МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ВИЯВЛЕННЯ ПРОСТОРОВОГО ПОЗИЦІЮВАННЯ ОБ'ЄКТІВ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ»**

Виконав: студент 2 курсу, групи 8.1264
спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)
освітньої програми Інформаційні системи та штучний інтелект
(назва освітньої програми)
Д.С. Бурковський
(ініціали та прізвище)
Керівник професор кафедри комп'ютерних наук, доцент, д.е.н. Полуектова Н.Р.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент професор кафедри програмної інженерії, к.ф.-м.н., доцент Кудін О.В..
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти магістр

Спеціальність 126 Інформаційні системи та технології
(шифр і назва)

Освітня програма Інформаційні системи та штучний інтелект

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д. т. н., професор

_____ Шило Г.М..
(підпис)

" ____ " _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Бурковському Дмитру Сергійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка нейронної мережі для виявлення позиціонування об'єктів в режимі реального часу

керівник роботи Полуектова Н.Р., професор кафедри комп'ютерних наук, д.е.н, професор
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 8 » лютого 2025 року № 326-с

2. Строк подання студентом роботи 27.11.2025

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка нейронної мережі для виявлення позиціонування об'єктів в режимі реального часу

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання 20.02.2025

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	22.03.25	
2.	Збір вихідних даних.	24.04.25	
3.	Обробка методичних та теоретичних	12.08.25	
	джерел.		
4.	Розробка першого та другого розділу.	12.19.25	
5.	Розробка третього розділу.	12.10.25	
6.	Оформлення та нормоконтроль	20.11.25	
	кваліфікаційної роботи магістра.		
		10.12.25	
7.	Захист кваліфікаційної роботи.		

Студент _____ Д. С. Бурковський
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.Р. Полуектова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка нейронної мережі для виявлення просторового позиціонування об’єктів в режимі реального часу»: 66 с., 23 рис., 2 табл., 22 джерела, 4 додатки.

ВІДЕОПОТІК У РЕАЛЬНОМУ ЧАСІ, ДЕТЕКЦІЯ ОБ’ЄКТІВ, КОМП’ЮТЕРНИЙ ЗІР, МАШИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, ПОЗИЦІЮВАННЯ ОБ’ЄКТА, YOLO

Об’єкт дослідження – проблема автоматичного просторового позиціонування об’єктів в режимі реального часу.

Предмет дослідження – інтелектуальні алгоритми відстежування об’єктів у просторі.

Мета роботи – дослідити сучасні методи виявлення об’єктів в кадрі, проаналізувати інтелектуальні алгоритми реалізації подібних додатків, розробити додаток для виявлення об’єкту та визначення його позиціювання відносно камери в режимі реального часу.

Методи дослідження – теоретичний аналіз, емпіричне дослідження, моделювання та симуляція, тестування.

У роботі досліджено алгоритми виявлення об’єкту, методи створення набору даних для навчання проблеми визначення просторового позиціювання об’єктів. Також в роботі описана розробка прототипу додатку для виявлення об’єкту із застосуванням штучного інтелекту та визначення його координат відносно камери.

SUMMARY

Master’s Qualifying Theses "Development of a neural network for real-time spatial positioning of objects" 66 pages, 23 figures, 2 tables, 22 sources, 4 supplements.

REAL-TIME VIDEO STREAM, OBJECT DETECTION, COMPUTER VISION, MACHINE LEARNING, NEURAL NETWORKS, OBJECT POSITIONING, YOLO.

Object of research: The algorithm for developing applications that enable object tracking in space using artificial intelligence.

Aim of the study: To explore contemporary methods of object detection within a frame, analyze model development techniques, create an application for object detection, and determine the spatial position of the object relative to the camera in real time.

Research methods: Comparative analysis of modern computer vision techniques, examination of dataset formation approaches for model training, and practical implementation of an algorithm for object detection and spatial positioning based on camera input or other real-time video streams.

This thesis investigates object detection algorithms and methods for constructing datasets to solve the problem of determining the spatial positioning of objects. It also presents the development of a prototype application for object detection using artificial intelligence, capable of determining object coordinates relative to the camera.

ЗМІСТ

Завдання на кваліфікаційну роботу 2

Реферат 4

Summary 5

Вступ 7

1 Теоретичні відомості 9

1.1 Загальний огляд CNN 9

1.2 Огляд нейронних мереж для виявлення об’єктів 13

1.3 Аналіз архітектури нейронних мереж для детекції в реальному часі 17

1.4 Методи визначення об’єкту у просторі 18

1.5 Висновок до розділу 1 20

2 Розробка додатку для спостереження 21

2.1 Вибір технологій 21

2.2 Підготовка датасету 22

2.3 Побудова моделі та навчання 27

2.4. Створення додатку для виявлення об’єкту 33

2.5 Висновок до розділу 2 40

3 Тестування 40

3.1. Експериментальні дослідження та спостереження 40

3.2. Подальші кроки 41

4 Висновки 43

Перелік посилань 44

Додаток А Отримання зображень з відеофайлів 46

Додаток Б Заповнення датасету випадковим набором відповідних файлів 50

Додаток В Скрипт для навчання моделі 54

Додаток Г Детектор 58

ВСТУП

Зі стрімким розвитком та поширенням штучного інтелекту пов’язано швидке розповсюдження використання машинного зору. Ось неповний список прикладів використання машинного зору в різних галузях:

- Транспорт та автомобілі – безпілотні автомобілі (Tesla Autopilot, Waymo), системи допомоги водію для контролю смуги, екстренного гальмування (ADAS);
- Охорона здоров’я та медицина – аналіз медичних зображень на КТ, МРТ, рентгенівських світлинах;
- Безпека та спостереження – розпізнавання обличь, для виявлення підозрілих у містах скупчення людей, магазини без кас, аналіз зображення з камер відеоспостереження для виявлення підозрілої активності;
- Виробництво та промисловість – системи спостереження за конвеєрами для виявлення браку;
- Розваги та електроніка – фільтри в Instagram, TikTok, Snapchat розпізнають риси обличчя та додають фільтри, шоломи ВР аналізують напрям зору ока для кращого налаштування зображення.

Одною з важливіших функцій для багатьох цих галузей є визначення об’єкту у просторі відносно камери визначення відстані до об’єкту. Така необхідність виникає в наступних ситуаціях:

- Прийняття рішення у реальному часі, що, наприклад, застосовується у робототехніці;
- Забезпечення безпечного та точного руху для безпілотних систем;
- Автоматизація у промисловості, наприклад, при виявлені браку на конвеєрі;
- Аналітика та прогнозування траєкторій руху об’єктів.

Розробка нових дешевих та ефективних методів для реалізації цієї функціональності є актуальною науково-технічною задачею.

Метою цієї роботи є аналіз існуючих методів виявлення об’єкту, створення простого способу розрахування відстані до нього, розробка прототипу додатку для відстеження необхідного об’єкту та обчислення його координат відносно камери.

Для досягнення поставлених цілей буди визначені наступні кроки:

- Провести детальний аналіз існуючих методів та алгоритмів для виявлення об’єктів та визначення їхнього просторового положення, виявити їхні переваги та недоліки, визначити необхідний метод для розробки;
- Обрати метод збору даних для навчання моделі, побудувати модель, створити прототип додатку для обчислення координат;
- Провести експериментальні дослідження, оцінити отримані результати.
- Визначити напрями подальших досліджень.

Наукова новизна одержаних результатів полягає у розробці архітектури однокрокового детектора виявлення об’єкту з визначенням відстані до нього без використання додаткових сенсорів та стереоскопічної камери.

Отримані результати можуть бути застосовані у різних галузях, в тому числі: автоматизовані лінії на виробництві, системи спостереження за рухом об’єктів, такі як CCTV, системи допомоги людям з пошкодженням зору, тощо.

Кваліфікаційна робота складається з вступу, трьох розділів , загальних висновків, списку використаних джерел та додатків.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Загальний огляд CNN

Серед нейронних мереж є відокремлений тип, який «вміє бачити» об’єкти на зображеннях – згорткові нейронні мережі (Convolutional Neural Networks, CNN). CNN автоматично виділяють ознаки, не потребують обов’язкового ручного створення фільтрів. Такі моделі вчяться знаходити закономірності, які будуть використані для класифікації або детекції об’єктів, прямо з даних. М. Ельгенді в свої книзі «Deep Learning for Vision Systems» наводить загальну схему CNN мережі (рис. 1).

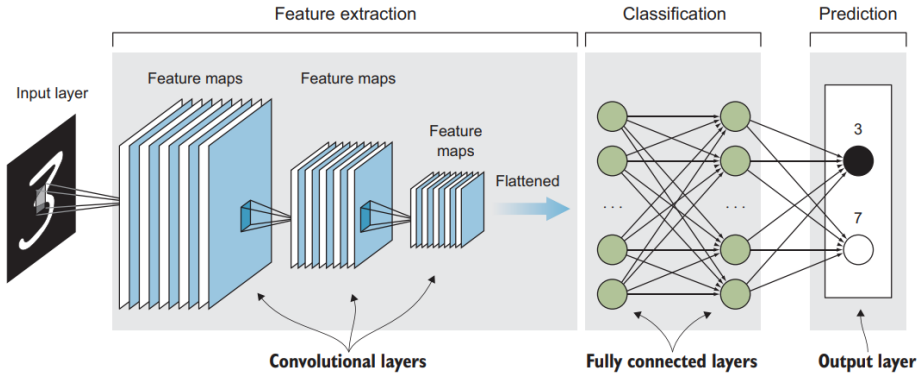


Рисунок 1 – Блок-схема CNN: вхідний шар (input layer), згорткові шари (convolutional layers), повністю зв’язані шари (fully connected layers) та прогнозування виходу (output prediction)[1]

Опис кожного елементу CNN представлений нижче.

Input layer

Мережа отримує вхідні дані, наприклад, кольорове зображення розміром 320x320, що для мережі є матрицею 320x320x3 (висота * ширина * кількість каналів).

Зазвичай кожен піксель це значення в діапазоні від 0 до 255, що є незручним для нейронної мережі: градієнти можуть бути занадто великими

(або малими), складно мінімізувати втрати, нейронна мережа може бути упередженою до каналів з високим (на відміну від інших каналів) показником. Отже на цьому етапі проводиться нормалізація пікселів за двома можливими варіантами.

Нормалізація пікселів, варіант 1. Значення кожного пікселя масштабується до значення в межах [0, 1]:

$$x' = \frac{x}{\mathbf{255}}, \tag{1.1}$$

де: x' – нове значення пікселя;

x – вхідне значення пікселя;

255 – максимальне значення пікселя.

Нормалізація пікселів, варіант 2. Стандартизація (або z-нормалізація) пікселя:

$$x' = \frac{x - \boldsymbol{\mu}}{\sigma}, \tag{1.2}$$

де: x' – нове значення пікселя;

x – вхідне значення пікселя;

$\boldsymbol{\mu}$ – середнє значення пікселів по датасету (наприклад [0.485,0.456,0.406] для ImageNet);

σ – стандартне відхилення по датасету (наприклад [0.229,0.224,0.225] для ImageNet).

Також на першому етапі проводиться аугментація даних – випадкові зміни вхідних зображень під час тренування моделі. Така процедура дає можливість більше навчатися та менше «запам’ятовувати» зображення.

Аугментація може мати різні типи:

– Вирізається випадкова частина зображення – Crop;

- Віддзеркалення зображення – Flip;
- Поворот зображення – Rotation;
- Зміна освітлення – Brightness/Contras;
- Додавання розмиття або шуму – Blur/Noise;
- Вирізається випадкова частина об’єкту – Cutout;
- Поєднання двох зображень – MixUp / CutMix.

Convolutional layers

Цей фрагмент є ключовим для згортових нейронних мереж, призначений для автоматичного виділення ознак зображення. CNN можуть самостійно навчатися фільтрам, які знаходять окремі патерни, такі як краї, кути, текстури, фрагменти об’єктів або їх комбінації.

Властивості згорткових шарів представлені далі.

Згортка (Convolution) – маленький фільтр (ядро, kernel) обробляє невелику область зображення. Ядро це матриця чисел розмірністю 3x3, що виконує скалярний добуток до фрагменту зображення того ж розміру. Отримане число – відгук фільтра, що показує, наскільки локальна структура зображення збігається з тим патерном, який «вивчив» фільтр під час навчання. Таким чином формується карта ознак, яка подається на наступні шари нейронної мережі. Цей крок суттєво зменшує кількість параметрів для нейронів, наприклад: 320x320x3 = 307200 входів для 64 нейронів це 64x307200 = 19660800. У випадку з фільтром кількість параметрів лише 3 x 3 x 3 x 64 = 1728.

Спільне використання ваг (Weight sharing). Через застосування одного фільтру для всіх частин зображення відповідного розміру, в нейронах застосовуються одні й ті самі ваги, що суттєво знижує обсяг пам’яті та ризик перенавчання.

Використання карт ознак (feature maps). Кожен фільтр (kernel) застосовується для пошуку на зображенні ознак притаманних цьому фільтру: вертикальні лінії, кути, форми, краї, тощо. Таким чином на ранніх шарах карти представлять примітивні речі, а за рахунок комбінацій карт на більш

глибоких шарах відбувається об’єднання карт, для виявлення більш складних об’єктів. Також карти допомагають відділення реальних об’єктів від шуму (перешкоди, шум матриці камери, освітлення, тощо)

Зростання рецептивного поля. Рецептивне поле — це область вхідного зображення, яка впливає на активацію конкретного нейрона. У міру поглиблення CNN рецептивне поле збільшується, оскільки кожен наступний згортковий шар операційно охоплює дедалі більшу частину вхідного простору. Зростання рецептивного поля дозволяє моделі переходити від локальних ознак до високорівневих структур і цілісних об’єктів.

Активації після згорток. Після кожного згорткового шару стоїть нелінійна функція активації (зазвичай ReLU, але може бути й Leaky ReLU, GELU, SiLU, тощо), завдяки чому модель здатна розпізнавати складні залежності та відокремлювати реальні ознаки від шуму.

Наприклад реакція фільтра за допомогою ReLU: $f(x) = \max(0, x)$, може бути:

- сильна – ознака присутня,
- слабка – ознака відсутня,
- шумова – відсічена до нуля.

Fully connected layers

Завершальний шар CNN, який виконує роль класифікатора. На відміну від згоркових шарів, FC-шари мають повні зв’язки між усіма нейронами, таким чином відстежуються усі зв’язки між ознаками, що в свою чергу дозволяє «бачити» об’єкти повністю, класифікувати їх або приймати рішення. Звісно цей шар має свої недоліки, такі як велика кількість параметрів та схильність до перенавчання. Використання FC-шару суттєво збільшує складність моделі.

Output prediction

Фінальний етап роботи згорткової нейронної мережі, на якому модель перетворює всі представлення ознак на конкретний результат, залежно від задачі – класифікація, детекція, сегментація, тощо.

1.2 Огляд нейронних мереж для виявлення об’єктів

Звичайні CNN, які застосовуються в завданнях розпізнавання зображень, обмежуються класифікацією всього зображення як єдиного цілого, та нездатні виконувати локалізацію кількох об’єктів. Але останнім часом було розроблені спеціалізовані архітектури, що суттєво поширюють функціональні можливості класичних CNN. Розглянемо ключові відмінності сучасних детекторів.

Багатовихідна структура

На відміну від класичних CNN, детектори об’єктів виконують дві задачі одночасно: класифікація об’єкту (визначення його категорії) та локалізація об’єкту (регресія координат обмежувального прямокутника, bounding box). Це досягається за рахунок використання детектором двох або більше паралельних вихідних гілок, класифікації та регресії відповідно. Такий підхід суттєво ускладнює детектор, через те що тренування моделі вимагає одночасної оптимізації класифікаційної та регресійної помилки.

Обробка множини об’єктів на зображенні

Сучасні детектори застосовують наступні додаткові механізми.

- Розбиття зображення на сітку, визначення в якій клітинці знаходиться центр об’єкту. Так кожна клітина відповідає за маленьку частину зображення, відповідно зменшує кількість конфліктів під час навчання. Що спрощує тренування.
- Прогноз на різних рівнях масштабів – на ранніх шарах визначаються мали об’єкти, великі на глибших шарах.
- Генерація великої кількості рамок- кандидатів (bounding boxes), та їх одночасна валідація. Це дозволяє працювати з динамічним числом об’єктів різних типів.

Використовуються, також, додаткові архітектурні компоненти, яких не існує в класичних CNN.

– Region Proposal. Попередня генерація області-кандидата, де потенційно може бути об’єкт. Валідація таких областей зменшує їх кількість, що робить обчислення можливим.
– Anchor Boxes. Попереднє визначення шаблонів bounding boxes різних форм і розмірів, які «закріплені» у кожній точці feature map. Такий підхід уніфікує процес регресії координат, передбачає об’єкти різних масштабів.
– Feature Pyramid Networks (FPN). Створення багаторівневих карт ознак для детекції за різними масштабами.
– Non-Maximum Suppression (NMS). Валідація згенерованих областей-кандидатів, де потенційно можуть бути об’єкти.

Оптимізація під сценарії реального часу

Найважливіша функція детекторів – можливість працювати в режимі реального часу. Тому багато моделей орієнтуються на високу швидкість обробки (до 200 FPS), ефективне розподілене обчислення, можливість роботи на вбудованих чи мобільних пристроях.

Розглянемо приклади таких архітектур.

R-CNN (Regions with Convolutional Neural Networks). Принцип роботи полягає у генерації багатьох регіонів-кандидатів. Кожний такий регіон передається до CNN та обробляється окремо, для кожного регіону виконується два завдання – визначається клас об’єкту та уточнює координати bounding box. Схематично можна відобразити як на рисунку 2.

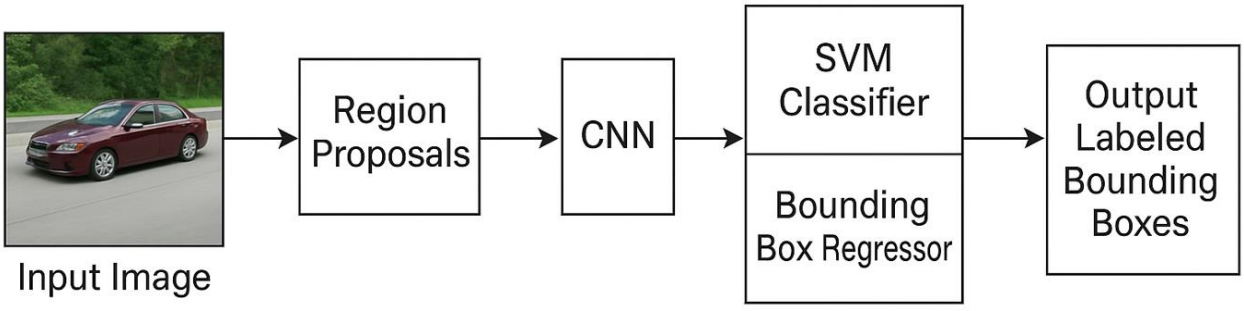


Рисунок 2 – Схематичне представлення роботи R-CNN

Faster R-CNN. Ця архітектура усуває недоліки попередньої. Створюється карта ознак через CNN, RPN формує anchor boxes різних розмірів в кожній точці цієї карти, для кожного anchor виконується 2 прогнози, чи є тут об’єкт та виконується корекція рамки. Далі регіони приводяться до одного масштабу (за допомогою алгоритмів Rol Pooling у Faster R-CNN або Rol Align у більш сучасній Mask R-CNN) та додаються у full-connected шари. У фіналі модель прогнозує клас об’єкту та уточнену рамку за координатами та розмірами. Схема зображена на рисунку 3.

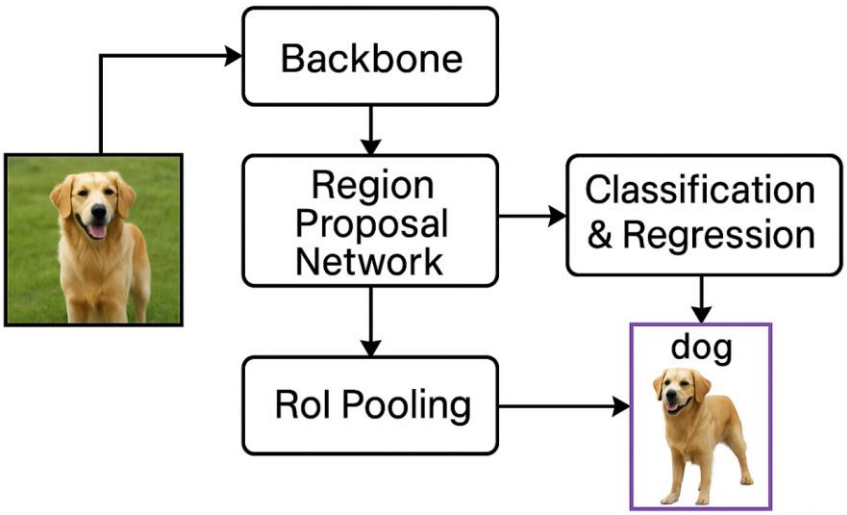


Рисунок 3 – Схема роботи Faster R-CNN

SSD (Single Shot MultiBox Detector). Є так званим одноетапним детектором, який виконує виконує класифікацію і локалізацію за один прохід мережі. Спочатку генеруються карти ознак різного розміру, на кожному пікселі кожної карти додаються anchor boxes із різним співвідношенням сторін (таким чином в середньому перевіряються тисячі bounding boxes). Відсіюються багато overlapping рамок, залишаються найкращі. Далі, у вигляді простих згорткових операцій, паралельно виконується прогнозування класів і координат. Схема представлена на рисунку 4.

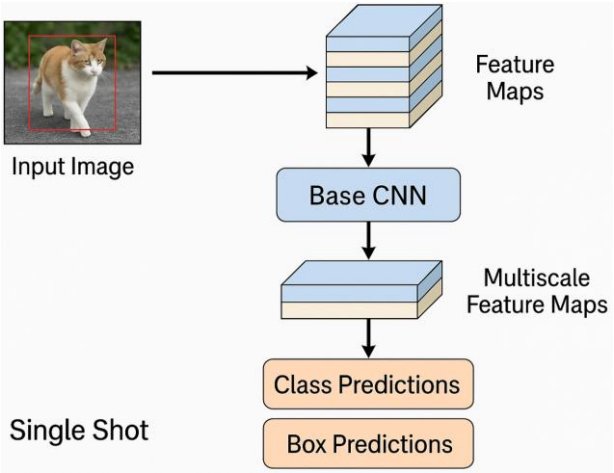


Рисунок 4 – Схема роботи Single Shot MultiBox Detector

YOLO (You Only Look Once). Станом на сьогодні це найвідоміша та найефективніша одноетапна архітектура. Створюється карта ознак (зображення проходить через глибоку CNN), виконується мульти-масштабна детекція з використанням Feature Pyramid Network (дає трирівневу систему з малими, середніми та великими об’єктами). Наступним кроком виконується прогноз bounding boxes через anchor boxes або anchor-free – для кожної точки feature map прогнозується зміщення центру, ширина та висота, клас. NMS об’єднує та очищує рамки. Схема представлена на рисунку 5.

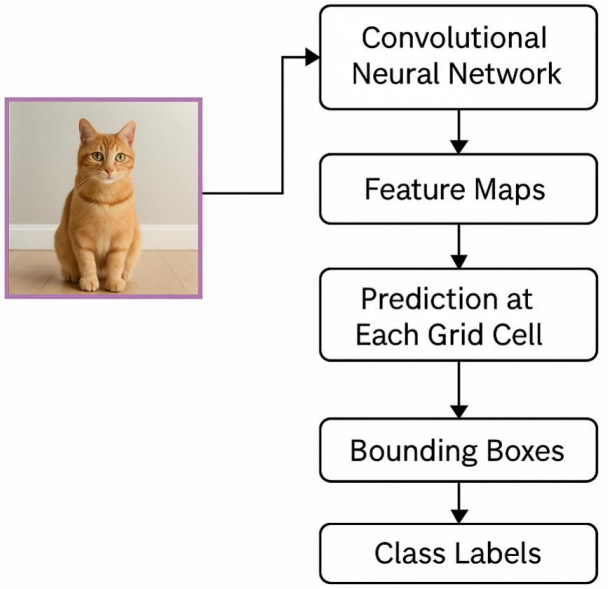


Рисунок 5 – Схема роботи YOLO

1.3 Аналіз архітектури нейронних мереж для детекції в реальному часі

Узагальнюючи огляд з розділу 1.2, можна визначити ключові вимоги щодо архітектури нейронних мереж для виявлення об’єктів, які можуть працювати в режимі реального часу.

Проблема real-time, вимоги до мережі.

Затримка (latency). Кожна камера має одну з ключових властивостей – частоту кадрів (FPS). Оскільки зчитування наступного кадру відбувається після завершення обробки попереднього, надмірна обчислювальна складність призведе до зменшення FPS Мета – не допустити падіння нижче 30 кадрів (залежить від заліза).

Пропускна здатність моделі. Прискорення обробки кадрів напряму залежить від кількості операцій яку виконує нейронна мережа, отже задача обрати модель мінімальною кількістю операцій, мінімально можливу кількість параметрів

Ефективний backbone – це та частина нейронної мережі, яка витягує ознаки зображення з максимальною ефективністю при мінімальних обчислювальних витратах. Отже слід обратти мережу яка має необхідну архітектурну оптимізацію.

Підтримка паралельних обчислень. Такий підхід дає змогу збільшувати обчислювальну потужність у випадках коли це необхідно. Гіганти IT індустрії такі як Microsoft, Google, AWS, IBM пропонують хмарні рішення для паралельних обчислень та надають необхідні інструменти для керування цим. Втім існують і локальні рішення для вирішення задач з паралельним обчислюванням.

Архітектурний підхід до оптимізації обчислення, вимоги.

One-Stage detectors – за один прохід нейронної мережі ми маємо визначити все необхідне для досягнення поставленої задачі в межах обробки одного кадру.

Відсутність етапу region proposal – це додатковий етап первинної обчислення зображення, що може підвищити точність, але суттєво збільшує час обчислення.

Постійна складність обчислення (O(1)) – час обробки кадрів має бути стабільним незважаючи на кількість об’єктів у кадрі. Кількість обчислень має залежати тільки від сітки вихідних прогнозів. Отже детектор має мати grid-підхід – створювати регулярну сітку для детекції об’єкту.

Перераховані пункти вимог не є обов’язковими для беззаперечного виконання робот по оптимізації, але слід враховувати, що саме ці риси нейронної мережі можуть суттєво підвищити продуктивність.

1.4 Методи визначення об’єкту у просторі

Визначення просторових координат є ключовою задачею для комп’ютерного зору та робототехніки (подрібніше це розглянуто у вступі). Задача полягає у з’ясуванні розташування об’єкта відносно умовної точки відліку. У цій роботі за точку відліку рахується камера спостереження. Методи визначення умовно можна поділити на 2 групи – з використанням ШІ (базується на роботі глибинної нейронної мережі) та без нього (розрахунки базуються на показниках фізичних сенсорів та геометричних закономірностях).

Методи без застосування штучного інтелекту.

Тріангуляція. Метод, що дозволяє визначити координати точки завдяки спостереженням з двох або більше камер (іноді з однієї камери але з декількох позицій). З відомої матриці параметрів та напрямків на об’єкт знаходиться точка перетину (необхідне позиціювання об’єкту). Метод має високу точність але складний в реалізації, за рахунок складної синхронізації камер.

Perspective-and-Point. Метод використовується коли в сцені (у просторі за яким спостерігають) заздалегідь існують маркери, що дозволяють визначити глибину. Це один з найточніших методів, але не завжди доступний.

Лазерні далекоміри (LiDAR). Такі сенсори вимірюють глибину, за рахунок вимірювання часу повернення лазерного імпульсу. Це досить точний метод, але є проблеми з вимірюванням, якщо об'єкт недостатньо щільний, чи має поверхню, що здатна поглинати світло.

Ультразвук. Має аналогічний принцип що і LiDAR. Дешевший, але чутливий до перешкод.

Time-of-Flight камери. Метод використовує спеціальні інфрачервоні камери які вимірюють час повернення сигналу для визначення глибини. Точний, але дорогий метод, має обмежену дальність та чутливий до зовнішніх перешкод з ІЧ-світлом.

Методи з використанням ШІ.

Методи цієї групи використовують глибинні нейронні мережі для визначення глибини зображення, побудови 3D сцени або прямого положення за координатами.

Монокулярна оцінка глибини. Мережа вчиться перетворювати одне зображення в мапу глибин за ознаками на зображенні. Приклади: MiDaS, DPT, MegaDepth. Відрізнається відносною точністю, краще підходить для великих об'єктів.

Стереозір. Побудова мапи невідповідностей двох зображень, яке потім перетвориться глибину. Приклади моделей: PSMNet, GA-Net, RAFT-Stereo. Вимагає або стереоскопічну камеру (стереомодуль), або синхронізацію двох камер.

Побудова прямої 3D-локалізації. Моделі одразу намагаються будувати тривимірну сцену та обчислювати координати об'єкту на ній. Приклади: CenterNet3D, YOLO-POSE 3D, SMOKE, RTMDet-3D, PoseCNN, PVNet. Характеризується дуже високою складністю навчання, потребою у великих датасетах для побудови сцени, втрачає гнучкість коли сцена змінюється.

Комбінація детекції та оцінки глибини. Це один з найпоширених методів для роботи в режимі реально часу. Він полягає в наступному:

- Визначається контур об’єкту (за допомогою Faster R-CNN ,YOLO, SSD);
- Контур поєднується з мапою глибини;
- Обчислюються координати об’єкта.

Відзначається високою швидкістю при відносній точності.

1.5 Висновок до розділу 1

У цьому розділі було розглянуто основні архітектури нейронних мереж, що використовуються для задачі виявлення об’єктів, а також проведено їх аналіз з точки зору придатності для роботи в режимі реального часу. Було проведено аналіз принципів побудови двоетапних та одноетапних детекторів, з’ясовано їх переваги та обмеження, а також визначено ключові вимоги, які висуває робота зі стрімким відеопотоком: низька затримка, висока пропускна здатність моделі, ефективний backbone та підтримка паралельних обчислень.

У результаті проведеного огляду встановлено, що саме одноетапні моделі (one-stage detectors) найкраще відповідають вимогам реального часу завдяки відсутності етапу region proposal, фіксованій обчислювальній складності та використанню grid-підходу. На основі порівняння різних архітектур визначено, що моделі сімейства YOLO забезпечують оптимальний баланс між швидкодією, точністю та апаратною ефективністю, що робить їх найбільш доцільним вибором для подальшої реалізації системи виявлення об’єкта у режимі реального часу.

У розділі було проведено огляд методів визначення відстані (глибини зображення) до об’єкту, що дає можливість підібрати найбільше вигідний для виконання завдання до цієї роботи.

2 РОЗРОБКА ДОДАТКУ ДЛЯ СПОСТЕРЕЖЕННЯ

В цьому розділі розглядається практична сторона розробки додатку для спостереження за об’єктом та його визначення в просторі відносно камери.

2.1 Вибір технологій

Технічні та програмні засоби створення додатку:

- Камера для підготовки датасету – 12-мегапіксельна телефотографічна (45^0) з 2-кратним зумом та програмою оптичної стабілізацією для запобігання розмиттю зображень (Samsung Galaxy S10);
- Розробка та навчання моделі – MacBook Pro M4 24Gb/1Tb 14”;
- Камера для спостереження 12-мегапіксельна центральна камера з MacBook Pro M4, з програмним зниженням роздільної здатності до 1280x720;
- Мова програмування Python з додатковими бібліотеками: ultralytics, opencv-python.

Для розробки було обрано YOLO. Чому саме YOLO?

З огляду на вимоги до задачі — виявлення об’єкта і визначення його координат у просторі в режимі реального часу, — YOLO є оптимальним вибором з таких причин:

- Забезпечує найнижчу затримку серед популярних архітектур (важливо при роботі з відеопотоком);
- Має фіксовану обчислювальну складність та стабільний FPS незалежно від кількості об’єктів у кадрі;

- Має оптимізований та ефективний backbone, що дозволяє працювати на різному обладнанні — від GPU до edge-пристроїв;
- Підтримує anchor-free head, що прискорює обчислення та зменшує кількість гіперпараметрів;
- Забезпечує високу точність при мінімальних витратах, що критично для реального часу;
- Має розвинену екосистему та інструменти, що значно пришвидшує практичну реалізацію.

Для визначення відстані до об’єкту обрано метод поєднання детекції нейронною мережею та додатковими розрахунками масштабу рамки з урахуванням геометричних властивостей перспективи камери.

Визначення позиціювання об’єкту відносно камери буде розраховано додатковим алгоритмом після отримання фрейму та необхідних показників що до об’єкту.

2.2 Підготовка датасету

Кожна робота з моделями штучного інтелекту починається з підготовки датасету. – набору даних який необхідно підготувати для навчання моделі. Це один з ключових процесів у розробці. Якість набору даних впливає на якість навчання моделі, на стабільність та точність нейронної мережі. Особливо це важливо для систем, що працюють в режимі реального часу. Емпірично доведено, що моделі можуть час від часу “вгадувати” об’єкти на статичних зображеннях.

Для збору даних є різні підходи, розглянемо їх, їхні переваги та недоліки.

Ручний збір даних. Метод полягає у створенні датасету самостійно за допомогою камери та варіюванні різних умов, що відрізняються за освітленням, ракурсі, чіткості зображення, тощо. Переважно використовувати

ту саму камеру, або найближчу за технічними характеристиками, що буде використовуватися в кінцевій системі. Основна перевага такого методу – використання необхідного об’єкту, модель навчатиметься на розпізнавати . Головний недолік – дуже затратно за часом.

Збір даних з відкритих джерел. Зі зростанням попиту на контент стали розповсюдженні відкриті бази зображень такі як COCO, Open Images, Pascal VOC, KITTI. Вони дають відкритий доступ до великого набору даних. Також можна застосувати API популярных пошукових систем, таких як Google, Bing, Yahoo, що дають можливість шукати зображення за тематикою, кольоровою гамою, розміром та іншим показникам. Це найшвидший спосіб збору даних, що є перевагою. Недолік – об’єкти не завжди співпадають з реальним об’єктом спостереження (виявлення), дійсно великі набори даних можуть потребувати додаткових коштів, особливо в пошукових системах.

Генерація зображень (Synthetic Data). Створення датасету за допомогою штучного інтелекту. Дешевий спосіб, що дозволяє створити будь які варіації зображень та 3D-об’єктів (diffusion/GenAI). Головний недолік – зображення не завжди відповідають об’єкту в реальному світі, що може ускладнювати навчання моделі.

Додатково слід згадати такий метод як *аугментація.* Ідея полягає не в створенні зображення, а в модифікації існуючого. За допомогою додаткових інструментів можна змінити освітлення, додати блюр, інвертувати зображення, збільшити чи зменшити. Таким чином можна збільшити набір даних для навчання.

Отже для підготовки датасету слід враховувати:

- Кількість примірників (зображень) з об’єктом та без нього;
- Рівномірний розподіл зображень з різними відстанями до об’єкту;
- Запобігати надмірної аугментації (перебільшено додавання шумових ефектів).

В цій роботі використовувався перший метод. Об’єкт знімався на різних локація з різним освітленням та з різними наближенням на камеру.

Використовувалася відеозйомка в роздільній здатності 1280x720. Після цього за допомогою невеликого скрипта на Python з усіх знятих відео вирізаються та зберігаються зображення (з об’єктом, позитивні) для набору даних. Чим більше тривалість відео, тим більше зображень з них було отримано. Загальна кількість 700. У такий самий спосіб знімалися відео без об’єкту (негативні) які необхідні для навчання моделі, і зберігалися окремо. До сортування було вилучено по 700 позитивних та негативних зображень. Для побудови прототипу такого обсягу кадрів буде достатньо, але для реального проєкту слід забезпечити модель 5000+ зображеннь. Скрипт додано у додатку А.

Фільтрація та анотація даних. Коли файли отримані, слід перевірити позитивні зображення та видалити неприпустимі – де об’єкт майже не видно, зображення змазане чи містить зайві шуми, через яких об’єкт не видно очима. В межах цієї роботи після фільтрації лишилося 425 зображень.

Анотація – процес позначення об’єкту на зображенні, за рахунок чого будуть створені вказівки для нейронної мережі як виглядає об’єкт. Фактично це маркування об’єкту та створення опису, за якими координатами на зображенні цей об’єкт знаходиться. Для анотації існує багато інструментів:

- CVAT — професійний інструмент, підтримує всі типи анотацій;
- LabelImg — для простих bounding boxes;
- Label Studio — універсальний інструмент;
- Roboflow Annotate — зручний онлайн-сервіс.

Для цієї роботи було обрано makesense (https://www.makesense.ai/). Це простий інструмент, який працює одразу в браузері, але при цьому не треба нічого відправляти на сторонній сервіс. Інтерфейс та приклад анотації зображено на рисунку 6.

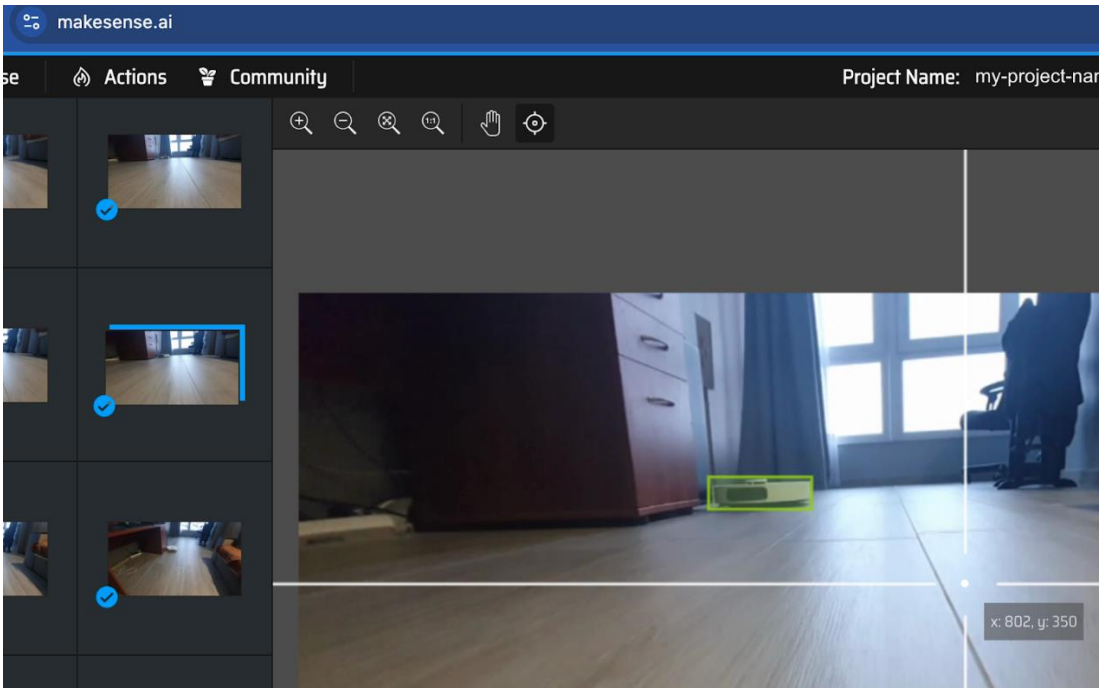


Рисунок 6 – Приклад анотації за допомогою makesense

Після маркування під кожний файл зображення буде створено текстовий файл з вказівками де знаходиться об’єкт на рисунку 7.

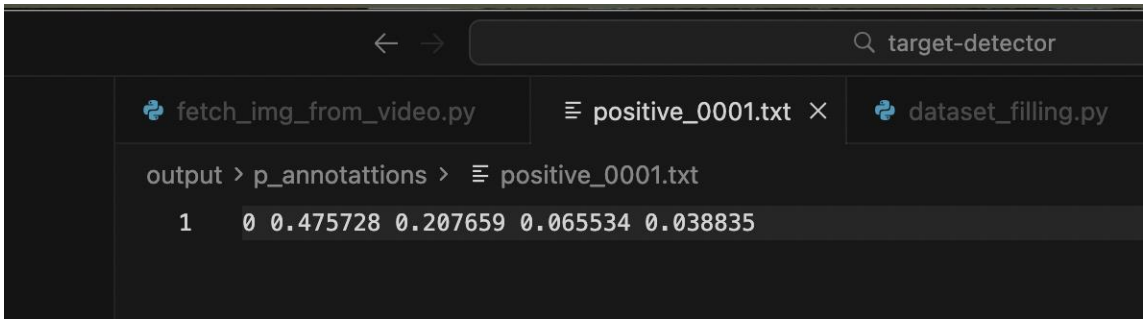


Рисунок 7 – Файл анотації зображення з ім’ям positive_0001

Таким чином на кожне позитивне зображення отримуємо файл позитивної анотації, в межах цього проєкту 425. Також необхідно створити необхідні порожні анотації для негативних зображень. В проєкті їх 700. Така пропорція позитивних:негативних зображень зазвичай краща. Завеликий обсяг позитивних зображень відносно негативних може призвести до того, що модель частіше буде бачити об’єкт там де його нема. Вірно і зворотне – за багато негативу відносно позитиву може призвести до того, що модель

більше пропускатиме об’єкт. Отже слід прямувати до відношення позитив:негатив = 1:1,5 (до 1,75).

В результаті для початку навчання є необхідний набір – 425 позитивних зображень з анотаціями та 700 негативних зображень з анотаціями, структурованих як на рисунку 8.

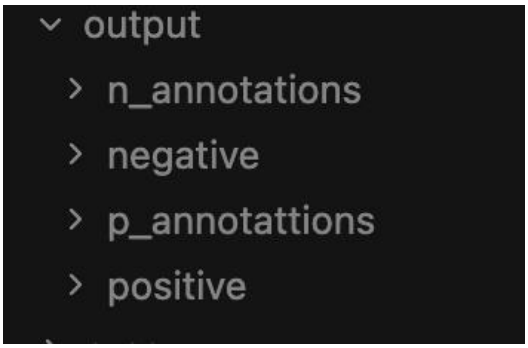


Рисунок 8 – Набір для формування датасету

Останній крок перед навчанням – формування датасету, структурованого набору даних, який буде використовувати модель на проєкті. Для навчання моделі будемо використовувати такі пропорції : ~70% – для тренування, ~20% – для валідації, ~10% – для тестування. За кількістю файлів це буде – 788 (298 позитивних + 490 негативних), 225 (85 позитивних + 140 негативних) та 112 (42 позитивних + 70 негативних) зображень відповідно. У такий спосіб розподілено і анотації і збережені в окрему теку.

Слід створити окрему директорію для зберігання структури датасету для навчання, як на рисунку 9.

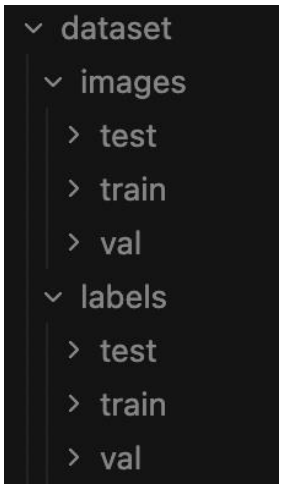


Рисунок 9 – Структура датасету для навчання моделі:
images – зображення, labels – відповідні анотації

Скрипт який заповнює теку dataset наведено у додатку Б. Слід зауважити, що заповнення випадковими зображеннями з набору не є кращою практикою, через те, що є вірогідність, що наприклад до тесту потраплять занадто дрібні зображення об’єкту, а в тренувальному наборі їх буде мало. Такий розподіл може вплинути на якість роботи нейронної мережі. Краще зберігати баланс.

2.3 Побудова моделі та навчання

Для запуску тренування слід додати ще один файл – data.yaml. Він міститиме шляхи до нашого датасету для моделі

```
path: ./dataset  # шлях до папки, де зберігаються images/ та labels/
train: images/train
val: images/val
test: images/test
nc: 1  # кількість класів
names: [‘Target’]  # назва твого єдиного класу, можеш змінити
```

Рисунок 10 – Зміст файлу data.yaml

Тепер створюємо скрипт який запустить навчання моделі. Його лістинг з коментарями наведений в додатку В.

Результат тренування отримуємо у вигляді звіту, як за кожну епоху, так і за все навчання (додано у скороченому вигляді):

50 epochs completed in 14.048 hours.

Epoch | box_loss | cls_loss | dfl_loss | P | R | mAP50 | mAP50-95

------+---------+---------+---------+--------+--------+-------+---------

1 | 0.8143 | 6.5410 | 0.9638 | 0.998 | 0.612 | 0.723 | 0.549

10 | 0.8369 | 0.6304 | 0.9563 | 0.948 | 0.929 | 0.946 | 0.766

20 | 0.7354 | 0.4899 | 0.9399 | 0.991 | 0.929 | 0.962 | 0.805

30 | 0.6928 | 0.4485 | 0.9029 | 0.988 | 0.953 | 0.965 | 0.820

40 | 0.6104 | 0.3586 | 0.8755 | 0.999 | 0.953 | 0.972 | 0.839

50 | 0.5418 | 0.2892 | 0.8584 | 1.000 | 0.964 | 0.980 | 0.843

Optimizer stripped from …/znu/target-detector/runs/detect/yolov8_training/weights/last.pt, 22.5MB

Optimizer stripped from …/znu/target-detector/runs/detect/yolov8_training/weights/best.pt, 22.5MB

Validating …/znu/target-detector/runs/detect/yolov8_training/weights/best.pt...

Ultralytics 8.3.230 Python-3.13.9 torch-2.9.1 CPU (Apple M4 Pro)

Model summary (fused): 72 layers, 11,125,971 parameters, 0 gradients, 28.4 GFLOPs

Class Images Instances Box(P R mAP50 mAP50-95): 100% ━━━━━━━━━━━━ 8/8 6.8s/it 54.2s

all 225 85 1 0.964 0.98 0.849

Speed: 0.8ms preprocess, 236.1ms inference, 0.0ms loss, 0.1ms postprocess per image

Results saved to **/Users/d.burkovskyi/server/znu/target-detector/runs/detect/yolov8_training**

Рисунок 11 – Результат навчання моделі

Якщо розглянути подробніше за різними показниками то ефективність моделі під час тренування змінювалася наступним чином рисунки 12 – 18.

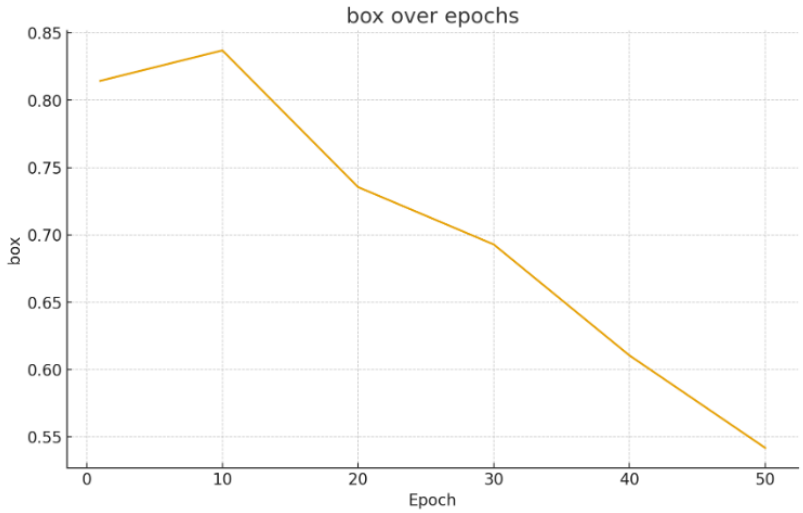


Рисунок 12 – box_loss — помилка регресії boundig

На рисунку 12 бачимо, що модель швидко навчилася правильно визначати розмір та позицію об’єкту, плавне падіння вказує на стабільний процес тренування.

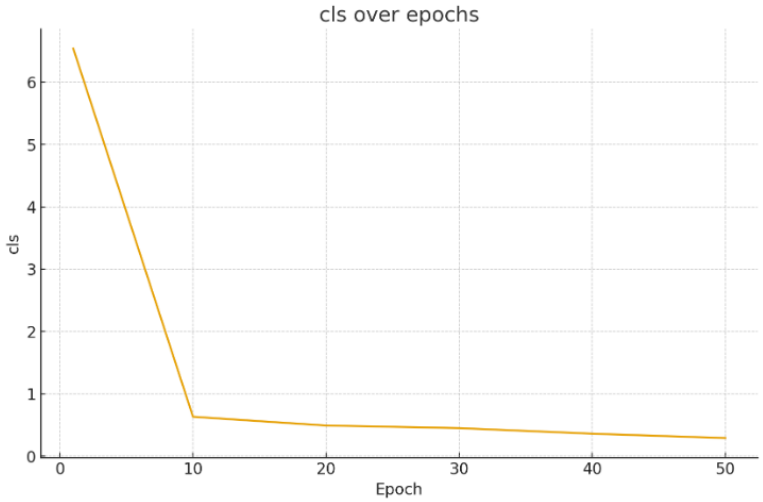


Рисунок 13 – cls_loss — помилка класифікації

Модель швидко навчилася розпізнавати клас об’єкту (рисунок 13). Хоча цей графік для роботи не дуже інформативний, через те що маємо лише один клас.

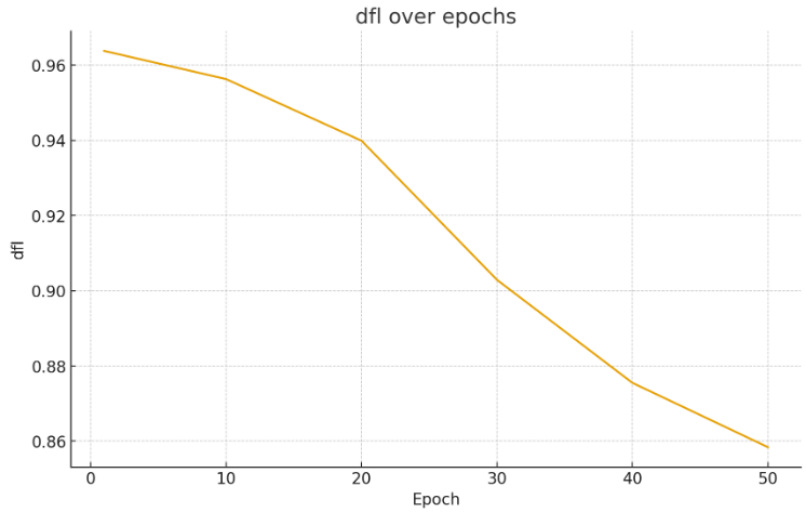


Рисунок 14 – dfl_loss — помилка Distribution Focal Loss

Модель стабільно вчилася визначати границі об’єкту. Те що не було «просідань» графіку, свідчить про ефективність моделі та відсутність випадкових передбачень.

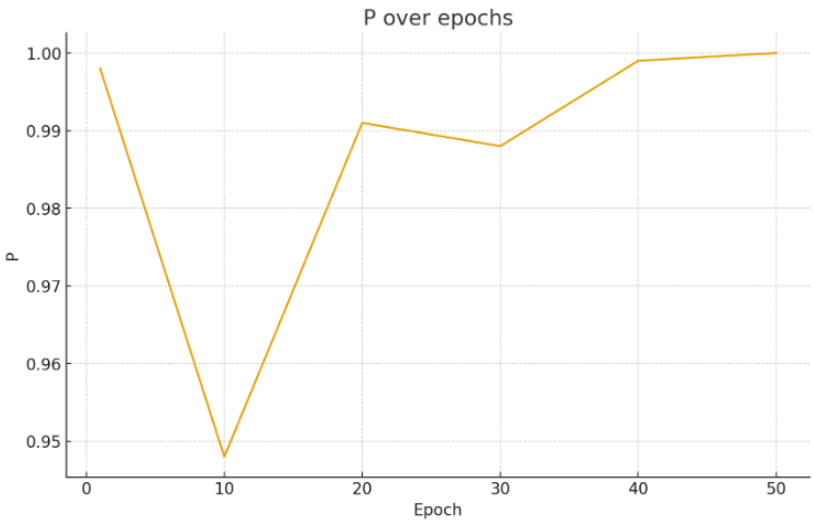


Рисунок 15 – Precision (P) — точність

На рисунку 15. Графік показує, що під кінець навчання модель робить всі передбачення правильно. Отже відбувається правильний підбір параметрів навчання та вказує на якісний датасет.

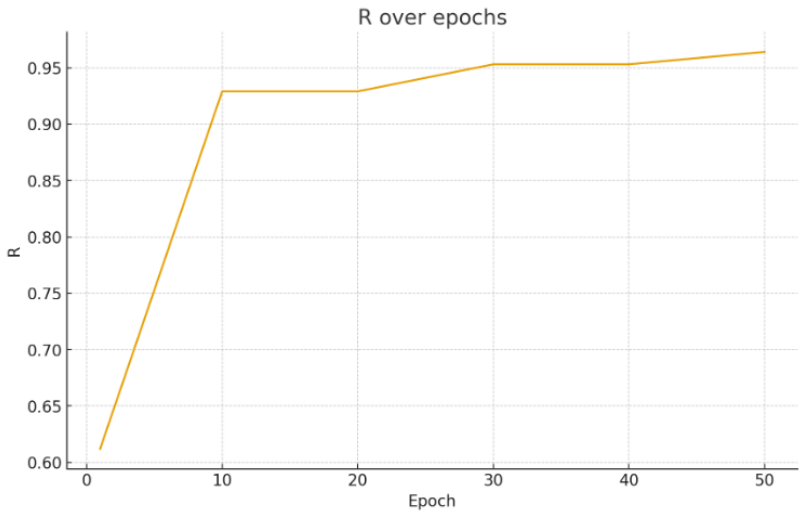


Рисунок 16 – Recall ® – повнота

Графік на рисунку 16 показує, що модель майже перестала пропускати об’єкти. Стабільний показник Recall вказує на те, що модель «знає» що шукати на кадрі.

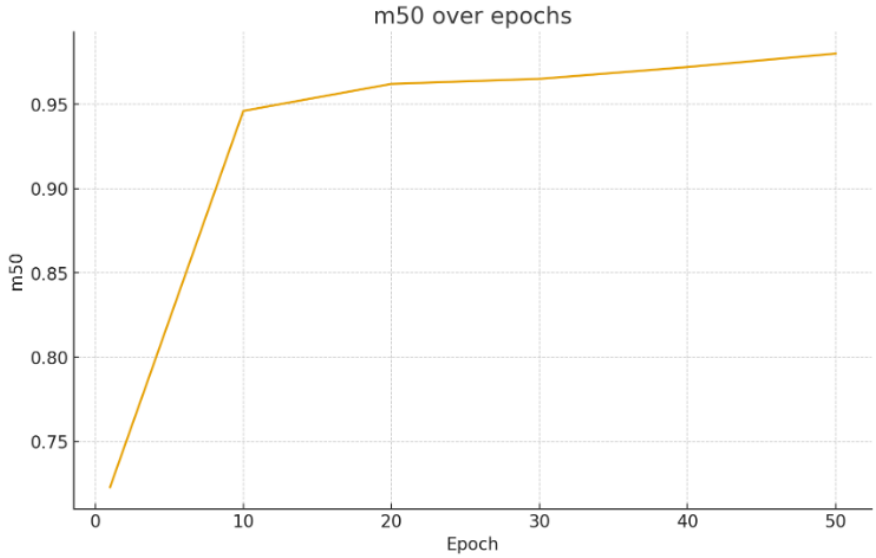


Рисунок 17 – mAP50 — точність детекції при IoU=0.5

Рисунок 17 показує графік з основною метрикою детекції об’єкту IoU – Intersection of Union. Це відношення площі перетину двох прямокутників (передбаченої рамки коло об’єкту та реальні границі об’єкту, що були визначені під час анотації).

$$IoU = \frac{Area(Predicated \cap True)}{Area(Predicated \cup True)} \quad (2.1)$$

де: IoU – Intersection over Union – показник наскільки добре передбачена рамка, показник змінюється від 0 (немає беретину) до 1 (повний перетин);

$Predicated$ – передбачена рамка;

$True$ – реальна рамка;

$\cap$ – площа перетину рамок;

$\cup$ – площа їх об’єднання (сума площ мінус сума перетину).

Значення IoU=0.5 при оцінюванні ефективності моделі, рахує правильну детекцію якщо площа перетину рамок становить 50% площі їх об’єднання.

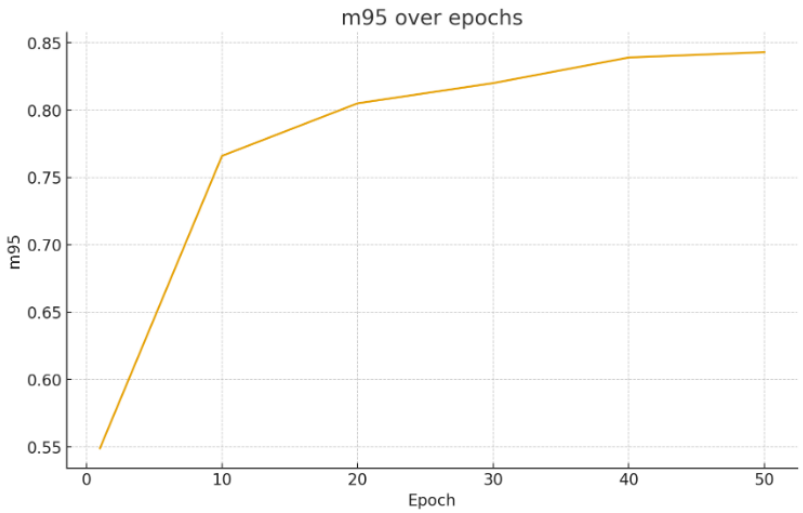


Рисунок 18 – mAP50-95 — строгіша метрика

На рисунку 18 зображено аналогічний з 17 графік, але правильним співпадінням рахується трохи інакше. Цей показник бере середнє значення метрики де IoU змінюється від 0.5 до 0.95 з кроком 0.05.

З цих графіків можна зробити висновок, що був зібраний якісний датасет, об’єкт та його анотація на зображеннях дозволяють моделі ефективно навчатися. Тренування протягом 50 епох демонструє стабільний та ефективний процес оптимізації. Вже після 10 епох більшість показників показували стабільні високі значення, та продовжували покращуватися. Такі показники навчання характерні для YOLO з невеликою кількістю класів. Під час навчання моделі було вказано припинити навчання у разі відсутності прогресу після 30 епох, але навчання продовжилося. Збільшити їх кількість не мало значення через незначну корисність подальшого навчання.

2.4. Створення додатку для виявлення об’єкту

Після того як нейронна мережа пройшла навчання буде згенеровано модель з епохи з найкращими показниками, яка буде розташована за адресою …/target-detector/runs/detect/yolov8_training/ weights/best.pt, та займаттиме 22.5MB після видалення оптимізатору.

За необхідністю ми можемо продовжити навчання моделі використовуючи модель що зберігається у файлі …/last.pt, де зберігається історія градієнтів, історія оновлення ваг по епохам та інша інформація, яка може бути корисною для продовження навчання. Але в цій роботі така процедура не проводиться.

Її і буде використано в програмі для аналізу кадрів (фреймів)

Для роботи з видеопотоком застосовуємо найпопулярнішу бібліотеку для Python – OpenCV. Для пошуку необхідного об’єкту застосовуємо навчену модель, аналізуємо кожний отриманий кадр залежно від результатів.

Додаток реалізує блок-схему, як на рисунку 19.

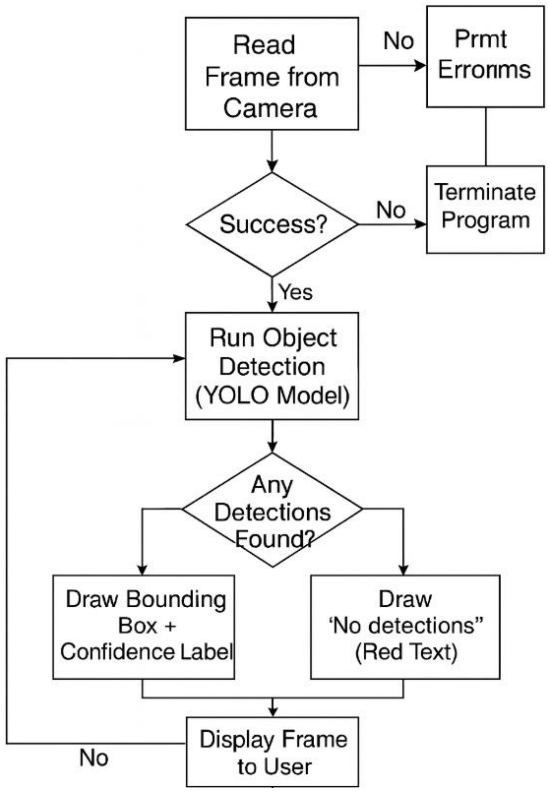


Рисунок 19 – Блок-схема детекції об’єкту з відеопотоку

Опис блок-схеми представлений нижче.

– Старт основного циклу. Запускається нескінченний цикл для постійного отримання та обробки кадрів з камери в реальному часі.
– Читання кадру з камери. На початку кожної ітерації зчитується новий кадр із відеопотоку. Якщо зчитування неможливе (наприклад, камера відключена), система виводить повідомлення про помилку та завершує роботу.
– Виконання детекції об’єктів. Поточний кадр передається до нейронної мережі YOLO для аналізу. Модель виконує пошук об’єктів на зображенні, орієнтуючись на заданий поріг впевненості. Цей крок буде найважливішим в подальшому алгоритмі аналізу фрейму.
– Візуалізація результату. На кадрі відображається прямокутник навколо виявленого об’єкта, а поруч виводиться текст із рівнем

впевненості. Це дозволяє візуально ідентифікувати об’єкт на зображенні.

- Обробка ситуації без детекцій. Якщо об’єкти не знайдено, на зображенні виводиться повідомлення «No detections» червоним кольором, яке сповіщає користувача про відсутність розпізнаваних цілей.
- Відображення кадру користувачу. Оброблений кадр із нанесеними елементами виводиться у вікні з відеопотоком.
- Перевірка на завершення. Програма постійно очікує натискання клавіші ESC. У разі її натискання цикл зупиняється і ініціюється процедура завершення роботи.
- Завершення та очищення ресурсів. Після виходу з циклу камера відключається, вікна закриваються, і користувачу виводиться повідомлення про завершення роботи програми.

Отриманий результат представлений на рисунку 20



Рисунок 20 – Визначення необхідного об’єкту в кадрі

Визначення положення об’єкту відносно центру кадру. Першим кроком слід визначити за яким алгоритмом буде відбуватися розрахунок відстані. Враховуючи, що в нашому розпорядженні є тільки камера, то розрахувати можна лише на зображення яке вона віддає. Отже з того що є камера + bbox на об’єкті, який надає нейронна мережа.

Калібрування камери. Заміряємо відстань при якій об’єкт займає всю ширину екрану – в цій роботі це 1280 пікселів на відстані 20 см, точніше 0,2 м. Реальні розміри об’єкту – 0,35 м.

Проекція об’єкту на сенсор камеры буде тим менша, чим більше відстань між камерою та об’єктом. Отже отримаємо пропорцію

$$\frac{perceived_{width}}{focal_{length}} = \frac{real_{width}}{known_{distance}}, \tag{2.2}$$

де: $real_{width}$ – реальна ширина об’єкта, в метрах;

$known_{distance}$ – відстань до об’єкта, в метрах;

$perceived_{width}$ – ширина об’єкта в кадрі, пікселі;

$focal_{length}$ – відстань від центру проєкції до сенсора, в пікселях.

З пропорції 2.2 отримуємо, відстань від центру проєкції до сенсора (фокусна відстань, focal_length), що будемо використовувати для розрахунку відстані:

$$focal_{length} = \frac{perceived_{width} * known_{distance}}{real_{width}} \tag{2.3}$$

Відстань буде дорівнювати:

$$distance = \frac{real_{width} * focal_{length}}{percieved_{width}} \tag{2.3}$$

де: $real_{width}$ – реальна ширина об’єкта, в метрах;

$distance$ – відстань до об’єкта, в метрах;

$percieved_{width}$ – ширина об’єкта в кадрі, пікселі;

$focal_{length}$ – відстань від центру проєкції до сенсора, в пікселях.

В код на кроці, коли вже отримали фрейм, та визначили на ньому об’єкт, додаємо розрахунок відстані, та перевіряємо (рис. 21).



Рисунок 21 – Розрахунок відстані до об’єкту

Відстань від центру фокусу та градус повороту камери. Оскільки після калібрування ми розуміємо пропорції зображення, то можемо розрахувати конвертер пікселів у метри:

$$pixel_{to_{meter_{ratio}}} = \frac{real_{width}}{percieved_{width}} \quad (2.5)$$

де: $pixel_{to_{meter_{ratio}}}$ – скільки метрів в одному пікселі;

$real_{width}$ – реальний розмір об’єкту в метрах;

$percieved_{width}$ – розмір об’єкту в пікселях у кадрі;

Таким чином, з формули 2.5, pixel_to_meter_ratio — це скільки метрів припадає на один піксель при поточній дистанції.

Наступним кроком визначаємо наскільки об’єкт зсунений від центру кадру в метрах по горизонталі (X) або вертикалі (Y) та за допомогою тригонометричної функції отримуємо кут між напрямком на об’єкт і напрямком «прямо», щоб знати, на скільки треба повернути камеру, щоб об’єкт був у центрі фокусу:

$$\theta = \frac{180}{\pi} * \left(\arctan \frac{\Delta x}{distance}\right) \tag{2.6}$$

$$\theta = \frac{180}{\pi} * \left(\arctan \frac{\Delta y}{distance}\right) \tag{2.7}$$

де: θ – кут відхилення по вертикалі (2.6) чи горизонталі (2.7);

Δx – горизонтальне зміщення об’єкту в метрах після конвертації;

Δy – вертикальне зміщення об’єкту в метрах після конвертації;

$\arctan$ – арктангенс;

$\frac{180}{\pi}$ – коефіцієнт переведення радіанів в градуси;

Формули 2.6 та 2.7 вказують на який кут в градусах треба повернути камеру по горизонталі або вертикалі відповідно.



Рисунок 22 – Розрахунок кута для оберту камери, щоб об’єкт був у центрі

2.5 Висновок до розділу 2

В цьому розділі було описано та крок за кроком створено додаток для виявлення об’єкту з камери в режимі реального часу:

– Продемонстровано повний цикл створення власного датасету.
– Виконано навчання моделі.
– Продемонстровано один з методів калібрування камери.
– Проведено аналіз кожного кадру для досягнення результату.
– Застосовані розрахунки відстані та позиціювання у просторі відносно камери на базі лише зображення.

Додаток дозволяє використовувати і інші розрахунки до аналізу кадрів, наприклад, визначення чи наближається об’єкт.

3 ТЕСТУВАННЯ

3.1. Експериментальні дослідження та спостереження

Точність визначення відстані. Через обмеження простору проводився контрольний замір відстані для отримання похибки за показниками, представленими в таблиця 3.1.

Реальна відстань	Похибка
1 м	~ 0
2 м	~ 0,15 м
3 м	~ 0,3 м

Таблиця 3.1 – Похибка визначення відстані

Згідно таблиці 3.1 робимо висновок, що алгоритм розрахунку відстані потребує удосконалення з урахуванням перспективи яку сприймає фокус камери. Також слід протестувати камери з більшими кутами огляду, та порівняти показники похибки на різних сенсорах.

Показник частоти кадрів в залежності від роздільної здатності (при номінальній 30 FPS) представлені в таблиця 3.2.

Роздільна здатність	Частота FPS
1280x1080	11,5 – 13,0
1024x768	25 – 26

Таблиця 3.2 – Залежність FPS від роздільної здатності

Таблиця 3.2. показує відчутний приріст продуктивності зі зниженням частоти кадрів. Це зрозуміло – нейронна мережа обробляє меншу кількість пікселів, але в той самий час втрачає точність на більшій відстані об’єкту від камери. Збільшення роздільної здатності вимагає суттєво більшою обчислювальної потужності, принаймні з тим алгоритмом що використовується в цій роботі. Також слід враховувати що програма на Python працює повільніше за C++, отже наступним кроком слід використати C++ як платформу для написання програми.

Також при спостереженні було помічено неправильні розрахунки відстані до об’єкту. Алгоритм враховує ширину, яка може змінюватися не від наближення чи віддалення об’єкту, а під час коли він «ховається» за іншими об’єктами, що не відстежуються. Коли нейронна мережа бачить фрагмент, миттєво додає рамку, яка збільшується під час появи об’єкту спостереження. Це теж свідчить що слід враховувати особливі випадки на кадрі.

3.2. Подальші кроки

Для визначення наступних кроків що до розвитку додатку слід в першу чергу отримати зворотній зв’язок від потенційної цільової аудиторії. Визначення критичних для них моментів визначає напрямок, що слід удосконалювати в першу чергу, на що звернути увагу. Вирішення технічних проблем, що виникли у процесі розробки, покращення алгоритму, використання більш потужного «заліза» – наступна мета залежить від потенційних користувачів системи.

Ця робота спиралася на схему організації аналітичного блоку обробки зображень для подальшого використання (рис. 23).



Рисунок 23 – Схема блоку аналізу кадрів для виявлення об’єкту.

Отже є розуміння, що розроблена програма не може існувати сама в собі, та потрібно оброблені дані віддавати до наступного процесу для подальшого прийняття рішення.

4 ВИСНОВКИ

Ця робота охоплює теоретичну та практичну частину вивчення такого методу штучного інтелекту як машинний зір. У ході виконання було проведено теоретичний аналіз сучасних нейронних мереж для виявлення об’єктів., сучасних архітектур: R-CNN, Faster R-CNN, SSD та YOLO. Під час дослідження було розглянуто ключові характеристики, відмінності та сфери їх застосування.

Практична частина охоплює весь процес створення додатку для відстеження об’єкту та визначення його позиціювання у просторі в режимі реального часу. Розробка охопила всі етапи – від збору даних для навчання (датасету), до програмної реалізації аналізу кадрів. Зроблено огляд необхідних інструментів.

Під час розробки особлива увага була приділена аналізу навчання моделі, зроблені висновки на базі необхідних метрик (Precision, Recall, mAP50, mAP50-95). Також практична частина була зосереджена на аналізі фреймів нейронною мережею, практичне використання отриманих показників для подальшого аналізу та необхідних розрахунків – координати відносно камери спостереження.

Тестові дослідження та їх аналіз після розробки показали що виявлення об’єктів та визначення їх положення повністю реальна задача навіть за умовами «слабких ресурсів». Робота над недоліками є підґрунтям для подальшої роботи у цій сфері.

Враховуючи все сказане, приходимо висновку, що всі поставлені задачі досягнуто в повному обсязі. Зібрано датасет, навчена модель, створена програма для виявлення об’єкту, розрахунки при аналізі кадрів дають інформацію стосовно положення об’єкту.

ПЕРЕЛІК ПОСИЛАНЬ

1. Elgendy M. Deep Learning for Vision Systems. Manning Publications Co., 2020. 480 p. ISBN 9781617296192.
2. Ramalho L. Fluent Python : Clear, Concise, and Effective Programming. 2nd ed. O'Reilly Media, 2022. ISBN 9781492056355.
3. Schmitt S. G. Practical OpenCV with Python by Examples : Learn Tools and Techniques from Hands-On Examples in Python. 2024.
4. Guru D. S., Manjunath S. Real-Time Object Detection : Algorithms and Applications. CRC Press, 2022. 268 p. ISBN 9781032101342.
5. Szeliski R. Computer Vision : Algorithms and Applications. 2nd ed. Springer, 2021. 979 p. ISBN 9783030343712.
6. Ayyadevara V. K., Reddy Y. Modern Computer Vision with PyTorch. Packt Publishing, 2023. 546 p. ISBN 9781803231315.
7. Brownlee J. Deep Learning for Computer Vision. Machine Learning Mastery, 2022. 572 p. ISBN 9781098119170.
8. Reimer F. Hands-On Vision and Behavior for Self-Driving Cars. O'Reilly Media, 2022. 364 p. ISBN 9781098109607.
9. Aggarwal C. C. Neural Networks and Deep Learning: A Textbook. 2nd ed. Springer, 2021. 520 p. ISBN 9783030807047.
10. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2021. 800 p. ISBN 9780262046824.
11. O'Shea K., Nash R. Convolutional Neural Networks for Image Processing. Springer, 2022. 340 p. ISBN 9783030958732.
12. Офіційна документація Ultralytics YOLO. URL : https://docs.ultralytics.com/ (дата звернення : 01.04.2025).
13. Офіційний веб-сайт OpenCV. URL : https://opencv.org/ (дата звернення : 01.04.2025).

14. Офіційний веб-сайт Python. URL: https://www.python.org/ (дата звернення 08.09.2024)

15. Платформа для анотації зображень MakeSense.ai. URL: https://www.makesense.ai/ (дата звернення 01.04.2025)

16. Проєкт ImageNet: база даних для навчання систем комп’ютерного зору. URL: https://www.image-net.org/

17. Офіційна документація OpenCV Python Tutorials. URL: https://docs.opencv.org/ (дата звернення 02.09.2025)

18. Офіційний репозиторій YOLOv8 на GitHub. URL: https://github.com/ultralytics/ultralytics (дата звернення 02.09.2025)

19. TensorFlow Object Detection API Documentation. URL: https://tensorflow-object-detection-api-tutorial.readthedocs.io/ (дата звернення 15.06.2025)

20. NVIDIA CUDA Toolkit Documentation. URL: https://docs.nvidia.com/cuda/ (дата звернення 15.06.2025)

21. Форум Reddit r/MachineLearning: дискусії, огляди моделей та сучасні дослідження у сфері ШІ. URL: https://www.reddit.com/r/MachineLearning/ (дата звернення 15.06.2025)

22. Форум NVIDIA Developer: обговорення GPU, CUDA, нейронних мереж, глибинного навчання та комп’ютерного зору. URL: https://forums.developer.nvidia.com/ (дата звернення 15.06.2025)

ДОДАТОК А

Отримання зображень з відеофайлів

```
import cv2
import os
from pathlib import Path
def extract_frames_from_videos(input_dir, output_dir, total_frames, prefix="frame"):
    # Створюємо вихідну папку, якщо її немає
    Path(output_dir).mkdir(parents=True, exist_ok=True)
    # Підтримувані формати відео
    video_extensions = {‘.mp4’, ‘.avi’, ‘.mov’, ‘.mkv’, ‘.flv’, ‘.wmv’}
    # Знаходимо всі відео файли
    video_files = []
    for file in os.listdir(input_dir):
        if Path(file).suffix.lower() in video_extensions:
            video_files.append(os.path.join(input_dir, file))
    if not video_files:
        print(f"Не знайдено відео файлів у папці {input_dir}")
        return
    print(f"Знайдено {len(video_files)} відео файлів")
    # Отримуємо тривалість кожного відео
    video_info = []
    total_duration = 0

    for video_path in video_files:
        cap = cv2.VideoCapture(video_path)
        if not cap.isOpened():
```

```
            print(f"Не вдалося відкрити відео: {video_path}")
            continue

        fps = cap.get(cv2.CAP_PROP_FPS)
        frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
        duration = frame_count / fps if fps > 0 else 0

        video_info.append({
            'path': video_path,
            'name': Path(video_path).stem,
            'fps': fps,
            'frame_count': frame_count,
            'duration': duration
        })

        total_duration += duration
        cap.release()

        print(f"  {Path(video_path).name}: {duration:.2f}s, {frame_count} кадрів, {fps:.2f} fps")

    if total_duration == 0:
        print("Помилка: загальна тривалість відео дорівнює 0")
        return

    # Розраховуємо кількість кадрів для кожного відео пропорційно
    frames_extracted = 0
    global_frame_number = 1  # Глобальний лічильник для всіх кадрів

    for i, info in enumerate(video_info):
        # Розраховуємо пропорційну кількість кадрів
```

```
            if i == len(video_info) – 1:
                # Для останнього відео беремо залишок, щоб точно отримати потрібну кількість
                frames_to_extract = total_frames – frames_extracted
            else:
                proportion = info[‘duration’] / total_duration
                frames_to_extract = int(total_frames * proportion)

            if frames_to_extract <= 0:
                continue

            print(f"\nВитягуємо {frames_to_extract} кадрів з {Path(info[‘path’]).name}")

            # Відкриваємо відео
            cap = cv2.VideoCapture(info[‘path’])
            # Розраховуємо інтервал між кадрами
            if frames_to_extract >= info[‘frame_count’]:
                # Якщо потрібно більше кадрів, ніж є у відео, беремо всі
                frame_indices = list(range(info[‘frame_count’]))
            else:
                # Рівномірно розподіляємо кадри по всій тривалості відео
                step = info[‘frame_count’] / frames_to_extract
                frame_indices = [int(j * step) for j in range(frames_to_extract)]

            # Витягуємо кадри
            for j, frame_idx in enumerate(frame_indices):
                cap.set(cv2.CAP_PROP_POS_FRAMES, frame_idx)
                ret, frame = cap.read()

                if ret:
```

```
                # Формуємо ім’я файлу: prefix_номер.jpg
                output_filename = f"{prefix}_{global_frame_number}.jpg"
                output_path = os.path.join(output_dir, output_filename)
                cv2.imwrite(output_path, frame)
                frames_extracted += 1
                global_frame_number += 1
            else:
                print(f"  Помилка читання кадру {frame_idx}")

        cap.release()
        print(f"  Витягнуто {len(frame_indices)} кадрів")

    print(f"\nГотово! Всього витягнуто {frames_extracted} кадрів у папку {output_dir}")
if __name__ == "__main__":
    # Приклад використання
    # Обробка позитивних прикладів
    extract_frames_from_videos(
        input_dir="input/positive",
        output_dir="output/positive",
        total_frames=700,
        prefix="positive"
    )
    # Обробка негативних прикладів
    extract_frames_from_videos(
        input_dir="input/negative",
        output_dir="output/negative",
        total_frames=700,
        prefix="negative"
    )
```

ДОДАТОК Б

Заповнення датасету випадковим набором відповідних файлів

```
import os
import shutil
import random
from pathlib import Path

def copy_files_to_dataset(images_source, annotations_source, dataset_root,
                          train_count, val_count, prefix="positive"):
    # Створюємо необхідні папки
    for split in ['train', 'val', 'test']:
        Path(os.path.join(dataset_root, 'images', split)).mkdir(parents=True, exist_ok=True)
        Path(os.path.join(dataset_root, 'labels', split)).mkdir(parents=True, exist_ok=True)

    # Отримуємо всі файли зображень
    image_files = []
    for file in os.listdir(images_source):
        if file.startswith(prefix) and file.endswith(('.jpg', '.jpeg', '.png')):
            image_files.append(file)

    if not image_files:
        print(f"Не знайдено зображень з префіксом '{prefix}' у папці {images_source}")
        return
```

```
        # Перемішуємо файли у випадковому порядку
        random.shuffle(image_files)
        total_files = len(image_files)
        test_count = total_files – train_count – val_count

        print(f"\n{prefix.upper()} набір:")
        print(f"  Всього файлів: {total_files}")
        print(f"  Train: {train_count}")
        print(f"  Val: {val_count}")
        print(f"  Test: {test_count}")

        # Розподіляємо файли
        splits = {
            'train': image_files[:train_count],
            'val': image_files[train_count:train_count + val_count],
            'test': image_files[train_count + val_count:]
        }

        # Копіюємо файли
        for split_name, files in splits.items():
            print(f"\nКопіюємо {len(files)} файлів у {split_name}...")

            for image_file in files:
                # Копіюємо зображення
                src_image = os.path.join(images_source, image_file)
                dst_image = os.path.join(dataset_root, 'images', split_name,
image_file)
                shutil.copy2(src_image, dst_image)
```

```
            # Копіюємо анотацію
            annotation_file = Path(image_file).stem + '.txt'
            src_annotation = os.path.join(annotations_source, annotation_file)
            dst_annotation = os.path.join(dataset_root, 'labels', split_name, annotation_file)

            if os.path.exists(src_annotation):
                shutil.copy2(src_annotation, dst_annotation)
            else:
                print(f" Попередження: не знайдено анотацію {annotation_file}")

        print(f" Скопійовано {len(files)} зображень та анотацій у {split_name}")

    def fill_dataset(dataset_root="dataset", seed=42):
        # Встановлюємо seed для відтворюваності
        random.seed(seed)

        print("=" * 60)
        print("ЗАПОВНЕННЯ ДАТАСЕТУ")
        print("=" * 60)
        # Копіюємо позитивні приклади
        copy_files_to_dataset(
            images_source="output/positive",
            annotations_source="output/p_annotattions",
            dataset_root=dataset_root,
            train_count=298,
            val_count=85,
            prefix="positive"
```

```
    )
    # Копіюємо негативні приклади
    copy_files_to_dataset(
        images_source="output/negative",
        annotations_source="output/n_annotations",
        dataset_root=dataset_root,
        train_count=490,
        val_count=140,
        prefix="negative"
    )
    print("\n" + "=" * 60)
    print("ГОТОВО!")
    print("=" * 60)

    # Виводимо підсумкову статистику
    print("\nПідсумкова статистика:")
    for split in ['train', 'val', 'test']:
        images_path = os.path.join(dataset_root, 'images', split)
        labels_path = os.path.join(dataset_root, 'labels', split)
        if os.path.exists(images_path):
            images_count = len([f for f in os.listdir(images_path) if f.endswith(('.jpg', '.jpeg', '.png'))])
            labels_count = len([f for f in os.listdir(labels_path) if f.endswith('.txt')])
            print(f"  {split.upper()}: {images_count} зображень, {labels_count} анотацій")
if __name__ == "__main__":
    # Заповнюємо датасет
    fill_dataset(dataset_root="dataset", seed=42)
```

ДОДАТОК В

Скрипт для навчання моделі

```
from ultralytics import YOLO
import os

def train_yolo_model(
    data_yaml="data.yaml",
    model_size="n",  # n, s, m, l, x (nano, small, medium, large, xlarge)
    epochs=50,
    imgsz=640,
    batch=16,
    device=None,  # None для auto, 0 для GPU, 'cpu' для CPU
    project="runs/detect",
    name="yolov8_training",
    patience=50,
    save=True,
    verbose=True
):
    """
    Тренує модель YOLOv8 на базі датасету, описаного в data.yaml.

    Args:
        data_yaml (str): Шлях до файлу конфігурації датасету
        model_size (str): Розмір моделі – 'n' (nano), 's' (small), 'm' (medium),
                          'l' (large), 'x' (xlarge)
        epochs (int): Кількість епох для тренування
```

```
        imgsz (int): Розмір зображення для тренування
        batch (int): Розмір батчу
        device (str|int): Пристрій для тренування (None – auto, 0 – GPU, 'cpu' – CPU)
        project (str): Папка для збереження результатів
        name (str): Назва експерименту
        patience (int): Кількість епох без покращення перед зупинкою
        save (bool): Зберігати чекпоінти
        verbose (bool): Виводити детальну інформацію
    """
    # Перевіряємо, чи існує файл конфігурації
    if not os.path.exists(data_yaml):
        raise FileNotFoundError(f"Файл конфігурації {data_yaml} не знайдено!")

    print("=" * 60)
    print("ТРЕНУВАННЯ МОДЕЛІ YOLOv8")
    print("=" * 60)
    print(f"Файл конфігурації: {data_yaml}")
    print(f"Модель: YOLOv8{model_size}")
    print(f"Епохи: {epochs}")
    print(f"Розмір зображення: {imgsz}")
    print(f"Розмір батчу: {batch}")
    print(f"Пристрій: {device if device is not None else 'auto'}")
    print("=" * 60)
    print()

    # Завантажуємо модель YOLOv8
    model_name = f"yolov8{model_size}.pt"
    print(f"Завантажуємо модель: {model_name}")
```

```
    model = YOLO(model_name)

    # Починаємо тренування
    print(f"\nПочинаємо тренування...")
    print("-" * 60)

    results = model.train(
        data=data_yaml,
        epochs=epochs,
        imgsz=imgsz,
        batch=batch,
        device=device,
        project=project,
        name=name,
        patience=patience,
        save=save,
        verbose=verbose
    )

    print("-" * 60)
    print("Тренування завершено!")
    print(f"Результати збережено у: {project}/{name}")
    print("=" * 60)
    return model, results

if __name__ == "__main__":
    # Налаштування тренування
    # Змініть параметри за потреби

    train_yolo_model(
```

data_yaml="data.yaml",

model_size="s", *# Використайте ‘n’ для швидкого тренування, ‘s’/’m’/’l’/’x’ для кращої точності*

epochs=50, *# Кількість епох*

imgsz=1024, *# Розмір зображення*

batch=16, *# Зменшіть, якщо немає пам’яті (4, 8, 16, 32)*

device=None, *# None – auto, 0 – GPU, ‘cpu’ – CPU*

project="runs/detect",

name="yolov8_training",

patience=30 *# Early stopping – зупинка, якщо немає покращення протягом 50 епох*

)

ДОДАТОК Г

Детектор

```
import cv2
import math
import time
from ultralytics import YOLO
# ========== НАЛАШТУВАННЯ ==========
MODEL_PATH = "runs/detect/yolov8_training/weights/best.pt"
# CAMERA_ID = 0  # Тимчасово закоментовано – використовуємо відео
CONFIDENCE_THRESHOLD = 0.25  # Поріг впевненості для детекції
VIDEO_PATH = "input/positive/20251021_140518.mp4"  # Шлях до відеофайлу
START_TIME_MINUTES = 5.45  # З якої хвилини починати відтворення відео

# Параметри для вимірювання відстані
KNOWN_DISTANCE = 0.2  # Відома відстань в метрах (для калібрування)
REAL_WIDTH = 0.35     # Реальна ширина об'єкта в метрах
TARGET_FRAME_WIDTH = 1280  # Бажана ширина кадру
TARGET_FRAME_HEIGHT = 1024  # Бажана висота кадру

# ========== ЗАВАНТАЖЕННЯ МОДЕЛІ ==========
try:
    model = YOLO(MODEL_PATH)
    print("Модель завантажена успішно!")
```

```
except Exception as e:
    print(f"Помилка завантаження моделі: {e}")
    exit(1)

# ========== ІНІЦІАЛІЗАЦІЯ ДЖЕРЕЛА ==========
# ВАРІАНТ 1: Робота з відеофайлом (поточний режим)
print(f"Відкриваємо відеофайл: {VIDEO_PATH}...")
cap = cv2.VideoCapture(VIDEO_PATH)

# ВАРІАНТ 2: Робота з камерою
# print(f"Підключаємося до камери {CAMERA_ID}...")
# cap = cv2.VideoCapture(CAMERA_ID)
# # Встановлюємо роздільну здатність
# cap.set(cv2.CAP_PROP_FRAME_WIDTH, TARGET_FRAME_WIDTH)
# cap.set(cv2.CAP_PROP_FRAME_HEIGHT, TARGET_FRAME_HEIGHT)

if not cap.isOpened():
    print(f"Помилка: не вдалося відкрити відеофайл {VIDEO_PATH}")
    print("Перевірте, чи існує файл і чи правильний шлях вказано")
    exit(1)

# Отримуємо реальну роздільну здатність
actual_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
actual_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS)) or 30
total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
total_duration_sec = total_frames / fps if fps > 0 else 0
print(f"Відео завантажено: {actual_width}x{actual_height} @ {fps} FPS")
print(f"Тривалість відео: {total_duration_sec:.1f} секунд ({total_duration_sec/60:.1f} хвилин)")
```

```
    # Встановлюємо стартову позицію відео
    start_time_ms = int(START_TIME_MINUTES * 60 * 1000)  # Переводимо в мілісекунди

    if total_duration_sec > START_TIME_MINUTES * 60:
      cap.set(cv2.CAP_PROP_POS_MSEC, start_time_ms)
      print(f"Відео встановлено на позицію: {START_TIME_MINUTES} хвилин ({start_time_ms/1000:.1f} секунд)")
    else:
      print(f"Попередження: відео коротше за {START_TIME_MINUTES} хвилин, починаємо з початку")
    # Обчислення фокусної відстані на основі реальної ширини кадру
    focal_length = (actual_width * KNOWN_DISTANCE) / REAL_WIDTH
    print("\nДетекція запущена! Натисніть 'q' або ESC для виходу")
    # Змінні для відстеження руху
    previous_distance = None
    MOVEMENT_THRESHOLD = 0.01  # Поріг для визначення руху (в метрах) – зменшено для кращої чутливості
    MOVEMENT_HISTORY = []  # Історія для фільтрації
    HISTORY_SIZE = 3  # Розмір історії для усереднення – зменшено для швидшого реагування
    # ========== ОСНОВНИЙ ЦИКЛ ==========
    try:
      while True:
        ret, frame = cap.read()
        if not ret:
          print("Помилка читання кадру")
          break
        frame_height, frame_width = frame.shape[:2]
```

```
cx_frame = frame_width // 2  # Центр кадру по X
cy_frame = frame_height // 2  # Центр кадру по Y

# Детекція об'єктів
results = model.predict(
    frame,
    conf=CONFIDENCE_THRESHOLD,
    imgsz=1024,
    verbose=False
)
# Обробка результатів
if results[0].boxes is not None and len(results[0].boxes) > 0:
    # Беремо об'єкт з найвищою впевненістю
    box = results[0].boxes[0]
    confidence = float(box.conf[0])
    # Координати bbox
    x1, y1, x2, y2 = map(int, box.xyxy[0])

    # Обчислення відстані
    perceived_width = x2 – x1
    if perceived_width > 0:
        distance = (REAL_WIDTH * focal_length) / perceived_width
    else:
        distance = 0

    # Визначення напрямку руху (наближається/віддаляється)
    movement_status = "Static"
    movement_color = (128, 128, 128)  # Сірий
    speed_change = 0.0  # Швидкість зміни відстані (м/кадр)
    if previous_distance is not None and distance > 0:
```

```
                # Обчислюємо зміну відстані (негативна = наближається, позитивна = віддаляється)
                distance_change = distance – previous_distance
                # Додаємо до історії для фільтрації
                MOVEMENT_HISTORY.append(distance_change)
                if len(MOVEMENT_HISTORY) > HISTORY_SIZE:
                    MOVEMENT_HISTORY.pop(0)
                # Обчислюємо усереднену зміну для стабільності
                if len(MOVEMENT_HISTORY) > 0:
                    avg_difference = sum(MOVEMENT_HISTORY) / len(MOVEMENT_HISTORY)
                    speed_change = avg_difference
                    # Перевіряємо, чи середня зміна перевищує поріг
                    if abs(avg_difference) > MOVEMENT_THRESHOLD:
                        if avg_difference < 0:
                            # Відстань зменшується – наближається (ТРИВОГА)
                            movement_status = "Approaching"
                            movement_color = (0, 0, 255)  # Червоний
                        else:
                            # Відстань збільшується – віддаляється (безпечно)
                            movement_status = "Moving Away"
                            movement_color = (0, 255, 0)  # Зелений

                    # Зберігаємо поточну відстань для наступного кадру
                    previous_distance = distance
                elif distance > 0:
                    # Перша детекція
                    previous_distance = distance
                    MOVEMENT_HISTORY = []
                else:
```

```
            # Немає валідної відстані
            previous_distance = None
            MOVEMENT_HISTORY = []
        # Центр об'єкта
        cx_box = (x1 + x2) // 2
        cy_box = (y1 + y2) // 2

        # Зміщення від центру кадру
        dx = cx_box – cx_frame
        dy = cy_frame – cy_box  # + вгору, – вниз
        # Розрахунок кутів повороту камери
        # Використовуємо arctg(зміщення_в_метрах / відстань_до_об'єкта)
        # Конвертуємо пікселі у метри на основі відстані
        pixel_to_meter_ratio = distance / focal_length
        # Зміщення в метрах
        dx_meters = dx * pixel_to_meter_ratio
        dy_meters = dy * pixel_to_meter_ratio

        # Кути повороту в градусах (одразу через atan2 та degrees)
        angle_horizontal = math.degrees(math.atan2(dx_meters, distance))
        angle_vertical = math.degrees(math.atan2(dy_meters, distance))

        # Малюємо bbox
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

        # # Малюємо центр об'єкта
        cv2.circle(frame, (cx_box, cy_box), 5, (0, 0, 255), -1)

        # # Малюємо центр кадру
```

```
            cv2.circle(frame, (cx_frame, cy_frame), 5, (255, 0, 0), -1)
            cv2.line(frame, (cx_frame, cy_frame), (cx_box, cy_box), (255, 0, 0),
1)

            # Текст з інформацією
            distance_label = f"Distance: {distance:.2f} m"
            conf_label = f"Confidence: {confidence:.2f}"
            movement_label = f"Status: {movement_status}"
            angle_h_label = f"Angle H: {angle_horizontal:+.2f} deg"
            angle_v_label = f"Angle V: {angle_vertical:+.2f} deg"

            # Відображаємо інформацію
            y_offset = 30
            cv2.putText(frame, distance_label, (10, y_offset),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
            y_offset += 30
            cv2.putText(frame, movement_label, (10, y_offset),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, movement_color, 2)
            y_offset += 30

            # Показуємо кути повороту
            cv2.putText(frame, angle_h_label, (10, y_offset),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 200, 0), 2)
            y_offset += 30
            cv2.putText(frame, angle_v_label, (10, y_offset),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 200, 0), 2)
            y_offset += 30

            # Показуємо швидкість зміни відстані, якщо є рух
```

```
            if previous_distance is not None and abs(speed_change) > MOVEMENT_THRESHOLD:
                speed_label = f"Speed: {abs(speed_change)*1000:.1f} mm/frame"
                cv2.putText(frame, speed_label, (10, y_offset),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.6, movement_color, 2)
                y_offset += 30

            cv2.putText(frame, conf_label, (10, y_offset),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 255), 2)
        else:
            # Немає детекцій – скидаємо попередню відстань
            previous_distance = None
            MOVEMENT_HISTORY = []

            # Відображаємо повідомлення
            cv2.putText(frame, "No detections", (10, 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

        # Відображаємо FPS у правому верхньому куті
        # Реальний FPS обробки
        fps_label = f"FPS: {current_fps:.1f}"
        (text_width, text_height), _ = cv2.getTextSize(fps_label, cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2)
        fps_x = frame_width – text_width – 10
        fps_y = 30
        cv2.putText(frame, fps_label, (fps_x, fps_y),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

        # Відображаємо кадр
```

```
        cv2.imshow("YOLO Detection – Press Q or ESC to quit", frame)

        # Перевірка натискання клавіш
        key = cv2.waitKey(1) & 0xFF
        if key == ord('q') or key == 27:  # 'q' або ESC
            print("\nВихід з програми...")
            break

except KeyboardInterrupt:
    print("\nПрограму перервано користувачем")

finally:
    # Звільняємо ресурси
    cap.release()
    cv2.destroyAllWindows()
    print("Ресурси звільнено. Програма завершена.")
```