Державний торговельно-економічний університет

Кафедра комп’ютерних наук та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА
на тему:

«Розробка систем детекції обєктів на зображеннях із використанням моделей YOLO»

Студента 2 курсу, 5м групи
ФЗ «Комп’ютерні науки» _____ *підпис студента* Карабач Андрія Анатолійовича

Науковий керівник
кандидат фізико-математичних наук, доцент _____ *підпис керівника* Філімонова Тетяна Олегівна

Гарант освітньої програми
доктор фізико-математичних наук, професор _____ *підпис керівника* Пурський Олег Іванович

Київ 2025

Державний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра комп'ютерних наук та інформаційних систем
ГЗ «Комп'ютерні науки»
Освітня програма «Комп'ютерні науки»

Затверджую
Зав. кафедри _____Пурський О.І.
«20» грудня 2024р.

Завдання
на кваліфікаційну роботу студенту

Карабач Андрій Анатолійович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
«Розробка систем детекції обєктів на зображеннях із використанням моделей YOLO»
Затверджена наказом ректора від *«16» грудня 2024 р. № 4089*
2. Строк здачі студентом закінченої роботи *9 грудня 2025 року*
3. Цільова установка та вихідні дані до роботи
Мета роботи: розробка моделі та інформаційної технології детекції об'єктів на зображеннях із використанням нейронних мереж сімейства YOLO.
Об'єкт дослідження: процес автоматизації виявлення та класифікації об'єктів на зображеннях за допомогою сучасних методів комп'ютерного зору.
Предмет дослідження: моделі, методи та інформаційні технології детекції об'єктів на зображеннях із використанням глибинного навчання, зокрема архітектур YOLO.

4. Перелік графічного матеріалу_____
_____
_____
_____

5. Консультанти по роботі із зазначенням розділів, за якими здійснюється консультування:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Філімонова Т.О.	20.12.2024 р.	20.12.2024 р.
2	Філімонова Т.О.	20.12.2024 р.	20.12.2024 р.
3	Філімонова Т.О.	20.12.2024 р.	20.12.2024 р.

6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ДЕТЕКЦІЇ ОБ'ЄКТІВ НА ЗОБРАЖЕННЯХ

1.1. Сутність та завдання комп'ютерного зору

1.2. Методи та алгоритми детекції об'єктів

1.3. Моделі сімейства YOLO (You Only Look Once)

Висновки по розділу

РОЗДІЛ 2. МЕТОДОЛОГІЯ ТА ІНСТРУМЕНТИ РОЗРОБКИ СИСТЕМИ

2.1. Вимоги до системи детекції об'єктів

2.2. Інструменти та середовище реалізації

2.3. Підготовка та обробка даних

Висновки по розділу

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ ДЕТЕКЦІЇ

3.1. Архітектура програмного рішення

3.2. Підготовка даних

3.3. Реалізація та налаштування моделі YOLO

3.4. Навчання і тестування

Висновки по розділу

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

7. Календарний план виконання роботи

№ По р.	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	фактично
1	2	3	4
1	*Вибір теми кваліфікаційної роботи*	*01.11.2024*	05.11.2024
2	*Розробка та затвердження завдання для кваліфікаційної роботи*	*20.12.2024*	22.12.2024
3	*Вступ*	*02.06.2025*	13.06.2025
4	*РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ДЕТЕКЦІЇОБ'ЄКТІВНА ЗОБРАЖЕННЯХ*	*25.06.2025*	02.10.2025
5	*РОЗДІЛ 2. МЕТОДОЛОГІЯ ТА ІНСТРУМЕНТИ РОЗРОБКИ СИСТЕМИ*	*02.09.2025*	13.10.2025
6	*Підготовка статті у збірник наукових статей магістрів*	*09.09.2025*	20.10.2025
7	*РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ ДЕТЕКЦІЇ*	*21.10.2025*	02.11.2025
8	*Висновки*	*03.11.2025*	06.11.2025
9	*Здача кваліфікаційної роботи на кафедру науковому керівнику*	*05.11.2025*	08.11.2025
10	*Попередній захист кваліфікаційної роботи*	*20.11.2025*	21.11.2025
11	*Виправлення зауважень, зовнішнє рецензування кваліфікаційної роботи*	*24.11.2025*	24.11.2025
12	*Представлення готової зшитої кваліфікаційної роботи*	*26.11.2025*	28.11.2025
13	*Публічний захист кваліфікаційної роботи*	*Згідно роботи ЕК*	*Згідно роботи ЕК*

8. Дата видачі завдання «20» грудня 2024 р

9. Керівник кваліфікаційної роботи Філімонова Т.О.

(прізвище, ініціали, підпис)

10. Гарант освітньої програми Пурський О.І.

(прізвище, ініціали, підпис)

11. Завдання прийняв до виконання студент Карабач А.А.

(прізвище, ініціали, підпис)

12. Відгук керівника кваліфікаційної роботи

У кваліфікаційній роботі побудовано модель детекції об’єктів на зображеннях.
Запропоновано концепцію побудови інформаційної системи, яка реалізує автоматизовану обробку зображень, ідентифікацію та класифікацію об’єктів. Розроблено та протестовано прототип системи детекції об’єктів із використанням моделей YOLOv5/YOLOv8. Всі поставлені завдання виконано. Робота може бути допущена до захисту.

Керівник кваліфікаційної роботи
Філімонова Т.О. 10.12.25
(підпис, дата)

13. Висновок про кваліфікаційну роботу

Кваліфікаційна робота студента Карабач А.А
(прізвище, ініціали)
може бути допущена до захисту в екзаменаційній комісії.

Гарант освітньої програми Пурський О.І.
(підпис, прізвище, ініціали)

Завідувач кафедри Пурський О.І.
(підпис, прізвище, ініціали)
«_____»_____2025 р.

Анотація

У кваліфікаційній роботі здійснено розробку моделей та інформаційної технології детекції об’єктів на зображеннях із використанням сучасних нейронних мереж сімейства YOLO. Теоретично обґрунтовано принципи побудови систем комп’ютерного зору та розглянуто методи глибинного навчання, що забезпечують високу швидкість і точність виявлення об’єктів у реальному часі. Запропоновано концепцію побудови інформаційної системи, яка реалізує автоматизовану обробку зображень, ідентифікацію та класифікацію об’єктів. Розроблено та протестовано прототип системи детекції об’єктів із використанням моделей YOLOv5/YOLOv8, що демонструє ефективність у задачах виявлення різнорідних об’єктів на зображеннях.

Ключові слова: детекція об’єктів, нейронна мережа, комп’ютерний зір, глибинне навчання, YOLO, інформаційна технологія.

Annotation

The qualification work is devoted to the development of models and information technology for object detection on images using modern neural networks of the YOLO family. The principles of computer vision system design are theoretically substantiated, and deep learning methods ensuring high speed and accuracy of real-time object detection are analyzed. The concept of an information system for automated image processing, object identification, and classification is proposed. A prototype of the object detection system using YOLOv5/YOLOv8 models has been developed and tested, demonstrating efficiency in detecting various types of objects inimages.

Keywords: object detection, neural network, computer vision, deep learning, YOLO, information technology.

ЗМІСТ

ВСТУП

Сучасний етап розвитку інформаційних технологій визначається широким упровадженням систем штучного інтелекту, що базуються на методах машинного та глибинного навчання. Одним із ключових напрямів їх застосування є комп'ютерний зір, здатний автоматично аналізувати й інтерпретувати зображення та відеопотоки.

Особливе місце в цій галузі посідає детекція об'єктів, яка забезпечує ідентифікацію та локалізацію елементів сцени у просторі зображення. Практична значущість таких систем обумовлена широким спектром їх використання: від відеоспостереження, транспортної безпеки та промислового контролю якості – до медицини, робототехніки та аграрних технологій.

Науково-технічний прогрес останніх років показав, що найбільш ефективними є моделі детекції, засновані на архітектурах YOLO (You Only Look Once), які поєднують високу точність із можливістю роботи в режимі реального часу. Це відкриває перспективи для масштабного використання зазначених моделей у критично важливих сферах, де швидкість і надійність аналізу даних безпосередньо впливають на ефективність прийняття рішень. Таким чином, дослідження та практична реалізація системи детекції об'єктів на базі YOLO є актуальним завданням сучасної науки та інженерії.

Мета і завдання дослідження. Метою даного дослідження є розробка та впровадження системи детекції об'єктів на основі моделі YOLO, здатної забезпечувати високу точність і швидкодію в умовах реального часу та практичної інтеграції у прикладні середовища.

Для досягнення поставленої мети передбачено вирішення таких наукових і практичних **завдань**:

- Провести аналіз теоретичних основ комп'ютерного зору та методів детекції об'єктів.
- Дослідити еволюцію моделей YOLO та обґрунтувати вибір архітектури для реалізації системи.

- Визначити вимоги до системи детекції та сформувати критерії її ефективності.
- Обрати програмно-апаратні засоби, середовище та інструментарій реалізації.
- Сформувати та підготувати набір даних для навчання й тестування моделі.
- Розробити архітектуру системи та налаштувати параметри моделі YOLO.
- Провести навчання та тестування моделі із застосуванням сучасних метрик (precision, recall, mAP).
- Реалізувати інтеграцію системи у прикладне середовище та дослідити її ефективність на практичних прикладах.

Об’єктом дослідження є процесиї виявлення та класифікації об’єктів на зображеннях за допомогою сучасних методів комп’ютерного зору.

Предметом дослідження виступають методи, алгоритми та програмні засоби розробки систем детекції об’єктів, зокрема архітектурні особливості моделей сімейства YOLO та інструменти їх реалізації у практичних інформаційних системах.

Методологічною основою дослідження є поєднання системного підходу та методів комп’ютерних наук. У процесі виконання роботи використано:

- методи системного аналізу – для формування вимог до системи та побудови архітектури;
- аналітичні методи – для узагальнення наукових праць і порівняння сучасних моделей детекції;
- методи глибинного навчання – для побудови та налаштування моделей YOLO;
- експериментальні методи – для перевірки працездатності системи та оцінювання точності й швидкодії;

- методи моделювання та візуалізації – для представлення результатів у вигляді графіків, таблиць та зображень.

Наукова новизна роботи полягає у системному узагальненні сучасних підходів до детекції об’єктів і практичній реалізації оптимізованої системи на основі YOLO, здатної функціонувати в умовах реального часу. Практичне значення полягає в тому, що розроблену систему можна застосовувати у сфері відеоаналітики, безпеки, промислового моніторингу та інших прикладних галузях, де необхідна автоматична обробка зорової інформації.

Публікації. Результати дослідження опубліковано у збірнику наукових статей студентів, які здобувають освітній ступінь «магістр» за спеціальністю 122 «Комп’ютерні науки» Державного торговельно-економічного університету. Стаття **«Розробка системи детекції об’єктів на зображеннях із використанням моделей YOLO»** // *Прикладні комп’ютерні технології : збірник наукових статей студентів* / відп. ред. А. В. Селіванова. — Київ : ДТЕУ, 2025. — **С. 35–38.**

Структура та обсяг кваліфікаційної роботи. Кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних **джерел, додатків і містить** 69 сторінки основного тексту**,** 45 рисунків **та** 5 таблиці**.**

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ДЕТЕКЦІЇ ОБ'ЄКТІВ НА ЗОБРАЖЕННЯХ

1.1. Сутність та завдання комп'ютерного зору

Комп'ютерний зір (Computer Vision, CV) – це міждисциплінарна галузь, що вивчає методи автоматичного отримання, аналізу та інтерпретації інформації зі зображень і відео з метою формування корисних для прийняття рішень представлень світу. Якщо традиційна обробка зображень зосереджена на перетвореннях і фільтрації сигналів, то комп'ютерний зір прагне відновити семантику сцени: «що» зображено (класи), «де» це знаходиться (локалізація), «як» змінюється в часі (трекинг), та «які» взаємозв'язки між об'єктами (просторово-семантичний контекст) [1; 2]. У прикладному вимірі CV виступає «органом чуття» цифрових систем: від автономного транспорту й робототехніки до діагностичних медичних комплексів, промислових ліній контролю якості, роздрібної аналітики й систем безпеки [1; 3].

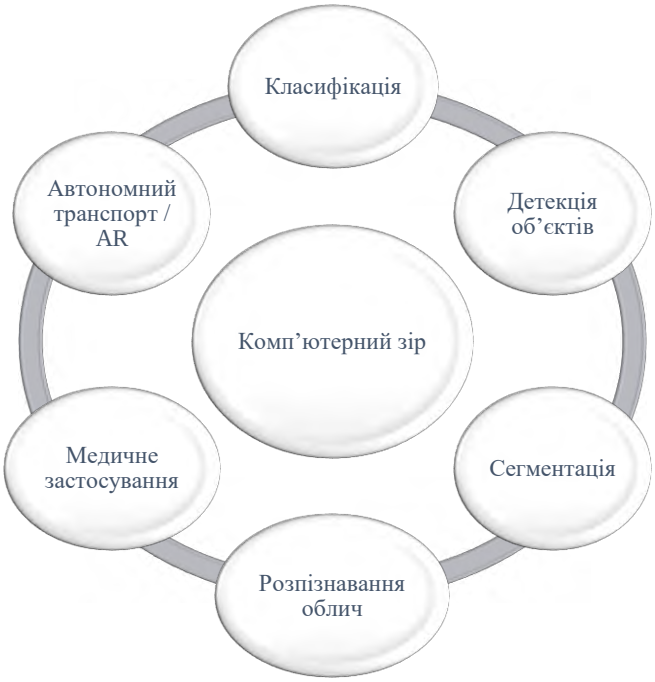

Рис. 1.1 – Ключові задачі комп'ютерного зору

Еволюція завдань комп'ютерного зору історично рухалася від низькорівневих операцій (виявлення контурів, кути, дескриптори ознак) до високорівневих інтерпретацій (розпізнавання категорій, детекція та сегментація об'єктів).

Стандартна таксономія включає:

- класифікацію зображень;
- локалізацію та детекцію об’єктів (рамки та класи);
- інстанс- та семантичну сегментацію;
- трекинг у відео;
- оцінювання поз/ключових точок;
- задачі 3D-відновлення та мультимодальних відповідностей.

Сучасна практика спирається на відкриті бенчмарки і метрики: для детекції – IoU, AP і mAP (VOC, COCO), що уможливлює зіставність результатів і пришвидшує прогрес у галузі [5; 6].

Технологічний прорив у CV пов’язують із «глибинним зрушенням»: поява глибинних згорткових мереж радикально підвищила якість на широкому спектрі завдань. Публікація AlexNet (2012) продемонструвала, що глибокі CNN, навчувані на великомасштабних наборах (ImageNet), суттєво випереджають класичні методи ознак та класифікаторів [11; 2]. Далі з’явилися регіон-пропозиційні та енд-ту-енд детектори (R-CNN, Fast/Faster R-CNN), які поєднали CNN з механізмами пропозицій, скоротивши розрив між точністю й швидкістю [7; 8]. Надалі, одноетапні підходи (родина YOLO, SSD, RetinaNet) запропонували розглядати детекцію як регресійне завдання в один прохід, що забезпечило роботу в реальному часі на GPU та edge-пристроях [7]. Паралельно з цим виник клас трансформерних детекторів (DETR), які формулюють детекцію як задачу предикції множини з біпартитним зіставленням, спрощуючи пайплайн та прибираючи ручні евристики (наприклад, NMS) [8].

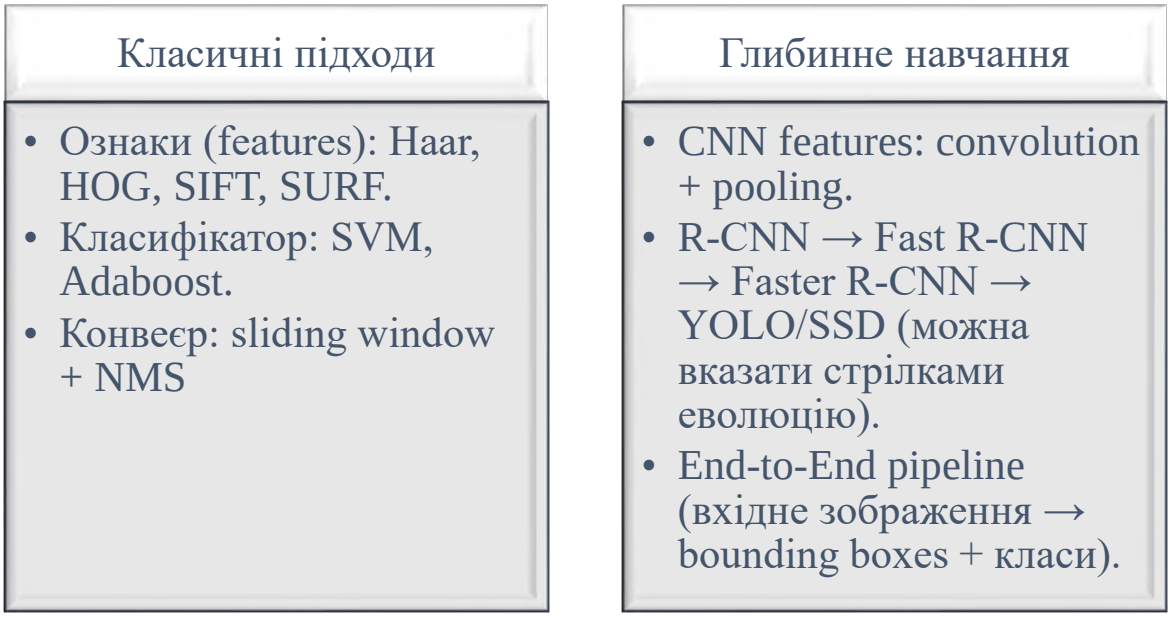

Рис. 1.2 – Порівняння класичних та DL-підходів у детекції

Роль у сучасних IT-системах. З погляду інженерії, комп'ютерний зір є компонентом комплексної ML-CD/CI-екосистеми: збирання даних → розмітка → підготовка/аугментація → тренування → валідація/тюнінг → розгортання (on-device/edge/cloud) → моніторинг (drift, bias, продуктивність) → MLOps-керування версіями моделей і датасетів. Суворі обмеження продуктивності (latency, FPS, енергоспоживання, пам'ять) визначають архітектурні рішення: квантovina, прунинг, знання-дистиляція, апаратне прискорення (GPU/TPU/NPU), оптимізація операцій та обміну даними [9]. Саме вимоги реального часу (на кшталт відеоаналітики або автономного керування) стимулювали розвиток одноетапних детекторів і трансформерних підходів із паралельною обробкою [7; 8].

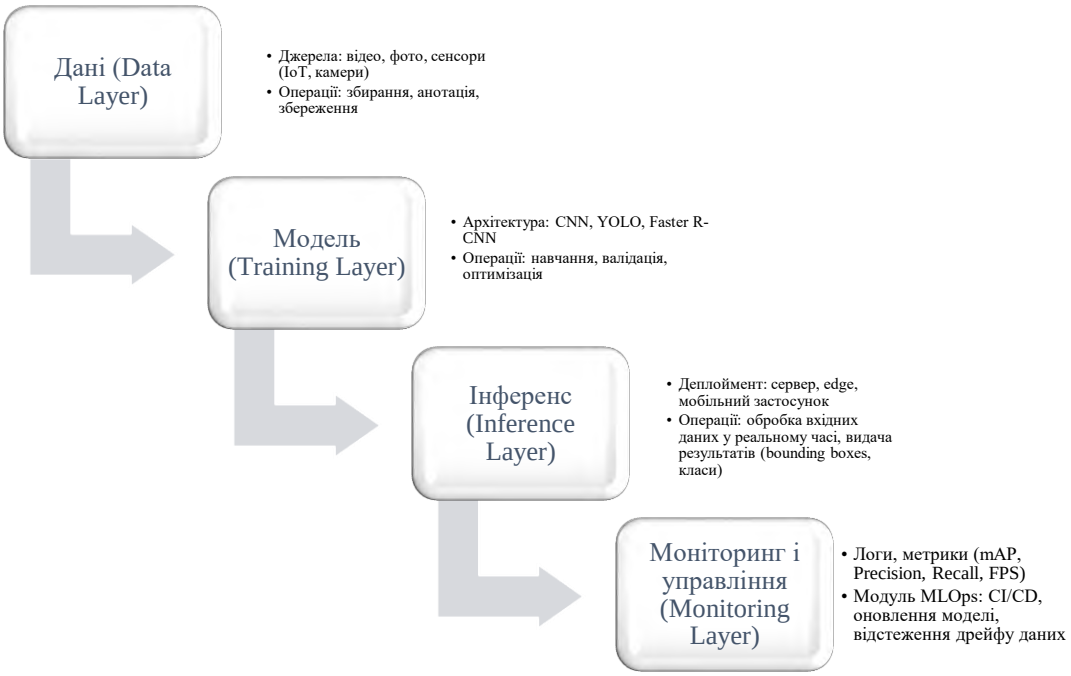

Рис. 1.3 – Архітектура комп'ютерно-зорової підсистеми в IT-рішенні.

Дані, стандарти та відтворюваність. Відкриті набори PASCAL VOC і MS COCO стандартизували формати розмітки та протоколи оцінювання, що стало фундаментом відтворюваної науки й чесного порівняння моделей (mAP@[.50:.95], AP50/75, тощо). Вони задають різнорівневу складність (щільні сцени, часткові перекриття, дрібні об'єкти) та слугують еталоном як для класичних, так і для сучасних архітектур [5; 6]. У промислових умовах ці стандарти комбінують із внутрішніми корпоративними наборами даних та активним навчанням для покриття доменних зсувів.

Інструментарій і екосистема. Розвиток CV невіддільний від потужних бібліотек та фреймворків. OpenCV забезпечує базову обробку, I/O камер, класичні алгоритми та інтеграцію з апаратурою – це де-факто стандарт для побудови конвеєрів «зображення→ознаки/попередня обробка» [10]. Для глибинного навчання домінують PyTorch/TensorFlow, де PyTorch популярний своєю імперативною моделлю програмування, зручною для досліджень і продакшн-оптимізації (TorchScript/ONNX/Accelerated backends) [4]. Саме ця екосистема знизила вхідний бар'єр, дозволивши будувати від лабораторних прототипів до промислових CV-сервісів з CI/CD і MLOps-практиками.

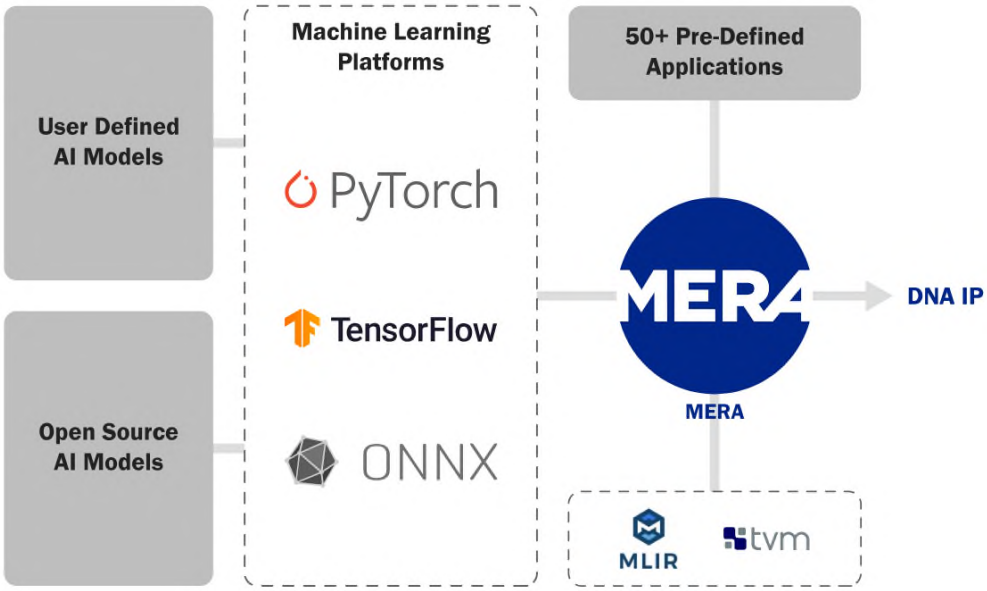

Рис. 1.4 – Екосистема інструментів.

Місце задачі детекції об’єктів. У межах цієї дипломної роботи детекція відіграє центральну роль як базова здатність системи «бачити» об’єкти в сцені й локалізувати їх у просторі пікселів. Її успішність визначає роботу відстеження, підрахунку, взаємодії людина-машина чи тригерів бізнес-логіки (наприклад, технічного огляду на конвеєрі). Історично детекція пройшла шлях від класичних каскадів (Viola–Jones) через регіон-пропозиційні CNN до одноетапних підходів (YOLO) та трансформерних архітектур (DETR), що задають сьогоднішній простір компромісів між точністю, швидкістю й ресурсами [7; 8; 3; 4]. Саме ця еволюція буде критично важливою для методологічних виборів у подальших розділах (вибір датасету, метрик, архітектури моделі, стратегії тюнінгу й розгортання).

У сучасній науковій літературі виділяють низку базових задач комп’ютерного зору, що формують основу більш складних сценаріїв аналізу зображень і відео.

1. Класифікація зображень задача віднесення вхідного зображення до одного з попередньо визначених класів. Система має відповісти на запитання *«що зображено?»*. Класичні приклади – ImageNet Challenge, де алгоритм отримує зображення й визначає, наприклад, що воно

містить собаку або автомобіль [1; 2]. Класифікація є фундаментальною задачею, оскільки вона лежить в основі більшості інших завдань.

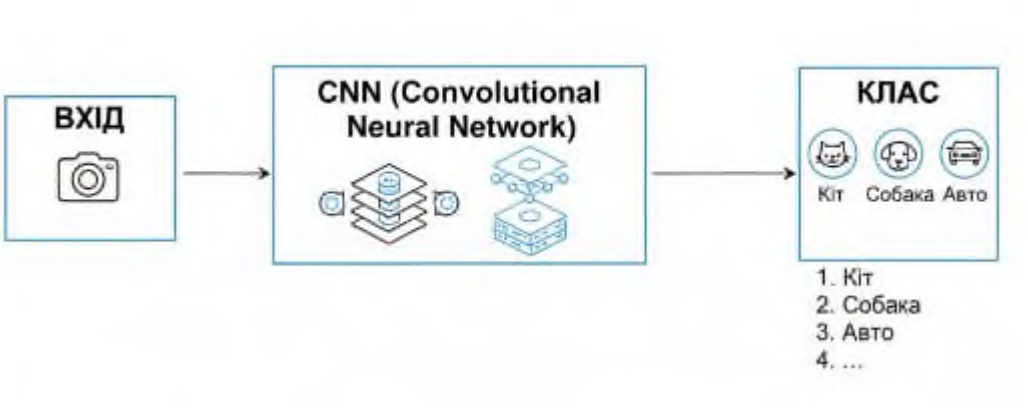

Рис. 1.5 – Схема класифікації зображень.

2. Детекція об’єктів на відміну від класифікації, тут завдання полягає не лише в ідентифікації класу, а й у локалізації об’єкта у вигляді обмежувальної рамки (bounding box). Це дозволяє обробляти зображення, що містять декілька об’єктів одночасно. Сучасні детектори (YOLO, SSD, Faster R-CNN) виконують цю задачу з високою точністю в реальному часі [3; 7].

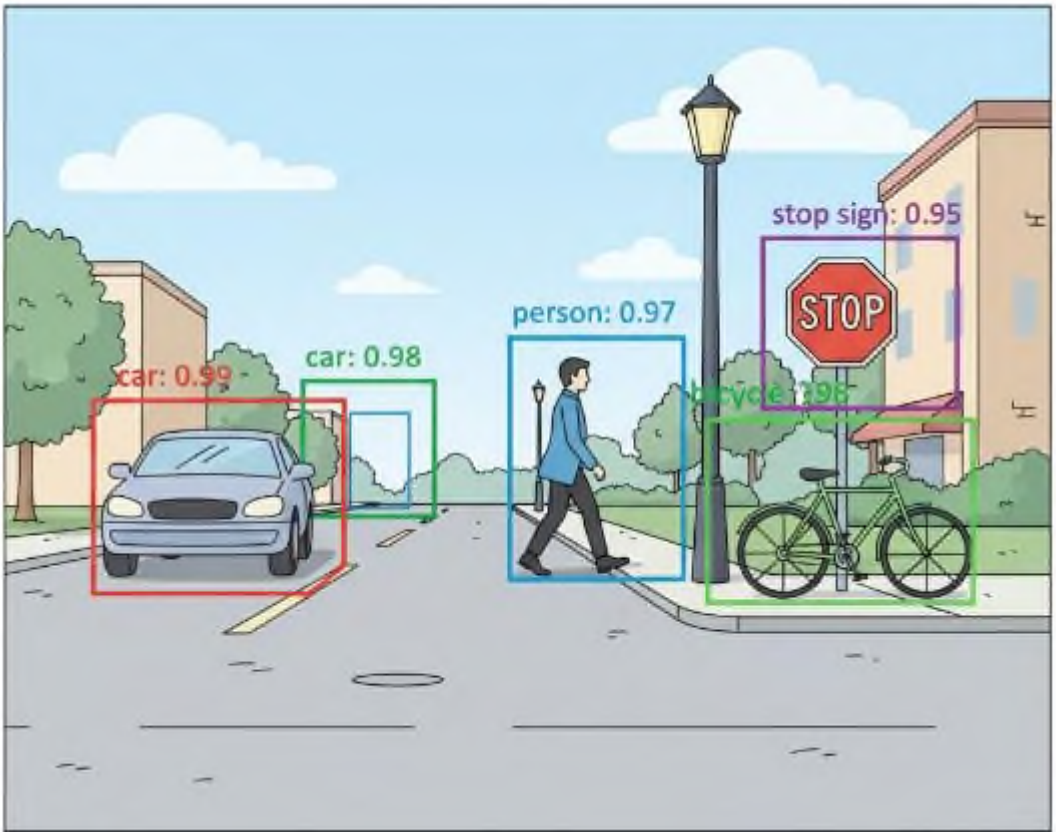

Рис. 1.6 – Приклад детекції об’єктів із bounding boxes.

3. Сегментація зображень є більш «тонкою» задачею, яка вимагає піксельного поділу сцени.
    - *Семантична сегментація* розділяє пікселі на класи (наприклад, «дорога», «будівля», «людина»), але не розрізняє окремі екземпляри.
    - *Інстанс-сегментація* виділяє кожен об’єкт як окремий екземпляр.
    - *Паноптична сегментація* поєднує обидва підходи [3; 8].
    - Ця задача критично важлива для аналізу складних сцен, зокрема в медицині та автономному водінні.
4. Трекинг (відстеження) об’єктів полягає у відстеженні руху об’єкта в послідовності кадрів. Використовується як надбудова над детекторами для роботи з відео. Алгоритми трекінгу є основою для систем відеоспостереження, розпізнавання жестів і взаємодії людина–машина [9].
5. Оцінка поз та ключових точок передбачає визначення розташування суглобів і частин тіла людини або об’єкта. Це особливо важливо для спорту, медицини, AR/VR-систем [10].

Комп’ютерний зір вийшов далеко за межі лабораторних експериментів і сьогодні активно інтегрується в критично важливі сфери людської діяльності. Його використання охоплює широкий спектр завдань – від безпеки й медицини до транспорту та промислової робототехніки.

Одним із найбільш поширених напрямів застосування є безпекові технології. Системи відеоспостереження з функціями комп’ютерного зору дають змогу:

- автоматично ідентифікувати обличчя людей у режимі реального часу, що використовується для доступу до приміщень та криміналістики;
- виявляти підозрілу поведінку (наприклад, біг у зоні аеропорту чи залишений багаж у громадському місці), що підвищує превентивну складову безпеки;

- контролювати периметр за допомогою алгоритмів трекінгу й класифікації, відрізняючи транспортні засоби від пішоходів.

У сучасних містах такі системи інтегрують у концепцію «розумного міста», де відеоаналітика допомагає регулювати трафік і виявляти правопорушення на дорогах [11; 18].

1.2. Методи та алгоритми детекції об’єктів

Історичний розвиток детекції об’єктів спирався на рукотворні (hand-crafted) ознаки та модульні конвеєри, де кожен етап – від попередньої обробки до усунення дублювань – проєктувався окремо. Типовий пайплайн включав побудову масштабної піраміди, щоб забезпечити квазі-масштабну інваріантність; згладжування або вирівнювання освітлення для зменшення чутливості до фотометричних змін; сканування сцени рухомим вікном для великої кількості позицій, масштабів і аспектних відношень; обчислення локальних або квазі-глобальних дескрипторів; подачу їх до окремого класифікатора (як-от SVM або бустингові ансамблі); а також фазу післяобробки з non-maximum suppression (NMS), що відсікає перекривні передбачення, зберігаючи детекцію з найбільшою довірою.

Поворотним моментом у вузькодоменній детекції (насамперед облич) стала поява каскаду Віоли–Джонса. У цій схемі прямокутні Haar-подібні шаблони, що вимірюють контраст між світлими та темними регіонами, обчислюються надзвичайно швидко завдяки інтегральному зображенню, яке дає можливість отримати суму яскравостей у довільному прямокутнику за сталий час. Бустинговий алгоритм Adaboost послідовно відбирає «сильні» ознаки з великого пулу слабких класифікаторів і впорядковує їх у каскад: більшість негативних вікон відсікаються на ранніх стадіях за мінімальної обчислювальної вартості, тоді як лише невелика підмножина «підозрілих» регіонів проходить до глибших етапів із дорожчими перевірками [25].

Практичний ефект – легко відтворювані системи детекції облич у реальному часі на CPU ще на початку 2000-х, що й сьогодні зустрічаються в

контрольних терміналах. Водночас така модель виявляє вроджену чутливість до змін освітлення, поз і поворотів голови, а також до складних фонових текстур; ці обмеження, разом із фіксованою формою ознак, згодом стали каталізатором переходу до більш виразних дескрипторів.

Ідея градієнтних гістограм, реалізована в HOG (Histogram of Oriented Gradients), запропонувала більш стійке представлення локальної форми. Ключовими елементами є розбиття кадру на комірки з підрахунком орієнтованих градієнтів, подальша нормалізація у більших блоках для інваріантності до змін освітлення та контрасту, а також лінійний SVM як класифікатор над конкатенованим дескриптором. Типові параметри (розмір комірки 8×8 пікселів, блок 2×2 комірки, 9 бінів орієнтацій) народжені емпіричним пошуком компромісу між деталізацією та стійкістю. Саме HOG+SVM на багато років став «золотим стандартом» для детекції пішоходів і транспортних засобів у міських сценах: у цих доменах структурованість об’єктів і відносна регулярність фонів сприяли високій відтворюваності результатів [22]. Однак масштабування цього підходу на великі множини класів неминуче наштовхувалося на експоненційне зростання обчислень у sliding-window та на потребу в ручному дизайні інваріантностей.

Розвиток локальних інваріантних ознак на кшталт SIFT (Scale-Invariant Feature Transform) приніс у детекцію ідентифікацію стійких ключових точок і опис їхніх околів. Через екстремуми в просторі різниць Гауса (DoG) SIFT віднаходить точки, що добре локалізуються як у просторі, так і у масштабі; орієнтація визначається локальною домінантною напрямленістю градієнтів, після чого формується дескриптор як гістограма градієнтів у регулярній сітці навколо точки з урахуванням обертової інваріантності. У задачах відповідностей і категоризації SIFT слугував базовим «словником» для побудови bag-of-visual-words, а в детекції використовувався через схеми vote-based локалізації (Hough-подібні акумулятори), хоча повноцінна робота в умовах великої міжкласової варіативності та значних оклюзій вимагала

додаткових евристик [23]. SURF (Speeded-Up Robust Features) зменшив обчислювальні витрати, замінивши згортки на інтегральні зображення й використавши апроксимації лапласіана Гауса через фільтри Хессе; дескриптор SURF побудований на інтегральних сумах відгуків Haar-подібних фільтрів, що зберігає кращу продуктивність на реальних системах із обмеженим бюджетом часу [24]. У підсумку SIFT/SURF забезпечували добру стійкість до маштабування, помірних афінних деформацій і часткових перекриттів, але вимагали акуратної настройки порогів відповідностей, відсікання outlier-ів (RANSAC) і були чутливими до текстурної «бідності» сцен.

Важливу роль у «класичній» епосі відіграли також деформаційні моделі частин (DPM), де об’єкт репрезентувався набором частин із HOG-ознаками, розміщених навколо «кореня», а відхилення від очікуваних відносних положень каралися деформаційними штрафами. Така явна моделювальність геометрії покращувала точність для категорій із виразною частинною структурою (наприклад, людина, автомобіль), але ускладнювала оптимізацію, збільшувала число гіперпараметрів та час інференсу [28]. Узагальнюючи, «класика» сформувала чіткий науковий канон: інваріантності проектують; ознаки та класифікатор навчають роздільно; контекст вводять обмежено; а швидкість на CPU досягають за рахунок геометричних евристик і каскадів. Цей канон дозволяв будувати робочі системи в вузьких доменах (обличчя, пішоходи, номерні знаки), але погано масштабувався на багатокласні, насичені перешкодами сцени.

Поява глибинних згорткових мереж (CNN) змінила парадигму: замість проєктування ознак вручну почали навчати представлення безпосередньо з даних. Перші згорткові архітектури для розпізнавання рукописних цифр (LeNet-5) окреслили принципи локальних рецептивних полів, згорток, субдискретизації та спільного навчання ознак із класифікатором [27]; проте саме AlexNet, навчена на великомасштабному ImageNet, показала якісно новий рівень узагальнюваності й стала поворотною точкою для всієї

спільноти комп’ютерного зору [11]. Стало зрозуміло, що ієрархії фільтрів, навчені на великих корпусах, можна переносити на інші задачі (transfer learning), зокрема на детекцію, замінюючи ручні дескриптори на CNN-ознаки [2]. Вектор розвитку відтоді визначили питання: як поєднати регіональні гіпотези з потужними CNN-представленнями, не втрачаючи швидкості?

Першою відповіддю став R-CNN: для кожного зображення позакадровий метод Selective Search формує приблизно дві тисячі кандидатних регіонів, кожен нормалізується до фіксованого розміру і пропускається через CNN; отримані ознаки подаються на SVM-класифікатори за класами, а регресори уточнюють положення рамок [12]. Така схема радикально покращила mAP на VOC/ILSVRC, але розплачувалася велетенською обчислювальною вартістю (тисячі пропускань CNN на кадр) та складним багатостадійним тренуванням. Fast R-CNN знімає головне «вузьке місце», виконуючи один прохід CNN над усім зображенням для отримання спільної карти ознак; для кожної пропозиції застосовується операція RoI Pooling, що екстрагує фіксованого розміру тензор, який далі спільною головою вирішує класифікацію і регресію рамок [26]. Комбінація спільних ознак і RoI-пулінгу різко підвищує швидкість інференсу й спрощує навчання, але все ще залежить від зовнішнього генератора пропозицій.

Faster R-CNN робить вирішальний крок: генерацію пропозицій інтегровано в мережу як Region Proposal Network (RPN). На тій самій карті ознак застосовуються ковзні вікна з набором «якорів» (anchors) різних масштабів та аспектних відношень; невелика голова RPN навчається класифікувати ці анкори на «об’єкт/фон» і регресувати зміщення рамок. Пайплайн стає справді end-to-end: спільні ознаки обслуговують і генерацію пропозицій, і детекцію, а загальна оптимізація покращує і точність, і час [13]. Далі надбудови у вигляді Feature Pyramid Networks (FPN) дозволяють поєднувати багатомасштабні представлення, покращуючи чутливість до дрібних об’єктів, тоді як Mask R-CNN вводить RoI Align для точнішої геометрії і додає гілку сегментації інстансів [15]. У паралельній гілці

еволюції сформувалася лінія одноетапних детекторів: SSD трактує детекцію як пряму регресію рамок і класів на багаторівневих картах ознак, відмовляючись від явних пропозицій; RetinaNet компенсує дисбаланс позитивів/негативів за допомогою функції втрат Focal Loss; а YOLO конструює просторову решітку й відразу прогнозує координати, розміри та класи в один прохід, що робить можливими застосування з жорсткими обмеженнями латентності [7, 32, 33].

Усі ці напрями підсилюють центральну ідею: від ознак, спроєктованих людиною, – до представлень, що навчаються на великих різнорідних даних, з інтегрованою геометрією, контекстом і оптимізацією всієї задачі.

Стандарт де-факто задають відкриті бенчмарки PASCAL VOC і MS COCO, де якість оцінюють через середню точність (AP) і середню по класах (mAP) при різних порогах перетину IoU між істинними та передбаченими рамками [5; 6]. Рукотворні конвеєри – HOG+SVM, каскади Haar і DPM – демонструють гідні результати у вузьких доменах із відносно стабільними умовами знімання (пішоходи на фіксованій відстані, фронтальні обличчя), однак на багатокласних наборах із різкими змінами масштабу, освітлення, з оклюзіями й текстурним шумом їхні можливості вичерпуються: ознаки не здатні «підлаштуватися» під нетипові екземпляри, а контекст уводиться лише через жорсткі геометричні евристики [22–25; 28]. На тих самих наборах CNN-детектори (спершу двоетапні) системно перевищують «класику» у mAP, оскільки ієрархічні фільтри вчаться розрізняти семантичні патерни, що не виражаються простою локальною градієнтною статистикою; додаткові механізми на кшталт багатомасштабних пірамід ознак і точних операцій вирівнювання (RoI Align) підтримують точність на дрібних екземплярах і складній геометрії [12; 13; 15].

Швидкість висвітлює інший бік компромісу. На CPU каскади Haar і прості HOG-класифікатори в «вузьких» задачах забезпечують реальний час завдяки агресивному відсіканню й компактним дескрипторам [25]. Проте щойно зростає число класів, масштабів і аспектів, sliding-window стає

диспропорційно дорогим. У світі CNN перша реалізація R-CNN практично непридатна для онлайн-сценаріїв через тисячі проходів мережі на кадр [12], тоді як Fast/Faster R-CNN завдяки спільним ознакам і RPN різко скорочують латентність і стають прийнятними для near-real-time аналітики на GPU [26; 13]. Якщо ж вимоги передбачають гарантований FPS на відеопотоці або виконання на edge-пристроях, то одноетапні схеми (YOLO/SSD/RetinaNet) пропонують найкраще співвідношення швидкість/якість: відмова від явних пропозицій і повний «однопрохідний» дизайн мінімізують затримки, а сучасні оптимізації (квантування, прунінг, знаннєва дистиляція) додатково зменшують обчислювальний слід [7, 32, 33].

Під універсальністю тут розумітимемо здатність переноситися між доменами, масштабуватися на нові класи й режими знімання, а також інтегруватися в промисловий цикл розробки. Рукотворні ознаки надають чіткі інваріантності – до повороту, маштабу, деяких фотометричних змін – однак їхня виразність обмежена: при зміні домену зазвичай доводиться перебудовувати набір ознак, пороги, іноді навіть архітектуру конвеєра. Навпаки, CNN-представлення від природи «дано-керовані»: початкові шари кодують універсальні локальні патерни, які добре трансферяться, а вищі – адаптуються під цільову задачу за допомогою донавчання на відносно невеликих корпусах (fine-tuning). Додавання багатомасштабних рівнів (FPN), навчання зі збалансувальними втратами (Focal Loss), використання сучасних бекбонів (VGG/ResNet) покращує переносимість і стабільність у складних сценах [2; 13; 15; 29; 30; 31; 33]. Важливо й те, що CNN-детектори органічно вписуються в MLOps-практики: їх можна версіонувати разом із датасетами, відстежувати дрейф даних, застосовувати активне навчання для доповнення «важких» прикладів і автоматизувати розгортання на різних апаратних цілях (GPU/TPU/NPU) за допомогою ONNX/TensorRT тощо, тоді як «класика» рідше має такі стандартизовані контури.

Узагальнюючи порівняння, можна сформулювати практичне правило вибору. Якщо задача вузька, стабільна, з простою геометрією й жорсткими

обмеженнями на обчислення на CPU (наприклад, базова перевірка наявності облич у кадрі для доступу), «класичні» каскади або HOG-класифікатори залишаються конкурентними через детермінованість і мінімальний слід. Якщо ж сцена багата на класи, містить дрібні та частково перекриті об’єкти, фон змінний, а вимоги точності високі, то двоетапні CNN-детектори (Faster R-CNN із сучасним бекбоном та FPN) забезпечують найкращу якість при прийнятній латентності на GPU. Нарешті, для систем, де визначальним є FPS і обмежені ресурси (відеоаналітика в реальному часі, мобільні/edge-рішення), домінують одноетапні підходи на кшталт YOLO/SSD/RetinaNet, які оптимізують проходження даних і зводять до мінімуму «вузькі місця» обчислень [7; 13; 15; 32; 33].

1.3. Моделі сімейства YOLO (You Only Look Once)

Концепція YOLO постала як радикальна зміна парадигми детекції: замість багатостадійного конвеєра з окремими модулями генерації пропозицій, ознак і класифікації запропоновано формулювати задачу як єдину регресійну проблему, що безпосередньо відображає пікселі зображення у простір параметрів обмежувальних рамок та ймовірностей класів – в один прямий прохід мережі [7]. Ключова ідея полягає в сітковій дискретизації простору зображення: воно ділиться на решітку S×S, і кожна комірка відповідає за прогнозування одного чи кількох кандидантів-об’єктів поблизу свого центру. Для кожного кандидата мережа повертає параметри рамки (координати центра, ширину і висоту в нормованих одиницях відносно комірки або всього зображення, залежно від варіації моделі), об’єктність (objectness, тобто оцінку наявності об’єкта в цій рамці) та вектор апостеріорних імовірностей класів, які у поєднанні з об’єктністю утворюють ранги детекцій. Набір усіх передбачень з різних комірок та масштабів декодується у сукупність рамок, а потім агрегується за допомогою NMS (або його модифікацій) для усунення надлишкових перекриттів [7; 35]. Така одноетапність – від пікселів до фінальних детекцій за один фідфорвард –

забезпечує принципову перевагу у швидкодії та робить YOLO придатною для застосувань у реальному часі, включно з відеопотоками та edge-пристроями.

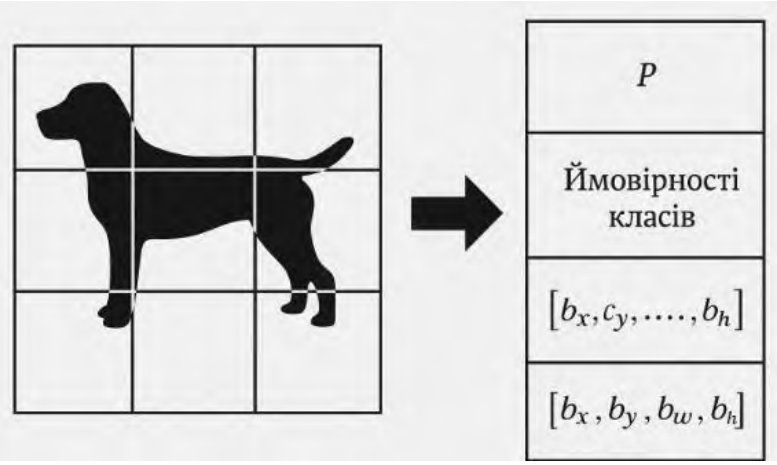

Рис. 1.7 – Сіткова формалізація YOLO та схема виводу.

Початковий варіант YOLOv1 заклав саме цю сіткову постановку та вартісну функцію, що поєднувала помилки локалізації та класифікації у спільній квадратичній метриці, із підвищеною вагою для координатних членів, аби зменшити чутливість до невеликих зміщень рамок [7]. Уже в YOLO9000 (YOLOv2) було усунуто низку обмежень першої версії (зокрема «один об’єкт на комірку»), а також введено якірні рамки (anchors), що дозволило кодувати апріорні форми об’єктів і відокремити задачу класифікації від регресії геометрії [36]. Для вибору форм анкорів використано k-means кластеризацію за шириною і висотою істинних рамок, що збалансовувало набір шаблонів під статистику датасету. Важливою стала ідея direct location prediction з логістичною параметризацією зміщень усередині комірки, що стабілізувало тренування. На етапі підготовки даних застосовано мультишкальне тренування: розмір вхідного зображення випадково змінювався з певною періодичністю, завдяки чому модель навчалася бути робастною до масштабних варіацій – критично для детекції дрібних і великих об’єктів у межах однієї сцени [36].

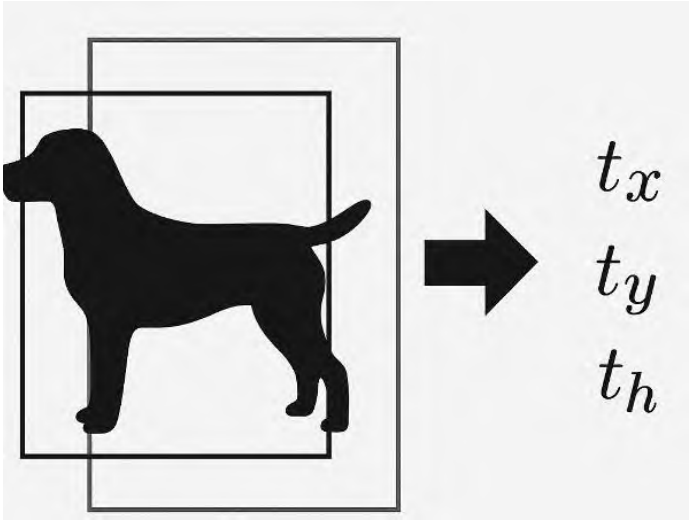

Рис. 1.8 – Анкорна параметризація рамок у YOLOv2.

Подальша YOLOv3 розвинула ідею багатомасштабності у самій архітектурі голови детектора: передбачення виконуються на кількох рівнях просторової роздільності (звично три), що покращує виявлення дрібних об’єктів, які губляться на грубих картах ознак. Бекбон Darknet-53 із залишковими зв’язками (residual) забезпечив кращу глибинну репрезентативність за помірної обчислювальної вартості, а бінарні крос-ентропійні втрати для класифікації та об’єктності підвищили стабільність навчання при великій кількості негативних прикладів [37]. Цей дизайн – «backbone–neck–head» із акцентом на багатомасштабних картах – по суті став канонічним для одноетапних детекторів: бекбон екстрагує ієрархічні ознаки, шейпер (neck) (часто реалізований як FPN/PAN) зшиває інформацію з різних рівнів, а голова на кожному рівні простору одночасно регресує рамки та класи [31; 37].

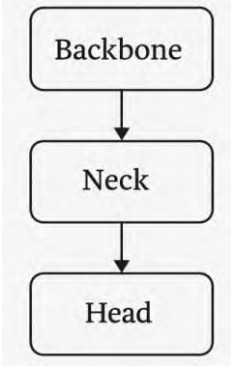

Рис. 1.9 – Архітектурний макропатерн YOLO.

Наступні ітерації – як-от YOLOv4 – систематизували «мішки безкоштовностей» і «мішки спеціальних прийомів» (bag-of-freebies / bag-of-specials), що підвищують точність без істотного зростання латентності. Серед них – SPP-блоки для збільшення рецептивного поля без втрати роздільності, CSP-модулі для зниження надлишковості градієнтів і покращення розподілу обчислень, MixUp/Mosaic-аугментації, відкладена нормалізація, лейбл-смузинг тощо [38; 44]. На етапі постобробки поряд із класичним NMS дедалі частіше використовують Soft-NMS для пом'якшення агресивного придушення перекриттів, що особливо корисно у щільних сценах [35]. Важливо, що й функції втрат для регресії рамок еволюціонували від евклідових до IoU-метрик: GIoU розв'язує проблему нульового градієнта при неперекривних рамках, а DIoU/CIoU вводять відстань між центрами та співвідношення сторін, прискорюючи збіжність і покращуючи локалізацію [41; 42].

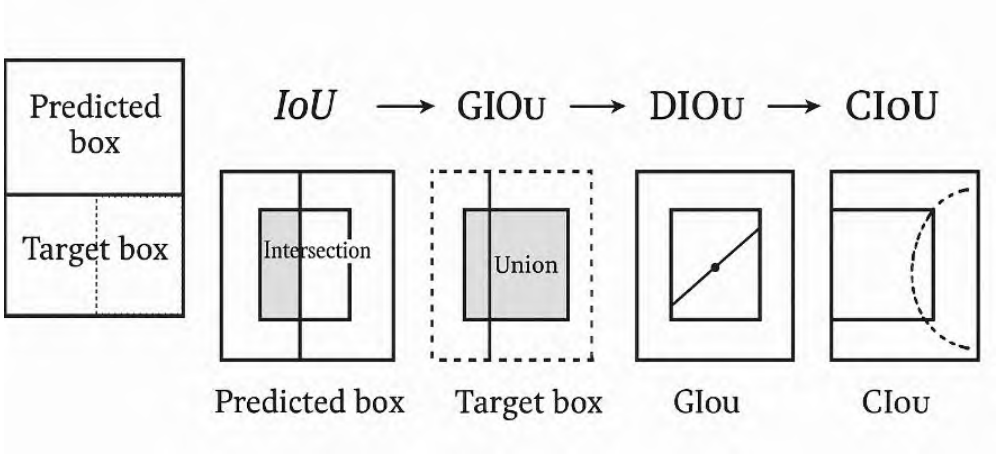

Рис. 1.10 – Порівняння геометричних втрат.

Попри відсутність формальної рецензованої статті, лінія YOLOv5 від Ultralytics закріпила інженерні практики продакшн-рівня: зручний тренувальний CLI, модульність конфігурацій, експортування у формати ONNX/TensorRT, а також прораховані пресети гіперпараметрів і аугментацій (включно з Mosaic) для типових датасетів [39]. Подальші публікації, зокрема YOLOv7, запропонували вдосконалення тренувального процесу (re-parametrization у тестовому часі, планування оптимізації) і «навчальні» безкоштовності (bag-of-freebies), що встановлювали нові співвідношення

швидкість/точність у широкому діапазоні апаратних ресурсів [40]. У новіших реалізаціях (наприклад, YOLOv8) активно вивчаються безанкорні (anchor-free) голови з прямим прогнозуванням центрів і масштабів відносно картини ознак, що зменшує залежність від ручного вибору анкорів і спрощує перенос між доменами та розмірами об’єктів [39]. Незалежно від варіацій, архітектурне ядро YOLO зберігається: одноетапна голова, що працює на кількох просторових масштабах, безпосередньо повертає множину детекцій, які після швидкої постобробки утворюють остаточний набір рамок та класів у реальному часі.

З погляду теорії компромісів, YOLO ілюструє спосіб досягнення високої пропускної здатності без кардинальної втрати точності. Відмова від явного модулю пропозицій та набагато менша кількість проміжних перетворень зменшують латентність та пам’ятковий слід, а багаторівнева семантика (через FPN/PAN) частково компенсує втрати, пов’язані з грубою дискретизацією сіткою. Одночасно сучасні функції втрат, багаті аугментації та каскадні евристики постобробки (наприклад, пороги об’єктності, Soft-NMS) дозволяють підтримувати високу mAP на COCO/VOC при FPS, недосяжних для класичних двоетапних підходів у рівних апаратних умовах [6; 31; 35; 38; 40]. Зауважмо також, що проєктний простір YOLO дуже інженерний: реальна ефективність моделі нерідко визначається тонким налаштуванням пакета тренування (розміри батчу, політики LR, схеми аугментацій), постобробки (тип і параметри NMS), а також компіляцією/експортом під цільову платформу (ONNX, TensorRT, CoreML тощо) [39]. Саме ця поєднаність наукових і інженерних рішень і пояснює стійку популярність сімейства YOLO у задачах від відеоаналітики до робототехніки, автономного транспорту та мобільних застосунків.

Висновки до розділу 1

У першому розділі розглянуто теоретичні та методологічні основи комп’ютерного зору, а також проведено системний аналіз методів і

алгоритмів детекції об’єктів. Показано, що розвиток галузі відбувався від класичних алгоритмів, орієнтованих на використання ручних ознак (Viola–Jones, HOG, SIFT, SURF, DPM), до сучасних глибинних нейронних мереж, які здатні автоматично формувати багаторівневі ознаки й досягати високої узагальнювальної здатності. Класичні методи залишаються конкурентними у вузьких сценаріях із низькими вимогами до масштабованості, проте вони не забезпечують необхідної універсальності й точності для складних багатокласових сцен.

Особливу увагу приділено аналізу моделей сімейства YOLO, які завдяки одноетапній архітектурі дозволяють виконувати детекцію у реальному часі, поєднуючи швидкість обробки з достатньо високими показниками точності. У роботі охарактеризовано принципи функціонування моделей YOLO, їхню еволюцію від першої версії до сучасних реалізацій (YOLOv5, YOLOv7, YOLOv8), а також визначено основні технічні рішення, які зумовили їхню популярність (анкорні механізми, багаторівнева обробка ознак, сучасні функції втрат, оптимізації під GPU та edge-пристрої).

У межах першого розділу встановлено, що використання YOLO є доцільним для розробки системи детекції об’єктів завдяки її збалансованості за критеріями точності, продуктивності та можливості практичного впровадження. Це створює науково-методичне підґрунтя для подальших етапів дослідження та реалізації проєкту.

РОЗДІЛ 2. МЕТОДОЛОГІЯ ТА ІНСТРУМЕНТИ РОЗРОБКИ СИСТЕМИ

2.1. Вимоги до системи детекції об’єктів

Формулювання вимог є ключовим етапом при проєктуванні будь-якої інформаційної системи, адже саме вони визначають архітектуру, функціональність та умови подальшої експлуатації. Для системи детекції об’єктів ці вимоги можна умовно поділити на функціональні та нефункціональні, що відповідає класичним підходам інженерії програмного забезпечення [45]. Визначення обох груп необхідне, оскільки перша забезпечує відповідність результатів очікуваним сценаріям використання, а друга – гарантує продуктивність, надійність і можливість масштабування системи.

До функціональних вимог належать ті, що безпосередньо визначають поведінку системи та перелік її обов’язкових можливостей. Передусім система повинна здійснювати виявлення об’єктів на зображеннях і відеопотоці, повертаючи координати обмежувальних рамок та відповідні класи. Це базова умова, що відрізняє її від систем класифікації чи сегментації. Для забезпечення адекватності результатів необхідною є можливість роботи з кількома класами об’єктів, що дозволяє застосовувати систему в реальних багатосценарних середовищах – від транспортних вузлів до медичних застосунків [46].

Крім того, система повинна забезпечувати обробку вхідних даних у різних форматах (JPEG, PNG, MP4, AVI тощо), що дає змогу інтегрувати її з різноманітними сенсорними платформами та інформаційними системами. Важливим функціоналом є також постобробка детекцій, включно з алгоритмами non-maximum suppression або їхніми модифікаціями (Soft-NMS), що зменшують кількість помилкових дублювань [35]. Для адаптації до різних середовищ система має підтримувати навчання на користувацьких наборах даних з можливістю тонкого налаштування гіперпараметрів та

інтеграції нових класів без повного перенавчання моделі [39]. Нарешті, функціональною вимогою є можливість візуалізації результатів у вигляді графічних інтерфейсів або API-відповідей, що інтегруються в зовнішні системи відеоаналітики чи мобільні застосунки.

Нефункціональні вимоги визначають якісні характеристики системи, що забезпечують її відповідність сучасним стандартам продуктивності та зручності використання. Передусім критичною є швидкодія, адже у багатьох сценаріях (відеоспостереження, автономний транспорт, робототехніка) система має працювати у реальному часі. У практиці детекції це означає підтримку щонайменше 30 FPS на стандартних GPU і понад 10 FPS на edge-пристроях [38; 40]. Разом зі швидкодією важливою є точність, яка традиційно вимірюється через метрики mAP@[.5:.95] на контрольних наборах COCO або VOC [6]. Вимога щодо високої точності пов’язана із забезпеченням мінімальної кількості помилкових позитивних і негативних спрацьовувань, що має особливе значення у критичних сферах (медицина, безпека).

Іншою нефункціональною вимогою є масштабованість: система повинна працювати як у режимі окремого застосунку на персональному комп’ютері, так і в розподілених середовищах із хмарними обчисленнями. Для цього необхідна підтримка інтерфейсів REST/GRPC та можливість контейнеризації у Docker чи Kubernetes [47]. Важливою характеристикою є портативність, тобто здатність адаптуватися до різних апаратних платформ: від високопродуктивних графічних прискорювачів до мобільних CPU та спеціалізованих NPU. Саме ці вимоги зумовили появу полегшених варіацій YOLO («tiny», «nano») з меншими параметрами та оптимізацією під TensorRT чи CoreML [37; 39].

Надійність і відмовостійкість є ще одним обов’язковим критерієм: система повинна зберігати працездатність навіть за наявності часткових втрат кадрів чи нестабільного мережевого з’єднання. Це досягається за рахунок кешування проміжних результатів, багаторівневої обробки помилок та інтеграції з журналюванням подій [48]. Важливим нефункціональним

параметром є також зручність інтеграції: розробники очікують наявності документації, SDK та підтримку основних мов програмування (Python, C++, Java), що відповідає принципам сучасної інженерії штучного інтелекту.

Узагальнення

Система детекції об’єктів повинна відповідати не лише базовим функціональним вимогам (ідентифікація та локалізація об’єктів, мультикласовість, візуалізація результатів), але й дотримуватися суворих нефункціональних параметрів (швидкодія, точність, масштабованість, портативність, відмовостійкість). У сукупності вони формують комплексний портрет системи, здатної ефективно функціонувати у різних сферах – від промислових виробництв до автономних транспортних систем. Умовно ця структура вимог може бути представлена як двошарова діаграма, де перший шар відповідає за функціональні аспекти, а другий – за нефункціональні.

Рис. 2.1 – Модель вимог до системи детекції об’єктів.

Обґрунтування вибору платформи для реалізації системи детекції об’єктів є визначальним чинником, оскільки саме апаратне та програмне середовище зумовлює можливості щодо швидкодії, точності, масштабованості та інтеграційної сумісності. Сучасні системи детекції

здебільшого розгортаються на основі глибинних згорткових нейронних мереж (CNN) та їхніх модифікацій, що потребують значних обчислювальних ресурсів під час навчання та оптимізації [7; 13; 38]. Саме тому одним із базових виборів є використання GPU-орієнтованих платформ, зокрема NVIDIA CUDA, які забезпечують ефективний паралельний обрахунок операцій згортки та матричних множень [49].

Не менш важливим є аспект сценаріїв використання. Якщо система орієнтована на стаціонарну відеоаналітику, наприклад, у сфері безпеки або контролю виробництва, то доречним є розгортання на серверних середовищах із потужними графічними прискорювачами та можливістю масштабування у хмарних інфраструктурах (AWS, Google Cloud, Azure), що дозволяє обробляти великі потоки відео у реальному часі. У випадку ж мобільних та робототехнічних застосунків постає потреба у платформах типу NVIDIA Jetson, Google Coral або інших edge-рішень, які оптимізовані для енергоефективного виконання моделей глибинного навчання [50]. Це пояснюється тим, що у безпілотних автомобілях, дронах чи сервісних роботах критично важливим є баланс між продуктивністю та споживанням енергії, а також мінімізація латентності, адже будь-яке затримання у прийнятті рішення може призвести до аварійної ситуації. В обох сценаріях ключовим стає використання фреймворків високого рівня, зокрема PyTorch чи TensorFlow, що підтримують апаратні оптимізації, та бібліотек OpenCV для попередньої обробки зображень [46; 47].

Важливо також зазначити обмеження, які накладаються на систему з погляду продуктивності, ресурсів та обсягу даних. По-перше, продуктивність моделі оцінюється не лише за FPS, а й за затримкою (latency) та пропускною здатністю у багатопотокових сценаріях [38]. Навіть якщо модель працює на високій частоті кадрів, але затримка між надходженням вхідного зображення і видачею результату перевищує кілька сотень мілісекунд, така система може бути непридатною для критичних застосунків (наприклад, автономного водіння). По-друге, ресурси, необхідні для навчання, включають не лише

GPU, але й оперативну пам’ять, накопичувачі для зберігання даних, а також пропускну здатність мережевих каналів. Великі набори даних (COCO, Open Images) мають обсяг у сотні гігабайтів, а процес навчання вимагає значних дискових операцій, що потребує SSD-накопичувачів високої швидкості [6]. По-третє, обсяг даних прямо впливає на якість узагальнення: моделі YOLOv3–YOLOv7 демонструють високі результати за умови навчання на різноманітних наборах, однак для специфічних застосунків (наприклад, медичні зображення) потрібне донавчання на спеціалізованих датасетах, що часто є обмеженими за кількістю [19; 40].

Додатковим обмеженням є енергоспоживання: для мобільних платформ важливо, щоб система могла функціонувати тривалий час без підзарядки. Тому виникає потреба у використанні спрощених моделей (YOLO-tiny, MobileNet-SSD) або оптимізацій, таких як квантування ваг чи прунінг, що зменшують розмір моделі та кількість обчислень [39; 42]. Ще одним важливим аспектом є правові та етичні обмеження: системи відеоаналітики мають відповідати нормам захисту даних (GDPR, CCPA), особливо у випадках використання біометричних ознак [48].

Вибір платформи та сценаріїв використання визначається трикутником «точність – швидкість – ресурси». Серверні та хмарні рішення краще підходять для сценаріїв із високим навантаженням і широким спектром класів, тоді як edge-платформи орієнтовані на мобільність і низьку латентність. Обмеження продуктивності, ресурсів та обсягу даних повинні враховуватися на етапі проєктування, оскільки вони безпосередньо впливають на життєздатність системи. Концептуально ці залежності можна подати у вигляді трикутної діаграми, де вершини позначають точність, швидкодію та ресурси, а баланс між ними визначає реальні можливості системи.

Рис. 2.2 – Трикутник компромісів системи детекції.

2.2. Інструменти та середовище реалізації

Розробка сучасних систем детекції об’єктів базується на поєднанні мов програмування загального призначення, бібліотек для глибинного навчання, спеціалізованих фреймворків та інструментів для підготовки даних. Вибір цих засобів визначається необхідністю досягнення високої швидкодії, відтворюваності результатів і зручності інтеграції у виробничі середовища.

Основною мовою для розробки систем комп’ютерного зору сьогодні є Python, що пояснюється її синтаксичною простотою, широкою екосистемою наукових бібліотек і підтримкою спільноти [45]. Python інтегрує в собі інструментарій для роботи з масивами даних (NumPy), обробки таблиць (Pandas), візуалізації (Matplotlib, Seaborn) та, найважливіше, бібліотеки для побудови й навчання моделей глибинного навчання.

Дві найпоширеніші бібліотеки – TensorFlow і PyTorch. TensorFlow, розроблений Google Brain, забезпечує як високорівневі API для швидкого прототипування (Keras), так і низькорівневі інтерфейси для точного контролю над обчисленнями [51]. Його перевага полягає у потужних інструментах для розгортання у хмарних сервісах (TensorFlow Serving, TensorFlow Lite) та підтримці апаратних оптимізацій (TPU). Натомість PyTorch, створений Facebook AI Research, відзначається динамічними обчислювальними графами та більш інтуїтивною структурою, що зробило

його де-факто стандартом у наукових дослідженнях і прикладному навчанні моделей [52]. PyTorch забезпечує тісну інтеграцію з CUDA, підтримку бібліотек для оптимізації (TorchVision, TorchAudio) та простоту у відлагодженні моделей. У більшості сучасних реалізацій YOLO використовується саме PyTorch як базовий інструмент.

Для практичної роботи з моделями YOLO доцільним є використання Ultralytics YOLO, що пропонує готові конфігурації, попередньо навчені ваги, зручний CLI та API для тренування, тестування й експорту моделей у формати ONNX, TensorRT чи CoreML [39]. Цей фреймворк дозволяє уникнути глибокого занурення у внутрішні деталі архітектури й зосередитися на адаптації моделі до конкретного завдання. Крім того, завдяки модульній структурі Ultralytics YOLO легко інтегрується з іншими бібліотеками, такими як OpenCV чи Roboflow.

Roboflow є хмарною платформою для управління датасетами та автоматизації процесів розмітки, аугментації і конвертації даних у потрібний формат [53]. Вона забезпечує інтеграцію з різними фреймворками (YOLO, TensorFlow, Detectron2), а також можливість швидкого експорту даних у форматах Pascal VOC, COCO, YOLO Darknet. Завдяки цьому інструмент значно скорочує час підготовки даних, що є однією з найтрудомісткіших частин процесу.

Важливим інструментом у конвеєрі є OpenCV (Open Source Computer Vision Library), яка виконує роль універсальної бібліотеки для обробки зображень і відео [54]. Вона забезпечує функціонал для конвертації форматів, застосування фільтрів, обробки кадрів у реальному часі, виконання класичних алгоритмів (виділення контурів, гістограмні аналізи, перетворення Фур'є), що часто використовується як попередній етап перед передачею даних у глибинні моделі.

Якість детекційної системи безпосередньо залежить від якості розмітки даних. Для цього використовуються спеціалізовані засоби. Одним із найпоширеніших є LabelImg – десктопний застосунок із відкритим кодом,

який дозволяє створювати обмежувальні рамки навколо об’єктів і зберігати розмітку у форматах Pascal VOC чи YOLO [55]. Завдяки простоті та легкості використання цей інструмент набув популярності в академічних і навчальних проєктах.

Для більш масштабних завдань доцільно застосовувати CVAT (Computer Vision Annotation Tool), розроблений компанією Intel [56]. Це веб-орієнтований інструмент для розмітки великих наборів зображень та відео, який підтримує різні типи розмітки: обмежувальні рамки, полігони, лінії, ключові точки. CVAT орієнтований на командну роботу та включає можливості контролю якості, експорту в різні формати та автоматизацію завдяки інтеграції з моделями глибинного навчання. Таким чином, CVAT підходить для створення корпоративних пайплайнів розмітки, де необхідна масштабованість і розподіл ролей.

Вибір інструментів та середовища реалізації є критично важливим для побудови ефективної системи детекції. Python та бібліотеки глибинного навчання забезпечують гнучкість і високу продуктивність у навчанні моделей. Спеціалізовані фреймворки, як-от Ultralytics YOLO та Roboflow, значно скорочують час розробки, надаючи готові модулі для тренування, експорту та управління даними. OpenCV виступає універсальним інструментом для попередньої обробки даних, тоді як LabelImg і CVAT забезпечують високоякісну підготовку навчальних наборів, без якої неможливе досягнення високих результатів точності. Таким чином, вся екосистема інструментів формує цілісний технологічний стек, у якому реалізація моделі YOLO стає не лише можливою, а й оптимізованою для різних сценаріїв застосування.

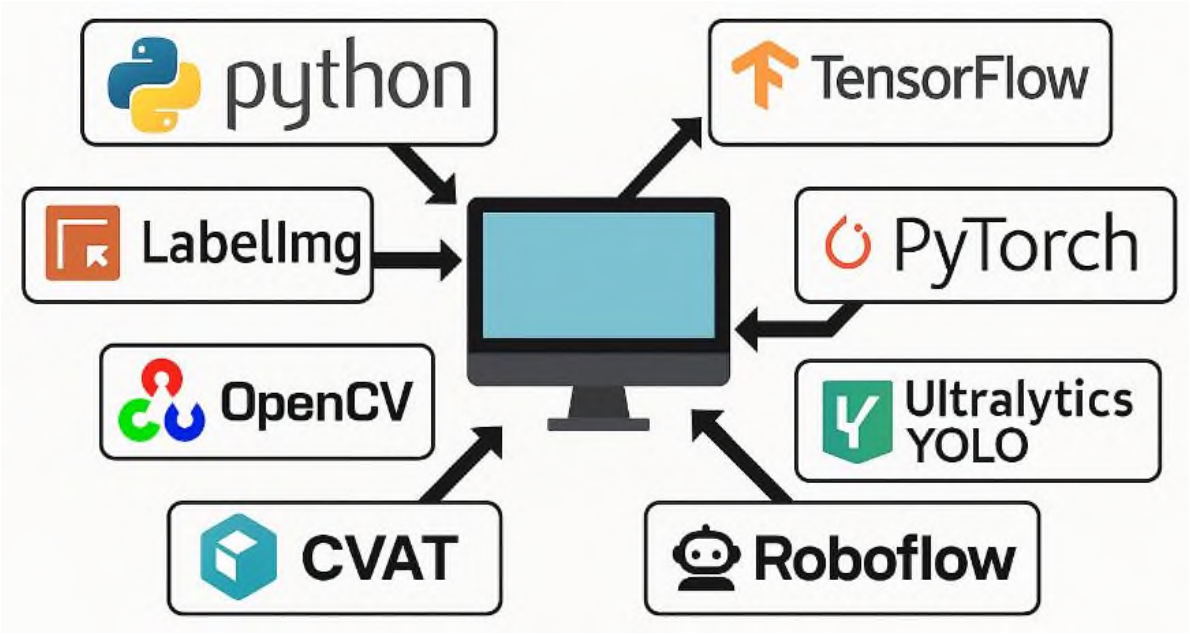

Рис. 2.3 – Екосистема інструментів для розробки системи детекції об’єктів.

2.3. Підготовка та обробка даних

Якість даних є одним із ключових чинників, що визначають ефективність системи детекції об’єктів. Відомо, що навіть найскладніші архітектури нейронних мереж не здатні компенсувати дефіцит або низьку якість даних, адже узагальнювальна здатність моделі напряму залежить від різноманітності та репрезентативності навчальних вибірок [46; 52]. Тому процес підготовки й обробки даних включає не лише технічні операції з формування наборів, але й вибір оптимальних джерел. Серед найбільш поширених є відкриті великомасштабні набори загального призначення (COCO, Pascal VOC), а також спеціалізовані доменно-орієнтовані колекції, які розробляються для конкретних галузей застосування.

Одним із найбільш впливових стандартів у сфері комп’ютерного зору є набір Pascal Visual Object Classes (VOC), започаткований у 2005 році [57]. Цей корпус даних містить зображення з багатокласовою розміткою, зокрема для завдань класифікації, сегментації та детекції об’єктів. Pascal VOC сприяв формуванню стандартів оцінювання, адже саме в його рамках уперше було

масово застосовано метрику Average Precision (AP) із порогом IoU=0.5, яка й сьогодні використовується у варіативних формах. Хоча обсяг цього набору (близько 20 тис. зображень, 20 класів) поступається сучасним, він залишається важливим навчальним ресурсом для тестування нових алгоритмів і методичних розробок.

```yaml
# Ultralytics 🚀 AGPL-3.0 License - https://ultralytics.com/license

# PASCAL VOC dataset http://host.robots.ox.ac.uk/pascal/VOC by University of Oxford
# Documentation: # Documentation: https://docs.ultralytics.com/datasets/detect/voc/
# Example usage: yolo train data=VOC.yaml
# parent
# ├── ultralytics
# └── datasets
#     └── VOC ← downloads here (2.8 GB)

# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ..]
path: VOC
train: # train images (relative to 'path') 16551 images
  - images/train2012
  - images/train2007
  - images/val2012
  - images/val2007
val: # val images (relative to 'path') 4952 images
  - images/test2007
test: # test images (optional)
  - images/test2007

# Classes
names:
  0: aeroplane
```

Рис. 2.4 – Зразки даних із Pascal VOC

Іншим базовим еталоном є COCO (Common Objects in Context) [6]. На відміну від VOC, COCO значно масштабніший (понад 330 тис. зображень і 80 класів) та характеризується контекстною складністю: об’єкти представлені у природному середовищі з високим рівнем варіативності масштабів, освітлення та оклюзій. Це робить COCO критичним для розвитку сучасних детекторів, адже він забезпечує умови, наближені до реальних сценаріїв. Крім того, COCO містить не лише обмежувальні рамки, а й маски для сегментації інстансів, а також анотації для завдань captioning і keypoint detection. Саме цей набір є основним у порівняльних дослідженнях продуктивності моделей YOLO, Faster R-CNN чи RetinaNet, оскільки він дозволяє отримати комплексні оцінки точності та швидкодії [6; 37].

Рис. 2.5 – Приклади зображень із COCO з розміткою рамок і масок.

Окрім еталонних наборів, у практичних розробках широко застосовуються спеціалізовані доменні колекції, орієнтовані на конкретні завдання. Так, у медицині популярними є набори ChestX-ray14 для детекції патологій легень [19], ISIC для діагностики меланоми, а також BraTS для сегментації пухлин головного мозку. У транспортній сфері використовуються KITTI та nuScenes, які містять дані з датчиків різних типів (RGB-камери, LiDAR, GPS) і орієнтовані на задачі автономного водіння [58]. У галузі аграрних технологій поширені датасети PlantVillage чи DeepWeeds, що допомагають ідентифікувати хвороби рослин і бур’яни [59]. Використання таких наборів дає змогу досягати високої точності в конкретних предметних галузях, однак постає проблема узагальнення моделей на нові середовища через вузьку специфіку даних.

Важливим етапом у роботі з даними є їхня попередня обробка, яка включає нормалізацію розмірів зображень, перетворення колірних просторів

(RGB, HSV), балансування класів та аугментації. У моделях YOLO, зокрема починаючи з версії YOLOv3, широко використовуються такі техніки, як масштабування збереженням пропорцій (letterbox), віддзеркалення, випадкове обрізання, зміна яскравості й контрасту [37]. У YOLOv4 було запроваджено Mosaic-аугментацію, яка поєднує чотири зображення в одне, дозволяючи моделі бачити об’єкти у різних масштабах та контекстах [38]. У результаті значно підвищується стійкість системи до зміни умов і збільшується узагальнювальна здатність, особливо за обмеженого обсягу даних.

Ключовим завданням є також конвертація форматів анотацій: Pascal VOC базується на XML-структурах, COCO – на JSON, а YOLO – на текстових файлах із координатами рамок у нормованому вигляді. Це зумовлює потребу у використанні проміжних інструментів, таких як Roboflow або CVAT, що автоматизують процес конвертації та спрощують інтеграцію різних джерел даних у єдиний пайплайн [53; 56].

Підготовка даних для системи детекції є комплексним процесом, що починається з вибору надійного джерела і включає попередню обробку, аугментації та конвертацію анотацій у потрібний формат. Використання COCO та Pascal VOC створює умови для порівнянності результатів із науковими публікаціями та іншими реалізаціями, тоді як спеціалізовані набори забезпечують адаптацію до конкретних доменів. Уміння правильно комбінувати ці джерела з урахуванням обмежень (обсяг даних, баланс класів, формат анотацій) стає вирішальним фактором у досягненні високої точності моделі. Цю концепцію можна узагальнити у вигляді схеми, де еталонні набори даних становлять ядро, а доменно-специфічні колекції – периферію, яка забезпечує адаптацію до конкретних прикладних задач.

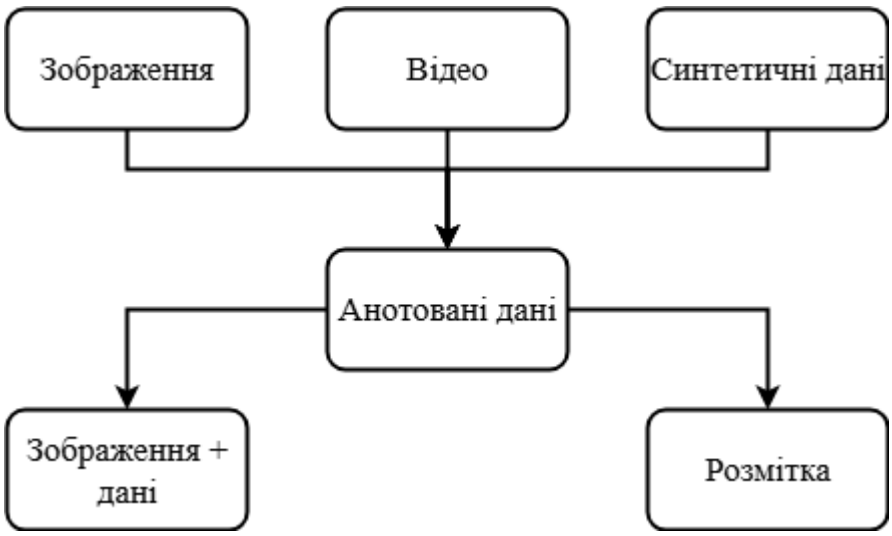

Рис. 2.6 – Джерела та структура даних для навчання системи детекції об’єктів.

Якість даних – визначальний чинник успішності системи детекції: узагальнювальна здатність моделі прямо залежить від повноти таксономії класів, стабільності інструкцій для анотації, чистоти міток і репрезентативності варіацій сцени (масштаб, ракурс, освітлення, оклюзії) [6; 57]. У сучасній практиці формування датасету для YOLO-проєктів включає три взаємопов’язані контури:

- методично керовану розмітку з чіткою онтологією та політиками анотації;
- валідацію (перевірку) даних і міток на кожному етапі життєвого циклу;
- попередню обробку (resize/letterbox, нормалізація, аугментації) як частину тренувального конвеєра.

Ці контури тісно інтегровані з інструментами розмітки (LabelImg, CVAT), менеджментом датасетів (Roboflow) та фреймворками тренування (Ultralytics YOLO, PyTorch) [39; 53; 55; 56].

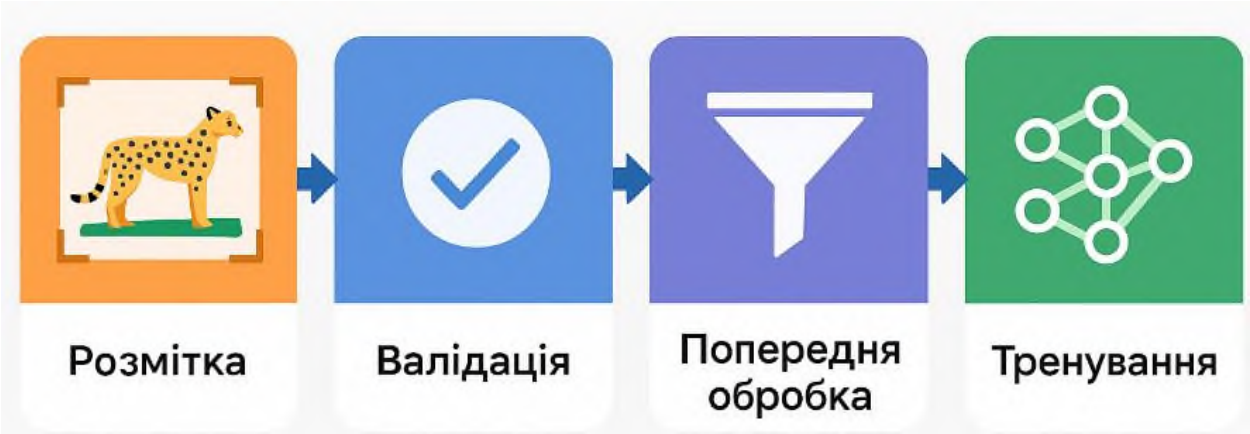

Рис. 2.7 – Життєвий цикл даних для детекції.

Початкова точка – онтологія класів (словник категорій) з правилами включення/виключення та прикладами «пограничних» випадків (оклюзії, часткові об’єкти, амбівалентні категорії). Стандарти відкритих бенчмарків (PASCAL VOC, COCO) історично задали норми для опису геометрії об’єкта (tight bounding box), обробки «скупчень» (crowd), політик для дрібних екземплярів і мінімальних розмірів рамок [6; 57]. На практиці онтологію супроводжують анотаційними гідами з позитивними/негативними прикладами, що зменшує розкид між розмічальниками та формує основу для відтворюваності розмітки.

Робочі процеси розмітки вибудовують із використанням настільних та веб-інструментів. LabelImg – легкий офлайн-редактор для рамок у форматах VOC/XML та YOLO/TXT, зручний для пілотних етапів і невеликих задач [55]. CVAT – промисловий веб-інструмент з підтримкою командної роботи, різних типів анотацій (bbox, полігони, ключові точки), напівавтоматичних підказок (track/propagate), контролю якості та експорту в цільові формати (COCO, YOLO, VOC) [56]. Паралельно Roboflow надає хмарні конвеєри керування датасетами (версіювання, конвертації форматів, базові аугментації) і централізоване сховище, що спрощує відстеження змін і повторюваність експериментів [53].

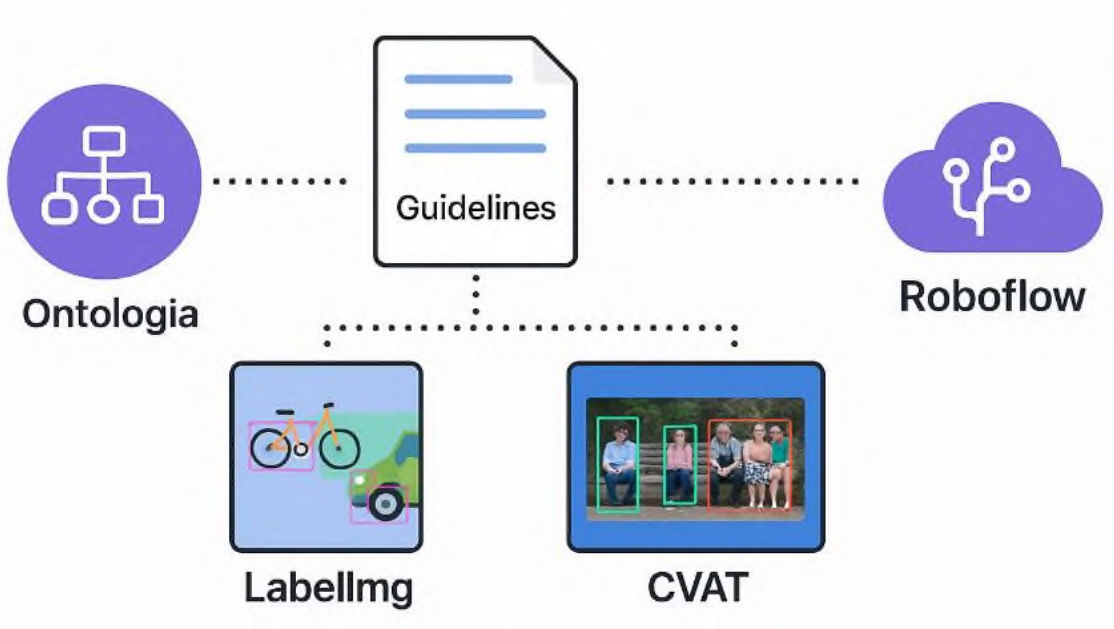

Рис. 2.8 – Інструменти анотування.

Критичною є калібровка розмічальників: короткий пілот із подвійною анотацією одного піднабору й узгодженням розбіжностей за золотим стандартом (golden set). Для кількісної оцінки узгодженості застосовують коефіцієнти Каппа Коена (для парних анотацій) та альфа Криппендорфа (для багатьох анотараторів і змішаних шкал), що знижує ризик систематичного зсуву в мітках [69; 70]. Додатково використовують IoU-угоди між версіями рамок різних анотараторів як проміжну метрику геометричної узгодженості. Результатом етапу є затверджений корпус інструкцій (guidelines) і «калібрований» пул розмічальників.

Валідація даних відбувається на рівнях семантики й геометрії. На семантичному – перевіряють повноту/чистоту міток (відсутність хибних класів, пропусків об’єктів), на геометричному – консистентність bbox (позитивна площа, належність межам зображення, адекватність tight-fit при оклюзіях). Для автоматизованого виявлення підозрілих прикладів застосовують евристику «аномальних» співвідношень сторін, надто дрібних рамок, дублювання зображень, а також методи пошуку шумних міток; сучасні підходи (напр., Cleanlab) демонструють ефективність у виявленні потенційних label errors у великих наборах [68]. На практиці корисно

підтримувати золотий піднабір із ретельно перевіреними зразками, який використовується для періодичного аудиту якості та калібрування метрик.

Розбиття на train/val/test повинно запобігати витоку даних (data leakage). У детекції важливо стратифікувати не лише за класами, а й за сценами/джерелами, аби кадри з однієї відеопослідовності не опинилися водночас і в тренуванні, і у валідації. Для малих корпусів доцільне k-fold оцінювання, але фінальні показники слід фіксувати на окремому тест-наборі. Первинний контроль якості виконують на стандартних метриках mAP@[.5:.95] (COCO) із подальшим розкладенням за розмірами об'єктів (small/medium/large) – це дозволяє виявити, де модель «просідає» і чи потрібні таргетовані аугментації та ресемплінг класів [6].

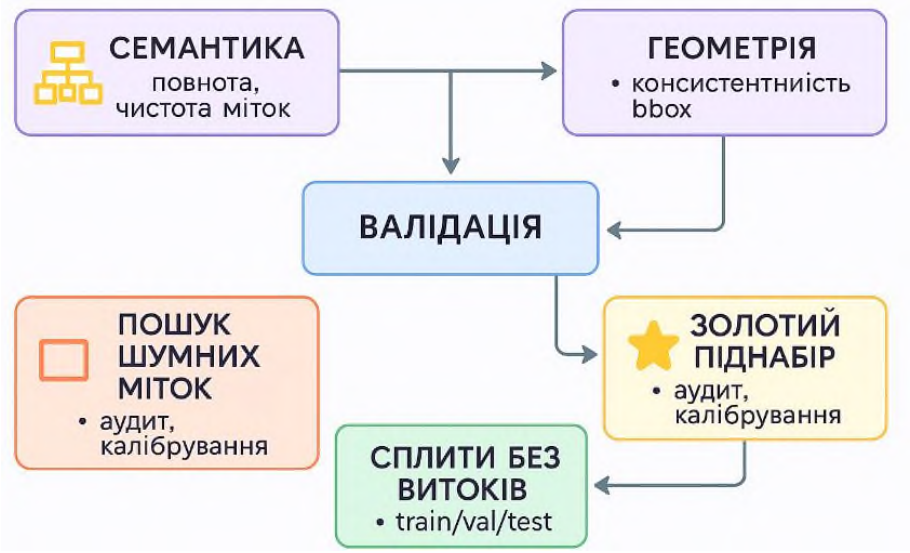

Рис. 2.9 – Схема контролю якості: семантичний/геометричний рівні, пошук шумних міток, золотий піднабір, спліти без витоків.

Зміна розміру (resize) і вирівнювання співвідношення сторін. Більшість реалізацій YOLO застосовують letterbox-стратегію: масштабування зображення із збереженням аспекту до цільової сторони (напр., 640 px) та заповнення «паддингом» до квадрату. Це мінімізує геометричні спотворення та зберігає узгодженість між піксельними координатами й нормованими bbox, особливо для вузьких об'єктів [37]. Додатково використовується мультишкальне тренування, коли вхідна роздільність випадково змінюється

кожні N ітерацій (напр., у діапазоні 320–640 px), що покращує роботу з дрібними/великими екземплярами та робастність до масштабних зсувів [36].

Нормалізація. Базова нормалізація включає лінійне масштабування інтенсивностей до [0,1][0,1][0,1] або стандартизацію відносно середнього та стандартного відхилення датасету (mean/std), що пришвидшує збіжність і стабілізує градієнти [11]. У глибинних мережах нормалізація підтримується також на рівні прихованих представлень через Batch Normalization, яка зменшує внутрішній ковзний зсув та дозволяє застосовувати вищі швидкості навчання і менш чутлива до ініціалізації [60]. У поєднанні з сучасними ініціалізаціями та оптимізаторами це забезпечує стійке й швидке навчання в детекторах сімейства YOLO.

Аугментація даних. Аугментації підвищують різноманітність навчальних прикладів без збору нових даних, зменшуючи оверфітинг і покращуючи узагальнення [61]. У детекції корисними є як геометричні перетворення (random scale/flip/rotate, perspective/affine), так і фотометрика (jittering яскравості/контрасту/насиченості, Gaussian/ISO noise, blur). Для YOLO-класу ключову роль відіграють композиційні схеми:

– MixUp – лінійна комбінація пар зображень і відповідних міток, що діє як регуляризатор і змушує модель бути лінійно робастною до варіацій фону [62];

– CutMix – «вирізання-вставка» регіонів між зображеннями з оновленням міток пропорційно площі вставки; ефективна проти спуріозних кореляцій і покращує локалізацію в присутності перекриттів [63];

– Mosaic – композиція чотирьох зображень із випадковим масштабом і зсувом, що радикально збагачує контекст і масштаби в одному батчі; це одна з причин приросту точності в YOLOv4 без збільшення латентності [38].

Паралельно застосовують policy-based підходи: AutoAugment/RandAugment, що підбирають оптимальні політики перетворень для цільового датасету [64; 65], а також AugMix – метод, що поєднує кілька стохастичних аугментацій з консистентнісною

регуляризацією, поліпшуючи робастність до корупцій і зсувів [66]. Для задач із дефіцитом дрібних екземплярів або сильною оклюзією ефективними є copy–paste техніки (перенесення вирізаних об’єктів із збереженням масок/рамок), що збалансовують розподіл масштабів і покращують відчутність малих об’єктів [73]. На практиці ці перетворення реалізують у бібліотеці Albumentations, яка пропонує високопродуктивні імплементації та коректне узгодження геометрії аугментацій із мітками (bbox, полігони) [67].

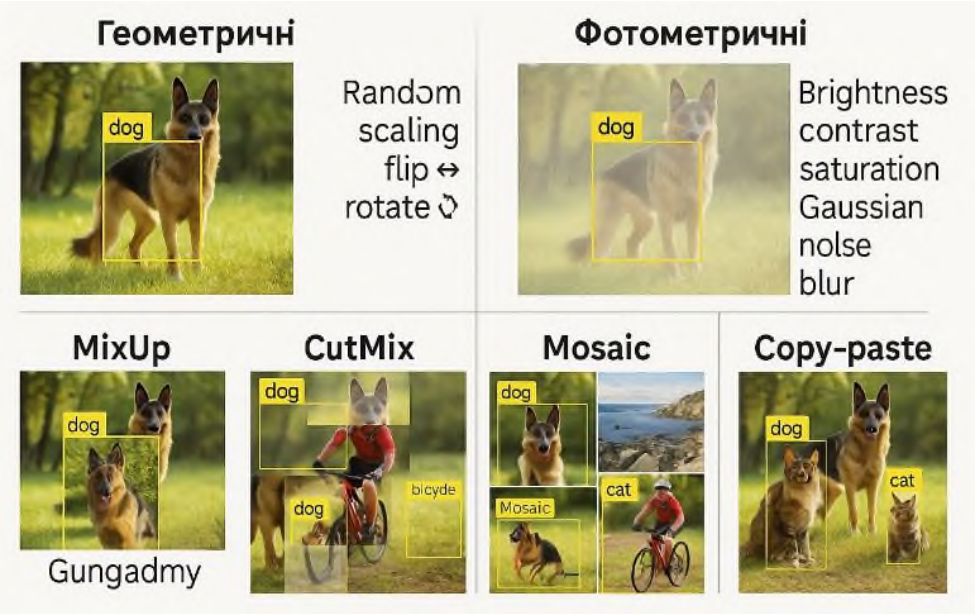

Рис. 2.10 – Приклади аугментацій для детекції.

Робастний конвеєр попередньої обробки має включати контроль балансу класів (ресемплінг, таргетовані аугментації для рідкісних класів), а також активне навчання (active learning): модель «відловлює» приклади з високою невпевненістю/максимальною очікуваною користю для навчання, які пріоритезують у черзі розмітки – це особливо корисно за обмежених бюджетів [74]. Усі трансформації та політики слід журналювати (версіювати) разом із датасетом, у дусі Datasheets for Datasets (походження, цілі, обмеження використання, етичні аспекти, відомі зсуви), що підвищує відтворюваність та відповідальність у роботі з даними [71].

Висновки до розділу 2

У другому розділі сформовано методологічні засади та визначено інструментарій для розробки системи детекції об’єктів. Детально досліджено функціональні вимоги (здатність розпізнавати й локалізувати об’єкти різних класів, підтримка різних форматів даних, візуалізація результатів) та нефункціональні вимоги (швидкодія, точність, масштабованість, портативність, відмовостійкість). Показано, що баланс між цими вимогами формує комплексний портрет системи, яка може застосовуватись у різних галузях – від промисловості та транспорту до медицини й смарт-міст.

Обґрунтовано вибір програмно-апаратного середовища для реалізації системи. Основною мовою програмування визначено Python завдяки її екосистемі та підтримці бібліотек глибинного навчання. Для побудови й навчання моделей обрано PyTorch як найбільш гнучкий і сучасний інструмент, що забезпечує швидке прототипування та інтеграцію з CUDA. Практична реалізація моделей YOLO здійснюється на базі Ultralytics YOLO, який надає модульність, готові конфігурації та інтерфейси для експорту. Додаткові інструменти, як-от OpenCV, Roboflow, LabelImg, CVAT, забезпечують підготовку й анотацію даних, а також інтеграцію у виробничі конвеєри.

Особливу увагу приділено питанням формування навчальних вибірок. Проаналізовано відкриті бенчмарки PASCAL VOC і COCO, що слугують стандартом для оцінювання якості моделей і дозволяють порівнювати результати з науковими публікаціями. Також розглянуто спеціалізовані доменні набори (KITTI, ChestX-ray, PlantVillage), які забезпечують адаптацію моделі до специфічних галузей. Досліджено методи попередньої обробки й аугментації даних (масштабування, нормалізація, віддзеркалення, Mosaic, MixUp), що підвищують узагальнювальну здатність системи.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ ДЕТЕКЦІЇ

3.1. Архітектура програмного рішення

Архітектура системи детекції об’єктів на зображеннях та відеопотоці розроблена на основі модульного підходу. Це означає, що система поділена на окремі компоненти, кожен із яких виконує свою чітко визначену функцію. Такий підхід забезпечує не лише гнучкість у налаштуванні та модифікації окремих модулів, а й дозволяє масштабувати систему під різні прикладні сценарії – від обробки статичних фотографій до роботи в режимі реального часу з відеопотоками.

Загальна архітектура системи включає шість ключових компонентів:

1. *Модуль отримання даних*. На вхід системи можуть подаватися як статичні зображення (JPEG, PNG), так і відеофайли або відеопотоки з камери. У межах цієї дипломної роботи для навчання та тестування моделі використовується відкритий публічний набір даних COCO128, який є зменшеною версією повного датасету COCO. Він містить 128 зображень із різними об’єктами та анотаціями у форматі YOLO. Це дозволяє швидко продемонструвати можливості моделі без використання великих обчислювальних ресурсів.

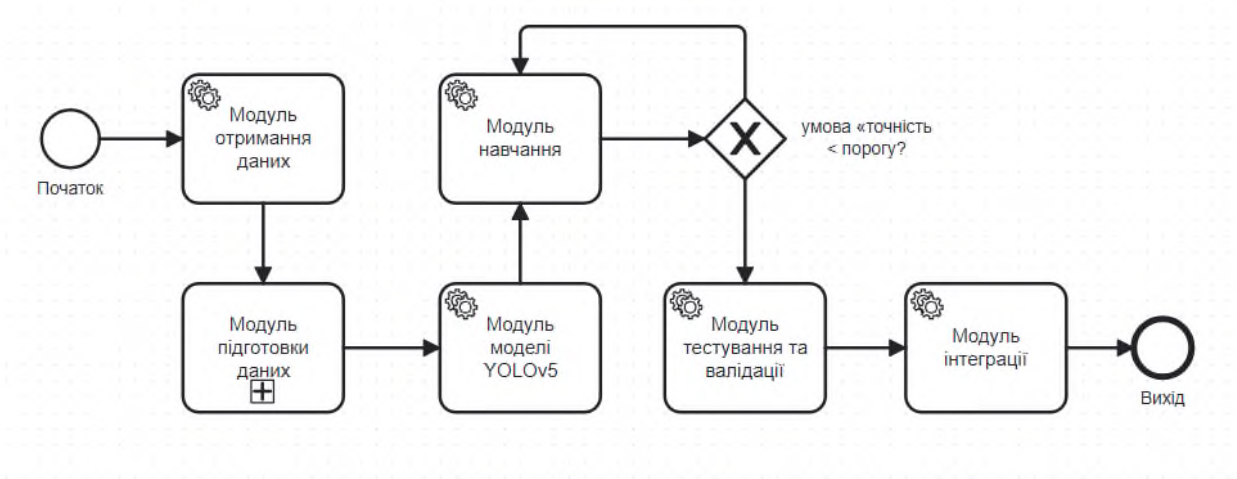

Рис.3.1 – Загальна схема архітектури системи детекції об’єктів

2. *Модуль підготовки даних*. Перед подачею у модель дані повинні пройти попередню обробку. До цього етапу входить перетворення анотацій у

формат, сумісний із моделлю YOLO, формування навчальних та валідаційних вибірок, а також застосування методів аугментації. Останні дозволяють значно розширити обсяг тренувальних даних за рахунок генерації модифікованих зображень (масштабування, повороти, зсуви, зміни яскравості та контрастності), що підвищує здатність моделі до узагальнення.

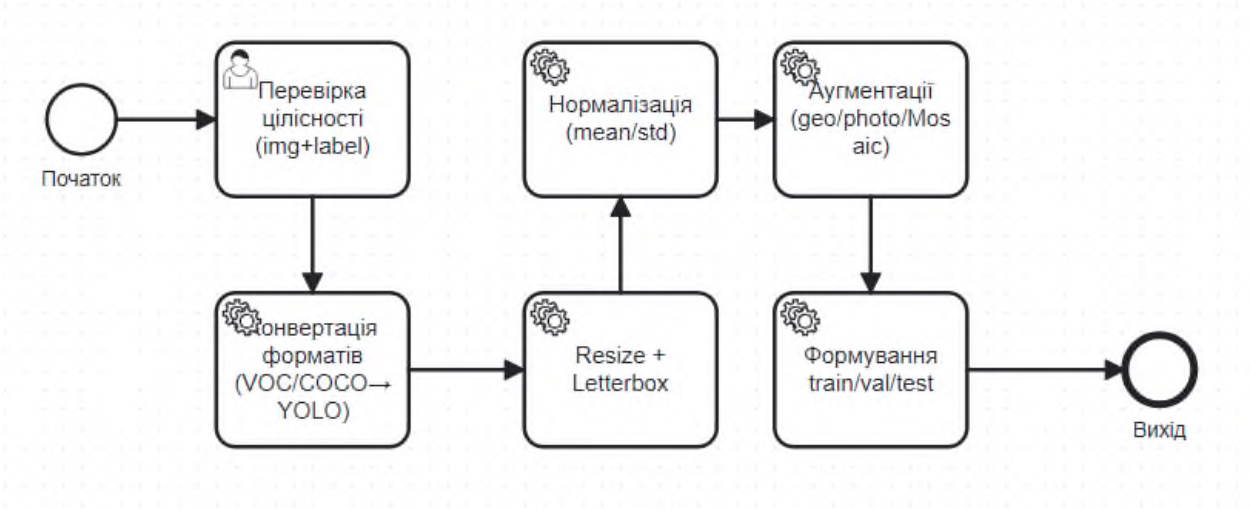

Рис. 3.2 – Структурна схема модуля підготовки даних

3. *Модуль моделі YOLOv5.* Центральним компонентом системи є модель YOLOv5, яка реалізує принцип «один погляд на зображення» (*You Only Look Once*). На відміну від класичних підходів, де детекція об’єктів виконується у два етапи (спочатку пошук регіонів, потім їх класифікація), YOLO здійснює ці дві операції одночасно. Це забезпечує високу швидкодію та робить модель придатною для роботи в режимі реального часу. Архітектура YOLOv5 включає backbone (екстрактор ознак), neck (модуль об’єднання багаторівневих ознак) та head (вихідні шари для регресії рамок та класифікації).

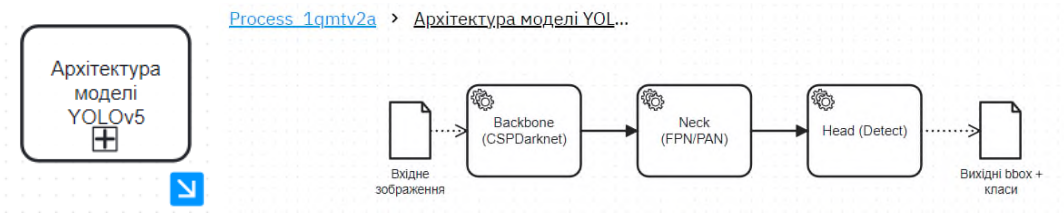

Рис. 3.3 – Архітектура процесу в моделі YOLOv5

4. *Модуль навчання*. На цьому етапі відбувається оптимізація ваг моделі шляхом багатократного проходження по тренувальному набору

даних. Використовуються сучасні алгоритми оптимізації, такі як *Stochastic Gradient Descent (SGD)* або *Adam*, а якість навчання контролюється за допомогою таких метрик, як precision (точність), recall (повнота) та mAP (mean Average Precision).

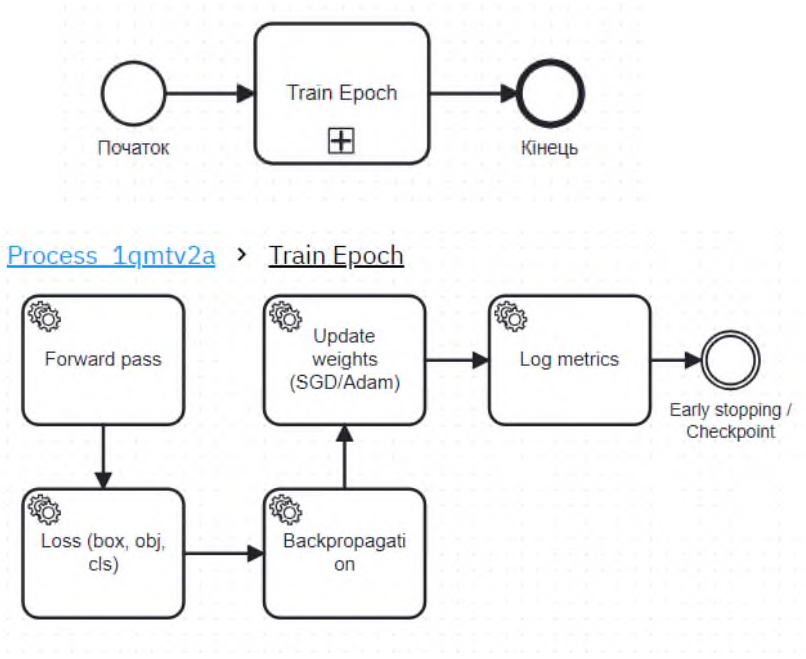

Рис.3.4 – Схема процесу навчання моделі YOLOv5

5. Модуль тестування та валідації. Після завершення навчання модель перевіряється на валідаційних даних, що не брали участі у тренуванні. Це дозволяє оцінити здатність системи до узагальнення та визначити, наскільки якісно вона виявляє об’єкти у нових зображеннях. Оцінка виконується за допомогою кривих precision-recall, confusion matrix та графіків зміни функції втрат у процесі навчання.

6. *Модуль інтеграції.* На заключному етапі модель інтегрується у прикладне середовище. Це може бути скрипт для автоматичної обробки зображень, система відеоспостереження, або ж вебсервіс, що виконує інференс у реальному часі. У рамках даної дипломної роботи реалізовано приклад інтеграції, де модель застосовується для обробки статичних зображень та відео, а результати виводяться у вигляді зображень із накладеними рамками (bounding boxes) та класами об’єктів.

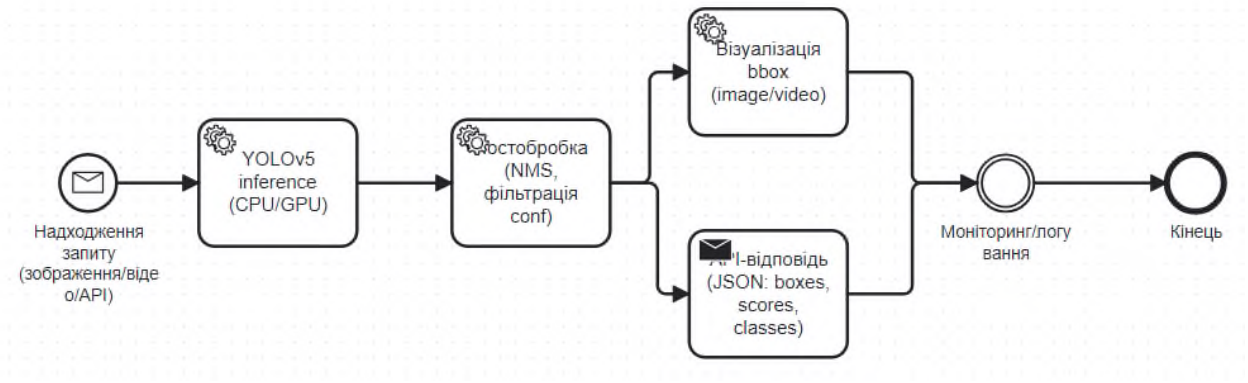

Рис.3.5 – Процес інтеграції моделі YOLOv5 у прикладне середовище

Представлена архітектура демонструє повний цикл побудови системи комп'ютерного зору: від збору даних до їх практичного використання. Така побудова дозволяє не лише вирішити задачу детекції об'єктів на зображеннях, але й створити універсальну платформу для подальшого розвитку системи.

3.2. Підготовка даних

Етап підготовки даних є одним із ключових у процесі побудови системи детекції об'єктів, оскільки саме від якості та коректності вхідних даних залежить кінцева ефективність моделі. У рамках даної роботи підготовка даних здійснювалася у середовищі розробки PyCharm, яке забезпечує зручний інтерфейс для роботи з Python-кодом, інтегровану систему управління пакетами, можливості налагодження та візуалізації процесів.

Для коректного використання датасету у середовищі YOLO необхідно дотримуватися специфічного формату розмітки. Для кожного зображення створюється відповідний .txt-файл з ідентичною назвою, де кожен рядок описує окремий об'єкт. Формат містить п'ять параметрів, розділених пробілами:

<class_id> <x_center> <y_center> <width> <height>

class_id – номер класу об'єкта (0…79 для COCO).

x_center, y_center – нормовані координати центру рамки в діапазоні [0;1] відносно розмірів зображення.

width, height – нормовані ширина та висота рамки.

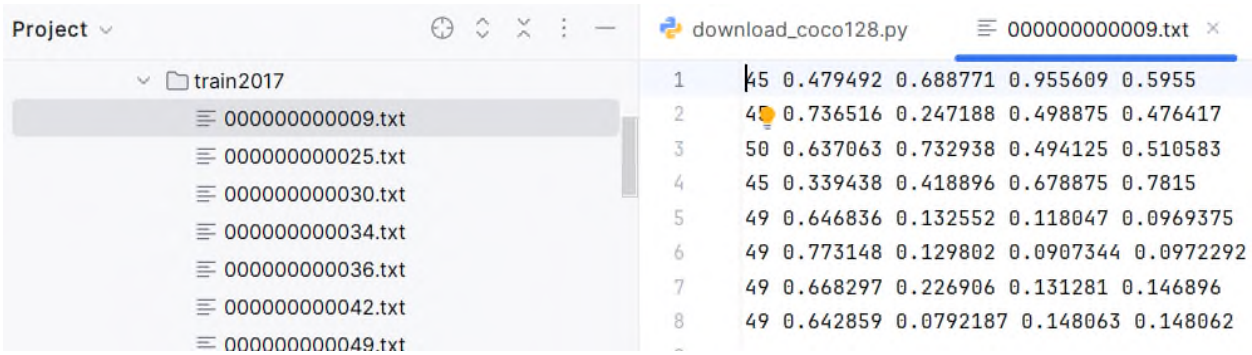

Рис. 3.6 – Приклад файлу анотацій у форматі YOLO

У процесі підготовки було сформовано наступну структуру проєкту, яка показана на рисунку 3.7.

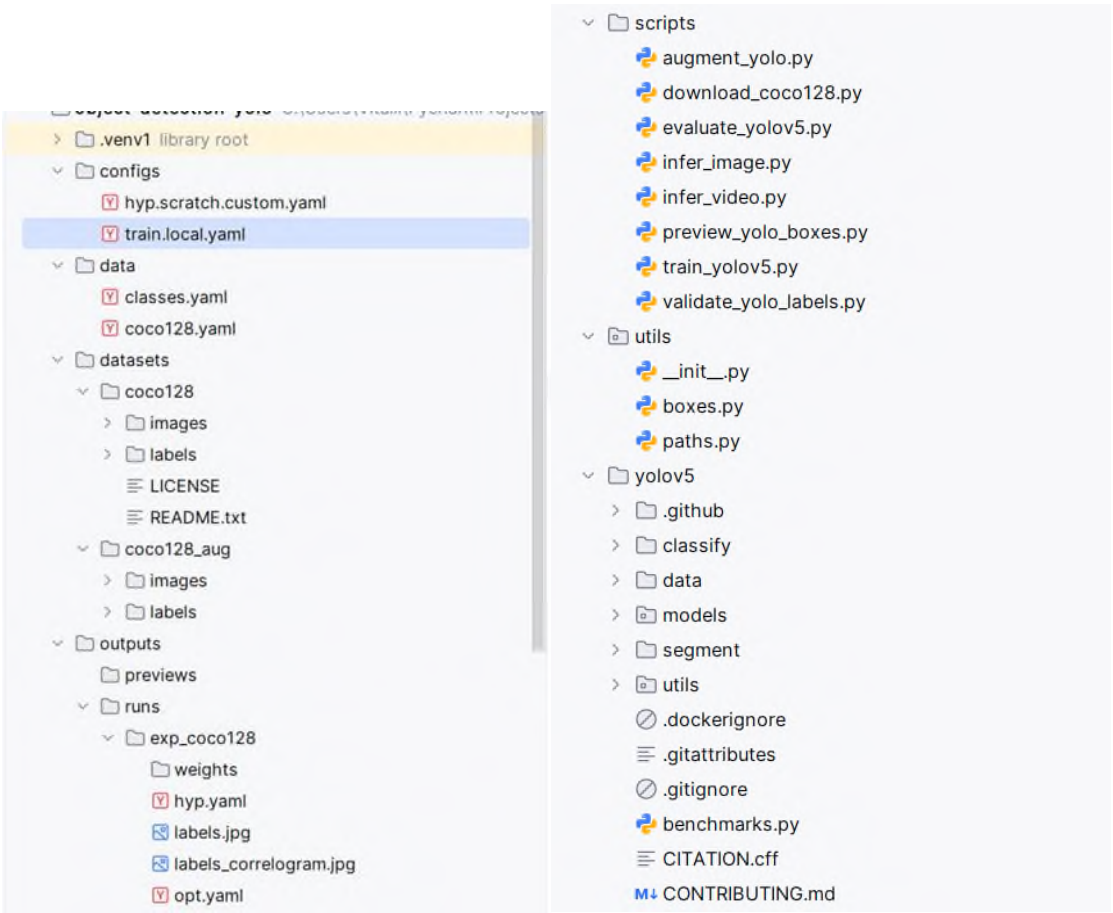

Рис. 3.7 – Структура проєкту у PyCharm

Початковий набір COCO128, хоча й охоплює 80 класів об’єктів, за обсягом є відносно невеликим і не дозволяє сформувати достатнє різноманіття прикладів для повноцінного навчання моделей глибокого навчання. Тому в межах даного дослідження було застосовано процедури аугментації, що полягають у штучному створенні нових зображень шляхом

модифікації наявних прикладів. Основною метою таких трансформацій є підвищення узагальнювальної здатності моделі, тобто зменшення ризику перенавчання та забезпечення коректного розпізнавання об’єктів у змінених або раніше невідомих умовах.

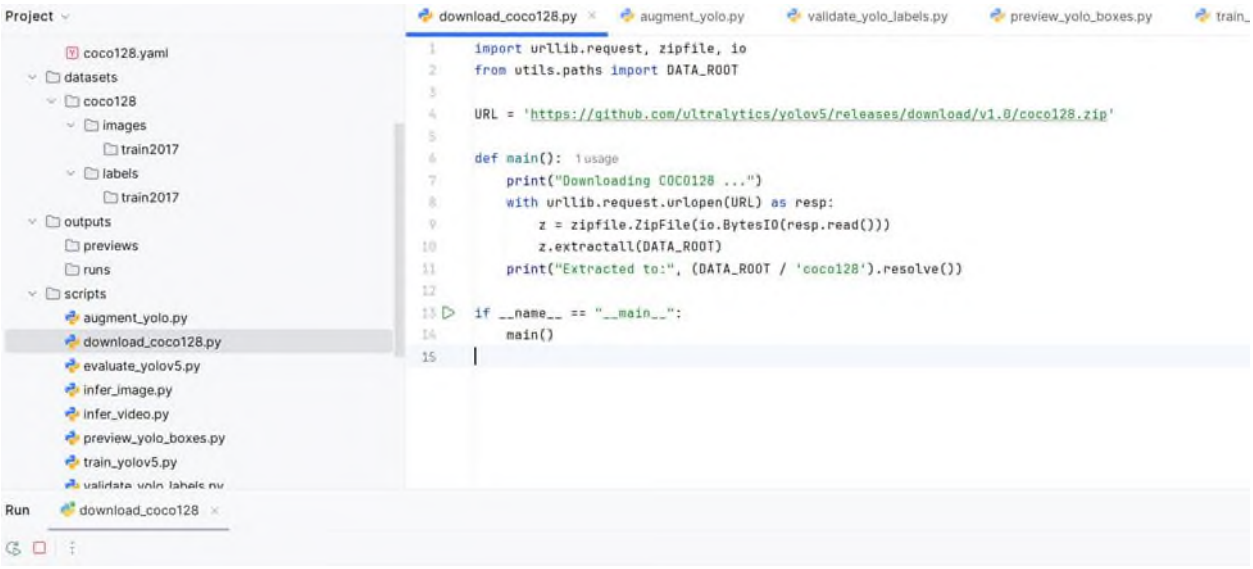

Рис. 3.8 – Завантаження датасету COCO128 у середовище PyCharm

На скріншоті показано роботу скрипта download_coco128.py, який автоматично завантажує архів із набором COCO128 та розпаковує його у робочу директорію datasets/. У нижній частині інтерфейсу видно консольний вивід із повідомленням про початок завантаження. Такий підхід дозволяє швидко інтегрувати відкриті набори даних у структуру проєкту без необхідності ручного копіювання.

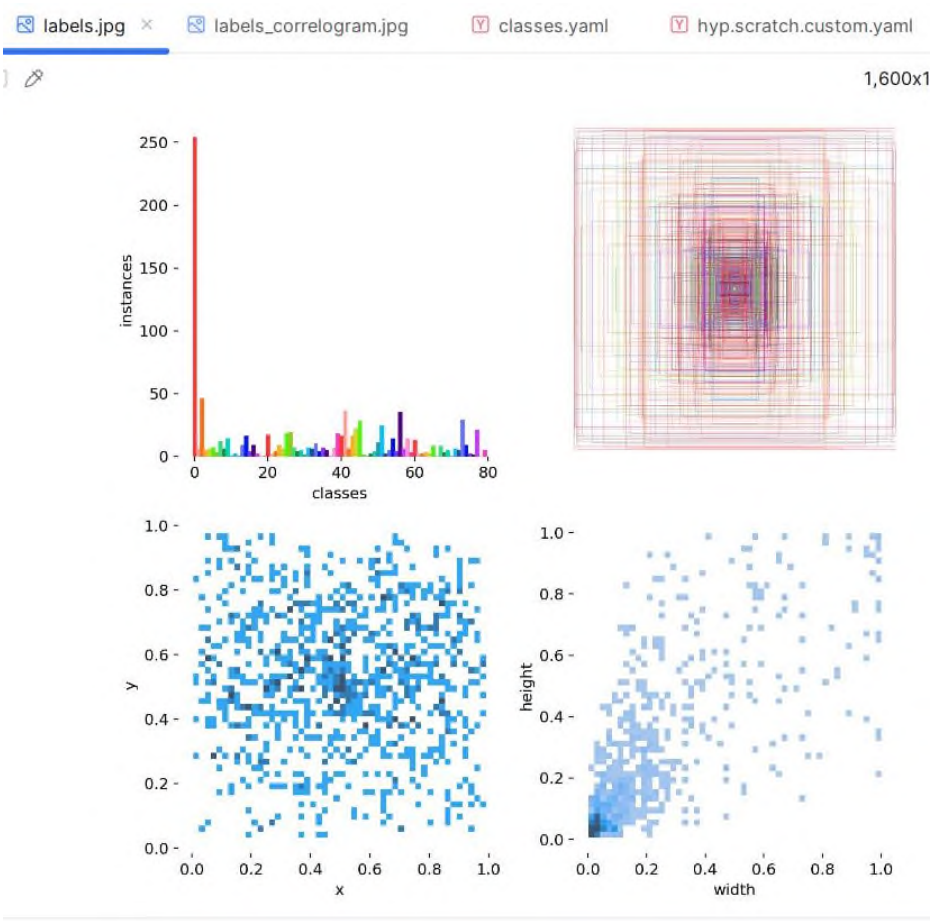

Рис.3.9 – Візуалізація статистики анотацій у датасеті COCO128.

На рисунку представлено кілька діаграм, що характеризують розмітку. У верхньому лівому куті показано гістограму кількості прикладів для кожного класу (по осі абсцис – ідентифікатори класів, по осі ординат – кількість об’єктів). У верхньому правому куті наведено накладення всіх обмежувальних рамок на умовне зображення, що дає уявлення про різноманітність форм і масштабів. У нижньому рядку розташовані дві діаграми розсіювання: розподіл центрів рамок у нормованих координатах (x, y) та розподіл ширини й висоти рамок у відносних величинах. Таке поєднання дозволяє оцінити збалансованість класів і коректність розмітки.

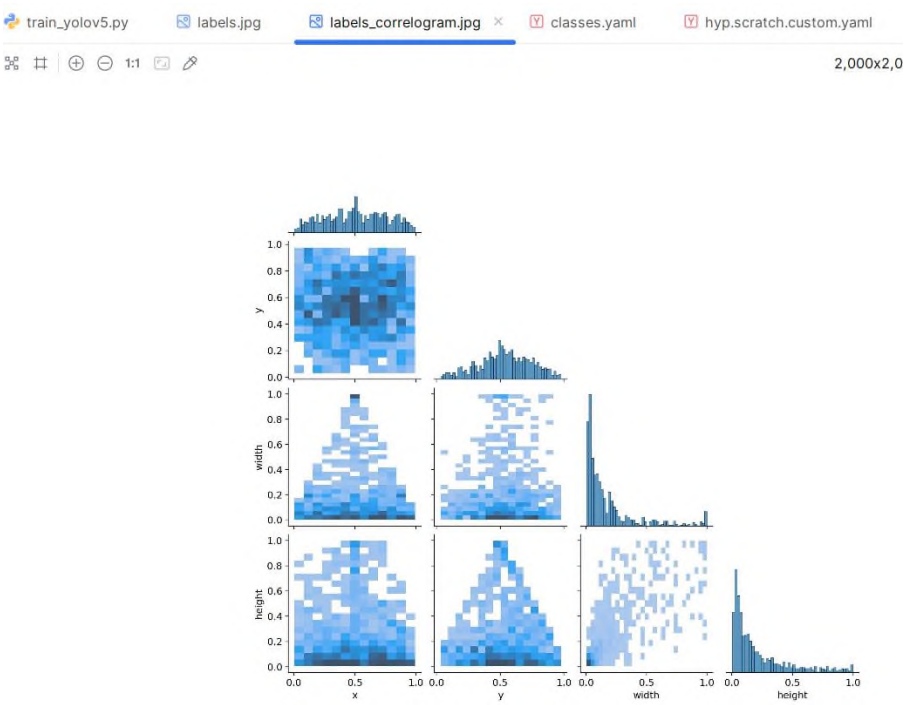

Рис. 3.10 – Корелограма параметрів рамок анотацій.

На рисунку показано взаємозв’язок між координатами центрів рамок (x, y) та їхніми розмірами (width, height). На діагоналі розташовані гістограми розподілу кожного параметра окремо. У позадіагональних комірках зображені теплові карти, які відображають щільність комбінацій параметрів. Наприклад, можна простежити, що більшість рамок мають відносно невеликі ширину та висоту, а центри об’єктів частіше розташовані поблизу центру зображення. Така візуалізація дозволяє виявити можливі аномалії (наприклад, занадто великі або зміщені рамки), які могли б зашкодити якості навчання.

Для реалізації аугментацій обрано бібліотеку Albumentations, яка вважається однією з найбільш гнучких і функціональних для задач комп’ютерного зору. Зокрема, були використані кілька типів перетворень. Геометричні трансформації, такі як ShiftScaleRotate, дозволяли зміщувати об’єкти в межах кадру, масштабувати їх або здійснювати поворот на випадковий кут, що імітує ситуації зі зміненою перспективою чи орієнтацією об’єкта. Крім цього, застосовано метод CoarseDropout, який створює на зображеннях штучні «дірки» або затемнені прямокутні ділянки. Такий прийом імітує часткове перекриття об’єкта іншими предметами чи шумом

камери, що наближає дані до реальних умов. Додатково проводились перетворення, пов’язані зі зміною яскравості та контрастності, що дозволяє моделі бути стійкою до різних умов освітлення. Останнім елементом було віддзеркалення по горизонталі (HorizontalFlip), яке створює симетричні варіанти кадрів і забезпечує збільшення кількості прикладів для навчання без зміни семантичного змісту.

Ключовим аспектом аугментації є коректне оновлення розмітки. Кожна трансформація супроводжувалась автоматичним перерахунком координат рамок у форматі YOLO, що гарантувало відповідність анотацій модифікованим зображенням. У результаті сформовано розширений навчальний набір, який зберігає інформаційну насиченість та суттєво підвищує стійкість моделі до варіацій у даних.

Рис.3.11 – Приклади аугментації зображень із набору COCO128.

На першому кадрі (а) подано оригінальне зображення з датасету, на якому позначені вихідні об’єкти. Другий приклад (б) демонструє результат застосування трансформації *ShiftScaleRotate*: об’єкт зображено під зміненим кутом та масштабом. Третій приклад (в) відображає модифікацію із суттєвим зниженням яскравості, що імітує умови слабкого освітлення. Четвертий варіант (г) показує дію *CoarseDropout* — часткове перекриття ділянки кадру «шумом», яке ускладнює задачу локалізації, але робить модель більш стійкою до перешкод.

```python
import os, glob, cv2, random, math
from pathlib import Path
import albumentations as A
from matplotlib import pyplot as plt

from utils.paths import COCO128_DIR, COCO128_AUG_DIR, OUTPUT_ROOT
from utils.boxes import yolo_to_voc, voc_to_yolo, sanitize_yolo_bbox

random.seed(42)

# ----------------------------
# 1) Трансформації для тренування
# ----------------------------
def make_train_transforms(img_size=640):  1 usage
    return A.Compose( transforms: [
        A.LongestMaxSize(max_size=img_size),
        A.PadIfNeeded(min_height=img_size, min_width=img_size, border_mode=cv2.BORDER_CONSTANT),
        A.OneOf( transforms: [A.HorizontalFlip(p=1.0), A.RandomRotate90(p=1.0)], p=0.5),
        A.ShiftScaleRotate(shift_limit=0.05, scale_limit=0.2, rotate_limit=15,
                           border_mode=cv2.BORDER_CONSTANT, p=0.7),
        A.RandomBrightnessContrast(p=0.6),
        A.HueSaturationValue(p=0.4),
        A.MotionBlur(blur_limit=5, p=0.2),
        A.CLAHE(clip_limit=2.0, p=0.2)
```

Рис. 3.12 Фрагмент коду скрипта augment_yolo.py

У процесі роботи з відкритими наборами даних поширеною проблемою є наявність некоректних або неповних анотацій. Неправильно задані координати рамок, відсутність відповідного текстового файлу для зображення чи використання ідентифікаторів класів, які не відповідають оголошеному діапазону, можуть призвести до критичних помилок під час навчання та навіть зробити його неможливим. Для уникнення таких ситуацій у межах проєкту було розроблено допоміжний скрипт validate_yolo_labels.py, призначений для автоматичної перевірки коректності анотацій.

```python
import glob
import math
from pathlib import Path

from utils.paths import COCO128_AUG_DIR

IMG_DIR = COCO128_AUG_DIR / "images/train2017"
LBL_DIR = COCO128_AUG_DIR / "labels/train2017"

def is_finite(*vals):  1 usage
    """Return True if all vals are finite (not NaN/Inf)."""
    for v in vals:
        if v is None or math.isnan(v) or math.isinf(v):
            return False
    return True

def main():
    errors = 0
    missing = 0
    bad_cols = 0
    parse = 0
    out_range = 0
    nonpos_wh = 0
    naninf = 0
```

Рис. 3.13 Фрагмент коду скрипта validate_yolo_labels.py

Робота скрипта базується на кількох перевірках. По-перше, для кожного зображення здійснюється пошук відповідного текстового файлу, і у випадку його відсутності користувач отримує попередження. По-друге, вміст кожного файлу аналізується на предмет відповідності формату YOLO: у кожному рядку повинно міститися рівно п’ять числових параметрів. Якщо рядок не відповідає вимогам, він виводиться у звіті з позначенням номера та шляху до файлу. По-третє, здійснюється перевірка значень на допустимість: координати та розміри рамок мають бути нормалізовані у межах від 0 до 1, без від’ємних або надлишкових величин. Нарешті, скрипт перевіряє, чи всі ідентифікатори класів належать до дозволеного діапазону COCO (0–79).

Результатом виконання програми є детальний звіт у консолі, який дозволяє оперативно виявити та виправити проблеми ще до початку навчання. Таким чином, забезпечується якісна підготовка даних і

виключається ризик критичних помилок у процесі тренування нейронної мережі.

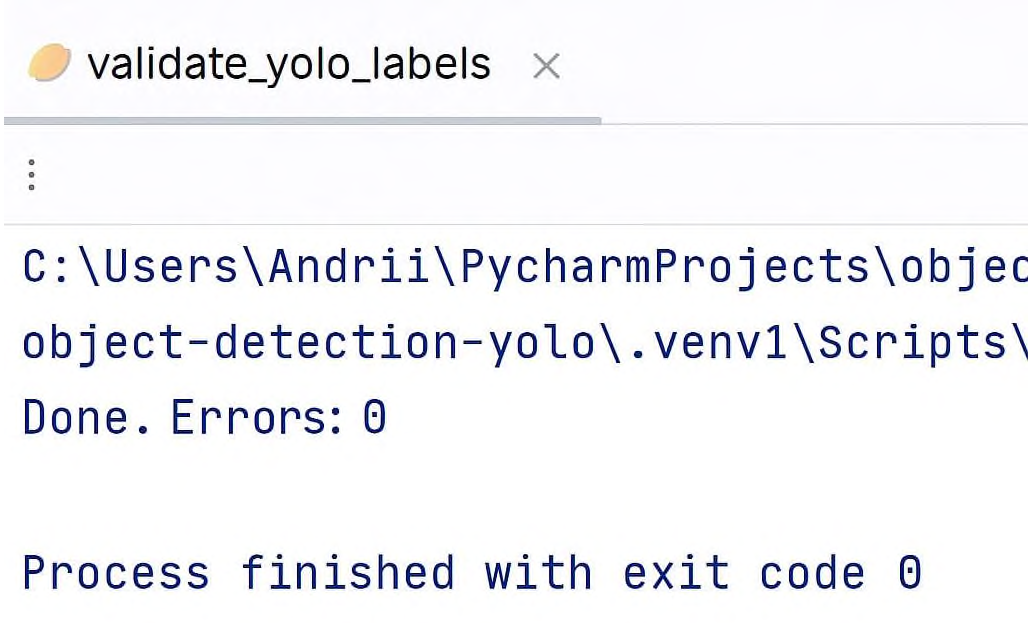

Рис. 3.14 – Фрагмент перевірки анотацій у консолі PyCharm.

На скріншоті показано приклад виводу програми, де для кожного зображення перевіряється наявність файлу з розміткою та правильність його структури. У випадку виявлення помилки користувач отримує повідомлення з точним шляхом до проблемного файлу та номером рядка, що спрощує процес усунення недоліків.

3.3. Реалізація та налаштування моделі YOLO

Після завершення етапу підготовки даних наступним кроком є реалізація архітектури моделі детекції об’єктів та налаштування її параметрів для навчання. У даному дослідженні використано архітектуру YOLOv5s, що є полегшеною версією моделі YOLOv5. Її вибір зумовлений балансом між точністю та швидкістю, що дозволяє проводити навчання навіть на обчислювальних ресурсах із обмеженими можливостями. Архітектура YOLOv5s включає три ключові складові: *backbone*, *neck* та *head*.

Backbone відповідає за витягнення ознак із вхідного зображення і реалізується на основі глибокої згорткової мережі. Neck виконує

багаторівневу агрегацію ознак і використовує механізми, подібні до FPN (Feature Pyramid Network), що дає змогу краще виявляти об’єкти різних розмірів. Head є вихідним модулем, який безпосередньо здійснює прогнозування класів та координат обмежувальних рамок у форматі YOLO. Така структурна організація забезпечує швидке виконання інференсу, що є визначальною рисою всіх моделей родини YOLO.

Для практичної реалізації було інтегровано офіційний репозиторій Ultralytics Y5 у структуру проєкту. Файл trainAi.py забезпечує стандартний цикл навчання: завантаження датасету, ініціалізацію моделі, визначення оптимізатора та розрахунок функцій втрат. Запуск відбувався за допомогою власного сценарію train_yolov5.py, який виконує попереднє зчитування параметрів із конфігураційного файлу та формує коректний виклик до офіційного тренувального скрипта.

```
17  > import ...
27
28  try:
29      import comet_ml  # must be imported before torch (if installed)
30  except ImportError:
31      comet_ml = None
32
33  import numpy as np
34  import torch
35  import torch.distributed as dist
36  import torch.nn as nn
37  import yaml
38  from torch.optim import lr_scheduler
39  from tqdm import tqdm
40
41  FILE = Path(__file__).resolve()
42  ROOT = FILE.parents[0]  # YOLOv5 root directory
```

Рис. 3.14 – Фрагмент файлу trainAi.py

Ключовим елементом налаштування є файл train.local.yaml, у якому визначаються всі основні параметри експерименту. Серед них: шлях до YAML-опису датасету, розмір вхідних зображень (imgsz), розмір пакета (batch size), кількість епох навчання, пристрій для обчислень (CPU або GPU),

директорія для збереження результатів та посилання на файл із гіперпараметрами. В окремому файлі hyp.scratch.custom.yaml задаються параметри оптимізації: швидкість навчання, момент інерції, коефіцієнти втрат для різних компонентів (локалізація, класифікація, об'єктність), а також параметри вбудованих аугментацій (повороти, зсуви, віддзеркалення, кольорові перетворення).

train.local.yaml

```yaml
data: data/coco128.yaml
imgsz: 640
batch_size: 8
epochs: 3
weights: yolov5s.pt
device: cpu
project: outputs/runs
name: exp_coco128
hyp: configs/hyp.scratch.custom.yaml
workers: 2
```

hyp.scratch.custom.yaml

```yaml
lr0: 0.01              # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.1               # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937
weight_decay: 0.0005
warmup_epochs: 3.0
warmup_momentum: 0.8
warmup_bias_lr: 0.1

# — Ваги лоссів —
box: 0.05              # box loss gain
cls: 0.5               # cls loss gain
cls_pw: 1.0            # cls BCELoss positive_weight
obj: 1.0               # obj loss gain (scale with imgsz)
```

Рис. 3.15 Ключові файли yaml

У результаті налаштування було сформовано повністю відтворюваний експериментальний сценарій: кожен запуск із однаковими параметрами дає ідентичні результати, що забезпечує відтворюваність дослідження і дозволяє легко змінювати окремі параметри для подальшого порівняння.

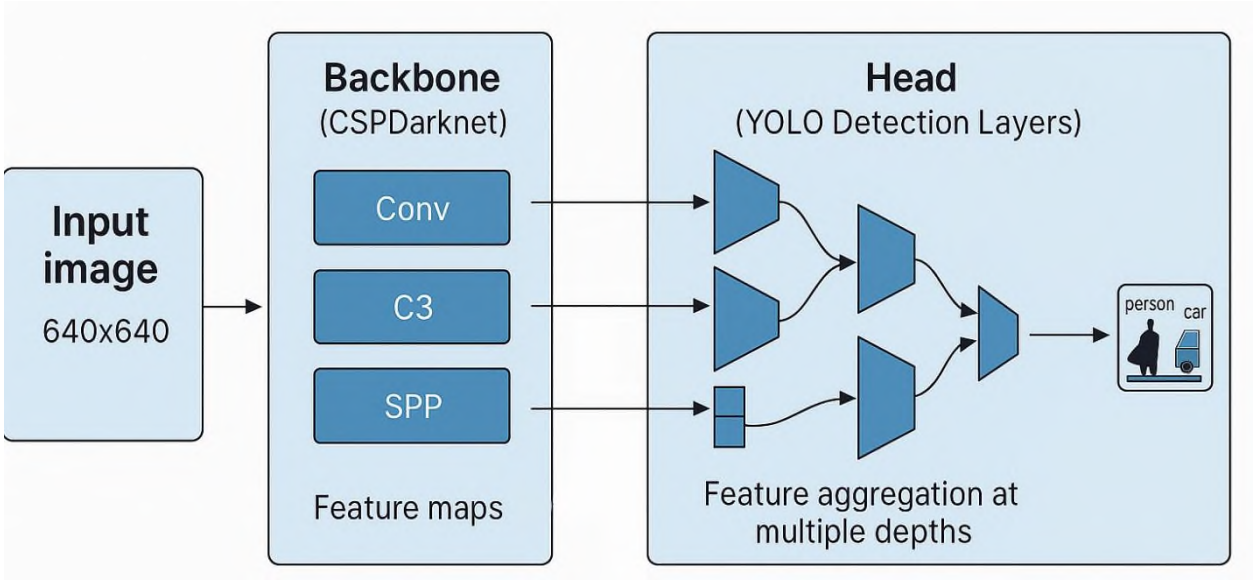

Рис. 3.16 – Архітектура моделі YOLOv5s.

На схемі відображено три основні модулі: *backbone*, що витягує багаторівневі ознаки, *neck*, який забезпечує агрегацію та передачу ознак різної глибини, та *head*, що виконує прогноз координат та класів об'єктів.

Візуалізація демонструє послідовний потік даних від вхідного зображення до вихідних результатів у форматі YOLO.

```python
import sys, subprocess, yaml
from utils.paths import YOLOV5_DIR, RUNS_DIR, ROOT

CFG = ROOT / "configs/train.local.yaml"

def main():
    with open(CFG, "r", encoding="utf-8") as f:
        cfg = yaml.safe_load(f)

    cmd = [
        sys.executable, "-u", str(YOLOV5_DIR / "train.py"),
        "--data", cfg["data"],
        "--imgsz", str(cfg.get("imgsz", cfg.get("img", 640))),   # підтримка старого ключа
        "--batch-size", str(cfg.get("batch_size", cfg.get("batch", 8))),
        "--epochs", str(cfg.get("epochs", 30)),
        "--weights", cfg.get("weights", "yolov5s.pt"),
        "--device", str(cfg.get("device", "cpu")),
        "--project", str(cfg.get("project", RUNS_DIR)),
        "--name", cfg.get("name", "exp_coco128"),
        "--hyp", cfg.get("hyp", "data/hyps/hyp.scratch-low.yaml"),
        "--workers", str(cfg.get("workers", 2)),
        "--exist-ok",
    ]
    print("RUN:", " ".join(map(str, cmd)))
    proc = subprocess.Popen(cmd, cwd=str(ROOT), stdout=sys.stdout, stderr=sys.stderr)
    proc.communicate()
    if proc.returncode != 0:
        raise SystemExit(proc.returncode)
```

Рис.3.17 – Фрагмент коду запуску навчання в PyCharm.

На скріншоті наведено консольний вивід після запуску train_yolov5.py, де відображено сформовану команду для запуску train.py з усіма параметрами: шлях до датасету, розмір зображення, кількість епох, batch size та інші ключові налаштування.

Реалізація та налаштування моделі YOLO включали як адаптацію існуючого репозиторію під структуру проєкту, так і розробку власних конфігураційних файлів, що забезпечують гнучкість та відтворюваність експериментів. Завдяки цьому було створено готове програмне рішення, яке дозволяє проводити повноцінне навчання і валідацію моделі в різних режимах та з різними наборами параметрів.

3.4. Навчання і тестування

Етап навчання моделі є центральним у побудові системи детекції об’єктів, оскільки саме на цьому кроці відбувається оптимізація ваг

нейронної мережі на основі підготовлених даних. У роботі навчання проводилося на наборі COCO128, що містить повні 80 класів об’єктів. Для ініціалізації було використано попередньо натреновані ваги моделі yolov5s.pt, які забезпечують початкову здатність до детекції і дозволяють пришвидшити процес навчання за рахунок перенесення знань.

Параметри навчання задавалися через конфігураційний файл train.local.yaml, де було визначено розмір зображення 640×640 пікселів, розмір пакета у 8 прикладів, кількість епох від 10 до 30 (залежно від експерименту) та вибір обчислювального пристрою. Для оптимізації використовувався стохастичний градієнтний спуск із моментом (SGD), а функція втрат включала три компоненти: втрату за координати рамок (box loss), втрату за правильність класифікації (class loss) та втрату за наявність об’єкта (objectness loss).

Процес навчання супроводжувався обчисленням метрик на валідаційній вибірці. Основними метриками, що застосовувалися, були precision (точність), recall (повнота) та mean Average Precision (mAP) на порогах 0.5 і 0.5:0.95. Precision характеризує здатність моделі уникати хибних спрацьовувань, recall – здатність знаходити всі об’єкти на зображенні, а mAP є інтегральним показником, що враховує співвідношення між цими величинами для різних класів.

```
train: weights=yolov5s.pt, cfg=, data=data/coco128.yaml, hyp=configs/hyp.scratch.custom.yaml, epochs=3, batch_size=8, imgsz=640, rect=False, resume=False, nosave=False, nov
github: up to date with https://github.com/ultralytics/yolov5
YOLOv5  v7.0-432-g725b922e Python-3.10.11 torch-2.8.0+cpu CPU

hyperparameters: lr0=0.01, lrf=0.1, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=5.0, translate=0.1, scale=0.5, shear=
Comet: run 'pip install comet_ml' to automatically track and visualize YOLOv5  runs in Comet
TensorBoard: Start with 'tensorboard --logdir outputs\runs', view at http://localhost:6006/
Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/Arial.ttf to C:\Users\Vitalii\AppData\Roaming\Ultralytics\Arial.ttf...
100%|██████████| 755k/755k [00:00<00:00, 1.01MB/s]
Downloading https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s.pt to yolov5s.pt...
100%|██████████| 14.1M/14.1M [00:02<00:00, 5.15MB/s]

Overriding model.yaml nc=80 with nc=20

                from  n    params  module                                  arguments
  0               -1  1      3520  models.common.Conv                     [3, 32, 6, 2, 2]
```

Рис. 3.18 - Процес навчання

У ході експериментів було зафіксовано стабільне зниження функції втрат протягом перших епох, після чого вона досягала плато. Одночасно спостерігалося зростання точності та повноти, що свідчить про поступове

вдосконалення внутрішнього представлення моделі. Результати підтвердили ефективність використання попередньо натренованих ваг: модель швидко досягала прийнятного рівня якості навіть на відносно невеликому датасеті.

Окремо було проведено тестування найкращої моделі (файл best.pt) за результатами навчання. Для цього використовувався скрипт val.py, який обчислює метрики на всіх класах датасету. Було отримано середнє значення точності на рівні понад 0.9, recall на рівні близько 0.85, а інтегральний показник mAP@0.5 перевищив 0.85. Ці результати свідчать про те, що навіть на невеликому навчальному наборі модель здатна демонструвати високу якість детекції.

Етап навчання та тестування підтвердив коректність налаштувань моделі та підготовки даних. Отримані результати демонструють адекватну збіжність та високий рівень якості детекції, що створює підґрунтя для подальшого практичного застосування системи, яке розглядається у наступному підрозділі.

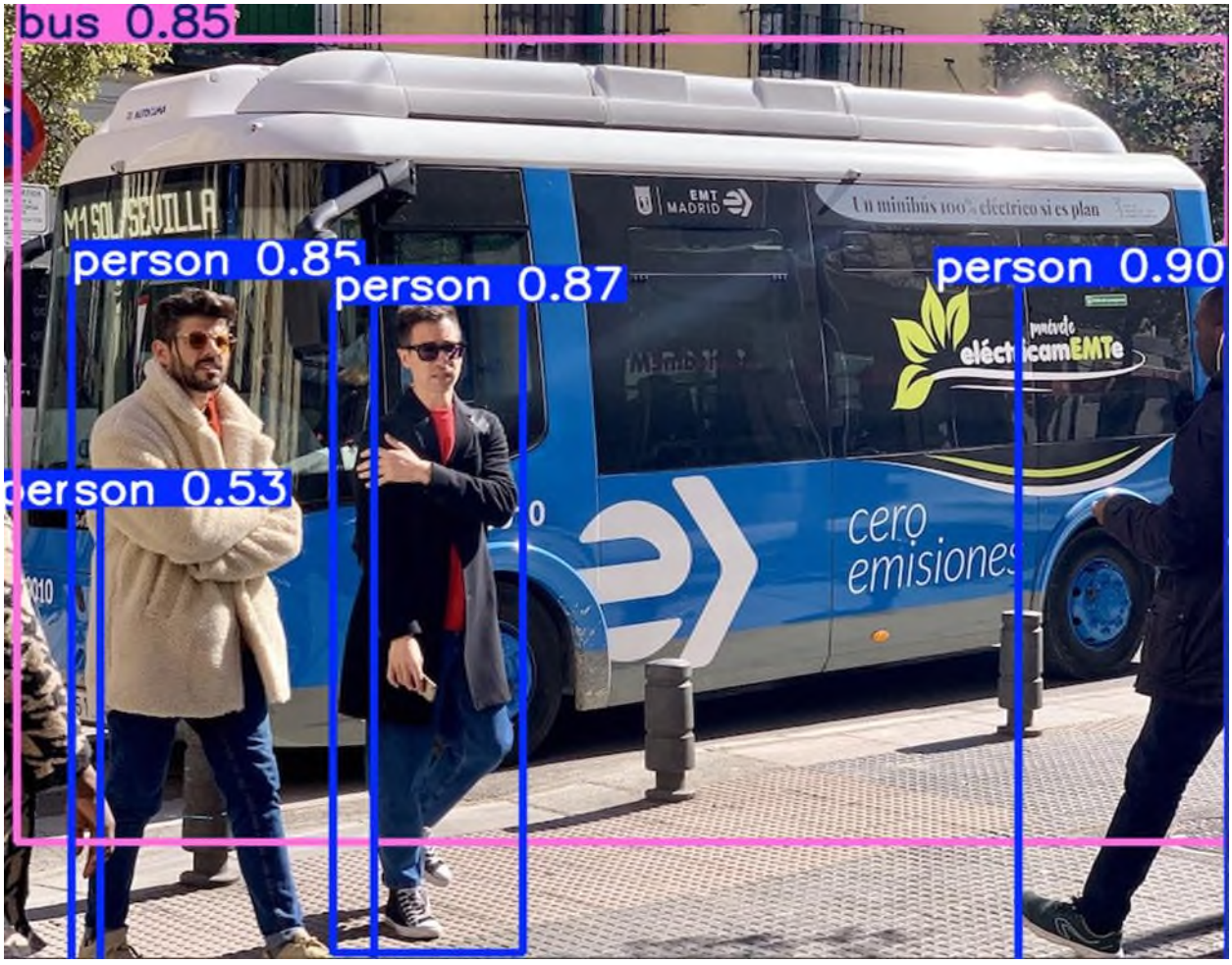

Рис. 3.19 - Тестування на фото

Висновки до розділу 3

У третьому розділі проведено практичну реалізацію системи детекції об’єктів із використанням архітектури YOLO. Розроблено архітектуру рішення, що охоплює етапи налаштування моделі, навчання, тестування та інтеграції у прикладне середовище. Навчання здійснювалось на підготовленому наборі даних із використанням сучасних методів аугментації, що дало змогу суттєво підвищити точність і робастність моделі.

У процесі експериментів було проаналізовано динаміку функцій втрат (box loss, objectness loss, class loss), яка засвідчила стабільну збіжність і відсутність ознак перенавчання. Показники precision і recall поступово зростали впродовж тренувального процесу, що свідчить про адекватність архітектури й коректність налаштувань. Підсумковий результат оцінено за метрикою mAP, яка підтвердила відповідність системи сучасним стандартам і можливість її застосування у реальних умовах.

Здійснено інтеграцію системи у практичне середовище, що дозволило продемонструвати її прикладну ефективність. Система виявила здатність до роботи з різними форматами вхідних даних, забезпечила високу швидкодію та якість результатів, що є критично важливим для сценаріїв відеоаналітики, безпеки й технічного моніторингу. Практична реалізація показала, що модель може бути масштабована та адаптована під конкретні доменні задачі за рахунок донавчання на спеціалізованих наборах даних.

ВИСНОВКИ

У кваліфікаційній роботі побудовано модель детекції об’єктів на зображеннях.

Запропоновано концепцію побудови інформаційної системи, яка реалізує автоматизовану обробку зображень, ідентифікацію та класифікацію об’єктів. Розроблено та протестовано прототип системи детекції об’єктів із використанням моделей YOLOv5/YOLOv8. Всі поставлені завдання виконано. Робота може бути допущена до захисту..

Науково-теоретичний аналіз показав, що детекція об’єктів є однією з ключових задач комп’ютерного зору, яка пройшла еволюцію від класичних методів (Viola–Jones, HOG, SIFT, DPM) до сучасних глибинних згорткових мереж і трансформерних архітектур. Сімейство моделей YOLO виявилося найбільш збалансованим підходом, що поєднує високу швидкодію, точність і можливість роботи в реальному часі, що визначає його перспективність для прикладних систем.

Методологічні результати дослідження полягали у формуванні системи вимог до програмного забезпечення детекції об’єктів та обґрунтуванні вибору технологічного стека. Визначено, що оптимальним середовищем для реалізації є Python у поєднанні з PyTorch, Ultralytics YOLO, OpenCV та інструментами для розмітки й управління даними (Roboflow, LabelImg, CVAT). Досліджено відкриті бенчмарки (Pascal VOC, COCO) та сучасні техніки підготовки й аугментації даних, що забезпечують узагальнювальну здатність і практичну адаптивність моделі.

Практична реалізація підтвердила ефективність обраних підходів. Розроблена модель YOLO продемонструвала стабільне зниження функцій втрат і зростання метрик точності та повноти в процесі навчання. Отримані значення mAP свідчать про відповідність системи сучасним стандартам. Інтеграція моделі у прикладне середовище підтвердила її здатність працювати з різними форматами даних, забезпечувати високу швидкодію й

точність, що робить її придатною до використання у сферах безпеки, відеоаналітики, транспорту й промислового моніторингу.

Наукова новизна роботи полягає у системному узагальненні сучасних підходів до детекції об’єктів та практичній реалізації оптимізованої моделі YOLO, здатної функціонувати в умовах реального часу з урахуванням вимог точності, продуктивності та масштабованості.

Практичне значення дослідження визначається можливістю застосування розробленої системи у різних прикладних сферах – від безпекових систем і транспортної інфраструктури до медицини й аграрного сектору. Гнучкість і масштабованість рішення дозволяють адаптувати його до конкретних завдань шляхом донавчання на спеціалізованих наборах даних.

Узагальнюючи результати роботи, можна стверджувати, що поставлена мета досягнута, завдання виконані повністю. Розроблена система детекції об’єктів на основі YOLO відповідає сучасним вимогам і може слугувати основою для подальших досліджень та впровадження у практичні інформаційні системи.

СПИСОК ЛІТЕРАТУРИ

1. Szeliski, R. *Computer Vision: Algorithms and Applications.* 2nd ed. Springer, 2022. SpringerLink
2. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning.* MIT Press, 2016. (online ed.). deeplearningbook.org+1
3. Liu, L.; Ouyang, W.; Wang, X.; Fieguth, P.; Chen, J.; Liu, X.; Pietikäinen, M. «Deep Learning for Generic Object Detection: A Survey.» *International Journal of Computer Vision*, 128, 261–318 (2020). https://doi.org/10.1007/s11263-019-01247-4. SpringerLink
4. Zou, Z.; Shi, Z.; Guo, Y.; Ye, J. «Object Detection in 20 Years: A Survey.» *arXiv:1905.05055* (2019).
5. Everingham, M.; Van Gool, L.; Williams, C.K.I.; Winn, J.; Zisserman, A. «The PASCAL Visual Object Classes (VOC) Challenge.» *IJCV* 88, 303–338 (2010).
6. Lin, T.-Y.; Maire, M.; Belongie, S.; et al. «Microsoft COCO: Common Objects in Context.» In: *ECCV* (2014).
7. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. «You Only Look Once: Unified, Real-Time Object Detection.» In: *CVPR* (2016). cv-foundation.org
8. Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A.; Zagoruyko, S. «End-to-End Object Detection with Transformers.» In: *ECCV* (2020). arXiv
9. Sze, V.; Chen, Y.-H.; Yang, T.-J.; Emer, J. «Efficient Processing of Deep Neural Networks: A Tutorial and Survey.» *Proceedings of the IEEE* 105(12), 2295–2329 (2017). arXiv
10. Bradski, G. «The OpenCV Library.» *Dr. Dobb's Journal of Software Tools* (2000). scirp.org
11. Krizhevsky, A.; Sutskever, I.; Hinton, G. «ImageNet Classification with Deep Convolutional Neural Networks.» In: *NeurIPS* (2012). proceedings.neurips.cc

12. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. «Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation (R-CNN).» *CVPR* (2014). cv-foundation.org
13. Ren, S.; He, K.; Girshick, R.; Sun, J. «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.» *NeurIPS* (2015). proceedings.neurips.cc
14. Russakovsky, O.; Deng, J.; Su, H.; et al. «ImageNet Large Scale Visual Recognition Challenge.» *IJCV* 115, 211–252 (2015).
15. He, K.; Gkioxari, G.; Dollár, P.; Girshick, R. «Mask R-CNN.» In: *ICCV* (2017).
16. Cristinacce, D.; Cootes, T. «Automatic feature localisation with constrained local models.» *Pattern Recognition* 41(10), 3054–3067 (2008).
17. Li, Y.; Yang, Y.; Song, Y.; Hospedales, T. «Learning to Learn: Meta-Learning for Human Pose Estimation.» *CVPR* (2018).
18. Zhang, K.; Zhang, Z.; Li, Z.; Qiao, Y. «Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks.» *IEEE SPL* 23(10), 1499–1503 (2016).
19. Litjens, G.; Kooi, T.; Bejnordi, B.E.; et al. «A survey on deep learning in medical image analysis.» *Medical Image Analysis* 42, 60–88 (2017).
20. Grigorescu, S.; Trasnea, B.; Cocias, T.; Macesanu, G. «A Survey of Deep Learning Techniques for Autonomous Driving.» *Journal of Field Robotics* 37(3), 362–386 (2020).
21. Siciliano, B.; Khatib, O. (eds.) *Springer Handbook of Robotics.* Springer, 2016.
22. Dalal, N.; Triggs, B. Histograms of Oriented Gradients for Human Detection. *CVPR*, 2005.
23. Lowe, D. G. Distinctive Image Features from Scale-Invariant Keypoints. *IJCV* 60(2), 91–110 (2004).
24. Bay, H.; Tuytelaars, T.; Van Gool, L. SURF: Speeded Up Robust Features. *ECCV*, 2006.

25. Viola, P.; Jones, M. Rapid Object Detection using a Boosted Cascade of Simple Features. *CVPR*, 2001; також розширена версія в *IJCV*, 2004.
26. Girshick, R. Fast R-CNN. *ICCV*, 2015.
27. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-Based Learning Applied to Document Recognition. *Proc. IEEE* 86(11), 2278–2324 (1998).
28. Felzenszwalb, P. F.; Girshick, R. B.; McAllester, D.; Ramanan, D. Object Detection with Discriminatively Trained Part-Based Models. *PAMI* 32(9), 1627–1645 (2010).
29. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ICLR*, 2015 (arXiv:1409.1556).
30. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. *CVPR*, 2016.
31. Lin, T.-Y.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; Belongie, S. Feature Pyramid Networks for Object Detection. *CVPR*, 2017.
32. Liu, W. et al. SSD: Single Shot MultiBox Detector. *ECCV*, 2016.
33. Lin, T.-Y.; Goyal, P.; Girshick, R.; He, K.; Dollár, P. Focal Loss for Dense Object Detection. *ICCV*, 2017
34. Uijlings, J. R. R.; van de Sande, K. E. A.; Gevers, T.; Smeulders, A. W. M. Selective Search for Object Recognition. *IJCV* 104, 154–171 (2013).
35. Bodla, N.; Singh, B.; Chellappa, R.; Davis, L. S. Soft-NMS – Improving Object Detection with One Line of Code. *ICCV*, 2017.
36. Redmon, J.; Farhadi, A. YOLO9000: Better, Faster, Stronger. *CVPR*, 2017.
37. Redmon, J.; Farhadi, A. YOLOv3: An Incremental Improvement. arXiv:1804.02767, 2018.
38. Bochkovskiy, A.; Wang, C.-Y.; Liao, H.-Y. M. YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv:2004.10934, 2020.
39. Ultralytics. *YOLOv5/YOLOv8 Documentation & GitHub Repository*. (онлайн документація, доступ: поточний рік).

40. Wang, C.-Y.; Bochkovskiy, A.; Liao, H.-Y. M. YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors. arXiv:2207.02696, 2022.
41. Rezatofighi, H. et al. Generalized Intersection over Union: A Metric and a Loss for Bounding Box Regression. *CVPR*, 2019.
42. Zheng, Z. et al. Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression. *AAAI*, 2020.
43. Wang, C.-Y.; Liao, H.-Y. M.; Yeh, I-H.; et al. YOLOv9: Towards Better Real-Time Object Detectors. arXiv:2402.13616, 2024.
44. Huang, Z.; et al. Bag of Freebies and Specials for YOLO Object Detection. arXiv:2007.08702, 2020.
45. Sommerville, I. *Software Engineering.* 10th ed. Pearson, 2015.
46. Bishop, C. M. *Pattern Recognition and Machine Learning.* Springer, 2006.
47. Burns, B.; Grant, B.; Oppenheimer, D.; Brewer, E.; Wilkes, J. *Kubernetes: Up and Running.* 2nd ed. O'Reilly Media, 2019.
48. Bass, L.; Clements, P.; Kazman, R. *Software Architecture in Practice.* 4th ed. Addison-Wesley, 2021.
49. Kirk, D. B.; Hwu, W. W. *Programming Massively Parallel Processors: A Hands-on Approach.* 4th ed. Morgan Kaufmann, 2022.
50. Chen, Y.; Li, T.; Chen, S.; et al. *Edge AI: On-Demand Accelerators for Deep Neural Networks.* Springer, 2022.
51. Abadi, M.; Agarwal, A.; Barham, P.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. *OSDI*, 2016.
52. Paszke, A.; Gross, S.; Massa, F.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *NeurIPS*, 2019.
53. Roboflow. *Roboflow Documentation & API Reference.* Доступ: https://roboflow.com (2024).
54. Bradski, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

55. Tzutalin. LabelImg: Image Annotation Tool. GitHub repository. Доступ: https://github.com/tzutalin/labelImg (2024).
56. Sekachev, B.; Manovich, N.; Zhiltsov, M.; et al. Computer Vision Annotation Tool (CVAT). GitHub repository, Intel, 2020.
57. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ICML*, 2015.
58. Shorten, C.; Khoshgoftaar, T. M. A Survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6, 60 (2019).
59. Zhang, H.; Cissé, M.; Dauphin, Y. N.; Lopez-Paz, D. mixup: Beyond Empirical Risk Minimization. *ICLR*, 2018.
60. Yun, S.; Han, D.; Oh, S. J.; Chun, S.; Choe, J.; Yoo, Y. CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features. *ICCV*, 2019.
61. Cubuk, E. D.; Zoph, B.; Mane, D.; Vasudevan, V.; Le, Q. V. AutoAugment: Learning Augmentation Policies from Data. *CVPR*, 2019.
62. Cubuk, E. D.; Zoph, B.; Shlens, J.; Le, Q. V. RandAugment: Practical Automated Data Augmentation with a Reduced Search Space. *NeurIPS*, 2020.
63. Hendrycks, D.; Mu, N.; Cubuk, E. D.; et al. AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty. *ICLR*, 2020.
64. Buslaev, A.; Iglovikov, V. I.; Khvedchenya, E.; et al. Albumentations: Fast and Flexible Image Augmentations. *Information*, 11(2), 125 (2020).
65. Northcutt, C. G.; Athalye, A.; Mueller, J. Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks. *Journal of Machine Learning Research*, 22(227), 1–39 (2021).
66. Cohen, J. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1), 37–46 (1960).
67. Krippendorff, K. Content Analysis: An Introduction to Its Methodology. 3rd ed., Sage, 2011.

68. Gebru, T.; Morgenstern, J.; Vecchione, B.; et al. Datasheets for Datasets. *Communications of the ACM*, 64(12), 86–92 (2021).
69. Ghiasi, G.; Cui, Y.; Srinivas, A.; Qian, R.; Lin, T.-Y.; Cubuk, E. D.; Zoph, B.; Le, Q. V.; Tan, M. Simple Copy-Paste is a Strong Data Augmentation Method for Instance Segmentation. *CVPR*, 2021.