МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
20 травня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 – Комп'ютерних наук,
освітньо-наукової програми «Інформатика»
на тему: «Інформаційна технологія підтримки рятувальних операцій Державної служби України з надзвичайних ситуацій»
здобувача групи ІН.м-21н. Скорохода Андрія Анатолійовича

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

Андрій СКОРОХОД
(підпис)

Керівник,
к.ф.-м.н., доц. Оксана ШОВКОПЛЯС _____
(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук, освітньо-наукової програми «Інформатика»
здобувача групи ІН.м-21н Скорохода Андрія Анатолійовича

1. Тема роботи: «Інформаційна технологія підтримки рятувальних операцій Державної служби України з надзвичайних ситуацій»
затверджую наказом по СумДУ від «08» березня 2024 р. № _____
2. Термін здачі здобувачем кваліфікаційної роботи до 21 травня 2024 року
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Літературний огляд 2)Огляд наявних рішень 3) Постановка завдання 4) Вибір методів реалізації 5) Розробка та повторне навчання моделі 6) Розробка системи додатків типу "клієнт-сервер" для підтримки рятувальних операцій 8) Висновок .*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____»_____20___р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Огляд літератури*		
2	*Постановка задачі та формування завдань дослідження*		
3	*Опис методів реалізації*		
4	*Розробка моделей та системи додатків*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 62 стор., 25 рис., 1 додаток, 22 джерело.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботиє актуальною, оскільки розкриває проблему розпізнавання людей в реальному часі шляхом вдосконалення існуючої моделі машинного навчання.

Об’єкт дослідження – процес розпізнавання людей в реальному часі.

Мета роботи – розроблення прототипу системи розпізнавання людей в реальному часі з використанням моделей глибокого машинного навчання.

Методи дослідження – моделі глибокого машинного навчання, техніки перенавчання моделей.

Результати – проведений аналіз літературних джерел, моделей та методів машинного навчання, котрі надають змогу розпізнавати людей в реальному часі з високою точністю та швидкістю, та на основі цього створено прототип, котрий можна використати при імплементації системи в реальне життя.

НАУКА ПРО ДАНІ, КОМП’ЮТЕРНИЙ ЗІР, ГЛИБИННЕ НАВЧАННЯ, РОЗПІЗНАВАННЯ ОБ’ЄКТІВ, НАВЧАННЯ МОДЕЛІ, НЕЙРОННА МЕРЕЖА, PYTHON, YOLOV8

ЗМІСТ

ВСТУП

Актуальність.

В умовах постійних військових конфліктів цивільне населення завжди перебуває під загрозою. У 2022 році кількість смертельних випадків серед мирного населення нашої країни зросла вдвічі. Негативна тенденція збільшується до сьогодні. Навіть перебуваючи в укриттях, люди не вбережені від ризику зазнати нещасних випадків. Істотною перешкодою при проведенні рятувальних операцій стає відсутність точних даних про постраждалих. Одним із шляхів вирішення цієї проблеми є використання сучасних технологій розпізнавання образів для збору візуальної інформації про відвідувачів громадських місць.

Об’єкт дослідження. Процес розпізнавання людей в реальному часі.

Предмет дослідження. Методологія розпізнавання людей у реальному часі.

Гіпотеза. Перенавчивши модель, призначену для розпізнавання об’єктів, лише для розпізнавання людей, можна отримати приріст у точності та/або швидкості розпізнавання.

Наукова новизна. Модель, що описана у даній роботі, дозволяє розпізнавати людей у реальному часі, наприклад, з камер відеоспостереження у громадських місцях. Інформація, отримана в ході розпізнавання, може допомогти рятувальним службам зосередитися на пошуках конкретної кількості людей, що теоретично може зменшити рівень загрози здоров’ю та життю, а також полегшить процес ідентифікації постраждалих.

Апробація матеріалів роботи. Основні результати роботи оприлюднені на міжнародній науково-технічній конференції студентів та молодих вчених «Інформатика, математика, автоматика» (ІМА – 2024).

Структура. Дана робота складається зі вступу, аналізу предметної області, постановки задачі, опису проектування та розробки системи, висновків, списку використаних джерел та додатків.

Зв’язок роботи з науковою темою. Кваліфікаційна робота виконана на кафедрі комп’ютерних наук та пов’язана з виконанням науково-дослідної роботи № 0118U006971 «Методи, математичні моделі та інформаційні технології аналізу і синтезу інфокомунікаційних систем» (2018-2023).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Літературний огляд

Комп’ютерний зір

Комп’ютерний зір – це галузь дослідження, що утворилася внаслідок поєднання комп’ютерних наук, машинного навчання та деяких суміжних дисциплін, метою якої є створення методів та алгоритмів, котрі дають змогу комп’ютерам сприймати, розуміти та проводити обробку зображень та відео. Комп’ютерний зір використовується для багатьох різнопланових задач, від керування автомобілями до медичної діагностики [1].

Комп’ютерний зір, як поле досліджень, з’явився у 50-х роках минулого століття, коли науковці намагалися розробити комп’ютерні алгоритми аналізу візуальних даних. Першими результатами були прості алгоритми, що розпізнавали примітивні контури та форми. Наступні десятиліття, завдяки розвитку обчислювальної техніки, розвивався і комп’ютерний зір, що дало змогу застосувати його для розпізнавання облич та об’єктів.

Основні напрями досліджень:

- Розпізнавання облич. Алгоритми та методи розпізнавання облич дають змогу ідентифікувати та відслідковувати людей на фото чи відео. Активно застосовується при розробці систем безпеки, соціальних мереж та інших систем [1].
- Сегментація зображень. Даний напрям націлений на розділення зображень на зони, що містять окремі об’єкти. Це застосовується в медичній діагностиці та системах керування транспортними засобами [2].
- Розпізнавання об’єктів. Алгоритми та методи розпізнавання об’єктів дають змогу ідентифікувати та класифікувати об’єкти на фото чи відео. Використовується в багатьох сферах, серед яких промислова автоматизація та рятувальні системи.
- Глибинне навчання. Глибинні нейронні мережі справили великий вплив на розвиток комп’ютерного зору. Глибинне навчання дало змогу створювати системи, що здатні навчатися розпізнавати складні закономірності у великій кількості даних, значно підвищили точність розпізнавання та швидкість обробки даних.

Комп’ютерний зір – популярна та неймовірно корисна сфера, котра стала незамінною в багатьох аспектах повсякденного життя. Вона відкриває нові можливості, які раніше видавалися недосяжними, для медицини, безпеки, промисловості та ін. Якщо тенденція розвитку збережеться, ми станемо свідками ще більш революційних змін.

Розпізнавання об’єктів

Як було сказано раніше, однією із головних підгалузей комп’ютерного зору є розпізнавання об’єктів. Основні задачі розпізнавання об’єктів – це ідентифікація та класифікація різних об’єктів на цифрових візуальних даних.

Процес розпізнавання об’єктів складається з таких етапів:

- Попередня обробка. Перед процесами ідентифікації та класифікації іноді корисно здійснити попередню обробку, що може полягати, наприклад, у виправленні яскравості та подавленні шумів.
- Виявлення об’єктів. Для цього застосовуються або алгоритми розпізнавання контурів, однорідних текстур, або ж згортаючі нейронні мережі (CNN).
- Класифікація об’єктів. На цьому етапі система відносить виявлені об’єкти до того чи іншого класу, базуючись на характеристиках та паттернах об’єктів.
- Відстеження та післяобробка. Результат класифікації можна оптимізувати засобами післяобробки, для прикладу, злиттям подібних об’єктів чи відстороненням невдало класифікованих об’єктів. Для відеоданих можна застосувати трекінг об’єкту [3].

Традиційними (ранніми) підходами для розпізнавання об’єктів є суто інженерні методи, без складової штучного інтелекту. Серед них порівняння шаблонів та інші алгоритм низького рівня абстракції. Більш робастним підходом є імплементація систем на основі глибинних нейронних мереж, зокрема згортаючих нейронних мереж. Такі моделі вчаться розпізнавати складні закономірності, особливості чи викиди в даних.

Моделі глибокого навчання

Глибинне навчання – це підхід до машинного навчання, котрий застосовує глибокі нейронні мережі, що мають багато шарів, для обробки комплексних даних.

У сфері комп’ютерного зору глибинне навчання показало високу ефективність розпізнавання об’єктів, сегментації зображень та вирішення інших проблем. Однією з найбільш популярних архітектур для обробки візуальних даних є згортаючі нейронні мережі (CNN) [4].

Згортаючі нейронні мережі є видом глибоких нейронних мереж, розроблених спеціально для обробки відео та зображень. Вони застосовують унікальні операції та архітектури, завдяки чому ефективно виявляють закономірності та особливості у візуальних даних.

Головним елементом CNN є шар згортки, котрий застосовує фільтри для обробки малих областей зображень. Такий процес дає нейронній мережі змогу розпізнавати контури, текстури та інші візуальні патерни. Зачасту, після процесу згортки дані обробляються шарами зниження розмірності, серед яких, наприклад, шар максимального об’єднання. Це дозволяє зменшити розмір даних, поданих на вхід, але при цьому зберегти найголовніші особливості. За рахунок цього зменшуються використання обчислювальних ресурсів та ризик перенавчання. Після цих процесів дані проходять через один або декілька пов’язаних шарів для класифікації чи іншої задачі, залежно від завдання [5].

Розглянемо переваги CNN.

- Автоматичне виявлення особливостей. CNN здатні самостійно виявити комплексні особливості у візуальних даних, що дозволяє відмовитися від створення власних ручних методів виявлення.
- Масштабованість. CNN здатні обробляти великі обсяги вхідних даних, що дає їм змогу ефективно навчатися на обширних наборах зображень та відео.
- Висока точність. Через наявність глибоких шарів та великої кількості параметрів, CNN досягають високої точності в різних задачах комп’ютерного зору.

Тепер розглянемо застосування CNN.

- Розпізнавання об’єктів. CNN – це основа сьогоденних систем розпізнавання об’єктів. Їх застосування дає високу точність класифікації та ідентифікації об’єктів.
- Сегментація зображень. CNN мають широке застосування для сегментації зображень, зокрема розпізнавання та виділення окремих структур та/або об’єктів у

зображеннях.

- Розпізнавання облич. CNN використовують при розпізнавання та ідентифікації облич у візуальних даних, що є основоположним компонентом систем безпеки та соцмереж.
- Інші застосування. Окрім основних задач, CNN вирішує і інші завдання комп'ютерного зору, серед яких розпізнавання дій, аналіз зображень у медичній діагностиці та ін.

Згортаючі нейронні мережі – це потужний інструмент обробки візуальних даних. Через свою здатність розпізнавати комплексні патерни та особливості вони мають широке застосування у багатьох галузях життя, від ідентифікації особи до медичної діагностики. Розвиток цієї галузі продовжує відкривати нові можливості для машинного навчання та комп'ютерного зору.

Навчання моделі

Процес навчання глибоких нейронних мереж – це критично важливий етап створення моделі. Від якості та різноманітності датасету залежить якість та ефективність моделі.

Розглянемо основні етапи підготовки наборів даних.

- Збір даних. Для навчання моделей глибоких нейронних мереж важливо зібрати обширний набір даних, що містять приклади різних класів, котрі модель має навчитися розпізнавати. Даними можуть виступати зображення, відео та інші візуальні дані.
- Анотація даних. По завершенню збору набору, дані мають бути анотованими: кожен приклад повинен бути позначеним відповідно до поставленої задачі. Для прикладу, у завданні розпізнавання об'єктів кожне зображення містить мітки з категоріями та розташуваннями об'єктів.
- Розділення на навчальний та тестовий набори. Зазвичай наявний датасет розділяють на три набори: навчальний, валідаційний та тестовий. Навчальний датасет використовують для навчання моделі, валідаційний – для конфігурації, а тестовий – для оцінки ефективності, точності та продуктивності [6].

Тепер розглянемо етапи безпосередньо навчання моделі.

- Ініціалізація моделі. Для початку навчання модель CNN зазвичай ініціалізують рандомними значеннями вагів.
- Подача даних. При навчанні моделі подають дані з навчального датасету. Ці дані проходять через мережу, котра обчислює прогнозований результат.
- Обчислення втрат. Відразу після обробки даних, CNN обраховує втрати, котрі показують різницю між прогнозованими та фактичними значеннями. Функцію втрат застосовують при оцінці якості моделі.
- Оновлення ваг. Після вирахування втрат модель встановлює нові значення вагів за допомогою алгоритмів оптимізації, серед яких стохастичний градієнтний спуск, що дає змогу моделі навчитися виявляти патерни в даних.
- Повторення циклу. Подача даних, вирахування втрат та оновлення вагів повторюється протягом багатьох ітерацій – так званих “епох”, допоки модель не досягне очікуваного рівня ефективності.

Завершальними етапами навчання моделі є валідація та тестування. По завершенню навчання модель тестують на тестовому датасеті, аби оцінити її здатність до узагальнення та ефективність на нових, невідомих даних [7].

Отже, навчання глибоких нейронних мереж – це комплексний процес, що вимагає педантичного підходу до збору та анотації даних та конфігурації гіперпараметрів. Завдяки ефективному навчанню моделі CNN можуть досягати високої ефективності, надійності та точності в різних задачах комп’ютерного зору.

1.2 Огляд наявних рішень

Розпізнавання об’єктів – це одна з головних задач у сфері комп’ютерного зору. Воно має застосування в багатьох галузях, серед яких системи безпеки, автоматичне керування транспортними засобами, медична діагностика, автоматизація промислових процесів та ін. Сьогодні існує велика кількість алгоритмів, підходів, методів, архітектур та моделей для розпізнавання об’єктів, починаючи традиційними алгоритмами і закінчуючи сучасними моделями глибинного навчання [4].

Традиційні методи

- Методи порівняння шаблонів. Перші підходи до розпізнавання об’єктів були

основані на порівнянні зображень зі заздалегідь заданими шаблонами та/або зразками. Наприклад, алгоритм розпізнавання контурів, виявлення шаблонів та інші алгоритми намагалися виявити відповідність між частинами зображення і шаблоном [8].

- Гістограми орієнтацій градієнтів (HOG). Метод HOG аналізує розподілення градієнтів у зображенні та реалізовує гістограми орієнтацій градієнтів для кожної з областей поданого зображення. Дані гістограми застосовуються для виявлення та класифікації.

Сучасні моделі глибинного навчання

- Згортаючі нейронні мережі (CNN). CNN – це одна з найпопулярніших архітектур у сфері комп'ютерного зору. Вони застосовують шари згортки, щоб виявити локальні особливості у зображеннях, шари зниження розмірності, щоб зберегти найважливіші дані, і пов'язані шари, щоб класифікувати об'єкти.

- Регіональні згортаючі нейронні мережі (R-CNN). R-CNN застосовує підхід, за якого зображення ділиться на регіони, котрі можуть мати об'єкти. Кожен з регіонів аналізується окремо, цим самим дозволяючи мережі досягти високої точності. Через певний час даний підхід було оптимізовано, внаслідок чого з'явилися Faster R-CNN [9].

- CornerNet. CornerNet – це метод розпізнавання об'єктів, націлений на виявленні не центрів об'єктів, а їх кутів. Такий підхід дає моделі змогу виявляти об'єкти, базуючись на визначенні кутів контурів.

- Deformable Convolutional Networks (DCN). DCN – це поліпшення стандартних згорток CNN. Даний метод дає моделі змогу пристосовуватися до варіативності форм об'єктів завдяки зміщенню ядер згортки, що дає покращення ефективності та точності розпізнавання [4].

Готові моделі

- YOLO. Модель YOLO (You Only Look Once) – це один з найпопулярніших підходів до розпізнавання об'єктів. Дана модель працює з усіма об'єктами на зображенні одночасно, шляхом розділення зображення сіткою та аналізу кожної комірки. Модель є швидкісною та ефективною, що робить її гарним вибором для

розпізнавання візуальних даних у реальному часі. YOLO v8 є новітньою версією, котра відрізняється приростом точності, масштабованості та швидкості [10].

- Faster R-CNN. Faster R-CNN – одна з найбільш ефективних регіональних моделей згортаючих нейронних мереж. Вона застосовує регіональні пропозиції, щоб визначити потенційні зони з об’єктами, після чого вдається до аналізу за допомогою CNN. Faster R-CNN пропонує високі точність і швидкість [9].
- Mask R-CNN. Mask R-CNN – це вдосконалення Faster R-CNN, що, окрім виявлення та класифікації об’єктів, має в собі здатність до їх сегментації. Такий функціонал дає моделі змогу створювати точні маски для кожного з розпізнаних об’єктів, що має застосування в аналізі сцен та медичних зображень.
- SSD (Single Shot MultiBox Detector) – це одностадійна модель, що поєднує в собі декілька рівнів функціональних шарів, призначених для одночасної обробки візуальних даних та розпізнавання об’єктів різних розмірів. Даний підхід дозволяє моделі обробляти швидко, при цьому маючи прийнятну точність.

Отже, розпізнавання об’єктів – це неймовірно важливе завдання комп’ютерного зору. Традиційні методи корисні в певних ситуаціях, але сьогоденні моделі глибинного навчання (CNN, R-CNN, YOLO, SSD та ін.) надають високу точність та продуктивність. Такі підходи мають широке застосування в численних сферах, серед яких транспорт, медицина, безпека та безліч інших галузей, де необхідно розпізнавати об’єкти у візуальних даних.

1.3 Постановка завдання

Метою даного дипломного дослідження є розробка інтелектуальної технології глибинного машинного навчання для системи підтримки рятувальних операцій, головною задачею якої є розпізнавання людей в реальному часі, фіксування їх візуальних даних та їх кількості.

Робота складатиметься з:

- використання моделі розпізнавання об’єктів для розпізнавання людей у реальному часі;

- повторному тренуванні моделі розпізнавання об’єктів для розпізнавання виключно людей;
- порівняння результатів оригінальної моделі та натренованої повторно;
- розробки системи додатків типу клієнт-сервер для постачання даних про людей та їх кількість в реальному часі.

Для виконання поставленої мети потрібно вирішити наступні задачі:

1. Здійснити аналіз проблемної галузі.
2. Провести аналіз існуючих рішень розпізнавання об’єктів.
3. Обрати технологічний стек для проекту.
4. Реалізувати інтелектуальну систему.

Предметом дослідження є методологія розпізнавання людей у реальному часі.

Наукова новизна полягає в тому, що розроблена система дозволяє розпізнавати людей у реальному часі, наприклад, з камер відеоспостереження у громадських місцях. Інформація, отримана в ході розпізнавання, може допомогти рятувальним службам зосередитися на пошуках конкретної кількості людей, що теоретично може зменшити рівень загрози здоров’ю та життю, а також полегшить процес ідентифікації постраждалих.

2 ВИБІР МЕТОДІВ РЕАЛІЗАЦІЇ

2.1 Вибір мови програмування

Під час проектування системи розпізнавання одним з ключових аспектів був вибір мови програмування. Було обрано мову Python, оскільки вона є популярним вибором для систем на основі штучного інтелекту, зокрема систем розпізнавання об’єктів, і на те є ряд причин.

Python – це інтерпретована, високорівнева мова програмування, що набула надзвичайної популярності в галузі науки про дані та комп’ютерного зору [11]. Перш за все, Python має великий вибір бібліотек та фреймворків, шо спеціалізуються на штучному інтелекті. TensorFlow, PyTorch, Keras – це лише декілька з безлічі популярних інструментів, що дають змогу створювати та навчати нейронні мережі [12]. Крім рішень для штучного інтелекту, Python має безліч рішень для обробки та візуалізації даних, наприклад, supervision для простої та ефективної роботи з візуальними даними, fiftyone для роботи з датасетами, matplotlib та seaborn для побудування графіків. Також Python має простий та лаконічний синтаксис, що підвищує швидкість розробки та тестування алгоритмів штучного інтелекту. Це особливо актуально для розпізнавання об’єктів, оскільки швидка розробка прототипу дає змогу ефективно вирішувати поставлені завдання, швидко експериментувати та перевіряти різні підходи.

Python має активну спільноту, яка щодня розвиває та створює нові інструменти та надає підтримку. База знань Python включає в себе безліч відеоматеріалів, лекцій, книг, статей, курсів та інших ресурсів.

В комплекті з Python також поставляється менеджер пакетів та залежностей pip, що дозволяє легко встановлювати необхідні бібліотеки, та засіб створення віртуальних оточень virtualenv, котрий дозволяє створювати унікальні оточення для кожного проекту [13].

З огляду на вищевказані переваги мови Python, її було обрано одним з основних інструментів розробки системи розпізнавання.

2.2 Вибір моделі та навчального датасету

Під час вибору моделі та навчального набору даних для досліджуваної системи розпізнавання було враховано різні аспекти, серед яких точність, швидкість та доступність.

У якості моделі глибинного навчання було прийнято рішення використати модель YOLOv8 (You Only Look Once). Якщо бути точним, YOLO – це лише алгоритм, ключовою відмінністю якого є обробка всіх об'єктів на зображенні одночасно. Але пакет ultralytics поставляє готову модель, навчену на датасеті COCO128 на 100 навчальних епохах [14].

YOLO відомий своєю здатністю розпізнавання об'єктів у візуальних даних у реальному часі. Його використання дозволить створити систему, здатну надавати миттєву відповідь та обробляти відео з камер відеоспостереження в реальному часі.

Відомі також і інші версії YOLO. Серед популярних можна назвати YOLOv4, YOLOv5 та YOLOv8. Порівняно з YOLOv4 та YOLOv5, YOLOv8 надає покращену точність та швидкість, що робить її прекрасним вибором для досліджуваної системи. Восьма версія, завдяки пакету ultralytics, надає набагато легший спосіб використання моделі; доступ наданий у двох варіаціях – консольна утилита та бібліотека для інтеграції в систему мовою Python. На момент написання даної роботи, найновішою версією є YOLOv9, але YOLOv8 поки що має більш стабільні позиції та добре протестована спільнотою.

Для повторного навчання моделі YOLOv8 було обрано навчальний датасет COCO-2017 [15]. COCO-2017 містить великий розмаїтий набір зображень, але для даної роботи потрібні лише зображення людей. Можливість завантажити та/або відфільтрувати набір можна за допомогою бібліотеки fiftyone, що працює з офіційною бібліотекою CocoAPI. Ще одним фактором вибору стала висока доступність COCO-2017 для використання в навчальних цілях – варто лише вказати відповідне джерело.

Для порівняння оригінальної (на COCO128) та повторно натренованої (на COCO-2017) моделей було обрано датасет CrowdHuman [16]. Перш за все, він не пов'язаний з COCO і не містить спідьних даних. Правила доступності для

навчальних цілей такі ж, як і у COCO. Крім того, CrowdHuman концентрується на анотації людських об’єктів, що робить його прекрасною опцією для порівняння моделей, що розпізнають людей. Також CrowdHuman містить фото густонаселених зон, що дає змогу перевірити, наскільки ефективною буде модель в громадських місцях.

2.3 Вибір бібліотек

Під час вибору бібліотек для розробки системи розпізнавання об’єктів було враховано необхідність легкої роботи з візуальними даними, відеопотоком та навчальним набором.

FiftyOne – це робастна бібліотека для аналізу та візуалізації даних, що використовується для роботи з навчальними наборами, в тому числі і COCO [17]. Дана бібліотека дає змогу легко завантажувати та оброляти дані, відображати анотації на зображеннях, а також реалізовувати взаємодію з анотаціями. FiftyOne має великий набір інструментів для обробки даних та їх підготовки до навчання моделі.

Supervision (від roboflow) – це набір інструментів для роботи з візуальними даними. Бібліотека легка в користуванні та універсальна, надійно інтегрується з NumPy, Pandas, Keras, OpenCV та іншими популярними бібліотеками. Також вона дозволяє завантажувати датасети, позначати розпізнані об’єкти та рахувати їх кількість, що робить її незамінною для потреб розроблюваної системи [18].

Ultralytics – це компіляція інструментів для роботи з алгоритмами та моделями розпізнавання об’єктів, зокрема YOLO. Дана бібліотека має в собі найновіші версії YOLO, дає змогу легко навчати, валідувати, тестувати та використовувати її для розпізнавання об’єктів у реальному часі. Ultralytics надає зручний інтерфейс та надзвичайну продуктивність у виконанні задач розпізнавання об’єктів [14].

OpenCV – одна з найбіільш популярних бібліотек для обробки візуальних даних. Вона надає різні функції для зчитування, обробки, запису та відображення зображень та/або відео. OpenCV дає змогу легкої інтеракції з відеопотоком, виконання операції обробки зображень та аналізу відеоданих [19].

Кожна з вищезазначених бібліотек виконує важливі функції для розробки досліджуваної системи розпізнавання об’єктів, забезпечує широкий інструментарій та функціонал та зручний процес взаємодії з даними, відео та алгоритмами комп’ютерного зору.

2.4 Додаткові інструменти

Окрім мови програмування та супутніх бібліотек, при розробці системи розпізнавання також необхідний додатковий інструментарій, що сприятиме процесу розробки та підтримає організацію проекту.

Jupyter Notebool представляє собою інтерактивне середовище для програмування, дозволяє розробляти та запускати код в окремих комірках, а також відображати результати запуску коду прямо в ноутбуці. Jupyter Notebook спрощує есперементування з кодом та візуалізацію даних [20].

VS Code (Visual Studio Code) – це редактор коду, що належить компанії Microsoft. Він надає широкий вибір можливостей для роботи з Python. Також має розширену підтримку Jupyter Notebook та вбудовану підтримку git.

Git – це система контролю версій, котра дає змогу моніторингу змін в коді та спільно працювати над проектом. Git дозволяє зберігати хронологію змін та відновлювати попередні версії.

Virtualenv – засіб створення ізольованих віртуальних середовищ Python, що дозволяють інсталювати та використовувати різні версії пакетів для різних проектів. Застосування таких середовищ дає змогу уникнути конфліктів між пакетами різних версій та зберігати чистоту середовища [21].

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Модель розпізнавання об’єктів

Даний розділ присвячено основам використання попередньо навченої моделі YOLOv8 для створення системи розпізнавання об’єктів, її особливості та чому з’явилася ідея повторного навчання моделі. Дослідницька робота виконана у середовищі Google Colab.

Перш за все, після створення нового ноутбуку, у налаштуваннях записника варто переключитися зі стандартного режиму, що опирається на CPU, на режим, основним ресурсом якого є GPU (рис. 3.1).

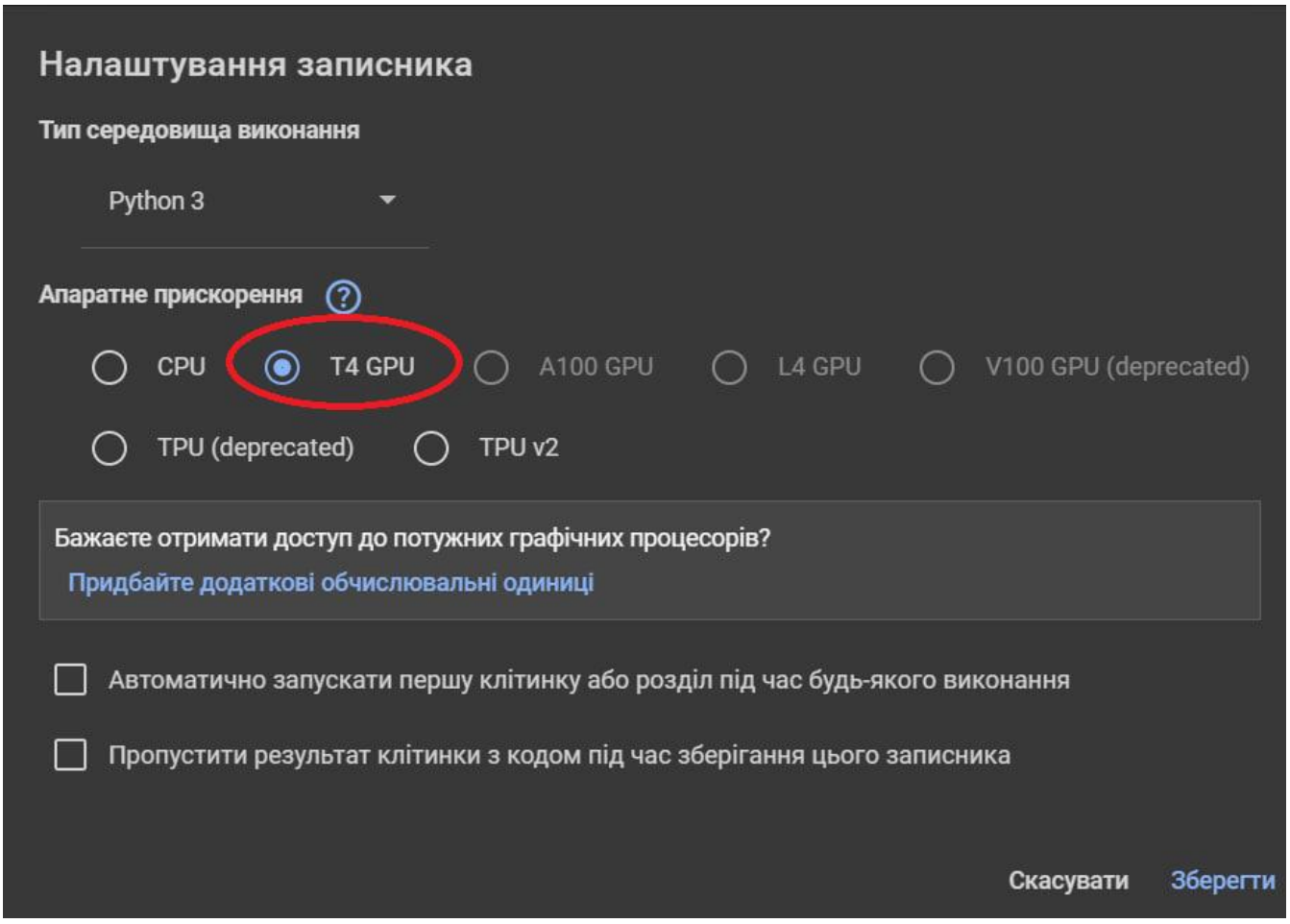

Рисунок 3.1 – Вибір типу апаратного прискорення

Щоб перевірити, що для апаратного прискорення справді використовується GPU, а не інший тип, скористаємося бібліотекою torch. Парамерт *torch.__version__* містить інформацію не тільки про версію бібліотеки, а і про версію CUDA. Якщо

серед результатів є версія CUDA, значить, апаратне прискорення справді відповідає GPU-типу та відповідні ресурси успішно виділено (рис 3.2).

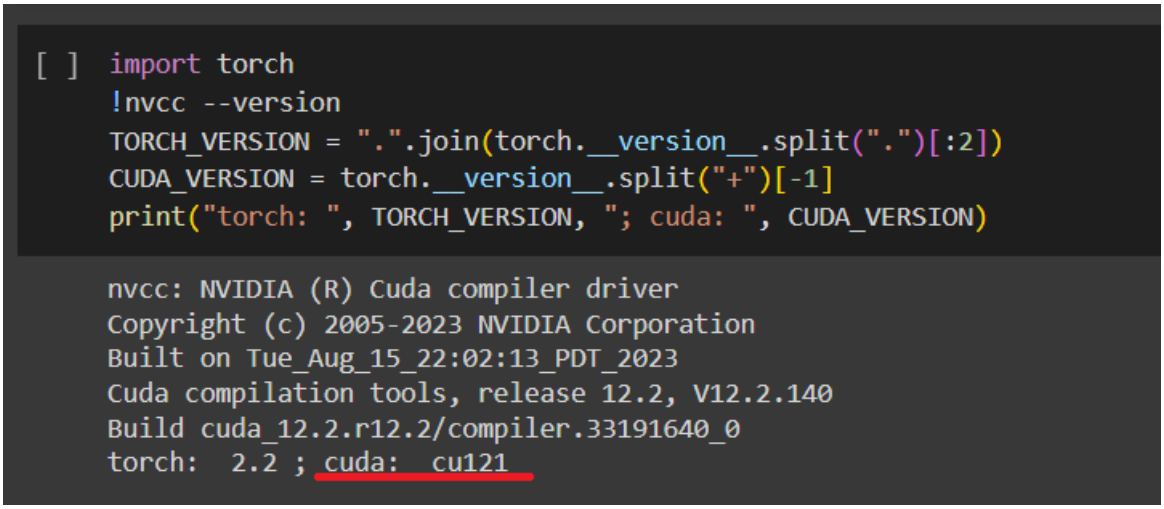

Рисунок 3.2 – Перевірка типу апаратного прискорення

Налаштування успішно виконано, а значить, варто розпочати імпорт необхідних бібліотек. Серед пакетів, що необхідні нам на даному етапі, є ultralytics та supervision. Ultralytics – це офіційний пакет, що, між іншого функціоналу, надає можливість скористатися YOLOv8. Як і більшість інших бібліотек, ultralytics встановлюється за допомогою менеджера пакетів pip. Після встановлення можна перевірити успішність, викликавши метод ultralytics.checks(). Результатом даного виклику має бути повідомлення, що містить інформацію про версію Python, бібліотеки torch та виділені ресурси (рис. 3.3).

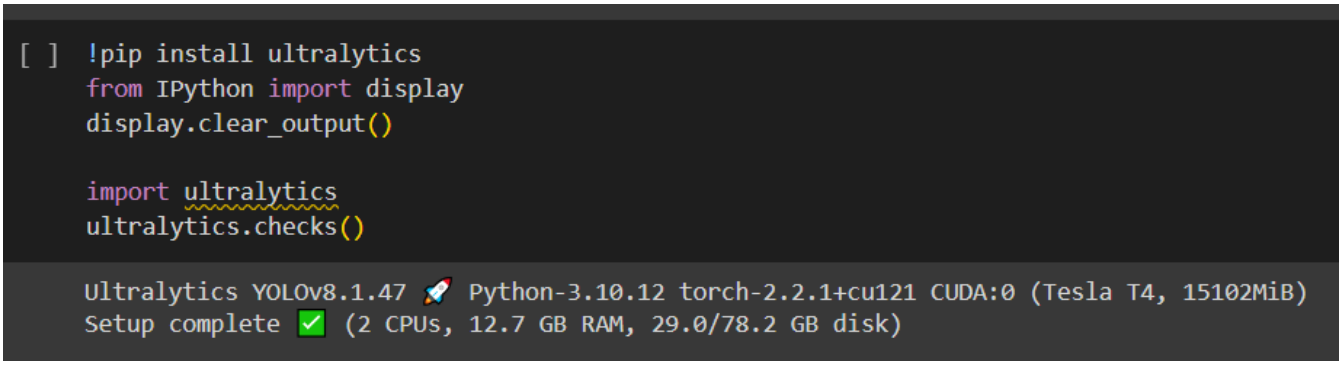

Рисунок 3.3 – Встановлення та перевірка пакету ultralytics

Аналогічно встановлюється і бібліотека supervision від команди roboflow. Даний пакет надасть нам змогу працювати з відео через високорівневий інтерфейс.

Перевірити успішність інсталяції можна викликавши параметр supervision.__version__ (рис. 3.4).

```
[ ]  !pip install supervision==0.2.0

     from IPython import display
     display.clear_output()

     import supervision as sv
     print("supervision", sv.__version__)

     supervision 0.2.0
```

Рисунок 3.4 – Встановлення та перевірка пакету supervision

Майбутня система підтримки рятувальних операцій ДСНС має працювати з “живим” відео, але для початку скористаємося відеоматеріалами roboflow. Використане відео містить декілька класів розпізнавання, відомих моделі, серед яких і “person”. Завантажимо відповідне відео (рис 3.5).

```
%cd {HOME}
!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&

/content
--2024-04-14 20:36:43--  https://docs.google.com/uc?export=download&confirm=&id=1M
Resolving docs.google.com (docs.google.com)... 74.125.69.139, 74.125.69.101, 74.12
Connecting to docs.google.com (docs.google.com)|74.125.69.139|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://drive.usercontent.google.com/download?id=1M3UuH3QNDWGiH0NmGgHtIg
--2024-04-14 20:36:43--  https://drive.usercontent.google.com/download?id=1M3UuH3Q
Resolving drive.usercontent.google.com (drive.usercontent.google.com)... 74.125.13
Connecting to drive.usercontent.google.com (drive.usercontent.google.com)|74.125.1
HTTP request sent, awaiting response... 200 OK
Length: 103282086 (98M) [video/mp4]
Saving to: ‘mall.mp4’

mall.mp4            100%[===================>]  98.50M   226MB/s    in 0.4s

2024-04-14 20:36:45 (226 MB/s) - ‘mall.mp4’ saved [103282086/103282086]
```

Рисунок 3.5 – Завантаження відео-“заглушки”

На даному етапі, маємо все необхідне для початку роботи з попередньо навченою моделлю на основі алгоритму YOLOv8. Для цього імпортуємо бібліотеку ultralytics та створимо потрібну модель (рис. 3.6). Варто зауважити, що при створенні моделі завантажуються додаткові файли, серед яких ваги моделі, тому уже на цьому етапі можлива коротка затримка. На роботу системи в реальному часі це не впливатиме, оскільки цей процес одноразовий.

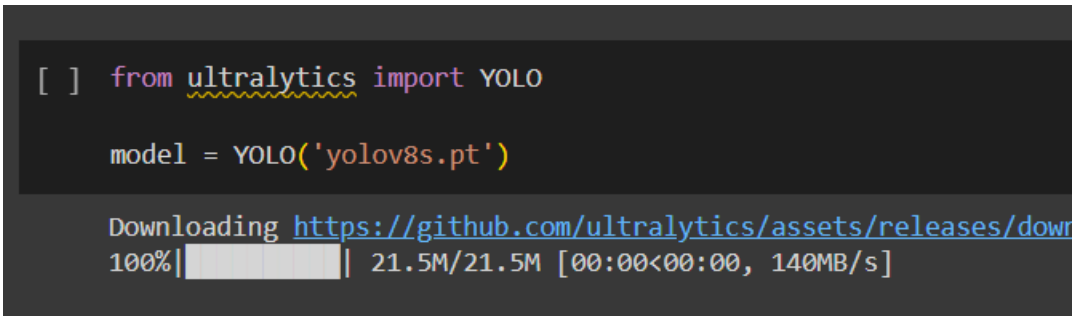

Рисунок 3.6 – Створення об’єкту моделі

Модель, що призначена для розпізнавання об’єктів, опрацьовує або зображення, або відео покадрово, тобто, застосовуючи алгоритми розпізнавання для кожного з зображень. Етапи обробки зображення у нашому проекті можна виділити наступним чином:

- отримання кадру з відео;
- розпізнавання об’єктів на кадрі;
- анотація (у даному випадку – виділення контуром) виявлених об’єктів.

Працюючи з YOLOv8 та з допомогою пакету supervision, спершу необхідно створити генератор, що надаватиме доступ до кожного з кадрів (рис. 3.7). Далі кадр передається до моделі, котра повертає нам масив розпізнаних об’єктів. Отримані результати розмічуються за допомогою об’єкту класу supervision.BoxAnnotator.

```python
import supervision as sv

# extract video frame
generator = sv.get_video_frames_generator(MALL_VIDEO_PATH)
iterator = iter(generator)
frame = next(iterator)

# detect
results = model(frame, imgsz=1280)[0]
detections = sv.Detections.from_yolov8(results)

# annotate
box_annotator = sv.BoxAnnotator(thickness=4, text_thickness=4, text_scale=2)
frame = box_annotator.annotate(scene=frame, detections=detections)

%matplotlib inline
sv.show_frame_in_notebook(frame, (16, 16))
```

Рисунок 3.7 – Використання моделі розпізнавання об’єктів

Отримано наступні результати: розпізнано 9 об’єктів, серед яких екземпляри класів “bottle”, “chair”, “tv”, “refrigerators” та один об’єкт класу “person” (рис. 3.8).

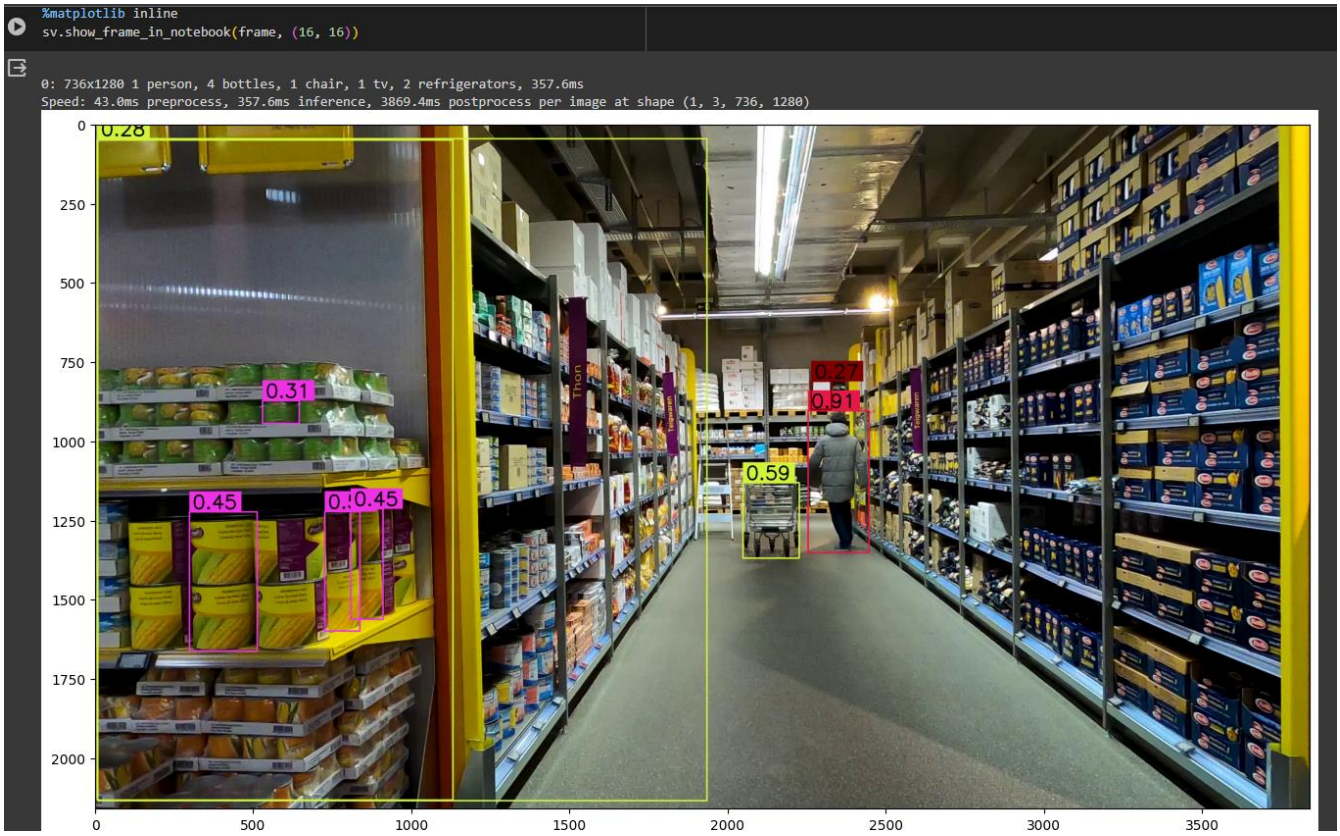

Рисунок 3.8 – Результати розпізнавання першого кадру

Тепер залишається лише застосувати вищеописані дії до кожного з кадрів (рис. 3.9). Пакет supervision надає для таких цілей метод supervision.process_video(), що дозволяє застосувати певний коллбек до всього відео покадрово. Для початку опишемо callback. Відмінність від попереднього алгоритму поляає в тому, що на цей раз ми відфільтруємо всі об’єкти, що не належать класу “person”, оскільки вони нас не цікавлять. Для цього створимо маску, що являє собою масив булевих значень, та передамо її до методу filter() об’єкта detections. Як результат роботи, callback має повертати анотований кадр, якщо маємо хоча б один виявлений об’єкт, або ж просто той же кадр в протилежному випадку.

```python
def process_frame(frame: np.ndarray, _) -> np.ndarray:
    # detect
    results = model(frame, imgsz=1280)[0]
    detections = sv.Detections.from_yolov8(results)

    if len(detections) == 0:
      return frame

    mask = np.array([d[2] == 0 for d in detections])
    detections = detections.filter(mask)

    # annotate
    box_annotator = sv.BoxAnnotator(thickness=4, text_thickness=4, text_scale=2)
    labels = [f"{model.names[class_id]} {confidence:0.2f}" for _, confidence, class_id, _ in detections]
    frame = box_annotator.annotate(scene=frame, detections=detections, labels=labels)

    return frame
```

Рисунок 3.9 – callback для покадрової обробки відео

Маючи callback, залишається лише застосувати його при викликові методу supervision.process_video() (рис. 3.10). Даний метод приймає наступні параметри:

- source_path – шлях до відео, яке необхідно розпізнати;
- target_path – шлях, за яким збережеться анотоване відео;
- callback – метод, що застосовується для кожного з кадрів вказаного відео.

```python
sv.process_video(
    source_path=MALL_VIDEO_PATH,
    target_path=f"{HOME}/mall-result.mp4",
    callback=process_frame
)
```

Рисунок 3.10 – Обробка відео покадрово

Майбутня система розпізнавання об’єктів в реальному часі має розпізнавати близько 30 кадрів в секунду, якщо частота камери відеоспостереження складає 30 FPS. Як бачимо з попередніх результатів, на першому кадрі розпізнано 9 об’єктів і лише один з них належить класу “person”. Крім розпізнавання зайвих об’єктів, маємо операцію відфільтровування зайвих результатів, що стає затратнішою з більшою кількістю об’єктів на кадрі та більшою кількістю кадрів в секунду.

Саме в цей момент і народилася основна гіпотеза даної дослідницької роботи – якщо повторно навчити модель для розпізнавання лише людей, можна отримати приріст у швидкості та/або точності розпізнавання.

3.2 Повторне навчання моделі

Повторне навчання моделі – це процес складний та ресурсозатратний. Для успішного виконання подібного завдання необхідно виконати такі етапи:

- створення датасету;
- розмічення зображень;
- безпосередньо повторне навчання.

Головними моментами, які варто врахувати при створенні датасету, є кількість та різноманітність даних. Від цих двох складових залежить ефективність майбутньої моделі. Розмічення зображень – це повторюваний та довготривалий процес, під час якого на кожному з зображень датасету виділяється область, що містить об’єкт розпізнавання. Існують також підходи, за яких частина датасету використовується для навчання моделі, модель розмічає решту зображень і навчається вдруге на повному датасеті.

На щастя, перші два етапи виконано у відкритому датасеті COCO-2017. Оригінальний датасет COCO, використаний при навчанні моделі з пакету ultralytics, надто обширний і, як наслідок, збільшує час навчання моделі до неможливого в рамках даної роботи. COCO-2017 складається з понад 200 000 зображень, але бібліотека fiftyone надає змогу завантажити потрібну кількість зображень, що

належать до вказаних класів (у нашому випадку, лише до класу “person”).

Як і в попередньому розділі, у нотатнику Google Colab варто перемкнути тип апаратного прискорення з CPU на GPU, аби навчання проходило помітно швидше. Наступним етапом є інсталювання пакету fiftyone (рис. 3.11), оскільки він не входить до списку попередньо встановлених бібліотек Google Colab, Anaconda та інших популярних дистрибутивів.

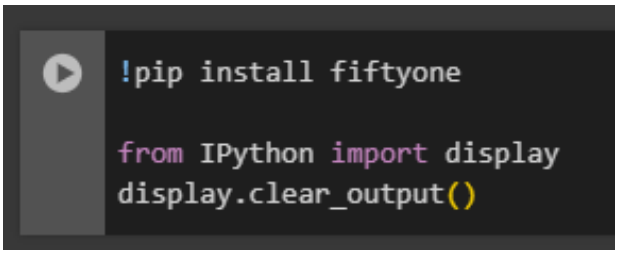

Рисунок 3.11 – Інсталювання пакету fiftyone

Далі, завантажимо набір даних, використовуючи метод fiftyone.zoo.load_zoo_dataset(). Даний метод доволі гнучкий та приймає наступні параметри:

- dataset (обов’язковий) – рядок, що представляє назву набору, який потрібно завантажити;
- split (необов’язковий) – вказує призначення частини набору, який необхідно завантажити. Наприклад, “train” – тренувальний датасет, “val” – валідаційний датасет, “test” – тестовий датасет;
- shuffle (необов’язковий) – булеве значення, що вказує, чи необхідно перемішувати дані після їх завантаження. За замовчуванням “False”;
- dataset_dir (необов’язковий) – шлях до каталогу, де зберігатимуться файли датасету. За замовчуванням обирається тимчасова директорія;
- download (необов’язковий) – булеве значення, що вказує, чи необхідно автоматично завантажувати датасет, якщо його ще не завантажено на локальний комп’ютер. За замовчуванням “True”;
- split_dict (необов’язковий) – словник, що складається з користувацьких розділів датасету та шляхи до них.

Використаємо даний метод для завантаження датасету (рис. 3.12).

```
coco_validation_50 = fiftyone.zoo.load_zoo_dataset(
    "coco-2017",
    split="validation",
    label_types=["detections", "segmentations"],
    classes=["person"],
    max_samples=1800,
)

coco_train = fiftyone.zoo.load_zoo_dataset(
    "coco-2017",
    label_types=["detections", "segmentations"],
    split='train',
    classes=["person"],
    max_samples=300,
)

coco_train = fiftyone.zoo.load_zoo_dataset(
    "coco-2017",
    label_types=["detections", "segmentations"],
    split='test',
    classes=["person"],
    max_samples=100,
)
```

Рисунок 3.12 – Завантаження наборів даних

Перевіримо завантаження датасету, викликавши coco_train у вільній комірці. Якщо змінна coco_train не порожня, то все пройшло вдало.

Наступне питання полягає в тому, що завантажений датасет ще не повністю готовий для навчання моделі YOLOv8. Для використання при навчанні, датасет потрібно правильно експортувати. Для цього існує метод export(), що викликається через об’єкт датасету. Даний метод приймає наступні параметри:

- export_dir (обов’язковий) – шлях до каталогу, в який буде експортовано датасет. Якщо такого каталогу не існує, його буде створено;
- dataset_type (обов’язковий) – тип датасету, який необхідно експортувати. Наприклад, “tf-dataset” чи “torch-dataset”;
- dataset_exporter – об’єкт, що реалізує можливості експорту для обраних типів

даних. Якщо не вказано, буде використано автоматичний підбір експортера на основі значення параметру “dataset_type”;

- export_params (необов’язковий) – доаткові опціональні параметри для конфігурації експортування. Наприклад, вони можуть включати параметри для процесингу зображень, обсяги вибірок тощо;
- overwrite (необов’язковий) – булеве значення, що вказує, чи необхідно перезаписувати уже існуючі файли експорту, якщо такі вже існують в цільовому каталозі. За замовчуванням “False”.

Експортуємо датасет для YOLOv8 (рис. 3.13). Для цього в якості значення параметру dataset_type вкажемо тип fiftyone.types.YOLOv8Dataset.

```
[ ]  # Save coco_train dataset to YAML
     coco_train.export(
         export_dir="/content/test4",
         dataset_type=fiftyone.types.YOLOv8Dataset,
     )
```

Рисунок 3.13 – Ескпорт датасету

Тепер, коли дані повністю готові, скористаємося консольною утилитою yolo, що поставляється разом з пакетом ultralytics (рис 3.14). Вкажемо наступні параметри:

- task=detect;
- mode=train – режим навчання моделі;
- model=yolo8s.pt – вибір необхідної моделі;
- data=dataset/dataset.yaml – шлях до yaml файлу, що містить інформацію про датасет;
- epochs=100 – кількість епох навчання (рекомендована кількість – 300, але в рамках даної роботи це неможливо через брак ресурсів).
- imgsz=1200.

```
%cd {HOME}

!yolo task=detect mode=train model=yolov8s.pt data=dataset/dataset.yaml epochs=100 imgsz=800

/content
Ultralytics YOLOv8.1.47 🚀 Python-3.10.12 torch-2.2.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
engine/trainer: task=detect, mode=train, model=yolov8s.pt, data=test2/dataset.yaml, epochs=5, time=None,
```

Рисунок 3.14 – Початок навчання моделі

Процес навчання моделі – довготривалий та споживає багато ресурсів. Через це важко підібрати оптимальну кількість епох навчання та розмір датасету.

Після завершення навчання, перевіримо працездатність моделі, використовуючи код з попередньо розділу (рис. 3.15).

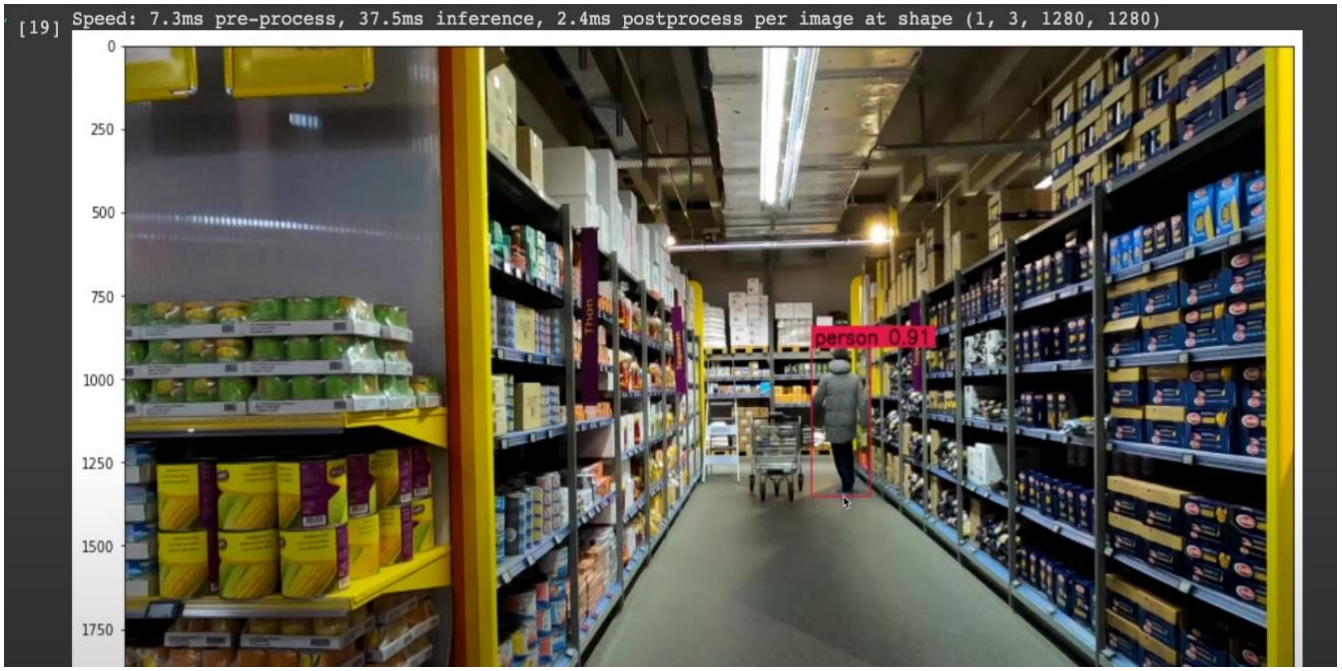

Рисунок 3.15 – Результат роботи повторно навченої моделі

Після успішного розпізнавання настав час порівняти оригінальну модель з повторно навченою. Через обмеженість ресурсів не вдалося встановити оптимальні параметри, але подані в подальшому значення підтверджують життєздатність гіпотези даної дослідницької роботи та залишають простір для можливих покращень.

Для порівняння моделей використання датасетів COCO уже не є доцільним, оскільки зображення з цих датасетів входити у тренувальні набори та можуть

спотворити результати. Тож було прийнято рішення використати датасет CrowdHuman. Він ідеально підходить для наших цілей, оскільки містить велику кількість різнопланових сцен з людьми у густонаселених локаціях; має екземпляри з різними позами, масштабами, орієнтаціями тощо.

Таблиця 3.1 – Порівняння параметрів навчання моделей

Джерело: Побудовано автором

Параметри	YOLOv8 Оригінальна	YOLOv8 Перенавчена
К-сть епох	300	100
Датасет	COCO	COCO-2017
К-сть зображень	понад 200	2200
К-сть класів	1203	1

Таблиця 3.2 – Порівняння швидкості моделей

Джерело: Побудовано автором

Параметри	YOLOv8 Оригінальна	YOLOv8 Перенавчена
Середня швидкість обробки кадру, мс	≈ 19.1	≈ 19.1

Результати порівняння відрізняються мільйонними частинами секунди, що дозволяє вважати їх рівними. Ймовірно, швидкість розпізнавання більше залежить від самого алгоритму YOLO. Обидві моделі можуть розпізнавати близько 67 кадрів за секунду, що робить їх ідеальним вибором для досліджуваної системи, оскільки вони можуть опрацьовувати діапазон відео з камер частотою до 60 кадрів у секунду без додаткової оптимізації.

Таблиця 3.3 – Порівняння точності розпізнавання

Джерело: Побудовано автором

Параметри	YOLOv8 Оригінальна	YOLOv8 Перенавчена
Середня точність розпізнавання об’єкта, %	91.498	91.503
Стандартне відхилення	4.71	4.52

Результати порівняння точності дещо підтверджують життєздатність досліджуваної гіпотези. Маємо незначне, але позитивне зрушення відносно номінальної точності розпізнавання об’єкта. Крім того, розраховані на масивах результатів розпізнавання показники стандартного відхилення показують, що результати повторно навченої моделі має “стабільніші” результати, тобто, маємо менший розкид даних від центра. Відповідно, маємо менше результатів з низькою точністю розпізнавання. Вищевказані показники порівняння залишають великий простір для подальшого дослідження.

3.3 Розроблення системи розпізнавання у реальному часі

Останнім та найголовнішим завданням даної дослідницької роботи є реалізація системи додатків типу “клієнт-сервер” для підтримки рятувальних операцій. Вона має складатися з двох частин – клієнтської та серверної. Клієнтська програма має встановлюватися на стороні користувача (держава для державних громадських місць, на кшталт сховищ, та приватний підприємець для приватних громадських місць, на кшталт кафе чи супермаркетів) [22]. Серверна сторона має розгортатися на стороні рятувальних служб, наприклад, на стороні Державної служби України з надзвичайних ситуацій.

Загальна структура системи та її основні елементи подані на IDEF0 діаграмі, зображеній на рисунку 3.16.

Розпочнемо реалізацію з клієнтської частини. Структура та архітектура додатку будуть доволі простими (рис. 3.17). Файл main.py – це файл, який є точкою старту додатку та містить основну логіку. Каталог model містить повторно навчену модель, розроблену у попередньому розділі. Каталог utils призначений для зберігання файлу з додатковою логікою, у нашому випадку, логікою для відправки даних на сервер.

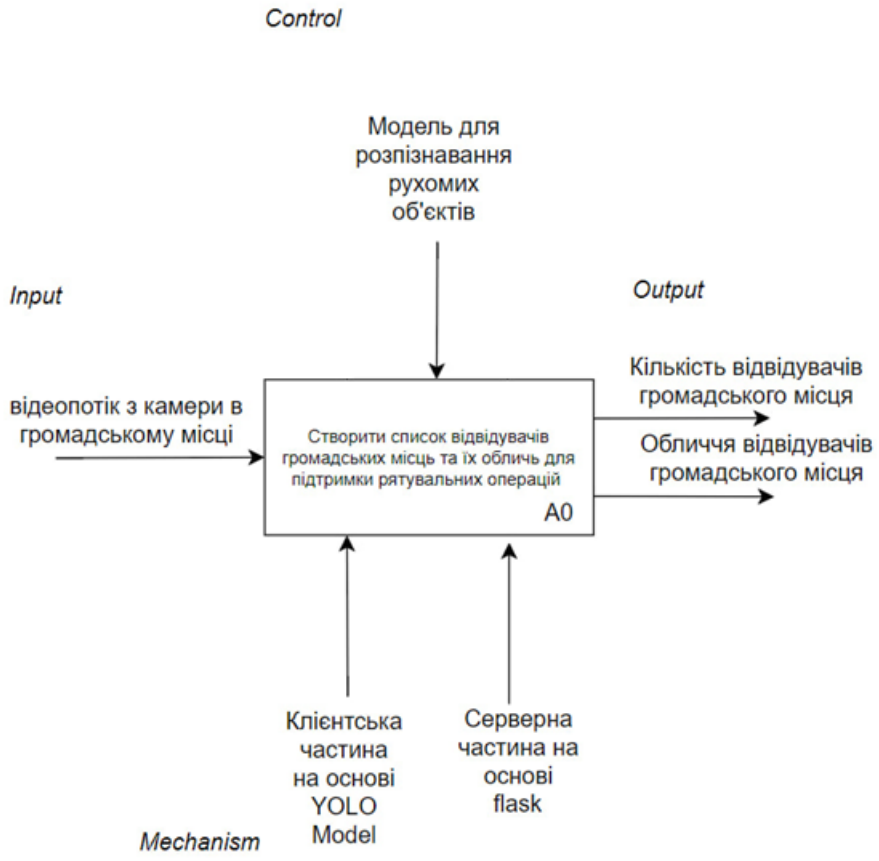

Рисунок 3.16 – IDEF0 системи

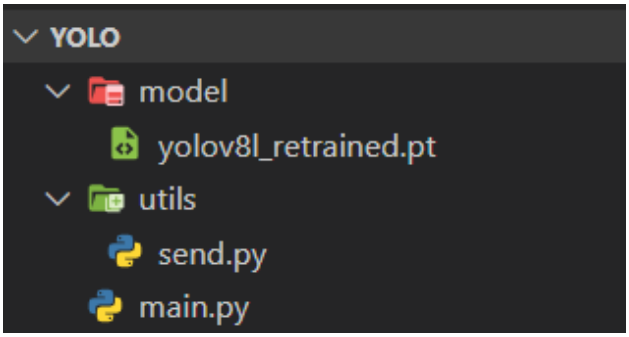

Рисунок 3.17 – Структура проекту клієнтського додатку

Розглянемо файл main.py. Починається він з імпорту необхідних пакетів. Для розробки клієнтського додатку необхідні наступні бібліотеки:

- argparse – стандартна бібліотека для парсингу параметрів консольного рядка;
- cv2 (OpenCV) – для фіксування відеопотоку та відображення результатів;
- ultralytics – для роботи з повторно навченою моделлю;
- supervision – для роботи з відео.

Додатково імпортуємо метод send_to_server() з пакету utils.send (рис. 3.18), який ми розглянемо пізніше.

```
1  import cv2
2  import argparse
3
4  from ultralytics import YOLO
5  import supervision as sv
6
7  from utils.send import send_to_server
```

Рисунок 3.18 – Імпорт бібліотек

Щоб зробити наш додаток більш гнучким – а саме надати йому можливість ефективно працювати з камерами різної розподільної здатності – ми додамо метод parse_arguments(), що при запуску скрипта з консолі приймає аргумент –webcam-resolution. Для додавання аргументу скористаємося методом parser.add_argument() (рис. 3.19), встановивши наступні значення:

- ‘—webcam-resolution’ – назва аргументу;
- default=[1280, 720] – значення аргументу за замовчуваннямм;
- nargs=2 – кількість значень, що містить аргумент;
- type=int – тип даних.

```python
def parse_argumments() -> argparse.Namespace:
    parser = argparse.ArgumentParser(description='yolov8')
    parser.add_argument(
        '--webcam-resolution',
        default=[1280, 720],
        nargs=2,
        type=int
    )

    args = parser.parse_args()

    return args
```

Рисунок 3.19 – Метод parse_arguments()

Далі необхідно отримати значення розподільної здатності камери та встановити відповідні значення для об’єкта з захоплення відео (рис. 3.20).

```python
def main():
    args = parse_argumments()

    frame_width, frame_height = args.webcam_resolution

    cap = cv2.VideoCapture(0)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, frame_width)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, frame_height)

```

Рисунок 3.20 – Встановлення значень розподільної здатності

Тепер ініціалізуємо модель, використовуючи ваги моделі, повторно навченої у попередньому розділі. Для цього варто лише у конструктор YOLO() передати актуальний шлях до вагів. У нашому випадку, це model/yolov8l_retrained.pt.

Для демонстрації роботи системи та позначення людей на фото, що буде передане на серверну частину, створимо BoxAnnotator (рис. 3.21). BoxAnnotator дозволяє додавати анотації до зображення, позначаючи людей рамкою.

```
34
35      box_annotator = sv.BoxAnnotator(
36          thickness=2,
37          text_thickness=2,
38          text_scale=1
39      )
40
```

Рисунок 3.21 – Створення BoxAnnotator

Тепер все готово для безпосереднього процесу розпізнавання. Він складатиметься з наступних етапів:

1) перехоплення кадру;
2) виявлення людей;
3) анотація зображення;
4) виведення зображення для демонстрації та відправка даних на сервер.

Ці етапи мають відбуватися циклічно, поки працює система.

Перехоплення зображення можливе завдяки методу read() попередньо створеного об’єкту захоплення. Для виявлення людей відповідний фрейм OpenCV варто просто передати до моделі. Результат роботи моделі необхідно конвертувати до класу Detections за допомогою методу Detections.from_utrlytics(). Це дозволить нам використати розпізнавання для анотації фрейму (рис. 3.22).

```
40
41      while True:
42          ret, frame = cap.read()
43
44          result = model(frame)[0]
45          detections = sv.Detections.from_ultralytics(result)
46
47          frame = box_annotator.annotate(
48              scene=frame,
49              detections=detections,
50              labels=['Person' for _ in detections]
51          )
52
53          persons_count = len(detections)
54
```

Рисунок 3.22 – Цикл розпізнавання

У цьому ж циклі ми відправляємо дані на серверну частину та відображаємо їх для демонстрації. Результати роботи клієнтської частини подано на рисунку 3.23.

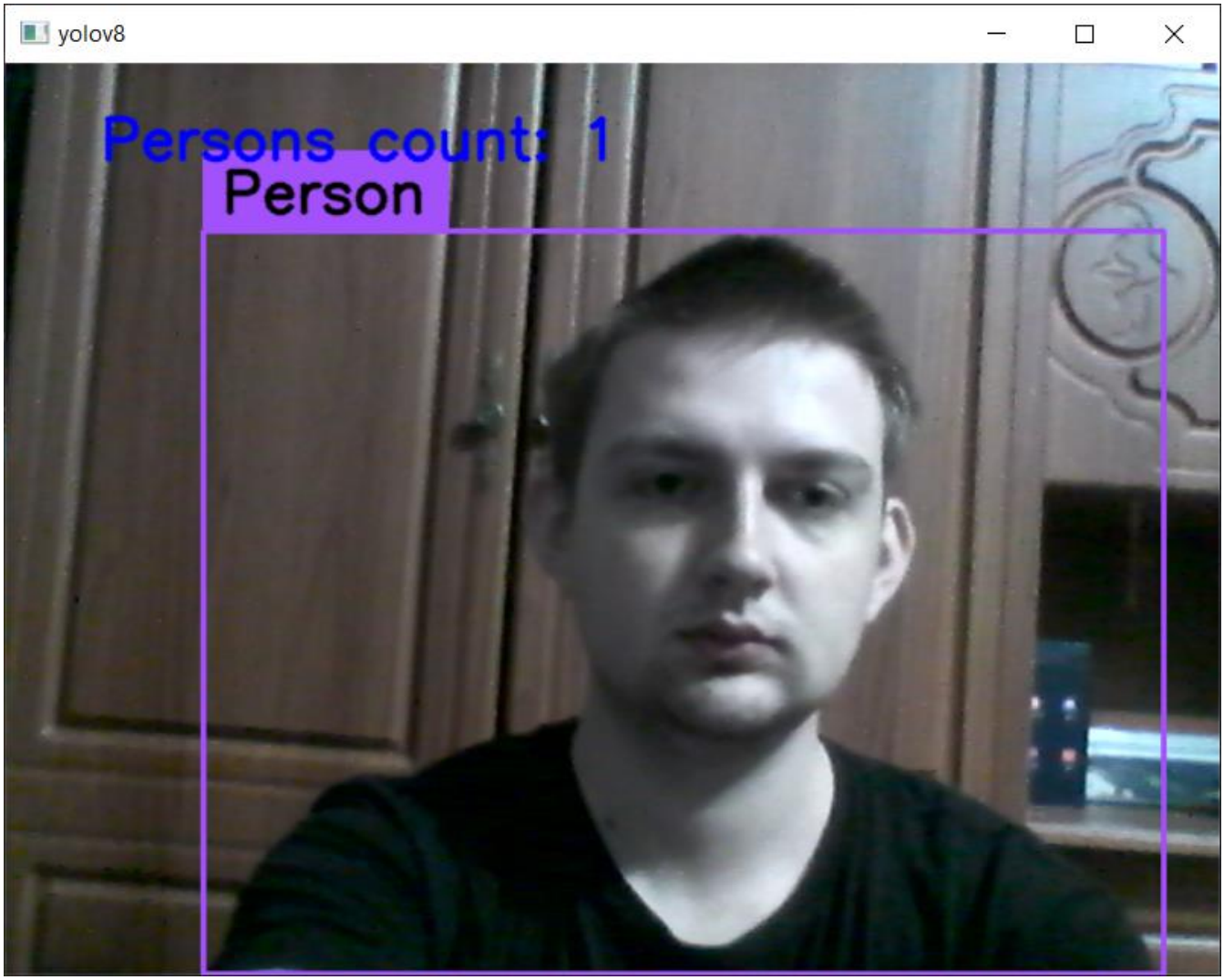

Рисунок 3.23 – Зразок результуючого зображення

Для зв’язку з серверною стороною – а саме відправки даних – використовується метод utils.send_to_server() (рис. 3.24). Даний метод приймає два параметри: count (кількість людей) та frame – зображення, що містить виявлених людей.

```python
import cv2
import requests
import numpy as np

def send_to_server(count, frame):
    # Конвертуємо зображення OpenCV в формат JPEG для відправки на сервер
    _, encoded_image = cv2.imencode('.jpg', frame)
    image_bytes = encoded_image.tobytes()

    # Параметри, які ми відправимо на сервер
    data = {'count': count}

    # Відправляємо POST-запит на сервер
    response = requests.post('http://mock_server.local/add', data=data, files={'image': image_bytes})

    # Перевіряємо статус відповіді
    if response.status_code == 200:
        print("Дані успішно відправлено на сервер")
    else:
        print(f"Сталася помилка при відправленні на сервер: {response.status_code}")
```

Рисунок 3.24 – Метод send_to_server()

Розробка серверної частини наразі передбачає лише імітацію серверу ДСНС, своєрідний mock-сервер. Це звичайний додаток на Flask, що отримує дані з клієнтської частини, та відображає їх в панелі адміністратора. Роботу додатку зображено на рисунку 3.25.

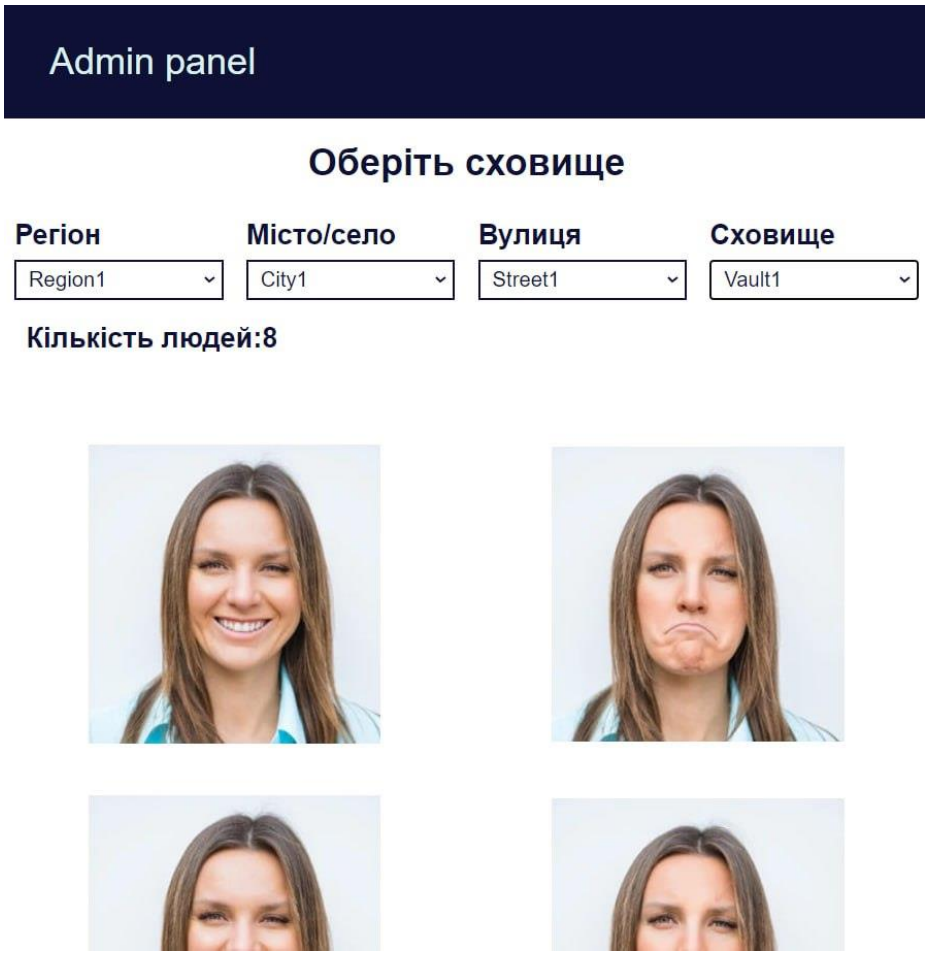

Рисунок 3.25 – Результат роботи серверної частини

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи була створена інтелектуальна технологія глибинного машинного навчання для системи підтримки рятувальних операцій, що розпізнає людей в реальному часі, фіксує їх візуальні дані та їх кількість. Під час розробки проекту були враховано усі вимоги до програмної реалізації системи.

У ході виконання кваліфікаційної роботи магістра

- використана модель розпізнавання об’єктів для розпізнавання людей у реальному часі;
- натренована модель розпізнавання об’єктів для розпізнавання виключно людей;
- проведено порівняння результатів оригінальної моделі та натренованої повторно;
- розроблено систему додатків типу клієнт-сервер для постачання даних про людей та їх кількість в реальному часі.

Для цього були виконані такі завдання:

1. Здійснено аналіз проблемної галузі.
2. Проведено аналіз існуючих рішень розпізнавання об’єктів.
3. Обрано технологічний стек проекту.
4. Розроблена інтелектуальна технологія розпізнавання людей в реальному часі.

За результатами роботи інтелектуальної системи можна зробити висновок, що застосування розробленої системи надасть рятівним службам додаткову інформацію про потенційних постраждалих, а повторно навчена модель YOLO v8 показує стабільніші результати розпізнавання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Richard Szeliski. Computer Vision: Alghorithms and Applications (Texts in Computer Science). 2nd ed. 2022 Edition. P. 45–50.
2. Rafael C. Gonzalez, Richard E. Woods. Digital Image Processing, 2017. P. 112-118.
3. Charu C. Aggarwal. Neural Networks and Deep Learning: A Textbook. 1st ed. 2018 Edition. P. 48-66.
4. Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning (Adaptive Computation and Machine Learning series), 2016. P. 25-39.
5. Josh Patterson, Adam Gibson. Deep Learning: A Practiotioner's Approach, 2017. P. 144-151.
6. Simon Haykin. Neural Networks: A Comprehensive Foundation, 2022. P. 207-222.
7. Francois Cholett. Deep Learning with Python, 2017. P. 232-240.
8. Christopher M. Bishop. Pattern Recognition and Machine Learning (Information Science and Statistics), 2011. P. 20-44.
9. Simon J. D. Prince. Computer Vision (1st Edition), 2012. P. 177-183.
10. YOLOv8 [Electronic resource]. URL: https://yolov8.com (accessed: 29.04.2024)
11. Welcome to Python.org [Electronic resource]. URL: https://www.python.org (accessed: 29.04.2024).
12. Aston Zhang et al. Dive into Deep Learning (1st Edition), 2023. P. 301-312.
13. Virtualenv [Electronic resource]. URL: https://virtualenv.pypa.io/en/latest/ (accessed: 29.04.2024).
14. Ultralytics [Electronic resource]. URL: https://www.ultralytics.com (accessed: 29.04.2024).
15. Tsung-Yi Lin, Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., … Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. *CoRR, abs/1405.0312*. Retrieved from http://arxiv.org/abs/1405.0312
16. Shao, S., Zhao, Z., Li, B., Xiao, T., Yu, G., Zhang, X., & Sun, J. (2018).

CrowdHuman: A Benchmark for Detecting Human in a Crowd. *arXiv preprint arXiv:1805.00123*.

17. Fiftyone [Electronic Resource] URL: https://docs.voxel51.com (accessed: 29.04.2024).
18. Supervision by roboflow [Electronic resource]. URL: https://supervision.roboflow.com/latest (accessed: 29.04.2024).
19. Samarth Brahmbhatt. Practical OpenCV (1st Edition), 2013. P. 113-121.
20. Project Jupyter | Home [Electronic resource] URL: https://jupyter.org (accessed: 29.04.2024).
21. Al Sweigart. Automate the Boring Stuff with Python, 2nd Edition: Practical Programming for Total Beginners, 2019. P. 299-307.
22. Скороход А. А., Шовкопляс О. А. Інформаційна технологія підтримки рятувальних операцій Державної служби України з надзвичайних ситуацій // Інформатика, математика, автоматика (IMA – 2024) : матеріали та програма міжнародної науково-технічної конференції, м. Суми, 22–26 квітня 2024 р. Суми : СумДУ, 2024. С. 91.

ДОДАТКИ

Додаток А – Лістинг програмного коду

Клієнтська частина

main.py

```
import cv2
import argparse


from ultralytics import
YOLO
import supervision as sv


from utils.send import
send_to_server



def parse_argumments() ->
argparse.Namespace:
    parser =
argparse.ArgumentParser(des
cription='yolov8')
    parser.add_argument(
        '--webcam-
resolution',
        default=[1280,
720],
        nargs=2,
        type=int
```

```
    )

    args =
parser.parse_args()

    return args



def main():
    args =
parse_argumments()

    frame_width,
frame_height =
args.webcam_resolution

    cap =
cv2.VideoCapture(0)

cap.set(cv2.CAP_PROP_FRAME_
WIDTH, frame_width)

cap.set(cv2.CAP_PROP_FRAME_
HEIGHT, frame_height)

    model =
YOLO('model/yolov8l_retrain
ed.pt')
```

```
    box_annotator =
sv.BoxAnnotator(
        thickness=2,
        text_thickness=2,
        text_scale=1
    )

    while True:
        ret, frame =
cap.read()

        result =
model(frame)[0]
        detections =
sv.Detections.from_ultralyt
ics(result)

        frame =
box_annotator.annotate(
            scene=frame,
detections=detections,
labels=['Person' for _ in
detections]
        )

        persons_count =
```

```
len(detections)

        label_text =
f"Persons count:
{persons_count}"

        cv2.putText(
            frame,
            label_text,
            (50, 50),
cv2.FONT_HERSHEY_SIMPLEX,
            1,
            (255, 0, 0),
            2,
            cv2.LINE_AA
        )

        send_to_server(
            persons_count,
            frame
        )

cv2.imshow('yolov8', frame)

        if (cv2.waitKey(30)
== 27):
```

```
            break


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

send.py

```
import cv2
import requests
import numpy as np


def send_to_server(count,
frame):
    # Конвертуємо
зображення OpenCV в формат
JPEG для відправки на
сервер
    _, encoded_image =
cv2.imencode('.jpg', frame)
    image_bytes =
encoded_image.tobytes()


    # Параметри, які ми
відправимо на сервер
    data = {'count': count}


    # Відправляємо POST-
запит на сервер
```

```
    response =
requests.post('http://mock_
server.local/add',
data=data, files={'image':
image_bytes})


    # Перевіряємо статус
відповіді
    if response.status_code
== 200:
        print("Дані успішно
відправлено на сервер")
    else:
        print(f"Сталася
помилка при відправленні на
сервер:
{response.status_code}")
```

Серверна частина

templates/admin.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
content="width=device-
width, initial-scale=1.0">
    <title>Admin
panel</title>
```

```
    <link rel="stylesheet"
href="{{ url_for('static',
filename='css/clean.css')
}}">
    <link rel="stylesheet"
href="{{ url_for('static',
filename='css/admin.css')
}}">
</head>
<body>
    <header>
        Admin panel
    </header>

    <main>
        <h1>Оберіть
сховище</h1>
        <section
class="select__container">
            <section>

<h2>Регіон</h2>
                <select
name="region" id="region">
                </select>
            </section>

            <section>
```

```
<h2>Місто/село</h2>
                <select
name="city" id="city"
disabled>
                </select>
            </section>

            <section>

<h2>Вулиця</h2>
                <select
name="street" id="street"
disabled>
                </select>
            </section>

            <section>

<h2>Сховище</h2>
                <select
name="vault" id="vault"
disabled
onchange="handleVaultSelect
ion()">
                </select>
            </section>
        </section>
```

```
        <div
id="counter"></div>


        <section class=""
id="people-container">


        </section>
    </main>


    <script src="{{
url_for('static',
filename='js/admin_fetch.js
') }}"></script>
    <script>
        // Define
handleVaultSelection
function
        async function
handleVaultSelection() {
            const region =
document.getElementById('re
gion').value;
            const city =
document.getElementById('ci
ty').value;
            const street =
document.getElementById('st
reet').value;
            const vault =
document.getElementById('va
```

```
ult').value;

            // Call
fetchPeople function and
use the result as needed
            const people =
await fetchPeople(region,
city, street, vault);

            // Implement
your logic to display or
update the people list

console.log('Fetched
people:', people);

            const
people_container =
document.getElementById('pe
ople-container');

            // Clear
existing children

people_container.innerHTML
= '';

document.getElementById('co
unter').innerHTML =
'<h2>Кількість людей:' +
```

```
people.length + '</h2>';


            for (const
person of people) {
                // Create a
new <div> element for each
person
                const
personDiv =
document.createElement('div
');


                // If a
photo URL is available,
create an <img> element and
set its src attribute
                if
(person.photo_url) {
                    const
imgElement =
document.createElement('img
');

console.log(person.photo_ur
l);

imgElement.src = '{{
url_for("static",
filename="img/") }}' +
person.photo_url.split('\\'
)[1];
```

```
console.log(imgElement.src)
;
                    // You
can customize the image
size or other attributes as
needed

personDiv.appendChild(imgEl
ement);
                }

                // Append
the person's div to the
people container

people_container.appendChil
d(personDiv);
            }

        }

        // Add event
listener using modern
approach

document.getElementById('va
ult').addEventListener('cha
nge',
handleVaultSelection);
```

```
        async function
fetchPeople(region, city,
street, vault) {
            const endpoint
=
`/admin/people/${encodeURIC
omponent(region)}/${encodeU
RIComponent(city)}/${encode
URIComponent(street)}/${enc
odeURIComponent(vault)}`;

            try {
                const
response = await
fetch(endpoint);

                if
(!response.ok) {
                    throw
new Error(`Failed to fetch
people: ${response.status}
${response.statusText}`);
                }

                const data
= await response.json();
                return
data.people;
            } catch (error)
```

```
{


console.error('Error
fetching people:',
error.message);
                return [];
// Return an empty array or
handle the error as needed
            }
        }
    </script>
</body>
</html>
```

Використання базової моделі

```
import torch
!nvcc --version
TORCH_VERSION =
".".join(torch.__version__.
split(".")[:2])
CUDA_VERSION =
torch.__version__.split("+"
)[-1]
print("torch: ",
TORCH_VERSION, "; cuda: ",
CUDA_VERSION)


import os
HOME = os.getcwd()
```

```
HOME

!pip install ultralytics

from IPython import display

display.clear_output()

import ultralytics

ultralytics.checks()

!pip install
supervision==0.2.0

from IPython import display

display.clear_output()

import supervision as sv

print("supervision",
sv.__version__)

%cd {HOME}

!wget --load-cookies
/tmp/cookies.txt
"https://docs.google.com/uc
?export=download&confirm=$(
wget --quiet --save-cookies
/tmp/cookies.txt --keep-
session-cookies --no-check-
certificate
```

```
'https://docs.google.com/uc
?export=download&id=1M3UuH3
QNDWGiH0NmGgHtIgXXGDo_nigm'
-O- | sed -rn
's/.*confirm=([0-9A-Za-
z_]+).*/\1\n/p')&id=1M3UuH3
QNDWGiH0NmGgHtIgXXGDo_nigm"
-O mall.mp4 && rm -rf
/tmp/cookies.txt



MALL_VIDEO_PATH =
f"{HOME}/mall.mp4"


from ultralytics import
YOLO


model = YOLO('yolov8s.pt')


import supervision as sv


# extract video frame

generator =
sv.get_video_frames_generat
or(MALL_VIDEO_PATH)

iterator = iter(generator)

frame = next(iterator)


# detect
```

```
results = model(frame,
imgsz=1280)[0]

detections =
sv.Detections.from_yolov8(r
esults)


# annotate

box_annotator =
sv.BoxAnnotator(thickness=4
, text_thickness=4,
text_scale=2)

frame =
box_annotator.annotate(scen
e=frame,
detections=detections)


%matplotlib inline

sv.show_frame_in_notebook(f
rame, (16, 16))


import supervision as sv

import numpy as np


# extract video frame

generator =
sv.get_video_frames_generat
or(MALL_VIDEO_PATH)

iterator = iter(generator)

frame = next(iterator)
```

```
# detect

results = model(frame,
imgsz=1280)[0]

detections =
sv.Detections.from_yolov8(r
esults)


mask = np.array([d[2] == 0
for d in detections])

detections =
detections.filter(mask)


# annotate

box_annotator =
sv.BoxAnnotator(thickness=4
, text_thickness=4,
text_scale=2)

labels =
[f"{model.names[class_id]}
{confidence:0.2f}" for _,
confidence, class_id, _ in
detections]

frame =
box_annotator.annotate(scen
e=frame,
detections=detections,
labels=labels)


%matplotlib inline
```

```
sv.show_frame_in_notebook(f
rame, (16, 16))


import numpy as np
import supervision as sv



# initiate annotators
box_annotator =
sv.BoxAnnotator(thickness=4
, text_thickness=4,
text_scale=2)


def process_frame(frame:
np.ndarray, _) ->
np.ndarray:
    # detect
    results = model(frame,
imgsz=1280)[0]
    detections =
sv.Detections.from_yolov8(r
esults)


    if len(detections) ==
0:
      return frame


    mask = np.array([d[2]
== 0 for d in detections])
```

```
    detections =
detections.filter(mask)


    # annotate
    box_annotator =
sv.BoxAnnotator(thickness=4
, text_thickness=4,
text_scale=2)
    labels =
[f"{model.names[class_id]}
{confidence:0.2f}" for _,
confidence, class_id, _ in
detections]
    frame =
box_annotator.annotate(scen
e=frame,
detections=detections,
labels=labels)


    return frame


sv.process_video(

source_path=MALL_VIDEO_PATH
,

target_path=f"{HOME}/mall-
result.mp4",
    callback=process_frame
)
```

```
from IPython import display
display.clear_output()
```

Перенавчання моделі

```
!nvidia-smi

import os
HOME = os.getcwd()
HOME

!pip install ultralytics

from IPython import display
display.clear_output()

!pip install fiftyone

from IPython import display
display.clear_output()

import fiftyone

coco_validation_50 =
fiftyone.zoo.load_zoo_datas
et(
    "coco-2017",
```

```
    split="validation",

label_types=["detections",
"segmentations"],
    classes=["person"],
    max_samples=1800,
)

coco_train =
fiftyone.zoo.load_zoo_datas
et(
    "coco-2017",

label_types=["detections",
"segmentations"],
    split='train',
    classes=["person"],
    max_samples=300,
)

coco_train =
fiftyone.zoo.load_zoo_datas
et(
    "coco-2017",

label_types=["detections",
"segmentations"],
    split='test',
```

```
    classes=["person"],
    max_samples=100,
)

coco_train

# Save coco_train dataset
to YAML
coco_train.export(

export_dir="/content/test4"
,

dataset_type=fiftyone.types
.YOLOv8Dataset,
)

print(list(coco_train.to_di
ct()))



%cd {HOME}

!yolo task=detect
mode=train model=yolov8s.pt
data=dataset/dataset.yaml
epochs=100 imgsz=800
```