Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій
(повна назва)

Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

перший (бакалаврський)
(рівень вищої освіти)

Розроблення системи автоматизації розпізнавання об'єктів у виробничому приміщенні

(тема)

Виконав:
здобувач 4 року навчання,
групи АКТСІ-21-3
Віталій Бас
(власне ім'я, прізвище)

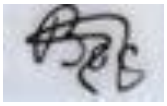
Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Системна інженерія
(повна назва освітньої програми)

Керівник професор Цимбал О.М.
(посада, власне ім'я, прізвище)

Допускається до захисту
Зав. кафедри КІТАР _____ Ігор НЕВЛЮДОВ
(підпис) (власне ім'я, прізвище)

2025 р.

Я, Бас Віталій Олегович, як здобувач вищої освіти ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Я не використовував штучний інтелект для підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на віповідне джерело.

"04" червень 2025 р.  Віталій БАС

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет АКТ
Кафедра КІТАР
Рівень вищої освіти перший (бакалаврський)
Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
(код і повна назва)
Тип програми Освітньо-професійна
Освітня програма Системна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри КІТАР _____
(підпис)
« 19 » травня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Бас Віталію Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення системи автоматизації розпізнавання об'єктів у виробничому приміщенні

Затверджена наказом по університету від 19.05.2025 р. № 391 Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 15.06.2025 р.
3. Вихідні дані до роботи
3.1 Методи глибокого навчання
3.2 Модель виявлення об'єктів YOLOv8
3.3 Набори даних зображень для навчання (публічні та влсноруч зібрані)

4. Перелік питань, що потрібно опрацювати в роботі
Аналіз предметної області та вибір методології розробки
Підготовка даних та навчання моделі розпізнавання YOLOv8
Реалізація програмного забезпечення системи
Тестування функціональності та продуктивності розробленої системи

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)

Демонстраційний матеріал, представлений у форматі презентації PowerPoint (*.ppt) 15 с. формату А4

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Приміт-ка
1	Аналіз предметної області та огляд існуючих рішень для розпізнавання об'єктів	28.04-04.05.2025	виконано
2	Вибір та обґрунтування архітектури моделі (YOLOv8) та необхідних програмних засобів розробки.	05.05-10.05.2025	виконано
3	Розробка та підготовка наборів даних	11.05-17.05.2025	виконано
4	Навчання, доналаштування та оцінка моделі розпізнавання об'єктів	18.05-25.05.2025	виконано
5	Розробка та тестування програмного модуля системи автоматизації	26.05-31.05.2025	виконано
6	Оформлення пояснювальної записки	01.06-04.06.2025	виконано
7	Подання роботи на перевірку Інтернет-сервісом StrikePlagiarism	05.06.2024	виконано
8	Подання роботи на рецензію	06.06.2024	виконано
9	Подання роботи на підпис зав. кафедри	14.06.2025	виконано
11	Подання кваліфікаційної роботи в ЕК	15.06.2025	виконано

Дата видачі завдання 28.04.2025 р.

Здобувач  Віталій БАС

(підпис) (власне ім'я, прізвище)

Керівник роботи професор Цимбал О. М.

(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 100 с., 1 табл., 40 рис., 3 дод., 17 джерел.

РОЗПІЗНАВАННЯ ОБ’ЄКТІВ, ГЛИБОКЕ НАВЧАННЯ, YOLOv8, КОМП'ЮТЕРНИЙ ЗІР, АВТОМАТИЗОВАНА СИСТЕМА, МОНІТОРИНГ БЕЗПЕКИ.

Об’єкт розробки – процес автоматичного виявлення об’єктів у кадрі засобами комп’ютерного зору.

Предмет розробки – методи та засоби розпізнавання об’єктів на базі YOLOv8 для контролю дотримання правил безпеки у виробничих приміщеннях.

Мета роботи – створення системи, що дозволяє виявляти відсутність засобів індивідуального захисту у працівників у реальному часі.

У першому розділі проаналізовано основні поняття та технології глибокого навчання, комп’ютерного зору та згорткових нейронних мереж, а також методи виявлення об’єктів.

У другому – розглянуто алгоритми виявлення об’єктів на основі глибокого навчання, порівняно одно- та двоступеневі детектори, обґрунтовано вибір моделі YOLOv8 і необхідних бібліотек.

У третьому – описано етапи розробки: постановку задачі, проєктування, підготовку даних, навчання YOLOv8, створення графічного інтерфейсу та тестування.

Розроблена система виконує розпізнавання об’єктів у реальному часі, фіксує порушення й зберігає підтверджуючі дані.

ABSTRACT

Explanatory note: 100 p., 1 table, 40 figures, 3 appendices, 17 sources.

OBJECT RECOGNITION, DEEP LEARNING, YOLOv8, COMPUTER VISION, AUTOMATED SYSTEM, SAFETY MONITORING.

The object of development is the process of automatic detection of objects in the frame using computer vision.

The subject of development is methods and tools for object recognition based on YOLOv8 to monitor compliance with safety rules in production facilities.

The purpose of the work is to create a system that allows detecting the absence of personal protective equipment among employees in real time.

The first section analyzes the basic concepts and technologies of deep learning, computer vision and convolutional neural networks, as well as methods for object detection.

The second section examines object detection algorithms based on deep learning, compares one- and two-stage detectors, justifies the choice of the YOLOv8 model and the necessary libraries.

The third section describes the development stages: problem statement, design, data preparation, YOLOv8 training, creation of a graphical interface and testing.

The developed system performs object recognition in real time, records violations and stores supporting data.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ЗІЗ – засоби індивідуального захисту;

CV – Computer Vision (Комп'ютерний зір);

DL – Deep Learning (Глибоке навчання);

YOLO – You Only Look Once;

RGB – Red, Green, Blue;

OpenCV – open source computer vision library.

ВСТУП

На сьогодні автоматичне виявлення об’єктів є важливим компонентом багатьох технічних систем, зокрема у сфері промислової автоматизації. У ситуаціях, де потрібен постійний контроль за виконанням правил техніки безпеки, звичайних методів часто недостатньо. Саме в таких випадках системи на основі комп’ютерного зору можуть стати ефективним інструментом допомоги.

Сучасні моделі глибокого навчання активно впроваджуються у сферу машинного зору, і вже зараз їх використовують для вирішення практичних задач – від виявлення пішоходів на дорогах до автоматичного контролю процесів на підприємствах. Зокрема, у промисловості актуальними стають рішення, які здатні виявляти, чи дотримуються працівники правил безпеки. І якщо система може в реальному часі помітити, що хтось працює без шолома або іншого засобу захисту – це значно підвищує рівень загальної безпеки.

Використання нейромереж у таких завданнях дозволяє зменшити навантаження на працівників служб охорони праці та мінімізувати людський фактор. Завдяки сучасним алгоритмам стало можливим реалізувати систему, яка швидко аналізує зображення або відео і реагує на порушення автоматично.

Об’єктом розробки є процес автоматичного виявлення об’єктів у кадрі за допомогою комп’ютерного зору.

Предметом розробки виступає програмний модуль, який виконує розпізнавання порушень правил техніки безпеки на основі моделі YOLOv8.

Метою роботи є створення системи, яка дозволяє виявляти відсутність засобів індивідуального захисту у працівників у режимі реального часу.

Реалізація даної системи сприяє досягненню Цілей Сталого Розвитку, зокрема ЦСР 3 “Міцне здоров'я та благополуччя” через зменшення виробничого травматизму та підвищення рівня охорони праці, ЦСР 8 “Гідна праця та економічне зростання” шляхом забезпечення безпечних умов праці, а також ЦСР 9 “Промисловість, інновації та інфраструктура” через

впровадження інноваційних технологій у виробничі процеси.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати сучасні методи комп’ютерного зору та глибокого навчання, які застосовуються для виявлення об’єктів;

– порівняти архітектури алгоритмів виявлення об’єктів і обґрунтувати вибір YOLOv8;

– підібрати програмні засоби та інструменти, необхідні для реалізації системи;

– зібрати та розмітити набір зображень для навчання моделі;

– провести навчання моделі YOLOv8 з використанням власного датасету;

– розробити програмний модуль для обробки відеопотоку та фіксації порушень;

– протестувати систему в умовах, наближених до реального використання, та оцінити її ефективність.

Кваліфікаційна робота оформлена відповідно до вимог ДСТУ 3008:2015 [1] та методичних рекомендацій з знаписання кваліфікаційної роботи [2].

1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Глибоке навчання (Deep Learning)

Машинне навчання – це метод, який використовується передовими технологіями, щоб дозволити комп’ютерним системам вчитися на базі вхідних даних і розпізнавати шаблони для майбутніх даних. Машинне навчання дозволяє комп’ютерним системам робити прогнози, створювати категорії та виконувати інші обчислення за вхідними даними [3].

У машинному навчанні є три основні шляхи, якими можна піти: це контрольоване навчання, неконтрольоване та напівконтрольоване. Кожен із них відрізняється тим, як багато інформації людина передає моделі на старті. У першому випадку все просто – ми одразу даємо моделі дані разом із правильними відповідями. Вона вивчає приклади й потім може робити передбачення на нових, ще не бачених даних. Якщо ж міток немає, доводиться працювати методом неконтрольованого навчання, де модель сама намагається щось зрозуміти з даних. А напівконтрольоване – це щось середнє: частина прикладів має мітки, а частина – ні. Цей підхід дозволяє заощадити час на ручному маркуванні, не втрачаючи при цьому якість. До речі, глибокі нейромережі можуть працювати в усіх трьох режимах.

Окрім вже розглянутого машинного навчання, останніми роками активного розвитку набула його підмножина – глибоке навчання. Це напрям, який вважається більш потужним і здатним до складнішого аналізу завдяки принципам, що нагадують роботу людського мозку. Основу складають багаторівневі штучні нейронні мережі, де кожен рівень поступово підвищує абстракцію, трансформуючи вхідні дані за допомогою нелінійних операцій. Передача інформації відбувається через численні з’єднання між шарами, які мають власну вагу – саме ці значення і формують навчальну суть алгоритму. До кожного нейрона додається зміщення, що разом із вагою впливає на

результат обчислення. Після цього запускається активаційна функція, яка вирішує, чи буде сигнал передано далі. Якщо нейрон активується, його вихід прямує до наступного шару. Так триває до останнього етапу – вихідного шару, який і формує фінальний результат для подальшої обробки. Сфера глибокого навчання знайшла практичне застосування в задачах, пов’язаних із виявленням об’єктів, а також у голосовому розпізнаванні. У 2013 році особливу популярність здобув алгоритм R-CNN, який першим ефективно застосував глибоку нейромережу для виявлення об’єктів. Він суттєво перевершив попередні методи, показавши понад 30% переваги у Visual Object Classes Challenge 23. Це забезпечило йому провідне місце в галузі, значно піднявши рівень точності та продуктивності. При виявленні об’єктів особливу складність становлять завдання точного визначення розташування та одночасного присвоєння правильного класу кожному об’єкту. Моделі виявлення об’єктів на основі глибокого навчання з використанням згорткових нейронних мереж і трансформаторів зараз відіграють ключову роль в еволюції цієї області. Ці моделі можуть надати важливу інформацію для семантичного розуміння зображень і відео. Вони швидко впроваджуються в різних галузях. Приклади включають підтримку автономних автомобілів для безпечної навігації в русі, розпізнавання обличчя, аналіз поведінки людини і медичну візуалізацію, таку як виявлення раку, робототехніка, загальні методи обробки зображень, такі як кадрування, визначення орієнтації та підвищення контрастності, і багато інших випадків використання. Щодо майбутніх варіантів використання для виявлення об'єктів, то тут можливості безмежні.

1.2 Комп'ютерний зір (Computer Vision)

Комп’ютерне зір – це галузь штучного інтелекту, метою якої є штучна симуляція зорової системи людини шляхом вилучення характеристик реального світу, починаючи з зображень і відео, зазвичай зроблених камерою. Він включає кілька завдань з різними цілями: класифікація зображень,

виявлення об’єктів, відстеження об’єктів, семантична сегментація, сегментація екземплярів та багато іншого. Комп’ютерний зір допомагає комп’ютеру «бачити» поведінку людини, дозволяючи йому розуміти та правильно реагувати на вивчену поведінку [3].

Робота систем CV базується на трьох основних кроках: отримання, аналіз візуальних даних та реакція на них. Проте, імітувати людський зір у машинах є значною проблемою. Людський мозок, зокрема зорова кора, надає нам неймовірну здатність розрізняти кольори, форми та контури, а також безліч інших ознак, навіть за численних варіацій. Цей інструмент надзвичайно складно відтворити. При цьому, комп'ютер використовує інший підхід до візуальних даних. Він перетворює фізичний світ на масштабну сукупність числових значень, що позначають колір та інтенсивність пікселів. Так, цифрові зображення розглядаються ним як багатовимірні структури, утворені одним або кількома каналами відповідно до еталонної хроматичної шкали, тобто для кольору або відтінків сірого. Сучасні технології дозволяють системам комп'ютерного зору отримувати візуальні дані, спираючись на різноманітні джерела, такі як датчики, камери чи візуальні бази даних. Після цього, система може взаємодіяти з низкою механічних пристроїв – від моніторів до мобільних телефонів і навіть транспортних засобів, аби відповідним чином реагувати на отриману візуальну інформацію.

Попри певні складнощі, інтерес до комп'ютерного зору (CV) виник та активно розвивався з 1960-х років. Тоді Робертс, Марр та інші дослідники зробили фундаментальний внесок у еволюцію штучного зору. Їхні роботи стали основою. Згодом проводилися нові дослідження. Вони вивчали можливість об'єднання комп'ютерного зору (на початковому етапі) з іншими суміжними областями. Це включало обробку цифрових зображень, розпізнавання образів та комп'ютерну графіку. Справжній прорив, однак, відбувся завдяки штучним нейронним мережам. Їх поява та подальший успіх змінили галузь. Зараз вони є ефективним рішенням для багатьох проблем штучного інтелекту, особливо тих, що стосуються зору.

Цікаво, як ще в середині 20-го століття вчені вже думали над тим, як би змусити машину «бачити». Ну, не зовсім бачити, звісно – швидше реагувати на зображення. Спочатку це були спроби відтворити процес навчання, схожий на людський. Так з’явився персептрон – по суті, нейронка, але ще дуже проста.

Далі справа пішла трохи швидше: з’явились завдання з розпізнавання символів. Тут уже знадобилося щось нове – і так з’явились згорткові мережі. LeNet-5 – одна з перших таких. Вона, здається, навіть досі згадується у багатьох матеріалах, бо могла читати рукописні цифри.

Але далі все загальмувалося. Мало даних, повільне залізо. Про комп’ютерний зір ще довго не згадували. Аж поки не з’явились хороші відеокарти та великі набори зображень – тоді й повернулась увага до цієї теми.

Тепер усе інакше. Задачі складніші, моделі потужніші. І вже зовсім не дивно, що комп’ютер бачить майже краще за людину.

1.3 Згорткові нейронні мережі (Convolutional Neural Networks, CNN)

Згорткова нейронна мережа (CNN) вперше з'явилася близько 1998 року, але була доведена як ефективний інструмент для класифікації зображень у 2012 році на конкурсі Large Scale Visual Recognition Challenge. Згорткові нейронні мережі – це алгоритм глибокого навчання [4]. Приклад CNN можемо побачити на рисунку 1.1. CNN використовуються для розкладання зображення на матрицю пікселів. Кожній точці в матриці присвоюється мітка на основі значення RGB пікселя: 0 – чорний та 255 – білий. У CNN обробка зображень заснована на навчанні цих матриць і визначенні зв'язку між пікселем і його сусідами. CNN використовує виявлення країв для визначення країв і жорстких ліній, і завдяки цьому ідентифікує об'єкти, пов'язані з ними сутності на зображенні за допомогою таких процесів, як виявлення об'єктів і сегментація зображення [3]. Щоб хоч якось зрозуміти, що зображено на фото чи відео, комп’ютеру доводиться виконувати купу операцій. Там є і згортки, і якісь активації, і об’єднання – усе це працює разом у так званих згорткових

нейронних мережах. Вони переглядають зображення, «проходять» його через шари, пробують щось передбачити, а потім уточнюють – і так багато разів, поки результат не стане схожим на правду.

Згорткові мережі виявились дуже корисними – і не лише для простого розпізнавання. Їх застосовують для класифікації, сегментації, виявлення різних об'єктів. Навіть у текстах і мовленні вони вже використовуються. Якщо дуже спрощено, то така мережа має три основні частини: згорткові шари, об’єднання, і ті, де вже приймаються остаточні рішення – повністю зв’язані.

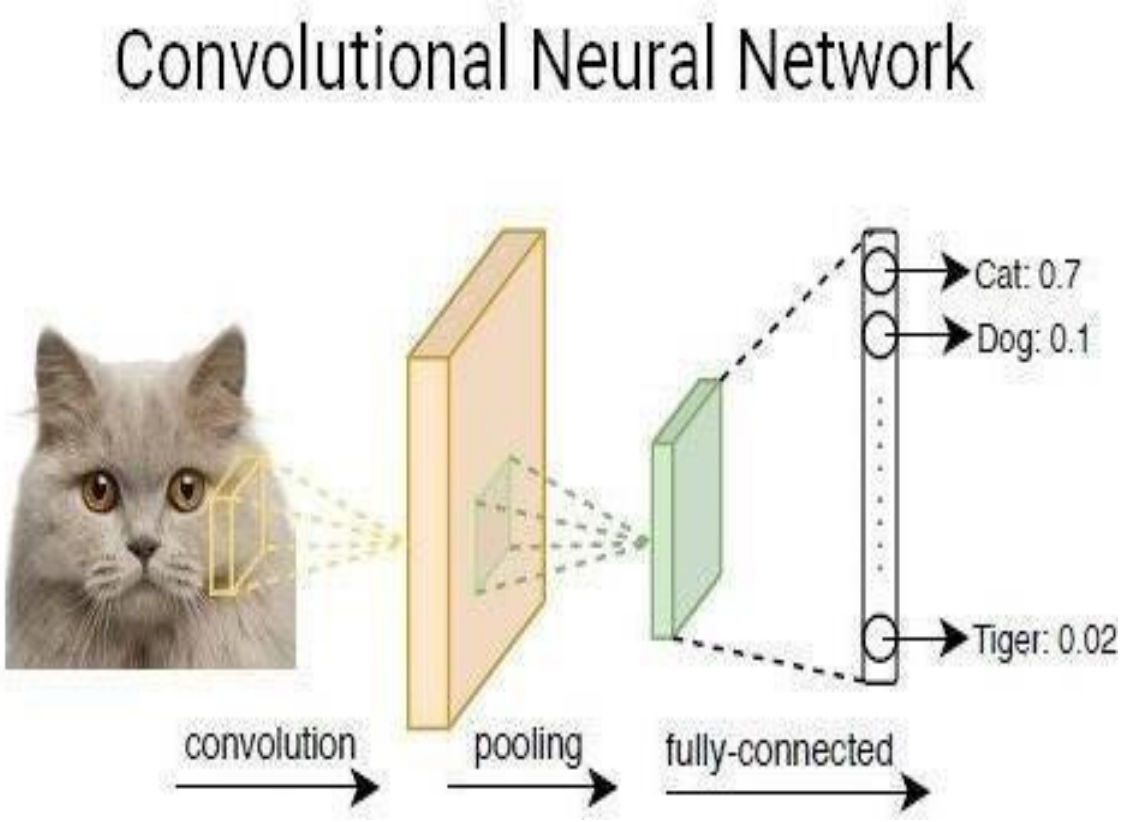


Рисунок 1.1 – Згорткова нейронна мережа [5]

1.3.1 Згортковий шар (Convolutional Layer)

Один з ключових елементів CNN – це згортковий шар. Саме він виконує найбільше обчислювальної роботи: бере кілька пікселів поруч, обробляє їх і дає щось схоже на спрощене уявлення ділянки зображення. Часто для цього використовують невеликі фільтри розміром 3×3 або 5×5. Раніше програмісти задавали ці ядра вручну, користуючись певними шаблонами. Зараз у deep learning ці фільтри підлаштовуються автоматично, просто під час навчання.

Спочатку значення фільтра випадкові, але після кількох епох мережа змінює їх так, щоб знаходити потрібні ознаки. Метод – градієнтний спуск. Сам процес згортки виглядає так: фільтр рухається по матриці зображення, бере маленькі фрагменти, перемножує їх зі своїми значеннями і формує нову карту, яка показує, що де знайшлося. Це і є карта ознак або активності – вона передається далі.

1.3.2 Шар об’єднання (Pooling Layer)

Об'єднувальні шари грають важливу роль в згорткових нейронних мережах. Вони допомагають зменшити розмір просторовий даних, які обробляє мережею, що дозволяє скоротити обсяг обчислень і зменшує кількість параметрів. Після того, як фільтри виділили головні ознаки на зображенні інформація про їх точне місцезнаходження вже не така критична. Об'єднання спростить цю інформацію що особливо допомагає знизити ризик перенавчання – коли модель занадто запам'ятовує приклади із навчальної вибірки та гірше працюють з новими даними.

На рисунку 1.2 показано приклад шару об'єднання. Є кілька різних способів, але найбільш популярним є максимальне об'єднання. Він дозволяє зменшити розміри карти ознак, залишаючи число каналів незмінним. Приклад такого об'єднання для області 2x2 можна побачити на рисунку 1.3. Завдяки цим шарам мережа краще виділяє важливі деталі зображення полегшуючи подальший аналіз.

Так само, як і згорткові шари, об’єднувальні є частиною процесу вилучення ознак. Вони працюють із картами особливостей, які отримують після фільтрації та виділення дрібних сегментів зображення. Спочатку вся обробка ведеться над двовимірними матрицями, але для отримання кінцевого результату – числового прогнозу – дані потрібно підготувати інакше. Тому в кінці процесу вилучення ознак застосовують операцію «згладжування», яка перетворює 2D матриці в одномірні масиви. Ці масиви далі передаються у повнозв’язні шари мережі, де й відбувається формування остаточного результату.

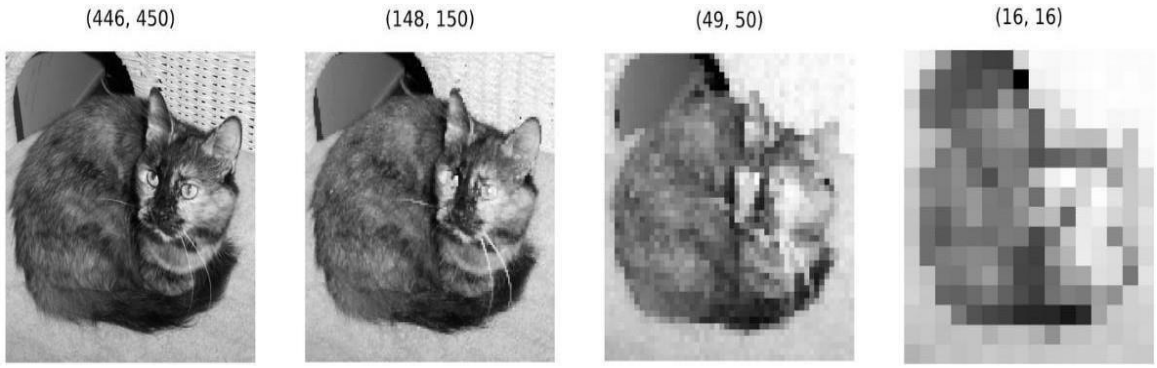


Рисунок 1.2 – Приклад шару об’єднання [6]

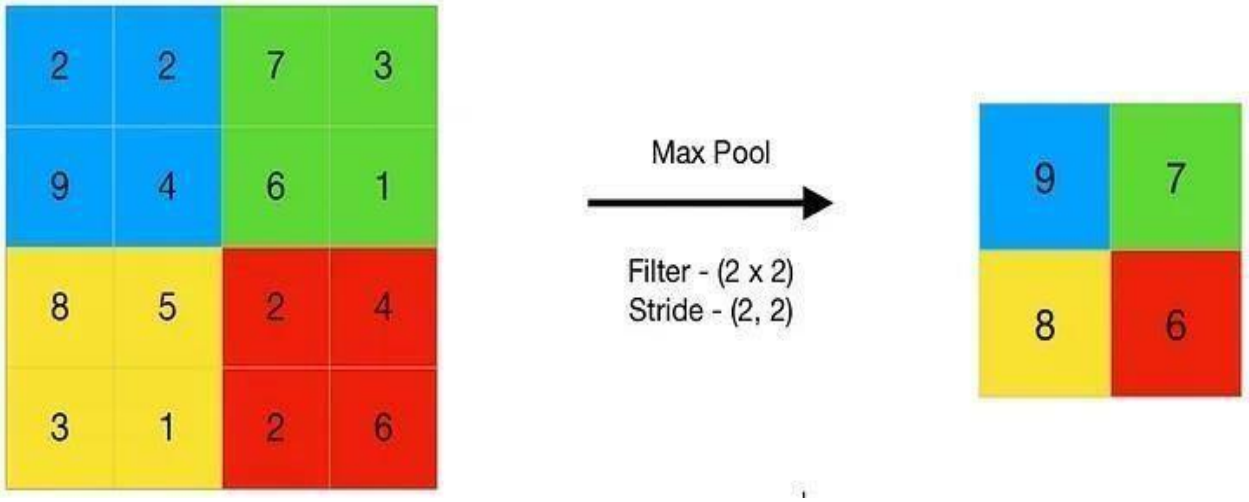


Рисунок 1.3 – Приклад максимального об’єднання шару [7]

1.3.3 Повністю зв’язаний шар (Fully Connected Layer)

Повністю зв’язаний шар у згортковій нейронній мережі – це шар, який не складається з фільтрів, а складається з нейронів, які зберігають одне значення. Такий тип шару розмішується на кінці CNN. При переході з карти активацій до повнозв’язаного шару кожне значення цієї карти з’єднується з нейроном у наступному повнозв’язаному шарі. Далі ці нейрони передають сигнал наступному шару через вагові коефіцієнти. Кількість нейронів у вихідному шарі відповідає кількості можливих категорій зображення (рис. 1.4).

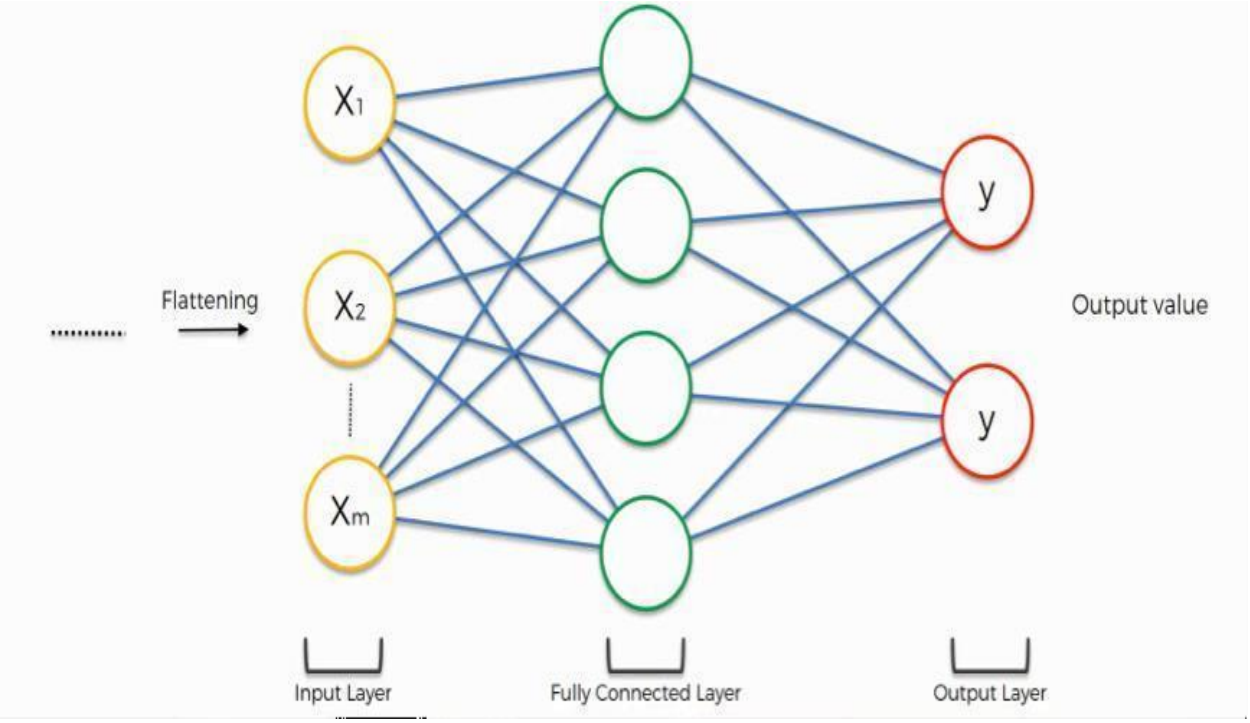


Рисунок 1.4 – Приклад повністю зв’язаного шару [8]

1.4 Виявлення об’єктів (Object Detection)

Коли йдеться про виявлення об’єктів у промисловому середовищі, насправді мається на увазі дещо більше, ніж просто знаходження предметів на зображенні. Усе починається з аналізу кадрів із камери або з відео, але далі важливо зрозуміти, які саме об’єкти присутні, як вони розміщені, і що з цього можна зробити. Це стає особливо актуальним, коли стоїть завдання автоматизувати контроль за дотриманням правил безпеки або просто отримати більше інформації для покращення процесів на виробництві. Поява Інтернету речей у поєднанні з великою кількістю сенсорів, зокрема камер, дала змогу збирати величезну кількість даних, які тепер можна обробляти за допомогою комп’ютерного зору.

Саме виявлення об’єктів – це спосіб знайти знайомі елементи на зображенні. Усе починається з ідеї, що на зображенні можуть бути певні типи об'єктів, наприклад, люди, каски, рукавиці або щось інше, і завдання полягає в тому, щоб визначити, де саме вони розташовані. Якщо при звичайній класифікації зображення модель просто каже, що на ньому є, то у випадку з розпізнаванням об'єктів потрібно ще й показати координати – об'єкти обводяться прямокутниками, для кожного з яких задаються координати верхнього кута, ширина і висота. Приклад такого розпізнавання наведено на рисунку 1.5.

Ще одна важлива відмінність – система може знаходити не один об'єкт, а кілька одночасно. Це, з одного боку, робить задачу складнішою, бо ніколи не відомо заздалегідь, скільки саме об’єктів зображено, а з іншого – відкриває більше можливостей. Наприклад, на одному кадрі можуть бути і працівник, і каска, і рукавиці, й система повинна знайти все одразу. Стандартна згорткова нейронна мережа з таким не справиться, тому в задачах виявлення використовуються спеціалізовані архітектури, які вміють працювати з динамічною кількістю об'єктів. На згаданому рисунку 1.5 система, наприклад, змогла виявити 8 об'єктів різних класів.

Усе це добре працює в теорії, але на практиці виникає безліч проблем. Якість розпізнавання сильно залежить від того, наскільки хороше зображення: чи достатньо світла, чи є чіткість, чи об'єкти не перекриті. Якщо, наприклад, камера зафіксувала працівника ззаду або під кутом, система може взагалі не впізнати каску. Також трапляється, що фон на зображенні занадто подібний до об'єкта, через що виникає плутанина. Під час навчання моделі теж можуть виникати труднощі: іноді вона запам’ятовує надто багато деталей із навчальних прикладів і не здатна адаптуватися до нових ситуацій – це називається перенавчанням. Або ж, навпаки, виявляється, що вона погано розуміє закономірності у даних і дає неточні результати – тоді це недонавчання.

У процесі роботи над проєктом довелось неодноразово зіштовхуватись із цими складнощами. Наприклад, було помічено, що зміна умов освітлення або позиції об'єкта відчутно впливала на точність результатів. Це змусило глибше розібратися в тому, як саме модель навчається, які параметри впливають на її поведінку і як краще готувати датасет. Без цього побудувати справді надійну систему для виявлення об'єктів було б складно.

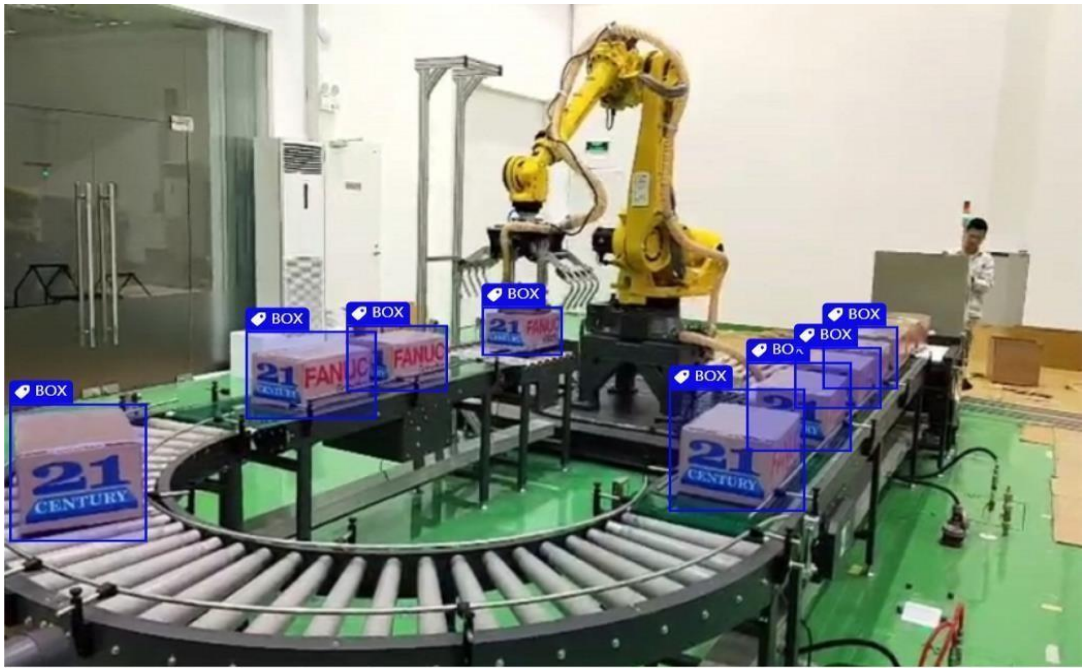


Рисунок 1.5 – Приклад розпізнавання об’єктів [9]

У класичній семантичній сегментації модель призначає кожному пікселю на зображенні певну мітку. Якщо, наприклад, це людина, всі відповідні пікселі будуть виділені в єдиний блок класу «людина». Але якщо в кадрі знаходяться дві людини, система не розділить їх – пікселі об’єднаються, і вийде одна сукупність. Це обмеження стає критичним у ситуаціях, де потрібно працювати з кожним об’єктом окремо.

Щоб подолати це, використовують інші підходи. Виявлення об’єктів дозволяє визначити координати об’єкта й задати для нього рамку. Сегментація екземплярів іде ще далі – вона точно виділяє кожен піксель, що належить конкретному екземпляру. У підсумку виходить, що такі методи дають не просто «що є на зображенні», а ще й «де саме» й «хто з чим взаємодіє». У задачах комп’ютерного зору на виробництві, в медицині чи безпеці це має велике значення, оскільки дозволяє системам працювати не просто з картинкою, а з логікою її побудови.

Виявлення об’єктів обширно застосовується в різноманітних задачах комп’ютерного зору, таких як анотація зображень, розпізнавання активності, розпізнавання облич, ко-сегментація об’єктів відео, контроль якості продукції на виробництві, навігація безпілотних засобів. Він також використовується для відстеження об’єктів. Наприклад, відстеження м’яча або окремих гравців під час футбольного матчу, у відеоспостереженні для моніторингу переміщення людей є одними з його різноманітних застосувань.

1.5 Традиційні методи виявлення об’єктів

Коли нейромережі ще не вийшли на передній план, розпізнавання об’єктів будувалося переважно вручну. Готових рішень тоді просто не існувало – усе доводилося створювати самостійно, навіть для об’єктів із неочевидною формою чи заплутаним фоном. Щоб хоч якось дати комп’ютеру уявлення про вміст кадру, будували векторні ознаки, підбирали методи вручну, намагалися полегшити

обчислення. Через нестачу ресурсів багато речей доводилося прописувати логікою – і цей ручний підхід був головним інструментом. Як правило, така система включала кілька етапів, які й брали на себе основну частину завдань.

Спершу стояло завдання знайти, де саме на зображенні може бути об’єкт. Через те, що об’єкти могли мати різні розміри і розташовуватись у довільних місцях, використовували так званий метод ковзного вікна: по всьому зображенню пересувався прямокутник, що поступово змінював свої пропорції та розміри. Так переглядали весь простір кадру. Принцип працював, але був дуже ресурсозатратним – система генерувала величезну кількість варіантів, більшість з яких виявлялися марними, і це серйозно впливало на швидкість роботи.

Після того як ймовірну область знаходили, необхідно було проаналізувати її детальніше. Саме тут у гру вступали методи витягування ознак – наприклад, HOG або SIFT. Вони дозволяли виділити характерні риси: краї, кути, напрямки або текстури, які допомагали відрізнити один об’єкт від іншого. Але на практиці вручну описати всі можливі варіації виявлялося доволі складно: освітлення могло змінитися, фон – збивати з пантелику, а сама форма об’єкта – трохи відрізнятися від очікуваного зразка. Це робило задачу нестабільною й залежною від конкретних умов зйомки.

На цьому все не закінчувалося – потрібно було ще зрозуміти, що саме потрапило в кадр. Для цього використовували класифікатори. Вони брали ознаки, які вдалося витягнути, і намагалися зіставити їх з відомими прикладами. У результаті система могла зробити припущення, до якого класу належить об’єкт. Завдяки цьому вся обробка набувала хоч якогось сенсу – хаотичні пікселі перетворювались на зрозумілу категорію, з якою вже можна було працювати далі.

Основні традиційні методи виявлення об’єктів:

– SIFT – це метод, який з’явився ще наприкінці 90-х років завдяки Девіду Лоу. Його суть у тому, щоб знаходити ключові точки, які не залежать від повороту, масштабу чи зміни положення об’єкта в кадрі. Навіть при іншому освітленні він здатен впевнено «впізнавати» елементи. Цей підхід часто

використовували для того, щоб порівнювати або ідентифікувати схожі об’єкти на різних зображеннях;

– метод HOG з’явився у 2005 році, коли його представили Далал і Триггс. Він дозволяв аналізувати напрямки змін яскравості – тобто те, як змінюється світло і тіні на зображенні. Завдяки цьому можна було визначати контури та форми. Найчастіше його використовували для виявлення людей, оскільки навіть при зміні масштабу HOG залишався надійним. Щоб це працювало, саму картинку масштабували, а рамку для аналізу не змінювали;

– метод SURF з’явився трохи пізніше – у 2006 році. Його розробив Герберт Бей разом із командою. У цьому випадку зробили ставку на швидкість. Алгоритм дозволяв оперативно знаходити характерні точки на зображенні, використовуючи прямокутні фільтри. Це стало особливо корисним для задач, де потрібно було працювати із відео або відстежувати об’єкти у реальному часі;

– ORB – це метод, який запропонував Лі Сяохун разом із командою у 2012 році, щоб краще працювати з динамічними сценами, де все рухається. Ідея в тому, що він може розпізнавати, як саме об’єкт або камера повертаються чи зміщуються – для цього використовують модель з восьми параметрів, яка допомагає зрозуміти, куди і як рухається об’єкт. Потім застосовують метод найменших квадратів, щоб компенсувати всі ці рухи і не плутатися в кадрах. Щоб визначити, які саме об’єкти рухаються, порівнюють сусідні кадри, виділяючи області, що змінюються. ORB поєднує в собі те найкраще, що було у SURF, але при цьому працює набагато швидше і точніше. Саме тому цей метод часто використовують, коли потрібно виявляти рухомі об’єкти в реальному часі.

2 АЛГОРИТМИ ВИЯВЛЕННЯ ОБ’ЄКТІВ НА ОСНОВІ ГЛИБОКОГО НАВЧАННЯ І ЗАСОБИ ЇХ РЕАЛІЗАЦІЇ

Старі методи розпізнавання об'єктів були колись важливими, але в них був один великий недолік: все трималося на тому, чи зможе людина, той самий розробник, придумати й пояснити машині, за якими саме деталями вона має впізнавати, скажімо, кота чи машину. Потрібно було бути справжнім професіоналом в цій темі, але всеодно, як тільки зображення трохи складніше – інше світло, об'єкт дивно виглядає чи маленький – система часто хибила. А потім з'явилося глибоке навчання і, все сильно змінилося. Тепер не треба сидіти й вигадувати ці ознаки. Потрібно просто показати моделі багато прикладів – тисячі зображень і вона сама навчиться їх знаходити. Завядки цій здатність докопатися до суті прямо з картинок і зробила такий прорив, піднявши розпізнавання об'єктів на зовсім інший рівень і відкривши йому дорогу в купу різних сфер.

Отже глибоке навчання (DL) справді дає нам потужний та ефективний інструмент, щоб навчити машини автоматично бачити та виділяти ключові ознаки в зображеннях. Але коли мова заходить про виявлення об'єктів, просто сказати, що на зображенні є автомобіль – це лише пів справи. Критично важливо ще й точно вказати, де саме цей автомобіль знаходиться, вказавши його межі. Для вирішення цієї задачі – одночасної класифікації та локалізації, інженерна думка пішла кількома основними шляхами. Оскільки універсального рішення не знайдено, з плином часу з’явилися два основні підходи до розробки архітектур детекторів.

2.1 Двоступеневі детектори (Two-Stage Detectors)

Перший такий підхід – це двоступеневі детектори (Two-Stage Detectors). Суть у тому, що вони не намагаються одразу розпізнати все, а спочатку просто передбачають, де може бути щось цікаве. Якби це робила людина, то вона б

спочатку подивилась на фото й сказала б: "о, тут може бути людина, а тут – коробка". А вже потім підійшовши ближче й подивившись уважно – що саме це таке.

У цих моделях перший крок – це створення таких прямокутних зон, де можливо є об’єкт. Це робить окремий блок (наприклад, RPN – Region Proposal Network або інший). Він просто вказує, куди дивитись. А потім інша частина моделі перевіряє кожну із запропонованих зон і каже, що в ній знаходиться, та уточнює межі об’єкта.

Чому це зручно? Бо не треба витрачати ресурси на всю картинку одразу. Але є й мінуси – це не найшвидший метод. Попри це, такі підходи вважаються досить точними, особливо коли об’єкти маленькі або складно розрізняються. У цьому напрямку придумали багато моделей – типу R-CNN, Fast R-CNN і так далі. Усі вони будувались на тому ж принципі, але щоразу з певними покращеннями. Розглянемо їх детальніше.

2.1.1 R-CNN

R-CNN запропонував Hirschik та ін., вона є першим варіантом регіонального класу TSD. Hirschik використовував архітектуру AlexNet і показав, як CNN можна використовувати для покращення продуктивності OD. У R-CNN вхідне зображення подається в мережу, а його середнє значення віднімається від нього. Змінене зображення потім подається в модуль пропозицій регіону, який використовує підхід вибіркового пошуку для створення 2000 регіонів інтересу (ROI), тобто областей зображення, які, швидше за все, містять об’єкти. Потім кандидати ознак, отримані з ROI, перетворюються та передаються через мережу CNN, що складається із згорткових і повністю зв’язаних шарів (п’ять згорткових шарів і два повністю зв’язані шари), яка обчислює вектор ознак із 4096 вимірами для кожного речення. Вхідний розмір отриманого зображення 227×227 пікселів. Потім виконується немаксимальне придушення (NMS) для регіонів, оцінених SVM на основі їх категорій і перетину над об’єднанням (IoU). Після визначення класу навчений регресор

обмежувальної рамки використовується для прогнозування обмежувальної рамки шляхом оцінки чотирьох параметрів, якими є координати центру рамки, а також її ширина та висота. Приклад можемо побачити на рисунку 2.1. R-CNN характеризується низькою продуктивністю (47 секунд на зображення) і займає багато часу і місця. Ця проблема особливо серйозна при роботі з меншими наборами даних [10].

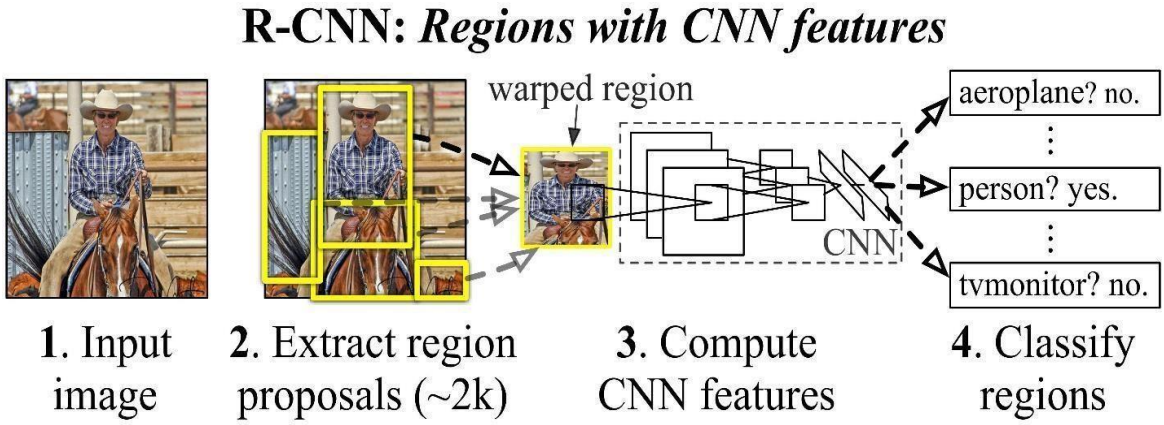


Рисунок 2.1 – R-CNN архітектура [11]

2.1.2 SPP-Net

R-CNN використовує операції обгортання та обрізання за пропозицією кожного регіону для повністю з'єднаного шару, який приймає тільки вхідне зображення фіксованого розміру. Операція обрізання може призвести до часткової втрати вмісту бажаного об'єкта, а операція обгортання може спричинити геометричне спотворення [12]. Щоб якось обійти цю неприємність, свого часу запропонували досить цікаву архітектуру, яку назвали SPP-net. Її основна ідея полягає у просторовому пірамідальному зіставленні. Це дозволяє системі працювати з вхідними даними довільного розміру. Це великий плюс, бо, на відміну від класичних згорткових нейронних мереж (CNN), які завжди вимагають фіксованого розміру зображення, SPP-net не має таких обмежень.

У цьому підході застосовується кілька рівнів масштабування. Вони розбивають зображення на частини, а потім з цих частин збираються

(агрегуються) локальні ознаки. У результаті ми отримуємо більш узагальнене представлення. На рисунку 2.2 можна побачити, як ця архітектура повторно використовує вже витягнуті ознаки з п'ятого шару мережі (conv5), щоб отримати вектори сталої довжини, причому з регіонів, які були довільного розміру. Це дуже зручно для подальшої роботи.

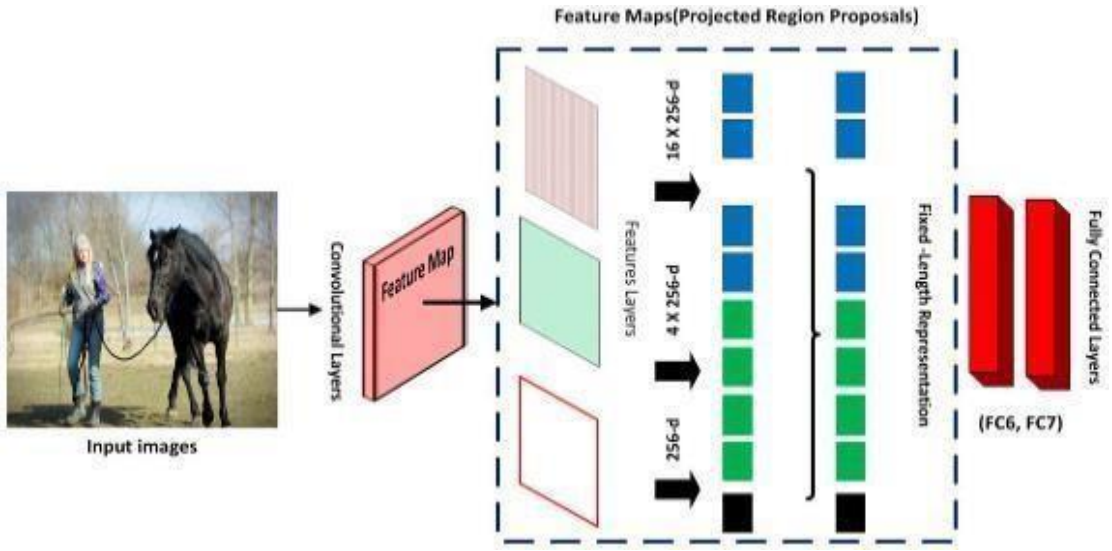


Рисунок 2.2 – SPP-Net [12]

2.1.3 Fast R-CNN

Fast R-CNN – це детектор об'єктів, розроблений спеціально в 2015 році Россом Герчіком, дослідником штучного інтелекту в Facebook і колишнім дослідником Microsoft. Fast R-CNN вирішує багато проблем R-CNN. Як випливає з назви, однією з переваг Fast R-CNN над R-CNN є його швидкість (Girshick, 2015) [12]. У Fast R-CNN принцип обробки зображення суттєво змінено порівняно з R-CNN. Якщо раніше кожну область інтересу (RoI) вирізали окремо і подавали до мережі, то тут вся картинка обробляється згортковою мережею лише один раз. Це дозволяє не дублювати обчислення та істотно знижує навантаження. Коли карта ознак вже сформована, за допомогою алгоритму selective search визначаються області інтересу, які потім передаються на шар RoI pooling. Його роль – стандартизувати розмір кожної області, щоб їх можна було подавати на вхід наступним шарам. Після цього працює класифікатор softmax,

який призначає кожному регіону відповідний клас. Паралельно система виконує точне визначення меж об'єкта через модуль регресії обмежувальних рамок. Такий підхід дозволяє об'єднати точність і швидкість, що було головною метою Fast R-CNN. Схема моделі подана на рисунку 2.3.

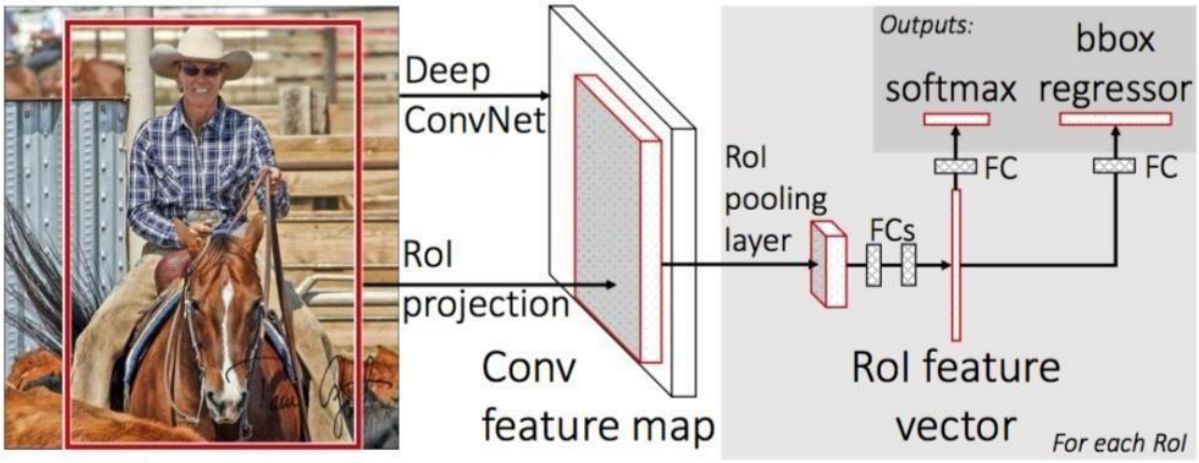


Рисунок 2.3 – Fast R-CNN [11]

2.1.4 Faster R-CNN

Алгоритми, які ми вже обговорювали вище, базуються на вибірковому пошуку для виявлення пропозицій регіонів. Цей метод досить повільний і вимагає великої трати ресурсів. Тож це сильно впливає на продуктивність. Щоб вирішити цю проблему, розробники Faster R-CNN запропонували іншу структуру для виявлення об’єктів. В цій структурі формування регіонів відбувалося всередині самої нейромережі. Її суть полягала у використанні ознак зображення, які вже обчислені під час прямого проходу через нейронну мережу, для генерації пропозицій регіонів. Для цього вони створили окрему мережу – Region Proposal Network. Вона сама генерує регіони-пропозиції й одночасно передбачає обмежувальні рамки.

У Faster R-CNN була реалізована інтеграція двох ключових складових – регіональної пропозиційної мережі (RPN) та детектора Fast R-CNN – у єдину архітектуру. На першому етапі згорткова нейромережа обробляє зображення та

будує карту ознак. Саме з цією картою далі працює RPN, яка аналізує її за допомогою ковзного вікна розміром 3×3. У кожній позиції такого вікна генеруються пропозиції регіонів об’єктів, причому базуються вони на заздалегідь визначених анкерних прямокутниках (anchor boxes), які відрізняються між собою за формою й масштабом. Архітектурна схема цієї моделі показана на рисунку 2.4.

Згенеровані RPN області передаються до шару RoI Pooling, який відповідає за те, щоб привести кожну область до стандартного розміру, незалежно від її початкових параметрів. Завдяки цьому обробка подальшими шарами відбувається без ускладнень. Після цього підготовлені карти ознак подаються на вхід повнозв’язного шару, де реалізовано два основні завдання: класифікація об’єкта за допомогою softmax-функції та уточнення меж об’єкта через регресію координат. Такий підхід дав змогу суттєво підвищити ефективність та швидкість роботи всієї моделі.

Потім він класифікує об'єкти та прогнозує обмежувальні рамки для виявлених об'єктів. У швидшій R-CNN застосовується лише одна CNN для пропозицій регіонів та класифікації. Швидша R-CNN оптимізована для багатозадачної функції втрат, що включає класифікацію та регресійні втрати. Мережа пропозицій регіонів (RPN) – це згорткова нейронна мережа, яка пропонує регіони. Водночас, друга мережа є швидкою R-CNN для вилучення ознак та виведення обмежувальної рамки та міток класів. RPN оптимізована для заданої багатозадачної функції втрат [14].

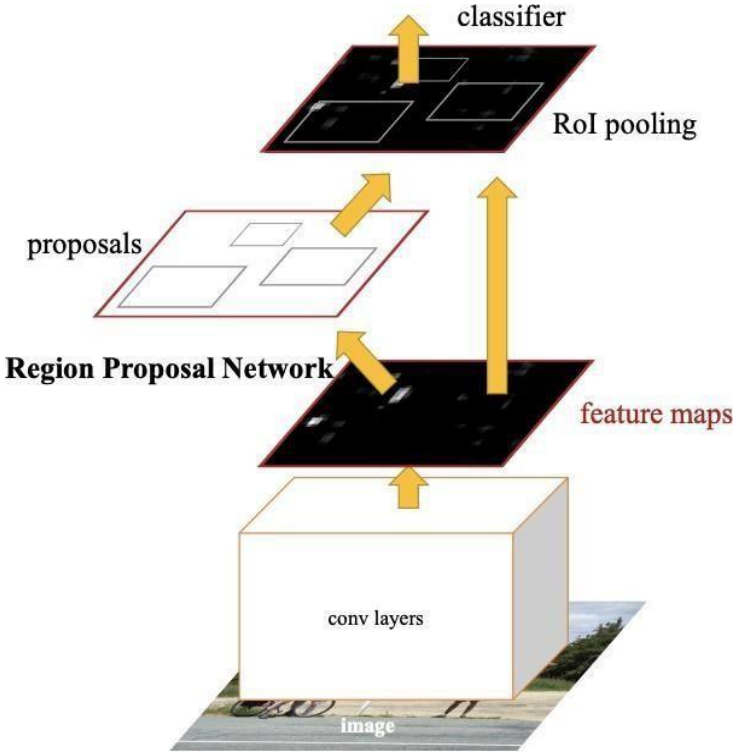


Рисунок 2.4 – Схематичне зображення архітектури Faster R-CNN [14]

2.1.5 Mask R-CNN

Архітектура Mask R-CNN представляє собою значне функціональне доопрацювання Faster R-CNN. Вона додає ще один блок до тих, що вже є для класифікації та визначення рамок. Що робить цей новий блок? Генерує точну маску сегментації. Це відбувається для кожної ідентифікованої області інтересу (RoI). Тобто, система не просто малює рамку, а точно каже, які пікселі всередині RoI належать об'єкту. Прогнозування цієї маски здійснюється на піксельному рівні, зазвичай за допомогою компактної згорткової підмережі. Добре те, що ця нова функція майже не гальмує систему. Додаткове обчислювальне навантаження порівняно з Faster R-CNN є мінімальним. Але є один недолік, для якісної сегментації потрібна дуже висока просторова точність. А звичайний шар RoI Pooling, який був у Faster R-CNN округлювa координати (це називається квантування). Через це точність втрачалася. Для простих рамок цього вистачало, але для точних масок – вже ні. Тому автори Mask R-CNN придумали RoIAlign. Цей вдосконалений шар виконує вибірку ознак без операцій квантування. Він зберігає точне просторове вирівнювання.

Як результат – маски виходять набагато якісніші. Загальну схему архітектури можемо побачити на рисунку 2.5.

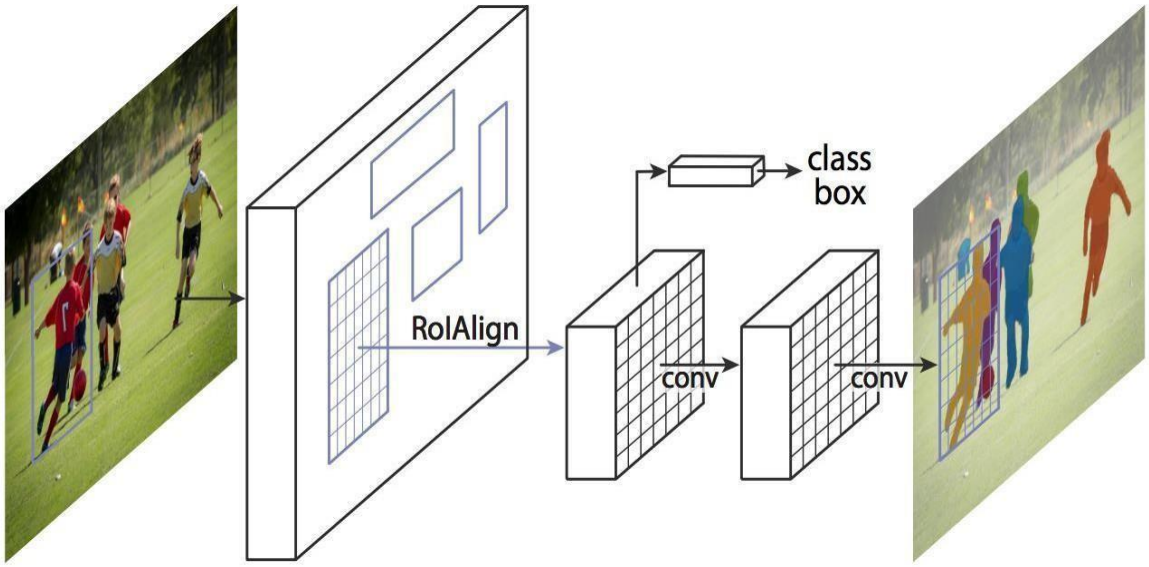


Рисунок 2.5 – Mask R-CNN [15]

2.2 Одноступеневі детектори

Попри схожість із двоступеневими методами, одноступеневі підходи поступово вдосконалюються – їхня точність зростає, а в окремих випадках вони навіть випереджають моделі з двоетапною обробкою. На відміну від двоступеневих детекторів, які розділяють процес виявлення на дві фази – генерацію пропозицій та класифікацію областей, одноступеневі детектори не мають окремого етапу для генерації пропозицій. Зазвичай вони розглядають усі позиції на зображенні як потенційні об'єкти та намагаються класифікувати кожну область, що представляє інтерес, як фон або об'єкт-ціль [15].

Глибина мережі, звісно, має значення – чим більше шарів, тим більше нюансів модель може вловити. Але тут є зворотна сторона: чим глибше, тим повільніше. І якщо потрібно щось знайти прямо зараз – така повільність шкодить. З дрібними об’єктами взагалі біда. Втім, поєднання ознак із різних

шарів рятує ситуацію: тоді модель не пропускає важливе, навіть якщо об’єкт ледь видно. Це реально критично, коли мова про роботу в реальному часі – наприклад, для безпілотників чи систем безпеки. Одноступеневі моделі мають тут перевагу: вони швидші, не ловлять зайвого з фону. Але є нюанси – часом вони промахуються, особливо якщо об’єкт невеликий або малоконтрастний. Що ж, саме тому далі розглядаються конкретні моделі, які належать до цього підходу.

2.2.1 OverFeat

OverFeat – один із перших успішних одноступеневих детекторів на основі глибокого навчання. Основна інновація методу OverFeat полягає в інтеграції багатомасштабних методів, методів ковзного вікна, зміщення пулу, ідентифікації, локалізації та виявлення на основі AlexNet [16]. OverFeat демонструє суттєву перевагу в швидкості роботи завдяки тому, що використовує згорткові шари для повторного використання перекриваних областей. Це дозволяє уникати зайвих обчислень, які були притаманні попереднім підходам, наприклад, R-CNN. Проте, незважаючи на таку ефективність, у моделі все ж є обмеження: класифікатор і регресор навчаються окремо, що ускладнює їх спільну оптимізацію. На рисунку 2.6 подано приклад, як саме побудовано архітектуру OverFeat.

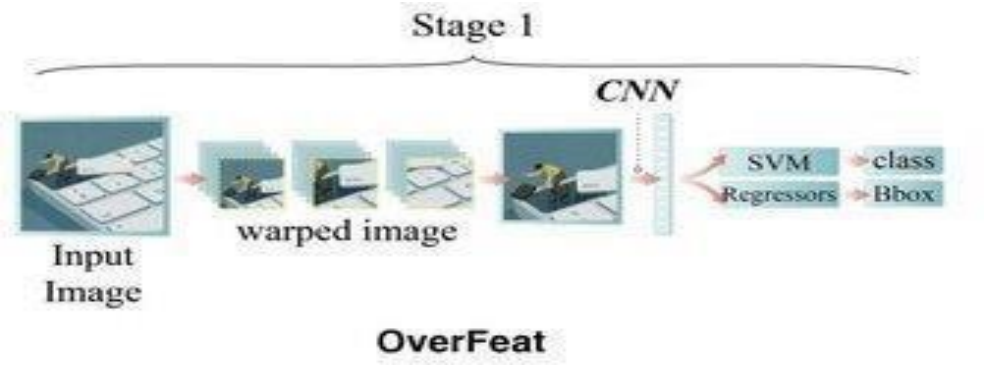


Рисунок 2.6 – OverFeat [16]

2.2.2 Single shot detector

SSD (Single shot detector) використовується як багатоблочний детектор для локалізації на зображенні. На рисунку 2.7 зображено як побудувати SSD за допомогою VGG-16 та інших додаткових шарів елементів. Побудова детектора SSD для ефективного багатоблокового виявлення об'єктів на окремому зображенні відбувається поетапно. Спочатку вхідне зображення слугує вихідними даними для обробки мережею VGG-16, яка виступає як базова структура. Далі ця архітектура розширюється, тобто до неї приєднуються чотири спеціально розроблені додаткові шари та один повністю зв'язаний шар. Саме така послідовність дій та конфігурація компонентів дозволяють сформувати детектор SSD, готовий до ефективного багатоблокового пошуку на одному зображенні. Модель створює певну кількість обмежувальних рамок для кожного зображення. Вони не беруться навмання – у кожної заздалегідь визначені розміри й пропорції. Це важливо, бо різні об’єкти мають різну форму. Людина, наприклад, у кадрі частіше виглядає витягнутою вгору, а авто – розташоване горизонтально. Через це й розміри рамок підбирають по-різному: щоб краще відповідали тій формі, що очікується. Так легше знайти потрібний об’єкт.

Щоб отримати високу точності на наборі даних для навчання та валідації використовується збільшення даних та Inferencetime. Існує два типи моделей SSD: SSD300, що означає вхідне зображення з низькою роздільною здатністю (300×300), та SSD512, що означає вхідне зображення з високою роздільною здатністю (512×512) та високою ефективністю.

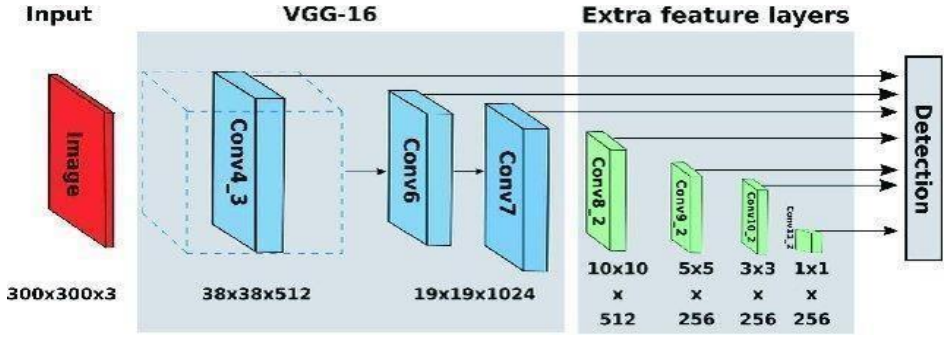


Рисунок 2.7 – SSD (Single shot detector) [17]

2.2.3 RetinaNet

RetinaNet – це сучасна модель об’єктної детекції, яка поєднує простоту одноступеневого підходу та точність, близьку до складніших двоступеневих архітектур. Основою її ефективності є функція втрат focal loss, яка допомагає розв’язати проблему сильного дисбалансу між позитивними й негативними прикладами. У задачах детекції фонові приклади часто переважають, через що стандартні функції втрат можуть “придушувати” сигнали від об’єктів. Focal loss компенсує це, зменшуючи вагу легких прикладів і посилюючи важливість складних.

Модель побудована на базі ResNet, яка виступає як екстрактор ознак. Поверх неї накладається FPN – структура, що дозволяє аналізувати зображення на кількох масштабах. Завдяки цьому RetinaNet здатна виявляти об’єкти незалежно від їхнього розміру. Далі мережа розгалужується у два шляхи: один відповідає за класифікацію, інший – за регресію координат. Обидва модулі побудовані виключно на згорткових шарах, що дає моделі змогу працювати швидко й ефективно навіть на великих зображеннях.

Класифікаційна гілка генерує ймовірності класів для заздалегідь визначених anchor boxes. Регресійна – видає зміщення координат, щоб адаптувати ці рамки до реальних об’єктів. Обидві частини працюють синхронно, а підсумкове рішення формується після постобробки. Як видно на рисунку 2.8, архітектура RetinaNet включає всі ці елементи в компактній, але потужній структурі.

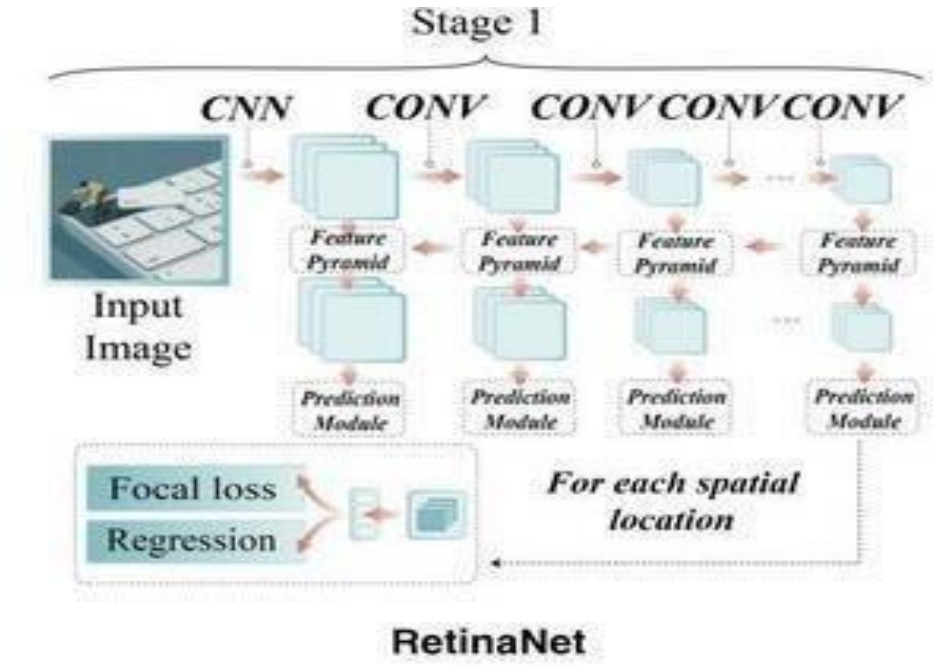


Рисунок 2.8 – RetinaNet [16]

2.2.4 YOLO

YOLO (You Only Look Once) – популярний метод виявлення об'єктів (розробка Редмона та ін., з подальшими покращеннями), який підходить до задачі виявлення як до регресійної. Нейромережа визначає об'єкти безпосередньо з пікселів зображення. Кожен виявлений об'єкт описується обмежувальною рамкою (положення, розмір). Також для нього розраховуються ймовірності належності до всіх релевантних класів (сума = 1). Об'єкт класифікується за найвищою ймовірністю.

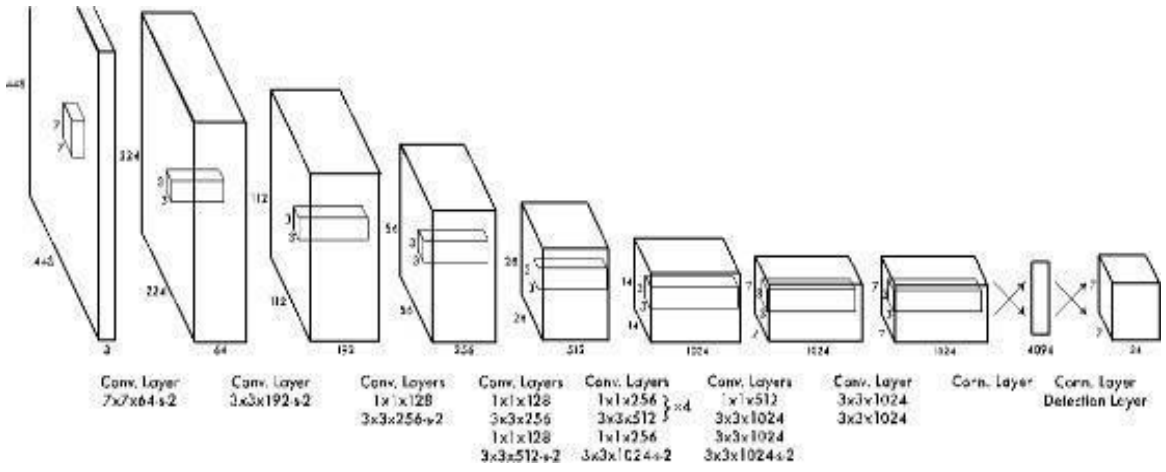


Рисунок 2.9 – Архітектура YOLO [10]

Стандартний робочий процес YOLO:

– зображення масштабується до стандартного розміру, на нього накладається сітка;

– згорткова нейромережа вилучає ознаки. Комірки сітки через повнозв'язні шари виконують регресію: кожна прогнозує K рамок з 5 значеннями (4 – позиція/розмір, 1 – достовірність/точність);

– для комірок з об'єктами повнозв'язний шар прогнозує умовні ймовірності їхніх класів.

Обмеження YOLO:

– мала кількість прогнозованих рамок на комірку сітки (2 в оригіналі);

– лише одне передбачення класу на комірку. Це обмежує виявлення, коли кілька об'єктів (особливо дрібних) знаходяться в одній комірці. Тому виникають труднощі з детектуванням щільних груп дрібних об'єктів.

Проте розвиток YOLO не стояв на місці. Початкові обмеження поступово вирішили. Вже YOLOv3 принесла значні зміни: здатність працювати з кількома об'єктами в одній зоні сітки, краще розпізнавання різнорозмірних цілей. Допомогли цьому anchor boxes та feature pyramid networks. Точність на складних сценах помітно зросла.

Порівняння між одноступеневими детекторами можемо побачити в таблиці 2.1.

Таблиця 2.1 – Порівняння одноступеневих детекторів

Детектор	Перевага	Недолік
OverFeat	Простота реалізації; швидка обробка зображень	Низька точність; погано працює з дрібними об'єктами
SSD	Висока швидкість роботи; добре справляється з великими об'єктами; підтримка мобільних платформ	Менша точність на дрібних об'єктах; гірша робота при перекритті
RetinaNet	Збалансована точність і швидкість; використовує механізм фокусної втрати (focal loss) для роботи з важкими прикладами	Більше споживання пам’яті; довший час навчання
YOLOv8	Висока точність; стабільна робота з дрібними й перекритими об’єктами; проста адаптація до власного датасету; сучасна архітектура без фіксованої сітки	Потребує більше ресурсів для навчання; не така легка для запуску на слабкому залізі; частково новий формат анотацій

Хоча існують новіші версії, для своєї дипломної я обрав YOLOv8. Причина – її ефективність. Вона точніша за попередників. Це особливо цінно, коли об'єкти різні за розміром, перекриваються чи розташовані близько. До того ж, v8 не ділить зображення на жорстку сітку. Це вирішує стару проблему "один клас на комірку". І дрібні об'єкти вона знаходить краще. Щодо новіших версій, то хоч вони мають деякі покращення, вони поки що менш стабільні, вимагають більше ресурсів і часто не мають такого ж рівня підтримки, як

YOLOv8. Крім того, новіші версії ще не так добре документовані або оптимізовані під практичне застосування в реальному середовищі.

2.3 Обрані бібліотеки та інструменти для реалізації

Для реалізації системи розпізнавання об’єктів у рамках цього проєкту планую використовувати низку перевірених бібліотек, які допоможуть ефективно організувати процес навчання, обробки зображень та оцінювання результатів.

Основна роль у проєкті відводиться бібліотеці Ultralytics. Це зручний інструмент для роботи з моделями YOLO, зокрема з версією YOLOv8, яку я обрав для реалізації. Ця бібліотека дозволяє досить просто запускати тренування моделі, перевіряти її на валідаційному наборі та адаптувати її під власний набір даних. Також у ній вже реалізовані популярні методи аугментації, логування, збереження результатів та запуск інференції.

Для роботи зі зображеннями буду використовувати OpenCV – через неї планується організувати захоплення відео з вебкамери, а також збереження кадрів, де зафіксовано порушення. Щоб ці зображення можна було зручно показати у вікні, буду використовувати Pillow, який допомагає конвертувати кадри для графічного інтерфейсу.

До речі, графічний інтерфейс сам по собі буде побудований на Tkinter. Це проста, але зручна бібліотека, яка дозволяє зробити невелике вікно, де буде відео з камери та повідомлення про стан – чи є порушення, чи ні. Мені це потрібно, щоб система могла не просто працювати всередині, а й зручно подавати інформацію користувачу.

Ще кілька бібліотек будуть допоміжними. Наприклад, os – щоб створювати папки для збереження кадрів, або читати шляхи до файлів. gc (garbage collector) – для звільнення пам’яті після тренування, що особливо актуально при роботі на локальному комп’ютері. csv – планую використовувати для ведення логів: коли було зафіксовано порушення, що

саме було виявлено, і так далі. Ну і datetime з time – вони мені потрібні, щоб фіксувати час збереження кадру і уникати дублювань, якщо кілька порушень відбуваються підряд.

Усі ці інструменти перевірені, мають активну підтримку і дозволяють гнучко реалізувати як навчальну, так і прикладну частину проєкту.

3 РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ У ВИРОБНИЧОМУ ПРИМІЩЕННІ

3.1. Постановка задачі розробки

3.1.1. Актуальність та мета розробки автоматизованої системи

Сучасні виробничі підприємства з кожним днем все більше залежать від автоматизації технологічних процесів та систем контролю. Один з головних напрямків для підвищення безпеки праці та якості продукції – це впровадження інтелектуальних систем моніторингу. Завдяки ним можна аналізувати візуальну інформацію з виробничих приміщень. Завдяки розвитку комп’ютерного зору та глибокого навчання з’явилась можливість автоматизувати процес контролю дотримання техніки безпеки і оптимізувати роботу служби охорони праці.

Мета цієї роботи: розробити автоматизовану систему розпізнавання об’єктів у виробничому приміщенні, а саме вона нам дозволить в режимі реального часу аналізувати відеопотік із камери і виявляти, чи має працівник необхідне спорядження відповідно до правил безпеки. Ця система не покладається на здогадки – вона точно знає, чого бракує, і фіксує це: фото, час, тип порушення. Така фіксація може стати серйозним інструментом для підвищення безпеки на підприємстві. Дана система не є заміною людині, вона – це засіб автоматизації того, що зазвичай може загубитись через людський фактор.

3.1.2. Вимоги до системи

Перш за все, система має вміти виконувати свої прямі задачі – це її функціональні вимоги. Серед основного: розпізнавання людей у кадрі та перевірка, чи мають вони відповідне спорядження. Якщо немає шолома або респіратора – програма має це помітити і створити запис. Як мінімум: кадр із

відео, інформація про час, можливо, навіть камера, яка це зафіксувала. Також важливо, щоби програма працювала в режимі реального часу. Бо якщо є затримка в кілька секунд, порушення можна просто не встигнути зафіксувати. Підтримка кількох типів камер – як звичайної вебки, так і IP-камер – це теж необхідність, адже в різних умовах можуть використовуватись різні джерела відео. Картинка з відео має бути зрозумілою. Якщо система щось розпізнала – це потрібно якось відобразити: рамки, написи, кольорові позначки. Людина, яка стежить за монітором, повинна одразу розуміти, що саме вона бачить.

Тепер – нефункціональні речі. По суті, це все, що впливає на зручність, стабільність і життєвий цикл програми. Система повинна працювати надійно, без частих зависань або помилок. Її має бути легко запустити навіть тим, хто не є фахівцем у програмуванні. Мінімум налаштувань, максимум зрозумілих кнопок. Платформа – бажано не обмежуватись лише Windows. Якщо є можливість запускати на Linux – це величезний плюс. Також важливо, щоби проєкт можна було надалі розвивати: додати нову камеру, новий клас об’єкта, змінити конфігурацію – усе без переписування з нуля.

3.1.3. Опис об’єктів розпізнавання та специфіка виробничого серидовища

Для того щоб наша система ефективно сприяла безпеці, її "зір" націлений на досить конкретний перелік об'єктів. У центрі уваги завжди перебуває людина (Person) – саме її безпека є кінцевою метою моніторингу. Виявивши людину у полі зору камери, програма переходить до детальнішого аналізу, а саме – перевірки наявності чи, навпаки, відсутності на ній життєво важливих елементів захисного спорядження. Наприклад, якщо на голові працівника відсутній шолом, система це фіксує, маркуючи ситуацію як "No_Helmet", що, погодьтеся, є тривожним дзвіночком про потенційну небезпеку. Подібний алгоритм застосовується і для контролю наявності захисних рукавиць; їх відсутність на руках буде позначена міткою "No_Glove". Не меншою є увага до засобів, що оберігають органи дихання – респіраторів,

їх ігнорування, позначене як "No_BreathingApparatus", може мати серйозні наслідки для здоров'я. Система також навчена ідентифікувати окуляри ("Glasses"), котрі, хоч і можуть бути просто коригувальними, у багатьох виробничих сценаріях виконують важливу захисну функцію. Вибір об'єктів, на які система звертає першочергову увагу, зумовлений наступним: аналіз виробничих небезпек та стандартні інструкції з охорони праці чітко вказують, де найчастіше криються загрози, отже, програма концентрується на тому, що справді важить для безпеки людей.

Розробити таку систему – це непроста задача. Адже у виробничих приміщеннях умови сильно відрізняються від звичайних. Наприклад, у таких умовах часто буває нерівномірне освітлення. В деяких місцях занадто темно, а десь навпаки аж засліплює. Також може ускладнювати задачу, чистота повітря, а саме пар, пил або якісь дрібні частинки. Також на виробництві люди постійно рухаються, працює техніка і предемети можуть затуляти потрібні нам об'єкти. Всі ці чинники вимагають від алгоритмів гнучкості. Зрозуміло, що навчити модель однаково добре працювати за всіх цих умов – це велика робота. Тому важливо розуміти, що будь-яка подібна система, і наша зокрема, потребуватиме тестування в реальних умовах та, можливо, подальших доопрацювань, щоб її ефективність була максимальною саме на конкретному виробництві.

3.2. Проєктування системи

3.2.1 Обґрунтування вибору архітектури системи

Було вирішено, що розроблювана система моніторингу функціонуватиме на базі локальної (standalone) архітектури. Суть її в тому, що всі операції, від прийому відео з веб-камери (через Anaconda) до аналізу нейромережею і показу результатів у "Моніторі безпеки", виконуються на одному ПК.

Для цілей кваліфікаційної роботи вибирати складніші структури,

наприклад, клієнт-серверні, не було практичної потреби. Такий крок лише ускладнив би процес створення, не покращивши демонстрацію ключової функції – розпізнавання об'єктів. Саме локальний варіант дав змогу сфокусуватися на алгоритмах машинного зору. Сучасний комп'ютер цілком справляється з обробкою відео з однієї камери; дані про виявлені порушення також зберігаються на місці, що зручно для поточних завдань. У підсумку, локальна побудова системи стала оптимальним поєднанням необхідних можливостей та простоти втілення.

3.2.2 Розробка загальної схеми

Логіку роботи розробленої автоматизованої системи розпізнавання об'єктів демонструє загальна схема її функціонування (рис. 3.1). На цій схемі детально відображено ключові етапи, які система проходить від старту до виведення результатів моніторингу. Блок-схема також ілюструє основні цикли та точки прийняття рішень в алгоритмі.

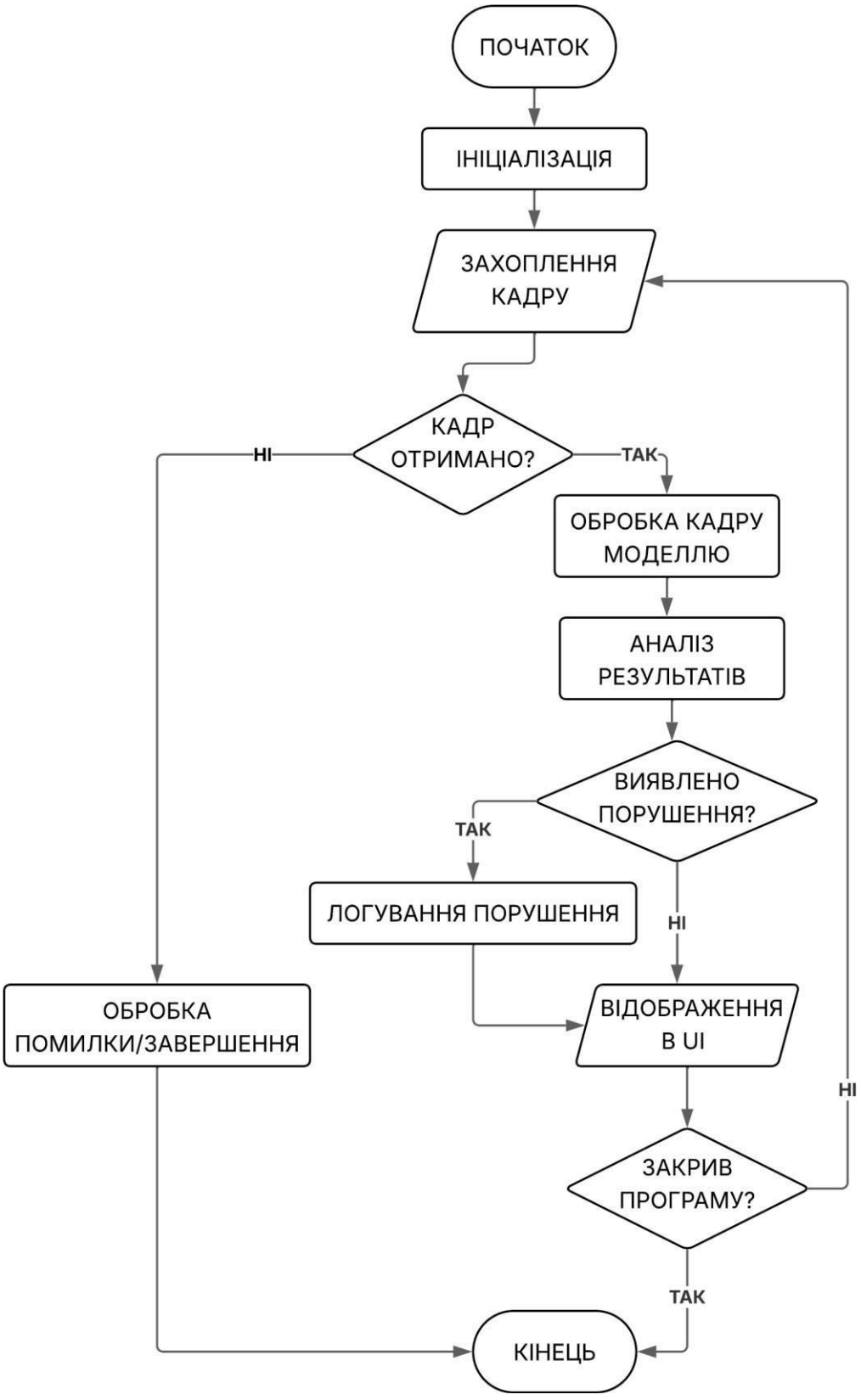


Рисунок 3.1 – Загальна блок схема системи

Отже, як видно зі схеми на рисунку 3.1, робота системи починається з блоку “Початок” – це момент, коли програма отримує команду на запуск. Відразу після цього система переходить до етапу Ініціалізація системи. Тут відбувається вся необхідна підготовка: завантажуються програмні бібліотеки, підключається веб-камера, а головне – завантажується навчена модель YOLOv8 завдяки якій будуть розпізнаватись об'єкти. Також готується вікно програми, де все буде відображатися.

Далі починається основний цикл. Перший крок у цьому циклі – це захоплення кадру з веб-камери: система робить знімок поточного моменту з веб-камери. Настпний блок "Кадр отримано?" запитує чи вдалося отримати цей кадр. Це точка, де алгоритм робить вибір. Якщо з кадром щось не так, наприклад, камера не відповідає (гілка "Ні"), то система переходить до "Обробки помилки/Завершення", де намагається обробити та записати інформацію про помилку, і далі шлях веде до завершення роботи.

Якщо ж кадр отримано успішно (гілка "Так"), він йде на найважливіший етап – “Обробка кадру моделлю YOLOv8”. Тут вже починає працювати нейронна мережа. Вона уважно розглядає зображення, намагаючись знайти на ньому знайомі об'єкти – людей, каски (чи їх відсутність), рукавиці, респіратори, окуляри. Кожен знайдений об'єкт отримує свою позначку: що це, де воно знаходиться на кадрі, і наскільки модель впевнена у своєму розпізнаванні.

Коли модель видасть свої результати, система переходить до "Аналізу результатів". На даному кроці програма ідентифікує сценарії, коли розпізнані об'єкти разом створюють ситуацію, що є сигналом про порушення безпеки – типовий випадок, коли на людині відсутня каска. Наступний блок, "Виявлено порушення?", він ставить пряме питання: чи є такі порушення? Якщо відповідь "Так", то всі деталі – час, тип порушення, сам кадр – записуються в "Логування порушення". Ця інформація зберігається локально, щоб потім можна було все переглянути. Якщо ж порушень немає ("Ні"), цей крок просто пропускається.

Незалежно від того, чи були зафіксовані порушення, далі йде

Відображення результатів в UI. Тут вже користувач бачить результат роботи: на екрані з'являється оброблений кадр, де всі розпізнані об'єкти обведені рамочками, підписані, вказана впевненість розпізнавання, а якщо були порушення – то й інформація про них.

Після відображення результатів цикл обробки підходить до блоку "Закрив програму?". Цей блок визначає подальший шлях виконання: у разі відсутності команди на вихід (гілка "Ні"), система повертається до блоку "Захоплення кадру" з веб-камери для продовження моніторингу. Якщо ж користувач вирішив завершити роботу (гілка "Так"), або робота була перервана через помилку, програма переходить до блоку "Кінець". Тут здійснюються необхідні завершальні операції: звільнення ресурсів, вимкнення камери та вивантаження моделі перед остаточним припиненням роботи системи.

3.2.3 Вибір апаратного забезпечення

Залізо для роботи системи підбиралось так, щоб вистачало потужності. Нічого зайвого, але й без слабких компонентів.

Зображення надходило з камери Anker PowerConf C200. Вона видає 2K (2560×1440), тож дрібні деталі було добре видно. Це важливо для точного спрацювання алгоритмів.

Процесор – AMD Ryzen 5 5600, шість ядер. Він відповідав за основні обчислення, не дуже складні. Справлявся без проблем.

Для нейромережі, а саме YOLOv8, використовувалась відеокарта RTX 3070 від NVIDIA. Без неї кадри оброблялись би значно повільніше.

Оперативної пам’яті – 32 ГБ. Це дало змогу нормально передавати дані між процесором, відеокартою і пам’яттю, навіть якщо обсяги були великі.

Загалом система працювала стабільно. Цього вистачало і для налаштування, і для тестів, і для показу, як все функціонує.

3.2.4 Вибір програмних засобів та технологій

Після того, як було проаналізовано можливі інструменти (про це йшлося в підрозділі 2.3 розділу 2) і визначено архітектуру системи (пункт 3.2.1, рис. 3.1), настав час остаточно визначитися з набором програмних інструментів. Мову програмування вибрано – Python. Його переваги: простий синтаксис, легкість підтримки коду. До того ж, Python має безліч бібліотек для обробки зображень, GUI та відео.

Для роботи з відеопотоком з веб-камери, аналізу кадрів, нанесення на них графічних елементів (рамок) та збереження зображень було використано бібліотеку OpenCV. Для самого розпізнавання об'єктів обрано YOLOv8 через бібліотеку Ultralytics – вона дозволяє досить швидко отримувати результати. А от щоб користувач міг все це бачити і взаємодіяти з програмою, графічний інтерфейс було зроблено на Tkinter, якому для роботи з картинками з OpenCV допомагала Pillow. Ну і для всяких технічних моментів – створення папок, запису логів у CSV-файл, роботи з датою і часом – використовуються стандартні модулі: os, csv, datetime, time. Все це було налаштовано та запускалося через Anaconda.

3.3. Розробка та підготовка набору даних (Dataset)

Щоб модель комп'ютерного зору добре вчилася, їй потрібен якісний та різноманітний набір даних, тобто датасет – це один з найголовніших моментів. Під час моєї роботи над цим проектом, підготовка даних проходила в кілька стадій: спочатку я працював з датасетами, які знайшов в інтернеті, а потім вже взявся за створення власного, спеціального набору зображень, щоб доналаштувати модель. Адже саме від того, яким буде цей фінальний набір даних, наскільки він різноманітний і точно розмічений, багато в чому залежатиме, чи зможе система потім ефективно "бачити" об'єкти та помічати порушення правил безпеки. Тому цьому етапу підготовки довелося приділити дуже пильну увагу.

3.3.1 Збір вихідних даних

Спочатку навчання моделі проводилось на вже готовому датасеті з інтернету. Загальна кількість зображень десь 11500. Дані вже містили анотації, що значно спростило роботу на початковому етапі навчання.

Зображення цього датасету були поділені на три категорії: навчальні – 9901, валідаційні – 1579, тестові – 216. Формат – стандартний для YOLO: окрема папка для кожного типу файлів, а також YAML-файл із описом класів.

Спершу все працювало добре. Але під час перевірки в реальному часі, через вебкамеру, з'явилися проблеми. Модель розпізнавала не всі об'єкти або робила це з похибками. Це стало помітно навіть на простих прикладах.

Щоб покращити результат, довелося створити власний датасет. Я зібрав 250 зображень вручну. Знімки містили ті об'єкти, які потрібно було визначати – наприклад, людина в шоломі, рукавицях, окулярах чи респіраторі.

Ці зображення я розбив на три частини: 176 – для навчання, 53 – для перевірки (валідації), ще 21 – для тестування. Анотації я робив сам – вручну, через спеціальний інструмент. Класи в новому наборі були синхронізовані з основним, однак частину я виключив, бо модель їх не використовувала.

3.3.2 Визначаення класів об'єктів та принципи формування вибірок

У початковому датасеті, завантаженому з відкритого джерела, було 13 класів. Серед них – елементи засобів індивідуального захисту (шолом, окуляри, рукавиці) та класи, що позначають їхню відсутність (No_Helmet, No_Glove тощо). Структура класів уже була задана у YAML-файлі, тож орієнтація була саме на неї під час подальшої роботи.

Після створення власного набору даних за допомогою Label Studio стались деякі несумісності. Зокрема, структура каталогу та формат міток відрізнялися від того, як це організовано у YOLO-датасетах. Також деякі класи, які були в основному датасеті я не використовував. Наприклад, об'єкти типу 'Safety_Harness', 'Shoe' та їхні відсутні варіанти (No_Shoe, No_Harness).

Щоб забезпечити повну сумісність із базовим датасетом, довелося вручну узгодити нумерацію класів, адаптувавши список таким чином, аби всі ідентифікатори збігалися з основним YAML-файлом. Зайві класи були виключені, а решта залишені у відповідній послідовності, навіть якщо деякі з них не зустрічались у новому наборі – це було важливо для збереження стабільності конфігурації під час донавчання.

Додатково, власноруч зібраний датасет не мав готового поділу на вибірки, тому цей етап був реалізований окремо. Для цього був написаний Python-скрипт, що автоматизує розподіл зображень і відповідних анотацій по папках train, valid і test, а також зберігає загальноприйняту структуру форматів YOLO. Поділ було реалізовано з урахуванням класової рівномірності

3.3.3 Анотування даних

Коли власний набір зображень уже був готовий, постало питання розмітки – тобто визначення, де саме на фото розташовані потрібні об’єкти. Щоб позначити на знімках шоломи, окуляри, рукавиці та інші елементи, було використано спеціальний інструмент Label Studio.

Усю розмітку довелося робити вручну. Зображення завантажувалися в проєкт, після чого на кожному окремо виділялися потрібні об’єкти прямокутною рамкою. На рисунку 3.2 показано приклад такого виділення об'єкта та присвоєння йому класу в інтерфейсі Label Studio. До кожної рамки прив’язувався певний клас зі списку, наприклад: Helmet, Gloves, Person, No_Helmet тощо. Назви класів відповідали тим, що вже були в основному датасеті – це було важливо для сумісності під час донавчання. Частина деяких класів не використовувалася – таких, як, наприклад, Shoes або Safety_Harness, – оскільки вони просто не зустрічались на зроблених фото.

Розмітка зайняла трохи часу, адже доводилося бути уважним: неправильно встановлена рамка або не той клас можуть вплинути на результат. Після того як усі об’єкти були промарковані, анотації експортувались у формат YOLO. Для кожного зображення створювався

окремий .txt-файл з координатами рамок та номерами класів.

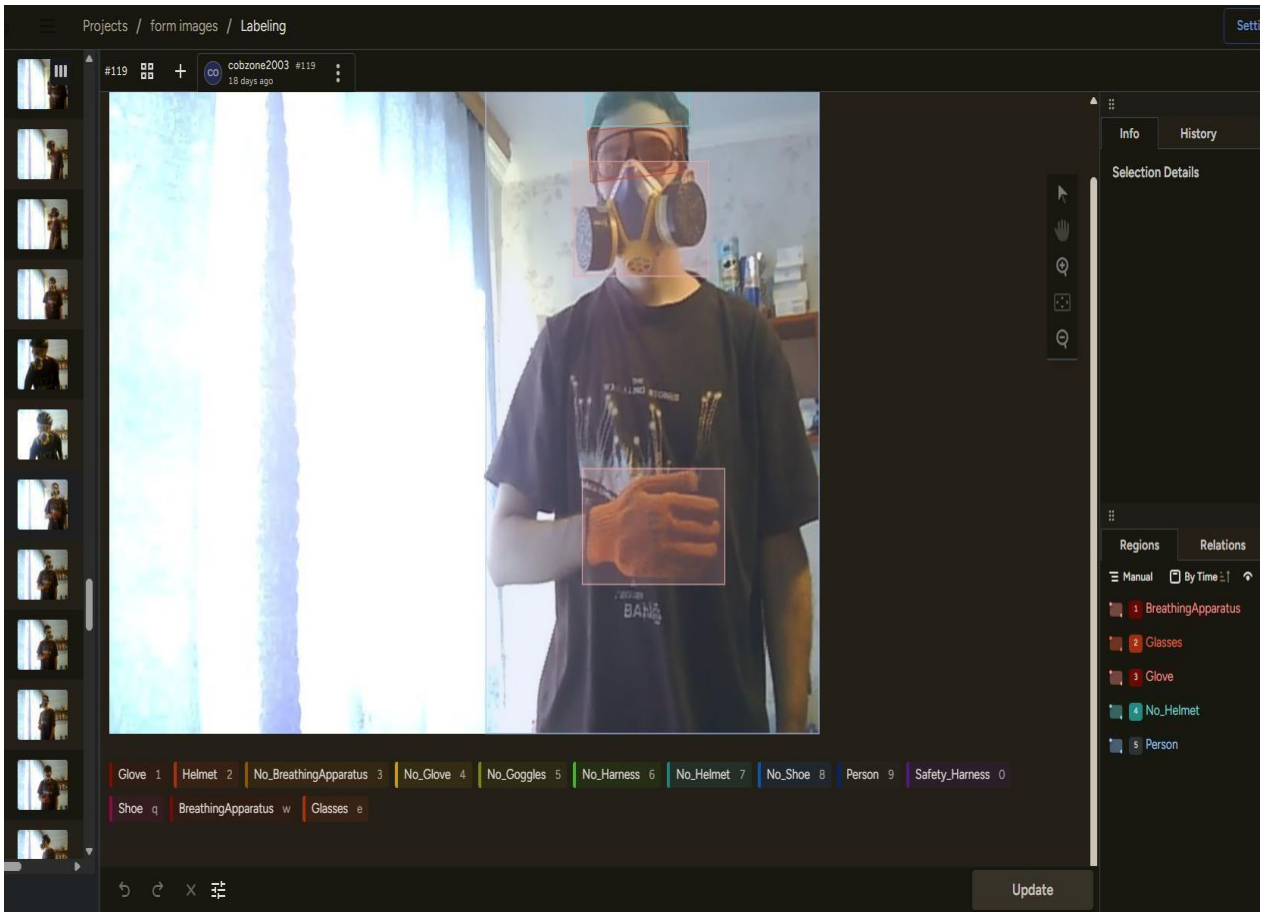


Рисунок 3.2 – Приклад процесу анотування зображень

Окрім цього, виникла потреба оновити нумерацію класів. У Label Studio класи йшли в одній послідовності, а у початковому датасеті – в іншій. Щоб не виникало плутанини при донавчанні, був використаний окремий скрипт на Python. Він автоматично проходив по кожному .txt-файлу, брав стару нумерацію та змінював її на ту, що відповідає структурі початкового набору. Це дозволило повністю синхронізувати класи між двома джерелами, навіть якщо вони мали іншу черговість.

Завдяки цьому етапу було отримано акуратно анотований і правильно структурований набір даних, який можна було без проблем використовувати для донавчання моделі.

3.3.4 Аугментація даних

Під час підготовки до етапу донавчання було вирішено активно використовувати різні види аугментацій, щоб надати моделі змогу краще орієнтуватися в різноманітних варіантах вхідних зображень. Усі ці перетворення виконувалися автоматично вбудованими засобами Ultralytics YOLO – нічого стороннього додавати не довелося.

Почалося все з кольорових змін. Значення hsv_h = 0,015 дозволяло змінювати відтінок, hsv_s = 0,7 – насиченість, а hsv_v = 0,5 – яскравість. Це допомогло створити варіації освітлення, щоб модель не «збивалася» при змінах умов зйомки. Далі я застосував translate = 0,2, що зміщує об'єкти по осях X і Y, і scale = 0,5 для зміни розміру об'єктів. Зсув контурів забезпечував параметр shear = 0,2.Також були активовані віддзеркалення: горизонтальні (fliplr = 0,5) і вертикальні (flipud = 0,1). Вони дозволяють моделі не прив’язуватися до одного напряму об’єкта.

Окремо варто згадати складні аугментації. Mosaic (mosaic = 1,0) поєднує чотири зображення в одне, створюючи новий контекст для об’єктів. Mixup (mixup = 0,2) змішує два зображення разом, а Copy-Paste (copy_paste = 0,1) додає об’єкти з одного кадру до іншого. Всі ці методи значно збагачують навчальний набір і допомагають моделі краще узагальнювати.

3.4. Розробка та навчання моделі розпізнавання об’єктів

Після завершення підготовки датасету почався етап практичної реалізації моделі розпізнавання. Він охоплює як технічну сторону запуску навчання, так і аналіз ефективності результатів. З огляду на вимоги до швидкості й точності, як було вже згадано раніше, вирішено використовувати YOLOv8. Подальший опис у цьому розділі демонструє, як поступово формувалася робоча модель – від підготовчих кроків до підбиття підсумкових показників.

3.4.1. Налаштування середовища для навчання

Перед початком роботи навчання моделі було налаштовано програмне середовище на базі Windows. Для управління пакетами та версіями була використана Anaconda. Це дало змогу зробити ізольоване середовище і допомогло уникнути конфлікти із сумісністю. Для написання коду було використано мову Python і використано такі бібліотеки та її необхідні залежності як: ultralytics, PyTorch (фреймворк глибокого навчання на якому базується YOLOv8), torchvision та інші. Вони допоможуть забезпечити роботу з YOLOv8 і обробку зображень. Були випробувані різні моделі YOLOv8, щоб знайти ту, яка може забезпечити як хорошу ефективність, так і помірне використання ресурсів. На рисунку 3.3 можна побачити встановлення основних бібліотек у консолі Anaconda.

Щоб навчання було більш ефективним та займало менше часу ніж із використанням лише центрального процесора було підключено дискретну відеокарту RTX 3070. Для її коректного підключення було вставлено драйвери, CUDA Toolkit і cuDNN. Це дозволило використовувати її повною мірою.

Також щоб не було ніяких проблем із несумісністю бібліотек, було додано такий рядок: os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE". Це забезпечить нам більшу стабільність роботи.

Рисунок 3.3 – Встановлення бібліотек у середовищі Anaconda

Отже, було створено стабільне і оптимізоване програмне середевище, що дозволило нам ефективно проводити навчання та донавчання обраної моделі.

3.4.2. Конфігурація параметрів навчання моделі YOLOv8

Далі відбувається перехід безпосередньо до навчання моделі. Цей перший етап навчання базується на готовому датасеті з інтернету (11500 зображень). Реалізація відбувалася за допомогою скрипта train2.py, який використовував попередньо навчену модель yolov8m.pt. Оскільки ця версія забезпечує збалансоване енергоспоживання, а також точність. Тому вона дала змогу ефективно працювати з досить великим датасетом і при цьому не перенавантажувала відеокарту. На рисунку 3.4 наведено приклад запуску скрипта навчання.

Рисунок 3.4 – Запуск навчання

Для того щоб навчання було не тільки ефективне, а й контрольоване, треба ретельно вибрати основні гіперпараметри. А саме такі показники як: кількість епох, розмір батчу, learning rate (швидкість навчання), розмір зображень, а також різні параметри, які впливають на стабільність та ефективність тренування. На рисунку 3.5 можна побачити яким чином параметри були визначені в скрипті.

```
    # Параметри навчання
    epochs = 200        # Збільшено кількість епох
    imgsz = 640
    batch = 16
    workers = 4
    device = 0
    lr0 = 0.001
    augment = True      # Увімкнено аугментації
    hsv_h = 0.01        # Налаштування аугментацій
    hsv_s = 0.7
    hsv_v = 0.4
    translate = 0.1
    scale = 0.5
    shear = 0.1
    flipud = 0.0
    fliplr = 0.5
    mosaic = 1.0
    mixup = 0.0
    copy_paste = 0.0
    patience = 15
```

Рисунок 3.5 – Параметри навчання

Параматери навчання:

– epochs = 200 – цей параметр означає скільки разів модель пройдеться по датасету для того щоб виявити основні закономірності і правильно адатпувати свої ваги;

– Imgsz = 640 – це розмір зображення, який забезпечує баланс між деталізацією і швидкістю обробки;

– batch = 16 – це оптимальний розмір пакету зображень для моєї відеокарти, який не викликає перенавантаження;

– workers = 4 – цей параметр дозволяє нам прискорити процес навчання. А саме завдяки цьому використовується 4 паралельні процеси (робітники) для попередньої обробки і завантаження батчів даних до відеокарти;

– device 0 – завдяки цьому параметру ми використовуємо нашу відеокарту, а це сильно прискорює навчання;

– augment = True – активуються аугментації зображень;

– hsv_h = 0,01, hsv_s = 0,7, hsv_v = 0,4 – налаштування колірних змін;

– translate = 0,1 scale = 0,5 shear = 0,1 fliplr = 0,5 – це прості зміни зображень: зсув, масштабування, нахил та відзеркалення;

– mosaic = 1,0 – змішування кількох зображень в одне;

– patience = 15 – зупиняє тренування, якщо модель не покращується протягом 15 епох.

Процес навчання тривав 147 епох, він зупинився завдяки параметру patience, оскільки суттєвий прогрес навчання закінчився. В результаті була отримана базова модель розпізнавання об’єктів. Інформацію про завершення навчання можемо побавити на рисунку 3.6.

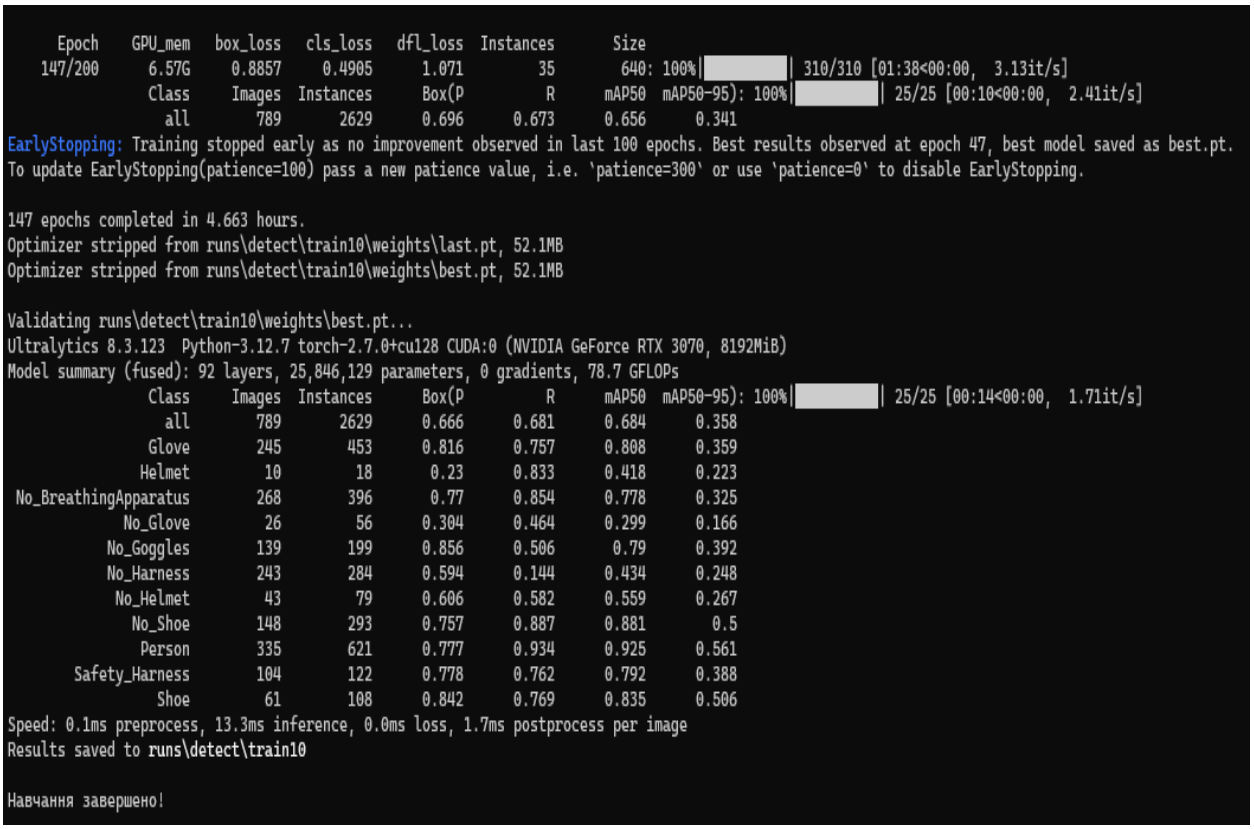


Рисунок 3.6 – Завершення початкового навчання

Щоб оцінити якість нашої базової моделі було проведено валідацію на окремій частині датасету, а саме 789 зображень, 2629 об’єктів. Оцінку моделі на валідаційному наборі можемо побачити на рисунку 3.7.

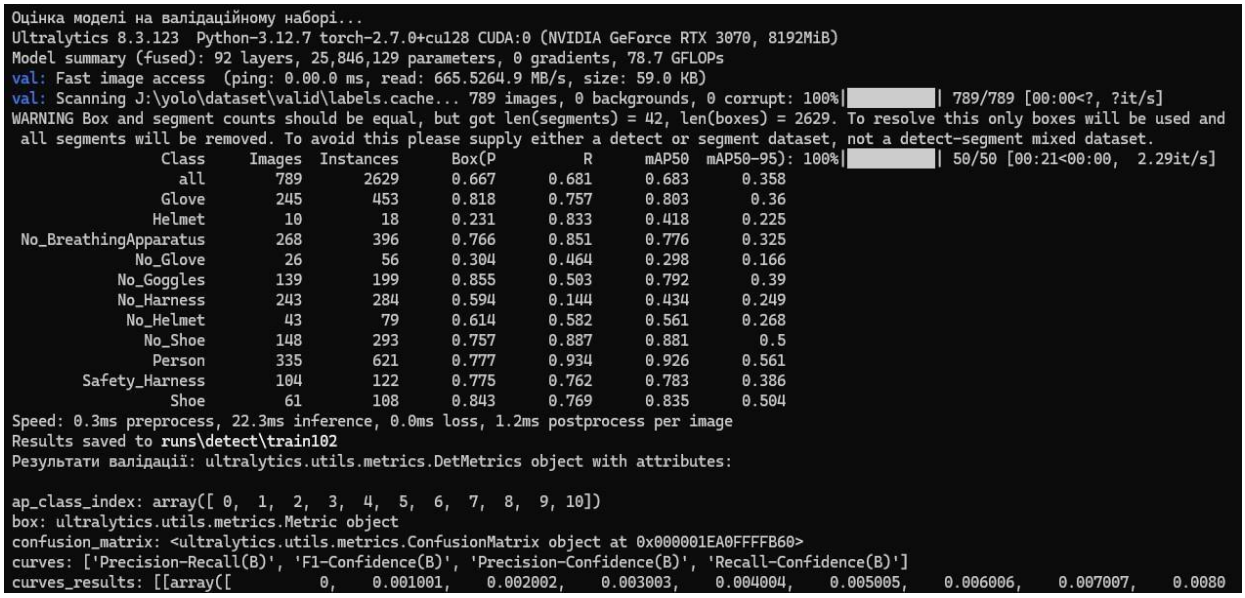


Рисунок 3.7 – Оцінка моделі на валідаційному наборі

Як можна побачити з рисунку 3.7, навчена модель показала такі результати: середня точність mAP50-95 була на рівні 0,358, а mAP50 з порогом 0,5 склала 0,683.

Для окремих класів дослідження були отримані такі результати(mAP50): 'Person' – 0,926; 'Helmet' – 0,418; 'No_Helmet' – 0,561; 'Glove' – 0,803; 'No_Glove' – 0,298; 'No_BreathingApparatus' – 0,776; 'No_Goggles' – 0,792. Можемо побачити ,що найкраще модель розпізнає людей.

Щоб краще оцінити результати навчання, було створено кілька графіків за допомогою бібліотеки ultralytics. Ці графіки допоможуть краще зрозуміти загальну ефективність моделі, а також її поведінку для конкретних класів.

На рисунку 3.8 зображена нормалізована матриця плутанини. За допомогою неї ми можемо побачити чи правильно модель класифікує об’єкти. З цього графіку видно, які об’єкти розпізнаються добре, а які плутає.

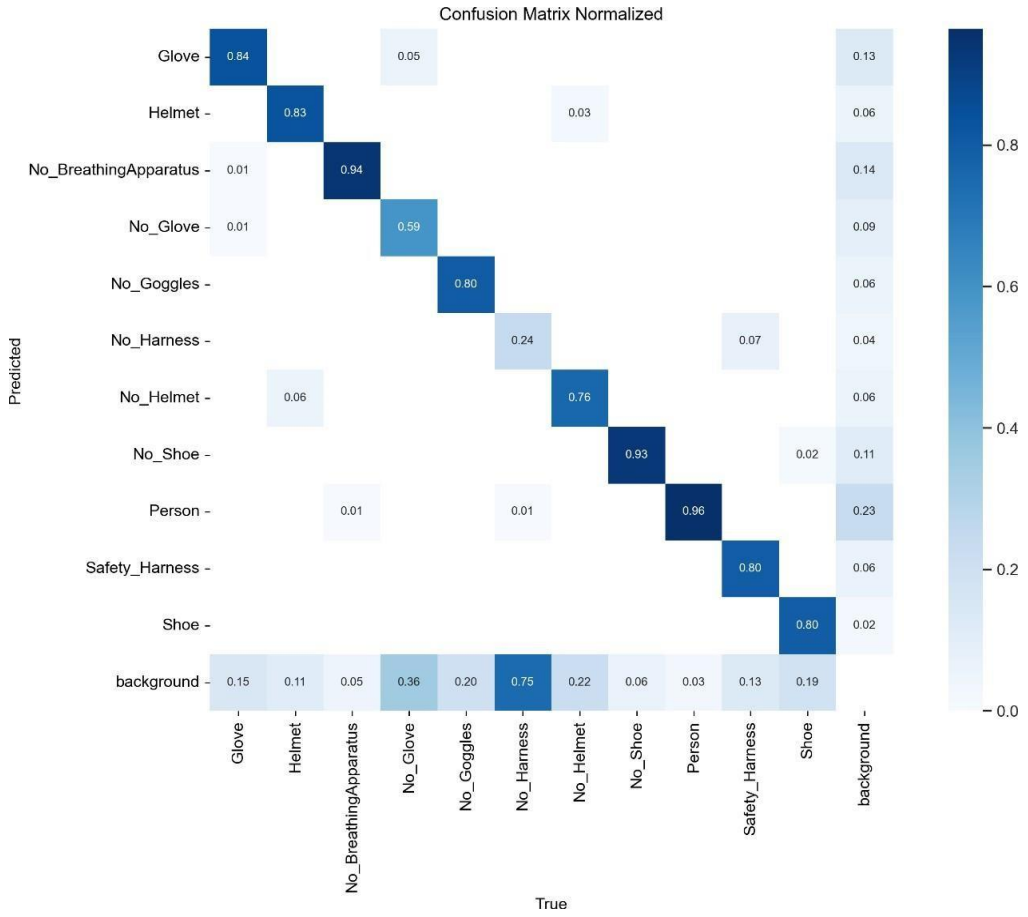


Рисунок 3.8 – Нормалізована матриця плутанини

З матриці бачимо , що модель найкраще розпізнає Person = 0,96, а також No_BreathingApparatus = 0,94 та No_Shoe = 0,93. Класи Glove = 0,84; Helmet = 0,83; No_Goggles = 0,80; No_Helmet = 0,76; Safety_Harness = 0,80; Shoe = 0,80 також показали досить гарні результати. Модель розпізнає їх з високою точністю, хоча певний потенціал для росту ще є. Це свідчить про те, що вона загалом навчилася розпізнавати ці об’єкти. Але от наприклад деякі класи, які позначають відсутність засобів захисту мають досить низькі показники: No_Herness = 0,24. Також можна побачити що на перетині background з класом No_Herness значення дорівнює 0,75. Це означає, що модель у 75%, коли на зображенні є лише фон, може знайти клас No_Herness. Схожа ситуація і з класом No_Glove, значення сягає 0,36. Такі показники вказують на схильність моделі бачити ці порушення там, де їх насправді немає, що могло б призвести

до значної кількості помилкових тривог у реальній системі.

Далі розглянемо графік PR_curve (рис. 3.9). На ньому можна побачити залежність точності (Precision) від повноти (Recall) для класів з різними значеннями впевненості (Confidence). Тобто, стає зрозуміло, як змінюється точність при різних рівнях впевненості та наскільки повно модель може виявити об’єкти.

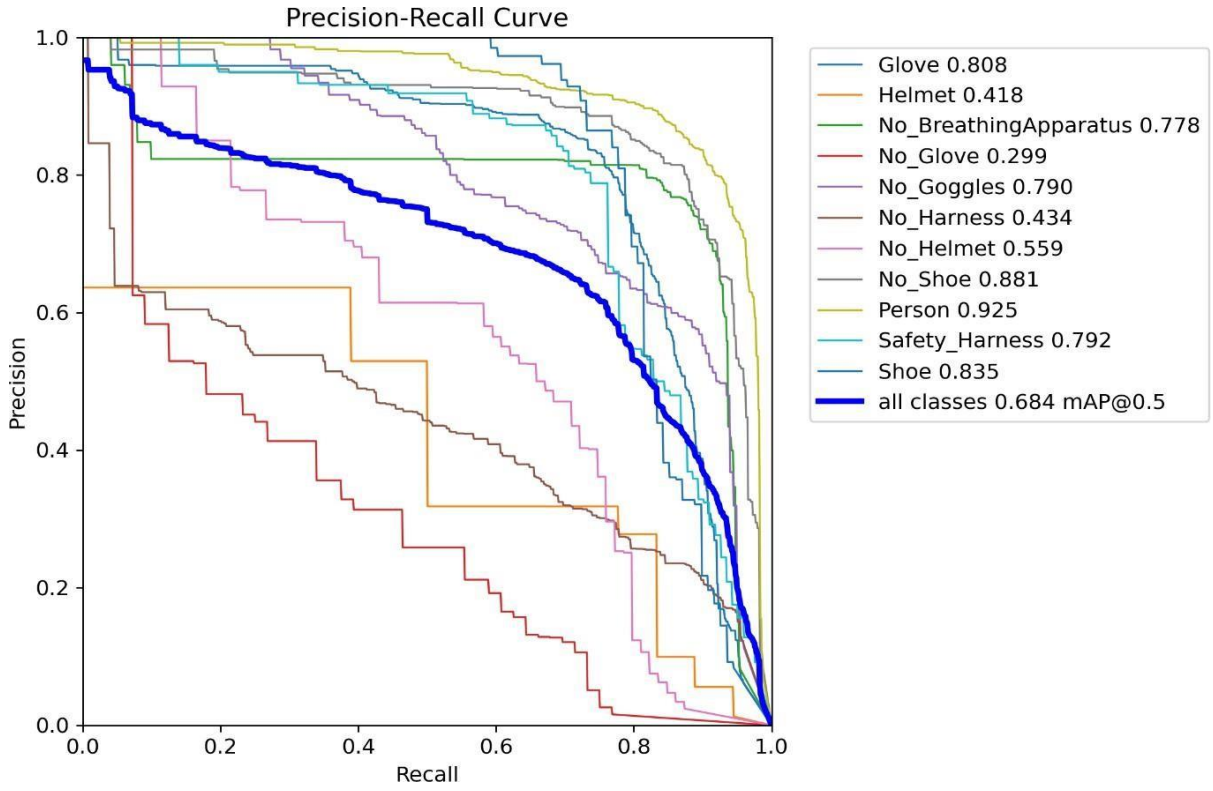


Рисунок 3.9 – Крива Precision-Recall (Точність-Повнота)

Згідно з результатами PR-кривої, модель на етапі початкового навчання показує mAP@0,5 на рівні 0,684. Серед усіх класів найкраще розпізнається Person - показник становить 0,925. Також досить високі результати у класів No Shoe (0,881), Shoe (0,835) та Glove (0,808). Це свідчить, що модель успішно ідентифікує як людей, так і наявність взуття й рукавиць.

У класів No_Goggles, Safety Harness і No BreathingApparatus значення перебувають на рівні близько 0,78-0,79, що можна вважати задовільним результатом. Але деякі класи показують гіршу якість: No_Helmet – 0,559,

тобто близько до порогового рівня, нижче якого модель вже не дає впевнених результатів.

Найгірше система справляється з виявленням "Helmet (0,418), No_Harness' (0,434) та особливо No_Glove – лише 0,299. Така низька ефективність, пов'язана з труднощами візуального відокремлення цих об'єктів, нестачею зображень у датасеті або надмірною схожістю з іншими класами. Про це також свідчать хибнопозитивні спрацьовування, виявлені у матриці плутанини.

Щоб краще зрозуміти, як відбувалося початкове навчання моделі на відкритому датасеті, було проаналізовано графіки змін основних метрик і втрат (рис. 3.10 – 3.11, згенеровані на основі results.csv, який містить повну статистику по 147 епохах). Візуалізація демонструє чітке зниження втрат для навчальної вибірки: значення train/box_loss зменшилося з 1,695 до 0,886, train/cls_loss – з 2,104 до 0,490, а train/dfl_loss – з 1,653 до 1,071. Це вказує на стабільне засвоєння модельних параметрів.

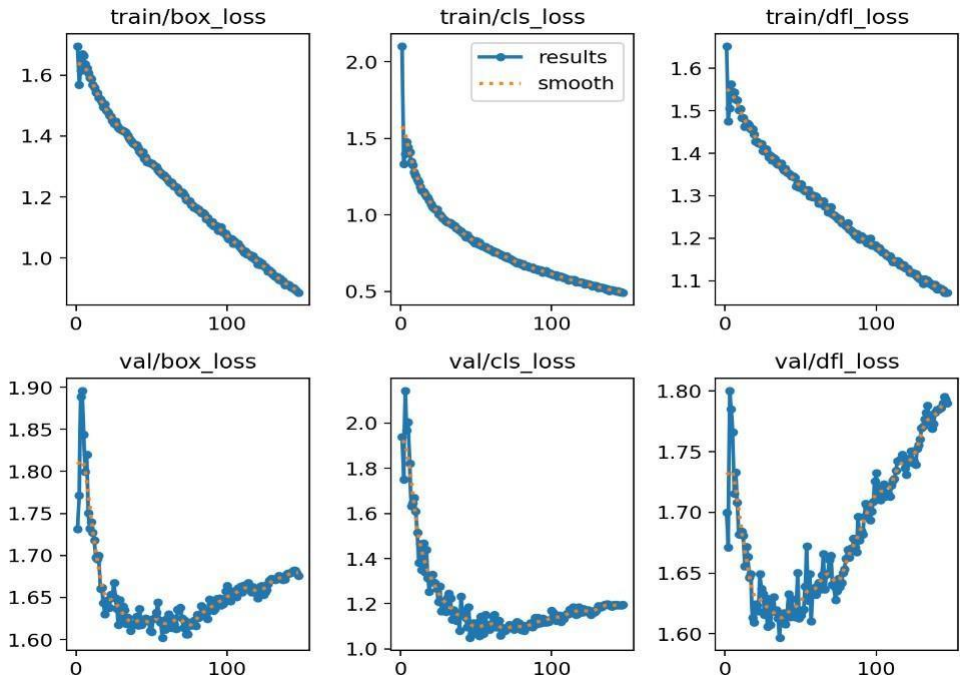


Рисунок 3.10 – Динаміка функцій втрат (loss)

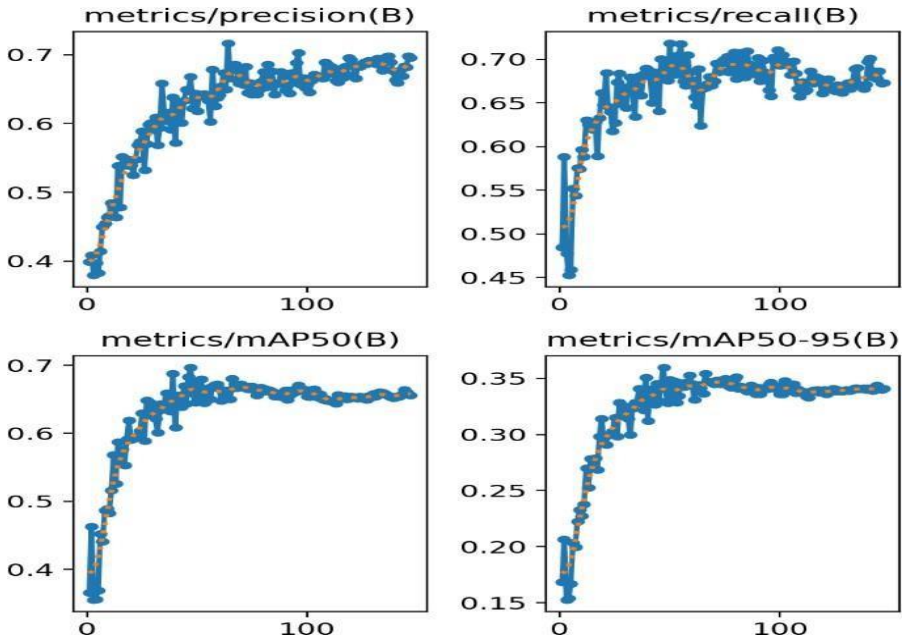


Рисунок 3.11 – Динаміка метрик якості (mAP, Precision, Recall)

На валідації ж ситуація була складнішою. Хоч спочатку втрати зменшувались, після приблизно 70-ї епохи темпи покращення сповільнились. Наприклад, val/cls_loss, досягнувши мінімуму (близько 1,05-1,06), знову почав рости й на останній епосі був уже 1,196. Такий розрив між тренувальними та валідаційними втратами зазвичай сигналізує про перенавчання.

Метрики якості, зокрема metrics/mAP50(B) Та metrics/mAP50-95(B), це підтверджують: вони сягнули максимуму ще на 47-й епосі (0,697 та 0,360 відповідно), після чого відзначалась стагнація або незначне погіршення. Саме версія моделі з 47-1 епохи була збережена як найоптимальніша. Хоча в налаштуваннях був заданий механізм ранньої зупинки (patience=15), навчання все ж продовжилось до 147-1 епохи – можливо, через особливості реалізації в бібліотеці.

Щоб краще оцінити ефективність моделі після початкового навчання, окрім числових метрик чи графіків, розглянемо і візуальні приклади з валідаційного набору. На рисунках 3.12 і 3.13 можна побачити зображення з реальними мітками (ground truth) та відповідними результатами, які повернула

модель.

Рисунок 3.12 – Приклад зображення з валідаційної вибірки з мітками

Рисунок 3.13 – Результат розпізнавання об’єктів моделлю

Порівнявши зображення, можна відзначити що модель успішно розпізнає об’єкт `Person` з високою впевненістю. А от наприклад `No_Herness` було пропущено кілька разів, ймовірно через неточну рамку. Також можна побачити, що в деяких випадках хоч модель і правильно ідентифікувала більшість класів, але впевненість занадто низька. Наприклад, клас `Glove` хоч і правильно ідентифікований в більшості випадків, але впевненість в цьому на деяких зображеннях лише 30%.

Отже, хоч графіки вказували на непоганий результат, після практичного тестування стало зрозуміло що потрібно ще доопрацювання. А саме цю готову модель я візьму за основу для подальшого донавчання.

3.4.3. Процес донавчання попередньо навченої моделі YOLOv8 на власному датасеті

Маючи базову модель, натреновану на загальному, було здійснено перехід до етапу донавчання, оскільки саме він дозволить покращити якість розпізнавання об’єктів. Для цього був зібраний власний набір зображень, до якого також були додані такі класи як `BreathingApparatus` і `Glasses`. Такий підхід сприяв помітному покращенню точності при розпізнаванні об’єктів, що мають найбільше практичне значення.

Для донавчання було використано найкращу версію моделі, яка отримана при минулому навчанні: `best.pt`. При додатковому навчанні моделі основні параметри не змінювались. Розмір зображення: imgsz = 640, розмір batch = 16.

Головною зміною стала заміна базоваої моделі `yolov8m` на `best.pt. Оскільки датасет за розміром був значно менший, тому для адаптації моделі до нових даних вистачить і 100 епох. Швидкість навчання задавав обережно. Початкове значення (lr0 = 0,001) вибрано тому, що модель вже навчена і занадто велике значення могло б погіршити раніше сформовані знання. Фінальне значення (lrf = 0,01) дає змогу поступово зменшувати темп навчання. Щоб перехід від початкового етапу навчання до кінцевого проходив більш

плавно було додане косинусне планування (cos_lr = True): замість різких падінь швидкості, значення плавно зменшувались.

Також як і в первинному навчанні, було використано аугментацію, оскільки датасет бев невеликим. Вона допомогла моделі бачити трохи змінені версії зображень, що допомголо при навчанні. Наприклад, параметри hsv_h = 0,015; hsv_s = 0,7; hsv_v = 0,5 змінювали відтінок, насиченість і яскравість, а translate = 0,2 та scale = 0,5 відповідали за зсув і масштаб. Нахили задавались через shear = 0,2, а повороти – через дзеркальне відображення (flipud = 0,1 та fliplr = 0,5). Також використовувався мозаїчний режим (mosaic = 1,0), що комбінує кілька зображень в одне, та параметри mixup = 0,2 і copy_paste = 0,1, які змішують об'єкти з різних прикладів – усе це розширює варіативність.

Щоб уникнути перенавчання, було включено ранню зупинку (patience = 15). Це дозволяло завершити процес, якщо протягом 15 епох не спостерігалось покращення на валідації.

Це навчання було також запущено на GPU RTX 3070. Хоч заплановано було 100 епох, але навчання закінчилося на 81 епосі, зявдяки `patience`. Отже модель вже на даному етапі перестала суттєво покращуватись, тому подальше навчання спричинило б перенавчання. Найкращий результат був досягнутий на 54 епосі, і саме цю версію моделі автоматично зберегло.

Для перевірки якості моделі, було проведено валідацію на наборі із 36 зображень та 177 об’єктів (рис. 3.14). В порівнянні з першою моделлю результати суттєво покращились. Середня точність за mAP50-95 склала 0,644, а за mAP50 – 0,874. Точність (precision) досягла 0,948, а повнота (recall) – 0,857. Такі показники підтверджують, що донавчання дало позивитивний результат.

```
Оцінка моделі на валідаційному наборі...
Ultralytics 8.3.123  Python-3.12.7 torch-2.7.0+cu128 CUDA:0 (NVIDIA GeForce RTX 3070, 8192MiB)
Model summary (fused): 72 layers, 11,130,615 parameters, 0 gradients, 28.5 GFLOPs
val: Fast image access  (ping: 0.00.0 ms, read: 644.540.9 MB/s, size: 58.8 KB)
val: Scanning J:\yolo4\dataset\split_dataset\valid\labels.cache... 36 images, 0 backgrounds, 0 corrupt: 100%| 36/36 [00:00<?, ?it/s]
WARNING cache='ram' may produce non-deterministic training results. Consider cache='disk' as a deterministic alternative if your disk space allows.
val: Caching images (0.0GB RAM): 100%| 36/36 [00:00<00:00, 2147.83it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%| 3/3 [00:01<00:00,  2.80it/s]
                   all         36        177      0.948      0.857      0.874      0.644
                 Glove         15         24      0.958      0.961      0.985      0.873
                Helmet         12         12      0.996          1      0.995      0.742
 No_BreathingApparatus         31         31      0.923          1      0.969      0.651
              No_Glove          9         10       0.97        0.9      0.986      0.683
           No_Goggles         23         23      0.958      0.997      0.988      0.523
            No_Helmet         22         22      0.795      0.864      0.861      0.533
              No_Shoe          1          1          1          0     0.0321     0.0321
               Person         35         37      0.972      0.949      0.958      0.892
    BreathingApparatus          4          4      0.909          1      0.995      0.822
              Glasses         13         13          1      0.903      0.973       0.69
Speed: 1.6ms preprocess, 22.7ms inference, 0.0ms loss, 0.9ms postprocess per image
Results saved to runs\detect\train52
```

Рисунок 3.14 – Оцінка моделі на валідаційному наборі

Також можна здійснити аналіз якості розпізнавання за класами. Наприклад, за метрикою mAP@0.5 було зафіксовано такі результати: `Helmet` і `BreathingApparatus` – по 0,995; `No_Glove` – 0,986; `No_Goggles` – 0,988; `Glove` – 0,985; `No_BreathingApparatus` – 0,969; Person – 0,958; Glasses – 0,973. Для класу `No_Helmet` якість теж була досить високою – 0,861.

Для більшого розуміння краще проаналізувати не тільки значення, а й візуальні графіки. На рисунку 3.15 можна побачити динаміку функцій втрат.

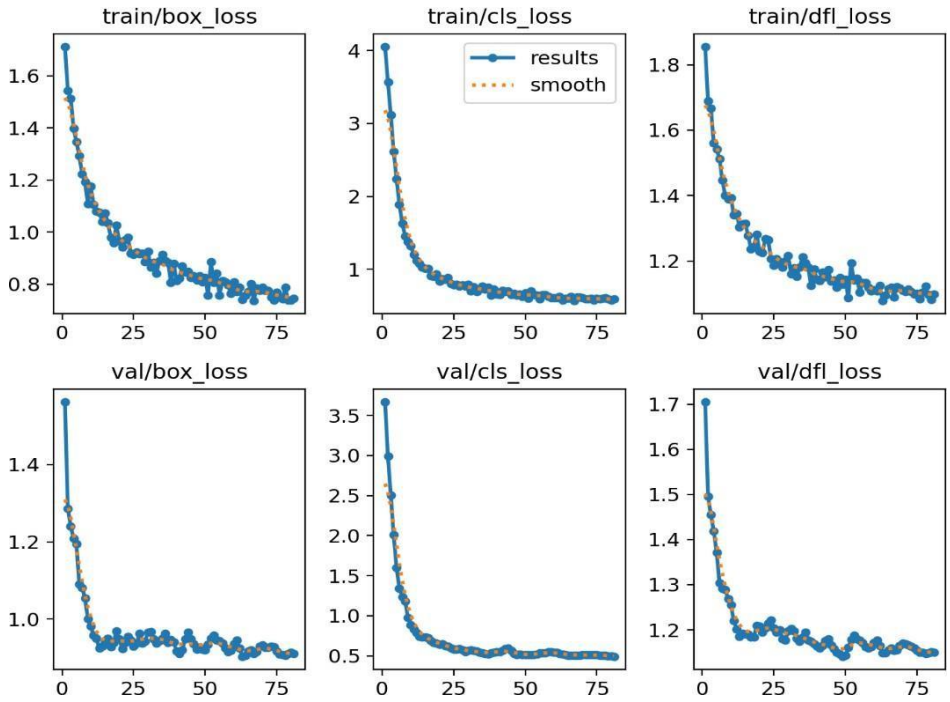


Рисунок 3.15 – Динаміка функцій втрат (loss)

Під час навчання модель поступово знижувала значення втрат, і це добре видно на графіках. Наприклад, помітно, як показник train/box_loss стартував приблизно з 1,8 і вже до 50-ї епохи впав до рівня 0,8 – 0,85. Найшвидше падіння спостерігалося на початку – особливо в перші 15 епох. Водночас train/cls_loss починався з досить високого значення, близько 4,0, але вже до 20-ї епохи знизився до 1,0, а далі ще повільніше впав до приблизно 0,6. Найстабільніша динаміка була у train/dfl_loss – там значення зменшувались від 1,8 до 1,0 без різких стрибків упродовж усього процесу.

На валідаційній вибірці спостерігалась подібна тенденція. Val/box_loss зменшився з 1,4 до 0,9, причому основне покращення припало на перші 20 епох, після чого залишалось відносно стабільним із незначними змінами. Val/cls_loss теж сильно знизився – з 3,5 до приблизно 0,5, хоча між 40-ю та 60-ю епохами помітні незначні сплески, мабуть пов’язані зі складнішими прикладами в даних. Розподіл втрат val/dfl_loss був найбільш стабільним: значення знизились із 1,7 до 1,1 без суттєвих стрибків, що вказує на хорошу здатність моделі узагальнювати розміри об’єктів на нових прикладах.

У сукупності ці результати свідчать про збалансоване навчання без вираженого перенавчання. Тобто модель ефективно навчається як локалізації об’єктів, так і їх класифікації на обох вибірках.

На рисунку 3.16 можна побачити зміну ключових метриків протягом 81 епохи. Точність (Precision) демонструє досить швидке зростання з 0,4 до 0,65 за 5 епох. Далі зростання стало більш помірним. Після 50-ї епохи залишилося стабільним в межах 0,92-0,96. Це означає, що модель рідко хибить. Повнота (recall) також досить швидко зростає, за 30 епох до 0,7. І потім вже помірно починаючи з 60-ї епохи дійшла до рівня 0,8-0,82. Отже модель досить точно виявляє значну кількість об’єктів, але не всі.

mAP@0.5 виріс до 0,85, що можна вважати хорошим результатом. А от більш жорсткий показник mAP@0,5-0,95 піднявся тільки до 0,60-0,65 – і це нормально, бо він вимагає кращої точності при побудові рамок.

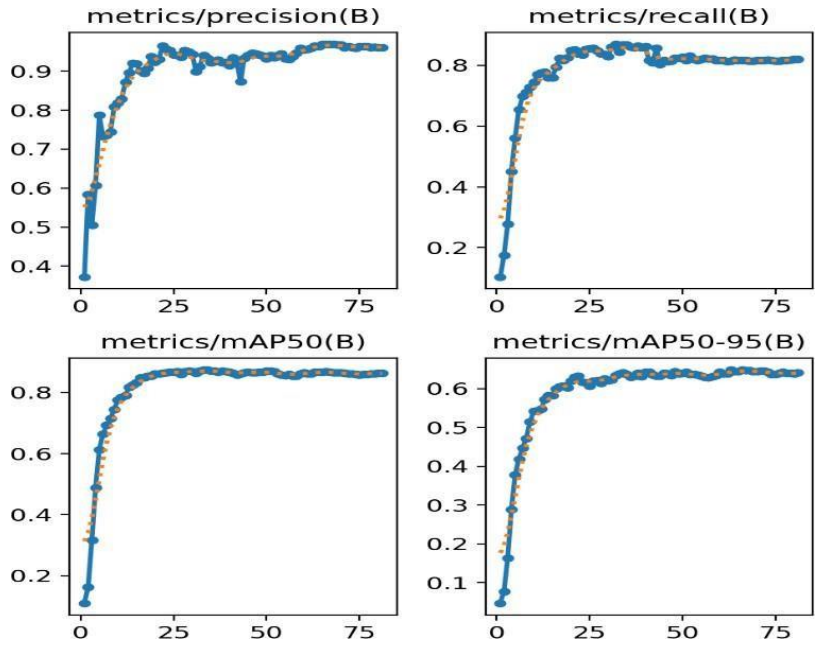


Рисунок 3.16 – Динаміка метрик якості (mAP, Precision, Recall)

Отже, динаміка цих показників вказує, що донавчання виявилося досить ефективним. Модель навчилась добре розпізнавати об’єкти й досягла стабільних значень якості.

Далі на рисунку 3.17 показана крива Precision-Recall.

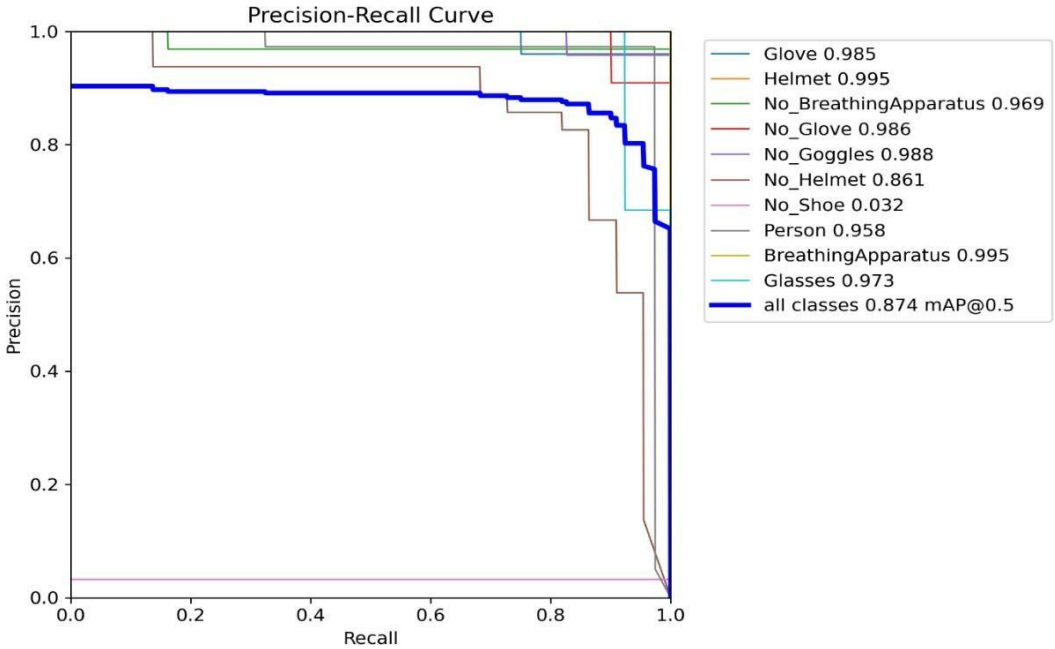


Рисунок 3.17 – Крива Precision-Recall (Точність-Повнота)

Після аналізу Precision-Recall графіка можна побачити, що модель дійсно добре справляється з основними класами. Наприклад, No_Goggles і BreathingApparatus показали майже ідеальні результати – обидва близько 0.988. Glasses також на високому рівні – 0,973. Це означає, що ці об'єкти модель майже не плутає і впевнено їх знаходить.

Дещо гірша ситуація з No_Helmet – тут значення 0,861, що вже не так вражає. Можливо, проблема в тому, що шолом не завжди чітко видно, або вони зливаються з фоном.

Особливо цікаво порівнювати з попередніми показниками. Раніше No_Helmet мав лише 0,559, а No_Glove ледве дотягував до 0,3. Тепер бачимо значне зростання, і це свідчить, що зміни в навчанні не були марними.

Загальний результат mAP@0,5 – 0,874 свідчить про високу якість розпізнавання, яка є достатньою для практичного використання моделі в реальних умовах. Далі на рисунку 3.18 можна побачити нормалізовану матрицю плутанини.

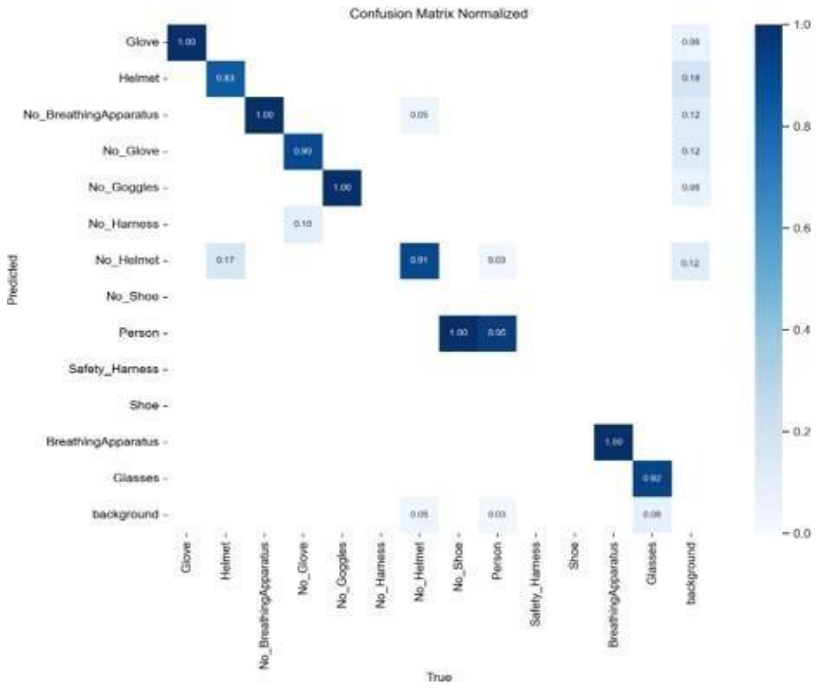


Рисунок 3.18 – Нормалізована матриця плутанини

Переглянувши матрицю плутанини, найбільше цікавила поведінка моделі щодо класу "background", бо саме тут часто виникають хибні спрацювання. Загалом ситуація непогана – більшість ділянок фону модель розпізнає коректно. Це свідчить, що в типових випадках вона добре відрізняє порожні області від об’єктів.

Що до помилок – певна кількість плутанини з фоном усе ж є. Наприклад, шолом (Helmet) класифікується як фон приблизно у 18% випадків, а класи без засобів захисту – No_BreathingApparatus, No_Glove та No_Helmet – мають близько 12% помилкового віднесення до background. Це не критично, але свідчить про те, що модель іноді “не помічає” деякі об’єкти або не розпізнає їх через часткове перекриття.

З іншого боку, класи з чіткою формою, як No_Goggles чи BreathingApparatus, практично не плутаються з фоном – модель виявляє їх дуже впевнено. І загалом, попри певні складнощі з деякими класами, серйозних систематичних помилок у розпізнаванні фону не було виявлено. Така точність цілком придатна для практичного використання.

Щоб краще зрозуміти, як саме модель розпізнає об'єкти, наведено кілька прикладів з реальними мітками та прогнозами (рис. 3.19 і 3.20).

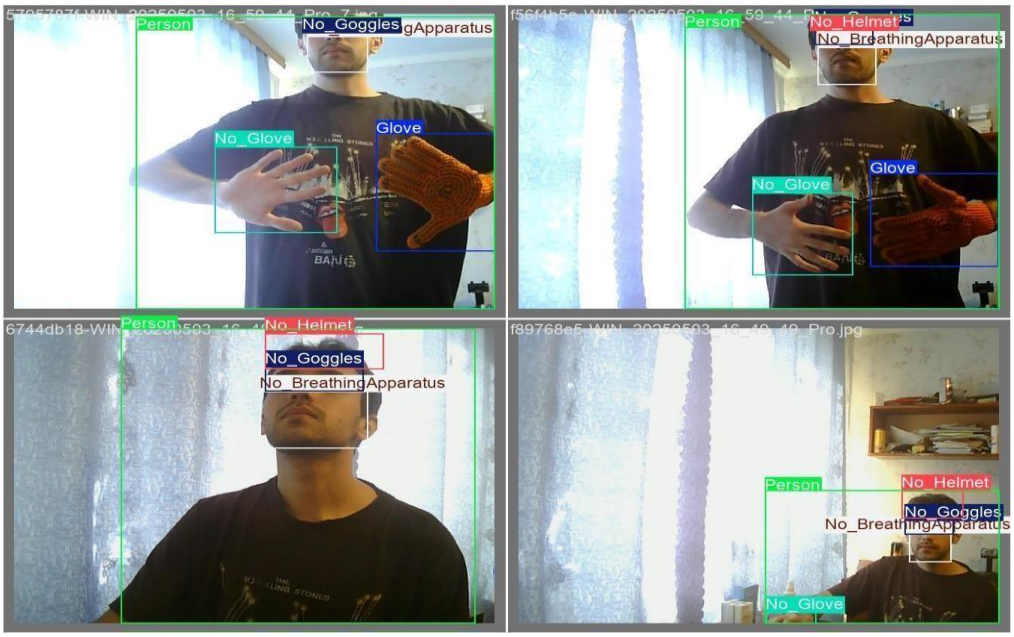


Рисунок 3.19 – Приклад зображення з валідаційної вибірки з мітками

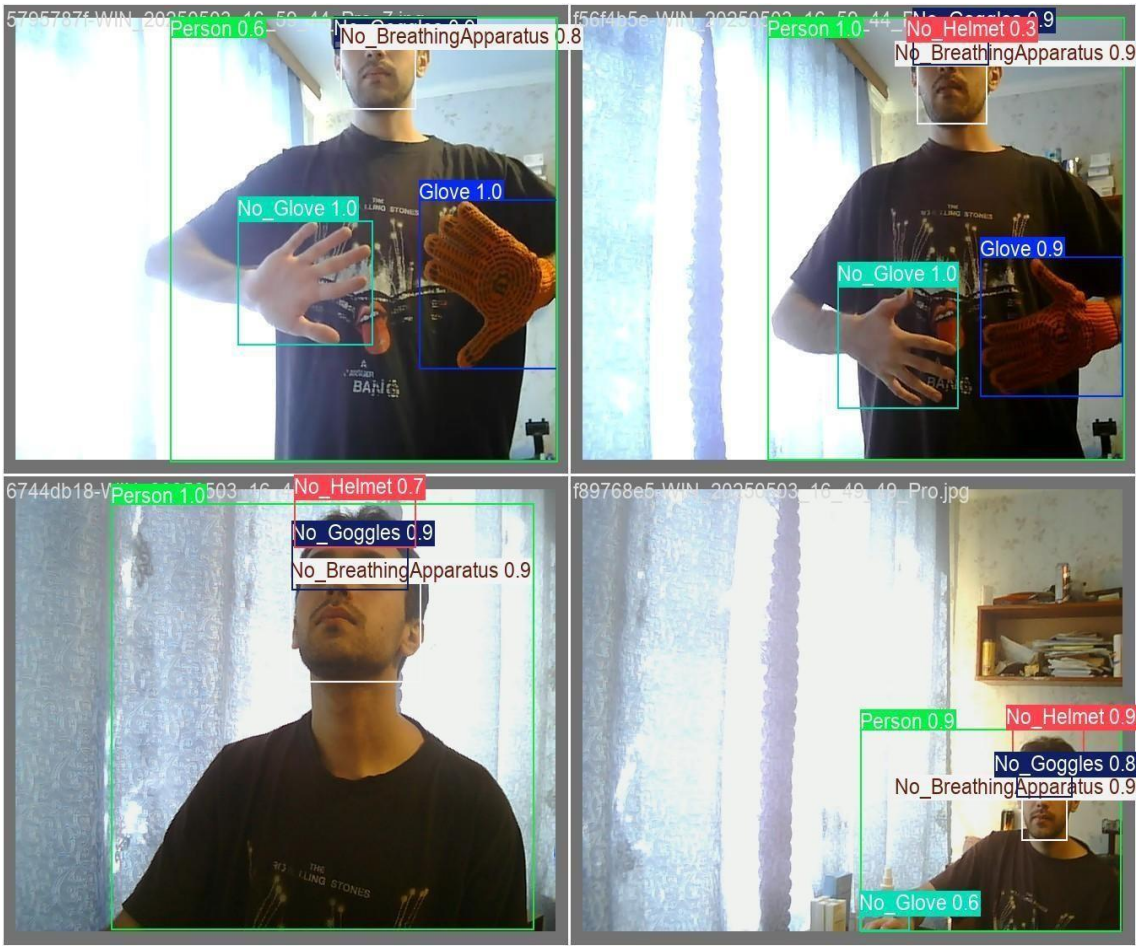


Рисунок 3.20 – Результат розпізнавання об’єктів моделлю

Поглянувши на справжні мітки (рис. 3.19) і порівнявши їх із прогнозами (рис. 3.20), видно, що модель непогано справляється зі своїм завданням. Вона впевнено визначає основні категорії, такі як Person і No_Helmet. Класи No_Goggles і No_BreathingApparatus також розпізнаються досить точно, а окуляри й рукавички знаходяться там, де потрібно. Хоч іноді трапляються невеликі неточності – рамки можуть трохи розходитися або впевненість для деяких об’єктів знижуватись, проте випадків пропусків чи зайвих детекцій небагато. Таким чином, можна сказати, що модель після донавчання помітно покращила свої результати.

3.5. Реалізація програмного модуля

3.5.1. Запуск та налаштування системи

Перш ніж система почне аналізувати відео з камери, потрібно підготувати кілька ключових компонентів. Найперше, що відбувається – це завантаження попередньо навченої моделі YOLOv8. Вона зберігається у файлі best.pt, і скрипт підтягує її за вказаним шляхом через змінну MODEL_PATH, щоб одразу бути готовою до розпізнавання об’єктів.

Паралельно перевіряється, чи існує папка для збереження знімків порушень. Якщо ні – автоматично створюється через os.makedirs(...), щоб не втратити жодного кадру з небезпечною ситуацією. Аналогічно з логуванням: якщо violations_log.csv ще не існує, програма додає файл і прописує в ньому заголовки колонок типу Time та Violations – це спрощує подальший аналіз.

Далі запускається відеозахоплення за допомогою cap = cv2.VideoCapture(0) – стандартного інструмента OpenCV для підключення камери. Саме звідси система отримує відео в реальному часі, яке потім буде аналізуватись на предмет порушень.

Окремо вже на старті ініціалізується основа графічного інтерфейсу, щоб користувач міг бачити, що система працює. Для цього використовується бібліотека Tkinter: створюється головне вікно root = tk.Tk(), на якому буде відеопотік (video_label) та індикатор статусу (status_label), наприклад "Стан: Перевірка…".

Усі ці дії виконуються до запуску основного циклу – так я можу бути впевнений, що все готове до роботи ще до появи інтерфейсу.

3.5.2. Виявлення об’єктів та логіка порушень

Після запуску програми запускається головна функція process_frame(), яка циклічно обробляє зображення з камери. Вона зчитує кадр через cap.read(), і якщо з якоїсь причини не вдається отримати зображення, обробка просто повторюється через короткий проміжок часу, задаючи новий виклик тієї ж

функції з затримкою root.after(10, process_frame).

Коли кадр успішно зчитано, він одразу передається в модель для обробки: results = model(frame, verbose=False)[0]. У результаті отримано список виявлених об'єктів із координатами (results.boxes.xyxy) і класами (results.boxes.cls). Назви всіх класів зберігаються у model. names.

Далі перевіряється, чи присутня на кадрі людина. Для цього перебираються всі знайдені об'єкти і шукається той, який має мітку "Person". Щойно такий об'єкт знаходиться, його координати (person_box) зберігаються окремо – надалі перевірка стосуватиметься саме цієї області.

Якщо людина на кадрі присутня, наступним кроком перевіряється, чи потрапляють якісь об’єкти з класів-порушень до її рамки. Список таких класів визначено на початку скрипта як CLASSES_VIOLATIONS, і до нього належать, наприклад, "No_Helmet", "No_Glove", "No_Goggles" та інші. Система знову проходить по всіх знайдених об’єктах, і якщо клас об’єкта належить до цього списку, перевіряється, чи перетинається його рамка з рамкою людини. Для цього порівнюються координати вручну: якщо хоча б частково обидва прямокутники накладаються – цей клас додається до списку порушень.

Після перевірки всі знайдені порушення записуються до множини violations_detected, а згодом – виводяться на екран статусу. Якщо є хоча б одне порушення, у полі статусу з’являється червоне повідомлення з їхнім переліком. Також, щоб уникнути дублювань, було встановлено інтервал збереження – нові порушення фіксуються лише тоді, коли від попереднього запису минуло не менше 5 секунд (SAVE_INTERVAL).

Якщо система фіксує порушення, зображення автоматично зберігається на диск через cv2.imwrite(...). До таблиці violations_log.csv при цьому додається новий запис – із зазначенням часу події (datetime.now().strftime(...)) і переліком того, що саме було виявлено. Усе це реалізовано просто всередині основного циклу обробки.

Як результат, кожен випадок порушення не тільки видно в реальному

часі, але й фіксується: зберігається кадр і записується вся інформація до лог-файлу.

3.5.3. Графічний інтерфейс користувача

Щоб користувач міг бачити результат роботи системи прямо під час її виконання, було створено простий інтерфейс на основі Tkinter. Основне вікно створюється один раз через виклик tk.Tk() і зберігається протягом усього часу роботи. У ньому розміщено два ключові елементи: рамку для зображення (Label) і ще одну нижче – для текстового повідомлення про статус безпеки.

Показ відеопотоку реалізується через те ж саме оновлення, яке відповідає за обробку кадрів. Тобто, коли process_frame() отримує новий кадр і модель накладає на нього результати (annotated_frame = results.plot()), цей кадр одразу перетворюється у формат, придатний для відображення у tkinter: спершу конвертую його в RGB, потім у PIL.Image, і вже тоді через ImageTk.PhotoImage() він потрапляє у віджет. Далі просто оновлюється зображення через video_label.configure(image=imgtk), а саму картинку зберігаємо у властивість video_label.imgtk, щоб уникнути проблем з відображенням.

Окремо нижче розміщується напис, який повідомляє, чи є якісь порушення. Коли система помічає порушення, на екрані одразу з’являється відповідний текст – червоний і помітний. Якщо ж усе в порядку, то замість нього висвічується просте “Все в нормі” зеленим кольором. Для оновлення використано status_label.config(...), і це працює швидко – ніяких миготінь чи зависань немає.

Було вирішено не додавати жодних кнопок – все починається автоматично. Після запуску скрипта відкривається вікно, і кадри починають оброблятися одразу. Оновлення йде в циклі через root.after(10, process_frame), і працює це доволі плавно.

На практиці такий підхід себе виправдав: увага не розсіюється, все просто й зрозуміло. Програма працювала стабільно й не навантажувала

систему навіть при тривалому використанні.

3.6. Перевірка роботи системи в реальному часі

Коли все було готове, було вирішено перевірити, як система поводиться під час роботи з відео в реальному часі. Для цього було запущено скрипт із підключеною вебкамерою – відеопотік з’являється одразу в окремому вікні, і модель починає аналіз кадрів просто на ходу.

На рисунку 3.21 видно момент, коли в кадрі людина без будь-яких засобів захисту. Програма правильно розпізнає всі порушення – показує, що немає каски (No_Helmet), окулярів (No_Goggles), респіратора (No_BreathingApparatus) та рукавиць (No_Glove). Всі ці порушення з’являються як окремі підписи на екрані.

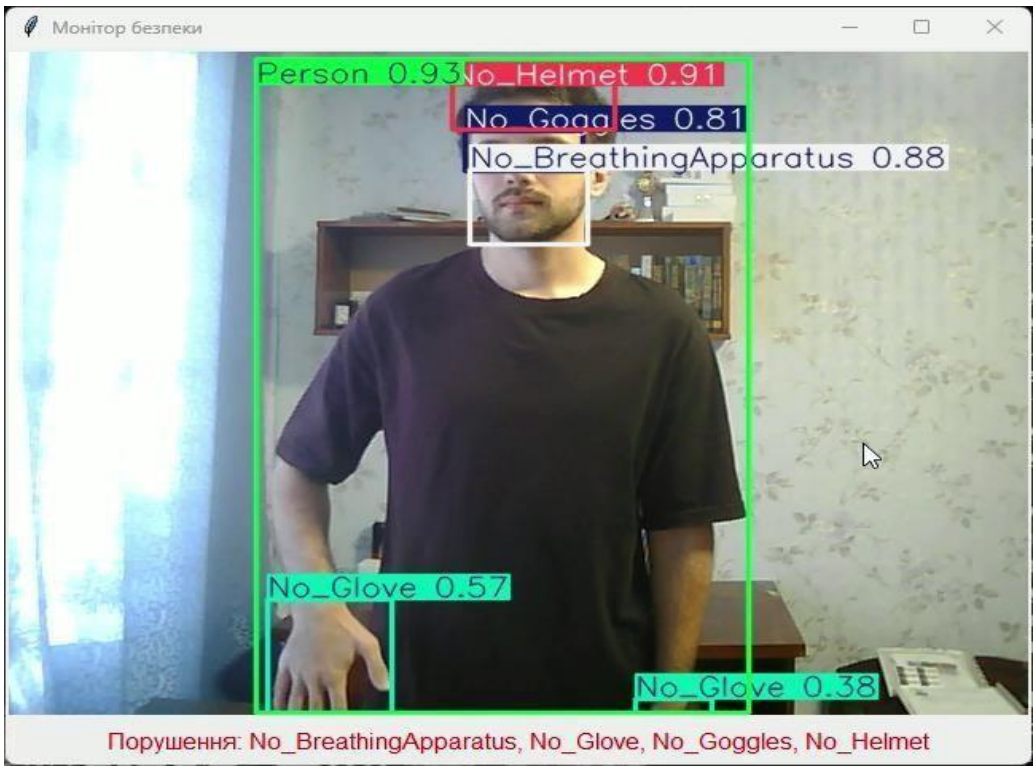


Рисунок 3.21 – Виявлено особу без ЗІЗ

Далі було перевірено, чи зможе модель виявити часткове порушення – коли більшість засобів захисту наявна, але відсутні деякі елементи. У цьому

прикладі було надягнуто каску та респіратор, однак спеціально не використовувалися захисні окуляри, а також одна рукавичка. На рисунку 3.22 показано результат: система зафіксувала порушення No_Goggles і No_Glove, що підтверджує її здатність точно виявляти навіть часткову відсутність необхідного спорядження.

Рисунок 3.22 – Приклад виявлення неповного комплекту ЗІЗ

Під час одного з тестів замість респіратора було навмисно використано звичайну медичну маску. На рисунку 3.23 показано приклад, коли система не зарахувала її як дихальний апарат і спрацювало попередження No_BreathingApparatus. Бували й інші випадки, коли маску все ж розпізнавалося неправильно, але загалом модель досить чітко відрізняє ці два об’єкти.

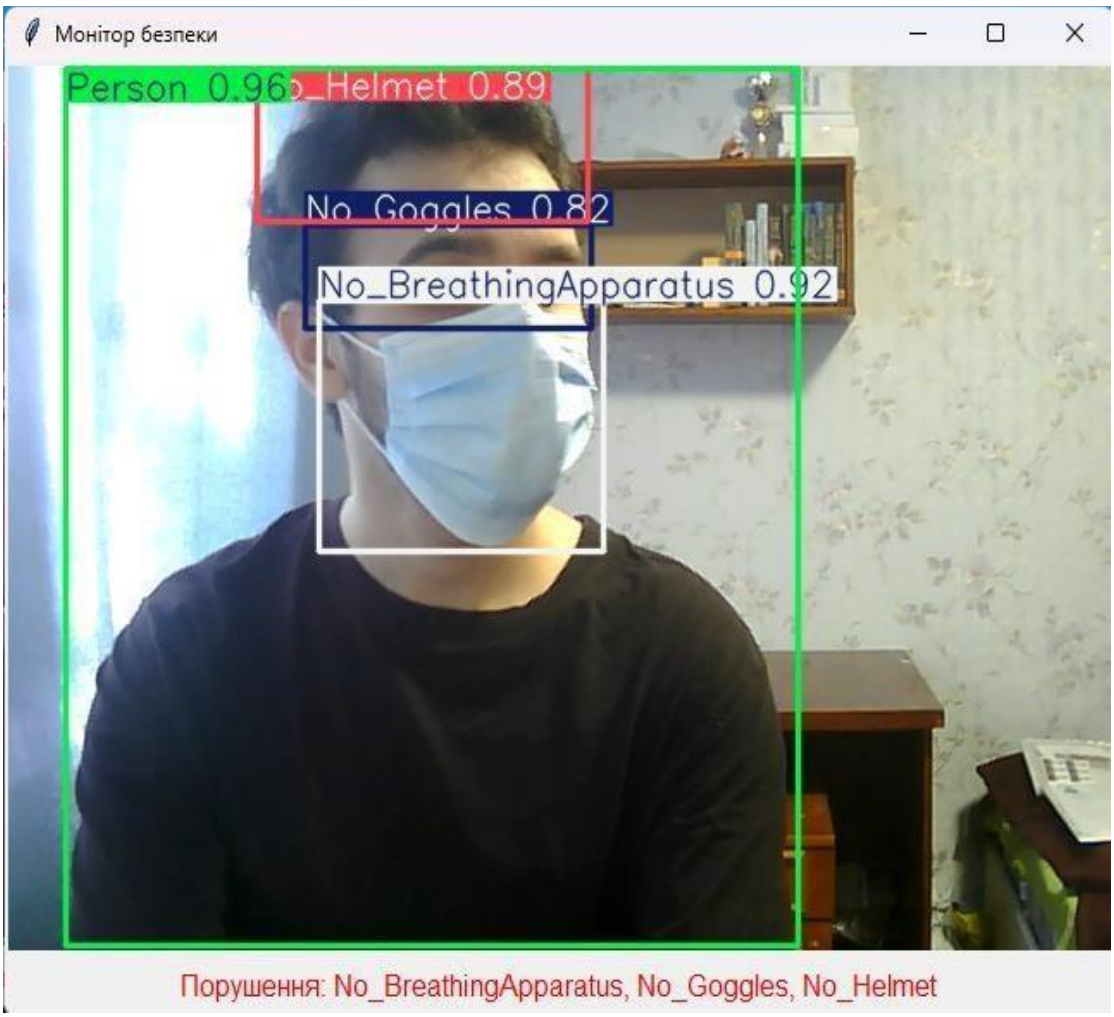


Рисунок 3.23 – Медична маска не розпізнана як респіратор, зафіксовано порушення

Далі на рисунку 3.24 можна побачити як система розпізнає об’єкти ЗІЗ у важчих умовах наближених до виробнчиого приміщення.

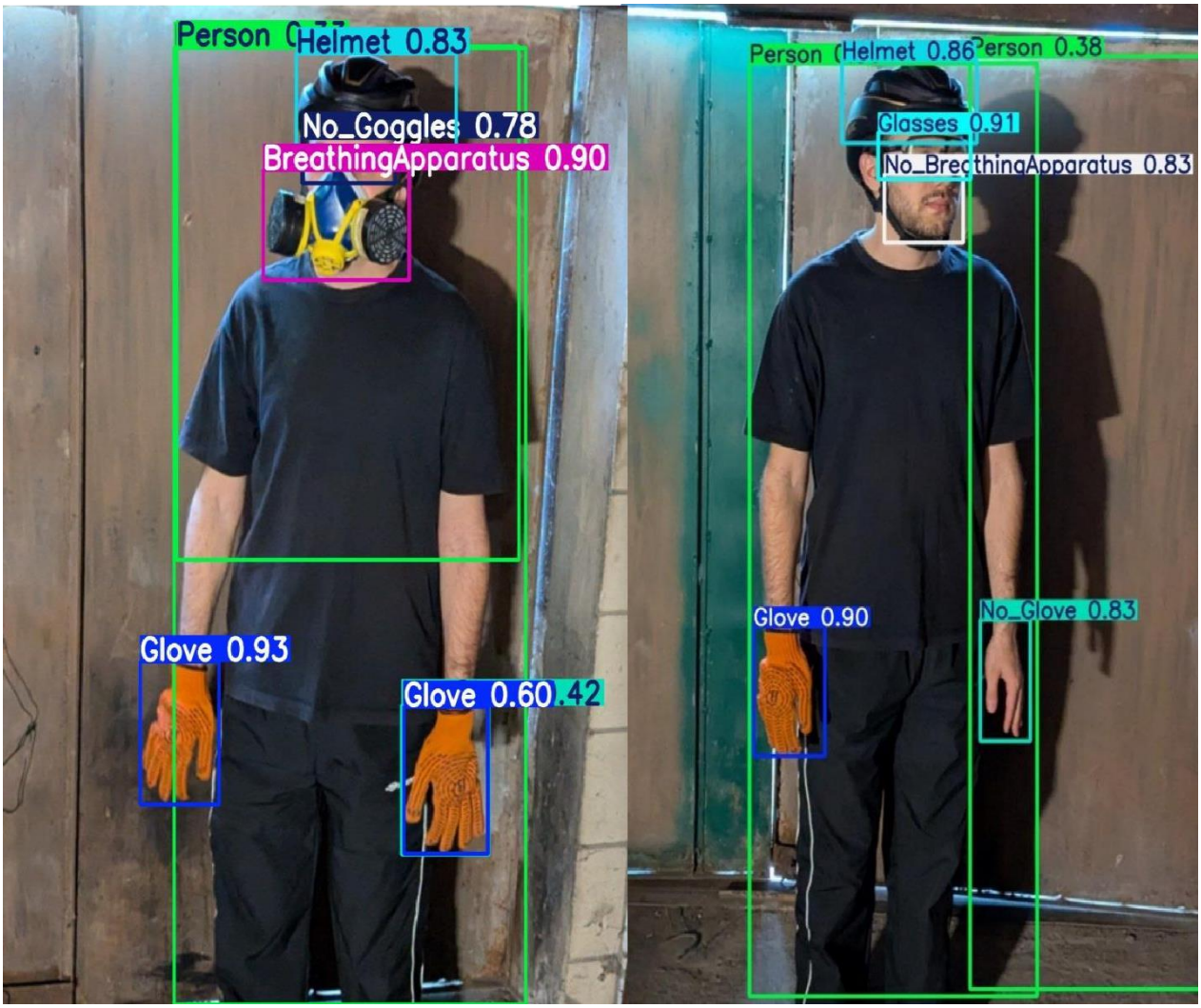


Рисунок 3.24 – Приклад розпізнавання ЗІЗ в важчих умовах

На цьому рисунку 3.24 можемо побачити декілька сценаріїв роботи розпізнавання. В першому випадку, на працівнику ідентифіковано: шолом (Helmet 0,83), респіратор (BreathingApparatus 0,90) та обидві рукавички (Glove 0,93 та Glove 0,60). Також було зафіксовано відсутністю окулярів (No_Goggles 0,78). В другому випадку, працівника було зафіксовано в шоломі (Helmet 0,86) та окулярах (Glasses 0,91), коректно визначивши наявність однієї рукавички (Glove 0,90) та відсутність іншої (No_Glove 0,83) і респіратора (No_BreathingApparatus 0,83). Отже система досить точно розпізнала об’єкти, хоча в другому випадку було помилково зафіксовано дві мітки Person для однієї людини, тож моделі ще є куди рости.

Крім виведення результатів на екран, система ще й автоматично записує всі порушення у файл .csv. Там зберігаються дата, час і тип кожного порушення. Це зручно, бо можна повернутись до історії та подивитися, коли і що було зафіксовано. На рисунку 3.24 видно приклад такого журналу.

```
486 20250529_194212,No_Helmet
487 20250529_194217,"No_BreathingApparatus, No_Helmet"
488 20250529_194222,"No_BreathingApparatus, No_Helmet"
489 20250529_194227,"No_BreathingApparatus, No_Goggles, No_Helmet"
490 20250529_194232,"No_BreathingApparatus, No_Goggles, No_Helmet"
491 20250529_194237,"No_BreathingApparatus, No_Goggles, No_Helmet"
492 20250529_194242,"No_BreathingApparatus, No_Goggles, No_Helmet"
493 20250529_194247,"No_BreathingApparatus, No_Goggles, No_Helmet"
494 20250529_194252,"No_BreathingApparatus, No_Goggles, No_Helmet"
495 20250529_194257,"No_BreathingApparatus, No_Goggles, No_Helmet"
496 20250529_194302,"No_BreathingApparatus, No_Goggles, No_Helmet"
497 20250529_194307,"No_Goggles, No_Helmet"
498 20250529_194312,"No_BreathingApparatus, No_Goggles, No_Helmet"
```

Рисунок 3.24 – Файл журналу порушень у форматі CSV

Ще один корисний момент – при кожному порушенні створюється скріншот з відео. Всі ці зображення зберігаються в окремій папці. Тобто є не лише текстовий запис, але й візуальне підтвердження. На рисунку 3.25 видно вміст цієї папки: там зберігаються знімки з назвами, в яких зашифрована дата і час події.

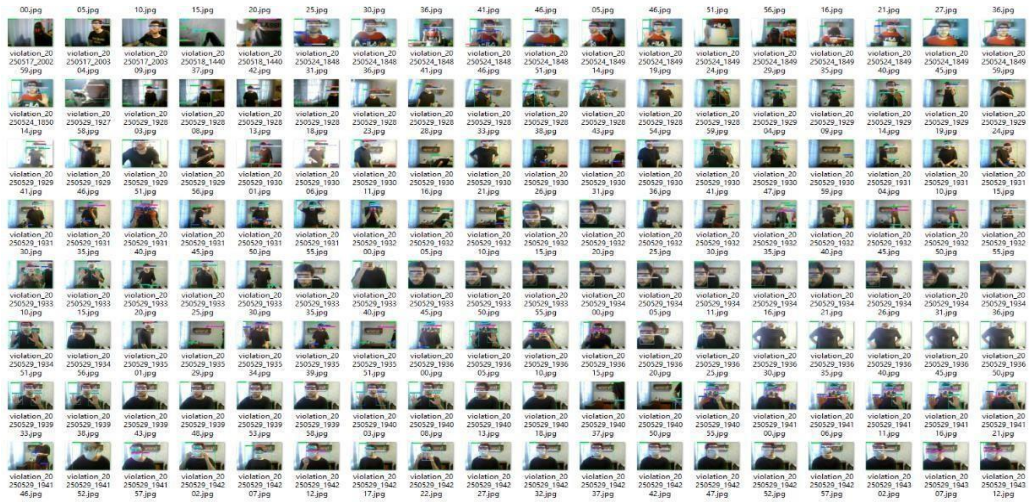


Рисунок 3.25 – Папка зі збереженими зображеннями моментів порушення

Загалом система показала себе з позитивного боку. Під час перевірки порушення фіксувались точно, об’єкти в більшості випадків розпізнавались без збоїв. Відеопотік оброблявся плавно, без помітних затримок. Водночас є ще потенціал для покращення – з часом модель можна допрацювати й зробити її ще надійнішою.

3.7. Розрахунок автоматичної реакції на порушення безпеки з урахуванням насичення

Системи, які реагують на якісь події, як от моя система моніторингу, що фіксує порушення безпеки (наприклад, відсутність шолома), часто мають справу не лише з тим, як сильно реагувати, але й з тим, що будь-яка реакція має свої межі. Наприклад, гучність сирени не може рости нескінченно. Цей ефект обмеження максимального вихідного сигналу називається насиченням. У цьому пункті я хочу розрахувати, як буде працювати простий пропорційний механізм реагування, якщо врахувати таке обмеження. Це допоможе зрозуміти, як поводяться елементи автоматики в більш реальних умовах.

В основі простого реагування лежить пропорційний елемент. Без урахування обмежень, його робота описується співвідношенням (3.1)

$$y_{ideal}(k) = K_p \cdot x(k). \quad (3.1)$$

Де:

– $x(k)$ – кількість порушень, яку система зафіксувала в момент k;

– $y_{ideal}(k)$ – розрахункова ("ідеальна") інтенсивність сигналу без обмежень

– K_p – коефіцієнт пропорційності, що визначає силу реакції на кожне порушення.

Але, якщо максимальна можлива інтенсивність сигналу обмежена величиною Y_{max} , то реальний вихідний сигнал $y(k)$ буде визначатися наступним чином:

$$y(k) = \begin{cases} K_p \cdot x(k), & \text{якщо } K_p \cdot x(k) < Y_{max} \\ Y_{max}, & \text{якщо } K_p \cdot x(k) \geq Y_{max} \end{cases}. \quad (3.2)$$

Це означає, що сигнал не може перевищити встановлену межу Y_{max}.

Для ілюстрації роботи розглянемо гіпотетичну послідовність кількості порушень x(k), зафіксованих системою моніторингу: $x(k)$={0,1,2,4,2,0,1}.

Приймемо максимальну інтенсивність сигналу $Y_{max} = 30$.

Випадок 1: Коефіцієнт пропорційності $K_p = 10$.

Спочатку розрахуємо ідеальну інтенсивність $y_{ideal}(k) = 10 \cdot x(k)$:

$$y_{ideal}(k) = \{0, 10, 20, 40, 20, 0, 10\}. \quad (3.3)$$

Далі застосуємо обмеження $Y_{max} = 30$ до кожного значення $y_{ideal}(k)$:

$y(0) = min(0; 30) = 0$;

$y(1) = min(10; 30) = 10$;

$y(2) = min(20; 30) = 20$;

$y(3) = min(40; 30) = 30$ (настало насичення);

$y(4) = min(20; 30) = 20$;

$y(5) = min(0; 30) = 0$;

$y(6) = min(10; 30) = 10$;

Таким чином, реальна інтенсивність сигналу з урахуванням насичення для $K_p = 10$: $y(k) = \{0, 10, 20, 30, 20, 0, 10\}$.

Випадок 2: збільшений коефіцієнт пропорційності $K_p = 25$.

Розрахуємо ідеальну інтенсивність $y_{ideal}(k) = 25 \cdot x(k)$:

$$y_{ideal}(k) = \{0, 25, 50, 100, 50, 0, 25\}. \quad (3.4)$$

Застосуємо обмеження $Y_{max} = 30$ до кожного значення $y_{ideal}(k)$:

$y(0) = min(0; 30) = 0;$

$y(1) = min(25; 30) = 25;$

$y(2) = min(50; 30) = 30;$ (насичення);

$y(3) = min(100; 30) = 30$ (насичення);

$y(4) = min(50; 30) = 30;$ (насичення);

$y(5) = min(0; 30) = 0;$

$y(6) = min(25; 30) = 25.$

Реальна інтенсивність сигналу з урахуванням насичення для $K_p = 10$: $y(k) = \{0, 25, 30, 30, 30, 0, 25\}$.

Отже, розрахунки показали, що ефект насичення справді обмежує сигнал на виході пропорційного елемента. Ми бачимо: навіть якщо зробити коефіцієнт K_p більшим, щоб отримати сильнішу відповідь, сигнал все одно не зможе перестрибнути через максимальну межу Y_{max}. Коли розробляють реальні системи автоматичного управління, важливо пам'ятати про такі нелінійні моменти, бо це допомагає точніше розуміти, як система себе поведе. Хоча це дослідження і нескладне, воно показує один з важливих практичних аспектів ТАУ.

3.8. Охорона праці

Розробка проводилась вдома, у звичайних умовах, без небезпечного обладнання. Проте робоче місце довелось організувати так, щоб не виникало проблем із перегрівом, надлишковим шумом чи нестабільною роботою системи. Під час навчання модель суттєво навантажувала відеокарту, тому доводилось постійно слідкувати за температурою через спеціальні утиліти. За потреби – відкривались вікна або ставив паузу, якщо навантаження ставало надто високим.

Живлення підключалось без додаткових фільтрів чи стабілізаторів. Одного разу через раптове вимкнення електроенергії втратились частина проміжних результатів, тому пізніше вирішив зберігати дані вручну через кожні кілька годин, щоб уникнути подібних ситуацій у майбутньому.

На рівні впровадження системи в реальних умовах є інші ризики. Наприклад, якщо хтось необережно розмістить камеру – вона може впасти або обірвати дроти. Тому обов’язково потрібно враховувати технічну сторону монтажу. Так само важливо, щоб система не займала багато ресурсів – інакше на слабких пк вона буде постійно зависати.

Ще один момент – обробка знімків. Система зберігає кадри, де видно порушення, і на деяких з них є обличчя людей. Тому файли зберігаються тільки локально, без підключення до хмарних сервісів. Так набагато безпечніше й простіше. Хоча на етапі розробки небезпеки практично не було, думати про охорону праці доводиться завжди.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було створено працюючу систему, яка здатна автоматично виявляти відсутність засобів індивідуального захисту у працівників на основі відеопотоку. Вона функціонує в режимі реального часу та може бути використана у виробничих умовах для підвищення рівня безпеки.

На першому етапі було розглянуто основи глибокого навчання, принципи роботи комп'ютерного зору та сучасні методи виявлення об'єктів. У тому числі проаналізовано підходи, що базуються на згорткових нейронних мережах, які найчастіше використовуються у задачах аналізу зображень.

Далі було проаналізовано популярні алгоритми виявлення об'єктів, серед яких порівнювалися одно- та двоступеневі архітектури. За результатами аналізу обґрунтовано вибір моделі YOLOv8 як найбільш відповідної для вирішення поставленої задачі. А саме завдяки швидкодії та достатньо високій точності.

Було підібрано необхідні програмні інструменти: мову програмування Python, бібліотеки OpenCV, Ultralytics, Tkinter та інші компоненти, потрібні для навчання моделі та розробки користувацького інтерфейсу.

Навчальний набір зображень створено вручну – зібрано 250 знімків, на яких вручну позначено класи: каска, рукавиці, окуляри, респіратор, а також ситуації, коли ці елементи відсутні. Такий підхід дозволив адаптувати модель до конкретної задачі.

Після навчання моделі YOLOv8 було досягнуто середньої точності mAP@0.5 на рівні 0.874, що підтвердило правильність налаштувань і якість підготовлених даних.

Окремо реалізовано програмний модуль, який уміє працювати з відео, виявляти на кадрах об'єкти, фіксувати порушення, зберігати відповідні знімки та логи у вигляді CSV-файлу. Програму протестовано на прикладах, близьких до реального виробничого середовища.

Підсумовуючи, можна відзначити, що запропонована система справді виконує свою функцію – автоматично визначає відсутність ЗІЗ на працівниках, фіксує це та зберігає докази. Такий підхід допомагає зменшити вплив людського фактору, підвищити контроль та вчасно реагувати на порушення.

У подальшому цю систему можна розширити – наприклад, додати нові класи об’єктів, покращити роботу в складніших умовах. Це дозволить зробити її ще кориснішою для підприємств, де питання безпеки має важливе значення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008-15. Документація. Звіти у сфері науки та техніки. структура та правила оформлення. Введ. 2015-06-22. К. Держстандарт України, 2017. 29 с.

2. Методичні вказівки з підготовки кваліфікаційної роботи бакалавра для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології освітньої програми «Системна інженерія» / Упоряд.: І.Ш. Невлюдов, О.М. Цимбал, О.В. Токарєва, А.І. Бронніков. Харків: ХНУРЕ, 2022. 66 с.

3. Lillian Davis. Object Recognition with Deep Neural Networks in Low-End Systems. WKU TopScholar Institutional Repository. URL: https://digitalcommons.wku.edu/cgi/viewcontent.cgi?article=2022&context=stu_hon_theses (date of access: 12.05.2025).

4. John Ajala. Object Detection and Recognition Using YOLO: Detect and Recognize URL(s) in an Image Scene. The Repository at St. Cloud State. URL: https://repository.stcloudstate.edu/csit_etds/37/ (date of access: 13.05.2025).

5. Mohamed Alaa Elabed. Understanding CNNs for Image Classification. URL: https://www.kaggle.com/discussions/getting-started/571488.

6. Olu-Ipinlaye O. Pooling In Convolutional Neural Networks. DigitalOcean | Cloud Infrastructure for Developers. URL: https://www.digitalocean.com/community/tutorials/pooling-in-convolutional-neural-networks (date of access: 13.05.2025).

7. CNN | Introduction to Pooling Layer - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/ (date of access: 13.05.2025).

8. Convolutional Neural Networks (CNN): Step 4 - Full Connection. SuperDataScience.
URL: https://www.superdatascience.com/blogs/convolutional-neural-networks-

cnn-step-4-full-connection (date of access: 14.05.2025).

9. Tarasiewicz J. 7 Problems You Can’t Ignore When Working on Object Detection. ATL | Translation and AI Services. URL: https://www.atltranslate.com/ai/blog/7-problems-in-object-detection-you-cant-ignore (date of access: 14.05.2025).

10. Md Faishal Rahaman. The Current Trends of Object Detection Algorithms: A Review. 2023.
URL: https://www.researchgate.net/publication/373392107_The_Current_Trends_of_Object_Detection_Algorithms_A_Review.

11. Anas AlhardiAnas Alhardi, Mustafa Ahmed Afeef. Object Detection Algorithms & Techniques.
URL: https://www.researchgate.net/publication/379120594_Object_Detection_Algorithms_Techniques (date of access: 01.05.2025).

12. Aziz, Lubna & Salam, Md Sah & Sheikh, Usman & Ayub, Sara. (2020). Exploring Deep Learning-Based Architecture, Strategies, Applications and Current Trends in Generic Object Detection: A Comprehensive Review. IEEE Access. 8. 170461-170495. 10.1109/ACCESS.2020.3021508.

13. A. B. Amjoud and M. Amrouch, "Object Detection Using Deep Learning, CNNs and Vision Transformers: A Review," in IEEE Access, vol. 11, pp. 35479-35516, 2023.
URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10098596&isnumber=10005208

14. Papers with Code - Faster R-CNN Explained. The latest in Machine Learning | Papers With Code.
URL: https://paperswithcode.com/method/faster-r-cnn (date of access: 08.05.2025).

15. Mask RCNN - HackMD. HackMD.
URL: https://hackmd.io/@imkushwaha/maskrcnn (дата звернення: 08.05.2025).

16. Object Detection and Distance Measurement in Teleoperation / A. Zhang et al. Machines. 2022. Vol. 10, no. 5. P. 402.
URL: https://doi.org/10.3390/machines10050402 (date of access: 11.05.2025).

17. Evaluation Method of Deep Learning-Based Embedded Systems for Traffic Sign Detection / M. Lopez-Montiel et al. IEEE Access. 2021. Vol. 9. P. 101217–101238. URL: https://doi.org/10.1109/access.2021.3097969 (date of access: 11.05.2025).