МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ СИСТЕМ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Комп'ютерна система для визначення положення людини в обмеженому просторі у реальному часі**

Виконав: студент 2 курсу, групи 8.1219-пзс
Спеціальності
121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми
Інженерія програмного забезпечення
(код і назва освітньої програми)
К. С. Антошин
(ініціали та прізвище)
Керівник доцент кафедри ПЗАС
А. І. Безверхий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дісітел»
П. О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ

Кафедра _____ програмного забезпечення автоматизованих систем
Рівень вищої освіти _____ другий (магістерський) _____
Спеціальність ___121 Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри В. Г. Вербицький
“ 01 ” вересня 2020 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Антошину Кирилу Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи ___Комп’ютерна система для визначення положення людини в обмеженому просторі у реальному часі

керівник роботи __Безверхий Анатолій Ігорович, доцент кафедри ПЗАС
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від _“25” травня 2020 року № 600-с_

2. Строк подання студентом кваліфікаційної роботи ___30.11.2020____
3. Вихідні дані магістерської роботи
   - комплект нормативних документів;
   - технічне завдання до роботи.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   - огляд та збір літератури стосовно теми кваліфікаційної роботи;
   - огляд та аналіз існуючих рішень та аналогів;
   - дослідження проблеми визначення положення людини в обмеженому просторі у реальному часі та розробка методів її вирішення;
   - створення програмного продукту та його опис;
   - перелік вимог для роботи програми;
   - дослідження поставленої проблеми, розробка висновків та пропозицій.
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
18 слайдів презентації

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.09.2020

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Формулювання задачі кваліфікаційної роботи та її узгодження з науковим керівником	01.09-02.09.2020	Виконано
2	Дослідження проблеми визначення положення людини в обмеженому просторі у реальному часі	02.09-16.09.2020	Виконано
3	Дослідження засобів вирішення поставленої проблеми	17.09-04.10.2020	Виконано
4	Узгодження подальших дій з науковим керівником	05.10-06.10.2020	Виконано
5	Навчання моделей нейронних мереж розпізнавати людей	06.10-18.10.2020	Виконано
6	Представлення результатів навчання моделей науковому керівнику та узгодження подальшого плану дослідження	19.10-20.10.2020	Виконано
7	Реалізація комп'ютерної системи для вирішення поставленої проблеми	20.10-03.11.2020	Виконано
8	Дослідження результатів навчання моделей, а також швидкості та точності розробленої системи, варіанти їхнього покращення	04.11-08.11.2020	Виконано
9	Демонстрація розробленої системи науковому керівнику та отримання вказівок щодо оформлення звіту до кваліфікаційної роботи	09.11-10.11.2020	Виконано
10	Оформлення звіту до кваліфікаційної роботи	10.11-27.11.2020	Виконано

Студент _____ К. С. Антошин
(підпис) (прізвище та ініціали)

Керівник роботи _____ А. І. Безверхий
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І. А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок: 106

Рисунків: 33

Таблиць: 10

Джерел: 36

Антошин Кирило Сергійович. Комп’ютерна система для визначення положення людини в обмеженому просторі у реальному часі.

Кваліфікаційна робота для здобуття ступеня вищої освіти магістра за спеціальністю 121 — Інженерія програмного забезпечення, науковий керівник А. І. Безверхий. Інженерний навчально-науковий інститут Запорізького національного університету.

Мета і завдання дослідження полягають у вивченні сучасних підходів до визначення положення людини в обмеженому просторі у реальному часі, а також у розробці комп’ютерної системи, що буде ефективно вирішувати цю задачу, працюючи на мікрокомп’ютері NVIDIA Jetson Nano та використовуючи в якості вхідних даних відеопотік із підключеної камери.

У процесі дослідження була розглянута проблема визначення положення людини в обмеженому просторі у реальному часі та підходи глибинного навчання до її вирішення. Як результат, була розроблена та навчена оптимальна модель глибокої нейронної мережі на базі YOLOv4-tiny. Дана мережа розпізнає людей у реальному часі та працює у парі з алгоритмом визначення їхнього положення відносно обмеженого простору. Окрім цього, була розроблена комп’ютерна система, що працює на мікрокомп’ютері NVIDIA Jetson Nano та використовує даний підхід для обробки відеопотоку з підключеної камери.

Ключові слова: *КОМП’ЮТЕРНА СИСТЕМА, РОЗПІЗНАВАННЯ ЛЮДИНИ, ВИЗНАЧЕННЯ ПОЛОЖЕННЯ ЛЮДИНИ, НЕЙРОННА МЕРЕЖА, ГЛИБИННЕ НАВЧАННЯ, YOLO, OPENCV, NVIDIA JETSON NANO.*

SUMMARY

Pages: 106

Figures: 33

Tables: 10

Sources: 36

Antoshyn Kyrylo. Computer system for determining a person’s position in a confined space in real time.

Qualification work for higher master's degree in specialty 121 — Software Engineering, supervisor Anatolii Bezverkhyi. Engineering Educational Scientific Institute of Zaporizhia National University.

The aim of the research is to study modern approaches to determining a person’s position in a confined space in real time, as well as to develop a computer system that will effectively solve this problem by working on NVIDIA Jetson Nano microcomputer and using video stream from the connected camera as input data.

In the course of the research the problem of determining a person’s position in a confined space in real time and deep learning approaches to its solution were considered. As a result, an optimal deep neural network model based on YOLOv4-tiny has been developed and trained. This network recognizes persons in real time and works in conjunction with an algorithm that determines their position in a confined space. In addition, a computer system based on the NVIDIA Jetson Nano microcomputer has been developed that uses this approach to process the video stream from the connected camera.

Keywords: *COMPUTER SYSTEM, PERSON DETECTION, PERSON POSITION DETECTION, NEURAL NETWORK, DEEP LEARNING, YOLO, OPENCV, NVIDIA JETSON NANO.*

ЗМІСТ

ВСТУП

Актуальність теми

Розвиток інформаційних технологій, зокрема штучного інтелекту, робить повсякденне життя людини більш оптимізованим та безпечним. Однією із задач, яку люди змогли успішно автоматизувати, є розпізнавання різних об’єктів у реальному часі, у тому числі людей. Можливість комп’ютерного розпізнавання людей є основою для вирішення більш складних задач, однією з яких є визначення положення людини в обмеженому просторі.

Комп’ютерна система, яка ефективно вирішує дану задачу, може бути впроваджена у різні сфери нашої життєдіяльності. Наприклад, ми можемо створити систему сповіщення, яка буде реагувати на переміщення людей у локації, що можуть бути небезпечними чи забороненими: території зі шкідливими речовинами на підприємствах, зони з приватною інформацією в органах безпеки, простір з коштовними експонатами у музеях тощо. Подібні системи сповіщення можуть бути розширені для покращення якості життя людей з обмеженими можливостями: система буде допомагати людям з вадами зору успішно переміщатися у будинку, надаючи голосові підказки в залежності від їхнього місцезнаходження. Окрім цього, ми можемо створити зони відпочинку для дітей та дорослих абсолютно нового рівня: система у торгово-розважальному центрі буде відображати зображення з проектора на підлогу та відтворювати ефекти під людьми саме у тих областях, по яких вони ходять.

Багато з існуючих комп’ютерних систем для вирішення цієї задачі потребують для своєї роботи наявності спеціалізованої інфраструктури: пристроїв, що прикріпляються до тіла людини, датчиків тощо. Такий підхід не є дешевим та не надає універсального рішення. Пристроєм, який наявний майже у кожному сучасному закладі, є камера, відеопотік з якої доцільно використовувати.

Дослідники в області штучного інтелекту створюють не тільки точні, але й швидкі та «легкі» моделі нейронних мереж (YOLOv4-tiny тощо), які можна

навчити розпізнавати об’єкти на порівняно не дорогих графічних процесорах (NVIDIA GeForce GTX 1050 тощо) та використовувати на оптимізованих для вирішення високопродуктивних задач мікрокомп’ютерах (NVIDIA Jetson Nano тощо). Такий підхід гарантує набагато нижчу ціну апаратного комплексу, а також безпеку даних, оскільки всі обчислення відбуваються локально.

Таким чином, актуальним є створення комп’ютерної системи для визначення положення людини в обмеженому просторі у реальному часі, що буде працювати на мікрокомп’ютері та використовувати в якості вхідних даних відеопотік із підключеної камери.

Мета і завдання дослідження

Мета і завдання дослідження полягають у вивченні сучасних підходів до визначення положення людини в обмеженому просторі у реальному часі, а також у розробці комп’ютерної системи, що буде ефективно вирішувати цю задачу, працюючи на мікрокомп’ютері NVIDIA Jetson Nano та використовуючи в якості вхідних даних відеопотік із підключеної камери.

Об’єкт дослідження

Об’єктом дослідження є відеопотік кадрів із камери.

Предмет дослідження

Предметом дослідження є визначення положення людини в обмеженому просторі у реальному часі на основі відеопотоку кадрів із камери.

Методи дослідження

Для вирішення поставленої задачі використовуються наступні методи дослідження:

1. Аналіз особливостей та існуючих рішень для проблеми визначення положення людини в обмеженому просторі.

2. Аналіз моделей глибоких нейронних мереж для розпізнавання об’єктів у реальному часі.
3. Аналіз бібліотек глибинного навчання та комп’ютерного зору.
4. Аналіз датасетів із зображеннями людей.
5. Аналіз методів визначення положення об’єктів відносно обмеженого простору.
6. Синтез отриманих в процесі дослідження проблеми та методів її вирішення знань.
7. Експериментування з використанням різних датасетів для навчання нейронної мережі.
8. Експериментування зі зміною параметрів роботи розробленої системи.

Наукова новизна одержаних результатів

Наукова новизна одержаних результатів дослідження полягає у тому, що для вирішення задачі визначення положення людини в обмеженому просторі у реальному часі був використаний новий та ефективний підхід, а саме: була розроблена та навчена оптимальна модель глибокої нейронної мережі для розпізнавання людини, що працює у парі з алгоритмом визначення її положення відносно обмеженого простору.

Практичне значення одержаних результатів

Практичне значення одержаних результатів дослідження полягає у тому, що була розроблена комп’ютерна система, яка у реальному часі визначає положення людей відносно області проекції на підлогу. Дана система є оптимальною з точки зору апаратних витрат, оскільки вона оброблює кадри з підключеної до мікрокомп’ютера NVIDIA Jetson Nano камери та не потребує додаткової інфраструктури для своєї роботи. З цього можна зробити висновок, що представлений у роботі підхід можливо використовувати в якості ефективного та дешевого рішення для створення подібних комп’ютерних систем.

Апробація одержаних результатів

Результати дослідження були представлені на XIII науково-практичній конференції студентів, аспірантів, докторантів і молодих вчених Запорізького національного університету «Молода наука-2020» [1], а також на XXV науково-технічній конференції студентів, аспірантів, магістрантів і викладачів Інженерного навчально-наукового інституту Запорізького національного університету [2].

Глосарій

Виявлення ключових точок людини (англ. Keypoint detection) — це проблема розпізнавання людини та виявлення ключових точок її тіла (ніг, плечей, рук тощо).

Впевненість нейронної мережі (англ. Confidence) — це значення на вихідному шарі нейронної мережі, що визначає ймовірність приналежності вхідних даних до певного класу об’єкта.

Глибинне навчання (англ. Deep learning) — це сукупність методів машинного навчання (з вчителем, з частковим залучення вчителя, без вчителя та з підкріпленням), що ґрунтуються на навчанні ознак з даних, використовуючи багатошарові нейронні мережі.

Глибока нейронна мережа (англ. Deep neural network) — це нейронна мережа з багатьма прихованими шарами.

Згорткова нейронна мережа (англ. Convolutional neural network) — це нейронна мережа прямого поширення спеціального виду, що працює на основі фільтрів, які займаються розпізнаванням характеристик певних класів об’єктів на зображенні: прямих ліній, кривих певного типу тощо.

Метод зворотного поширення помилки (англ. Backpropagation) — це ітеративний алгоритм, що є модифікацією методу градієнтного спуску, та використовується з метою мінімізації помилки роботи багатошарової нейронної мережі.

Набір даних, датасет (англ. Dataset) — це оброблена та структурована певним чином інформація, необхідна для навчання нейронної мережі.

Обмежувальна рамка (англ. Bounding box) — це прямокутник, який визначає положення розпізнаного на зображенні об’єкта.

Перетин над об'єднанням (англ. Intersection over Union) — це метрика для вимірювання точності розпізнавання об’єктів, яка математично визначається як відношення площі перетину розпізнаної та реальної обмежувальної рамок до площі їхнього об’єднання.

Перспективне перетворення (англ. Perspective transformation) — це перетворення зображення з тривимірного простору до іншого, при якому паралельні прямі сходяться, розмір об’єкта зменшується зі збільшенням відстані до центру проекції та відбувається неоднорідне спотворення ліній об’єкта, що залежить від орієнтації та відстані від об’єкта до центру проекції.

Помилка нейронної мережі (англ. Loss) — це значення, що визначається функцією помилки, відповіддю якої є дійсне число, що характеризує якість вирішення певної задачі нейронною мережею.

Придушення немаксимумів (англ. Non-maximum suppression) — це алгоритм, який застосовується для вибору найкращої обмежувальної рамки серед інших навколо розпізнаного об’єкта.

Розмір батча навчання нейронної мережі (англ. Batch size) — це гіперпараметр алгоритму навчання нейронної мережі, що визначає кількість прикладів навчальної вибірки, які повинні бути подані на вхід перед оновленням її внутрішніх параметрів.

Розпізнавання об’єктів (англ. Object detection) — це визначення положення об’єктів на зображенні за допомогою обмежувальних рамок, а також вірогідності їхньої приналежності до того чи іншого класу.

Точність розпізнавання об’єктів (англ. Mean average precision) — це метрика для вимірювання середньої точності детекторів об’єктів, яка математично визначається як площа області під кривою графіка залежності точності (precision) від відкликання (recall).

Трансферне навчання (англ. Transfer learning) — це тип навчання нейронної мережі, що дозволяє використовувати отримані при вирішенні однієї задачі знання для розв'язку аналогічної проблеми.

Функція активації штучного нейрону (англ. Activation function) — це залежність вихідного сигналу штучного нейрону від зваженої суми його вхідних сигналів.

Штучна нейронна мережа (англ. Artificial neural network) — це обчислювальна система, побудована за принципом організації та функціонування біологічних нейронних мереж, що складається зі штучних нейронів та зв'язків між ними — синапсів. Вона навчається розв'язувати задачу з прикладів: анотованих зображень, оброблених текстів тощо.

Штучний нейрон (англ. Artificial neuron) — це спрощена модель природного нейрону, що є складовою частиною штучної нейронної мережі, та математично представлений як деяка нелінійна функція від лінійної комбінації його вхідних сигналів.

CMake — це кросплатформна система генерації файлів, необхідних для компіляції програмного забезпечення із вихідного коду.

COCO — це масштабний датасет для навчання нейронної мережі вирішувати одну з наступних задач: розпізнавання об'єктів, виявлення ключових точок та оцінка поз людей, сегментація об'єктів та генерація текстового опису до зображень. Він включає у себе 80 класів об'єктів із анотаціями для кожної з перерахованих задач.

CUDA — це програмно-апаратна архітектура паралельних обчислень, яка дозволяє істотно збільшити продуктивність комп'ютерної системи завдяки паралельній роботі програмного коду на графічних процесорах NVIDIA.

cuDNN — це бібліотека примітивів для глибоких нейронних мереж, що є частиною CUDA.

Darknet — це відкритий фреймворк для глибинного навчання, який головним чином використовується для моделей YOLOv2, YOLOv3, YOLOv4, а також їхніх зменшених версій.

Keras — це відкритий фреймворк для глибинного навчання, розроблений з акцентом на швидке експериментування, що є надбудовою над бібліотекою TensorFlow.

NVIDIA Jetson Nano — це мікрокомп'ютер для вирішення задач в області штучного інтелекту, що підтримує безліч фреймворків та бібліотек, які використовуються у цій галузі, дозволяє створювати вбудовані високопродуктивні системи та має невисоку вартість.

OpenCV — це відкрита бібліотека комп'ютерного зору, що включає у себе алгоритми для обробки та перетворення зображень, функціонал для роботи з відео, а також модуль для використання глибоких нейронних мереж класифікації, розпізнавання, виявлення ключових точок та сегментації об'єктів на зображенні.

Open Images Dataset v4 — це масштабний датасет для навчання нейронної мережі розпізнавати чи класифікувати об'єкти. Він включає у себе 600 класів об'єктів для розпізнавання та 19794 класів об'єктів для класифікації.

Python — це інтерпретована об'єктно-орієнтована мова програмування високого рівня із динамічною типізацією.

PyTorch — це відкритий фреймворк для машинного навчання, що базується на бібліотеці Torch та використовується для вирішення багатьох задач штучного інтелекту: розпізнавання об'єктів, обробка природньої мови тощо.

Qt — це кросплатформна бібліотека, що містить класи для розробки прикладного програмного забезпечення: створення графічного інтерфейсу користувача, роботи з базами даних, потоками тощо.

TensorFlow — це відкрита бібліотека для машинного навчання. Вона може бути використана для широкого кола задач, але особлива увага приділяється навчанню та використанню глибоких нейронних мереж.

YOLO — це передова модель глибокої нейронної мережі для розпізнавання об'єктів у реальному часі. На даний момент її остання версія — YOLOv4. YOLO входить до складу фреймворка Darknet, але також існують реалізації за допомогою інших бібліотек.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ ВИЗНАЧЕННЯ ПОЛОЖЕННЯ ЛЮДИНИ В ОБМЕЖЕНОМУ ПРОСТОРІ У РЕАЛЬНОМУ ЧАСІ

1.1 Огляд проблеми визначення положення людини в обмеженому просторі у реальному часі

Райнер Маутц у дослідженні [3] проводить детальний огляд проблеми визначення положення об’єктів, у тому числі людей, в обмеженому просторі та наводить наявні технології для її вирішення. Також він описує сфери життєдіяльності, у яких можуть бути використані комп’ютерні системи, що ефективно вирішують дану задачу.

Після досягнень у розробці сервісів для визначення положення об’єктів у відкритому просторі на базі супутникових систем навігації з’явилася проблема вирішення цієї задачі для обмеженого простору. Можливість визначати положення об’єктів в обмеженому просторі є значною проблемою, що запобігає вирішенню задачі визначення положення об’єктів у будь-якому середовищі. Багато сфер життєдіяльності потребують оптимального технічного рішення даної проблеми. Покращення ефективності рішень визначення положення в обмеженому просторі має потенціал для створення безпрецедентних можливостей для бізнесу.

Райнер Маутц стверджує, що поки не існує цілісного рішення для цієї проблеми, яке б використовувало єдину технологію та працювало з точністю до 1 метра. Як результат, вимоги до кожного застосунку повинні бути проаналізовані окремо. Проблему ускладнюють два факти. По-перше, кількість вимог до таких систем досить велика: висока точність та швидкість роботи, висока доступність, низька ціна тощо. По-друге, на сьогодні існує велика кількість технологій, які можна використовувати для вирішення цієї задачі. Більшість з них потребують наявності спеціалізованої інфраструктури, що збільшує витрати для замовників подібних систем.

Виникає питання: чому взагалі існує різниця між визначенням положення об’єктів у відкритому та в обмеженому просторі? Райнер Маутц стверджує, що більшість систем для визначення положення об’єктів теоретично можуть бути використані як у відкритому, так і в обмеженому просторі, однак їхня точність буде відрізнятися у цих двох середовищах через ряд істотних відмінностей. Він виділяє причини, які ускладнюють та спрощують задачу визначення положення об’єктів в обмеженому просторі.

Визначення положення об’єктів в обмеженому просторі є складною задачею через ряд наступних причин (наведені причини не відносяться до систем вирішення даної проблеми, що використовують камери):

1. Сильна багатопроменевість через відбиття сигналу від стін та меблів.
2. Відсутність прямої видимості (Non-Line-of-Sight).
3. Високе ослаблення та розсіювання сигналу через велику щільність перешкод.
4. Швидкі часові зміни через присутність людей та відкривання дверей.
5. Необхідність високої точності роботи системи.

З іншого боку, обмежений простір спрощує задачу визначення положення об’єктів через ряд наступних причин (причини 1, 4 та 5 відносяться до систем вирішення даної проблеми, що використовують камери):

1. Невеликі зони покриття.
2. Низькі погодні впливи, такі як малі температурні градієнти та повільна циркуляція повітря.
3. Фіксовані геометричні обмеження: плоскі поверхні та ортогональні стіни.
4. Наявність наступної інфраструктури: електрика, доступ до Інтернету та стіни, придатні для встановлення системи.
5. Нижча динаміка рухомих об’єктів.

Райнер Маутц заявляє, що проблема визначення положення об’єктів в обмеженому просторі все частіше стає предметом досліджень, оскільки супу-

тникові системи навігації, які застосовуються для позиціонування у відкритому просторі, погано працюють у будівлях. Тобто, на сьогодні ми не маємо єдиної системи для визначення положення об’єктів в обмеженому просторі, яка б забезпечувала високу точність, коротку затримку, високу доступність та низькі витрати для кінцевих користувачів.

1.2 Технології для визначення положення людини в обмеженому просторі у реальному часі

Райнер Маутц у роботі [3] виділяє 13 технологій для визначення положення об’єктів, у тому числі людей, в обмеженому просторі. Ці технології використовують 3 різні фізичні принципи: інерціальна навігація (акселерометри та гіроскопи, що підтримують кутовий момент), механічні хвилі (звукові та ультразвукові) та електромагнітні хвилі (видимий, інфрачервоний, мікрохвильовий та радіо спектри). Також ці технології відрізняються за точністю роботи та областю покриття, що відображено у таблиці 1.

Таблиця *1*

Порівняння технологій для визначення положення об’єктів в обмеженому просторі за точністю роботи та областю покриття

Технологія	**Точність**	**Покриття (м)**
Камери	0.1 мм – 1 дм	1 – 10
Інфрачервоне випромінювання	1 см – 1 м	1 – 5
Тактильні та комбіновані полярні системи	1 мкм – 1 мм	3 – 2000
Звук	1 см	2 – 10
Wi-Fi	1 м	20 – 50
RFID	1 дм – 1 м	1 – 50

Технологія	Точність	Покриття (м)
Ultra-Wideband	1 см – 1 м	1 – 50
Високочутливі супутникові системи навігації	10 м	Глобальні
Псевдоліти	1 см – 1 дм	10 – 1000
Інші радіочастоти (ZigBee, Bluetooth тощо)	1 м	10 – 1000
Інерціальна навігація	1 %	10 – 100
Магнітні системи	1 мм – 1 см	1 – 20
Інфраструктурні системи	1 см – 1 м	Будівля

На рисунку 1 зображений графік залежності області покриття цих технологій від точності роботи, який наводить Райнер Маутц.

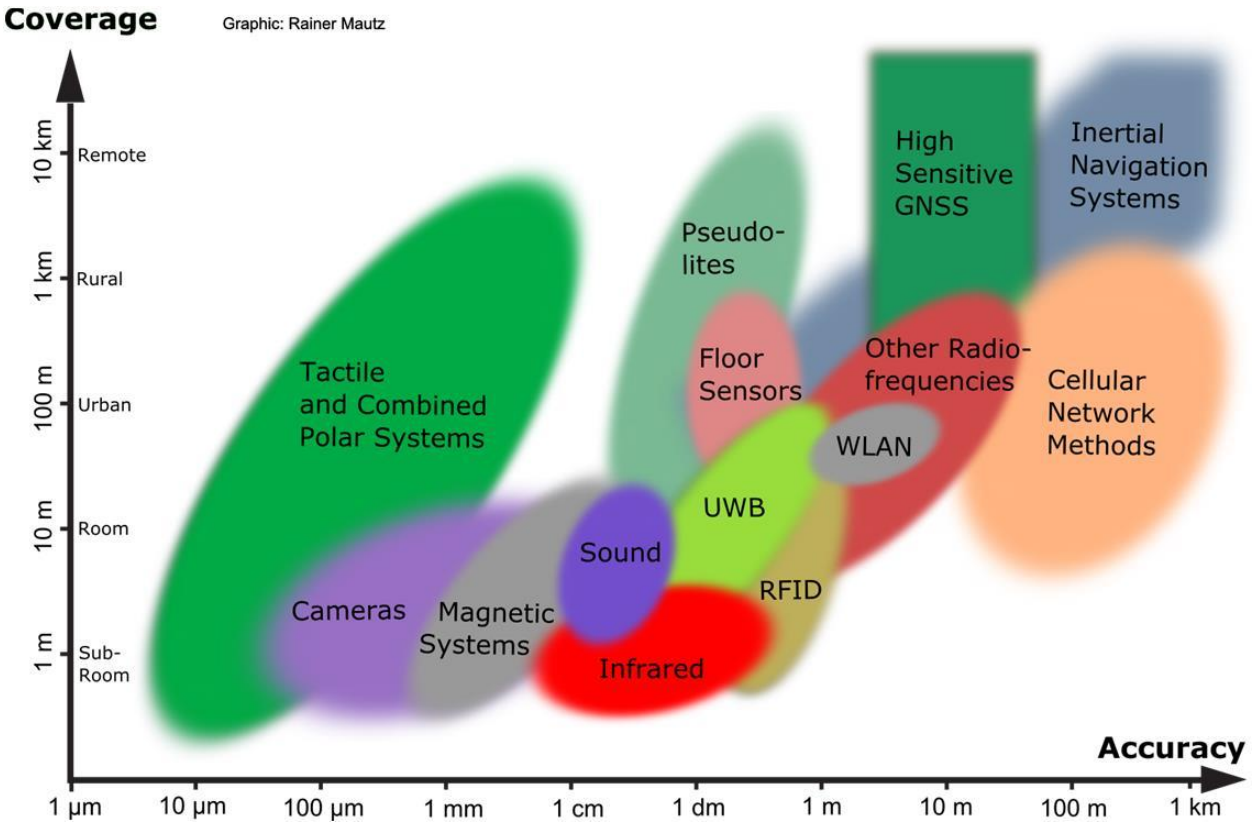

Рис. *1 Графік залежності області покриття технологій для визначення положення об'єктів в обмеженому просторі від точності роботи*

Виходячи з даних, представлених у таблиці та на графіку, можна побачити, що високочутливі супутникові системи навігації дають найменшу точність серед усіх технологій. Найкращим, на перший погляд, рішенням є використання тактильних та комбінованих полярних систем, однак через високу ціну ця технологія не може бути використана для створення доступних комп'ютерних систем для вирішення даної проблеми. Оптимальним є використання Wi-Fi, RFID, Ultra-Wideband та інших радіочастот, але такі рішення потребують для своєї роботи встановлення додаткової інфраструктури, носіння датчиків тощо. Камери дають найкращу точність (від 0.1 мм до 1 дм), але комп'ютерні системи на основі даної технології можуть працювати лише на рівні кімнати (від 1 до 10 м).

Сьогодні використання камер є ефективним рішенням для проблеми визначення положення людини в обмеженому просторі через дві основні причини. По-перше, камери наявні майже у кожній сучасній будівлі, а й отже кінцевим замовникам подібних систем не потрібно витрачати зайві кошти на придбання додаткової інфраструктури. По-друге, завдяки розвитку штучного інтелекту ми навчилися розпізнавати об'єкти у реальному часі, а й отже можемо використовувати цю інформацію для визначення їхнього положення в обмеженому просторі.

1.3 Сфери застосування визначення положення людини в обмеженому просторі у реальному часі

Вирішення проблеми визначення положення людини в обмеженому просторі у реальному часі активно впроваджується у різні сфери нашої життєдіяльності. Райнер Маутц у дослідженні [3] наводить вичерпний список із цих сфер, який демонструє потребу у створенні комп'ютерних систем для вирішення даної проблеми. Також він стверджує, що поряд із покращенням ефективності, майбутні системи для визначення положення об'єктів в обмеженому просторі знайдуть ще більше можливих варіантів використання.

Нижче наведені деякі зі сфер, описаних Райнером Маутцом, для яких можна створити комп'ютерну систему визначення положення людини в обмеженому просторі у реальному часі, яка б використовувала у якості вхідних даних відеопотік із підключеної камери. Частина цих сфер потребує також вирішення задачі ідентифікації розпізнаної людини.

Сервіси на основі положення (Location-Based Services) в приміщеннях є комерційно значимими для масового ринку. Такі сервіси використовують положення людини для надання контекстно-залежної інформації, що доступна з мобільного пристрою. Прикладом такого сервісу може слугувати система для направлення відвідувачів до потрібних їм стендів чи залів на великих виставках чи форумах: люди, які знаходяться далеко від місця початку доповіді, отримають повідомлення на смартфон з інформацією, куди вони повинні підійти. Подібні рішення представляють високу комерційну цінність, адже вони можуть бути використані для реклами послуг, якими можна скористатися у закладі. Іншим прикладом може слугувати система автоматизованого входу чи виходу на підприємствах: система розпізнає, який співробітник прийшов до офісу, а який вийшов із нього, та зберігає цю інформацію у базі даних.

Застосування комп'ютерних систем для визначення положення людини у приватних будинках включає у себе отримання різноманітних послуг. Наприклад, системи Ambient Assisted Living допомагають людям похилого віку при повсякденній активності, використовуючи їхнє поточне положення у будинку. Іншими прикладами застосування у будинку є моніторинг життєвих знаків людини, виявлення надзвичайних ситуацій та падіння людини, а також створення розумних розважальних систем.

У лікарнях все більшого значення набуває необхідність відстеження місцезнаходження персоналу при надзвичайних ситуаціях. Застосування у медицині також включає у себе відстеження пацієнтів, наприклад, для виявлення падіння. Точне позиціонування також потрібно для роботизованої допомоги хірургам під час операцій.

Визначення положення людини в обмеженому просторі для сфери безпеки включає у себе створення систем сповіщення, які будуть реагувати на переміщення людей у локації, що можуть бути небезпечними чи забороненими: території зі шкідливими речовинами на підприємствах, зони з приватною інформацією в органах безпеки, простір з коштовними експонатами у музеях тощо. Оскільки подібна система буде використовувати камеру, інформація, отримана з неї, може бути використана під час слідства.

Існує декілька варіантів використання подібних систем у музеях: відстеження відвідувачів для спостереження та вивчення їхньої поведінки, навігація, а також надання інформаційних сервісів на основі місцезнаходження. Власники музею можуть проаналізувати дані місцезнаходження відвідувачів для того, щоб зрозуміти, які експонати приваблюють їх найбільше, або визначити, які відвідувачі знаходяться у забороненій зоні. Тобто, використання у музеях включає у себе надання сервісів на основі положення та забезпечення безпеки.

Системи для визначення положення людини в обмеженому просторі, які націлені на надання допомоги людям з обмеженими можливостями, можуть допомогти людям з вадами зору успішно переміщатися усередині будівлі: система буде надавати голосові підказки в залежності від їхнього місцезнаходження. Такі системи можна об’єднати з системами визначення положення у відкритому просторі для забезпечення безперебійної навігації людям з вадами зору.

Визначення положення людини в обмеженому просторі також використовується для створення інтерактивних систем нового покоління, у яких потрібно відстежувати жести чи частини тіла людини. Наприклад, для створення системи Interactive Floor («Інтерактивна підлога») нам потрібно знати точне положення людей на підлозі (в області проекції), щоб відтворювати ефекти саме у тих областях, по яких вони ходять. На рисунку 2 зображений приклад роботи цієї системи.

Рис. *2 Робота системи Interactive Floor*

1.4 Рішення для визначення положення людини в обмеженому просторі у реальному часі

Достатньо повний огляд рішень для проблеми визначення положення людини в обмеженому просторі у реальному часі представлений у статті Адріана Косма та інших дослідників [4]. Вони стверджують, що усі існуючі рішення для визначення положення людини в обмеженому просторі можна поділити на дві групи: рішення, які потребують спеціалізованої інфраструктури для своєї роботи та рішення, що використовують наявну інфраструктуру: бездротові точки доступу або камери спостереження у будівлях та інерціальні датчики у мобільних пристроях.

Не зважаючи на те, що більшість рішень другої групи використовують сенсори смартфонів та Wi-Fi, також існують системи, що використовують відеопотік з камер. Системи на основі камер не вимагають від кінцевих користувачів носіння спеціальних датчиків, що набагато спрощує використання цих систем у випадках, коли користувачі недостатньо «підковані» у технічному плані (наприклад, у сценаріях Ambient Assisted Living, користувачами яких є

літні люди). Такі системи використовують алгоритми комп'ютерного зору для обробки відеопотоку з підключеної камери.

Дослідники у [4] наводять список рішень для визначення положення людини в обмеженому просторі, що використовують методи комп'ютерного зору. Райнер Маутц та Себастьян Тільх опублікували дослідження оптичних систем для позиціонування в обмеженому просторі [5]. Вони описали та класифікували оптичні системи на основі даних, які використовуються для визначення положення людини: зображення, проектовані шаблони та кодовані маркери. Багато з існуючих систем для вирішення даної проблеми використовують камери глибини (RGB-D). Так, наприклад, дослідники у [6] створили систему, що використовує відеопотік з двох підключених камер глибини Kinect для знаходження положення людини з можливою похибкою до 14 см. Для вирішення задачі вони використали функціонал відстеження скелета, наявний у Kinect SDK. Дослідники у [7] створили систему, яка вирішує дану проблему, використовуючи камери глибини Kinect у поєднанні з технологією Wi-Fi, та виявили значне покращення точності та продуктивності відносно попередніх рішень на основі Wi-Fi.

Самі ж дослідники у [4] розробили комп'ютерну систему, що використовує в якості вхідних даних відеопотік із RGB камери. Система оброблює відеопотік за допомогою глибокої нейронної мережі для оцінки ключових точок людини. Отримані дані використовуються для визначення її положення в обмеженому просторі. Положення людини — це середня точка між її ногами, трансформована з перспективи камери відносно підлоги. Середня похибка роботи їхньої системи становить 36 см, а швидкість — 6.25 FPS. Також дослідники порівняли свій метод з алгоритмом виявлення об'єктів YOLOv2 та виявили кращі показники з точки зору швидкості та використання пам'яті. Такий результат є досить очевидним, оскільки детектор YOLOv2 не здатен працювати на пристроях з обмеженими ресурсами. На рисунку 3 наведений приклад роботи системи CamLoc.

Рис. *3 Робота системи CamLoc*

Виходячи з огляду існуючих рішень на основі комп'ютерного зору, можна побачити, що більшість з них потребують для своєї роботи наявності камер глибини, які не можна вважати за широкодоступну інфраструктуру у будівлях. Найбільш оптимальне рішення з точки зору використання інфраструктури, обчислювальних ресурсів та точності роботи було представлено у статті [4]. Цей підхід використовує глибокі нейронні мережі, які на сьогодні показують найкращі результати при розпізнаванні об'єктів. Недоліком цього рішення є те, що система працює лише для випадку однієї людини у кадрі.

Дослідники в області штучного інтелекту створюють не тільки точні, але й швидкі та «легкі» моделі нейронних мереж (YOLOv4-tiny тощо), які можна використовувати на дешевих та оптимізованих для вирішення високопродуктивних задач мікрокомп'ютерах (NVIDIA Jetson Nano тощо). Отже, ми можемо створити комп'ютерну систему, яка буде розпізнавати людей у відеопотоці з підключеної RGB-камери за допомогою глибокої нейронної мережі, а потім визначати їхнє положення відносно обмеженого простору. Таке рішення є оптимальним не тільки з точки зору точності та швидкості роботи, а й з точки зору вартості для кінцевих користувачів, оскільки камери спостереження наявні майже у кожному сучасному закладі.

1.5 Сучасні підходи комп’ютерного зору до визначення положення людини в обмеженому просторі у реальному часі

Для визначення положення людини в обмеженому просторі за допомогою засобів комп’ютерного зору необхідно послідовно вирішити дві задачі. По-перше, необхідно розпізнати людину у кадрі. По-друге, треба визначити її положення з перспективи камери відносно обмеженого простору (підлоги). Для вирішення першої задачі ефективно використати глибоку нейронну мережу, як було описано вище. Дана мережа може бути навчена визначати ключові точки людини чи знаходити обмежувальну рамку навколо неї. За положення людини можна вважати середню точку між її ногами чи середню точку нижньої грані обмежувальної рамки, що трансформована з перспективи камери відносно підлоги. Визначення ключових точок людини є більш точним підходом, оскільки, якщо людина стане боком та відставить одну ногу назад, ми не втратимо цю інформацію. Дана ситуація яскраво проілюстрована на рисунку 3. Недоліком цього підходу є більша кількість кадрів, у яких нейронна мережа не може виявити людину, що описано у [4].

1.5.1 Ефективність підходів глибинного навчання до розпізнавання об’єктів

Може виникнути питання: чому підходи глибинного навчання для розпізнавання об’єктів, є настільки ефективними у порівнянні з класичними методами комп’ютерного зору? Це питання розкривається у дослідженні [8].

Традиційним підходом комп’ютерного зору до розпізнавання об’єктів є використання дескрипторів ознак (HOG [9] та інших). До появи глибинного навчання для вирішення таких задач, як класифікація зображень, виконувався крок вилучення ознак. Ознаки — це невеликі інформативні області на зображеннях, які однозначно описують той чи інший об’єкт. На цьому кроці можуть бути задіяні декілька алгоритмів комп’ютерного зору: виявлення країв, виявлення кутів чи сегментація порогових значень. Вилучені із зображень ознаки

формують визначення (опис) об’єктів кожного класу. Дескриптор ознак намагається знайти ці визначення на зображеннях, які подаються йому на вхід. Якщо значна кількість певних ознак присутня на вхідному зображенні, він класифікує його як таке, що містить об’єкт певного класу. Наприклад, для дескриптора ознак на основі гістограм напрямлених градієнтів (HOG) ознаками є розподіл градієнтних векторів (їх напрям та величина) на зображенні.

Складність традиційного підходу полягає в тому, що інженер повинен обрати, які саме ознаки найкраще описують той чи інший клас об’єктів. Це тривалий процес спроб та помилок. Зі збільшенням кількості класів об’єктів вилучення ознак стає більш громіздким. Більше того, процес вилучення ознак використовує безліч параметрів, кожен із яких повинен бути точно налаштований інженером.

Глибинне навчання — це концепція наскрізного навчання, у якій модель глибокої нейронної мережі навчається виявляти основні закономірності різних класів об’єктів на навчальній вибірці. Тобто, нейронна мережа не програмується під конкретну задачу, а навчається її вирішувати, як це роблять люди. Добре встановлено, що глибокі нейронні мережі працюють набагато краще, ніж традиційні алгоритми комп’ютерного зору, хоча і з компромісами щодо обчислювальних вимог та часу, необхідного для навчання моделі. Зважаючи на це, можна констатувати, що процес роботи інженера в області комп’ютерного зору кардинально змінився: знання та досвід у вилученні створених «вручну» ознак об’єктів були замінені знаннями та досвідом вибору чи створення найкращої архітектури нейронної мережі для вирішення задачі.

Розробка архітектури нейронної мережі, яка навчається вилучати ознаки (характеристики) об’єктів на зображенні, а саме згорткової нейронної мережі (Convolutional Neural Network), призвела до прориву у вирішенні задачі класифікації об’єктів. Архітектура згорткової нейронної мережі була описана ще у 1998 році Яном Лекуном та його командою у статті [10], але її активне використання почалося лише з 2012 року, коли інженери у дослідженні [11] розро-

били та навчили модель глибокої згорткової нейронної мережі AlexNet класифікувати 1.2 мільйони зображень на 1000 різних класів. Стрімкий розвиток методів глибинного навчання для класифікації, а пізніше — для розпізнавання об’єктів, обумовлений зростанням обчислювальної потужності та збільшенням обсягу даних, необхідних для навчання нейронних мереж. Принцип роботи згорткової нейронної мережі розглянутий у розділі 2 кваліфікаційної роботи.

1.5.2 Задача розпізнавання людини у реальному часі

Рос Гіршик та його команда представили у 2013 році один із перших методів розпізнавання об’єктів, у тому числі людей, з використанням глибокої нейронної мережі — R-CNN [12]. Підхід полягає у генерації приблизно 2000 регіонів інтересу на зображенні за допомогою алгоритму вибіркового пошуку (Selective Search). Приведені до однакового розміру регіони подаються на вхід до згорткової нейронної мережі для вилучення ознак. Після цього використовується метод опорних векторів (Support vector machine) та алгоритм визначення обмежувальної рамки (Bounding-box regression) для розпізнавання об’єктів. Головним недоліком методу R-CNN є подання на вхід нейронній мережі кожного регіону інтересу окремо. Розпізнавання об’єктів на зображенні за допомогою R-CNN займає 49 секунд. Схематично робота цього методу зображена на рисунку 4.

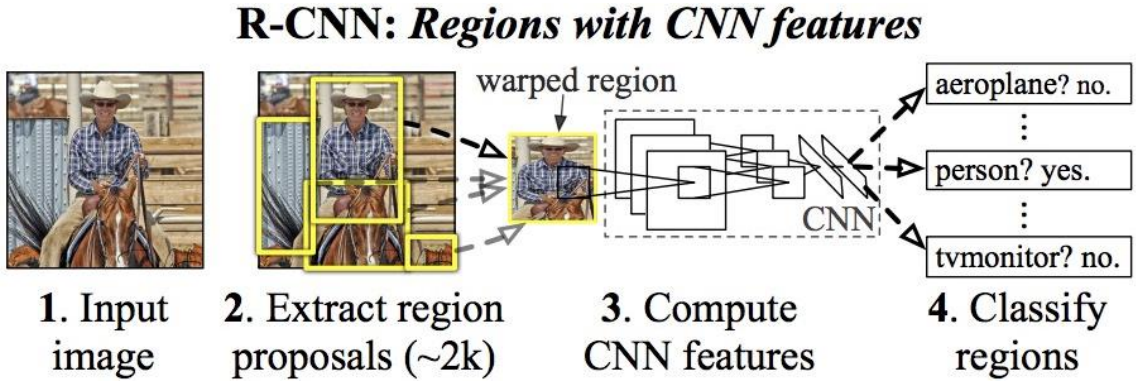

Рис. *4 Розпізнавання об’єктів за допомогою методу R-CNN*

У 2015 році Рос Гіршик створив покращену версію методу R-CNN — Fast R-CNN [13]. Замість того, щоб подавати кожен регіон інтересу до згорткової нейронної мережі, ми спочатку подаємо зображення цілком та вилучаємо характеристики. Потім ми застосовуємо алгоритм вибіркового пошуку для вилучення регіонів інтересу, які приводяться до одного розміру у шарі RoI Pooling нейронної мережі та подаються до повністю пов'язаного шару для визначення класу об'єкта та обмежувальної рамки. До недоліків цього методу відноситься використання алгоритму вибіркового пошуку для вилучення регіонів інтересу, оскільки він є повільним. Fast R-CNN розпізнає об'єкти на зображенні за 2.3 секунди. Схематично робота цього методу зображена на рисунку 5.

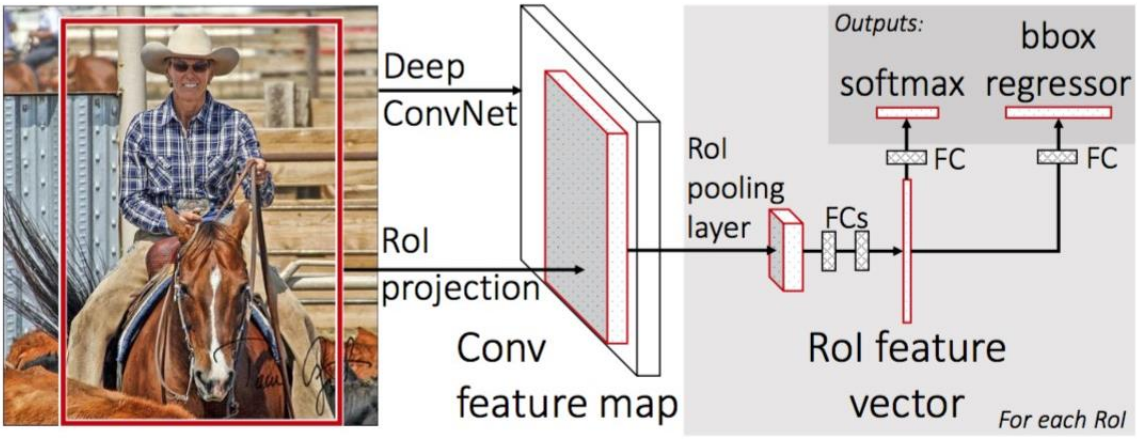

Рис. *5 Розпізнавання об'єктів за допомогою методу Fast R-CNN*

У цьому ж році Шаокін Рен та його команда створили покращену версію методу Fast R-CNN — Faster R-CNN [14]. Для визначення регіонів інтересу на зображенні замість алгоритму вибіркового пошуку даний підхід пропонує використання окремої нейронної мережі — Region Proposal Network. Faster R-CNN дозволяє розпізнавати об'єкти на зображенні за 200 мс (5 FPS), що є не достатнім для використання у режимі реального часу. Схематично робота цього методу зображена на рисунку 6.

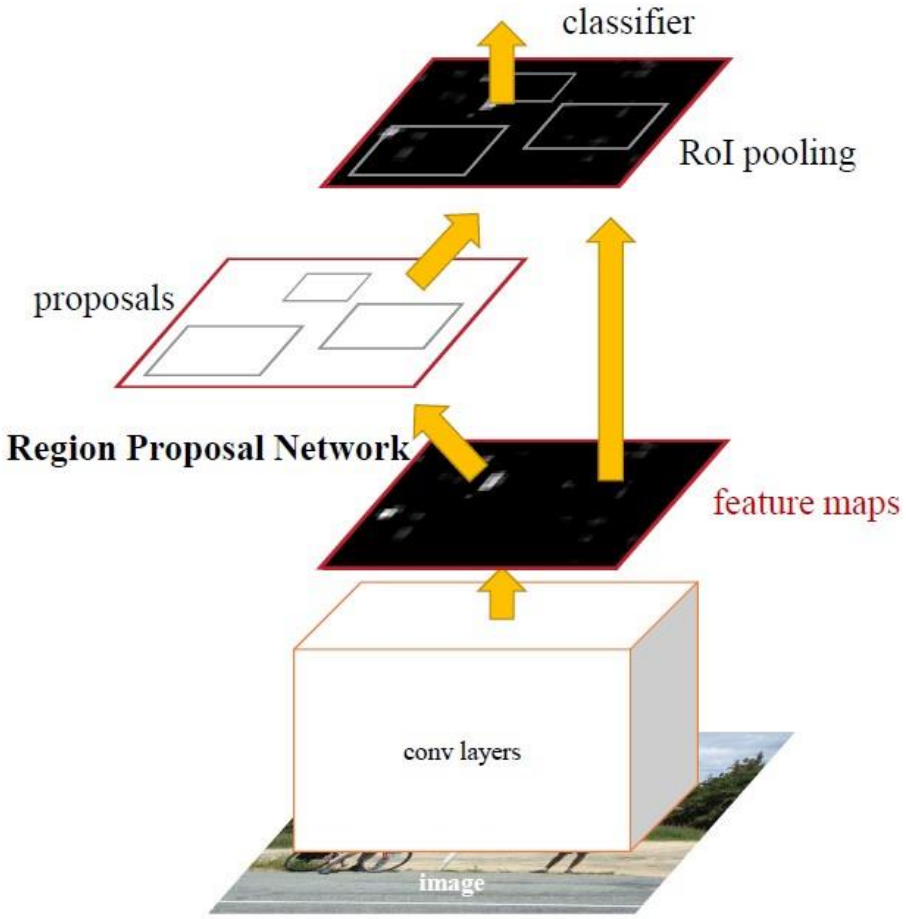

Рис. *6 Розпізнавання об'єктів за допомогою методу Faster R-CNN*

Усі методи сімейства R-CNN мають два недоліки. По-перше, алгоритм потребує багато проходів через одне зображення, щоб розпізнати на ньому об'єкти. По-друге, оскільки різні системи працюють одна за одною, швидкість роботи систем на вищому рівні залежить від швидкості систем на нижчому. У таблиці 2 наведено порівняння методів сімейства R-CNN за швидкістю та точністю роботи, взяте з оригінальних статей.

Таблиця *2*

Порівняння методів сімейства R-CNN за швидкістю та точністю роботи

Критерій / Метод	**R-CNN**	**Fast R-CNN**	**Faster R-CNN**
Швидкість (с)	49	2.3	0.2
Точність (%, VOC 2007)	66.0	66.9	66.9

Джозеф Редмон та його команда запропонували у 2015 році набагато швидший, але менш точний підхід — YOLO [15]. Розроблений ними підхід розпізнає об’єкти на зображенні за один прохід. Оскільки це усуває потребу в конвеєрі розпізнавання, який використовували методи сімейства R-CNN, систему можна оптимізувати в цілому. YOLO розпізнає об’єкти зі швидкістю 45 FPS. У 2020 році вийшла четверта версія цієї системи — YOLOv4 [16]. Існують також зменшені версії YOLO, наприклад YOLOv4-tiny, що мають меншу точність, але можуть працювати у реальному часі на мікрокомп’ютерах, наприклад NVIDIA Jetson Nano. Принципи роботи YOLO, а також деяких інших сучасних детекторів для розпізнавання об’єктів у реальному часі на пристроях з обмеженими ресурсами, розглянуті у розділі 2 кваліфікаційної роботи.

Задача розпізнавання людей за допомогою методів глибинного навчання має наступні складнощі:

1. Кімната, у якій встановлена камера, може мати недостатнє освітлення.
2. Люди носять різний одяг взимку та влітку.
3. Люди можуть приймати різні пози.
4. Можуть виникати ситуації перекриття людини різними об’єктами: людьми, меблями тощо.
5. У кімнаті можуть знаходитися неживі об’єкти, схожі на людей: манекени, статуї тощо.

Наведені вище проблеми вирішуються навчанням нейронної мережі на якісному датасеті, який містить зображення для кожної ситуації. Найкращим варіантом є створення власного датасета на локації, де встановлена камера, але такий процес вимагає багато часу.

1.5.3 Задача визначення положення розпізнаної людини відносно обмеженого простору

Після того, як глибока нейронна мережа розпізнала людину у кадрі, необхідно визначити її координату відносно обмеженого простору (підлоги) з

перспективи камери. Як уже було описано раніше, за положення людини можна прийняти центральну точку нижньої грані обмежувальної рамки, що трансформована відносно підлоги. Виникає питання: як виконати таку трансформацію? Розглянемо цю проблему з точки зору визначення положення розпізнаної людини відносно області проекції на підлогу. Для її вирішення ми повинні обрати на відеопотоці з камери 4 точки області проекції на підлогу, 4 вершини зображення на проекторі та обчислити матрицю для перспективного перетворення. Після цього ми зможемо трансформувати будь-яку точку в області проекції з перспективи камери. Реалізація цього підходу розглянута у розділі 3 кваліфікаційної роботи. Приклад перспективного перетворення зображений на рисунку 7.

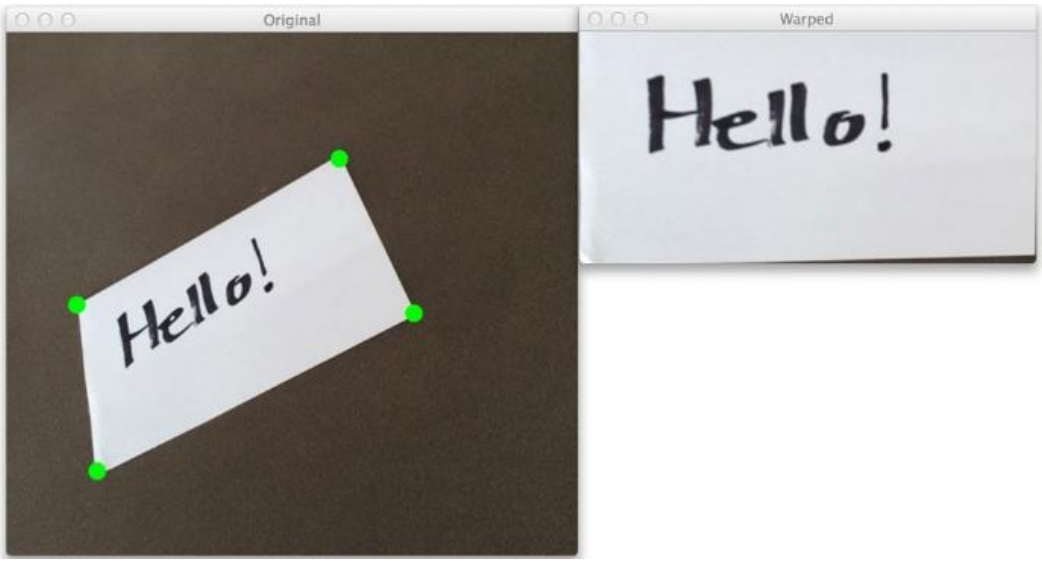

Рис. *7 Перспективне перетворення зображення*

1.6 Аналіз програмного забезпечення для визначення положення людини в обмеженому просторі у реальному часі

1.6.1 Рішення від компанії infsoft

infsoft — це німецька компанія, що пропонує комплексні рішення для визначення положення об’єктів, у тому числі людей, в обмеженому просторі.

Також увага компанії зосереджена на системах надання сервісів на основі положення з функціями аналітики та відстеження. У якості технологій для позиціонування вони використовують Bluetooth, Wi-Fi, Ultra-Wideband, RFID та камери. Одним із варіантів використання камер, зазначений на сайті компанії, є розпізнавання номерних знаків автомобілів для повідомлення працівників закладу при прибуття того чи іншого клієнта. Замовниками компанії є Audi, Siemens, Swisscom тощо.

Системи позиціонування від infsoft використовують потужні програмні засоби, вбудовані у платформу infsoft LocAware. Для визначення положення в обмеженому просторі компанія розробила власні апаратні рішення: infsoft Locator Nodes, infsoft Locator Beacons та infsoft Locator Tags.

Одним із рішень компанії є infsoft Wayfinding. Воно представляє собою мобільний застосунок, який допомагає людям при навігації у лікарнях, аеропортах, офісах тощо. Для реалізації даного рішення по всій будівлі встановлюють маяки Bluetooth (infsoft Locator Beacon або infsoft BLE Beacon). Вони передають Bluetooth-повідомлення, які приймаються смартфонами користувачів. На підставі отриманих даних застосунок визначає поточне положення людини. Робота застосунку представлена на рисунку 8.

Рис. *8 Рішення Wayfinding від infsoft*

Застосунок Wayfinding використовує infsoft SDK — бібліотеку для впровадження рішень визначення положення в обмеженому просторі в існуючі застосунки. Дана бібліотека доступна для створення додатків під операційні системи Android та iOS за допомогою нативних засобів, а також існують реалізації для фреймворків Xamarin та PhoneGap. infsoft SDK складається з наступних компонентів:

1. Maps Library — бібліотека для інтеграції векторних 2D та 3D карт приміщення.
2. Locator Library — бібліотека для визначення положення в обмеженому просторі, яка дозволяє використовувати GPS, Wi-Fi, Bluetooth та сенсори смартфона.
3. Routes Library — бібліотека для побудови маршрутів при навігації усередині будівлі з підтримкою вказівок.
4. GeoObjects Library — бібліотека для візуалізації місць усередині будівлі (ліфта, сходів тощо) з можливістю отримання додаткової інформації.

До переваг рішень від компанії infsoft можна віднести:

1. Надання повного апаратно-програмного комплексу для вирішення проблеми визначення положення в обмеженому просторі.
2. Можливість використання різного апаратного забезпечення для одного й того ж рішення.
3. Наявність SDK для впровадження системи в існуючі мобільні застосунки.

До недоліків рішень від компанії infsoft можна віднести:

1. Рішення компанії в основному зосереджені на використанні додаткової інфраструктури замість наявної, наприклад, камер спостереження.

1.6.2 Рішення від компанії GiPStech

GiPStech — це італійська компанія, що пропонує рішення для визначення положення об’єктів, у тому числі людей, в обмеженому просторі з використанням геомагнітних та інерціальних алгоритмів. Окрім цього, компанія використовує Bluetooth, Wi-Fi та камери. На сайті GiPStech зазначено, що камери використовуються з метою доповнення рішення на основі інерціальних алгоритмів. Їхні алгоритми досягають виняткової точності (у середньому 1 м). Окрім цього, компанія пропонує рішення для навігації в обмеженому просторі, а також аналізу поведінки користувачів. Замовниками компанії є Vodafone, Daimler, HERE тощо.

Одним із рішень компанії є система для визначення положення відвідувачів музею. Функціонально вона складається з 2 частин:

1. Мобільний додаток, встановлений на смартфонах відвідувачів, який дозволяє отримувати інформацію про їхнє поточне положення, а також цікаву інформацію про експонати.
2. Хмарна веб-платформа, що дозволяє працівникам музею управляти картою музею та контентом, що відображається на мобільних пристроях відвідувачів, а також проводити аналіз поведінки відвідувачів.

Робота системи представлена на рисунку 9.

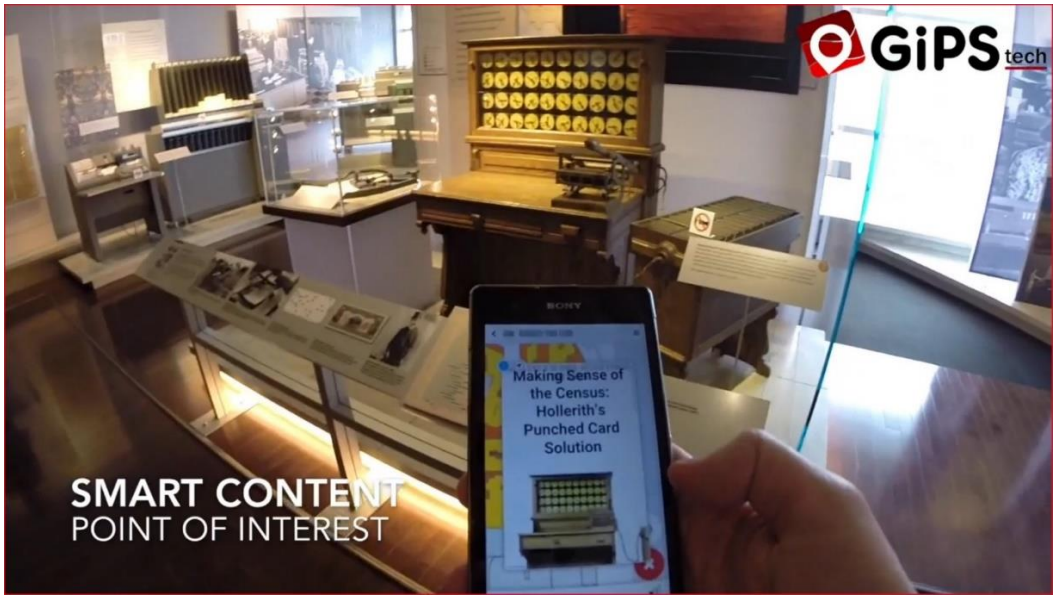

Рис. *9 Система для визначення положення відвідувачів музею від GiPStech*

До переваг рішень від компанії GiPStech можна віднести:

1. Використання ефективних алгоритмів на основі наявних у мобільних пристроях датчиків.
2. Поєднання різних технологій позиціонування для підвищення точності роботи системи.
3. Наявність готових до впровадження рішень.

До недоліків рішень від компанії GiPStech можна віднести:

1. Камери не використовуються в якості окремої технології.

1.7 Висновки з розділу 1

1. Системи для визначення положення людини в обмеженому просторі у реальному часі поділяються на два типи: ті, що потребують спеціалізованої інфраструктури та ті, що використовують широкодоступну інфраструктуру: датчики смартфонів, Wi-Fi та камери.

2. Головна проблема використання датчиків смартфонів та Wi-Fi полягає у потребі постійного носіння пристрою, що, наприклад, не може бути використано у системах, користувачами яких є літні люди.

3. Головна проблема використання камер полягає у тому, що більшість таких рішень використовують камери глибини, які не можна вважати частиною широкодоступної інфраструктури у будівлях.

4. Передові дослідження у глибинному навчанні дають можливість розпізнавати людину у реальному часі з використанням камери спостереження, яка підключена до мікрокомп’ютера. Після розпізнавання ми можемо визначити положення людини в обмеженому просторі.

5. Проблемою визначення положення в обмеженому просторі займаються передові компанії, тому створення подібних комп’ютерних систем є актуальним.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ЗАСОБІВ ВИЗНАЧЕННЯ ПОЛОЖЕННЯ ЛЮДИНИ В ОБМЕЖЕНОМУ ПРОСТОРІ У РЕАЛЬНОМУ ЧАСІ

2.1 Згорткові нейронні мережі

Згорткові нейронні мережі були вперше описані Яном Лекуном та його командою у дослідженні [10], де вони вирішували задачу розпізнавання рукописних цифр із датасета MNIST. Через недостатню обчислювальну потужність та кількість даних для навчання згорткові нейронні мережі не набували широкого розповсюдження. Усе змінилося у 2012 році, коли Алекс Крижевський та його команда розробили модель AlexNet [11], здатну класифікувати 1.2 мільйони зображень на 1000 різних класів з похибками top-1 та top-5 у 37.5% та 17% відповідно. У статті [17] наведений детальний огляд принципу роботи згорткової нейронної мережі.

Згорткова нейронна мережа — це нейронна мережа прямого поширення спеціального виду, яка займається розпізнаванням ознак певних класів об’єктів на зображенні. У порівнянні з нейронною мережею прямого поширення згорткова нейронна мережа використовує значно меншу кількість параметрів. На рисунку 10 зображена її архітектура.

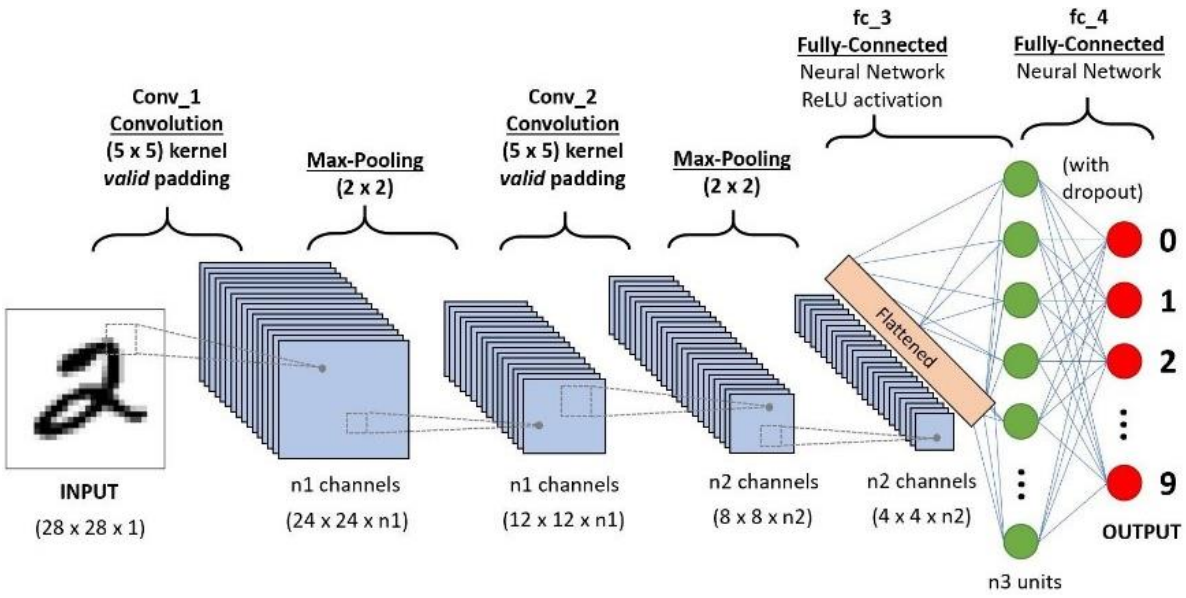

Рис. *10 Архітектура згорткової нейронної мережі*

Перший шар згорткової нейронної мережі — згортковий (convolution layer). Згорткова нейронна мережа працює на основі фільтрів, які займаються розпізнаванням ознак певних класів об’єктів на зображенні: прямих ліній, кривих певного типу тощо. Фільтр — це матриця чисел (ваг), які навчаються виявляти ознаки певних об’єктів на зображенні. Фільтр переміщається уздовж зображення та виявляє, чи присутня деяка ознака у його частині. Для отримання такої відповіді здійснюється операція згортки (convolution operation), яка математично визначається як сума добутків елементів фільтру та частини вхідного зображення. Назва згорткової нейронної мережі походить саме від назви цієї операції. На рисунку 11 зображена операція згортки.

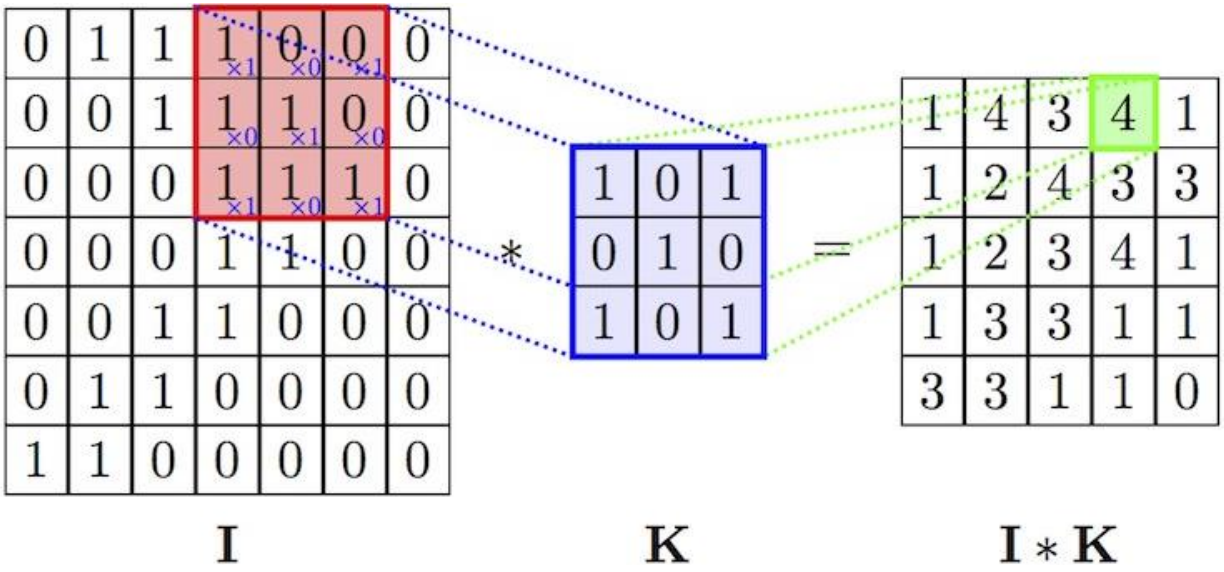

Рис. *11 Операція згортки*

Якщо ознака присутня у певній частині зображення, в результаті згортки ми отримаємо велике значення. В іншому випадку, ми отримаємо або мале значення, або 0. Цей факт наочно продемонстрований на рисунку 12. Фільтр займається розпізнаванням напрямлених праворуч кривих. У першому випадку він виявив шукану криву, оскільки в результаті ми отримали досить велике значення: $50 \times 30 + 50 \times 30 + 50 \times 30 + 20 \times 30 + 50 \times 30 = 6600$. У другому випадку шукана крива не була виявлена, оскільки ми отримали 0.

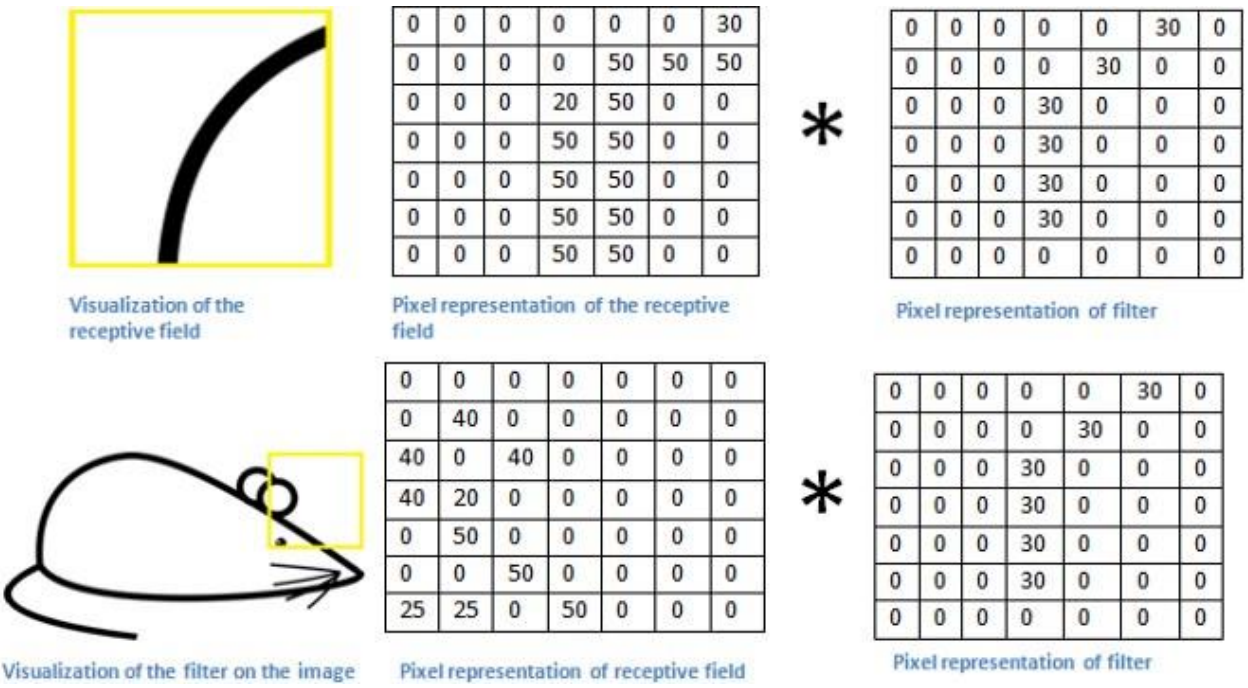

Рис. *12 Робота фільтру згорткової нейронної мережі*

Для виявлення характеристик на кольоровому (RGB) зображенні фільтр представляє собою тривимірну матрицю, кожен елемент якої містить значення для трьох каналів: червоного, зеленого та синього. Фільтр можна переміщувати уздовж зображення з кроком, відмінним від одиниці. Крок переміщення фільтру називається страйдом (stride). Розмірність матриці, отриманої після операції згортки, можна визначити за формулою 2.1, у якій: n_{in} — розмірність вхідного зображення, f — розмірність фільтру, s — страйд.

$$n_{out} = floor\left(\frac{n_{in}-f}{s}\right) + 1 \qquad (2.1)$$

Для випадку на рисунку 11 маємо: $(7-3)/1+1=5$.

Елементи матриці, отриманої після операції згортки, складаються з величиною зміщення (bias) та пропускаються через нелінійну функцію активації. Оскільки вхідні дані (наприклад, рукописні цифри) нелінійні, модель повинна це врахувати. Часто в якості такої функції використовують ReLU (Rectified Linear Unit), яка представлена формулою 2.2. З формули видно, що функція перетворює значення менші або рівні нулю в нуль, а додатні залишає без змін.

$$R(x) = \max(0, x) \tag{2.2}$$

Зазвичай у згортковому шарі використовується декілька фільтрів. Ознаки, отримані в результаті операції згортки, об’єднуються, формуючи n-вимірну матрицю.

Наступний після згорткового шар — шар підвибірки (pooling layer). З метою прискорення процесу навчання та зменшення обсягу пам’яті, що споживає нейронна мережа, виконують операцію зниження розмірності (downsampling) вхідних даних (матриці ознак). Існує два способи зниження розмірності: максимальне об’єднання (max pooling) та середнє об’єднання (average pooling). Під час операції об’єднання уздовж вхідних даних переміщається так зване вікно просіювання. З даних, що потрапляють в його «поле зору», знаходиться максимальне чи середнє значення, яке переміщається в результуючу матрицю. На рисунку 13 продемонстровані операції максимального та середнього об’єднання.

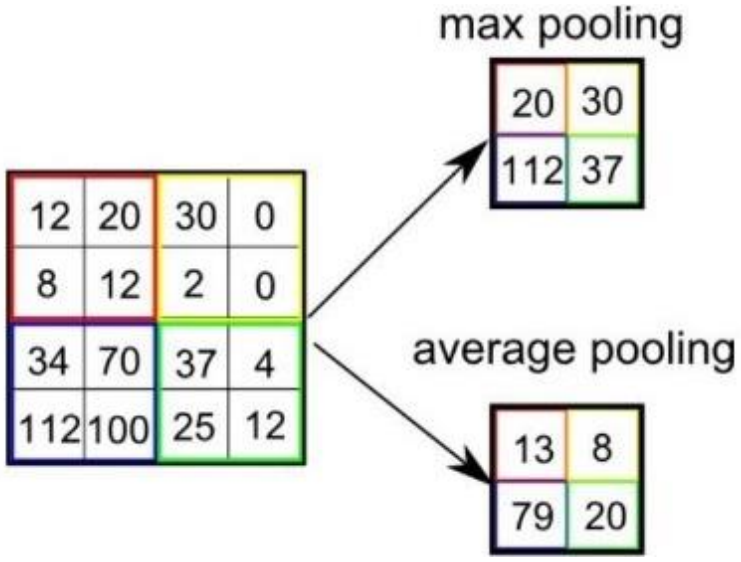

Рис. *13 Операції максимального та середнього об’єднання*

Перевагою операції об’єднання є те, що воно «змушує» нейронну мережу зосередитись на декількох нейронах замість усіх, що має регуляризуючий ефект, зменшуючи вірогідність перенавчання (overfitting).

Після згорткових шарів та шарів підвибірки йде один чи декілька повністю пов’язаних шарів (fully-connected layer). Отримані в результаті проходження через попередні шари ознаки (матриці) розгортаються у довгий вектор, який проходить через декілька повністю пов’язаних шарів. У кожному такому шарі вектор ознак множиться на ваги шару, підсумовується зі значенням зміщення та проходить через нелінійну функцію активації.

Останній шар згорткової нейронної мережі — вихідний (output layer). Вихідний шар відповідає за обчислення вірогідності приналежності вхідних даних до певного класу об’єкта, наприклад, рукописної цифри «3». Кількість нейронів у вихідному шарі зазвичай дорівнює кількості класів об’єктів, які ми хочемо розпізнати. Виходи останнього повністю пов’язаного шару пропускаються через функцію активації, наприклад Softmax, яка формує вектор ймовірностей приналежності вхідного зображення до того чи іншого класу об’єктів. Функція Softmax визначається формулою 2.3, де x_i — значення, отримане на i-му нейроні вихідного шару нейронної мережі.

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \tag{2.3}$$

Для того, щоб визначити, наскільки точно нейронна мережа розпізнає об’єкти, використовується функція втрат (loss function). Відповіддю цієї функції є дійсне число, що характеризує точність роботи нейронної мережі. Розповсюдженою функцією втрат, яка застосовується при прогнозуванні декількох класів об’єктів, є категоріальна функція крос-ентропійних втрат (Categorical Cross-Entropy Loss function), що представлена формулою 2.4, де $\hat{y}$ — фактична відповідь нейронної мережі, y — бажана.

$$H(y, \hat{y}) = -\sum_i y_i \times \log \hat{y}_i \tag{2.4}$$

Під час навчання згорткової нейронної мережі ми застосовуємо алгоритм зворотного поширення помилки (backpropagation) для мінімізації значення функції втрат.

2.2 Моделі нейронних мереж для розпізнавання об'єктів у реальному часі

2.2.1 Модель YOLO

У розділі 1 кваліфікаційної роботи були розглянуті методи розпізнавання об’єктів сімейства R-CNN. Проблемою цих методів є те, що процес розпізнавання відбувається у два етапи: виявлення регіонів інтересу на зображенні та розпізнавання об’єктів у цих регіонах. Такий підхід дозволяє досягти швидкості лише 5 FPS (у випадку методу Faster R-CNN) та не може бути використаний для розпізнавання у відеопотоці. Джозеф Редмон та його команда представили у 2015 році новий підхід до розпізнавання об’єктів — YOLO [15]. YOLO виявляє об’єкти за один прохід, використовуючи єдину нейронну мережу. Це дає можливість розпізнавати об’єкти у реальному часі. YOLO та її зменшена версія — Fast YOLO — працюють зі швидкістю 45 та 155 FPS відповідно. Остання версія цієї системи — YOLOv4.

Інженери із компанії Deep Systems зробили достатньо повний огляд [18] статті Джозефа Редмона. На рисунку 14 зображена архітектура моделі YOLO, яка здатна розпізнавати 20 різних класів об’єктів.

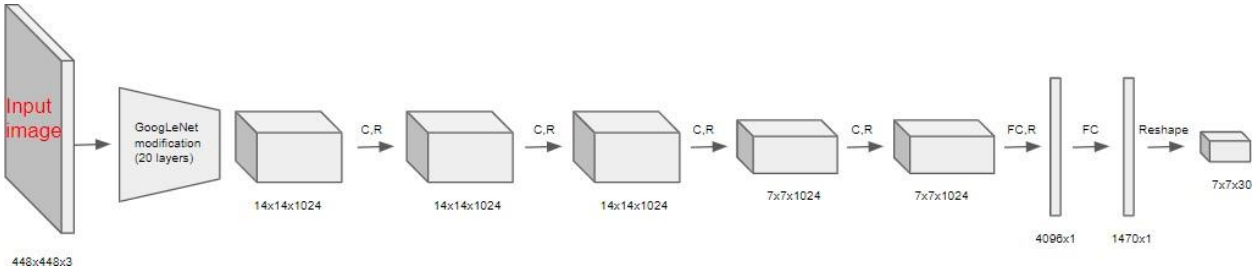

Рис. *14 Архітектура моделі YOLO*

На вхід мережі подається кольорове зображення розмірами $448 \times 448 \times 3$. Спочатку воно пропускається через частину модифікованої архітектури нейронної мережі GoogLeNet [19]. На виході ми отримаємо набір ознак (матрицю) розмірами $14 \times 14 \times 1024$. Потім застосовуємо чотири послідовні операції згортки для отримання матриці розмірами $7 \times 7 \times 1024$. Після третьої операції просторова розмірність стає меншою у 2 рази за рахунок страйда. Після цього ми розгортаємо матрицю у вектор 4096×1 та подаємо до повністю пов'язаного шару. На виході нейронної мережі отримуємо вектор 1470×1, який трансформуємо у матрицю $7 \times 7 \times 30$. Результуюча матриця містить інформацію про виявленні об'єкти. На рисунку 15 представлена її інтерпретація.

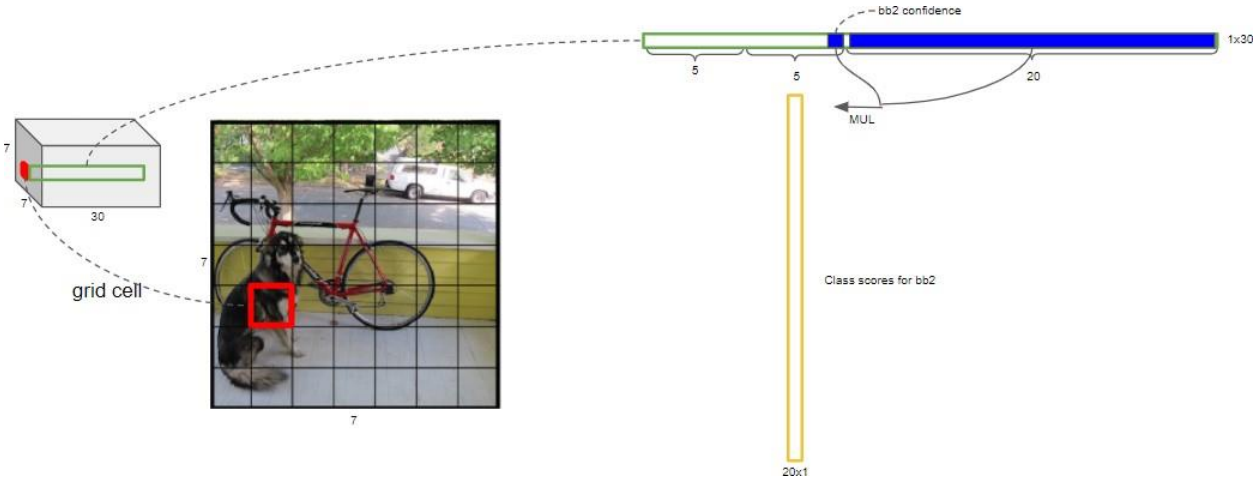

Рис. *15 Інтерпретація результуючої матриці моделі YOLO*

На вихідне зображення наноситься сітка розмірами 7×7. Кожна комірка результуючої матриці відповідає комірці на зображенні. Вона включає у себе вектор із 30 значень: перші 5 містять інформацію про першу обмежувальну рамку для цієї комірки, інші 5 — про другу, а останні визначають впевненість для кожного з 20 класів того, що центр об'єкта знаходиться усередині комірки. Інформація про кожну обмежувальну рамку складається з:

1. x та y — координати центру обмежувальної рамки (від 0 до 1 відносно розмірів комірки).

2. w — ширина обмежувальної рамки (від 0 до 1 відносно ширини зображення).
3. h — висота обмежувальної рамки (від 0 до 1 відносно висоти зображення).
4. c — впевненість того, що об’єкт був правильно розпізнаний обмежувальною рамкою (від 0 до 1).

Значення впевненості для кожного з 20 класів не прив’язані до обмежувальних рамок. Для того, щоб отримати впевненість того, на скільки добре об’єкт того чи іншого класу був розпізнаний обмежувальною рамкою треба помножити впевненість обмежувальної рамки на кожне значення впевненості по класам. Як результат, для кожної обмежувальної рамки ми отримаємо вектор із 20 елементів, кожен із яких визначає впевненість того, на скільки добре вона розпізнала об’єкт того чи іншого класу. Для вихідного зображення ми визначили 98 обмежувальних рамок з інформацією про їх просторові координати та значення впевненості для кожного класу. Фінальний крок — визначити обмежувальні рамки, які найбільш точно розпізнали об’єкти. На рисунку 16 представлена ця процедура.

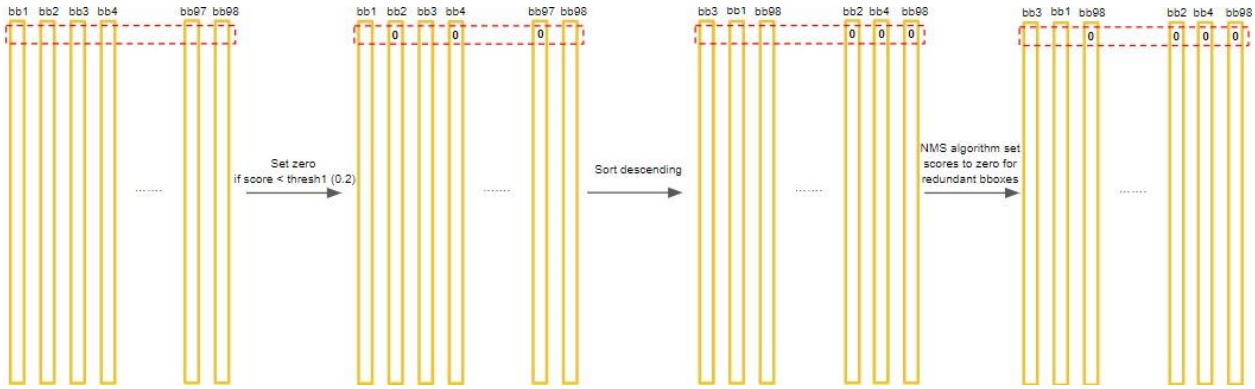

Рис. *16 Процедура визначення обмежувальних рамок, які найбільш точно розпізнали об’єкти, в моделі YOLO*

Процедура визначення обмежувальних рамок, які найбільш точно розпізнали об’єкти, складається з наступних кроків:

1. Якщо рядок у векторах впевненості не останній, фіксуємо його. Інакше переходимо до кроку 6.
2. Присвоюємо 0 значенням рядка, які менші за порогове.
3. Сортуємо отримані значення за зменшенням.
4. Застосовуємо алгоритм non-maximum suppression.
5. Переходимо до кроку 1.
6. Для кожного отриманого вектору впевненості по класам обираємо індекс елементу (класу) з максимальним значенням. Якщо значення максимального елементу > 0, зберігаємо його та інформацію про просторове розміщення відповідної обмежувальної рамки у результуючий список. Інакше відкидаємо.

Алгоритм non-maximum suppression складається з наступних кроків:

1. У рядку відсортованих за зменшенням значень впевненості для конкретного класу обираємо максимальний «невідпрацьований» елемент.
2. Порівнюємо цей елемент з іншими ненульовими елементами у рядку. Якщо величина Intersection over Union обмежувальних рамок елементів з максимальним та поточним значенням впевненості > 0.5, присвоюємо поточному елементу 0. При досягненні кінця рядка переходимо до кроку 1 до вичерпання елементів.

В оригінальній роботі [15] приводяться результати порівняння швидкості та точності роботи YOLO та Fast YOLO з Fast R-CNN та Faster R-CNN. Ці дані наведені у таблиці 3.

Таблиця *3*

Порівняння швидкості та точності роботи YOLO та Fast YOLO з Fast R-CNN та Faster R-CNN

Критерій / Метод	**Fast R-CNN**	**Faster R-CNN**	**YOLO**	**Fast YOLO**
Швидкість (FPS)	0.5	7	45	155
Точність (%, VOC 2007)	70.0	73.2	63.4	52.7

З отриманих дослідниками даних можна побачити, що модель YOLO програє методу Faster R-CNN на 10% у точності, але майже у 6.5 раз перевершує його за швидкістю роботи. Модель Fast YOLO програє методу Faster R-CNN на 20.5% у точності, але є швидшою у 22 рази.

До переваг моделі відносяться:

1. Має високу швидкість.
2. Має менше помилок при розпізнаванні на фоні.
3. Навчається загальним представленням об’єктів, через що може використана у різних предметних областях, наприклад, для виявлення об’єктів на картинах художників.

До недоліків моделі відносяться:

1. Має меншу точність у порівнянні з Fast R-CNN та Faster R-CNN.
2. Має проблеми при виявленні маленьких та близько розташованих один до одного об’єктів.

У цьому році Олексій Бочковський та його команда розробили четверту версію моделі YOLO — YOLOv4 [16]. YOLOv4-tiny, її зменшена версія, може працювати у режимі реального часу на пристроях з обмеженими ресурсами.

Модель YOLOv4-tiny — це глибока згорткова нейронна мережа, яка складається з 38 шарів, серед яких є: згорткові (convolutional) шари, шари маршрутизації (routing), шари підвибірки (max pooling), шар підвищення розмірності (upsampling) та 2 вихідних (yolo) шари. Головною перевагою YOLOv4-tiny є швидкість роботи: вона працює зі швидкістю 371 FPS на відеокарті NVIDIA GeForce GTX 1080 Ti та досягає точності 40.2% на датасеті COCO [20]. На рисунку 17 наведений графік порівняння YOLOv4-tiny з деякими іншими детекторами об’єктів за точністю та швидкістю роботи [21]. Автор моделі зазначає, що YOLOv4-tiny є найкращим «легким» детектором об’єктів з точки зору швидкості та точності.

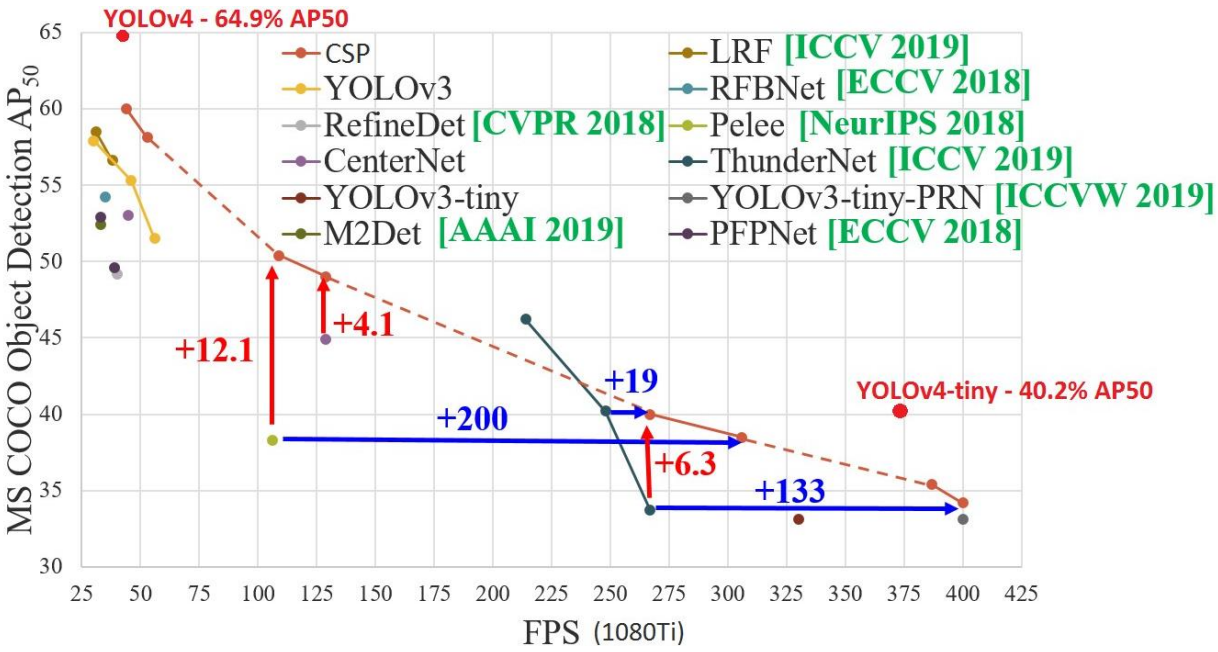

Рис. *17 Порівняння YOLOv4-tiny з деякими іншими детекторами об'єктів за точністю та швидкістю роботи*

2.2.2 Модель MobileNet

Окрім YOLOv4-tiny, існують також інші моделі згорткових нейронних мереж, здатні розпізнавати об'єкти у реальному часі, працюючи на пристроях з обмеженими ресурсами. Однією з таких є MobileNet, яка була представлена групою дослідників у роботі [22] у 2017 році. Виходячи з її назви можна зрозуміти, що вона покликана працювати на мобільних пристроях. MobileNet використовується в якості блоку для швидкого вилучення ознак об'єктів, які потім використовуються детекторами, наприклад SSD [23], для розпізнавання. Остання версія моделі, MobileNetV3, була представлена у 2019 році [24]. Основна інформація про особливості MobileNet висвітлена у [25, 26].

Згорткова нейронна мережа використовує операцію згортки для виявлення ознак об'єктів. Кількість операцій згорткового шару визначається за формулою 2.5, де D_K — розмірність фільтру, M — кількість каналів на вході, D_G — розмірність виходу, N — кількість фільтрів. На рисунку 18 зображена класична згортка.

$$Convolution\ operations = D_K^2 \times M \times D_G^2 \times N \qquad (2.5)$$

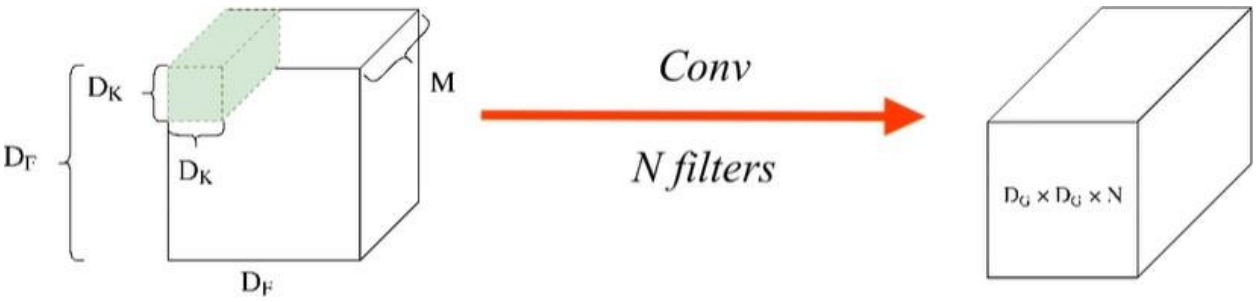

Рис. *18 Класична згортка*

За рахунок операцій множення згортка вимагає багато ресурсів. Архітектура MobileNet використовує спеціальний тип згортки — depthwise separable convolution, яка зменшує кількість математичних операцій та параметрів нейронної мережі. Ідея depthwise separable convolution полягає у тому, щоб розкласти класичну згортку на дві частини: depthwise-згортку, яка представляє собою фільтр по кожному каналу, а також pointwise-згортку — згортку 1×1.

Depthwise-згортка використовує фільтри розмірами $D_K \times D_K \times 1$ для кожного з M каналів на вході. Кількість операцій такої згортки визначається за формулою 2.6. На рисунку 19 зображена depthwise-згортка. У результаті ми отримаємо вихід G1, до якого застосуємо pointwise-згортку.

$$Depthwise\ convolution\ operations = D_K^2 \times D_G^2 \times M \qquad (2.6)$$

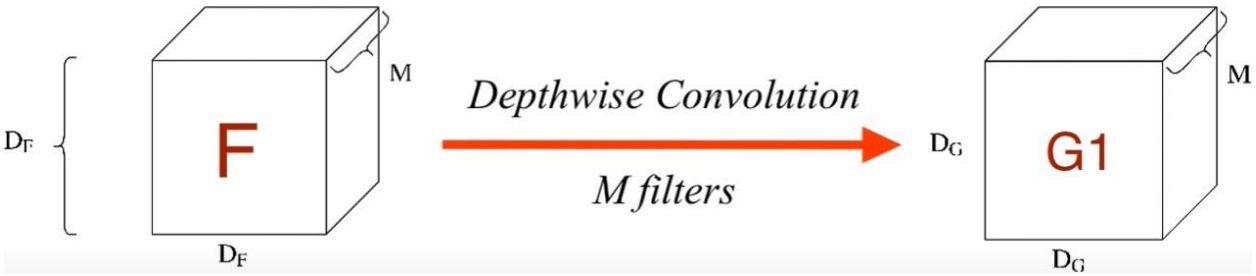

Рис. *19 Depthwise-згортка*

Pointwise-згортка використовує N фільтрів розмірами $1 \times 1 \times M$. Кількість операцій такої згортки визначається за формулою 2.7. На рисунку 20 зображена pointwise-згортка.

$$Pointwise\ convolution\ operations = M \times D_G^2 \times N \qquad (2.7)$$

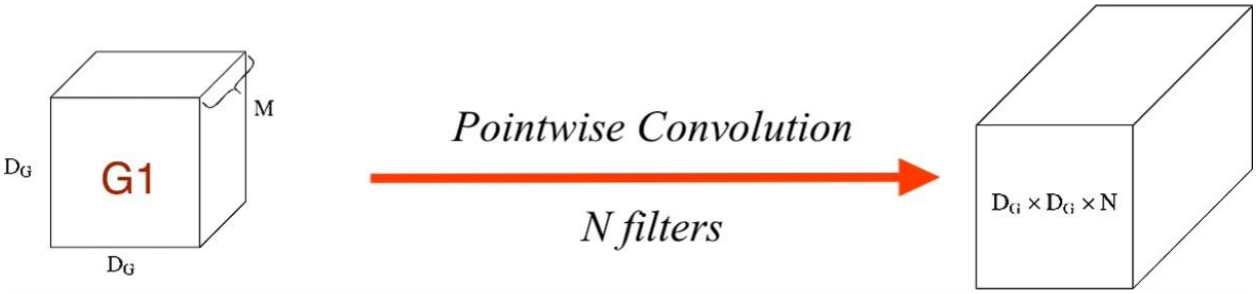

Рис. *20 Pointwise-згортка*

Загальна кількість операцій depthwise separable convolution є сумою кількості операцій depthwise-згортки та pointwise-згортки та визначається за формулою 2.8.

$$Depthwise\ separable\ conv.operations = D_G^2 \times M \times (D_K^2 + N) \qquad (2.8)$$

Формула 2.9 визначає співвідношення кількості операцій depthwise separable convolution до кількості операцій звичайної згортки. При $N = 1024$ та $D_K = 3$ ми отримаємо значення 0.11 — виграш у операціях майже у 9 разів.

$$Operations\ ratio = \frac{D_G^2 \times M \times (D_K^2 + N)}{D_K^2 \times M \times D_G^2 \times N} = \frac{1}{N} + \frac{1}{D_K^2} \qquad (2.9)$$

Автори моделі порівняли точність, кількість операцій та параметрів з використанням звичайних згорткових шарів та з використанням depthwise separable convolution [22]. Використання depthwise separable convolution значно зменшує кількість математичних операцій та параметрів мережі. Точність нейронної мережі зменшується лише на 1%. Дані, які отримали дослідники, наведені у таблиці 4.

Таблиця *4*

Порівняння точності, кількості операцій та параметрів з використанням звичайних згорткових шарів та depthwise separable convolution

Модель / Критерій	**Точність (%, ImageNet)**	**Мільйонів операцій**	**Мільйонів параметрів**
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Модель MobileNet складається з 1 згорткового шару та 13 блоків depthwise separable convolution. Особливістю архітектури є відсутність шарів підвибірки (max pooling). Замість них для зниження просторової розмірності використовується згортка зі страйдом 2. Двома гіперпараметрами архітектури є α (множник ширини) та ρ (множник розмірності). Множник ширини відповідає за кількість каналів у кожному шарі. При $\alpha = 0.5$ ми отримаємо архітектуру зі зменшеною у 2 рази кількістю каналів на виході кожного блоку. Множник розмірності відповідає за розмірність входів кожного шару. При $\rho = 0.5$ розмірність ознаки, що подається на вхід кожному шару, буде зменшена вдвоє. Зменшуючи α та ρ ми знижуємо точність розпізнавання, але збільшуємо швидкість роботи та отримаємо виграш у пам’яті.

Дослідники у [24] порівняли точність детектора SSDLite з використанням різних версій моделі MobileNet. Представлені у таблиці 5 дані свідчать про те, що різні версії MobileNet програють YOLOv4-tiny за точністю майже у 2 рази.

Таблиця *5*

Порівняння точності детектора SSDLite з використанням різних версій моделі MobileNet

Критерій / Модель	**MobileNet**	**MobileNetV2**	**MobileNetV3**
Точність (%, COCO)	22.2	22.1	22.0

2.2.3 Інші моделі для розпізнавання об’єктів у реальному часі

Серед інших моделей нейронних мереж для розпізнавання об’єктів у реальному часі, які здатні працювати на пристроях з обмеженими ресурсами, були також розглянуті SSD [23] та SqueezeNet [27]. У статті [28] висвітлені основні принципи їхньої роботи.

Детектор SSD є своєрідним продовженням детектора YOLO [15], оскільки він перейняв від нього багато принципів роботи (наприклад, алгоритм non-maximum suppression). Відмінною особливістю SSD є розпізнавання об’єктів за один прохід за допомогою заданої сітки вікон на піраміді зображень (різна розмірність зображень). Піраміда зображень закодована у виходах при послідовних операціях згортки та підвибірки. У такий спосіб SSD розпізнає як великі, так і маленькі об’єкти за один прохід. В оригінальній статті [23] для вилучення ознак дослідники використовували модель VGG [29], яка потребує багато обчислювальних ресурсів через велику кількість параметрів. Використання MobileNet у якості блоку для вилучення ознак дозволяє SSD працювати на пристроях з обмеженими ресурсами, однак такий підхід програє за точністю моделі YOLOv4-tiny, як було описано вище.

SqueezeNet — це маленька, але точна модель. Вона, як і MobileNet, може бути використана в якості блоку для вилучення ознак на зображенні та працювати на мобільних пристроях. Відмінною особливістю SqueezeNet є те, що дані спочатку стискаються до чотирьох 1×1 згорткових фільтрів, а потім розширюються до чотирьох 1×1 і чотирьох 3×3 згорткових фільтрів. Одна ітерація стиснення-розширення називається Fire Module. Модель SqueezeNet досягає точності AlexNet, використовуючи при цьому у 50 разів менше параметрів. Команда у дослідженні [30] займалася розробкою моделі Tinier-YOLO — покращеної версії YOLOv3-tiny, яка використовує ідеї SqueezeNet (Fire Module) та має точність 34% на датасеті COCO [20], програючи YOLOv4-tiny.

2.3 Фреймворки та бібліотеки глибинного навчання

2.3.1 Фреймворк Darknet

Darknet — це відкритий фреймворк для глибинного навчання, написаний на мовах програмування C та C++, який головним чином застосовується для навчання та використання моделей YOLOv2, YOLOv3, YOLOv4, а також їхніх зменшених версій. Його початковим розробником є Джозеф Редмон — автор моделі YOLO [15]. Олексій Бочковський, автор YOLOv4 [16], продовжив його справу, додавши у фреймворк підтримку нових шарів та функцій активації нейронної мережі, можливість роботи на Windows та багато іншого.

Фреймворк Darknet можна інсталювати на Linux чи Windows. Після встановлення він надає утиліти командного рядка за допомогою яких можна розпізнавати об’єкти на зображеннях чи відео, розпочати навчання нейронної мережі, перевірити точність навченої нейронної мережі тощо. Розробник Darknet написав покроковий алгоритм для навчання моделі YOLO. Модель YOLO складається з текстового файлу з параметрами навчання та архітектурою, а також бінарного файлу з вагами. Darknet включає у себе навчені моделі YOLO. Фреймворк також підтримує навчання одразу на декількох графічних процесорах. Важливим є те, що розробник фреймворка надав рекомендації щодо того, як покращити результати навчання та точність розпізнавання.

Олексій Бочковський приводить дані [21], згідно з якими Darknet працює повільніше за інші бібліотеки при використанні навченої нейронної мережі. Це означає, що ми можемо навчити модель за допомогою Darknet, а використовувати за допомогою більш оптимізованих бібліотек. Деякі з цих даних приведені у таблиці 6 (значення 416 — це розмір входу мережі, а FP — кількість біт дійсного числа). З наведених у таблиці даних видно, що конкурентами Darknet при використанні навченої моделі YOLOv4-tiny є бібліотеки OpenCV та tkDNN.

Таблиця *6*

Порівняння швидкості YOLOv4-tiny при використанні різних бібліотек на графічному процесорі NVIDIA GeForce RTX 2080 Ti

Модель / Бібліотека	**Darknet (FPS)**	**tkDNN TensorRT FP16 (FPS)**	**OpenCV FP16 (FPS)**
YOLOv4-tiny 416	443	790	773

2.3.2 Фреймворк Keras

Keras — це відкритий фреймворк для глибинного навчання, написаний на мові програмування Python, що є надбудовою над бібліотекою TensorFlow. TensorFlow — це відкрита бібліотека для машинного навчання від компанії Google, написана на мові програмування C++. Головним розробником Keras є Франсуа Шоллє, інженер із Google. При розробці фреймворка головний акцент був зроблений на надання дослідникам можливостей для швидкого експериментування.

Фреймворк Keras можна інсталювати на Linux, Windows чи macOS. Як і Darknet, він дозволяє навчати нейронну мережу на декількох графічних процесорах одночасно. Основні структури даних, які використовує Keras — це шари (layers) та моделі (models). Найпростішою моделлю є Sequential, яка представляє собою послідовність із шарів нейронної мережі. Фреймворк надає безліч готових шарів нейронної мережі: згортковий шар, шар підвибірки, повністю пов’язаний шар тощо. Для побудови більш складних архітектур Keras надає Functional API, який дозволяє будувати графи із шарів та створювати власні моделі (models). Keras включає у себе навчені моделі нейронних мереж (MobileNet [22], VGG [29] тощо), а також деякі датасети для різних задач (датасет рукописних цифр MNIST, датасет з відгуками про фільми з сайту IMDB тощо).

2.3.3 Фреймворк PyTorch

PyTorch — це відкритий фреймворк для машинного навчання, написаний на мовах програмування C, C++ та Python, що розроблений на базі бібліотеки Torch. Torch — це відкрита MATLAB-подібна бібліотека для мови програмування Lua, що реалізує велику кількість алгоритмів для глибинного навчання. PyTorch розроблюється лабораторією штучного інтелекту компанії Facebook.

Фреймворк PyTorch можна інсталювати на Linux, Windows чи macOS. PyTorch забезпечує дві основні функціональності: матричні обчислення з прискоренням на графічних процесорах та використання глибоких нейронних мереж, побудованих на базі системи автоматичного диференціювання. Фреймворк складається з трьох модулів: autograd, optim та nn. Модуль autograd включає у себе систему автоматичного диференціювання, модуль optim реалізує алгоритми оптимізації, які використовуються при побудові нейронних мереж, а модуль nn надає високорівневі абстракції над модулем autograd. Як і Darknet та Keras, PyTorch включає у себе навчені моделі нейронних мереж (AlexNet [11], SqueezeNet [27] тощо). Як і Keras, фреймворк включає у себе датасети для різних задач (COCO [20], датасет рукописних цифр MNIST тощо).

2.3.4 Бібліотека OpenCV

OpenCV — це відкрита бібліотека комп’ютерного зору, написана на мові програмування C++. Також існують прив’язки (bindings) на мовах програмування Python, Java, MATLAB та інших. Розробником OpenCV є компанія Intel. OpenCV включає у себе алгоритми для обробки та перетворення зображень, функціонал для роботи з відео, а також модуль для використання глибоких нейронних мереж класифікації, розпізнавання, виявлення ключових точок та сегментації об’єктів на зображенні.

OpenCV можна інсталювати на платформи Linux, Windows чи macOS. Готова збірка OpenCV може не включати підтримки необхідного функціоналу, тому нормальною практикою є створення файлів збірки за допомогою CMake

та подальша компіляція бібліотеки. Наприклад, готова збірка OpenCV не підтримує обчислення на графічному процесорі, що є важливим для застосунків, які повинні працювати у режимі реального часу.

OpenCV дозволяє використовувати навчені моделі нейронних мереж за допомогою модуля dnn. Бібліотека підтримує навчені на різних фреймворках моделі (моделі різного формату): Darknet, Keras, PyTorch та інших. Модуль dnn підтримує основні шари нейронної мережі. Також він містить клас моделі нейронної мережі (Model), який є базовим класом для моделей класифікації (ClassificationModel), розпізнавання (DetectionModel), виявлення ключових точок (KeypointsModel) та сегментації (SegmentationModel). Кожна із моделей вирішує конкретну задачу комп'ютерного зору.

2.3.5 Інші фреймворки та бібліотеки глибинного навчання

Також існують інші фреймворки та бібліотеки глибинного навчання. Серед них можна виділити Microsoft Cognitive Toolkit — інструментарій для глибинного навчання від компанії Microsoft, Theano — бібліотека чисельних обчислень для мови програмування Python, Deeplearning4j — бібліотека глибинного навчання для мов програмування Java та Scala, Apache MXNet — платформа для навчання та розгортання глибоких нейронних мереж.

2.4 Датасети із зображеннями людей

2.4.1 Датасет COCO

COCO (Common Objects in Context) [20] — це масштабний датасет для навчання нейронної мережі, що містить зображення складних повсякденних сцен із розповсюдженими об'єктами в їхньому природному контексті. Він містить анотації для вирішення наступних задач: розпізнавання об'єктів, виявлення ключових точок та оцінка поз людей, сегментація об'єктів та генерація текстового опису до зображень. COCO включає у себе 1.5 мільйона об'єктів 80 класів (категорій) та 200 тисяч анотованих зображень. Одним із класів є

клас Person — «людина». Об’єкти поділені на категорії та суперкатегорії. Наприклад, категорії «велосипед», «мотоцикл» та «автомобіль» відносяться до суперкатегорії «транспортний засіб». Датасет COCO використовується багатьма дослідниками для порівняння точності їхніх детекторів, що можна було побачити у розділі 2.2 кваліфікаційної роботи.

Завантажити датасети для навчання, валідації та тестування можна з офіційного сайту COCO. Зображення та анотації до них завантажуються окремо. Анотації представляють собою JSON-файли для кожної з перерахованих вище задач та містять інформацію про кожне зображення. Для того, щоб не обробляювати їх вручну, COCO надає зручний API для мов програмування Python, MATLAB та Lua. COCO API працює з файлом анотацій та дозволяє виконувати наступні операції: отримати всі категорії та суперкатегорії, отримати ідентифікатори конкретних категорій, отримати зображення за ідентифікаторами категорій, отримати зображення за їхніми ідентифікаторами, отримати анотації чи URL зображень тощо. Оскільки COCO API дозволяє отримати URL зображень відповідної категорії, можна написати скрипт, який буде завантажувати тільки необхідні зображення замість усього датасета. Такий підхід є оптимальним, оскільки розмір архіву із зображеннями для навчання нейронної мережі складає 13 ГБ.

2.4.2 Датасет Open Images

Open Images (The Open Images Dataset V4) [31] — це масштабний датасет, що містить близько 9.2 мільйонів анотованих зображень для навчання нейронної мережі класифікувати об’єкти, розпізнавати об’єкти чи виявляти візуальні стосунки між об’єктами. Open Images V4 включає у себе 19.8 тисяч класів об’єктів для класифікації, 600 класів об’єктів для розпізнавання та 57 класів об’єктів для виявлення візуальних стосунків. Зображення містять складні сцени з у середньому 8 об’єктами. Одним із класів об’єктів, як і в COCO, є клас Person — «людина». Зображення для задачі розпізнавання об’єктів мають наступні атрибути: GroupOf (група із більше, ніж 5 об’єктів одного класу, які

сильно перекривають один одного), Partially occluded (об’єкт частково перекритий іншим об’єктом), Truncated (частина об’єкта виходить за рамки зображення), Depiction (не справжній об’єкт, наприклад, статуя), Inside (зображення зроблено усередині об’єкта, наприклад, салону авто).

Завантажити датасети для навчання, валідації та тестування можна з офіційного сайту Open Images. Зображення, а також анотації для кожної з перерахованих вище задач завантажуються окремо. Для полегшення цього процесу окремі розробники створили програмний засіб — OIDv4 ToolKit. Він дозволяє завантажувати зображення окремих категорій у потрібній кількості з відповідного датасета, завантажувати анотації до зображень, обирати директорію для завантаження, фільтрувати зображення за атрибутами (GroupOf, Truncated та інші) тощо. Даний засіб можна використовувати тільки для задач класифікації та розпізнавання. Не менш важливим є те, що завантаження відбувається у різних потоках, що значно пришвидшує цей процес. Також засіб надає можливість відображати завантажені зображення із анотованими об’єктами.

2.4.3 Датасет EuroCity Persons

EuroCity Persons (The EuroCity Persons Dataset) [32] — це масштабний датасет, що містить велику кількість зображень та анотацій пішоходів та водіїв транспортних засобів у сценах дорожнього руху. Зображення для датасета були зібрані за допомогою транспортного засобу, який рухався 31 містом 12 європейських країн. Набір даних включає понад 238200 анотованих людей (пішоходів) на понад 47300 зображеннях. Розробники датасета також проаналізували здатність до узагальнення деяких детекторів (Faster-RCNN [14], SSD [23] та інших), які були на ньому навчені. Окрім цього, вони вивчали вплив розміру навчального набору даних, його різноманітності (зображення, зроблені вдень чи вночі, а також у різних географічних регіонах), деталізації (наявність інформації про орієнтацію об’єкта), а також якості анотацій на точність

роботи детектора. Важливим є той факт, що об’єкти, які частково перекриваються іншими об’єктами, були анотовані з урахуванням області перекриття. Завантажити датасет можна на офіційному сайті після попередньої реєстрації.

2.4.4 Інші датасети із зображеннями людей

Також існують інші датасети із зображеннями людей. Серед них можна виділити Berkley Deep Drive (BDD100K) [33] — датасет із 100 тисячами відео, знятими із транспортного засобу, WiderPerson [34] — датасет із 13382 зображеннями з різними ситуаціями перекриття людей та CrowdHuman [35] — датасет із 24370 зображеннями людей у сценаріях натовпу.

2.5 Висновки з розділу 2

1. Найбільш оптимальною моделлю глибокої нейронної мережі для розпізнавання людини, яка здатна працювати на мікрокомп’ютері, є YOLOv4-tiny.

2. Для її навчання доречно використати фреймворк Darknet, оскільки він містить необхідні вказівки та рекомендації для моделей YOLO.

3. У якості датасета можна обрати COCO чи Open Images, оскільки вони містять зображення складних сцен.

4. Навчену нейронну мережу можна використовувати за допомогою бібліотеки OpenCV, оскільки вона демонструє високу швидкість у порівнянні з іншими бібліотеками, надає засоби для роботи з відео, а також містить реалізації інших алгоритмів для обробки зображень.

5. Для визначення ефективності обраних засобів необхідно їх випробовувати, створивши комп’ютерну систему для визначення положення людини в обмеженому просторі у реальному часі, яка буде працювати на пристроях з обмеженими ресурсами.

РОЗДІЛ 3 РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ ДЛЯ ВИЗНАЧЕННЯ ПОЛОЖЕННЯ ЛЮДИНИ В ОБМЕЖЕНОМУ ПРОСТОРІ У РЕАЛЬНОМУ ЧАСІ

3.1 Навчання моделей YOLOv4-tiny розпізнавати людей

3.1.1 Налаштування середовища для навчання

Першим етапом, необхідним для навчання моделі YOLOv4-tiny розпізнавати людей, є налаштування робочого середовища. Навчання було проведено на комп'ютері з наступними характеристиками:

1. Чотирьохядерний процесор Intel Core i7-8550U з тактовою частотою 1.8 ГГц.
2. Графічний процесор NVIDIA GeForce GTX 1050.
3. 16 ГБ оперативної пам'яті.
4. Операційна система Windows 10.

Для навчання моделі YOLOv4-tiny необхідно встановити фреймворк Darknet та його залежності: набір інструментів CUDA Toolkit, бібліотеки cuDNN та OpenCV. Для встановлення фреймворка Darknet були виконані наступні кроки:

1. Встановлено мову програмування Python 3.7.9 та бібліотеку NumPy 1.19.1.
2. Встановлено систему генерації файлів збірки CMake 3.18.2.
3. Встановлено середовище розробки Visual Studio 2019 із підтримкою розробки на мові програмування C++.
4. Оновлено драйвер графічного процесора.
5. Встановлено набір інструментів CUDA Toolkit 10.2.89.
6. Встановлено бібліотеку cuDNN 7.6.5.32.
7. Завантажено репозиторії opencv та opencv_contrib (бібліотека OpenCV та її додаткові модулі) версії 4.4.0 з GitHub у директорію `D:\Projects\OpenCV`.

8. За допомогою CMake згенеровано проект для подальшої компіляції бібліотеки OpenCV у директорії `D:\Projects\OpenCV\build` з наступними прапорцями: `BUILD_opencv_world`, `OPENCV_EXTRA_MODULES_PATH` (`D:/Projects/OpenCV/opencv_contrib-4.4.0/modules`, шлях до додаткових модулів OpenCV), `WITH_CUDA`, `OPENCV_DNN_CUDA`, `ENABLE_FAST_MATH`, `WITH_CUDNN`, `CUDA_FAST_MATH`, `WITH_CUBLAS`, `CUDA_ARCH_BIN` (6.1, значення compute capability для графічного процесора).
9. Скомпільовано за допомогою Visual Studio Release-конфігурацію проектів `ALL_BUILD` та `INSTALL` для встановлення OpenCV.
10. Додано змінну середовища `OpenCV_DIR`, яка вказує на директорію зі встановленою бібліотекою `D:\Projects\OpenCV\build\install`, а також додано до змінної середовища `PATH` посилання на каталог із бінарними файлами бібліотеки: `D:\Projects\OpenCV\build\install\x64\vc16\bin`.
11. Завантажено репозиторій darknet останньої версії з GitHub (ім’я користувача: AlexeyAB), та виконано скрипт `build.ps1` для встановлення фреймворка.

Для перевірки того, що всі залежності та фреймворк Darknet були успішно встановлені, були виконані наступні кроки:

1. Завантажені з репозиторія darknet у директорію `weights` ваги попередньо навченої на датасеті COCO моделі YOLOv4-tiny: `yolov4-tiny.weights`.
2. Виконано команду для розпізнавання об’єктів на зображенні: `.\darknet.exe detector test .\cfg\coco.data .\cfg\yolov4-tiny.cfg .\weights\yolov4-tiny.weights .\data\dog.jpg`.

Після перевірки встановленого фреймворка Darknet за допомогою вищенаведеної команди був отриманий результат, зображений на рисунку 21.

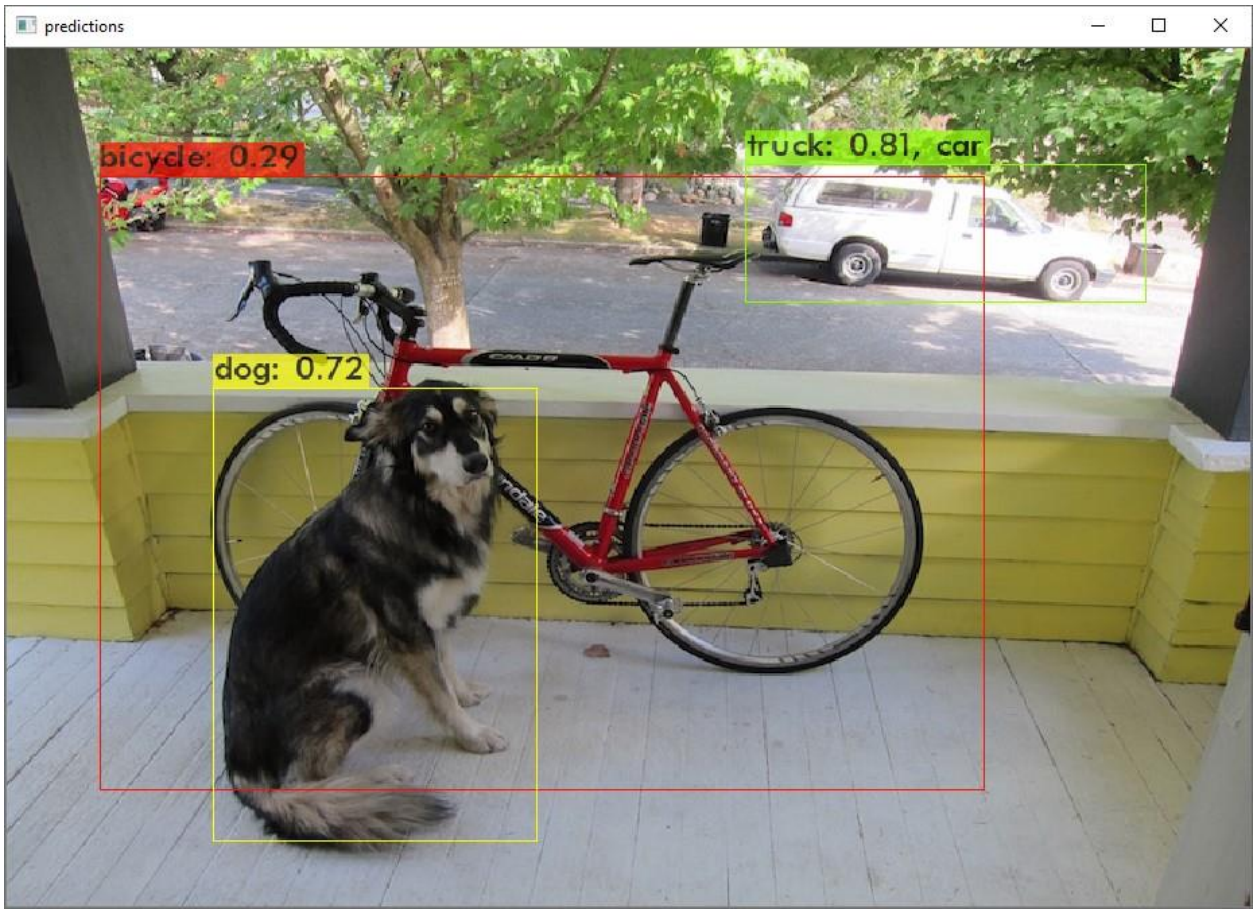

Рис. *21 Розпізнавання об'єктів на зображенні за допомогою встановленого фреймворка Darknet*

3.1.2 Підготовка зображень людей із датасетів COCO та Open Images

Наступним кроком після налаштування середовища для навчання є підготовка зображень із людьми. Для тренування було використано 10000 зображень для навчання та 2000 зображень для валідації із датасетів COCO та Open Images. Автор кваліфікаційної роботи поставив за мету порівняти точність навчених на різних датасетах моделей. Чому було обрано саме 10000 зображень для навчання? Розробник фреймворка Darknet зазначає, що розмір навчальної вибірки повинен складати від 2000 до 10000 зображень. Чому було обрано саме 2000 зображень для валідації? Існує загальна рекомендація, згідно з якою

розмір валідаційної вибірки повинен складати 20% від навчальної. Нижче будуть розглянуті кроки для підготовки обох датасетів.

Підготовка датасета COCO. Для підготовки датасета COCO були виконані наступні кроки:

1. Завантажені з офіційного сайту JSON-файли з анотаціями об'єктів: `instances_train2014.json` та `instances_val2014.json`.
2. Розроблено скрипт на мові програмування Python, який використовує COCO API для завантаження необхідної кількості зображень класу `Person` з навчальної та валідаційної вибірок.
3. Виконано розроблений на попередньому кроці Python-скрипт для завантаження навчальних та валідаційних зображень: `python.exe .\download_coco_single_class_images.py`.
4. Переміщено завантажені датасети до директорії `data` фреймворка Darknet.

Скрипт для завантаження навчальних та валідаційних зображень класу `Person` із датасета COCO працює наступним чином:

1. Ініціалізує об'єкт `COCO`, передаючи у його конструктор шлях до файлу з анотаціями навчальної чи валідаційної вибірки.
2. Отримує список об'єктів з інформацією про кожне зображення, що містить клас `Person`.
3. Фільтрує отримані об'єкти по кількості: 10000 чи 2000.
4. Завантажує кожне зображення за допомогою URL.
5. Для кожного завантаженого зображення створює анотації об'єктів у YOLO-форматі.

Лістинг 1 демонструє реалізацію 4 та 5 кроків роботи скрипта. Функція `download_and_save_images` створює директорію для датасета та завантажує у неї зображення за допомогою властивості `coco_url` кожного об'єкта-зображення. Функція `convert_and_write_annotations` створює анотації

для зображень. Для кожного завантаженого зображення вона створює текстовий файл із відповідним ім'ям та отримує список анотацій об'єктів на ньому. Для кожного об'єкта на зображенні датасет COCO містить анотацію у вигляді JSON-об'єкта з ідентифікатором категорії (`category_id`), обмежувальною рамкою (`bbox`) з координатою її лівої верхньої точки та розмірами, а також деякими іншими властивостями. Скрипт перетворює анотацію COCO до формату YOLO (всі розміри визначаються відносно ширини та висоти зображення): `<object-class> <x_center> <y_center> <width> <height>`, де `<object-class>` — ідентифікатор класу, `<x_center>` та `<y_center>` координати центра обмежувальної рамки, а `<width>` та `<height>` — її розміри. Функція записує перетворену анотацію у файл (значення координат та розміри обмежувальної рамки округлюються до 7 знаків).

Лістинг *1 Реалізація функціоналу завантаження зображень із датасета COCO та створення анотацій для кожного з них у форматі YOLO*

```
def download_and_save_images(images_to_download):
    if os.path.exists(DATASET_DIRECTORY_NAME):
        shutil.rmtree(DATASET_DIRECTORY_NAME)
    os.makedirs(DATASET_DIRECTORY_NAME)
    for image_to_download in images_to_download:
        with open(os.path.join(DATASET_DIRECTORY_NAME, imag
e_to_download["file_name"]), "wb") as image_file:
            image_file.write(requests.get(image_to_download
["coco_url"]).content)

def convert_and_write_annotations(coco, downloaded_images,
category_id):
    def truncate(number, number_of_decimals=0):
        factor = 10.0 ** number_of_decimals
        return math.trunc(number * factor) / factor
    for downloaded_image in downloaded_images:
        annotation_file_name = downloaded_image["file_name"
].replace(".jpg", ".txt")
        with open(os.path.join(DATASET_DIRECTORY_NAME, anno
tation_file_name), "a") as annotation_file:
            annotations = coco.loadAnns(coco.getAnnIds(imgI
ds=downloaded_image["id"], catIds=category_id, iscrowd=None
))
            image_width = downloaded_image["width"]
```

```
            image_height = downloaded_image["height"]
            for annotation in annotations:
                x_min = annotation["bbox"][0]
                y_min = annotation["bbox"][1]
                x_max = annotation["bbox"][0] + annotation[
"bbox"][2]
                y_max = annotation["bbox"][1] + annotation[
"bbox"][3]
                x_center = (x_min + x_max) / 2 / image_widt
h
                y_center = (y_min + y_max) / 2 / image_heig
ht
                width = (x_max - x_min) / image_width
                height = (y_max - y_min) / image_height
                annotation_file.write(
                    "0 " + str(truncate(x_center, NUMBER_OF
_DECIMALS_TO_LEAVE_IN_ANNOTATIONS)) + " " + str(
                        truncate(y_center, NUMBER_OF_DECIMA
LS_TO_LEAVE_IN_ANNOTATIONS)) + " " + str(
                        truncate(width, NUMBER_OF_DECIMALS_
TO_LEAVE_IN_ANNOTATIONS)) + " " + str(
                        truncate(height, NUMBER_OF_DECIMALS
_TO_LEAVE_IN_ANNOTATIONS)) + "\n")
```

На рисунку 22 зображені завантажені зображення та їх анотації з датасета COCO.

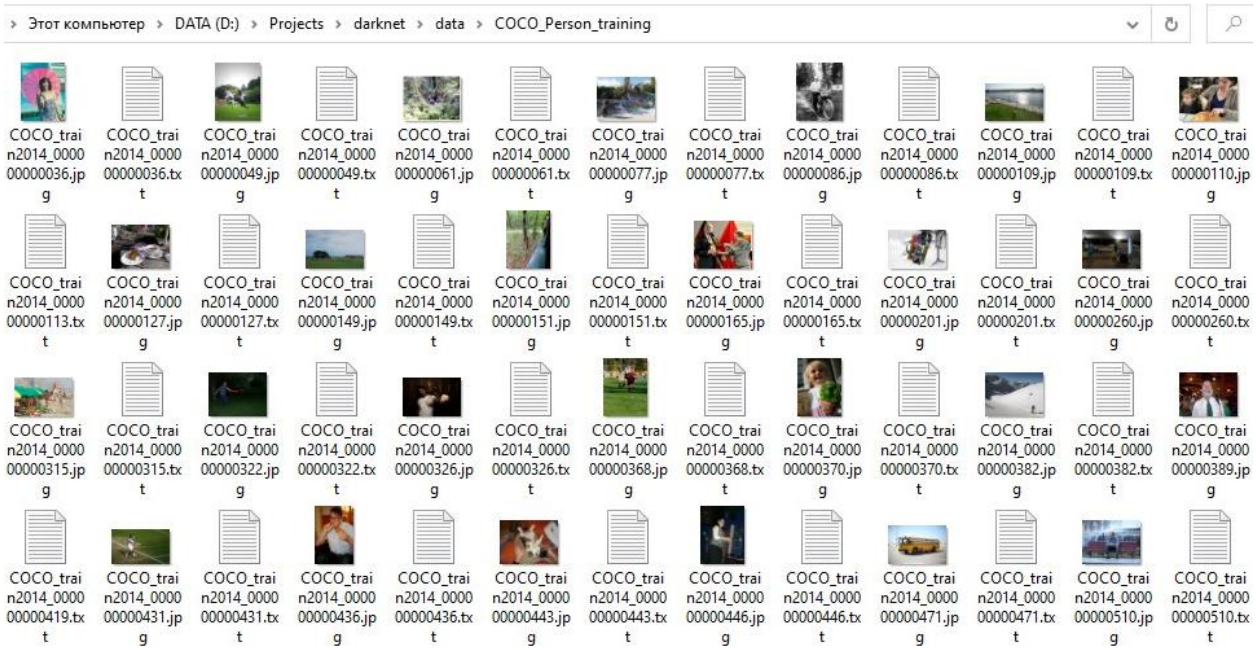

Рис. *22 Завантажені зображення та їх анотації з датасета COCO*

Підготовка датасета Open Images. Для підготовки датасета Open Images були виконані наступні кроки:

1. Завантажено репозиторій OIDv4_ToolKit з GitHub (ім’я користувача: theAIGuysCode).
2. Встановлені необхідні залежності за допомогою наступної команди: `pip install pandas awscli urllib3 tqdm`.
3. Доданий до файлу `classes.txt` рядок `Person`.
4. Виконана команда для завантаження 10000 зображень для навчання: `python.exe .\main.py downloader -y --classes .\classes.txt --type_csv train --image_IsTruncated 0 --image_IsGroupOf 0 --image_IsDepiction 0 --image_IsInside 0 --limit 10000`.
5. Виконана команда для завантаження 2000 зображень для валідації: `python.exe .\main.py downloader -y --classes .\classes.txt --type_csv validation --image_IsTruncated 0 --image_IsGroupOf 0 --image_IsDepiction 0 --image_IsInside 0 --limit 2000`.
6. Виконаний Python-скрипт для конвертації анотацій у YOLO-формат: `python.exe .\convert_annotations.py`.
7. Видалена директорія `Label` усередині директорій з датасетами.
8. Переміщені завантажені датасети до директорії `data` фреймворка Darknet.

На цьому етапі були підготовлені зображення із датасетів COCO та Open Images. Наступні кроки описуються відносно датасета COCO. Процес навчання на датасеті Open Images є подібним, за винятком назви файлів та директорій.

3.1.3 Підготовка файлів для навчання

Для навчання моделі необхідно підготувати наступні файли: файл з параметрами навчання та конфігурацією нейронної мережі, файл з назвами класів об’єктів, на яких ми навчаємо нейронну мережу, файл зі шляхами до відповідних файлів і директорій, необхідних для навчання, файли з відносними шляхами до навчальних і валідаційних зображень та файл з вагами для згорткових шарів попередньо навченої на датасеті COCO моделі YOLOv4-tiny.

На основі конфігураційного файлу `yolov4-tiny-custom.cfg`, що знаходиться у директорії `cfg` фреймворка Darknet, був створений новий файл, `yolov4-tiny-COCO-Person.cfg`, з наступними змінами:

1. Значення `width` та `height` були змінені на `width=416` та `height=416`.
2. Значення `batch` було змінено на `batch=64`.
3. Значення `subdivisions` було змінено на `subdivisions=16`.
4. Значення `max_batches` було змінено на `max_batches=10000`.
5. Значення `steps` було змінено на `steps=8000,9000`.
6. Значення `classes` у кожному `yolo`-шарі були змінені на `classes=1`.
7. Значення `filters` у кожному згортковому шарі перед шарами `yolo` були змінені на `filters=18`.

Значення `width` та `height` — це розмір нейронної мережі. Протягом навчання розмір кожного зображення змінюється під розмір нейронної мережі. Ці значення повинні бути кратні 32. Чим більший розмір нейронної мережі, тим більша точність розпізнавання, але менша швидкість її роботи. Оскільки нейронна мережа повинна працювати на мікрокомп’ютері, були обрані оптимальні з точки зору точності та швидкості значення.

Значення `batch` означає кількість зображень, які нейронна мережа обробляє за один батч. Значення `subdivisions` означає кількість міні-батчів в

одному батчі. Значення `max_batches` означає кількість ітерацій (батчів), протягом яких відбувається навчання нейронної мережі. Воно визначається за формулою 3.1.

$$max\ batches = \max(classes \times 2000, 6000, training\ images) \quad (3.1)$$

Графічний процесор за один раз оброблює міні-батч зображень, а ваги нейронної мережі оновлюються кожен батч. Чим менше значення `subdivisions`, тим швидше проходить навчання, але тим більше пам'яті повинен мати графічний процесор, щоб обробити міні-батч більшого розміру. Встановлені значення для `batch` та `subdivisions` є оптимальними для графічного процесора, на якому проводилося навчання. Розмір міні-батча визначається за формулою 3.2. У нашому випадку міні-батч складається з 4 зображень.

$$mini\ batch = \frac{batch}{subdivisions} \quad (3.2)$$

Значення `steps` задає порогові ітерації навчання, при досягненні яких швидкість навчання помножається на величину `scales`. Величина `steps` повинна складати 80% та 90% від значення `max_batches`. Для `scales` задано наступне значення: `scales=.1,.1`. Тобто, при досягненні 8000 та 9000 ітерацій швидкість навчання буде помножена на 0.1 (зменшена) обидва рази.

Значення `classes` відповідає кількості класів об'єктів, які нейронна мережа повинна розпізнавати, а значення `filters` задає кількість фільтрів, які має згортковий шар перед шаром `yolo`. Кількість фільтрів цього шару визначається за формулою 3.3.

$$filters = (classes + 5) \times 3 \quad (3.3)$$

Для створення файлу з назвами класів об’єктів, на яких ми навчаємо нейронну мережу, був створений файл `COCO_Person.names` в директорії `data`, який містить один рядок: `Person`.

Для створення файлу зі шляхами до відповідних файлів і директорій, необхідних для навчання, в директорії `data` був створений файл `COCO_Person.data`, який містить наступні рядки:

1. `classes = 1`.
2. `train = data/COCO_Person_training.txt`.
3. `valid = data/COCO_Person_validation.txt`.
4. `names = data/COCO_Person.names`.
5. `backup = weights/backup/COCO_Person`.

Значення `classes` позначає кількість класів, які нейронна мережа повинна розпізнавати. Значення `train` та `valid` вказують на шляхи до файлів з відносними шляхами до кожного зображення з навчальної та валідаційної вибірки. Значення `names` вказує на шлях до файлу з назвами класів об’єктів, на яких ми навчаємо нейронну мережу. Значення `backup` вказує на шлях до директорії (необхідно створити її вручну), у яку протягом навчання будуть зберігатися ваги нейронної мережі: ваги зберігаються протягом кожної 100 та 1000 ітерації, а також фінальні та найкращі. Фінальні ваги — це ваги, отримані у кінці навчання, а найкращі — це ваги на ітерації навчання, на якій нейронна мережа продемонструвала найбільшу точність на валідаційній вибірці.

Наступним етапом підготовки файлів для навчання є створення файлів у директорії `data` з відносними шляхами до навчальних та валідаційних зображень: `COCO_Person_training.txt` та `COCO_Person_validation.txt`. Кожен з них повинен містити рядки з відносними шляхами до зображень відповідного датасета. Для автоматизації цього процесу був написаний, покладений у директорію `scripts` та виконаний Python-скрипт (на вхід подається назва директорії датасета).

Лістинг 2 демонструє реалізацію функціоналу створення файлів з відносними шляхами до навчальних та валідаційних зображень. Функція `get_dataset_images_relative_paths` переходить до директорії з датасетом та створює список з відносними шляхами до кожного зображення. Функція `write_dataset_images_relative_paths` переходить до директорії `data` та записує у текстовий файл кожен елемент списку (відносний шлях до зображення) з нового рядка.

Лістинг *2 Реалізація функціоналу створення файлів з відносними шляхами до навчальних та валідаційних зображень*

```
def get_dataset_images_relative_paths(dataset_directory_nam
e):
    os.chdir("..")
    os.chdir(os.path.join("data", dataset_directory_name))
    dataset_images_relative_paths = []
    for dataset_image_name in os.listdir(os.getcwd()):
        if dataset_image_name.endswith(".jpg"):
            dataset_images_relative_paths.append(f"data/{da
taset_directory_name}/{dataset_image_name}")
    return dataset_images_relative_paths

def write_dataset_images_relative_paths(dataset_directory_n
ame, dataset_images_relative_paths):
    os.chdir("..")
    with open(f"{dataset_directory_name}.txt", "w") as data
set_images_relative_paths_file:
        for dataset_image_relative_path in dataset_images_r
elative_paths:
            dataset_images_relative_paths_file.write(f"{dat
aset_image_relative_path}\n")
```

Останній крок при підготовці файлів для навчання — це завантаження файлу з вагами для згорткових шарів попередньо навченої на датасеті COCO моделі YOLOv4-tiny. Слід зауважити, що такий тип навчання називається трансферним (transfer learning), оскільки ми використовуємо знання (ваги згорткових шарів), отримані нейронною мережею при вирішенні подібної задачі

(розпізнавання об’єктів). Такий підхід дозволяє пришвидшити процес навчання та збільшити вірогідність того, що нейронна мережа навчиться ефективно розв’язувати поставлену задачу.

Попередньо навчена на датасеті COCO мережа вміє розпізнавати 80 класів об’єктів, серед яких є клас `Person`. Як зазначає Олексій Бочковський, автор YOLOv4-tiny, кращі результати у розпізнаванні показує та нейронна мережа, для якої було виконано трансферне навчання на необхідних класах об’єктів. До того ж кількість параметрів нейронної мережі, яка розпізнає лише 1 об’єкт, є значно меншою за ту, яка розпізнає 80, а отже вона буде працювати швидше та використовувати менше пам’яті.

З офіційного репозиторія darknet були завантажені та покладені у директорію `weights` ваги для згорткових шарів попередньо навченої на датасеті COCO моделі YOLOv4-tiny: `yolov4-tiny.conv.29`.

3.1.4 Запуск процесу навчання

Після підготовки датасетів та необхідних файлів була виконана наступна команда для запуску процесу навчання: `.\darknet.exe detector train .\data\COCO_Person.data .\cfg\yolov4-tiny-COCO-Person.cfg .\weights\yolov4-tiny.conv.29 -map`.

Як можна побачити, параметрами команди є шляхи до підготовлених на попередньому етапі файлів. Окрім цього, ми додаємо прапорець `-map`, який інструктує Darknet обчислювати точність нейронної мережі на валідаційному датасеті. Процес навчання відображається на графіку, який містить інформацію про величину помилки (loss) та точність (mAP) нейронної мережі. Отримані результати навчання наведені у 4 розділі кваліфікаційної роботи.

3.1.5 Перевірка якості навчання

Після успішного навчання у директорії `weights\backup\COCO_Person` знаходяться ваги, з якими навчена модель продемонструвала найбільшу точність на валідаційному датасеті: `yolov4-`

`tiny-COCO-Person_best.weights`. Для перевірки навченої моделі у конфігураційному файлі `yolov4-tiny-COCO-Person.cfg` значення `batch` та `subdivisions` були змінені на: `batch=1` та `subdivisions=1`. Це необхідно для того, щоб «переключити» модель зі стану навчання у стан використання. Для перевірки якості навчання була виконана наступна команда для розпізнавання людей на тестовому відео: `.\darknet.exe detector demo .\data\COCO_Person.data .\cfg\yolov4-tiny-COCO-Person.cfg .\weights\backup\COCO_Person\yolov4-tiny-COCO-Person_best.weights .\data\test_videos\TownCentre.mp4`.

Успішно навчені на датасетах COCO та Open Images моделі використовується у комп'ютерній системі, розробка та функціонал якої описані у наступному розділі.

3.2 Розробка та функціонал комп'ютерної системи для визначення положення людини в обмеженому просторі у реальному часі

3.2.1 Налаштування середовища для розробки системи

Для розробки системи були використані наступні технології:

1. Мова програмування Python 3.7.9 — для реалізації функціоналу.
2. Фреймворк Qt 5.15.1 (PyQt5) — для створення графічного інтерфейсу користувача, роботи з ресурсами та потоками.
3. Бібліотека OpenCV 4.4.0 (OpenCV-Python) — для обробки відеопотоку з підключеної камери, розпізнавання людини за допомогою навченої нейронної мережі та визначення положення розпізнаної людини відносно заданого обмеженого простору.
4. Бібліотека Shapely 1.7.1 — для визначення того, чи знаходиться розпізнана людина в заданому обмеженому просторі.

Фреймворк Qt та бібліотека OpenCV були використані у вигляді прив’язок (bindings) для мови програмування Python. Найголовнішою причиною використання такого стеку технологій є кросплатформність, оскільки розроблена система повинна працювати на мікрокомп’ютері NVIDIA Jetson Nano, який працює під операційною системою Linux. Також це дає можливість розробляти систему на комп’ютері, на якому було проведено навчання моделей нейронних мереж (працює під операційною системою Windows). Серед вищенаведених технологій слід виділити OpenCV: вона має одну з найбільш ефективних реалізацій YOLOv4-tiny серед інших бібліотек, що надзвичай важливо, оскільки кінцевий пристрій для роботи системи має обмежені ресурси.

Оскільки розробка системи проводилася на комп’ютері, на якому було виконано навчання моделей нейронних мереж, мова програмування Python та бібліотека OpenCV не потребують встановлення. Для встановлення фреймворка Qt та бібліотеки Shapely була виконана наступна команда: `pip install PyQt5 Shapely`.

3.2.2 Налаштування середовища для впровадження системи

В якості кінцевого пристрою для впровадження системи був використаний NVIDIA Jetson Nano — мікрокомп’ютер, орієнтований на задачі штучного інтелекту. Він є молодшим із лінійки Jetson та коштує 99$. Нижче наведені деякі з його характеристик:

1. Чотирьохядерний процесор ARM A57 з тактовою частотою 1.43 ГГц.
2. Графічний процесор на основі мікроархітектури NVIDIA Maxwell із 128 ядрами CUDA.
3. 4 ГБ оперативної пам’яті.
4. Підтримка операційної системи Linux for Tegra (L4T).

Jetson Nano постачається без адаптера живлення, активної системи охолодження та microSD-картки. Для вирішення ресурсномістких задач важливо використовувати адаптер живлення 5V/4A, а також будь-який сумісний вентилятор для охолодження. Для роботи мікрокомп’ютера необхідно встановити

операційну систему на UHS-1 microSD-картку з мінімальним розміром 32 ГБ. Для встановлення та налаштування операційної системи були виконані наступні кроки:

1. Завантажено з офіційного сайту NVIDIA образ JetPack SDK 4.4.1, що включає у себе операційну систему Linux for Tegra, а також бібліотеки для обчислень на графічному процесорі, глибинного навчання, комп’ютерного зору та мультимедіа.
2. Відформатовано microSD-картку Samsung Evo Plus 32 GB за допомогою програми SD Card Formatter.
3. Записано образ операційної системи за допомогою програми balenaEtcher.
4. Вставлена microSD-картка та підключені монітор, клавіатура, миша, камера, адаптер Wi-Fi та живлення.
5. Виконано первинні налаштування системи.
6. Налаштовано VNC-сервер vino для віддаленого доступу.

Окрему увагу слід звернути на камеру, яка підключена до Jetson Nano. До рекомендованих USB-камер для цього мікрокомп’ютера відносяться Logitech C920 та Logitech C270. Перша може працювати у Full HD (1080p) та HD (720p) режимах, а друга — лише у HD (720p). Для роботи системи була використана Logitech C920, ціна якої становить 80$. У якості альтернативи можна використовувати Logitech C270, яка коштує 40$. Насправді, обробка Full HD відеопотоку засобами OpenCV працює досить повільно на Jetson Nano, тому використання Logitech C270 є більш оптимальним.

Після встановлення та налаштування операційної системи потрібно інсталювати бібліотеки, необхідні для роботи розробленої комп’ютерної системи. Мікрокомп'ютер Jetson Nano постачається зі встановленою бібліотекою OpenCV, яка не має підтримки CUDA, тому необхідно її скомпілювати з відповідними прапорцями CMake. Також він постачається зі встановленою мовою програмування Python 3.6.9, версія якої підходить для повноцінної роботи

розробленої системи. Для встановлення необхідних бібліотек були виконані наступні кроки:

1. Виконано команду для видалення поточної версії бібліотеки OpenCV: `sudo apt-get purge -y libopencv*`.
2. Завантажено репозиторій nano_build_opencv з GitHub (ім'я користувача: mdegans), який включає у себе скрипт для встановлення необхідних залежностей та компіляції OpenCV з підтримкою CUDA.
3. Виконано команду для встановлення OpenCV 4.4.0 з підтримкою CUDA: `./build_opencv.sh 4.4.0`.
4. Виконано команду для встановлення пакетів `pyqt5` та `shapely`: `sudo apt-get update && sudo apt-get install python3-pyqt5 python3-shapely`.

3.2.3 Програмна архітектура

На рисунку 23 зображена діаграма компонентів розробленої системи. Кожен компонент представляє собою модуль Python.

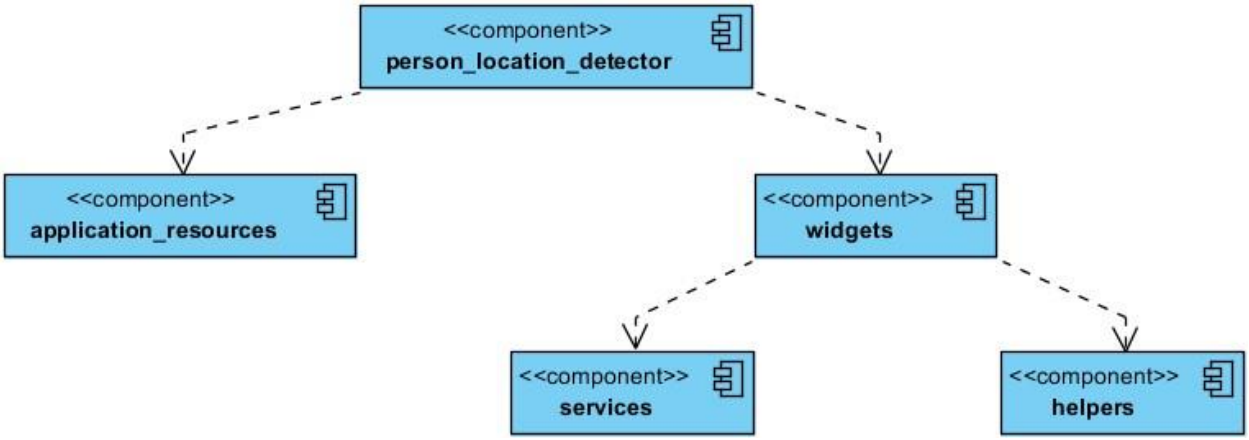

Рис. *23 Діаграма компонентів системи*

Кожен модуль містить відповідну частину функціоналу системи:

1. Модуль `person_location_detector` створює об'єкт застосунку, ініціалізує ресурси, створює та відображає віджет головного вікна, а

також запускає основний цикл роботи програми, у якому отримані від віконної системи події розподіляються по віджетам застосунку.

2. Модуль `application_resources` — згенерований за допомогою утиліти `pyrcc5` модуль з ресурсами (шрифтом та зображеннями).
3. Модуль `widgets` містить класи віджетів, які представляють собою елементи графічного інтерфейсу користувача.
4. Модуль `services` містить класи, що реалізують основний функціонал роботи системи.
5. Модуль `helpers` містить допоміжні функції.

Модуль `person_location_detector` залежить від модулів `application_resources`, та `widgets`. Перший потрібен для того, щоб ініціалізувати ресурси при запуску додатку, а другий містить клас віджета головного вікна, об'єкт якого необхідно створити при старті. Модуль `widgets` залежить від модулів `services` та `helpers`. Таким чином функціонал системи відділяється від інтерфейсу.

На рисунку 24 зображена діаграма класів модуля `services`.

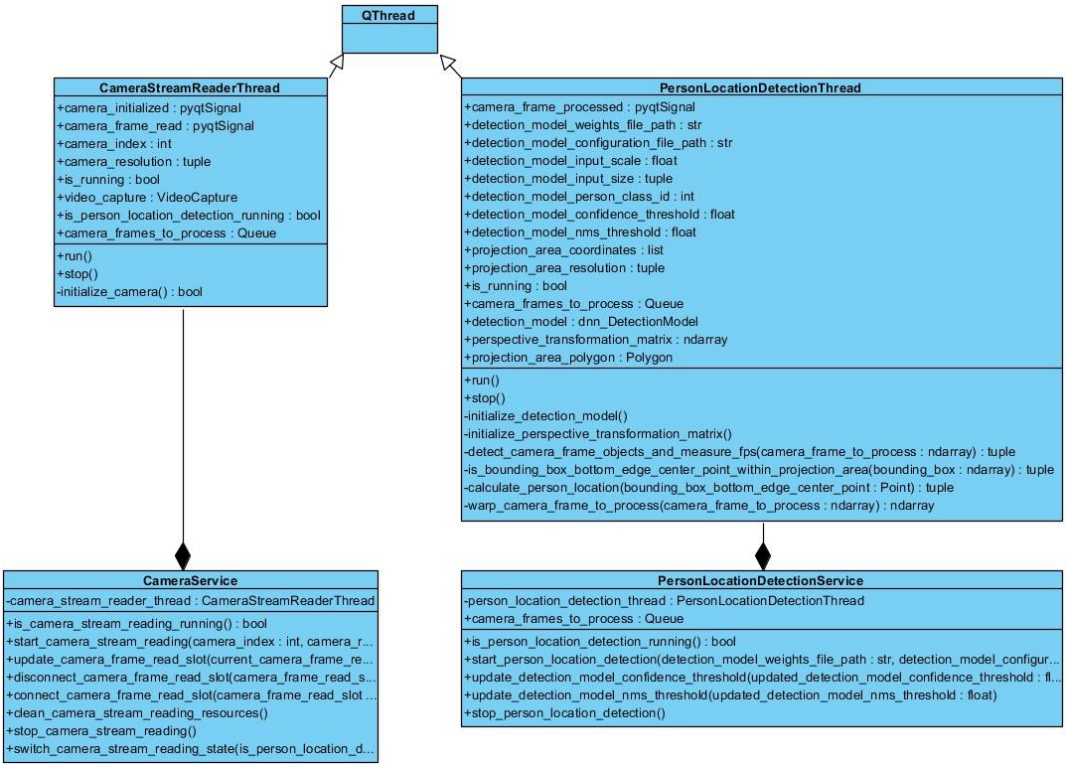

Рис. *24 Діаграма класів модуля services*

Основний функціонал роботи системи реалізований за допомогою двох класів: `CameraService` та `PersonLocationDetectionService`. Кожен з них має приватне поле, у якому зберігається посилання на відповідний потік: `CameraStreamReaderThread` та `PersonLocationDetectionThread`. Сервіси мають публічні методи для управління потоком. Кожен потік є потомком класу `QThread`. У потоці `CameraStreamReaderThread` відбувається ініціалізація підключеної камери та читання відеопотоку з неї. У потоці `PersonLocationDetectionThread` відбувається визначення положення людей в обмеженому просторі на кожному кадрі з камери. Для того, щоб організувати взаємодію між потоками, кожен з них має посилання на чергу — `camera_frames_to_process`. Коли користувач запускає потік для розпізнавання, потік роботи з камерою починає додавати кожен наступний кадр у чергу та чекати на його обробку. Потік визначення положення отримує кадр із черги, визначає положення людей на ньому та надсилає сигнал про кінець обробки. Детальна реалізація основного функціоналу наведена у розділі 3.2.4.

3.2.4 Реалізація основного функціоналу

Лістинг 3 демонструє реалізацію функціоналу ініціалізації камери, читання кадрів із неї, а також додавання кадрів у чергу для визначення положення людей. Даний функціонал відноситься до класу потоку `CameraStreamReaderThread`. Метод `__initialize_camera` створює об’єкт класу `VideoCapture` (з бібліотеки OpenCV), передаючи індекс підключеної камери. Якщо камера була успішно ініціалізована, задається її роздільна здатність. Метод `run` ініціалізує камеру та надсилає сигнал зі статусом ініціалізації. Якщо камера була ініціалізована успішно, запускається цикл зчитування кадрів із неї та надсилання сигналу з кожним успішно зчитаним кадром. Якщо потік для визначення положення людини був запущений, метод додає кожен кадр у чергу та чекає на його обробку. Після виходу з циклу виконується звільнення ресурсів камери.

Лістинг *3 Реалізація функціоналу ініціалізації камери, читання кадрів із неї, а також додавання кадрів у чергу для визначення положення людей*

```
def __initialize_camera(self):
    self.video_capture = cv.VideoCapture(self.camera_index)
    if not self.video_capture.isOpened():
        return False
    self.video_capture.set(cv.CAP_PROP_FRAME_WIDTH,
self.camera_resolution[0])
    self.video_capture.set(cv.CAP_PROP_FRAME_HEIGHT,
self.camera_resolution[1])
    return True

def run(self):
    self.is_running = True
    if not self.__initialize_camera():
        self.camera_initialized.emit(False)
        self.is_running = False
        self.video_capture = None
        return
    self.camera_initialized.emit(True)
    while self.is_running:
        is_successful_camera_frame_read, camera_frame =
self.video_capture.read()
        if is_successful_camera_frame_read:
            self.camera_frame_read.emit(camera_frame)
            if self.is_person_location_detection_running:

self.camera_frames_to_process.put(camera_frame)
                self.camera_frames_to_process.join()
    self.video_capture.release()
```

Лістинг 4 демонструє реалізацію функціоналу ініціалізації навченої моделі нейронної мережі для розпізнавання людей. Метод `__initialize_detection_model` створює об’єкт класу `dnn_DetectionModel` (з бібліотеки OpenCV), передаючи шлях до файлу з навченими вагами нейронної мережі та шлях до файлу з її конфігурацією. Після цього задаються параметри, щоб модель працювала на графічному процесорі з підтримкою CUDA. Останнім етапом виконується ініціалізація параметрів вхідного зображення. Перше значення — це множник для значень пікселів, який дорівнює 1 / 255, оскільки YOLO використовує нормалізовані значення.

Друге значення — це розмір нейронної мережі, який дорівнює 416×416 для відповідності розміру навченої моделі.

Лістинг *4 Реалізація функціоналу ініціалізації навченої моделі нейронної мережі для розпізнавання людей*

```
def __initialize_detection_model(self):
    self.detection_model =
cv.dnn_DetectionModel(self.detection_model_weights_file_pat
h,

self.detection_model_configuration_file_path)

self.detection_model.setPreferableBackend(cv.dnn.DNN_BACKEN
D_CUDA)

self.detection_model.setPreferableTarget(cv.dnn.DNN_TARGET_
CUDA)

self.detection_model.setInputParams(self.detection_model_in
put_scale, self.detection_model_input_size)
```

Лістинг 5 демонструє реалізацію функціоналу ініціалізації матриці для перспективного перетворення.

Метод `__initialize_perspective_transformation_matrix` ініціалізує дану матрицю за допомогою функції `getPerspectiveTransform` (з бібліотеки OpenCV) по 4 парам точок. Перші 4 точки — це координати області проекції з перспективи камери. Вони задаються за годинниковою стрілкою, починаючи з правої верхньої точки. Інші 4 точки — це координати дійсної області проекції на підлогу. Вони задаються подібним чином та залежать від роздільної здатності проектора. Якщо проектор має роздільну здатність 1920×1080, координати дійсної області проекції будуть наступними: (1920; 0), (1920; 1080), (0; 1080), (0; 0).

Лістинг *5 Реалізація функціоналу ініціалізації матриці для перспективного перетворення*

```
def __initialize_perspective_transformation_matrix(self):
    source_points =
np.float32(self.projection_area_coordinates)
    destination_points =
np.float32([[self.projection_area_resolution[0], 0],
self.projection_area_resolution,
                                        [0,
self.projection_area_resolution[1]], [0, 0]])
    self.perspective_transformation_matrix =
cv.getPerspectiveTransform(source_points,
destination_points)
```

Лістинг 6 демонструє реалізацію функціоналу розпізнавання об’єктів у кадрі та визначення швидкості роботи нейронної мережі. Метод `__detect_camera_frame_objects_and_measure_fps` використовує об’єкт навченої нейронної мережі для детектування людей у кадрі. На виході ми отримуємо ідентифікатори класів, впевненості та обмежувальні рамки розпізнаних об’єктів. Метод `detect` повертає тільки ті об’єкти, значення впевненості яких вищі за порогове, а зайві обмежувальні рамки відсіюються за допомогою порогового значення алгоритму non-maximum suppression. Швидкість роботи нейронної мережі (FPS) вимірюється як одиниця, поділена на різницю часу кінця та початку розпізнавання. Різниця часу кінця та початку розпізнавання — це дійсне значення у секундах.

Лістинг *6 Реалізація функціоналу розпізнавання об’єктів у кадрі та визначення швидкості роботи нейронної мережі*

```
def __detect_camera_frame_objects_and_measure_fps(self,
camera_frame_to_process):
    start_detection_time = time.time()
    class_ids, confidences, bounding_boxes =
self.detection_model.detect(
        camera_frame_to_process,
self.detection_model_confidence_threshold,
self.detection_model_nms_threshold)
    end_detection_time = time.time()
    fps_number = 1 / (end_detection_time -
```

```
start_detection_time)
    return class_ids, confidences, bounding_boxes,
fps_number
```

Лістинг 7 демонструє реалізацію функціоналу визначення того, чи знаходиться розпізнана людина в області проекції з перспективи камери. Для цього використовуються об'єкти класів `Point` (положення людини) та `Polygon` (область проекції) з бібліотеки Shapely. Метод `__is_bounding_box_bottom_edge_center_point_within_projection_area` визначає координату середини нижньої грані обмежувальної рамки навколо розпізнаної людини. Ця точка визначає її положення. Після цього він визначає за допомогою методу `within`, чи знаходиться дана точка у чотирикутнику області проекції.

Лістинг *7 Реалізація функціоналу визначення того, чи знаходиться розпізнана людина в області проекції з перспективи камери*

```
def
__is_bounding_box_bottom_edge_center_point_within_projectio
n_area(self, bounding_box):
    bounding_box_bottom_edge_center_point =
Point((bounding_box[0] + bounding_box[2] / 2,

bounding_box[1] + bounding_box[3]))
    return (bounding_box_bottom_edge_center_point,

bounding_box_bottom_edge_center_point.within(self.projectio
n_area_polygon))
```

За формулою 3.4 можна визначити координати точки у перспективі, де M_{ij} — елемент матриці для перспективного перетворення, а $(x;\ y)$ — початкові координати точки.

$$(\acute{x}\ ,\acute{y}) = (\frac{M_{11}\times x+M_{12}\times y+M_{13}}{M_{31}\times x+M_{32}\times y+M_{33}}, \frac{M_{21}\times x+M_{22}\times y+M_{23}}{M_{31}\times x+M_{32}\times y+M_{33}}) \quad (3.4)$$

Лістинг 8 демонструє реалізацію функціоналу визначення положення людини відносно області проекції на підлогу. Метод `__calculate_person_location` визначає координату положення людини відносно області проекції на підлогу, використовуючи формулу 3.4.

Лістинг *8 Реалізація функціоналу визначення положення людини відносно області проекції на підлогу*

```
def __calculate_person_location(self,
bounding_box_bottom_edge_center_point):
    return ((self.perspective_transformation_matrix[0][0] *
bounding_box_bottom_edge_center_point.x +
             self.perspective_transformation_matrix[0][1] *
bounding_box_bottom_edge_center_point.y +
             self.perspective_transformation_matrix[0][2])
/
            (self.perspective_transformation_matrix[2][0] *
bounding_box_bottom_edge_center_point.x +
             self.perspective_transformation_matrix[2][1] *
bounding_box_bottom_edge_center_point.y +
             self.perspective_transformation_matrix[2][2]),
            (self.perspective_transformation_matrix[1][0] *
bounding_box_bottom_edge_center_point.x +
             self.perspective_transformation_matrix[1][1] *
bounding_box_bottom_edge_center_point.y +
             self.perspective_transformation_matrix[1][2])
/
            (self.perspective_transformation_matrix[2][0] *
bounding_box_bottom_edge_center_point.x +
             self.perspective_transformation_matrix[2][1] *
bounding_box_bottom_edge_center_point.y +
             self.perspective_transformation_matrix[2][2]))
```

Лістинг 9 демонструє реалізацію функціоналу перетворення кадру з камери у перспективі. Метод `__warp_camera_frame_to_process` використовує функцію `warpPerspective` (з бібліотеки OpenCV) для перетворення кадру з камери таким чином, що перетворений кадр — це вид зверху на область проекції.

Лістинг *9 Реалізація функціоналу перетворення кадру з камери у перспективі*

```
def __warp_camera_frame_to_process(self,
camera_frame_to_process):
    return cv.warpPerspective(camera_frame_to_process,
self.perspective_transformation_matrix,

self.projection_area_resolution)
```

Лістинг 10 демонструє реалізацію функціоналу визначення положення людей в обмеженому просторі. Даний функціонал відноситься до класу потоку `PersonLocationDetectionThread`. Метод `run` послідовно викликає методи, наведені у лістингах 4-9. Він ініціалізує навчену модель нейронної мережі та матрицю для перспективного перетворення, а також створює об’єкт класу `Polygon`, який визначає чотирикутник області проекції. В основному циклі метод отримує наступний кадр із камери. Далі він розпізнає об’єкти на ньому та визначає значення FPS. Для кожної розпізнаної людини метод визначає, чи знаходиться вона в області проекції з перспективи камери. Якщо знаходиться, він визначає її положення відносно області проекції на підлогу. Впевненість, обмежувальна рамка та положення людини додаються у результуючі списки. Після цього він перетворює кадр із камери у перспективі, надсилає сигнал з результатами обробки та оповіщає потік роботи із камерою про те, що кадр був оброблений.

Лістинг *10 Реалізація функціоналу визначення положення людей в обмеженому просторі*

```
def run(self):
    self.is_running = True
    self.__initialize_detection_model()
    self.__initialize_perspective_transformation_matrix()
    self.projection_area_polygon =
Polygon(self.projection_area_coordinates)
    while self.is_running:
        try:
            camera_frame_to_process =
self.camera_frames_to_process.get_nowait()
        except queue.Empty:
            continue
```

```
        class_ids, confidences, bounding_boxes, fps_number
= self.__detect_camera_frame_objects_and_measure_fps(
            camera_frame_to_process)
        result_confidences = []
        result_bounding_boxes = []
        result_persons_locations = []
        for (class_id, confidence, bounding_box) in
zip(class_ids, confidences, bounding_boxes):
            if class_id[0] !=
self.detection_model_person_class_id:
                continue
            bounding_box_bottom_edge_center_point,
is_within_projection_area = \

self.__is_bounding_box_bottom_edge_center_point_within_proj
ection_area(bounding_box)
            if not is_within_projection_area:
                continue
            result_confidences.append(confidence[0])
            result_bounding_boxes.append(bounding_box)

result_persons_locations.append(self.__calculate_person_loc
ation(bounding_box_bottom_edge_center_point))
        camera_frame_to_process_warped =
self.__warp_camera_frame_to_process(camera_frame_to_process
)

self.camera_frame_processed.emit((camera_frame_to_process,
camera_frame_to_process_warped, fps_number,

result_confidences, result_bounding_boxes,
result_persons_locations))
        self.camera_frames_to_process.task_done()
```

3.2.5 Графічний інтерфейс користувача та функціональні можливості

Застосунок включає у себе меню, сторінки Detection та About. На рисунку 25 зображений графічний інтерфейс користувача сторінки Detection з позначеними елементами.

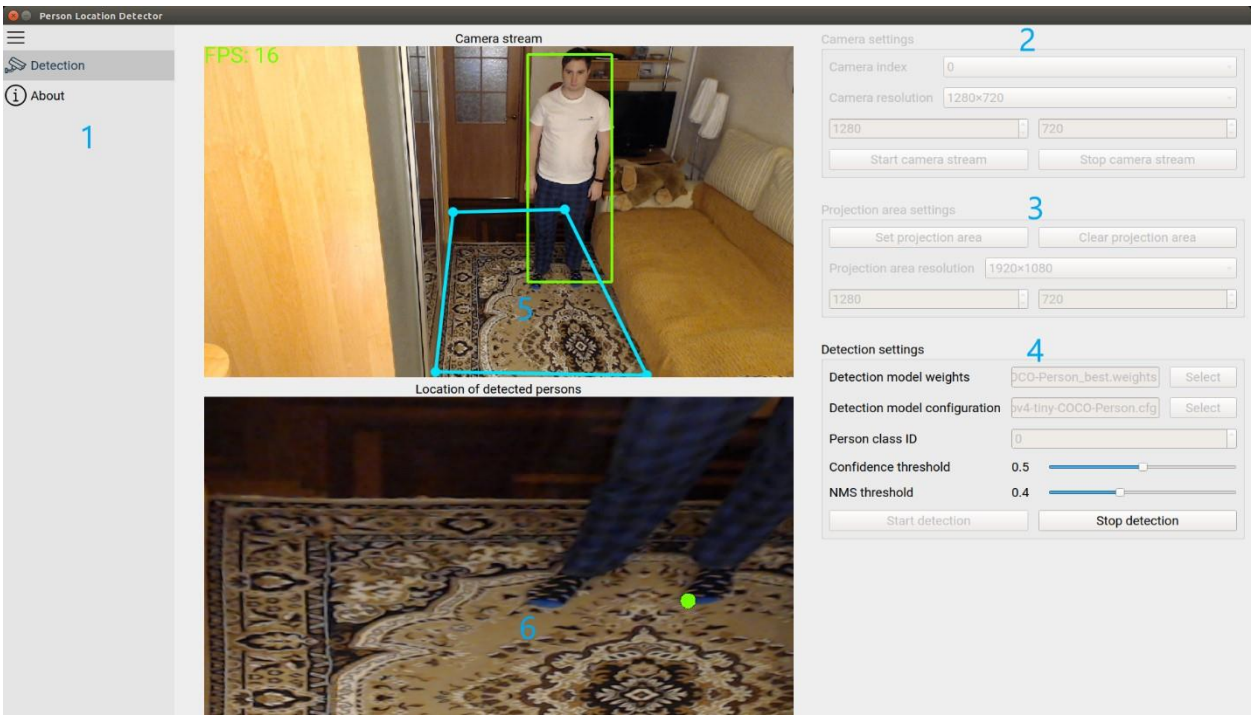

Рис. *25 Графічний інтерфейс користувача сторінки Detection*

Сторінка Detection складається з наступних елементів:

1. Меню.
2. Блок управління камерою.
3. Блок управління областю проекції.
4. Блок управління визначенням положення людини.
5. Блок відображення відеопотоку з підключеної камери.
6. Блок відображення відеопотоку для встановленої області проекції.

Меню представляє собою список із назвами сторінок застосунку та відображається на кожній сторінці. Воно має функціонал згортання та розгортання, щоб основний контент міг займати більше місця. Коли користувач натискає на елемент у меню, він переходить на відповідну сторінку.

Блок управління камерою дозволяє обрати індекс підключеної камери, її роздільну здатність та запустити чи зупинити відеопотік. Індекс підключеної камери може бути від 0 до 4: система підтримує до 5 підключених камер, між якими можна перемикатися. Для роздільної здатності підключеної камери передбачені наступні значення: 1920×1080, 1280×720, 640×480 та Other.

Якщо користувач обрав Other, він може встановити необхідне значення у полях знизу. Запущений користувачем відеопотік відображається зліва.

Блок управління областю проекції дозволяє встановити чи видалити чотирикутник проекції, а також обрати роздільну здатність, яку має проектор. Цей блок стає активним тільки у тому випадку, якщо запущений відеопотік з камери. Для того, щоб встановити область проекції, користувач повинен визначити 4 її крайні точки за годинниковою стрілкою, починаючи з правої верхньої. Для роздільної здатності проектора передбачений ідентичний роздільній здатності камери функціонал.

Блок управління визначенням положення людини дозволяє обрати файли з вагами та конфігурацією нейронної мережі, яка здатна розпізнавати людей. Також він дозволяє встановити ідентифікатор класу «людина», який може бути не 0 у випадку, якщо нейронна мережа може розпізнавати декілька класів об’єктів. Окрім цього, користувач може задати порогові значення для впевненості нейронної мережі та алгоритму non-maximum suppression, а також запустити чи зупинити процес визначення положення людини. Блок управління визначенням положення людини стає активним тільки у тому випадку, якщо встановлений чотирикутник області проекції на підлогу.

Блок відображення відеопотоку з підключеної камери відображає кадри, прочитані з неї, а також встановлену область проекції небесного кольору. Якщо визначення положення людини запущено, він відображає обмежувальні рамки навколо розпізнаних людей та значення FPS зеленим кольором.

Блок відображення відеопотоку для встановленої області проекції відображає вид зверху (у перспективі) на область проекції, а також визначені положення людей, які знаходяться у цій області. Положення людей — це точки зеленого кольору.

На рисунку 26 зображений графічний інтерфейс користувача сторінки About, яка містить інформацію про розробника.

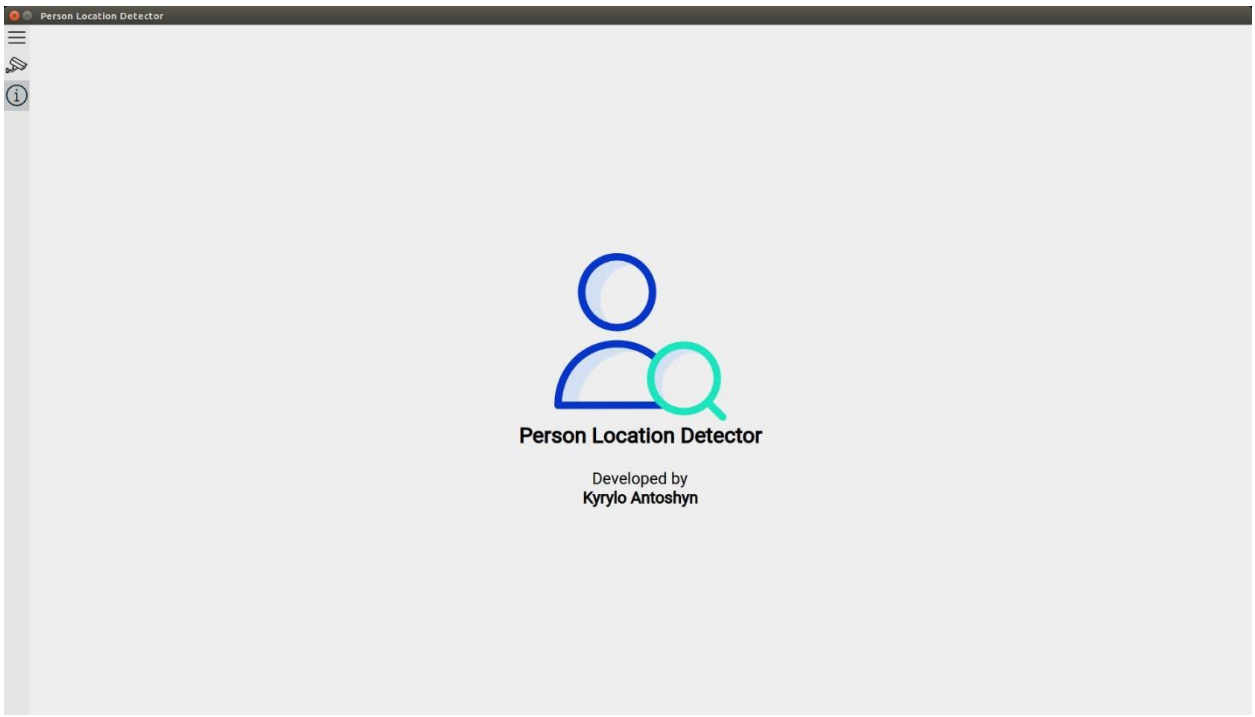

Рис. *26 Графічний інтерфейс користувача сторінки About*

3.3 Висновки з розділу 3

1. Було налаштоване середовище для навчання моделі нейронної мережі на базі YOLOv4-tiny за допомогою фреймворка Darknet.

2. У налаштованому середовищі були підготовлені датасети із зображеннями людей (COCO та Open Images) та файли для навчання. Окрім цього, були написані скрипти, які автоматизують певні кроки процесу підготовки до навчання: завантаження зображень з датасета COCO та генерація відносних шляхів до кожного зображення відповідного датасета.

3. Виконано навчання двох нейронних мереж: одна навчалася на датасеті COCO, а інша — на Open Images. Була протестована якість розпізнавання.

4. Було налаштоване середовище для розробки та впровадження системи на базі мікрокомп’ютера NVIDIA Jetson Nano.

5. Була розроблена комп’ютерна система, яка визначає положення людей в обмеженому просторі у реальному часі на основі відеопотоку з підключеної камери. Були описані її програмна архітектура, реалізація основного функціоналу, графічний інтерфейс користувача та функціональні можливості.

РОЗДІЛ 4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ РОБОТИ КОМП'ЮТЕРНОЇ СИСТЕМИ ДЛЯ ВИЗНАЧЕННЯ ПОЛОЖЕННЯ ЛЮДИНИ В ОБМЕЖЕНОМУ ПРОСТОРІ У РЕАЛЬНОМУ ЧАСІ

4.1 Результати навчання моделей нейронних мереж для розпізнавання людей

4.1.1 Аналіз результатів навчання моделей нейронних мереж

У процесі розробки комп'ютерної системи були навчені 2 моделі нейронних мереж: одна навчалася на датасеті COCO, а інша — на датасеті Open Images. Під час навчання кожної з них фреймворк Darknet побудував графік залежності значень помилки та точності нейронної мережі від номера батча. На рисунку 27 наведений графік навчання моделі нейронної мережі на датасеті COCO.

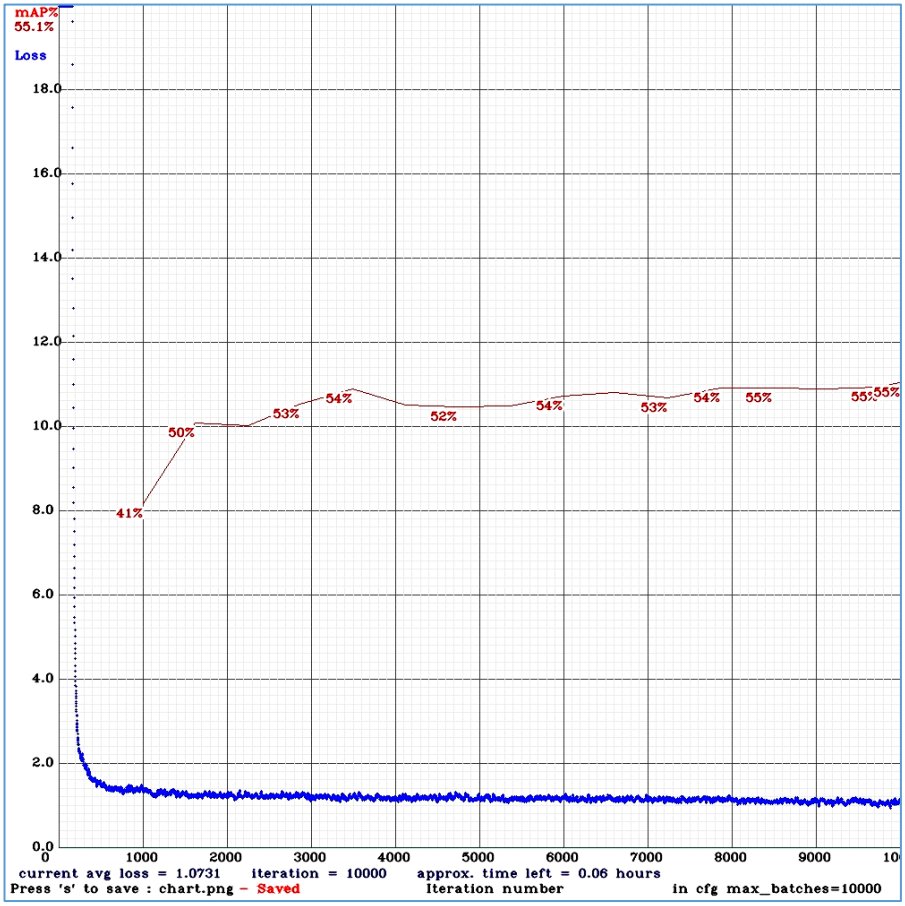

Рис. *27 Графік навчання моделі нейронної мережі на датасеті COCO*

Як можна побачити на графіку, кінцева помилка (синім кольором) нейронної мережі становить 1.07, а точність (червоним кольором) — 55.1%. Явно видно, що приблизно на 1000 ітерації була отримана точність у 41%, на 3500 ітерації — 54%, потім впала до 52% і лише під кінець навчання ми отримали 55.1%. Коли точність моделі падає, відбувається перенавчання моделі — процес, коли вона добре розпізнає об’єкти на навчальній вибірці, але починає погано розпізнавати на валідаційній. Окрім цього можна побачити, що знадобилося 6500 ітерацій (від 3500 до 10000) для того, щоб збільшити точність всього на 1%. Можна зробити висновок, що для цієї моделі та датасета оптимальним є навчання протягом 3500 батчів. Ми програємо лише 1% у точності, але заощадимо декілька годин часу. Фреймворк Darknet дозволяє не орієнтуватися на графік, зберігаючи ваги моделі з найбільшою точністю.

На рисунку 28 наведений графік навчання моделі нейронної мережі на датасеті Open Images.

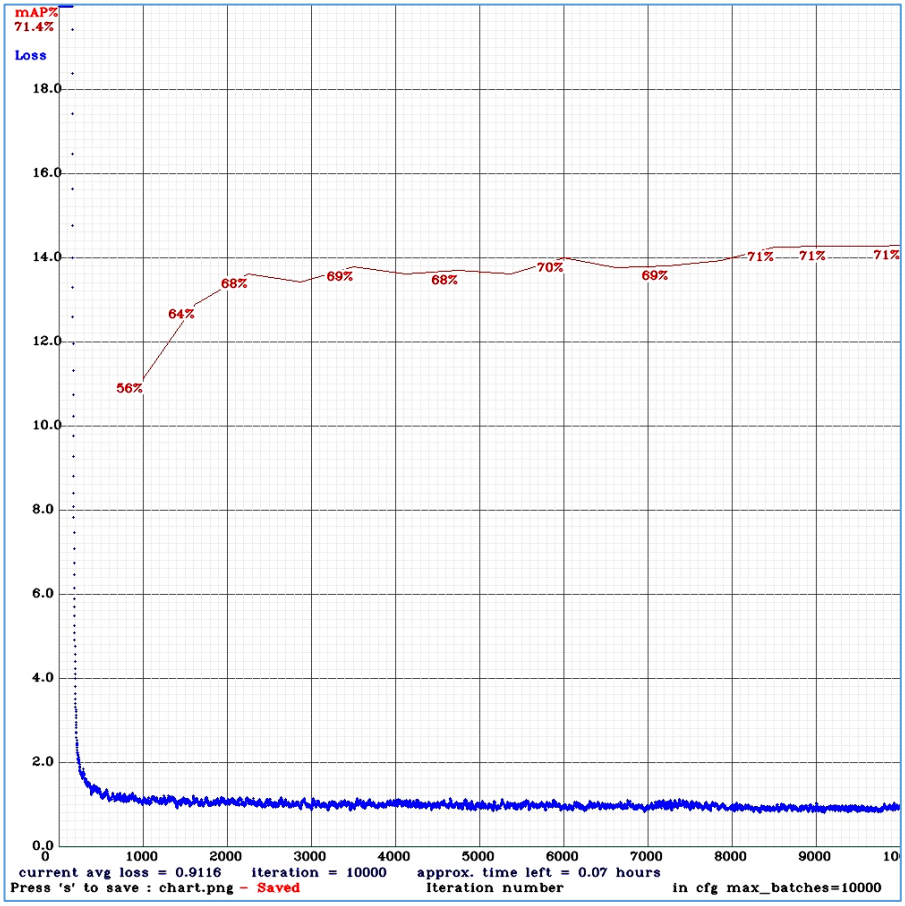

Рис. *28 Графік навчання моделі нейронної мережі на датасеті Open Images*

Як можна побачити на графіку, кінцева помилка нейронної мережі становить 0.91, а точність — 71.4%. Явно видно, що приблизно на 1000 ітерації була отримана точність у 56%, на 3500 ітерації — 69%, потім впала до 68% і лише під кінець навчання ми отримали 71.4%. Окрім цього можна побачити, що знадобилося 6500 ітерацій (від 3500 до 10000) для того, щоб збільшити точність всього на 2%. Можна зробити висновок, що для цієї моделі та датасета також оптимальним є навчання протягом 3500 батчів. Ми програємо лише 2% у точності, але заощадимо декілька годин часу.

У таблиці 7 наведено порівняння результатів навчання моделей нейронних мереж на датасетах COCO та Open Images.

Таблиця *7*

Порівняння результатів навчання моделей нейронних мереж на датасетах COCO та Open Images

Критерій / Модель	**YOLOv4-tiny-Person (COCO)**	**YOLOv4-tiny-Person (Open Images)**
Час навчання (год)	6:20	7:02
Початкова помилка	373.36	341.71
Кінцева помилка	1.07	0.91
Початкова точність (%)	41	56
Кінцева точність (%)	55.1	71.4
Оптимальна кількість батчів для навчання	3500	3500

З наведених у таблиці даних можна побачити, що моделі, яка навчалася на датасеті Open Images, знадобилося на 40 хвилин більше часу. Однією з причин цього є більший розмір зображень датасета Open Images: навчальні та валідаційні зображення датасета COCO займають 1.85 ГБ пам’яті на диску, а

Open Images — 3.52 ГБ. Меншу помилку та більшу точність демонструє модель, яка навчалася на датасеті Open Images. Однією із причин цього може бути те, що датасет COCO містить зображення з більш складними сценаріями, тому нейронній мережі важче виконувати узагальнення, тобто розпізнавати об’єкти на валідаційних зображеннях. Ще однією причиною низького проценту точності розпізнавання при навчанні є присутність у валідаційних зображеннях значного шуму, оскільки додаткова обробка датасетів не виконувалась. Обидві моделі мають однакову оптимальну кількість батчів для навчання. Це можна пояснити однаковою кількістю навчальних та валідаційних зображень, а також вирішенням однакової задачі — розпізнавання людини.

4.1.2 Аналіз точності роботи навчених моделей нейронних мереж

На рисунках 29 та 30 зображені виявлені на тестовому відео з людьми Oxford Town Centre проблеми точності роботи навчених на датасетах COCO та Open Images моделей нейронних мереж. У таблиці 8 наведено їх порівняння (цифра у клітинці означає номер пункту на відповідному рисунку).

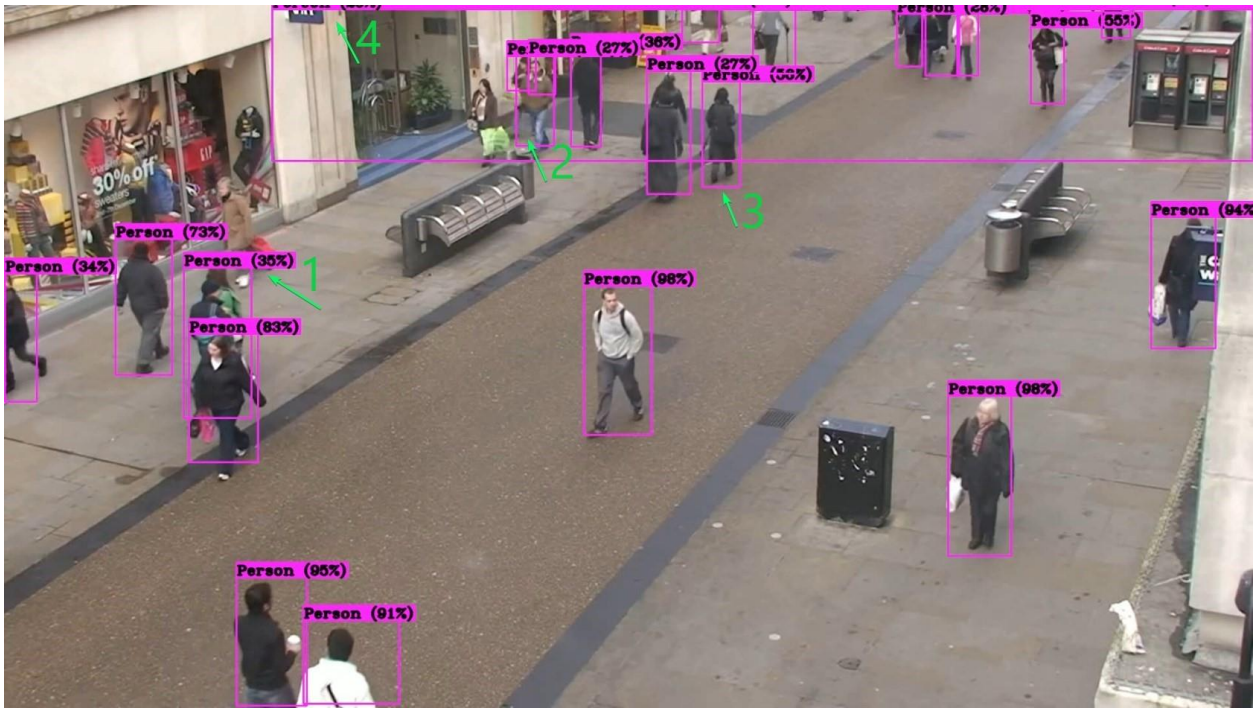

Рис. *29 Проблеми точності роботи навченої на датасеті COCO моделі нейронної мережі*

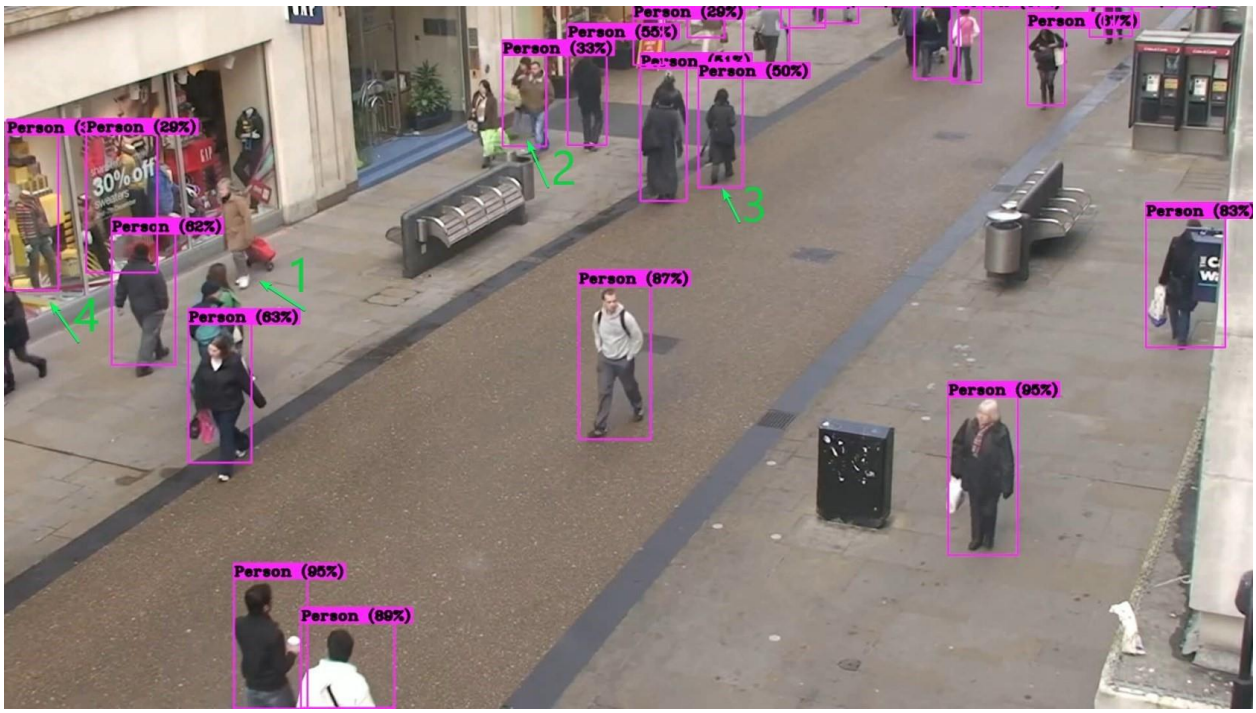

Рис. *30 Проблеми точності роботи навченої на датасеті Open Images моделі нейронної мережі*

Таблиця *8*

Порівняння проблем точності роботи навчених на датасетах COCO та Open Images моделей нейронних мереж

Проблема / Модель	**YOLOv4-tiny-Person (COCO)**	**YOLOv4-tiny-Person (Open Images)**
Не виявлення людини	1	1
Не виявлення перекритої іншим об’єктом людини	2	2
Визначення обмежувальної рамки, яка не щільно прилягає	3	3
Розпізнавання групи людей, як одну людину	4	Не виявлено
Розпізнавання не справжніх людей (манекенів)	Не виявлено	4

З таблиці видно, що обидві моделі мають наступні спільні проблеми точності роботи: не виявлення людини, не виявлення перекритої іншим об’єктом людини, визначення обмежувальної рамки, яка не щільно прилягає. Проблема з обмежувальною рамкою виділена через те, що ми вважаємо за положення людини трансформовану відносно області проекції центральну точку її нижньої грані. Якщо рамка не щільно прилягає, положення людини буде визначено ближче до камери, аніж є насправді.

Також існують проблеми при розпізнаванні, які притаманні кожній моделі окремо: модель, навчена на датасеті COCO, іноді розпізнає групу людей, як одну людину, а модель, навчена на датасеті Open Images, іноді розпізнає не справжніх людей. За допомогою інструменту LabelImg, завантаженого з GitHub (ім’я користувача: tzutalin), були перевірені навчальні зображення обох датасетів. Було виявлено, що датасет COCO містить анотації для групи людей, а Open Images — для не справжніх людей: манекенів, персонажів відеоігор, іграшок тощо. Для вирішення обох проблем необхідно вручну видалити зайві анотації та зображення. Автор YOLOv4-tiny зазначає, що модель буде працювати точніше, якщо зібрати та анотувати 2000-10000 зображень на локації, де буде встановлена камера. Такий підхід також вирішує дані проблеми.

Модель, навчена на датасеті COCO, показала на 16% меншу точність на валідаційній вибірці у порівнянні з моделлю, яка навчена на датасеті Open Images. При тестуванні обох моделей було виявлено, що вони працюють приблизно на одному рівні. З цього можна зробити висновок, що при навчанні моделі нейронної мережі на різних датасетах необхідно орієнтуватися не тільки на точність, але й на якість її роботи у реальних умовах, оскільки один із датасетів може бути складнішим.

У якості моделі, яку можна використовувати у розробленій комп’ютерній системі, можна обрати будь-яку з навчених, однак спочатку необхідно визначити, яка з них краще працює на локації, де буде встановлена камера. Якщо локація містить багато людей в костюмах супергероїв чи роботів, кращим вибором буде використання моделі, навченої на датасеті COCO, а якщо локація

— це переповнене людьми місце, кращим вибором буде використання моделі, навченої на датасеті Open Images.

4.1.3 Варіанти покращення точності роботи моделі нейронної мережі

Нижче наведені деякі рекомендації автора моделі YOLOv4-tiny для отримання кращих результатів при навчанні моделі:

1. Змінити значення `random` у кожному `yolo`-шарі на `random=1`. Це інструктує фреймворк Darknet випадковим чином змінювати розмір моделі, навчаючи її на вхідних зображеннях різного розміру.
2. Збільшити розмір моделі, встановивши значення `width` та `height`, наприклад, `width=608` та `height=608`.
3. Упевнитися, що кожен об’єкт у датасеті правильно анотований.
4. Для кожного об’єкта, який модель повинна розпізнавати, навчальний датасет повинен включати хоча б один об’єкт, схожий за формою, стороною, кутом повороту, нахилом та освітленням.
5. Бажано, щоб навчальний датасет включав «негативні» зображення — зображення без об’єктів, які потрібно розпізнавати. Кількість таких зображень повинна дорівнювати кількості зображень з об’єктами.
6. Навчальний датасет повинен містити об’єкти з відносними розмірами тих об’єктів, які необхідно розпізнавати. Тобто, якщо ми хочемо виявляти об’єкти, які займають повний кадр, датасет повинен включати зображення з об’єктами таких розмірів.

Для отримання кращих результатів при розпізнаванні об’єктів з використанням навченої моделі нейронної мережі рекомендується збільшити її розмір, наприклад, `width=608` та `height=608` чи `width=832` та `height=832`, для можливості розпізнавати маленькі об’єкти.

Як можна побачити з наведених рекомендацій, більшість із них стосуються якості датасета. Для їхнього автоматичного виконання необхідно зби-

рати та анотувати зображення на локації, де буде встановлена камера. Наприклад, якщо комп’ютерна система для визначення положення людини в обмеженому просторі буде встановлена у дитячому розважальному центрі, нейронну мережу необхідно навчити розпізнавати як дорослих, так і дітей при наступних умовах: різному освітленні, можливим проекціям зі складними шаблонами на підлогу чи стіни, різному одязі влітку чи взимку, різних позах, наявності статуй, перекритті іншими об’єктами тощо.

4.2 Результати роботи розробленої комп’ютерної системи для визначення положення людини в обмеженому просторі у реальному часі

4.2.1 Аналіз швидкості розпізнавання людини та використання пам’яті системою

У таблиці 9 наведено порівняння швидкості розпізнавання людини та використання пам’яті системою при використанні різних кінцевих пристроїв та моделей нейронної мережі.

Таблиця 9

Порівняння швидкості розпізнавання людини та використання пам’яті системою при використанні різних кінцевих пристроїв та моделей нейронної мережі

Кінцевий пристрій, модель / Параметр	**Швидкість розпізнавання людини (FPS)**	**Використання пам'яті системою (МБ)**
Комп’ютер для навчання, YOLOv4-tiny (COCO, Person)	83	1055
NVIDIA Jetson Nano, YOLOv4-tiny (COCO, Person)	16	745
Комп’ютер для навчання, YOLOv4 (COCO, 80 класів)	16	1620
NVIDIA Jetson Nano, YOLOv4 (COCO, 80 класів)	1	1200

Дані наведені для HD роздільної здатності камери та Full HD роздільної здатності проектора.

Порівняння проводилося з використанням комп'ютера для навчання (має графічний процесор NVIDIA GeForce GTX 1050) та NVIDIA Jetson Nano, а також моделей YOLOv4-tiny та YOLOv4. З наведених даних видно, що швидкість розпізнавання людини на NVIDIA Jetson Nano з використанням навченої на зображеннях класу Person датасета COCO моделі YOLOv4-tiny становить 16 FPS. Швидкість розпізнавання людини на комп'ютері для навчання з використанням цієї ж моделі становить 83 FPS. Такий результат є цілком очевидним через різницю потужності графічних процесорів обох пристроїв. Також у таблиці наведені дані про швидкість розпізнавання людини з використанням навченої на 80 класах об'єктів датасета COCO моделі YOLOv4. NVIDIA Jetson Nano видає всього 1 FPS, а комп'ютер для навчання — 16 FPS. Такий результат також є цілком очевидним через те, що модель YOLOv4 не здатна працювати пристроях з обмеженими ресурсами.

Цікавою статистикою є те, що на NVIDIA Jetson Nano розроблена система використовує значно менше оперативної пам'яті. Слід також зазначити, що при роботі з нейронними мережами бібліотека OpenCV використовує частину оперативної пам'яті, яку не можна звільнити викликом спеціальної функції чи видаленням посилання на об'єкт нейронної мережі у процесі роботи програми.

4.2.2 Аналіз точності роботи системи при різних сценаріях

Для аналізу точності роботи розробленої системи були перевірені 3 наступні сценарії:

1. Людина стоїть обличчям до камери.
2. Людина повернута спиною до камери.
3. Частина тіла людини перекрита іншим об'єктом.

На рисунку 31 зображений сценарій роботи системи, при якому людина стоїть обличчям до камери. Ліворуч зображена робота системи при використанні моделі нейронної мережі, яка навчена на датасеті COCO, а праворуч — на Open Images. В обох випадках система досить точно визначила положення людини відносно обмеженого простору з приблизною похибкою у 15 см для першої моделі та 30 см — для другої. Похибка вимірюється як різниця між визначеною точкою (позначена зеленим кольором) та точкою, що знаходиться між ногами людини (позначена блакитним кольором).

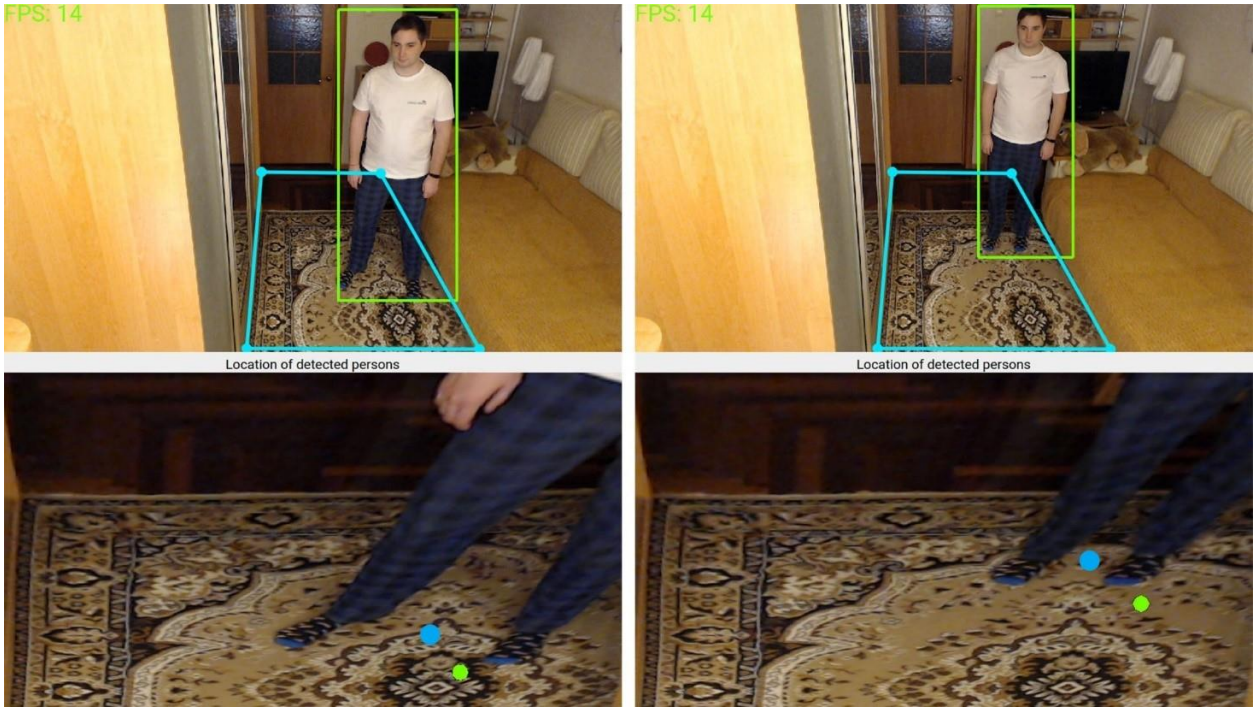

Рис. *31 Сценарій роботи системи, при якому людина стоїть обличчям до камери*

На рисунку 32 зображений сценарій роботи системи, при якому людина повернута спиною до камери. Ліворуч зображена робота системи при використанні моделі нейронної мережі, яка навчена на датасеті COCO, а праворуч — на Open Images. В обох випадках система також досить точно визначила положення людини відносно обмеженого простору з приблизною похибкою у 30 см для обох моделей.

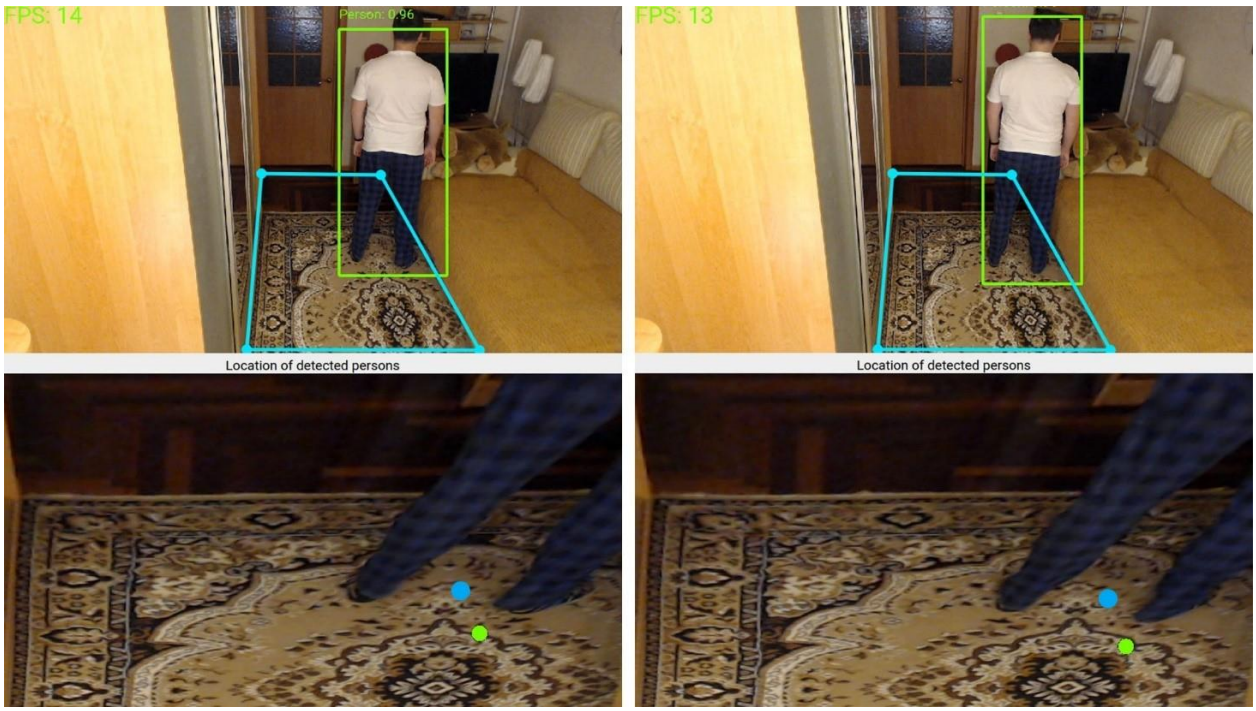

Рис. *32 Сценарій роботи системи, при якому людина повернута спиною до камери*

На рисунку 33 зображений сценарій роботи системи, при якому частина тіла людини перекрита іншим об’єктом. Ліворуч та праворуч зображена робота системи при використанні моделі нейронної мережі, яка навчена на датасеті COCO. Відмінність полягає у тому, що для першого випадку поріг впевненості нейронної мережі становив 0.5, а для другого — 0.9. Система змогла визначити положення людини тільки у першому випадку з приблизною похибкою у 25 см. У другому випадку система не змогла розпізнати людину у кадрі, тому що перекриття частини тіла людини іншим об’єктом значно знижує впевненість нейронної мережі: нейронна мережа не була «впевнена» на 90% та вище, що об’єкт у кадрі — це людина. Тобто, від встановленого значення впевненості нейронної мережі цілком залежить здатність системи визначати положення людини. Встановлене значення повинно бути оптимальним для середовища, у якому буде використовуватися система: при наявності натовпу значення впевненості не повинно бути зависоким.

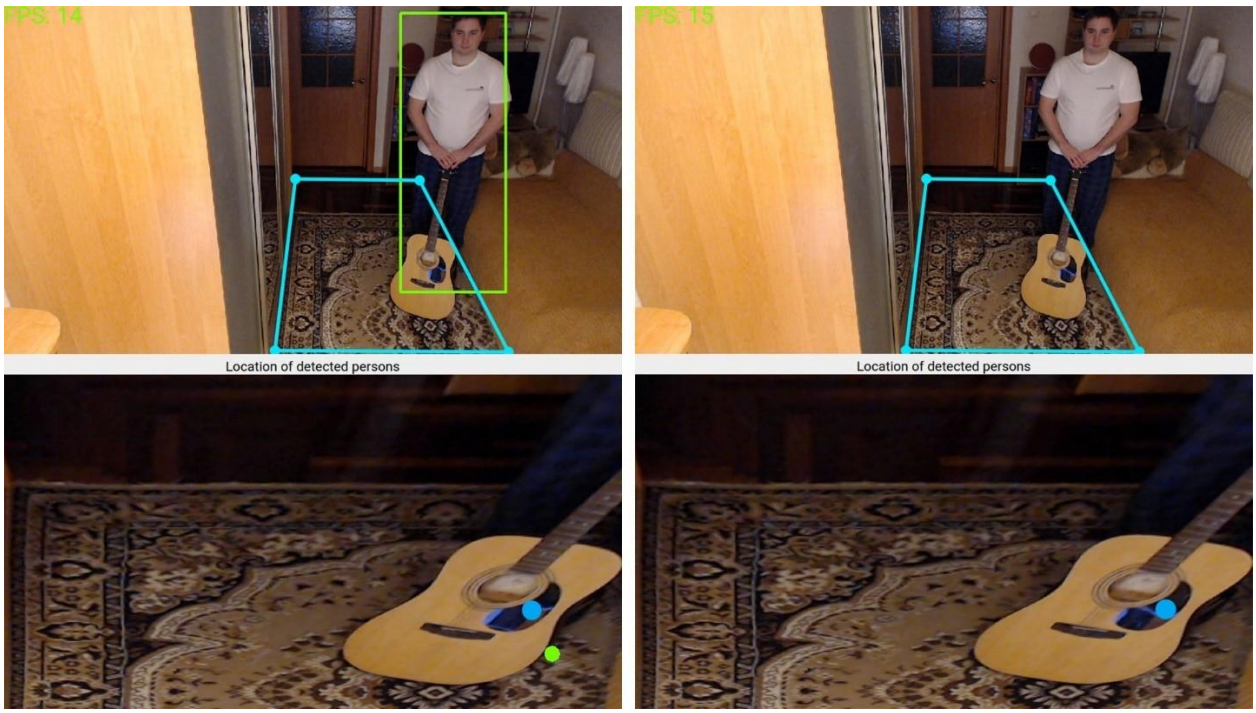

Рис. *33 Сценарій роботи системи, при якому частина тіла людини перекрита іншим об'єктом*

У таблиці 10 наведені значення приблизної похибки роботи системи при різних сценаріях. Для сценарію зазначається його номер та номер відповідного зображення через крапку.

Таблиця *10*

Значення приблизної похибки роботи системи при різних сценаріях

Сценарій	**Приблизна похибка (см)**
1.1	15
1.2	30
2.1	30
2.2	30
3.1	25
3.2	Людина не була розпізнана

З отриманих результатів видно, що для наведених сценаріїв система помиляється у середньому на 26 см. У реальному ж середовищі середнє значення похибки може бути більшим, але не набагато, оскільки навчені моделі нейрон-

них мереж досить точно розпізнають людей у кадрі. Отримане значення середньої похибки системи є прийнятним для багатьох сфер її можливого застосування. Наприклад, для системи Interactive Floor не потрібно знати ідеально точне положення людини відносно області проекції, оскільки під її ногами будуть відтворюватися ефекти, радіус дії яких є значно вищим за можливу похибку.

4.2.3 Варіанти покращення точності роботи системи

Точність роботи розробленої системи насамперед залежить від якості навчання нейронної мережі для розпізнавання людей. Недоліки навчених моделей, висвітлені у розділі 4.1.2 кваліфікаційної роботи, визначають «слабкі» місця системи. Їх можна частково чи повністю усунути, скориставшись рекомендаціями, наведеними у розділі 4.1.3.

Підхід з визначенням обмежувальної рамки має два недоліки. По-перше, обмежувальна рамка не завжди щільно прилягає до тіла людини, що дає похибку при визначенні її положення. По-друге, людина може відставити одну ногу назад і тоді її положення буде визначено відносно передньої ноги, що також є не зовсім точним. Для усунення цих недоліків можна спробувати визначати положення людини, як середню точку між її ногами. Для визначення координат ніг необхідно використовувати методи, які оцінюють позу людини (визначають її ключові точки — кінцівки), наприклад, OpenPose [36]. Такий підхід використовується у системі CamLoc [4] та демонструє високу точність. Підхід з визначенням ключових точок людини потребує більше обчислювальних ресурсів та частіше не виявляє людину у кадрі.

Насправді, більшість із наведених у розділі 1.3 кваліфікаційної роботи сфер застосування розробленої комп’ютерної системи не потребують визначення положення з ідеальною точністю, а й отже підхід з визначенням обмежувальної рамки є більш оптимальним.

4.3 Висновки з розділу 4

1. Модель, яка навчалася на датасеті Open Images, показала більшу точність при навчанні у порівнянні з моделлю, яка навчалася на датасеті COCO. Аналіз точності роботи навчених моделей на тестовому відео показав, що вони працюють приблизно на одному рівні, маючи спільні та індивідуальні недоліки.

2. Для покращення точності роботи навчених моделей існують загальні рекомендації, більшість із яких стосуються якості датасета.

3. Розроблена комп'ютерна система розпізнає людей зі швидкістю 16 FPS та 83 FPS, працюючи на мікрокомп'ютері NVIDIA Jetson Nano та графічному процесорі NVIDIA GeForce GTX 1050 відповідно.

4. Отримана при тестуванні системи середня похибка визначення положення людини становить 26 см.

5. Альтернативним підходом до визначення положення людини є визначення координати між її ногами за допомогою методів оцінки пози людини. Такий підхід потребує більше обчислювальних ресурсів та частіше не виявляє людину у кадрі. Підхід з визначенням обмежувальної рамки є більш оптимальним через те, що більшість сфер застосування розробленої системи не потребують визначення положення з ідеальною точністю.

ВИСНОВКИ

1. Досліджена проблема визначення положення людини в обмеженому просторі у реальному часі: розглянуті переваги та недоліки існуючих технологій та систем для її вирішення, сфери застосування таких систем та сучасні підходи комп'ютерного зору до її розв'язання.

2. Досліджені засоби для вирішення поставленої проблеми: з розглянутих моделей нейронних мереж для розпізнавання людини, здатних працювати на мікрокомп'ютері, обрана найбільш ефективна — YOLOv4-tiny; з розглянутих бібліотек для навчання та використання нейронних мереж обрані оптимальні — Darknet та OpenCV; з розглянутих датасетів із зображеннями людей обрані якісні — COCO та Open Images.

3. Проведено навчання двох моделей нейронних мереж для розпізнавання людини: налаштоване середовище для навчання, підготовлені зображення людей з відповідних датасетів, створені необхідні для тренування файли та успішно виконано навчання обох моделей.

4. Розроблена готова до впровадження комп'ютерна система, яка визначає положення людини в обмеженому просторі на основі відеопотоку з камери, та працює у реальному часі на мікрокомп'ютері NVIDIA Jetson Nano.

5. Досліджені результати навчання та точність навчених моделей нейронних мереж: модель, яка навчалася на датасеті Open Images, показала більшу точність при навчанні, однак у реальному житті обидві моделі працюють приблизно на одному рівні, маючи окремі недоліки.

6. Досліджені результати роботи розробленої комп'ютерної системи: система працює зі швидкістю 16 FPS на мікрокомп'ютері NVIDIA Jetson Nano з середньою похибкою у 26 см, що є цілком прийнятним для більшості сфер її застосування. Точність розробленої системи залежить від якості навчання моделі нейронної мережі, яка у свою чергу залежить від якості датасета.

7. Поставлена проблема визначення положення людини в обмеженому просторі у реальному часі повністю вирішена з допустимою точністю.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Антошин К. С., магістрант, Безверхий А. І., доцент — науковий керівник. Комп’ютерна система розпізнавання об’єктів у відеопотоці. *Молода наука-2020* : зб. наук. праць студентів, аспірантів і молодих вчених. Запоріжжя : ЗНУ, 2020. Т. 5. С. 73–75.

2. Антошин К. С., магістрант, Безверхий А. І., доцент — науковий керівник. Комп’ютерна система для визначення положення людини в обмеженому просторі у реальному часі. Матеріали XXV науково-технічної конференції студентів, магістрантів, аспірантів, молодих вчених та викладачів. Запоріжжя : ЗНУ, 2020. С. 156.

3. Mautz R. Indoor positioning technologies. *ETH Zurich Research Collection*. 2012. URL: https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/54888/eth-5659-01.pdf (дата звернення: 03.09.2020).

4. Cosma A., Radoi I., Radu V. CamLoc: Pedestrian Location Detection from Pose Estimation on Resource-constrained Smart-cameras. 2018. URL: https://arxiv.org/pdf/1812.11209.pdf (дата звернення: 05.09.2020).

5. Mautz R., Tilch S. Survey of optical indoor positioning systems. *International Conference on Indoor Positioning and Indoor Navigation* : conference materials, Guimaraes, 21-23 Sept. 2011. Guimaraes, 2011.

6. Saputra M., Widyawan W., Putra G., Santosa P. Indoor human tracking application using multiple depth-cameras. *International Conference on Advanced Computer Science and Information Systems* : conference materials, Depok, 1-2 Dec. 2012. Depok, 2012. P. 307–312.

7. Domingo J., Cerrada C., Valero E., Cerrada J. An Improved Indoor Positioning System Using RGB-D Cameras and Wireless Networks for Use in Complex Environments. *Sensors*. 2017. Vol. 17, No 10. URL: https://www.mdpi.com/1424-8220/17/10/2391/pdf (дата звернення: 10.09.2020).

8. Mahony N., Campbell S., Carvalho A. Deep Learning vs. Traditional Computer Vision. *Computer Vision Conference* : conference materials, Las Vegas, 25-26 Apr. 2019. Las Vegas, 2019. Vol. 1. P. 128–144.

9. Dalal N., Triggs B. Histograms of oriented gradients for human detection. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* : conference materials, San Diego, 20-25 June 2005. San Diego, 2005. P. 886–893.

10. LeCun Y., Bengio Y., Haffner P. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*. 1998. Vol. 86, No 11. P. 2278–2324.

11. Krizhevsky A., Sutskever I., Hinton G. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*. 2012. Vol. 25, No 2. P. 1097–1105.

12. Girshick R., Donahue J., Darrell T., Malik J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. *IEEE Conference on Computer Vision and Pattern Recognition* : conference materials, Columbus, 23-28 June 2014. Columbus, 2014. P. 580–587.

13. Girshick R. Fast R-CNN. *IEEE International Conference on Computer Vision* : conference materials, Santiago, 7-13 Dec. 2015. Santiago, 2015. P. 1440–1448.

14. Shaoqing R., He K., Girshick R., Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2016. Vol. 39, No 6. P. 1137–1149.

15. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. *IEEE Conference on Computer Vision and Pattern Recognition* : conference materials, Las Vegas, 27-30 June 2016. Las Vegas, 2016. P. 779–788.

16. Bochkovskiy A., Wang C., Liao H. YOLOv4: Optimal Speed and Accuracy of Object Detection. 2020. URL: https://arxiv.org/pdf/2004.10934.pdf (дата звернення: 25.09.2020).

17. Convolutional Neural Networks from the ground up. URL: https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1 (дата звернення: 18.11.2020).

18. YOLO review from Deep Systems. URL: https://docs.google.com/presentation/d/1aeRvtKG21KHdD5lg6Hgyhx5rPq_ZOsGjG5rJ1HP7BbA/ (дата звернення: 19.11.2020).

19. Szegedy C., Liu W., Jia Y. Going deeper with convolutions. *IEEE Conference on Computer Vision and Pattern Recognition* : conference materials, Boston, 7-12 June 2015. Boston, 2015. P. 1–9.

20. Lin T., Maire M., Belongie S. Microsoft COCO: Common Objects in Context. *European Conference on Computer Vision* : conference materials, Zurich, 6-12 Sept. 2014. Zurich, 2014. P. 740–755.

21. Yolo v4, v3 and v2 for Windows and Linux. URL: https://github.com/AlexeyAB/darknet (дата звернення: 20.11.2020).

22. Howard A., Zhu M., Kalenichenko D. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017. URL: https://arxiv.org/pdf/1704.04861.pdf (дата звернення: 20.11.2020).

23. Liu W., Anguelov D., Erhan D. SSD: Single Shot MultiBox Detector. *European Conference on Computer Vision* : conference materials, Amsterdam, 11-14 Oct. 2016. Amsterdam, 2016. P. 21–37.

24. Howard A., Sandler M., Chu G. Searching for MobileNetV3. *IEEE/CVF International Conference on Computer Vision* : conference materials, Seoul, 27 Oct.-2 Nov. 2019. Seoul, 2019. P. 1314–1324.

25. MobileNet: меньше, быстрее, точнее. URL: https://habr.com/ru/post/352804/ (дата звернення: 20.11.2020).

26. Depthwise Separable Convolution. URL: https://www.youtube.com/watch?v=T7o3xvJLuHk (дата звернення: 20.11.2020).

27. Iandola F., Han S., Moskewicz M. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. 2016. URL: https://arxiv.org/pdf/1602.07360.pdf (дата звернення: 21.11.2020).

28. Нейросети: куда это все движется. URL: https://habr.com/ru/post/482794/ (дата звернення: 21.11.2020).

29. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *International Conference on Learning Representations* : conference materials, San Diego, 7-9 May 2015. San Diego, 2015.

30. Fang W., Wang L., Ren P. Tinier-YOLO: A Real-time Object Detection Method for Constrained Environments. *IEEE Access*. 2019. Vol. 8. P. 1935–1944.

31. Kuznetsova A., Rom H., Alldrin N. The Open Images Dataset V4: Unified Image Classification, Object Detection, and Visual Relationship Detection at Scale. *International Journal of Computer Vision*. 2020. Vol. 128, No 4. P. 1956–1981.

32. Braun M., Krebs S., Flohr F., Gavrila D. EuroCity Persons: A Novel Benchmark for Person Detection in Traffic Scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2019. Vol. 41, No 8. P. 1844–1861.

33. Yu F., Chen H., Wang X. BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning. *IEEE/CVF Conference on Computer Vision and Pattern Recognition* : conference materials, Seattle, 13-19 June 2020. Seattle, 2020. P. 2633–2642.

34. Zhang S., Xie Y., Wan J. WiderPerson: A Diverse Dataset for Dense Pedestrian Detection in the Wild. *IEEE Transactions on Multimedia*. 2019. Vol. 22, No 2. P. 380–393.

35. Shao S., Zhao Z., Li B. CrowdHuman: A Benchmark for Detecting Human in a Crowd. 2018. URL: https://arxiv.org/pdf/1805.00123 (дата звернення: 23.11.2020).

36. Cao Z., Hidalgo G., Simon T. OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2019. Vol. 43, No 1. P. 172–186.

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Антошин Кирило Сергійович, студент _2_ курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти sp115-02@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Комп'ютерна система для визначення положення людини в обмеженому просторі у реальному часі**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-систем, а також на архівування моєї роботи в базі даних цієї системи.

Дата 30.11.2020 Підпис _____ Антошин Кирило Сергійович
(студент)

Дата 30.11.2020 Підпис _____ Безверхий Анатолій Ігорович
(науковий керівник)