МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему:

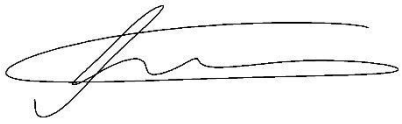
«Інтелектуальна система розпізнавання жестів долоні для взаємодії з інтерфейсом»

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп'ютерні науки»**

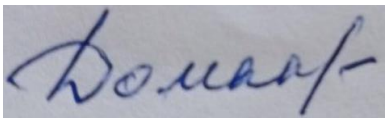
Освітня програма **«Аналітика даних»**

Освітній рівень: бакалавр



Виконав: студент 4 курсу, групи АнД-41

_____Шатохін А. В._____
(прізвище та ініціали)



Керівник _____Доманецька І. М_____
(прізвище та ініціали)

___к. т. н., доцент_____
(науковий ступінь, звання)

Випускна кваліфікаційна робота бакалавра допущена до захисту рішенням кафедри *інтелектуальних технологій*
Протокол № 11 від 06.06.2022 р.
зав. кафедри_____ доц. Іларіонов О.Є.

Київ – 2022

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра інтелектуальних технологій

Спеціальність 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
інтелектуальних технологій
Іларіонов О.Є.

_____

"___"_____2022 р.

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Шатохіну Андрію Владиславовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи)

«Інтелектуальна система розпізнавання жестів долоні для взаємодії з інтерфейсом» затверджена протоколом засідання кафедри від затверджена протоколом засідання кафедри від «23» грудня 2021 р. № 4

2. Термін здачі студентом закінченого проекту (роботи) 29 травня 2022 року

3. Вихідні дані до проекту (роботи)

Проекція розпізнаних жестів долоні на графічний інтерфейс користувача.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

Аналіз особливостей задачі розпізнавання образів у контексті розпізнавання жестів на відео, проектування програмного забезпечення, розробка застосунку.

5. Перелік презентаційного матеріалу (з точним зазначенням обов'язкових презентацій)

Об'єкт, предмет дослідження та мета розробки, актуальність розробки системи та вимоги (4 слайди), діаграми проектування (3 слайди), опис набору даних (1 слайд), опис графічного інтерфейсу (1 слайд), опис розробки програмного продукту (1 слайд), архітектура нейронної мережі (1 слайд), навчання нейронної мережі (3 слайди), проекція жестів на GUI (1 слайд), програмна реалізація (2 слайди)

6. Консультанти з випускної кваліфікаційної роботи із зазначенням розділів випускної кваліфікаційної роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 15 лютого 2022 року

Керівник _____ / Доманецька І. М. /
(підпис) (ПІБ)

Завдання прийняв до виконання _____ / Шатохін А. В. /
(підпис) (ПІБ)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів випускної кваліфікаційної роботи	Примітка
1	Опрацювання літератури	15.02.2022 – 01.03.2022	
2	Робота над розділом 1. Аналіз особливостей задачі розпізнавання жестів долоні для взаємодії з інтерфейсом	01.03.2022 – 20.03.2022	
3	Робота над розділом 2. Проектування архітектури системи розпізнавання жестів долоні для взаємодії з інтерфейсом	20.03.2022 – 11.04.2022	
4	Робота над розділом 3. Розробка програмного застосунку	11.04.2022 – 15.05.2022	
5	Робота над оформленням пояснювальної записки	25.05.2022	
6	Робота над презентацією	01.06.2022	

Студент-дипломник _____ / Шатохін А. В. /
(підпис) (ПІБ)

Керівник випускної кваліфікаційної роботи _____ / Доманецька І. М. /
(підпис) (ПІБ)

АНОТАЦІЯ

Шатохін Андрій Владиславович виконав випускну кваліфікаційну роботу на тему «Інтелектуальна система розпізнавання жестів долоні для взаємодії з інтерфейсом» за спеціальністю 122 – «Комп'ютерні науки».

У кваліфікаційній роботі досліджується процес розпізнавання жестів, визначаються основні недоліки наявних рішень. Розроблено програмний модуль системи із використанням графічного інтерфейсу. Розробка виконана на мові програмування Python. Робота має практичну значимість і результати роботи можуть використовуватися в будь-яких пристроях із підтримкою введення за допомогою миші.

Ключові слова: розпізнавання жестів, графічний інтерфейс, згорткові нейронні мережі, глибоке навчання

ANNOTATION

The degree project: «Intelligent palm gesture recognition system for interface interaction» has completed by **Shatokhin Andrii** specialty 122 – «Computer Science».

This graduation thesis examines the process of gesture recognition and identifies the main shortcomings of existing solutions. A software module of the system using a graphical interface has been developed. Development is performed in the Python programming language. The work is of practical importance and the results can be used in any device that supports mouse input.

Keywords: gestures recognition, graphic user interface, convolutional neural networks. deep learning

ЗМІСТ

ВСТУП 7

РОЗДІЛ 1. АНАЛІЗ ОСОБЛИВОСТЕЙ ЗАДАЧІ РОЗПІЗНАВАННЯ ЖЕСТІВ ДОЛОНІ ДЛЯ ВЗАЄМОДІЇ З ІНТЕРФЕЙСОМ 8

1.1 9

1.2 11

1.2.1 11

1.2.2 12

1.2.3 14

1.3 15

1.3.1 15

1.3.2 16

1.3.3 16

1.3.4 16

1.3.5 17

1.4 17

1.4.1 17

1.4.2 18

1.4.3 19

1.5 20

1.6 21

РОЗДІЛ 2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ РОЗПІЗНАВАННЯ ЖЕСТІВ ДОЛОНІ ДЛЯ ВЗАЄМОДІЇ З ІНТЕРФЕЙСОМ 21

2.1 23

2.2 23

2.2.1 24

2.2.2 26

2.2.3 28

2.2.4 30

2.3 31

2.3.1 32

2.3.2 33

2.4 34

2.5 37

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ 36

3.1 Програмне середовище 36

3.1.1 41

3.1.2 42

3.2 45

3.2.1 46

3.2.2 48

3.3 50

3.4 51

ВИСНОВКИ 50

ЛІТЕРАТУРА 51

ДОДАТОК 54

ВСТУП

Розвиток технологій іде в сторону спрощень взаємодії кінцевого користувача із системою. Спочатку це був перехід від консольних інтерфейсів до графічних, що приніс із собою появу комп’ютерної миші в якості пристрою введеня.

Наразі відбувається доповнення вже існуючої взаємодії із графічними інтерфейсами або ж програмами за допомогою розширення класичних пристроїв введення, таких як: клавіатури, пульти керування, миші, до більш модернізованих під конкретні варіанти використання: джойстики, пульти керування для шолому віртуальної реальності, тощо.

Розвиток у сфері пристроїв введення приніс появу бездротових пристроїв, що дають змогу взаємодіяти у більш комфортних умовах. Наступним кроком розвитку є поява пристроїв введення, для яких наявність самої апаратної частини не обов’язкова. Вже зараз для цього вживляють чіпи NFS, що дають користувачеві виконувати базові запрограмовані дії.

Актуальність розробки Інтелектуальної системи розпізнавання жестів долоні для взаємодії з інтерфейсом полягає у доповненні вже існуючих типів взаємодії із інтерфейсом за допомогою оптичних зчитувальних пристроїв, що не вимагають окремого зовнішнього обладнання. Таку систему можна використовувати аби виділити продукт на фоні конкурентів..

РОЗДІЛ 1. АНАЛІЗ ОСОБЛИВОСТЕЙ ЗАДАЧІ РОЗПІЗНАВАННЯ ЖЕСТІВ ДОЛОНІ ДЛЯ ВЗАЄМОДІЇ З ІНТЕРФЕЙСОМ

1.1 Людино-комп'ютерна взаємодія (англ. Human-computer interaction, HCI)

Людино-комп’ютерна взаємодія (HCI) – це дисципліна, що займається дослідженнями в галузі розробки, проектування та використання комп’ютерних технологій і, зокрема, взаємодії між людьми (користувачами) та комп’ютерами.

Загалом, як сфера досліджень, HCI знаходиться на стиці багатьох дисциплін, серед яких можна виділити: комп’ютерні науки, поведінкові науки, медіазнавство та дизайн, а список другорядних галузей дослідження можна продовжувати. І хоча спочатку HCI стосувався лише взаємодії людей із комп’ютерами, поступово його використання поширилося і на інші форми проектування інформаційних технологій.

Рисунок 1.1 Людино-комп’ютерна взаємодія

До основних цілей напряму людино-комп’ютерної взаємодії відносяться:

1. Створення придатних для використання програмних продуктів і користувацьких інтерфейсів
2. Підвищення зручності використання вже наявних продуктів

3. Визначення та вирішення наявних проблем, пов’язаних із використанням специфічних робочих інструментів, за допомогою спеціалізованих програмних продуктів

Зручність використання є ключовим моментом для HCI, оскільки будь-які високотехнічні аспекти інформатики завжди орієнтовані на цільову аудиторію – своїх користувачів. Безпека та функціональність, ефективність та задоволення користувацьких потреб – ось головні завдання, яка вирішує ця дисципліна.

Розглядають в основному такі типи взаємодій:

1. Візуальна взаємодія: Візуальна взаємодія людини та комп’ютера, ймовірно, є найпоширенішою областю дослідження взаємодії людини та комп’ютера (HCI).
2. Звукова взаємодія: Взаємодія між комп’ютером і людиною на основі аудіо є ще однією важливою областю систем HCI. Ця область стосується інформації, отриманої за допомогою різних звукових сигналів.
3. Машинна взаємодія: Взаємодія між комп’ютером і людиною за допомогою носимих гаджетів, під’єднаних до комп’ютерного середовища.

Загалом будь-яка взаємодія відбувається із візуальним інтерфейсом, на якому зображені різні графічні об’єкти (значки, діаграми, форми тощо), якими користувач може безпосередньо маніпулювати на екрані за допомогою курсора, або будь-яким із вище перелічених методів взаємодії. Основні приклади взаємодії:

1. Переміщення вказівника миші
2. Наведення
3. Виділення
4. Відкриття або запуск програми
5. Перетягування

Задача інтелектуальної системи розпізнавання жестів долоні для взаємодії з інтерфейсом полягає у заміні пристрою введення, а саме – комп’ютерної миші, на більш інформативні широким масам жести долоні, тим самим вивівши взаємодію людей із комп’ютерами на новий рівень.

1.2 Аналіз існуючих пристроїв та систем розпізнавання жестів

1.2.1 Ігровий контролер Kinect компанії Microsoft

Сенсор Kinect, розроблений компанією Microsoft, використовує інноваційні апаратні та програмні методи для захоплення руху в режимі реального часу з ювелірною точністю. 3D-сенсор складається з модуля RGB камери з роздільною здатністю 1920 × 1080 пікселів, а також інфрачервоної камери та випромінювача. Оцінка відстані між датчиком і точками на сцені, яку спостерігає камера, виконується бортовою електронікою з використанням технології Time of Flight (TOF) та методів модуляції інтенсивності. [3]

Рисунок 1.2 Kinect сенсор

Камера Kinect створює рухоме 3D-зображення об’єктів у своєму полі зору та розпізнає людей серед них. Раніше для розрізнення об’єктів та їх фону використовувалися відмінності в кольорі та текстурі, але Kinect використовує іншу модель, розроблену PrimeSense, за якою камера знімає майже невидиме

ІЧ-світло та вимірює час польоту, коли воно відбивається від об’єкта. Датчик менш чутливий до спотворень від видимого світла, оскільки він призначений для реєстрації лише ІЧ-променів. Вбудований процесор використовує деякі алгоритми для обробки отриманих даних і візуалізації 3D-зображення. [4]

Позиції тіла визначаються за допомогою методів структурованого світла та машинного навчання. Спочатку карта глибини обчислюється шляхом аналізу точок ІЧ-світла. Воно працює на основі структурованого світла, в якому відомий шаблон проектується на сцену, а інформація про глибину виводиться з деформації візерунка. Ліс випадкових рішень використовується для визначення частин тіла, задля цього модель було попередньо навчено на мільйоні схожих прикладів. Для створення скелету зображення, Kinect використовує окремі частини тіла та їх проекції. [5]

1.2.2 Контролер руху Leap Motion

Leap контролер функціонує в інфрачервоному (ІЧ) спектрі з можливістю фіксувати різні показники пальців і долоні для кожного кадру, який обробляється з точністю до міліметра [6]. Використання ІЧ-датчика робить його менш схильним до змін невидимого світла.

Апаратне забезпечення Leap Motion складається з двох камер і трьох ІЧ світлодіодів, здатних відстежувати ІЧ-світло поза видимим спектром з довжиною хвилі 850 нм.

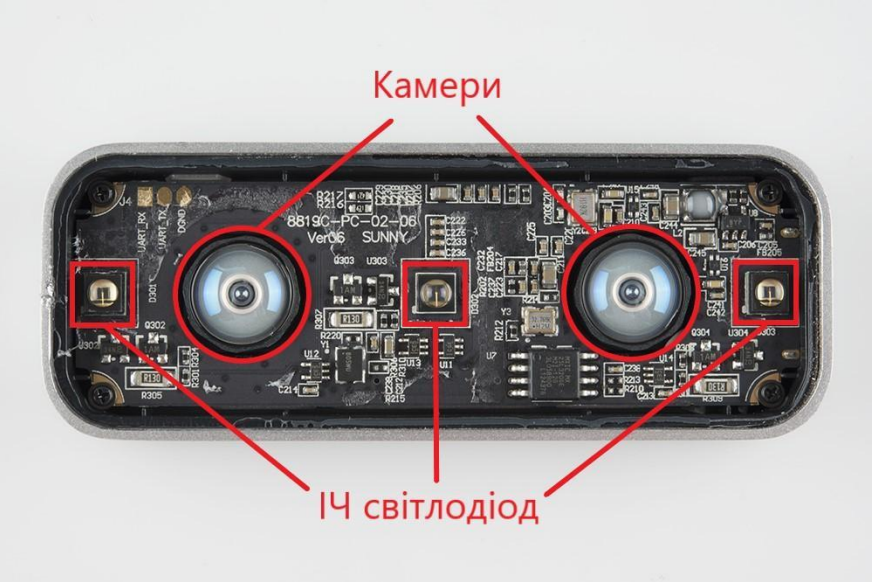


Рисунок 1.3 Leap Motion сенсор

Контролер має зону взаємодії, від 10 см до 60 см у висоту, із полем зору 140 × 120°. Цей діапазон обмежений поширенням світлодіодного світла в просторі, оскільки стає набагато важче визначити положення руки в 3D на відстані. Інтенсивність світлодіодного світла обмежена максимальним струмом, що можна спожити через USB-порт.

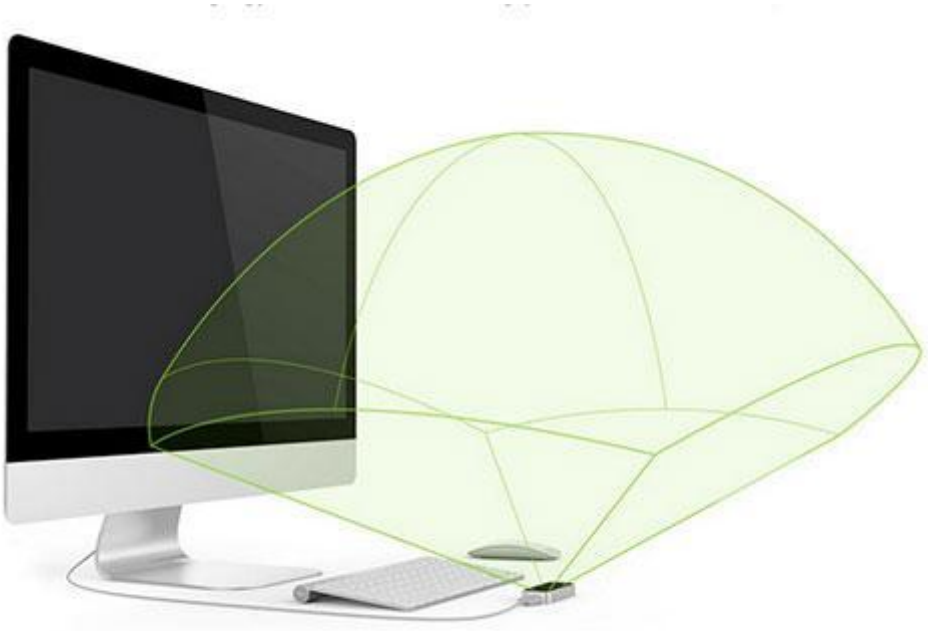


Рисунок 1.4 Leap Motion зона взаємодії

Відстеження рук і пальців у полі зору здійснюється шляхом оновлень кадрів даних. Камери руху фіксують вихідні зображення сенсора, що містять

значення ІЧ яскравості та інші дані калібрування, необхідні для виправлення спотворення об’єктива [7].

Деякі необхідні коригування роздільної здатності виконуються в локальній пам’яті на даних датчика, які в подальшому набувають форми зображення у відтінках сірого, розділеного в лівій і правій камерах. Замість того, щоб генерувати карту глибини, Leap сенсор застосовує деякий розширений алгоритм безпосередньо до необроблених даних для ефективного обчислення. Фонові об’єкти та середовища віднімаються, і відновлюється лише тривимірна структура руки.

1.2.3 Провідна рукавчика BeBop

Провідні рукавички забезпечують новий підхід до взаємодії з комп’ютерами, окрім використання стандартних миші та клавіатури. За рахунок датчиків зчитування, вони надають інформацію про рухи користувача. І хоча попередньо розглянуті методи, засновані на візуальній взаємодії, не зобов’язують до використання фізичних пристроїв, проте вплив таких зовнішніх факторів, як освітлення може призвести до хибного розпізнавання, у той час як рукавички з датчиками мають змогу легко зчитувати та передавати достатньо точні дані про будь-який рух.

Рисунок 1.5 BeBop рукавичка

Рукавичка створена з використанням гнучких датчиків, у поєднанні з мікроконтролерами та приймачем. Гнучкі резистори використовуються для виявлення руху, за рахунок визначення вигину пальців. Ці пускові резистори розміщені на пальцях рукавичок. Гнучкі смугові резистори забезпечують можливість віртуального натискання користувачем. Опір гнучкого стрічкового резистора продовжує збільшуватися від 10 кОм у невигнутому стані до 160 кОм при куті вигину 120 градусів. [2]

Для обробки цих сигналів використовується мікроконтролер, який оброблює дані та передає їх далі на блок приймача. Отримані значення використовуються для виконання певних операцій або дій, а сам блок може бути підключений напряму, або ж із використанням бездротових технологій.

1.3 Аналіз існуючих підходів до вирішення задачі розпізнавання жестів долоні

1.3.1 Виявляння контурів

Це техніка пошуку меж об’єкта. Він виявляє розрив у яскравості. Мета цієї технології — знайти найвищий градієнт на зображенні. До градієнтів застосовується порогове значення, щоб знайти найвище значення градієнта. За допомогою правильного порогового значення всі градієнти малої величини

будуть видалені. Підсумування похідних напрямків x і y розглядається як величина градієнтів. [8] Проблемою алгоритма є розпізнавання лише простих жестів, та залежність від заднього фону і освітлення.

1.3.2 Попіксельне порівняння

У цьому методі зображення порівнюються піксель за пікселем. Реалізація цієї техніки проста, але точність цього методу дуже низька в порівнянні з іншими методами. Для порівняння зображень використовується образ зображення. Образ зберігається в базі даних, який звідти виймається та порівнюється з вхідним зображенням. Для сегментації використовується порогове значення. Проблемою алгоритма є низька здатність до узагальнення, оскільки в реальному житті досить складно знайти однакові обставини для такого порівняння.

1.3.3 Гістограма направлених градієнтів

Отримане зображення спершу перетворюється у відтінки сірого, для легкості обчислення значень дескриптора гістограми орієнтованих градієнтів, оскільки доводиться працювати лише з одним каналом. Для обчислення цього орієнтованого градієнта, зображення розбивається на невеликі блоки, де всередині кожного з них обчислюється градієнт вектор для кожного пікселя. Враховуючи величину і напрям градієнта будується гістограма, а для усунення залежності між отриманими значенням градієнта та освітленням зображення, значення нормалізуються. В результаті всі отримані гістограми об’єднуються, утворюючи одновимірний вектор, відомий як дескриптор ознак HOG, що описує зображення. Кінцевим етапом розпізнавання об'єктів є класифікація дескрипторів за допомогою методів та систем для навчання з учителем.

1.3.4 Порівняння із шаблоном

Для ідентифікації та класифікації вхідного зображення використовується набір попередньо збережених даних. Зіставлення із шаблоном – це метод перевірки, що дає змогу класифікувати дані із порівнянням попередньо записаного для конкретної задачі набору. Сам метод складається із двох основних частин:

1. Запис, створення та збереження шаблонів відповідних картинок для кожного із окремих жестів

2. Знаходження шаблону, що найбільше відповідає поточному вхідному набору даних, із використанням різних метрик близькості

1.3.5 Штучні нейронні мережі

Штучні нейронні мережі (ШНМ) є математичною моделлю, що побудована на принципах організації і функціонування біологічних нейронних мереж. Первинною метою підходу ШНМ було розв'язання задач таким же способом, як це робив би людський мозок. З часом увага зосередилася на відповідності певним розумовим здібностям, ведучи до відхилень від біології. ШНМ використовували в ряді різноманітних задач, включно з комп'ютерним баченням, розпізнаванням мовлення, машинним перекладом, соціально-мережевим фільтруванням, грою в настільні та відеоігри, та медичним діагностуванням. [9] На вхід ШНМ передаються математичні характеристики жесту і на виході виходить номер розпізнаного жесту. ШНМ відрізняються між собою структурою, різноманітними моделями та методами навчання. Найбільш поширеними видами ШНМ, що використовуються в задачах розпізнавання, є мережі прямого поширення (англ. Feed forward neural networks), рекурентні нейронні мережі, карти Кохонена і, згорткові нейронні мережі.

1.4 Аналіз існуючих наборів даних для задачі розпізнавання жестів долоні

1.4.1 Набір відео-даних Jester

Набір даних включає 148 092 відеозаписів із 25 унікальними жестами, в тому числі із окремим класом за умови відсутності жесту, та є найбільшим набором даних жестів із використанням відеозаписів на сьогодні [10]. Короткі відеозаписи по 3 секунди кожен розбиваються на кадри зі швидкістю 12 кадрів на секунду та мають розміри 100 × 100 пікселів. У процесі збору даних було задіяно 1 376 людей, середня кількість відео, які записала кожна людина, становить 43. Таким чином, завдяки наявності відмінностей у фоні та зовнішньому вигляді акторів, було отримано найрізноманітніший набір даних, що надав змогу спільноті будувати наскрізні системи розпізнавання жестів у реальному часі.

Набір даних є актуальним для вирішення задачі даного дипломного проекту, проте є дуже об’ємним, що обмежує можливості та підвищує складність для навчання моделей.

1.4.2 Набір відео-даних EgoGesture

Набір даних складається з 2 081 RGB-D відео, на яких було записано 24 161 жестів на 2 953 224 кадрах. Усі дані були записані 50 окремими особами для взаємодії з носимими пристроями. Набір даних включає 83 класи статичних та динамічних жестів, що були записані у 6 різноманітних сценах, як у приміщенні, так і на вулиці. Були розглянуті такі сценарії:

1. статична сцена у приміщенні із засміченим фоном
2. статична сцена у приміщенні із динамічним фоном
3. статична сцена у приміщенні із різкими змінами освітлення навпроти вікна
4. динамічна сцена (ходьба) у приміщенні із динамічним фоном
5. статична сцена на вулиці із динамічним фоном
6. динамічна сцена (ходьба) на вулиці із динамічним фоном

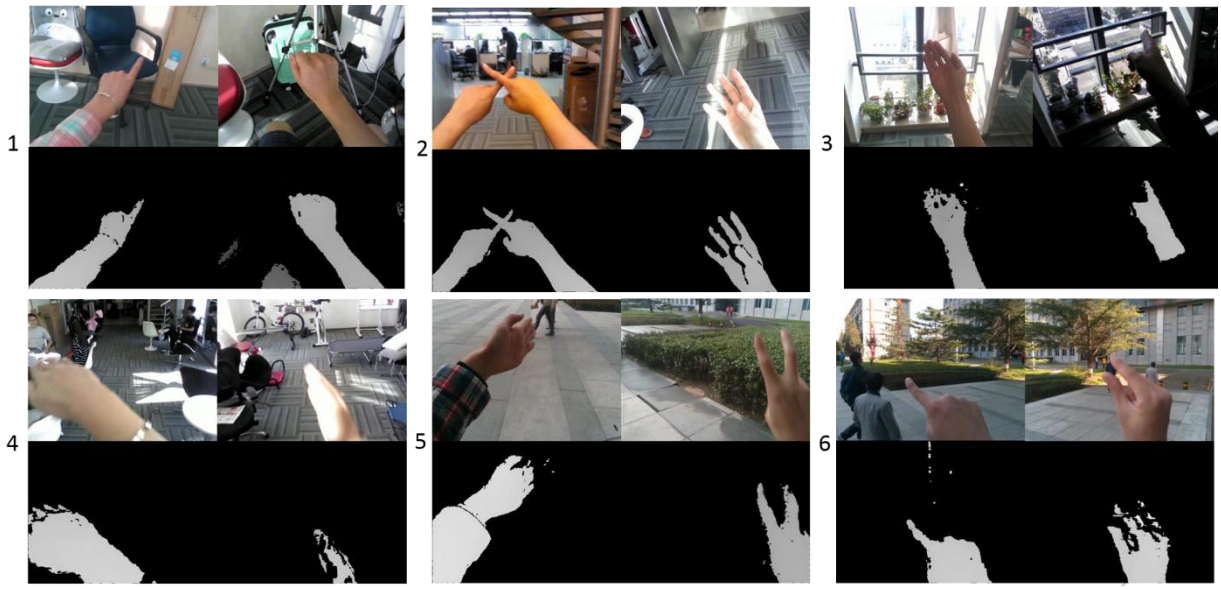


Рисунок 1.6 EgoGesture приклади жестів у кожній із 6 сцен

Для завдання даного дипломного проекту набір не підходить, оскільки використовується камера глибини, а також взаємодія із об’єктами відбувається від першої особи.

1.4.3 Набір відео-даних IPN Hand

П'ятдесят різних суб'єктів брали участь у виконанні 13 класів жестів, із додатковим класом за відсутності жесту, у 28 різноманітних сценах. Для збору набору даних RGB-відео записували в роздільній здатності 640 × 480 з частотою 30 кадрів в секунду. Учасникам було запропоновано записати жести за допомогою власного ПК або ноутбука, зберігаючи визначену роздільну здатність і частоту кадрів. Таким чином, відстань між камерою та кожним об’єктом змінюється, оскільки всі учасники мали розташо в зручному положенні, щоб маніпулювати екраном свого ПК за допомогою жестів рук. Всього в RGB було зібрано 4 218 екземплярів жестів і 800 491 кадр. Під час збору даних 21 жест розглядається як сеанс і записується в одне відео. У наборі даних мінімальна довжина жесту становить 9 кадрів. Максимальна довжина жесту становить 650 кадрів. [11]

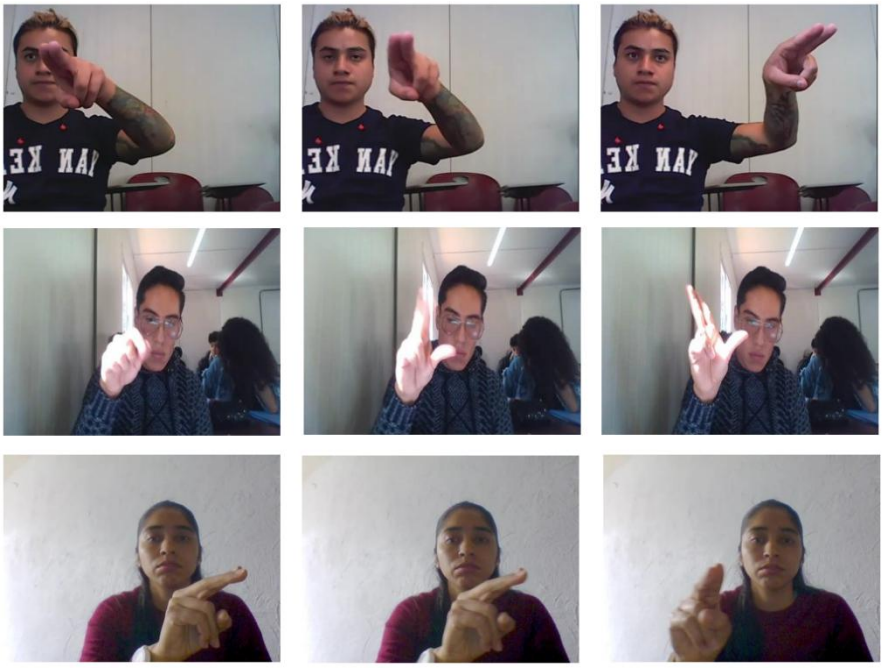


Рисунок 1.7 IPN Hand приклади жестів

Набір даних ідеально підходить для завдання даного дипломного проекту, оскільки використовується лише RGB канал, без камери глибини, взаємодія відбувається від другої особи, сам набір даних є досить різноманітним, а обсяг даних є помірним.

1.5 Аналіз сучасних досліджень у вирішенні задачі розпізнавання жестів долоні

В роботі [11] для виявлення жесту використовувалась ResLight-10 із 8 вхідними кадрами та отриманою точністю 75.4%, а для класифікації 25 жестів було використано ResNeXt-101 із 32 вхідними кадрами для отримання точності 83.79%.

Робота [12] із використанням 3D-CNN із використанням LSTM на 18 вхідних кадрів показала точність розпізнавання 10 жестів на рівні 87%.

У [10] роботі було для одного набору даних було порівняно наступні архітектури, що показали такі результати:

1. CNN + LSTM – 86.3%
2. 3D-CNN + ResNet18 – 81.55%

3. ResNeXt-101 – 85.98%
4. 3D-CNN – 85.49%

Використання HOG + SVM + CNN для швидкого виявлення та класифікації дало змогу у роботі [13] досягти точності 90% для 4 жестів.

Як видно з досліджень, застосування нейромережевих технологій до задачі розпізнавання жестів є актуальним та найбільш перспективним напрямом у вирішенні задачі. Загалом використовують різні варіації згорткових нейронних мережі, а також їх комбінації із рекурентними нейронними мережами.

1.6 Постановка задачі

Об’єктом дослідження роботи є процес розпізнавання жестів на відео.

Предметом дослідження роботи є нейромережеві архітектури для розпізнавання жестів на відео.

Суть задачі полягає в створенні програмного застосунку для взаємодії людей із інтерфейсом комп’ютера, що базується на використанні глибоких нейронних мереж для розпізнавання жестів на відео в режимі реального часу які дублюють основні жести комп’ютерної миші.

Метою роботи є розробка автоматизованої системи розпізнавання жестів долоні для покращення взаємодії користувачів із інтерфейсом комп’ютера.

Функціональні вимоги:

1. Жести користувачів зчитуються за допомогою RGB відеокамери
2. Взаємодія із системою виконується лише одною долонею
3. Курсор на екрані відповідає позиції долоні на отриманому відеокамерою зображенні
4. Розпізнаний жест передається на інтерфейс
5. Жести, що розпізнає система, дублюють основні дії комп’ютерної миші, а також пропонують більш розширений функціонал

Нефункціональні вимоги:

1. Точність розпізнавання для 14 жестів повинна бути більше 50%
2. Жести повинні бути класифіковані в режимі реального часу із затримкою менше 1 секунди
3. Використання операційної системи Windows
4. Використання GPU для роботи системи в режимі реального часу

РОЗДІЛ 2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ РОЗПІЗНАВАННЯ ЖЕСТІВ ДОЛОНІ ДЛЯ ВЗАЄМОДІЇ З ІНТЕРФЕЙСОМ

2.1 Архітектура інтелектуальної системи

Інтелектуальна система розпізнавання жестів долоні для взаємодії з інтерфейсом передбачає взаємодію програмної та апаратної частин. Програмна частина складається з двох основних об’єктів: програмного інтерфейсу та нейронної мережі, а апаратна частина передбачає роботу із зовнішніми відеозаписуючими пристроями.

Для опису інтелектуальної системи використовуються два типи UML діаграм:

1. поведінкові діаграми, що дозволяють візуалізувати, конкретизувати, конструювати та документувати динамічні аспекти системи [14]
2. структурні діаграми, що зображують статичну структуру елементів системи, а також зв’язок об’єктів один з одним [14]

2.2 Поведінкові діаграми

Поведінкові діаграми показують, як саме система поводиться і взаємодіє з собою та іншими об’єктами (користувачами, іншими системами). Вони показують, як дані переміщуються по системі, як об’єкти спілкуються один з одним, як плин часу впливає на систему або які події змушують систему змінювати внутрішні стани.

До основних поведінкових діаграм відносяться:

1. Діаграма діяльності
2. Діаграма прецедентів
3. Діаграма послідовностей
4. Діаграма станів

2.2.1 Діаграма прецедентів (Use Case Diagram)

Для моделювання системи найважливішим аспектом є розуміння та опис її динамічної поведінки, в яку входить поведінка системи коли вона працює. [15] Для обраної системи, динамічна поведінка має наступний вигляд:

- Користувач може запустити програму аби почати роботу (сесію)
- За замовчуванням користувачу відкривається стандартний пристрій введення (камера), який він може змінити на будь-яку з присутніх наразі у системі
- За замовчуванням користувачу надається зона взаємодії, яку він може налаштувати під особливості свого пристрою введення (камери)
- За умови що обрані параметри відповідають бажаним користувач може увімкнути можливість взаємодії з інтерфейсом
- Коли користувач показав жест, що відповідає зазначеному в переліку, система повинна його розпізнати та спроектувати на відповідний жест із інтерфейсом
- Користувач може закрити програму аби закінчити роботу (сесію)

Цей опис графічно можна представити за допомогою UML діаграми прецедентів. Діаграми прецедентів складаються з акторів, варіантів використання та їх взаємозв'язків. Діаграма використовується для моделювання системи або ж підсистеми програми і наведена нижче. (див. Рисунок 2.1)

Основними цілями такої діаграми є:

1. Збір вимог системи
2. Огляд системи зовні
3. Визначення зовнішніх та внутрішніх факторів, що впливають на систему
4. Відображення взаємодії між вимогами акторів

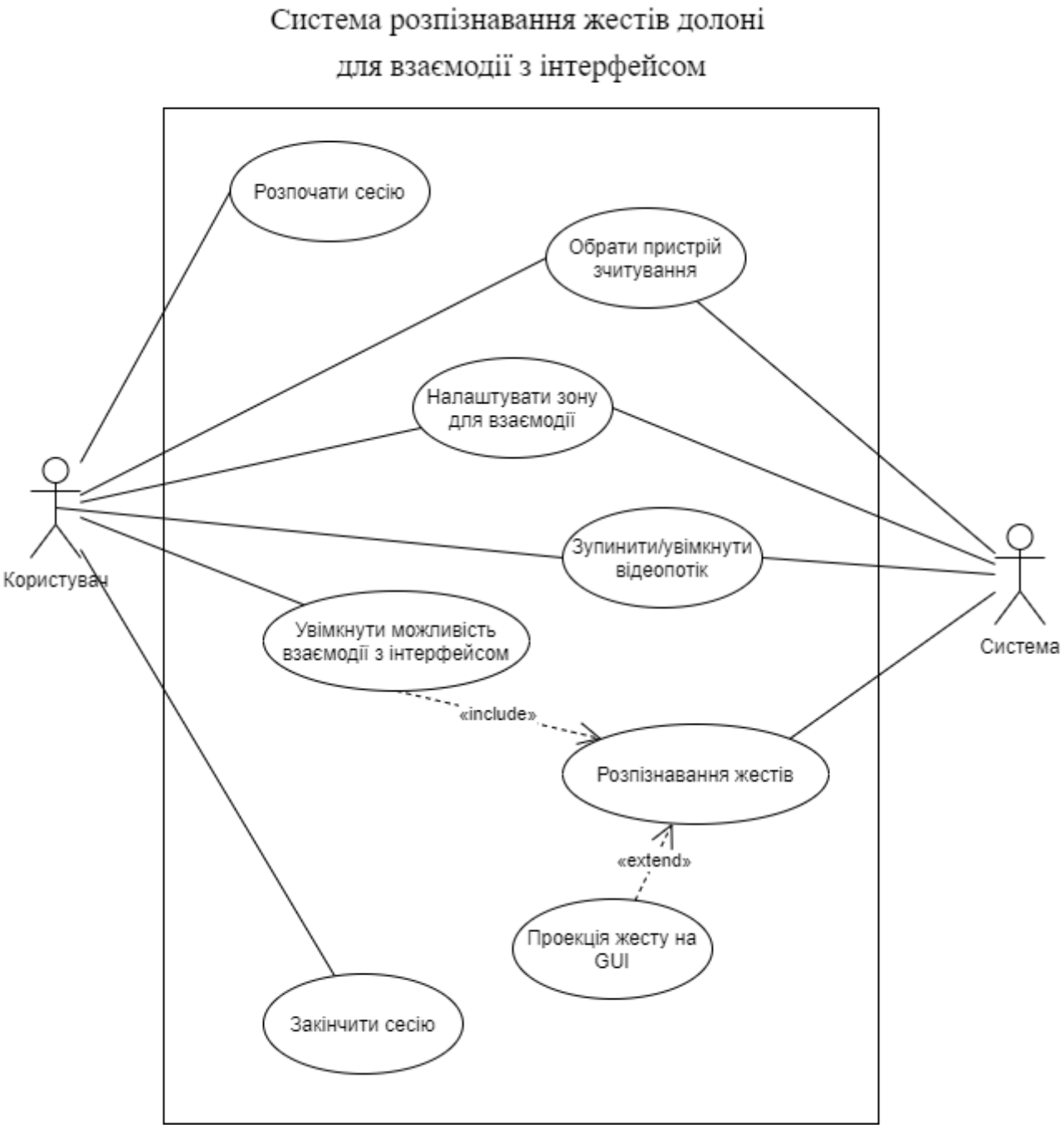


Рисунок 2.1 Діаграма прецедентів

Наведена вище діаграма описує взаємодію між двома акторами: користувачем та системою, і містить 7 випадків використання, що представляють конкретну функціональність системи розпізнавання жестів для взаємодії з інтерфейсом. Користувач-актор може розпочати чи закінчити сесію, обрати пристрій зчитування, налаштувати зону для взаємодії, зупинити або ж увімкнути відеопотік, а також увімкнути можливість взаємодії з інтерфейсом. Цей актор може виконувати лише ці взаємодії з системою, навіть якщо в

системі залишаються інші варіанти використання. Наступний актор – сама система, що встановлює налаштування обрані актором. До її сценаріїв використання відноситься обрати пристрій зчитування, налаштувати зону для взаємодії, зупинити або увімкнути відеопотік, а також розпізнавання жестів.

Ці взаємодії користувача та системи разом узагальнюють всю програму керування жестів долоні для взаємодії з інтерфейсом.

2.2.2 Діаграма діяльності (Activity Diagram)

Діаграма прецедентів як і діаграма діяльності описує динамічну природу системи. Різниця між ними полягає в тому, що діаграма прецедентів допомагає моделювати систему, взаємодію між користувачами та системою, тоді як діаграма діяльності допомагає моделювати саме робочий потік системи. [16]

Додаток може мати кілька систем. Діаграма діяльності також фіксує ці системи та описує перехід від однієї системи до іншої. Це конкретне використання недоступне на інших діаграмах. Прикладами таких систем можуть бути база даних, зовнішні черги або будь-яка інша система.

Діаграми діяльності є дуже високорівневими діаграмами, а отже в основному призначені для бізнес-користувачів або будь-яких інших осіб, які не пов’язані з технічною стороною питання.

Робочий потік системи розпізнавання жестів для взаємодії з інтерфейсом представлений на Рисунку 2.2 і складається з наступних компонентів:

1. Запуск програми
2. Одночасне налаштування системних параметрів та підключення до відеокамери
3. Оновлення існуючих налаштувань, за умови, що їх було змінено
4. За умови, що керування жестами було увімкнено, повторно попередити користувача для отримання згоди

5. Якщо керування жестами було увімкнено але відеозахват у налаштуваннях вимкнено – попередити про вимкнений відеозахват та вимкнути керування жестами
6. Якщо керування жестами було увімкнено і відеозахват у налаштуваннях увімкнено – почати керування жестів, до поки його не вимкнуть

До особливостей реалізації відноситься відсутність явного моменту закінчення процесу, оскільки вихід з GUI програми може бути здійснений у будь який момент часу.

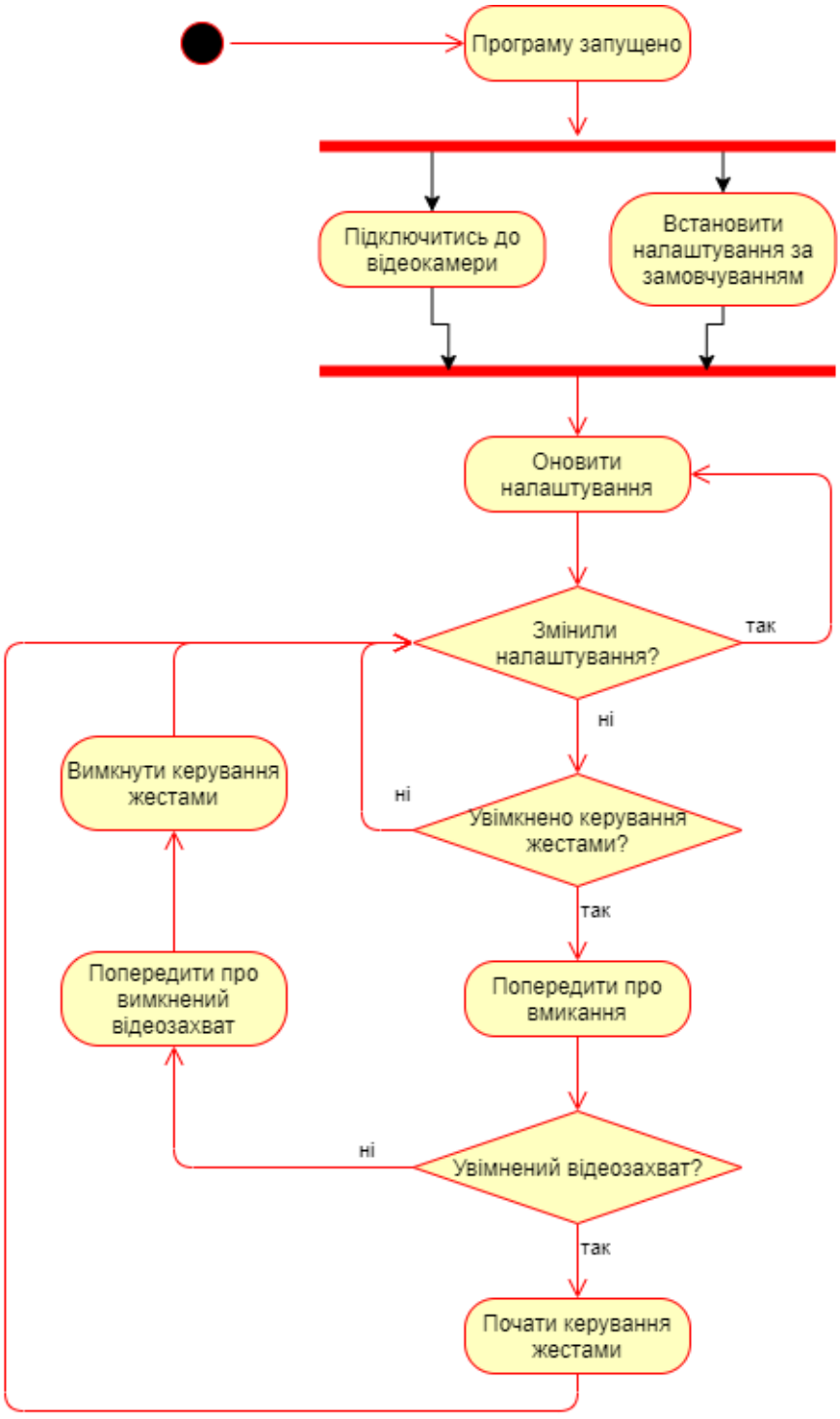


Рисунок 2.2 Діаграма діяльності

2.2.3 Діаграма послідовностей (Sequence Diagram)

Діаграми послідовностей — це діаграми взаємодії, які детально описують операції що виконуються, акцентуючи увагу на тому як саме вони виконуються у контексті співпраці. На цих діаграмах відображується порядок виконання а також проміжок часу, протягом якого об’єкти взаємодіють між собою. Для

цього уся візуалізація виконується уздовж вертикальної вісі діаграми, що відображає час (прогрес).

Переваги використання діаграм послідовностей:

1. Є більш детальним прикладом діаграми прецедентів
2. Моделюють логіку складних процедур, функцій або операцій
3. Дають змогу побачити взаємодію компонентів один з одним для завершення певного процесу
4. Дають змогу зрозуміти існуючу або спланувати майбутню детальну функціональність сценарію використання

Діаграма послідовностей системи розпізнавання жестів долоні для взаємодії з інтерфейсом зображена на Рисунку 2.3 і описує наступну структуру:

1. Спочатку користувач (актор) запускає програму
2. На наступному етапі відбувається послідовне підключення до камери та встановлення налаштувань за замовчуванням. Підключення до камери займає більше часу
3. Наступний крок – постійний цикл, що складається з двох альтернатив. Перша альтернатива – послідовність дій за умови вмикання керування жестами, друга альтернатива – можлива послідовність дій без увімкненого керування жестами
4. За умови вмикання керування жестами, жест від користувача йде на систему, де всередині відбувається його розпізнавання та подальша передача на графічний інтерфейс для його проекції
5. За умови вимкненого керування жестів, є можливість або змінити налаштування системи та увімкнути керування, або ж обрати інший пристрій введення

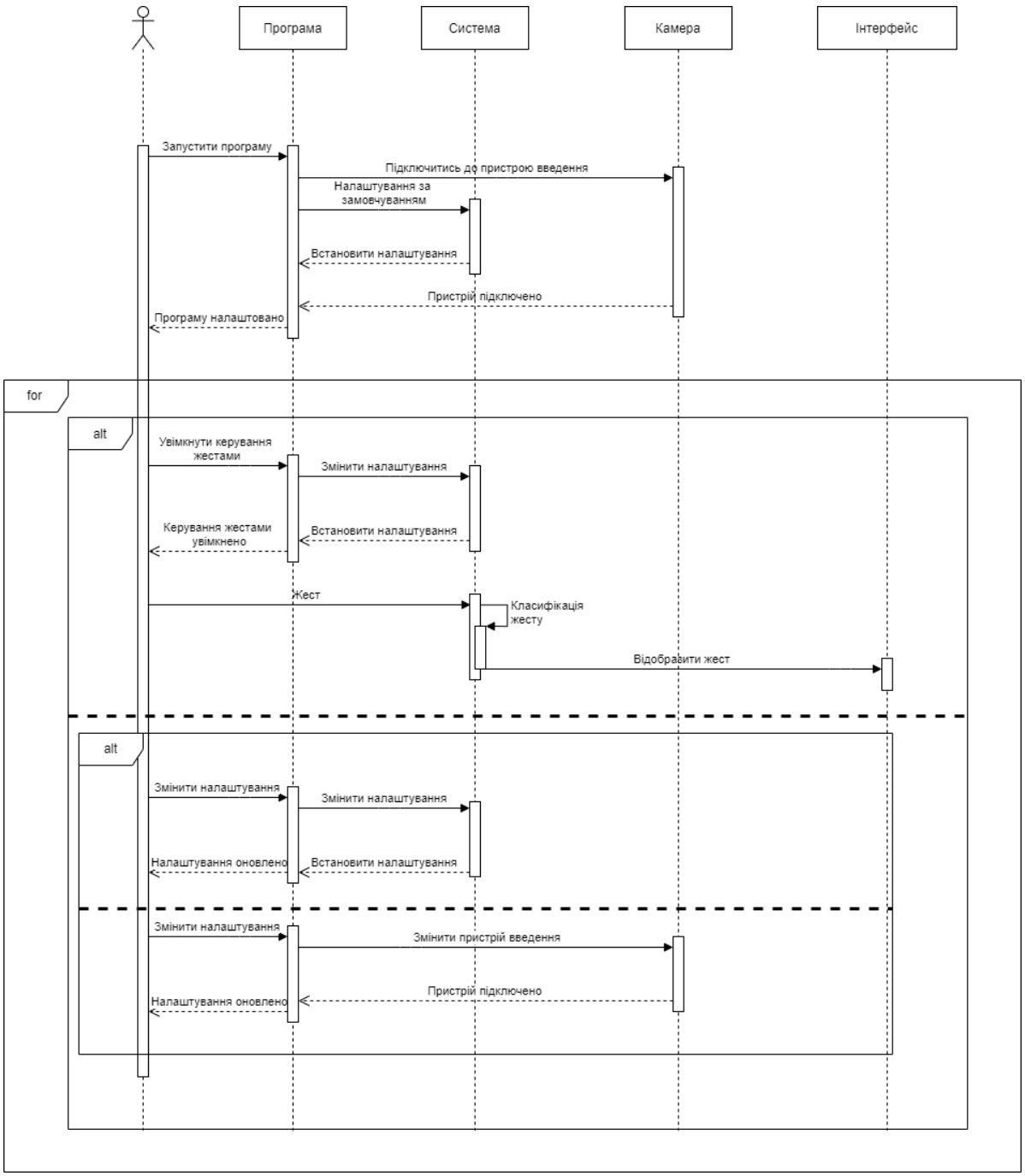


Рисунок 2.3 Діаграма послідовностей

2.2.4 Діаграма станів (State Diagram)

Діаграма станів використовується для представлення стану системи або частини системи в визначені моменти часу. Це поведінкова діаграма, яка відображує поведінку системи з використанням кінцевих переходів станів.

Діаграма станів використовується для моделювання динамічної поведінки класів відповідно до часу їх викликів або у відповідь на зміни зовнішніх

чинників. Передбачається, що класи із трьома або більше станами повинні відображатись на діаграмі.

Діаграми станів використовуються для:

1. Визначення подій, що відповідають за зміну стану
2. Моделювання динамічної поведінки системи
3. Розуміння реакції предметів/класів на внутрішні або зовнішні подразники

Діаграма станів системи розпізнавання жестів долоні для взаємодії з інтерфейсом зображена на Рисунку 2.4 і описує наступні стани системи:

1. Ініціалізація системи із сторонніми підключеннями
2. Налаштування отриманих параметрів
3. Умова перевірки керування жестами
4. Розпізнавання вхідних жестів

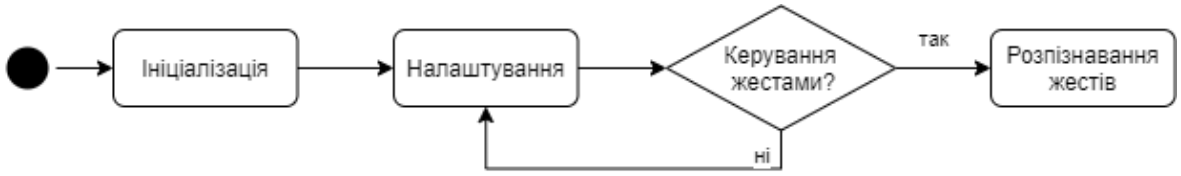


Рисунок 2.4 Діаграма станів

2.3 Структурні діаграми

Структурні діаграми відображають статичну структуру елементів у системі, зв’язок об’єктів один з одним. До таких елементів відносяться: класи, об’єкти, пакети чи модулі, фізичні вузли, компоненти та інтерфейси. Структурні діаграми широко використовуються при документуванні архітектури програмного забезпечення програмних систем.

До основних структурних діаграм відносяться:

1. Діаграма внутрішньої структури
2. Діаграма розгортання
3. Діаграма пакетів

4. Діаграма класів
5. Діаграма об’єктів
6. Діаграма компонентів

2.3.1 Діаграма класів (Class Diagram)

Діаграма класів – це тип діаграми, що описує систему в цілому шляхом візуалізації різних типів об'єктів всередині системи, видів статичних зв’язків, які існують між ними, атрибутів та операцій класу, а також обмежень, накладених на систему.

Зазвичай використовується для вивчення концепцій предметної області, розуміння вимог до програмного забезпечення та опису дизайну проекту. Діаграми класів широко використовуються при моделюванні об'єктно-орієнтованих систем.

Основними цілями такої діаграми є:

1. Аналіз та проектування статичного вигляду програми
2. Опис обов’язків системи
3. Створення підґрунтя для побудови діаграми компонентів та розгортання
4. База для розуміння й побудови прямої та зворотної інженерії

Діаграма класів системи розпізнавання жестів долоні для взаємодії з інтерфейсом зображена на Рисунку 2.5.

Основними класами, що використовуються для побудови системи є:

1. Window – клас, що відповідає за зовнішній вигляд інтерфейсу системи
2. CameraThread – клас, що відповідає за отримання вхідного зображення та передачу його на модель
3. Shape – допоміжний клас для роботи із координатами та зображеннями
4. Resnet3DBuilder – клас, що генерує модель, на основі якої буде проводитись розпізнавання жестів

5. ModelInfo – клас, що відповідає за налаштування моделі для розпізнавання жестів
6. ImagePreprocessing – клас, що проводить попередню обробку вхідного зображення
7. GuiGestures – клас із переліком дій на кожен окремий розпізнаний жест
8. DataInfo – допоміжний клас з інформацію про перелік жестів

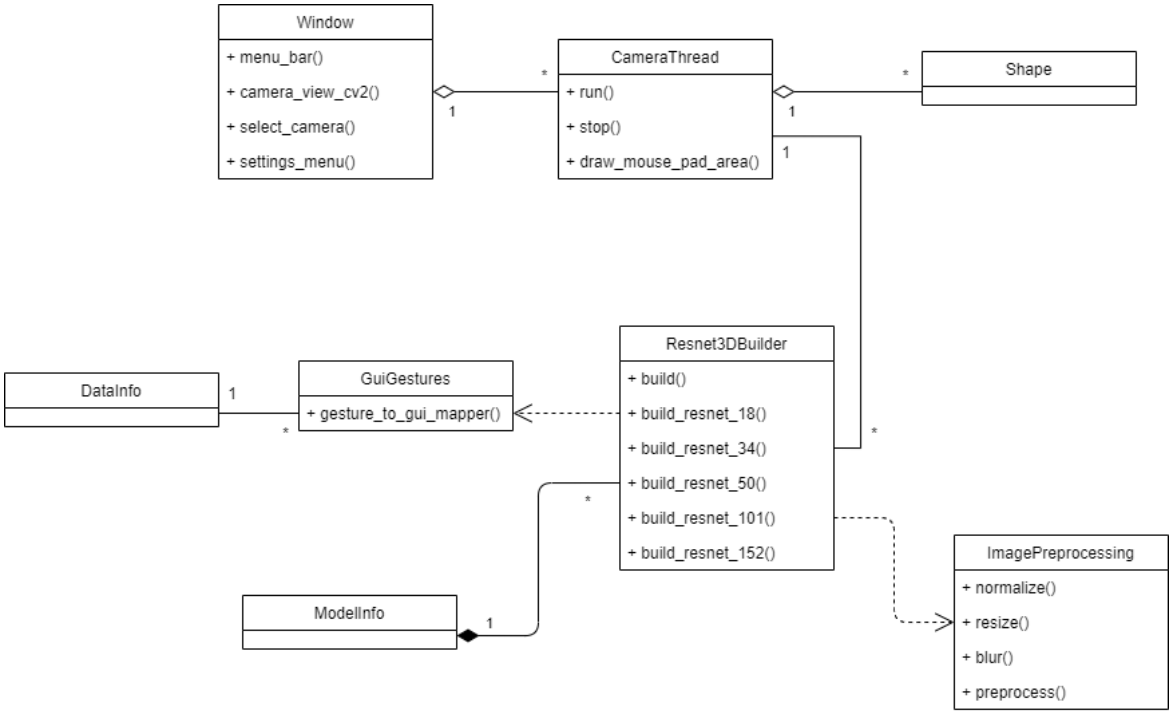


Рисунок 2.5 Діаграма класів

2.3.2 Діаграма розгортання (Deployment Diagram)

Діаграма розгортання відноситься до сімейства структурних діаграм, оскільки вона описує аспект самої системи. У цьому випадку діаграма розгортання описує фізичне розгортання інформації, згенерованої програмою, на апаратних компонентах.

Основні програмні або апаратні системні елементи представлені у вигляді тривимірних блоків, відомих як вузли. На наявність системних зв’язків вказує лінія між двома вузлами, а менші фігури, що містяться в полях, представляють програмні артефакти, які розгортаються.

Діаграми розгортання використовуються для:

1. Зображення залежності розгортання конкретних програмних елементів відповідними апаратними елементами.
2. Ілюстрації потоку обробки та виконання апаратним забезпеченням
3. Надання уявлення про топологію апаратної системи
4. Відображення схеми розгортання системи

Діаграма розгортання системи розпізнавання жестів долоні для взаємодії з інтерфейсом зображена на Рисунку 2.6.

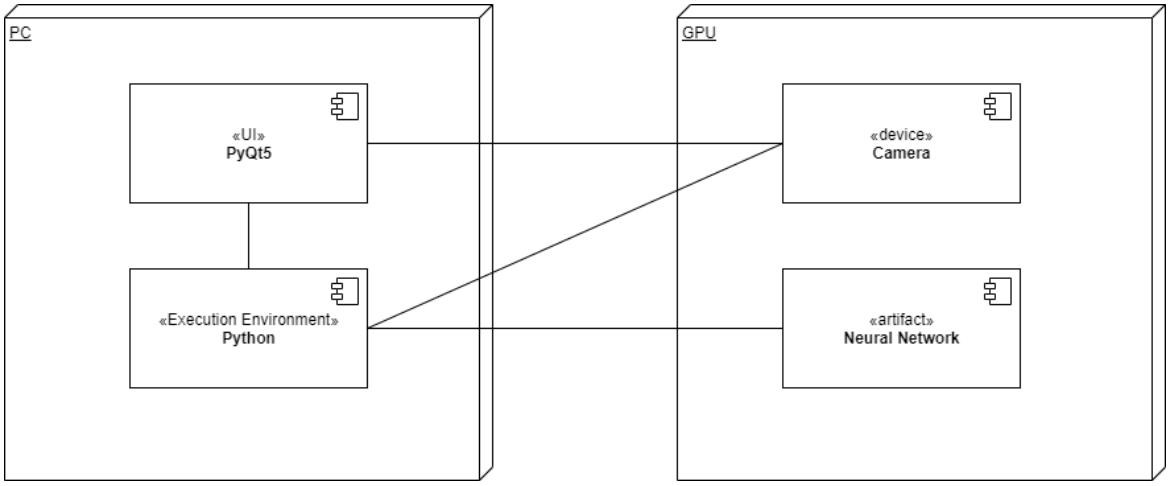


Рисунок 2.6 Діаграма розгортання

Розгортання системи здійснюється на настільному комп'ютері із використанням CPU та GPU. На CPU відбувається запуск середовища виконання Python та побудова графічного інтерфейсу на PyQt5, а на GPU відбувається підключення та трансляція відеопотоку в інтерфейс. Класифікація с використанням нейронної мережі теж відбувається на GPU, отримуючи дані від середовища виконання та повертаючи в зворотному напрямку готовий результат.

2.4 Архітектура нейронної мережі

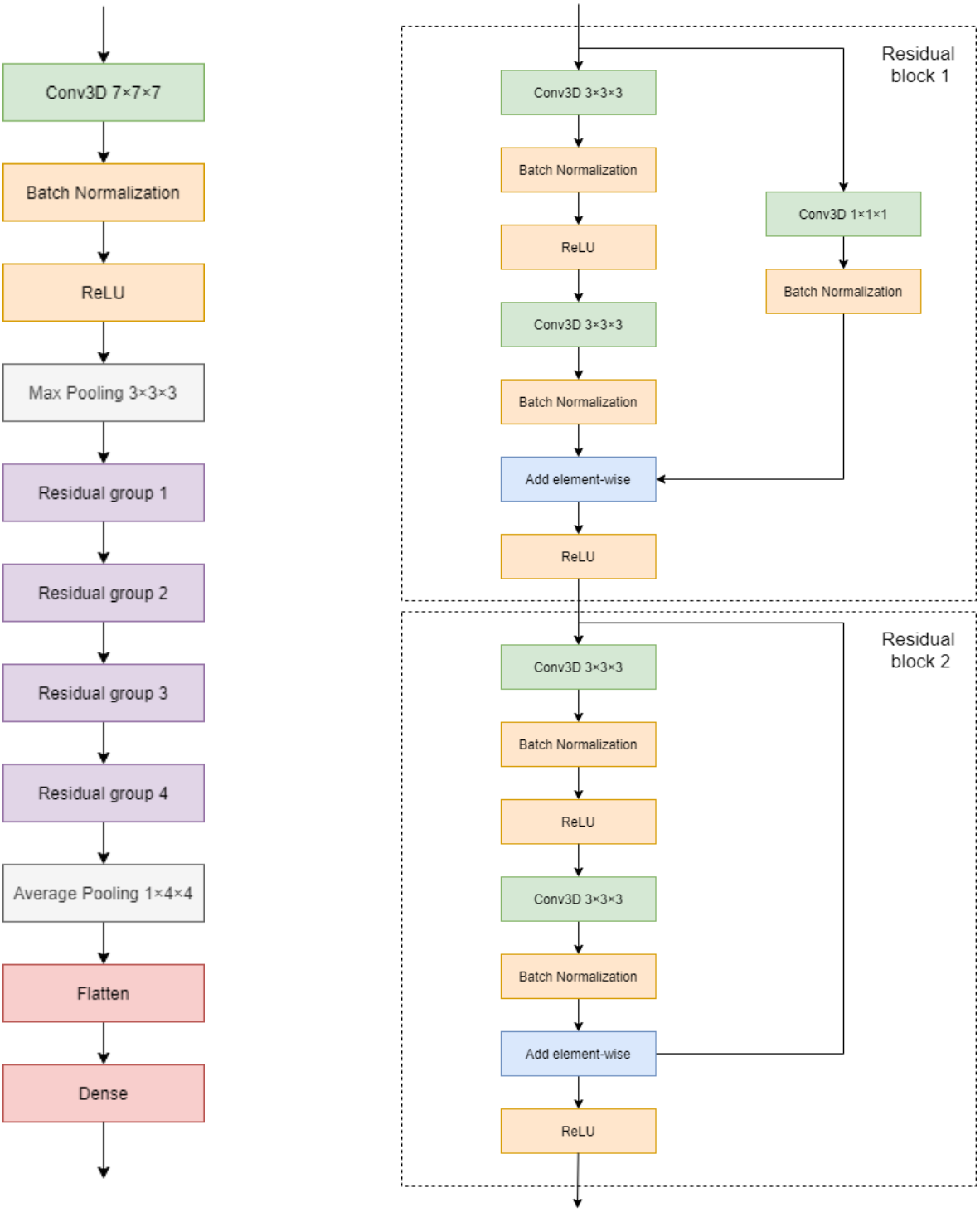
Нейронна мережа – основна частина системи розпізнавання жестів для взаємодії з інтерфейсом. Для класифікації жестів було обрано згорткову нейронну мережу, яка може опрацьовувати вхідний відеопотік із здатністю обробляти просторові зміни. Для вирішення цієї задачі було обрано 3D ResNet.

Мережа ResNet прийшла на зміну VGGNet, і була логічним продовженням розвитку сфери комп'ютерного зору для задач класифікації зображень. Мережі ResNet стали поширеними завдяки підходу до вирішення проблеми зникаючого градієнту. Ця проблема з'являється тоді, коли мережа занадто глибока і градієнти, з яких обчислюється функція втрат, зменшуються до нуля в результаті диференціювання складної функції. У результаті вагові коефіцієнти ніколи не оновлюють свої значення, а отже, навчання не виконується. Архітектура ResNet передбачає використання пропускних з'єднань, що дозволяють інформації пропускати шари, тому під час прямого проходу інформація з рівня l може бути безпосередньо подана в шар l+t (тобто активації рівня l додаються до активацій рівня l+t), для t ≥2, і під час прямого проходу градієнти також можуть перетікати без змін від шару l+t до шару l.

Кожна архітектура ResNet складається з таких етапів: початкова згортка, максимізаційне агрегування, 4 окремих етапи із використанням залишкових блоків та різною кількістю шарів всередині, усереднювальна агрегація і повнозв'язний шар. В залежності від задачі класифікації: відео або фото, розмірність ядра всередині згорткових шарів може відрізнятись. На рисунку 2.7 зображено загальну архітектуру 3D ResNet-34 із початковою згороткою ядром 7×7×7, максимізаційним агрегуванням ядром 3×3×3 та ядром 1×4×4 для усереднювальної агрегації, де розмірність останньої залежить від розмірності вхідних даних.

Структура залишкових блоків зображена на Рисунку 2.8 і складається з двох частин: Residual block 1 та Residual block 2. В залежності від обраної

архітектури ResNet залишкові блоки дублюються відповідну кількість разів, Residual block 1 використовується на початку відповідного залишкового блока, а решта повторень припадає на Residual block 2. Кількість фільтрів при цьому всередині залишкового блоку не змінюється, проте при переході в наступний збільшується удвічі.



Для реалізації моделі розпізнавання жестів було обрано 3D ResNet-34, що складається з 34 глибоких шарів.

Початкова ініціалізація моделі включає:

1. Кількість повторень залишкових блоків: 3, 4, 6, 3
2. Початкова кількість фільтрів: 64
3. Метод ініціалізації фільтрів: Kaiming Initialization, враховує нелінійність функції активації ReLU
4. Метод регуляризації ядра: L2-норма з коефіцієнтом регуляризації 0,0001
5. Розмірність вхідних даних: 30 кадрів, 120 пікселів у висоту, 120 пікселів у ширину, 3 канали світла
6. Розмірність вихідних даних: 14 жестів (разом із відсутністю жестів)
7. Алгоритм навчання: Adam

2.5 Інтерфейсні рішення

Для можливості повноцінної взаємодії користувача із інтерфейсом системи, попередньо було продумано та спроектовано наступний графічний інтерфейс користувача, зображений на Рисунку 2.9., що було візуалізовано за допомогою спеціального програмного забезпечення Pensil. Представлений інтерфейс відображає точний перелік налаштувань, які користувач має змогу змінити, а також їх відносне положення один до одного.

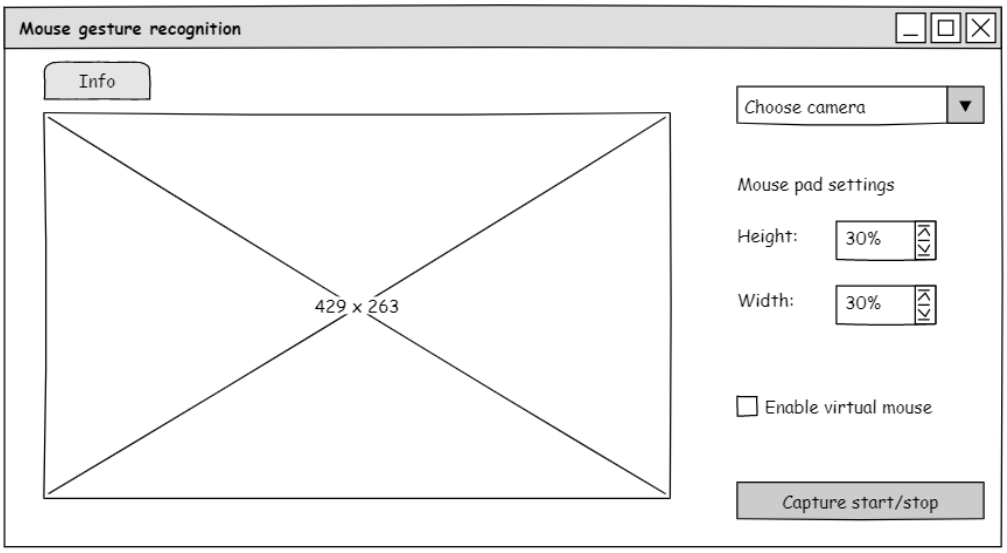


Рисунок 2.9 Основне вікно

Представлений нижче інтерфейс поділений на дві основні секції: секція візуалізації (зліва) та секція налаштувань (справа), і налічує 7 основних компонентів:

1. На панелі інструментів випадаючий список “Info”, який містить в собі елемент “Gestures guide” із зовнішнім посиланням на URL сторінку з прикладами жестів
2. Блок із вхідним зображенням, що зчитується в режимі реального часу із відеопотоку
3. Список що розкривається із можливістю вибору та підключення до різних засобів зчитування (камер)
4. Два лічильники зі шкалою 0-100, що відповідають за розміри віртуального вікна, у якому рука користувача буде рухати вказівник миші на екрані
5. Прапорцева кнопка “Enable virtual mouse” відповідає за підключення або відключення можливостей взаємодії користувача із GUI системи
6. Кнопка “Capture start/stop” зупиняє або відновлює відображення відео в режимі реального часу

Також в інтерфейсі присутні інформаційні та попереджувальні вікна, що попереджають користувача про виконання певних дій, наприклад:

1. За умови відсутності підключення до інтернету, під час спроби відкрити посилання на сторінку із прикладами жестів буде виведено попередження із текстом “Could not open url”
2. Після того як прапорцева кнопка “Enable virtual mouse” буде увімкнена, повинно з’явитися інформаційне табло для підтвердження дій користувача
3. За умови що прапорцева кнопка “Enable virtual mouse” увімкнена, підтвердження дій користувача було зроблено, проте відображення відео в режимі реального часу зупинено, буде виведено попередження із текстом “Video capturing is not started”.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ

3.1 Програмне середовище

Розробка системи передбачає такі основні етапи, як: попередній аналіз, проектування та реалізацію. Кожен із згаданих етапів повинен брати до уваги обмеження, що накладаються сучасними технологіями для виконання будь якої ідеї чи процесу. Саме тому вибір програмного середовища є одним із основних елементів, які потрібно брати до уваги, перш ніж починати розробку.

Для реалізації інтелектуальної системи розпізнавання жестів долоні для взаємодії з інтерфейсом потрібно брати до уваги момент створення зовнішнього інтерфейсу програми, момент розробки нейронної мережі, силами якої буде відбуватись класифікація жестів, що йдуть на вхід до системи, а також момент взаємодії із GUI. Для реалізації кожного з цих етапів було обрано мову програмування Python.

Python – це мова комп’ютерного програмування, яка часто використовується для створення веб-сайтів і програмного забезпечення, автоматизації завдань і аналізу даних. Оскільки ця мова програмування є мовою загального призначення, це означає, що її можна використовувати для створення різноманітних програм і вона не спеціалізується на будь-яких конкретних проблемах.

Розуміння переваг на недоліків мови програмування допомагає створювати надійні програми із правильним підходом та реалізацією відповідної архітектури.

Переваги мови програмування Python:

1. Простота у написанні та читанні коду
2. Кількість сторонніх бібліотек
3. Висока результативність розробника
4. Гнучкість вбудовування з іншими мовами
5. Об’єктно-орієнтована та процедурна
6. Розширюваність мови

7. Широка спільнота користувачів
8. Інтерпретована мова програмування

Недоліки мови програмування Python:

1. Обмеження швидкості
2. Динамічне типізування
3. Високе споживання пам'яті
4. Складність із багатопоточністю
5. Слабкість у мобільних обчисленнях
6. Доступ до бази даних

Мова програмування Python – чудовий вибір як для новачків, так і для професіоналів, що дає змогу розробнику швидко отримати бажаний результат. Ця мова програмування ідеально підходить для реалізації дипломного проекту, тому було обрано Python версії 3.8.

3.1.1 Засоби реалізації нейромережевого модуля

Серед великого вибору бібліотек для машинного навчання, що підтримує мова програмування Python: TensorFlow, PyTorch, CNTK, Apache MXNet, тощо, для створення, налаштування, навчання та візуалізації нейронної мережі було обрано саме бібліотеку TensorFlow.

Створена командою Google Brain, TensorFlow — це бібліотека з відкритим вихідним кодом для чисельних обчислень і великомасштабного машинного навчання. TensorFlow об’єднує безліч моделей і алгоритмів машинного навчання та глибокого навчання (він же нейронних мереж) і робить їх корисними за допомогою загальної метафори. Він використовує Python для забезпечення зручного інтерфейсного API для створення додатків із фреймворком під час виконання цих програм на високопродуктивному C++. [17]

Бібліотека TensorFlow передбачає підтримку роботи саме з графічними відеокартами NVIDIA і для повноцінної роботи з ними потребує відповідних

налаштувань, до яких відносяться встановлення CUDA та cuDNN. CUDA — це платформа паралельних обчислень і модель програмування, створена NVIDIA. Коли CUDA виконує обчислення з GPU, він гарантує, що компілятор спрямовується на частину, яка відповідає ядру GPU. Це робить процесор ефективним для виконання операції. cuDNN — це бібліотека примітивів із прискореним графічним процесором для глибоких нейронних мереж. Він забезпечує підвищення продуктивності під час навчання нейронної мережі за допомогою графічного процесора, де для більшості бібліотек це збільшення продуктивності становить приблизно у 5,6 разів.

3.1.2 Засоби реалізації інтерфейсу застосунку

Графічний користувальницький інтерфейс, або GUI, як його частіше називають, є одним із трьох основних наріжних каменів будь-якої програми, де двома іншими є безпека та продуктивність. І мова Python, зі своєю великою кількістю написаних бібліотек, має що запропонувати розробнику. Серед великого переліку бібліотек: Tkinter, Kivy, wxPython, PySimpleGUI, PySide2, тощо, з їх особливостями та недоліками, для реалізації інтерфейсу застосунку було обрано PyQt5, що є оновленою версією пакету PyQt, побудованого на основі Qt фреймворку.

Qt фреймворк – це крос-платформний фреймворк, що дозволяє створювати одну програму, яка буде запускатись та підтримуватись на різних платформах, як: Windows, Mac, Linux, iOS, Android тощо. За допомогою модульності мови програмування Python у вигляді розширень, та їх комбінацій з PyQt5 розробнику надається набагато більше функцій, ніж просто створення графічного інтерфейсу.

Таким чином в інтерфейс застосунку було вбудовано зовнішній модуль для роботи із відеопотоком засобами сторонньої бібліотеки cv2, більш відомої як OpenCV.

OpenCV (Open Source Computer Vision Library) — бібліотека комп’ютерного зору та машинного навчання з відкритим вихідним кодом. OpenCV було створено для забезпечення загальної інфраструктури для додатків комп’ютерного зору та для прискорення використання машинного сприйняття в комерційних продуктах. OpenCV написаний на C++ і має шаблонний інтерфейс, який безперебійно працює з контейнерами STL, що забезпечує високу швидкість роботи.

Засобами цієї бібліотеки і відбувається будь-яка робота із вхідним зображенням, починаючи від його зчитування та передачі в режимі реального часу, до нанесення додаткових елементів на картинку та подальшої її обробки для подання на вхід моделі.

Із використанням цих бібліотек було отримано наступний інтерфейс застосунку:

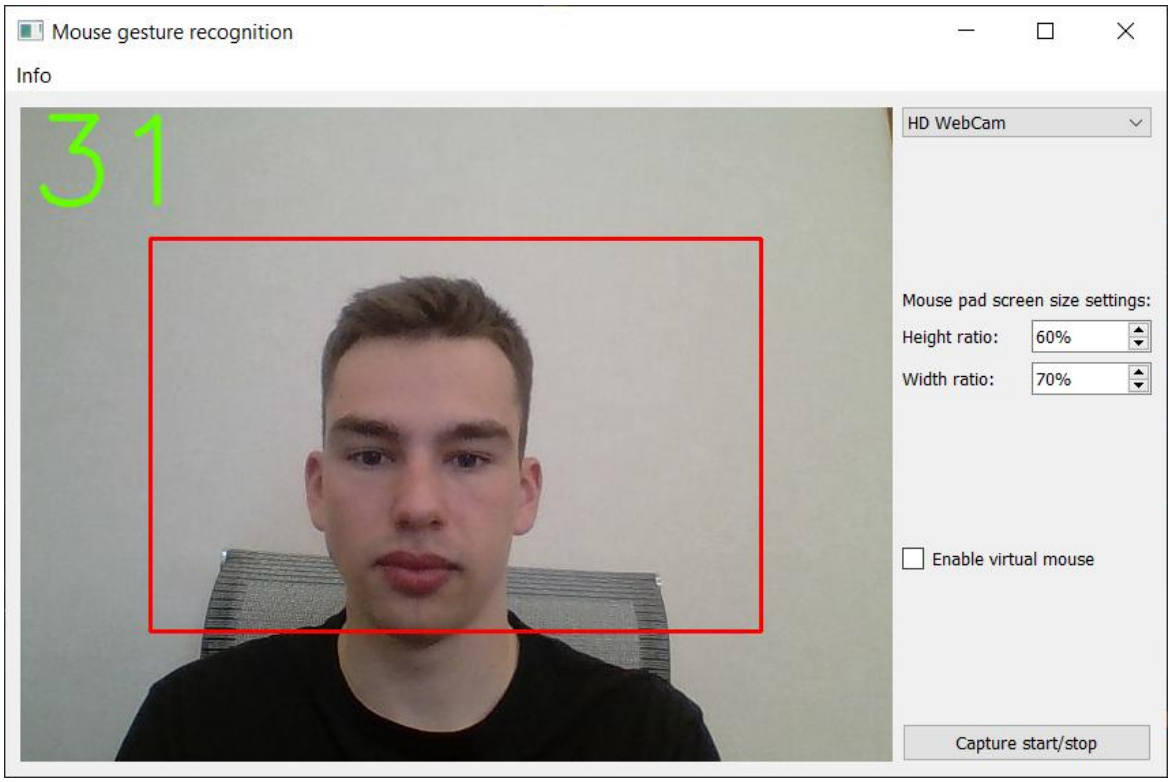


Рисунок 3.1 Інтерфейс застосунку

На Рисунку 3.1 видно, що порівняно із представленою на Рисунку 2.9 концепцією основного вікна інтерфейсу до програми було додано певні елементи поверх блоку із вхідним зображенням, а саме – червоний квадрат, що

відображає зону взаємодії користувача із курсором. Вона може змінювати свої розміри в режимі реального часу, в залежності від налаштувань, що встановлює користувач (Height ratio та Width ratio). Це було зроблено для спрощення розуміння користувачем зміни налаштувань за рахунок їх візуалізації. А також було додано цифри у верхньому лівому куті для відображення кількості кадрів за секунду, що зчитує та відображає система.

До особливостей реалізації інтерфейсу застосунку відноситься використання бібліотеки cv2 саме із підтримкою роботи з графічними процесорами для більш продуктивної взаємодії з зображеннями. У створюваному застосунку зчитування відеопотоку та передача зображення відбувається силами GPU, у той час як решта інтерфейсу викликається та працює із використанням CPU.

Для коректної роботи із вхідним зображенням, у бібліотеці PyQt5 передбачено клас QThread, що дозволяє реалізовувати роботу будь якого фонового процесу для запобігання підлагування самого графічного інтерфейсу. Для цього було створено клас CameraThread, що наслідує батьківський QThread, із ініціалізацією стандартних параметрів розміру вхідного зображення відеокамери та параметрами для зони взаємодії із GUI.

Для зчитування позиції долоні використовується бібліотека MediaPipe. MediaPipe — це фреймворк для створення конвеєрів машинного навчання для обробки даних часових рядів, таких як відео, аудіо тощо. Це крос-платформний фреймворк, що працює на настільних комп’ютерах/серверах, Android, iOS та інших вбудованих пристроях, таких як Raspberry Pi та Jetson Nano.

З його допомогою розпізнається та відмальовується скелет руки, за допомогою якого у межах червоного прямокутника відбувається пересування курсору. Для цього беруться відповідні координати точки, що знаходиться на зап’ясті, відносно позиції всередині червоного прямокутника, далі ці координати пропорційно перераховуються та проектуються відповідно до особливостей розміру кожного окремого екрану.

У налаштуваннях методу, що викликається бібліотекою для розпізнавання скелету долоні стоїть обмеження на одночасне розпізнавання лише 1 долоні із значенням нижньої межі впевненості розпізнавання долоні рівним 50%. Виглядає це наступним чином:

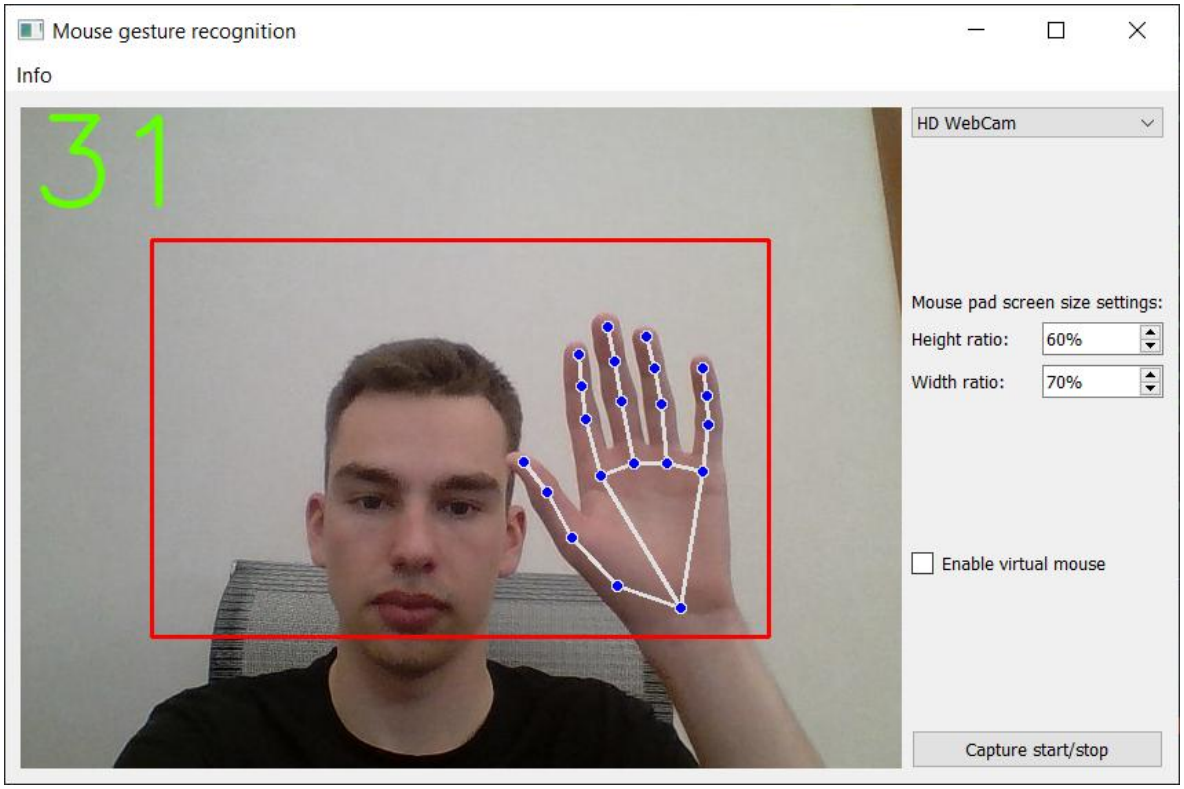


Рисунок 3.2 Розпізнавання скелету долоні за допомогою MediaPipe

3.2 Реалізація нейронної мережі

Етап реалізації нейронної мережі передбачає побудову та навчання самої мережі. Для обраної моделі 3D ResNet було реалізовано клас Resnet3DBuilder, що може створювати різні архітектури, в залежності від обраного методу та переданих параметрів, для цього передбачено методи:

- build_resnet_18
- build_resnet_34
- build_resnet_50
- build_resnet_101

- build_resnet_152

3.2.1 Налаштування нейронної мережі

Оскільки усі основні налаштування нейронної мережі були розглянуті у попередньому розділі, для етапу реалізації важливим було обрати та налаштувати саме функції зворотного виклику – callbacks.

Функція зворотного виклику – це об’єкт, який може виконувати дії на різних етапах навчання (наприклад, на початку або в кінці епохи, до або після однієї партії тощо). [18] Їх налаштування є одним із найважливіших етапів навчання, оскільки воно дає змогу отримувати уявлення про внутрішні стани та статистику моделі під час навчання, відслідковувати за метриками, зберігати проміжні етапи навчання, візуалізувати як зміну параметрів, так і саму архітектуру нейронної мережі, тощо.

Для навчання моделей були обрані та написані наступні функції зворотного виклику бібліотеки TensorFlow:

1. LearningRateScheduler – швидкість навчання залежить від епохи, і задається наступною формулою: lr = lr * 1 / (1 + decay * epoch)
2. EarlyStopping – завчасна зупинка, слідкує за зміною метрики val_categorical_accuracy, за умови що протягом 5 епох зміна не перевищить різницю в точності 0.001. Якщо спрацює ця функція зворотного виклику, модель повинна припинити навчання та повернути ваги моделі, що були знайдені відповідно до найкращої val_categorical_accuracy
3. ModelCheckpoint – збереження матриці ваг для найкращої епохи. Слідкує за тим, аби точність проміжної перевірки val_categorical_accuracy була максимальною і зберігає за шаблоном: “назва моделі”.best.h5, приклад: resnet34.best.h5
4. ModelCheckpoint – збереження матриці ваг для кожної епохи. Збереження відбувається за наступним шаблоном: “назва моделі”.

“номер епохи”-cat_acc_“точність під час навчання”-“значення фінкції втрат”-val_acc_“точність під час проміжної перевірки”.h5, приклад: model.05-cat_acc_0.6232-val_loss_1.6226-val_acc_0.5739.h5

5. TensorBoard – інструмент, що зберігає дані моделі для візуалізації. Надається разом із TensorFlow і включає в себе візуалізацію зведених графіків показників, візуалізацію графіків навчання, гістограми ваг та зразкове профілювання. Збереження включає в себе цілий набір файлів, для яких треба задати назву папки. Збереження відбувається за наступним шаблоном: “назва моделі”-“дата і час”, приклад: resnet34_2022-04-25_21-55-06
6. CSVLogger – функція зворотного виклику, що передає відповідні результати кожної епохи у файл із розширенням CSV. Для збереження метрик було обрано: epoch, categorical_accuracy, loss, lr, val_categorical_accuracy, val_loss, а сам файл збереження містить наступнку структуру: “назва моделі”-“дата і час”_train_logs.csv, приклад: resnet34_2022-04-24_15-48-19_train_logs.csv

Після того як усі налаштування для навчання нейронної мережі було зроблено, для виконання високоефективних обчислень на графічному процесорі, додатково було встановлено CUDA v11.6 та cuDNN v8.4.0, наглядніший опис представлений на Рисунку 3.3.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 511.65       Driver Version: 511.65       CUDA Version: 11.6     |
|-------------------------------+----------------------+----------------------+
| GPU  Name            TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ... WDDM  | 00000000:01:00.0 Off |                  N/A |
| N/A   46C    P8    N/A /  N/A |      0MiB /  4096MiB |      1%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Рисунок 3.3 Інтерфейс керування системою NVIDIA

3.2.2 Навчання нейронної мережі

Для навчання та подальшого їх порівняння було обрано дві архітектури: 3D-ResNet-34 та 3D-ResNet-50, оскільки менші архітектури програють у точності навчання, а більш масивні – є більш ресурсо-затратними для задіяння в режимі реального часу.

Навчання відбувалося засобами відеокарти NVIDIA GeForce GTX 1050 Ti Mobile, тренування 1 епохи на якій займало 12 хвилин, але за умови відсутності додаткового попередньо встановленого та налаштованого програмного забезпечення CUDA та cuDNN для підтримки GPU, навчання 1 епохи моделі тої ж самої архітектури виконане на процесорі Intel Core i7-7700HQ займало близько 2 годин.

Навчання виконувалось без попередньої ініціалізація ваг за рахунок вже навчених нейронних мереж на схожих задачах класифікації, а тому виступає базовою моделлю.

Порівняння графіків залежності точності на навчальній та валідаційній виборках від епохи для 3D-ResNet-34 та 3D-ResNet-50.

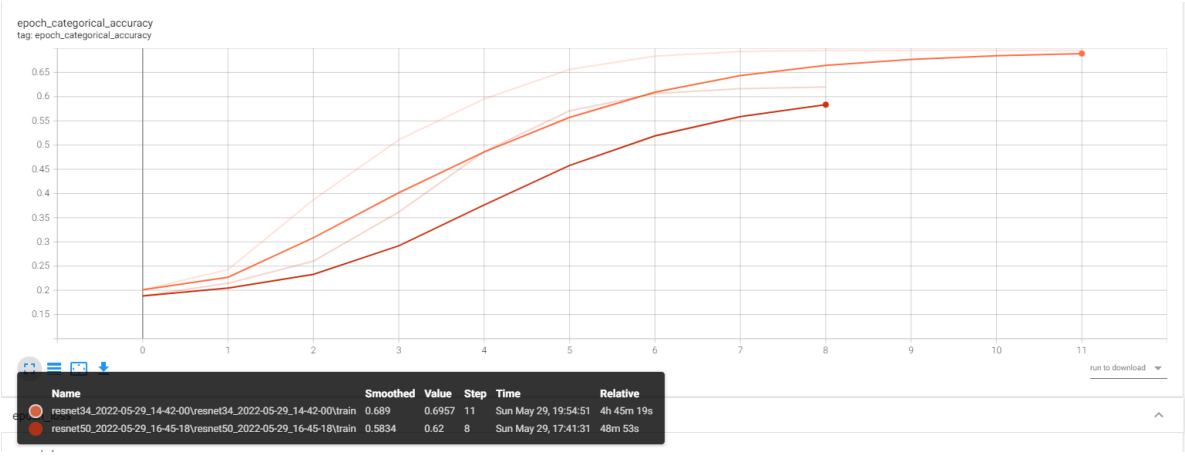


Рисунок 3.4 Графік точності 3D-ResNet-34 та 3D-ResNet-50 на навчальній виборці

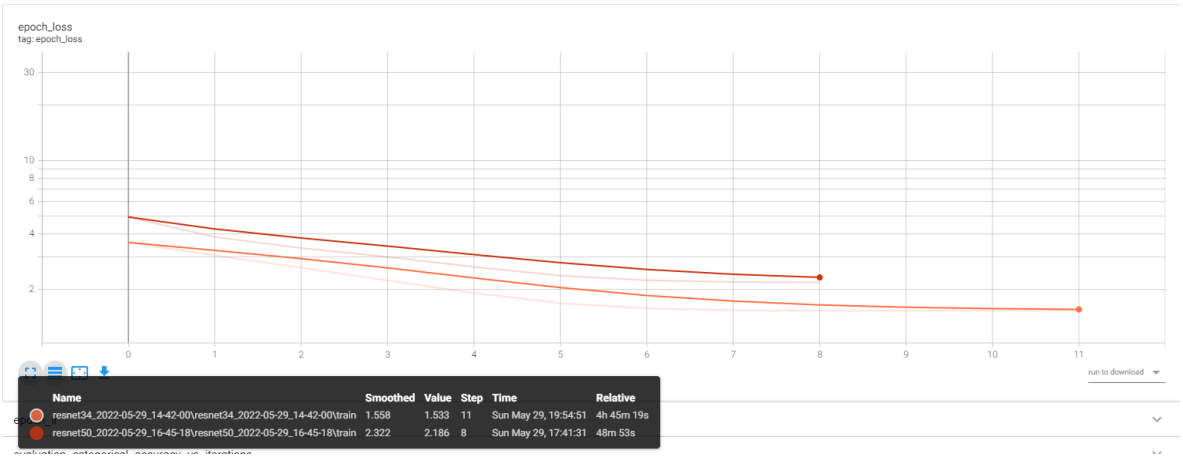


Рисунок 3.5 Графік значень функції втрат 3D-ResNet-34 та 3D-ResNet-50 на навчальній виборці

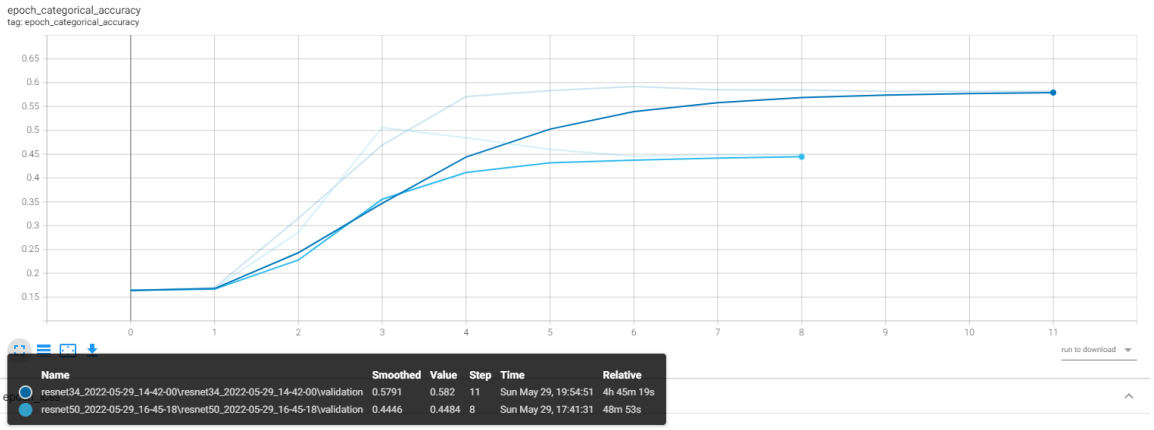


Рисунок 3.6 Графік точності 3D-ResNet-34 та 3D-ResNet-50 на валідаційній виборці

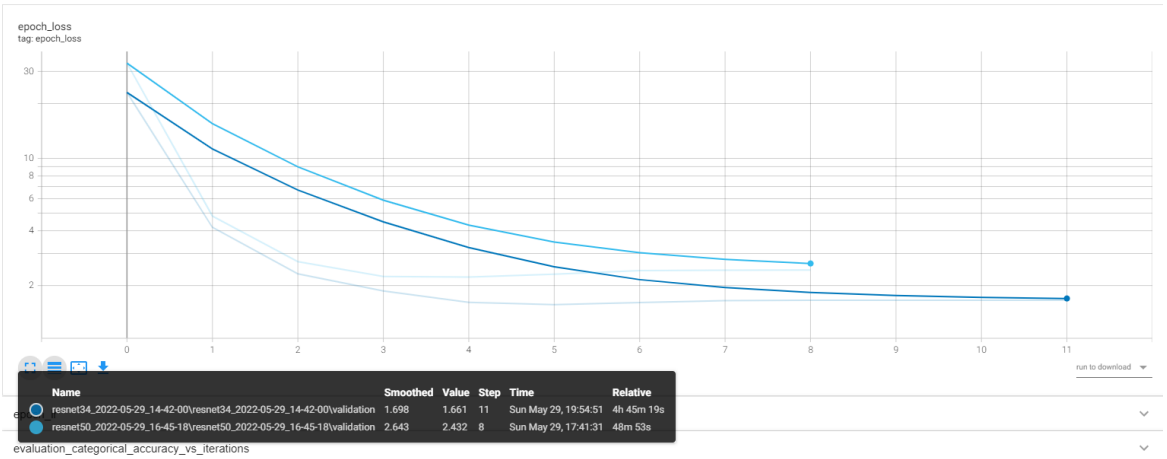


Рисунок 3.7 Графік значеннь функції втрат 3D-ResNet-34 та 3D-ResNet-50 на валідаційній виборці

У результаті навчання, 3D-ResNet-34 показала найвищу валідаційну точність на 8 епосі зі значенням 0.5919% на валідаційній виборці та 0.6836% на навчальній виборці, зупинившись на 11 епосі після активації функції зворотного виклику EarlyStopping. Модель 3D-ResNet-50, в свою чергу, показала найвищу валідаційну точність на 3 епосі зі значенням 0.5056% на валідаційній виборці та 0.3615% на навчальній виборці, так само зупинившись після активації функції зворотного виклику EarlyStopping, але вже на 8 епосі. Тим самим, зарахунок підвищенної складності архітектури, модель 3D-ResNet-50 була більш схильна до перенавчання і показала нижчі результати при рівних показниках.

Більш детально з етапами навчання та змінами метрик можна ознайомитись завдяки таблицям, отриманим за допомогою функції зворотного виклику CSVLogger, що записувала інформацію в кінці кожної епохи.

epoch	categorical_accuracy	loss	lr	val_categorical_accuracy	val_loss
0	0.18816538155078888	4.9371867179870605	0.001	0.16459627449512482	33.35795593261719
1	0.2144095003604889	3.8486602306365967	0.0006666667	0.16832298040390015	4.81087589263916
2	0.2602129280567169	3.3503854274749756	0.00033333336	0.28633540868759155	2.7054014205932617
3	0.3614756166934967	3.0035862922668457	0.00013333335	0.5055900812149048	2.242988348007202
4	0.48650655150413513	2.6455979347229004	4.444445e-05	0.48447203636169434	2.2285661697387695
5	0.5706858038902283	2.3758645057678223	1.2698414e-05	0.4602484405040741	2.3102903366088867
6	0.6063382029533386	2.242185592651367	3.1746035e-06	0.4453416168689728	2.4138615131378174
7	0.6164892315864563	2.1961348056793213	7.0546747e-07	0.44844719767570496	2.4283056259155273
8	0.6199554204940796	2.1856799125671387	1.410935e-07	0.44844719767570496	2.4323625564575195

Рисунок 3.8 Таблиця навчання 3D-ResNet-50 відповідно до епохи

epoch	categorical_accuracy	loss	lr	val_categorical_accuracy	val_loss
0	0.2010398656129837	3.5909814834594727	0.001	0.16397514939308167	23.028451919555664
1	0.24263431131839752	3.060307264328003	0.0006666667	0.1701863408088684	4.184625148773193
2	0.38722455501556396	2.620323657989502	0.00033333336	0.31552794575691223	2.3168556690216064
3	0.5107699632644653	2.2386844158172607	0.00013333335	0.46894410252571106	1.8655990362167358
4	0.5954444408416748	1.912612795829773	4.444445e-05	0.5708074569702148	1.6150727272033691
5	0.6561030149459839	1.6818841695785522	1.2698414e-05	0.5832298398017883	1.568958044052124
6	0.6835850477218628	1.5774356126785278	3.1746035e-06	0.5919254422187805	1.610453486442566
7	0.6932408809661865	1.5424596071243286	7.0546747e-07	0.5850931406021118	1.6502283811569214
8	0.6954692006111145	1.5344514846801758	1.410935e-07	0.5844720602035522	1.6590547561645508
9	0.6957167387008667	1.5329701900482178	2.5653362e-08	0.5819875597953796	1.6607563495635986
10	0.6957167387008667	1.5327123403549194	4.2755604e-09	0.5819875597953796	1.6610649824142456
11	0.6957167387008667	1.5326740741729736	6.577785e-10	0.5819875597953796	1.6611127853393555

Рисунок 3.9 Таблиця навчання 3D-ResNet-34 відповідно до епохи

3.3 Проекція жестів на GUI

Після того як отриману модель було інтегровано, для повноцінної роботи системи було описано клас GuiGestures, метод gesture_to_gui_mapper якого відповідає за проекцію розпізнаного жесту на графічний інтерфейс користувача за допомогою бібліотеки pyautogui. Кожен розпізнаний жест проектується відповідно до значень, вказаних у наступній таблиці:

Жест	Проекція
D0X: Non-gesture	-
B0A: Pointing with one finger	-
B0B: Pointing with two fingers	-
G01: Click with one finger	Ліва кнопка миші
G02 Click with two fingers	Права кнопка миші
G03: Throw up	-
G04: Throw down	-
G05: Throw left	-
G06: Throw right	-
G07: Open twice	Ліва кнопка миші двічі
G08: Double click with one finger	Ліва кнопка миші двічі
G09: Double click with two fingers	Права кнопка миші двічі
G10: Zoom in	Ctrl + “+”
G10: Zoom out	Ctrl + “-”

Рисунок 3.9 Таблиця проекції розпізнаних жестів на графічний інтерфейс

3.4 Тест-кейси, що демонструють коректність роботи застосунку

Приклад роботи системи за умови встановленого зовнішнього пристрою зчитування (відеокамери) та зміненими параметрами для зони взаємодії із інтерфейсом зображено на Рисунку 3.10.

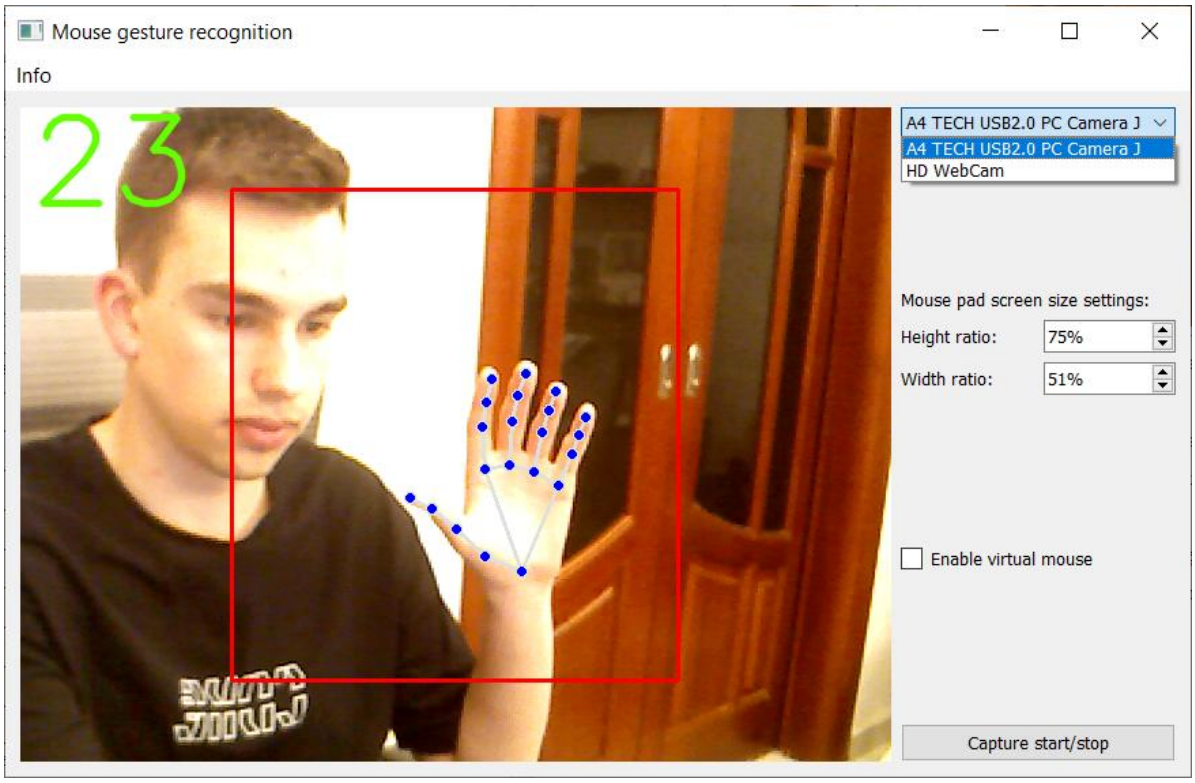


Рисунок 3.10 Приклад роботи системи із зміненими параметрами

Приклад роботи системи із увімкненою системою керування жестами містить 2 інформативних меню-попередження:

1. Попередження про увімкнення системи керування жестами

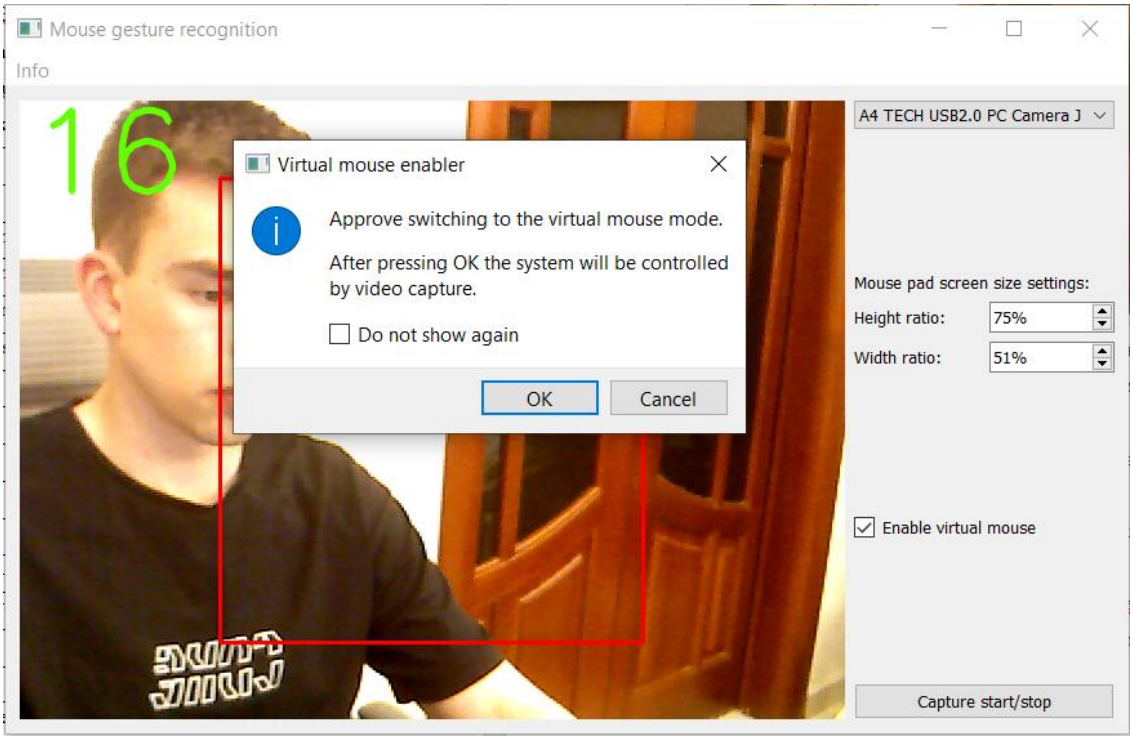


Рисунок 3.11 Приклад попередження про увімкнення системи керування жестами

2. Попередження про призупинений вхідний потік, що не дозволяє увімкнути керування жестами

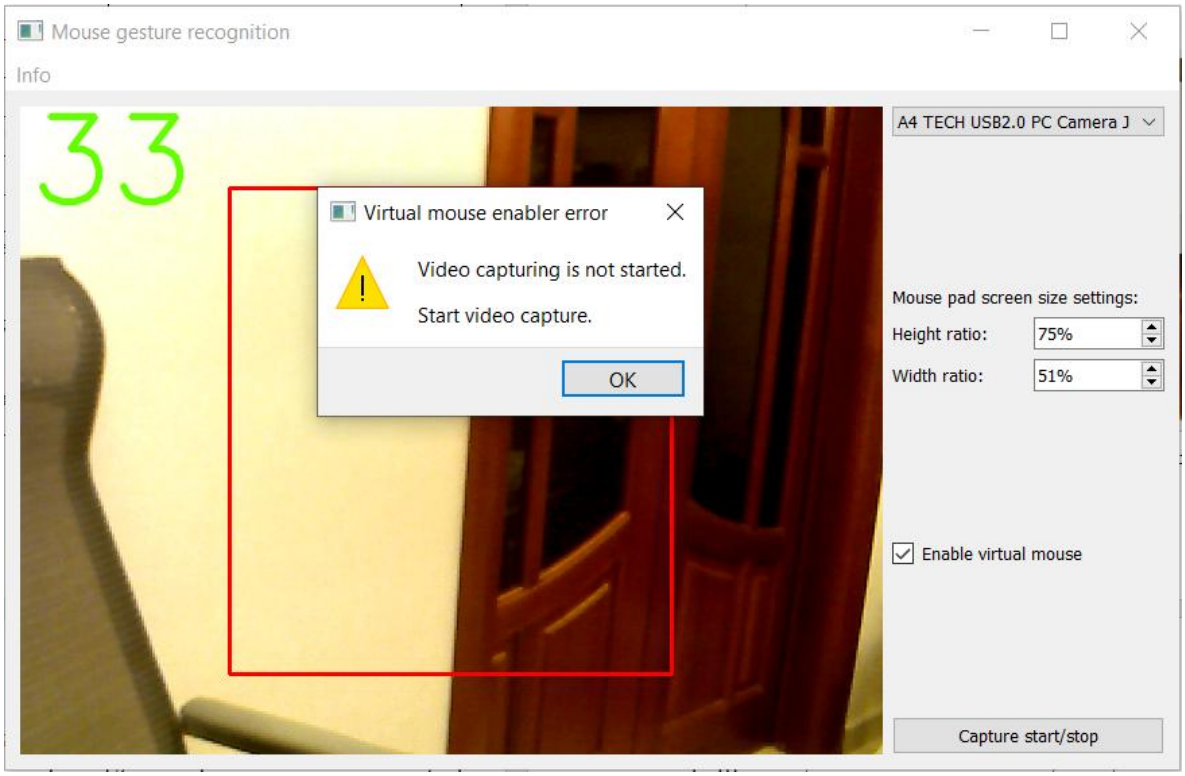


Рисунок 3.12 Приклад попередження про призупинений вхідний потік

ВИСНОВКИ

У ході виконання даної роботи було реалізовано повноцінну систему для взаємодії користувача та графічного інтерфейсу.

Робота передбачала створення двох функціональних частин: графічного інтерфейсу користувача та системи розпізнавання жестів. Система розпізнавання жестів – комплекс з двох модулів, першого – для визначення позиції курсора відносно обмеженої площини і її проекції у відповідності до позиції всередині площини на інтерфейс, та другого – самої нейронної мережі, що отримує вхідний потік зображень та розпізнає їх відповідно до закладених жестів для подальшої передачі на інтерфейс.

Для розпізнавання жестів було реалізовано, натреновано та порівняно моделі 3D-ResNet-34 та 3D-ResNet-50, з яких для роботи самої системи було обрано 3D-ResNet-34, який класифікує 14 жестів із точністю 59%, що у 8 разів вище за імовірнісне прогнозування жесту із набору.

Даної швидкодії та точності моделі недостатньо для побудови правильної та повноцінної система розпізнавання жестів долоні для взаємодії з інтерфейсом, тому для подальшої розробки над покращеннями є ідея замінити окрему нейронну мережу 3D-ResNet, що працює із зображеннями, на рекурентну нейронну мережу, що буде працювати із точками, які повертає бібліотека MediaPipe для побудови скелету долоні. Це дасть змогу підвищити швидкодію самої системи, а також може покращити точність. Також можна розширити функціональність системи, додавши нові команди на існуючі жести, що не задіються, або ж надати можливість користувачеві самому їх задавати.

ЛІТЕРАТУРА

1. Aashni Haria,Archanasri Subramanian,Nivedhitha Asokkumar,Shristi Poddar,Jyothi S Nayak, “Hand Gesture Recognition for Human Computer Interaction”
2. Aditya, Kunwar & Chacko, Praise & Kumari, Deeksha & Kumari, Divya & Bilgaiyan, Saurabh. (2018). Recent Trends in HCI: A survey on Data Glove, LEAP Motion and Microsoft Kinect. 1-5. 10.1109/ICSCAN.2018.8541163.
3. T. Osunkoya, and J.C. Chern, “Gesture-Based Human-Computer-Interaction Using Kinect for Windows Mouse Control and PowerPointPresentation,”
4. D. Chaudhari, C. Patil, M. Magar, and G. Pagar ”Gesture Based HCI andSign Language Recognition using Kinect Sensor,” International Journalof Computer Science and Mobile Computing, vol. 4, No. 10, pp. 168-174, October 2015.
5. L. Song, R. Hu, Y. Xiao, and L. Gong, ”Real- Time 3D Hand Trackingfrom Depth Images,” 2nd International Conference On Systems Engi-neering and Modeling, pp. 1119-1122
6. ”L. Motion”, API Reference - Leap Motion Java SDK. [Електронний ресурс] // Режим доступу: https://developer.leapmotion.com/documentation/java/api/Leap Classes.html
7. ”L. Motion”, API Reference - Leap Motion. [Електронний ресурс] // Режим доступу: http://blog.leapmotion.com/hardware-to- software-how-does- the-leap- motion-controller- work/.
8. Dipti Mathpal “Hand Gesture Recognition: Analysis of different Techniques”

9. Конкіній Н.С., “Система розпізнавання жестів для людинно-машинної взаємодії”
10. Joanna Materzynska, Guillaume Berger, Ingo Bax, Roland Memisevic “The Jester Dataset: A Large-Scale Video Dataset of Human Gestures”
11. Gibran Benitez-Garcia, Jesus Olivares-Mercado, Gabriel Sanchez-Perez, Keiji Yanai “IPN Hand: A Video Dataset and Benchmark for Real-Time Continuous Hand Gesture Recognition”
12. T. B. Neurons, “Gesture recognition using end-to-end learning from a large video database,” Sept. 2017.
13. Phat Nguyen Huu, Tan Phung Ngoc “Hand Gesture Recognition Algorithm Using SVM and HOG Model for Control of Robotic System”
14. UML - Behavioral Diagram vs Structural Diagram. [Електронний ресурс] // Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/behavior-vs-structural-diagram/#:~:text=Behavioral%20Diagrams,-UML's%20five%20behavioral&text=It%20shows%20how%20the%20system,system%20to%20change%20internal%20states
15. UML – Use Case Diagrams. [Електронний ресурс] // Режим доступу: https://www.tutorialspoint.com/uml/uml_use_case_diagram.htm#:~:text=Use%20case%20diagrams%20consists%20of,use%20case%20diagrams%20are%20used
16. Difference Between Use Case Diagram and Activity Diagram. [Електронний ресурс] // Режим доступу: https://www.differencebetween.com/difference-between-use-case-diagram-and-activity-diagram/#:~:text=Use%20case%20diagram%20and%20activity%20diagram%20are%20behavioral%20UML%20diagrams,work%20flow%20of%20the%20system.

17. What is TensorFlow? The machine learning library explained [Електронний ресурс] // Режим доступу: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html
18. Callbacks API [Електронний ресурс] // Режим доступу: https://keras.io/api/callbacks/#:~:text=A%20callback%20is%20an%20object,save%20your%20model%20to%20disk

ДОДАТОК

Клас для побудови 3D-ResNet

```
"""
Application :   Tensorflow ResNet
Module      :   resnet3d.py
Developers  :   Andrii Shatokhin
Description :   Defines ResNet class
"""

from __future__ import (
    absolute_import,
    division,
    print_function,
    unicode_literals
)

from math import ceil

import six
import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras.initializers import GlorotNormal
# TODO: Import from tf in future. Error with _keras_shape usage
# from keras.layers import Input
from tensorflow.keras.layers import (
    Activation, Dense, Flatten, Add,
    BatchNormalization, Input,
    Conv3D, AveragePooling3D, MaxPooling3D
)
from tensorflow.keras import Model
from tensorflow.keras.regularizers import L2


def _bn_relu(x_input):
    """Helper to build a BN -> relu block (by @raghakot)."""
    norm_input = BatchNormalization(axis=CHANNEL_AXIS)(x_input)
    norm_input = Activation("relu")(norm_input)
    return norm_input


def _conv_bn_relu3D(**conv_params):
    filters = conv_params["filters"]
    kernel_size = conv_params["kernel_size"]
    strides = conv_params.setdefault("strides", (1, 1, 1))
    kernel_initializer = conv_params.setdefault("kernel_initializer", GlorotNormal())
    padding = conv_params.setdefault("padding", "same")
    kernel_regularizer = conv_params.setdefault("kernel_regularizer", L2(1e-4))
```

```
    def f(x_input):
        conv = Conv3D(filters=filters, kernel_size=kernel_size,
                      strides=strides, kernel_initializer=kernel_initializer,
                      padding=padding,
                      kernel_regularizer=kernel_regularizer)(x_input)
        return _bn_relu(conv)

    return f


def _bn_relu_conv3d(**conv_params):
    """Helper to build a  BN -> relu -> conv3d block."""
    filters = conv_params["filters"]
    kernel_size = conv_params["kernel_size"]
    strides = conv_params.setdefault("strides", (1, 1, 1))
    kernel_initializer = conv_params.setdefault("kernel_initializer", GlorotNormal())
    padding = conv_params.setdefault("padding", "same")
    kernel_regularizer = conv_params.setdefault("kernel_regularizer", L2(1e-4))

    def f(x_input):
        activation = _bn_relu(x_input)
        return Conv3D(filters=filters, kernel_size=kernel_size,
                      strides=strides, kernel_initializer=kernel_initializer,
                      padding=padding,
                      kernel_regularizer=kernel_regularizer)(activation)

    return f


def _shortcut3d(x_input, residual):
    """3D shortcut to match input and residual and merges them with "sum"."""
    stride_dim1 = ceil(x_input.shape.as_list()[DIM1_AXIS] / residual.shape.as_list()[DIM1_AXIS])
    stride_dim2 = ceil(x_input.shape.as_list()[DIM2_AXIS] / residual.shape.as_list()[DIM2_AXIS])
    stride_dim3 = ceil(x_input.shape.as_list()[DIM3_AXIS] / residual.shape.as_list()[DIM3_AXIS])
    equal_channels = residual.shape.as_list()[CHANNEL_AXIS] == x_input.shape.as_list()[CHANNEL_AXIS]

    shortcut = x_input
    if stride_dim1 > 1 or stride_dim2 > 1 or stride_dim3 > 1 \
            or not equal_channels:
        shortcut = Conv3D(
            filters=residual.shape.as_list()[CHANNEL_AXIS],
            kernel_size=(1, 1, 1),
            strides=(stride_dim1, stride_dim2, stride_dim3),
            kernel_initializer="he_normal", padding="valid",
            kernel_regularizer=L2(1e-4)
        )(x_input)
    return Add()([shortcut, residual])


def _residual_block3d(block_function, filters, kernel_regularizer, repetitions,
                      is_first_layer=False):
```

```
    def f(x_input):
        for i in range(repetitions):
            strides = (1, 1, 1)
            if i == 0 and not is_first_layer:
                strides = (2, 2, 2)
            x_input = block_function(filters=filters, strides=strides,
                                     kernel_regularizer=kernel_regularizer,
                                     is_first_block_of_first_layer=(
                                             is_first_layer and i == 0)
                                     )(x_input)
        return x_input

    return f


def basic_block(filters, strides=(1, 1, 1), kernel_regularizer=L2(1e-4),
                is_first_block_of_first_layer=False):
    """Basic 3 X 3 X 3 convolution blocks. Extended from raghakot's 2D impl."""

    def f(x_input):
        if is_first_block_of_first_layer:
            # don't repeat bn->relu since we just did bn->relu->maxpool
            conv1 = Conv3D(filters=filters, kernel_size=(3, 3, 3),
                           strides=strides, padding="same",
                           kernel_initializer=GlorotNormal(),
                           kernel_regularizer=kernel_regularizer
                           )(x_input)
        else:
            conv1 = _bn_relu_conv3d(filters=filters,
                                    kernel_size=(3, 3, 3),
                                    strides=strides,
                                    kernel_regularizer=kernel_regularizer
                                    )(x_input)

        residual = _bn_relu_conv3d(filters=filters, kernel_size=(3, 3, 3),
                                   kernel_regularizer=kernel_regularizer
                                   )(conv1)
        return _shortcut3d(x_input, residual)

    return f


def bottleneck(filters, strides=(1, 1, 1), kernel_regularizer=L2(1e-4),
               is_first_block_of_first_layer=False):
    """Basic 3 X 3 X 3 convolution blocks. Extended from raghakot's 2D impl."""

    def f(x_input):
        if is_first_block_of_first_layer:
            # don't repeat bn->relu since we just did bn->relu->maxpool
            conv_1_1 = Conv3D(filters=filters, kernel_size=(1, 1, 1),
                              strides=strides, padding="same",
```

```
                            kernel_initializer=GlorotNormal(),
                            kernel_regularizer=kernel_regularizer
                            )(x_input)
        else:
            conv_1_1 = _bn_relu_conv3d(filters=filters, kernel_size=(1, 1, 1),
                                       strides=strides,
                                       kernel_regularizer=kernel_regularizer
                                       )(x_input)

        conv_3_3 = _bn_relu_conv3d(filters=filters, kernel_size=(3, 3, 3),
                                   kernel_regularizer=kernel_regularizer
                                   )(conv_1_1)
        residual = _bn_relu_conv3d(filters=filters * 4, kernel_size=(1, 1, 1),
                                   kernel_regularizer=kernel_regularizer
                                   )(conv_3_3)

        return _shortcut3d(x_input, residual)

    return f


def _handle_data_format():
    global DIM1_AXIS
    global DIM2_AXIS
    global DIM3_AXIS
    global CHANNEL_AXIS

    if K.image_data_format() == 'channels_last':
        DIM1_AXIS = 1
        DIM2_AXIS = 2
        DIM3_AXIS = 3
        CHANNEL_AXIS = 4
    else:
        CHANNEL_AXIS = 1
        DIM1_AXIS = 2
        DIM2_AXIS = 3
        DIM3_AXIS = 4


def _get_block(identifier):
    if isinstance(identifier, six.string_types):
        res = globals().get(identifier)
        if not res:
            raise ValueError('Invalid {}'.format(identifier))
        return res
    return identifier


class Resnet3DBuilder(object):
    """ResNet3D."""
```

```
@staticmethod
def build(input_shape, num_outputs, block_fn, repetitions, reg_factor):
    """Instantiate a vanilla ResNet3D keras model.

    # Arguments
        input_shape: Tuple of input shape in the format
        (conv_dim1, conv_dim2, conv_dim3, channels) if dim_ordering='tf'
        (filter, conv_dim1, conv_dim2, conv_dim3) if dim_ordering='th'
        num_outputs: The number of outputs at the final softmax layer
        block_fn: Unit block to use {'basic_block', 'bottlenack_block'}
        repetitions: Repetitions of unit blocks
    # Returns
        model: a 3D ResNet model that takes a 5D tensor (volumetric images
        in batch) as input and returns a 1D vector (prediction) as output.
    """
    _handle_data_format()
    if len(input_shape) != 4:
        raise ValueError("""
            Input shape should be a tuple
            (conv_dim1, conv_dim2, conv_dim3, channels)
            for tensorflow as backend or
            (channels, conv_dim1, conv_dim2, conv_dim3)
            for theano as backend
            """)

    block_fn = _get_block(block_fn)
    x_input = Input(shape=input_shape)
    # first conv
    conv1 = _conv_bn_relu3D(filters=64, kernel_size=(7, 7, 7),
                            strides=(2, 2, 2),
                            kernel_regularizer=L2(reg_factor)
                            )(x_input)
    pool1 = MaxPooling3D(pool_size=(3, 3, 3), strides=(2, 2, 2),
                         padding="same")(conv1)

    # repeat blocks
    block = pool1
    filters = 64
    for i, r in enumerate(repetitions):
        block = _residual_block3d(block_fn, filters=filters,
                                  kernel_regularizer=L2(reg_factor),
                                  repetitions=r, is_first_layer=(i == 0)
                                  )(block)
        filters *= 2

    # last activation
    block_output = _bn_relu(block)

    # average poll and classification
    pool2 = AveragePooling3D(pool_size=(block.shape.as_list()[DIM1_AXIS],
                                        block.shape.as_list()[DIM2_AXIS],
```

```
                                    block.shape.as_list()[DIM3_AXIS]),
                            strides=(1, 1, 1))(block_output)
    flatten1 = Flatten()(pool2)
    if num_outputs > 1:
        dense = Dense(units=num_outputs,
                      kernel_initializer="he_normal",
                      activation="softmax",
                      kernel_regularizer=L2(reg_factor))(flatten1)
    else:
        dense = Dense(units=num_outputs,
                      kernel_initializer="he_normal",
                      activation="sigmoid",
                      kernel_regularizer=L2(reg_factor))(flatten1)

    model = Model(inputs=x_input, outputs=dense)
    return model

@staticmethod
def build_resnet_18(input_shape, num_outputs, reg_factor=1e-4):
    """Build resnet 18."""
    return Resnet3DBuilder.build(input_shape, num_outputs, basic_block,
                                 [2, 2, 2, 2], reg_factor=reg_factor)

@staticmethod
def build_resnet_34(input_shape, num_outputs, reg_factor=1e-4):
    """Build resnet 34."""
    return Resnet3DBuilder.build(input_shape, num_outputs, basic_block,
                                 [3, 4, 6, 3], reg_factor=reg_factor)

@staticmethod
def build_resnet_50(input_shape, num_outputs, reg_factor=1e-4):
    """Build resnet 50."""
    return Resnet3DBuilder.build(input_shape, num_outputs, bottleneck,
                                 [3, 4, 6, 3], reg_factor=reg_factor)

@staticmethod
def build_resnet_101(input_shape, num_outputs, reg_factor=1e-4):
    """Build resnet 101."""
    return Resnet3DBuilder.build(input_shape, num_outputs, bottleneck,
                                 [3, 4, 23, 3], reg_factor=reg_factor)

@staticmethod
def build_resnet_152(input_shape, num_outputs, reg_factor=1e-4):
    """Build resnet 152."""
    return Resnet3DBuilder.build(input_shape, num_outputs, bottleneck,
                                 [3, 8, 36, 3], reg_factor=reg_factor)
```