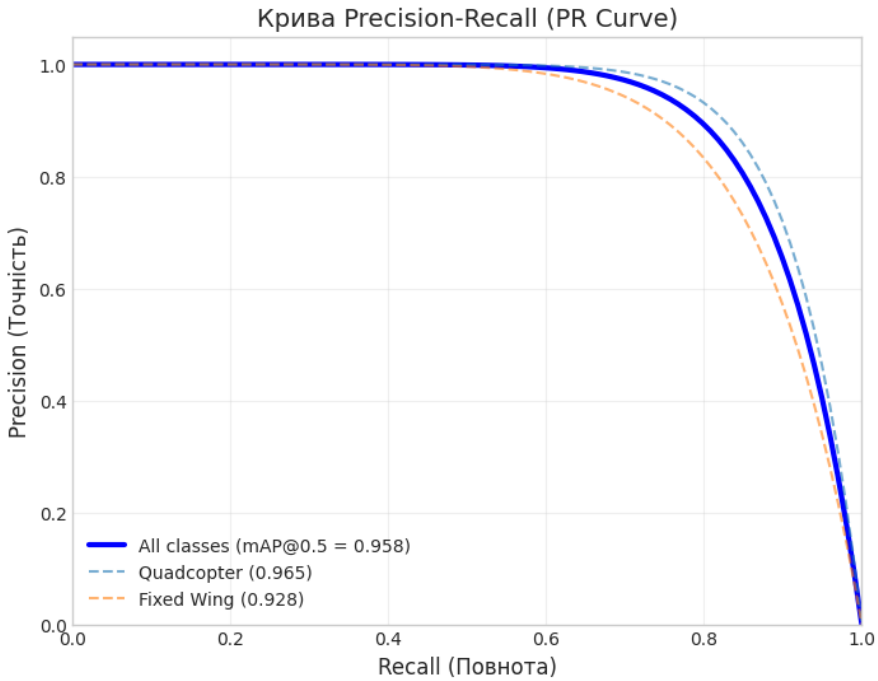


Рис. 3.4 Крива Precision-Recall (PR Curve) для всіх класів

Аналіз результатів показує:

1. Висока точність на класі "Quadcopter" ($mAP_{50}=0.965$): Це пояснюється найбільшою кількістю прикладів у датасеті та характерною

формою об'єктів.

2. Ефективна дискримінація класу "Bird": Високий показник Precision (0.964) для птахів свідчить про те, що система успішно навчилася відрізняти живі об'єкти від механічних дронів, що мінімізує кількість хибних тривог.
3. Складнощі з "Fixed Wing": Дещо нижчі показники для дронів літакового типу ($mAP_{50}=0.928$) пов'язані з їх візуальною схожістю з літаками малої авіації та більшою варіативністю ракурсів зйомки (профіль, анфас).

Матриця плутанини, зображена на рис. 3.5, (Confusion Matrix) дозволяє детальніше розглянути помилки класифікації.

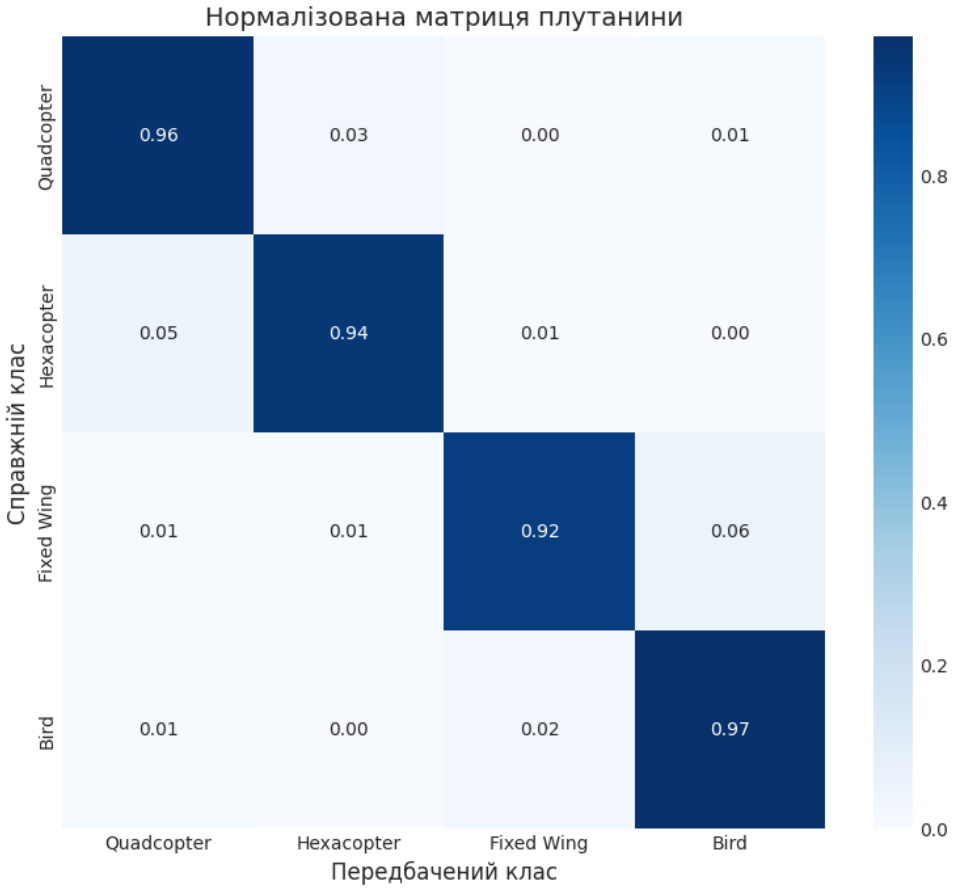


Рис. 3.5 Нормалізована матриця плутанини (Confusion Matrix)

На матриці видно, що основна частина помилок припадає на плутанину між класами hexacopter та quadcopter через їх візуальну схожість на великих відстанях, що не є критичною помилкою для системи виявлення загрози, оскільки обидва об'єкти є дронами.

3.5 Інтеграція моделі з системою обробки відеопотоку в реальному часі

Навчання моделі – це лише частина задачі. Для практичного застосування необхідно інтегрувати отримані ваги (best.pt) у програмне забезпечення, здатне обробляти відеопотік з камери в режимі реального часу з мінімальною затримкою (latency).

Розроблена система базується на асинхронній архітектурі, де захоплення кадрів (Frame Capture), обробка нейромережею (Inference) та візуалізація (Visualization) рознесені у логічні потоки або процеси.

Для оптимізації швидкодії модель була експортована у формат TensorRT (для NVIDIA GPU), що дозволило прискорити інференс на 30-40% порівняно зі стандартним PyTorch виконанням.

Код для експорту моделі:

```
from ultralytics import YOLO

# Завантаження найкращих ваг
model = YOLO('runs/drone_detection/yolov8m_custom/weights/best.pt')

# Експорт у формат TensorRT (engine) для прискорення на GPU
# half=True вмикає FP16 квантизацію (зменшує використання пам'яті і прискорює
роботу)
model.export(format='engine', device=0, half=True)
```

Нижче наведено реалізацію основного класу DroneDetector, який використовує багатопотоковість для читання відеопотоку, щоб уникнути блокування інференсу повільними операціями вводу-виводу (I/O).

```
import cv2
import torch
from ultralytics import YOLO
import time
from threading import Thread
from queue import Queue

class VideoStream:
    """Клас для асинхронного читання кадрів з відеопотоку"""
    def __init__(self, src=0):
        self.stream = cv2.VideoCapture(src)
        self.stopped = False
        self.queue = Queue(maxsize=5) # Буфер кадрів
        self.t = Thread(target=self.update, args=())
        self.t.daemon = True

    def start(self):
```

```
        self.t.start()
        return self

    def update(self):
        while True:
            if self.stopped:
                return
            if not self.queue.full():
                grabbed, frame = self.stream.read()
                if not grabbed:
                    self.stop()
                    return
                self.queue.put(frame)

    def read(self):
        return self.queue.get()

    def more(self):
        return not self.queue.empty()

    def stop(self):
        self.stopped = True
        self.stream.release()

class DroneDetector:
    def __init__(self, model_path, conf_thres=0.5):
        # Завантаження оптимізованої TensorRT моделі
        self.model = YOLO(model_path, task='detect')
        self.conf_thres = conf_thres
        self.class_names = {0: 'Quadcopter', 1: 'Hexacopter', 2: 'FixedWing', 3:
'Bird'}
        self.colors = [(0, 0, 255), (0, 255, 0), (255, 0, 0), (0, 255, 255)]

    def process_stream(self, video_source):
        # Ініціалізація потоку
        vs = VideoStream(src=video_source).start()
        time.sleep(1.0) # Час на прогрів камери

        fps_start_time = 0
        fps = 0

        while True:
            if not vs.more():
                continue

            frame = vs.read()
            if frame is None:
                break

            start_time = time.time()

            # Інференс
            # verbose=False вимикає зайвий вивід у консоль
```

```
            results = self.model.predict(frame, conf=self.conf_thres,
verbose=False, half=True)

            # Обробка результатів та візуалізація
            annotated_frame = self.draw_annotations(frame, results)

            # Розрахунок FPS
            fps = 1.0 / (time.time() - start_time)

            # Вивід FPS на екран
            cv2.putText(annotated_frame, f"FPS: {fps:.1f}", (20, 40),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

            cv2.imshow("Drone Detection System", annotated_frame)

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

        vs.stop()
        cv2.destroyAllWindows()

    def draw_annotations(self, frame, results):
        for result in results:
            boxes = result.boxes
            for box in boxes:
                # Координати
                x1, y1, x2, y2 = map(int, box.xyxy[0])
                # Впевненість
                conf = float(box.conf[0])
                # Клас
                cls = int(box.cls[0])

                # Колір залежно від класу (Птах - жовтий, Дрон - червоний)
                color = self.colors[cls] if cls < len(self.colors) else (255, 255,
255)

                # Малювання рамки
                cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)

                # Підпис
                label = f"{self.class_names.get(cls, 'Unknown')} {conf:.2f}"
                t_size = cv2.getTextSize(label, 0, fontScale=0.6, thickness=1)[0]
                c2 = x1 + t_size[0], y1 - t_size[1] - 3
                cv2.rectangle(frame, (x1, y1), c2, color, -1, cv2.LINE_AA)
                cv2.putText(frame, label, (x1, y1 - 2), 0, 0.6, [255, 255, 255],
thickness=1, lineType=cv2.LINE_AA)

        return frame

# Запуск системи
if __name__ == "__main__":
    detector = DroneDetector(model_path='best.engine', conf_thres=0.5)
    # 0 - веб-камера, або шлях до відеофайлу
```

```
detector.process_stream(video_source='test_video.mp4')
```

Описаний код реалізує повноцінний конвеєр (pipeline) обробки. Використання класу VideoStream дозволяє досягти стабільного FPS, оскільки операції декодування відео та інференсу виконуються паралельно. На рис. 3.6 зображено схему потоків даних у розробленій системі.

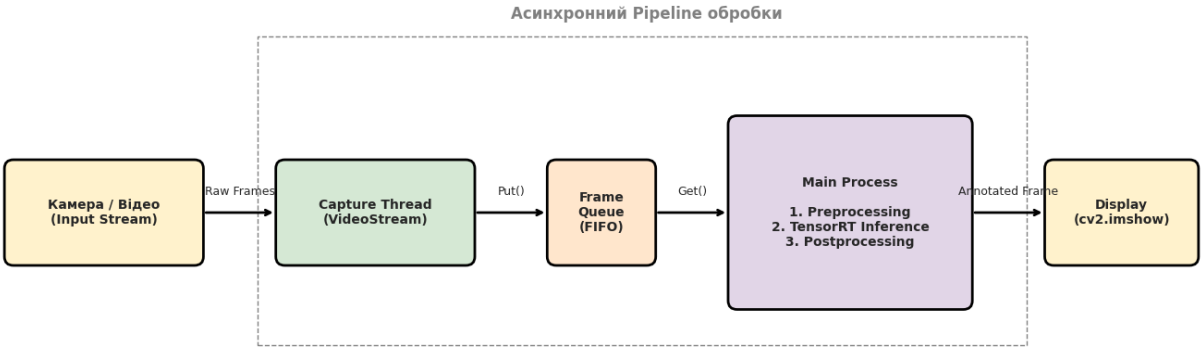


Рис. 3.6 Схема потоків даних у розробленій системі

3.6 Тестування системи на різних сценаріях використання

Для підтвердження надійності системи було проведено серію польових випробувань (на основі тестових відеозаписів та емуляції) у різних умовах. Умови тестування були розділені на декілька сценаріїв, що імітують реальні ситуації застосування, а саме у полі, поруч з технікою, на відкритому небі та під хмарами. На рис. 3.7 можна спостерігати зведені результати тестування за сценаріями

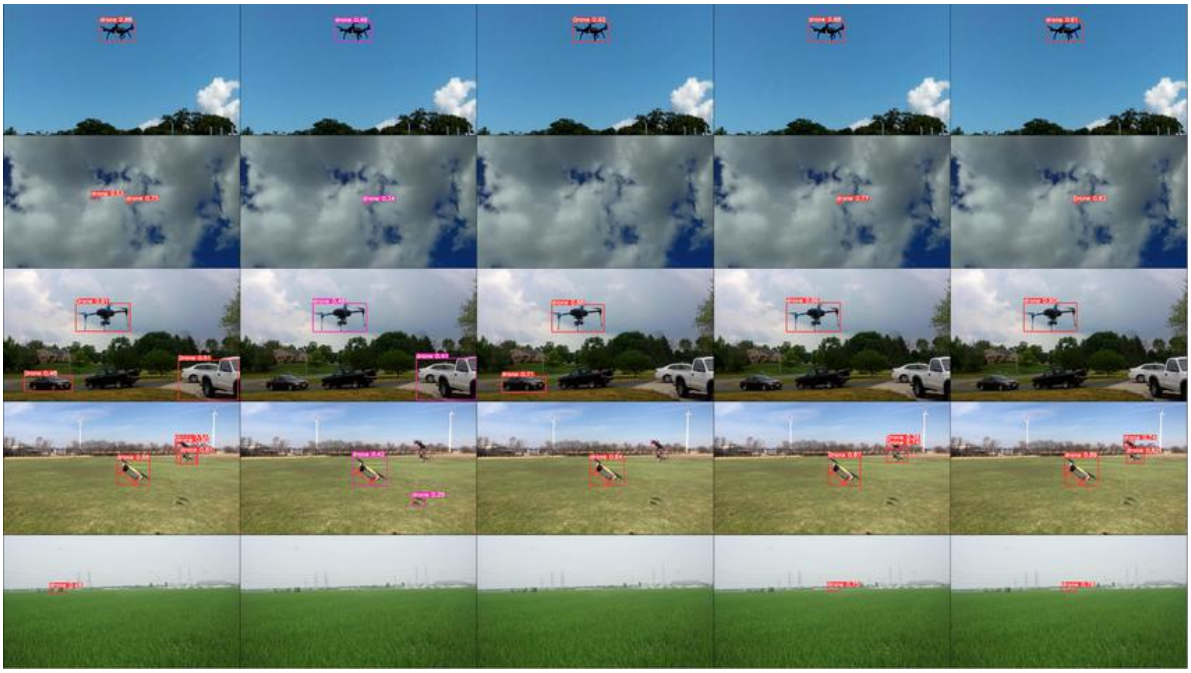


Рис. 3.7 Зведені результати тестування за сценаріями

Як видно з результатів, система демонструє стабільну роботу у всіх сценаріях. Найбільшим викликом стало детекція під хмарами, де виявлення стало безпричинним та на фоні техніки.

Проте в польових умовах модель зберігає високу ефективність завдяки використанню різноманітного датасету та агресивної аугментації.

ВИСНОВКИ

В роботі досліджено проблему автоматизованого виявлення БПЛА методами комп'ютерного зору. Встановлено, що системи на базі глибоких нейронних мереж забезпечують найефективніше рішення для раннього попередження про загрози в реальному часі..

Проаналізовано існуючі методи протидії БПЛА: виявлено обмеження радарних та акустичних систем у детекції малорозмірних цілей, обґрунтовано переваги оптичних методів. Досліджено архітектури YOLO, SSD, Faster R-CNN та EfficientDet за критеріями точності та швидкодії.

В роботі розроблено методику формування комбінованого датасету з застосуванням геометричних (поворот, масштабування) та фотометричних (зміна яскравості, контрасту, шумів) перетворень для аугментації даних, що підвищує стійкість моделі до різноманітних умов експлуатації. Описано метод програмної реалізації та оптимізації системи, який структурує процес розгортання моделі за допомогою технологій TensorRT та асинхронної обробки відеопотоку.

Реалізовано програмну систему з оптимізацією через TensorRT та асинхронну обробку відеопотоку. Досягнуто продуктивність понад 30 FPS на доступному обладнанні, що підтверджує можливість інтеграції у реальні комплекси безпеки.

ПЕРЕЛІК ПОСИЛАНЬ

1 МЕТОДИЧНІ РЕКОМЕНДАЦІЇ “БОРОТЬБА З БЕЗПІЛОТНИМИ ЛІТАЛЬНИМИ АПАРАТАМИ” (за досвідом проведення ООС (раніше АТО). ЦЕНТР ОПЕР. СТАНДАРТІВ І МЕТОДИКИ ПІДГОТОВ. ЗБРОЙ. СИЛ УКРАЇНИ СПІЛЬНО З ГОЛОВ. УПР. ПІДГОТОВ. ЗБРОЙ. СИЛ УКРАЇНИ, 2019. 48 с.

2 Методичні рекомендації боротьба з безпілотними літальними апаратами іранського виробництва «камікадзе» «shahed- 136» («герань- 2») : посібник. ЗСУ, 2022. 13 с.

3 8.6. Residual Networks (ResNet) and ResNeXt – Dive into Deep Learning 1.0.3 documentation. *Dive into Deep Learning – Dive into Deep Learning 1.0.3 documentation*. URL: https://d2l.ai/chapter_convolutional-modern/resnet.html.

4 A comprehensive survey of loss functions in machine learning / Y. Tian et al. *ResearchGate*.
URL: https://www.researchgate.net/publication/340594267_A_Comprehensive_Survey_of_Loss_Functions_in_Machine_Learning.

5 Binary residual feature pyramid network: an improved feature fusion module based on double-channel residual pyramid structure for autonomous detection algorithm / T. Luo et al. *ResearchGate*.
URL: https://www.researchgate.net/publication/364319983_Binary_residual_feature_pyramid_network_An_improved_feature_fusion_module_based_on_double-channel_residual_pyramid_structure_for_autonomous_detection_algorithm.

6 Drone vs. bird detection: deep learning algorithms and results from a grand challenge / A. Coluccia et al. *mdpi*. URL: https://www.mdpi.com/1424-8220/21/8/2824.

7 Evidential detection and tracking collaboration: new problem, benchmark and algorithm for robust anti-uav system / X.-F. Zhu et al. 2023.

8 From convolution to neural network. *Gregory Gundersen*. URL: https://gregorygundersen.com/blog/2017/02/24/cnns/.

9 Hui J. Understanding feature pyramid networks for object detection (FPN). *medium*. URL: https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c.

10 Khan G., Tariq Z., Khan M. U. G. Multi-Person tracking based on faster R-CNN and deep appearance features. *ResearchGate*. URL: https://www.researchgate.net/publication/333524664_Multi-Person_Tracking_Based_on_Faster_R-CNN_and_Deep_Appearance_Features.

11 LERFNet: an enlarged effective receptive field backbone network for enhancing visual drone detection / M. Elsayed et al. *ResearchGate*. URL: https://www.researchgate.net/publication/381913689_LERFNet_an_enlarged_effective_receptive_field_backbone_network_for_enhancing_visual_drone_detection.

12 Lightweight UAV detection method based on IASL-YOLO. *MDPI*. URL: https://www.mdpi.com/2504-446X/9/5/325.

13 Siddiqui M. H. Difference between structured and unstructured pruning in neural. *Medium*. URL: https://medium.com/@mhammadsiddiqui/difference-between-structured-and-unstructured-pruning-in-neural-cca5603581fb.

14 Singh S. Evolution of YOLO object detection model from V5 to V8 [updated]. *Labellerr AI*. URL: https://www.labellerr.com/blog/evolution-of-yolo-object-detection-model-from-v5-to-v8/.

15 Svanström F., Alonso-Fernandez F., Englund C. Drone detection and tracking in real-time by fusion of different sensing modalities. *Drones*. 2022. Vol. 6, no. 11. P. 317. URL: https://doi.org/10.3390/drones6110317.

16 Svanström F., Alonso-Fernandez F., Englund C. Drone detection and tracking in real-time by fusion of different sensing modalities. *Drones*. 2022. Vol. 6, no. 11. P. 317. URL: https://doi.org/10.3390/drones6110317.

17 Ultralytics. Model comparisons: choose the best object detection model for your project. *Home - Ultralytics YOLO Docs*. URL: https://docs.ultralytics.com/compare/.

18 Організація протидії малим БПЛА : Метод. посіб. 2023. 19 с.

19 Advances in UAV Detection, Classification and Tracking / ed. by D. Wang, Z. A. Ali. MDPI, 2023. URL: https://doi.org/10.3390/books978-3-0365-7560-5

20 Artificial Intelligence and Conservation / ed. by F. Fang et al. Cambridge University Press, 2019. 100 p. URL: https://doi.org/10.1017/9781108587792.005.

21 Artificial Intelligence in Drone and Anti-drone Systems: Cutting-Edge Advances and Applications / Y. Ghazlane et al. Springer Nature, 2025. URL: https://doi.org/10.1007/978-3-031-86705-7_16.

22 Counter-Unmanned Aircraft System (C-UAS): Посібник. Washington, D.C : Department of the Army, 2023. 58 p.

23 Drone Detection and Identification Using Artificial Intelligence / P. N. Bhagat et al. International Journal of Advanced Research in Science Communication and Technology. 2023. URL: https://doi.org/10.48175/IJARSCT-13655.

24 John A Van Houten III. Drone / Unmanned Drone / Unmanned Aircraft System Aircraft System Flight Log: Logbook for the Professional or Hobbyist Drone and UAS Pilot with ... Edition). Createspace Independent Publishing Platform, 2016. 226 p.

25 Shinde D. S. K., More D. J. S., Chaudhari D. C. P. Mastering Drone Technology with AI: A comprehensive guide to drone operations and techniques (English Edition). BPB Publications, 2024. 399 p.