МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____202___ р.

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

ІНТЕЛЕКТУАЛЬНА СИСТЕМА ПЕРЕТВОРЕННЯ УКРАЇНСЬКОЇ ЖЕСТОВОЇ МОВИ НА ТЕКСТ ТА АУДІО

Спеціальність 122 «Комп'ютерні науки»

122 – КРМ – 601.21810324

Виконала студентка 6-го курсу, групи 601
_____Т. О. Удовик
«___» _____ 2024 р.

Керівник: канд. фіз.-мат. наук, доцент
_____І. В. Кулаковська
«___» _____ 2024 р.

Миколаїв – 2024

Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Освітньо-кваліфікаційний рівень **магістр**
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____20___ р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Удовик Тетяні Олександрівні
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи магістра «Інтелектуальна система перетворення української жестової мови на текст та аудіо».
Керівник роботи Кулаковська Інесса Василівна, канд. фіз.-мат. наук, доцент.
Затв. наказом Ректора ЧНУ ім. Петра Могили від «01» лютого 2024 р. № 20
2. Строк подання студентом роботи 20 лютого 2024 р.
3. Вхідні (початкові) дані до роботи: набір даних для тренування моделі, відеопотік жестового мовлення користувачів у реальному часі.
Очікуваний результат роботи: розроблена система розпізнавання жестів з високою точністю та їх перетворення на текст та аудіо, використовуючи нейронні мережі.
4. Зміст пояснювальної записки (перелік питань, які потрібно розглянути):

– аналіз та огляд досягнень у сфері перекладу жестових мов;

– методи, інформаційні технології для вирішення задачі розпізнавання жестів;

– розробка інтелектуальної системи перетворення української жестової мови на текст та аудіо;

– програмна реалізація та тестування системи;

– аналіз результатів роботи.

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини: Охорона праці та безпека у надзвичайних ситуаціях.

7. Консультанти:

Розділ	**Прізвище, ініціали та посада консультанта**	**Підпис**
Спеціальна частина з охорони праці	д-р біол. наук, проф. Григор'єва Л. І.	
Методична частина	канд. фіз.-мат. наук, доцент. Кулаковська І. В.	

Керівник роботи __канд. фіз.-мат. наук, доцент. Кулаковська І. В.___
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ___Удовик Т. О._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «__31__» _____жовтня____ 2023 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи магістра

Тема: Інтелектуальна система перетворення української жестової мови на текст та аудіо.

№	Найменування роботи	Початок	Закінчення	Примітки
1	Визначення керівника і теми КРМ. Подання заяви на затвердження теми КРМ	01.09.2023	10.10.2023	Виконано
2	Отримання завдання на виконання КРМ	11.10.2023	01.11.2023	Виконано
3	Складання календарного плану на період виконання КРМ	02.11.2023	10.11.2023	Виконано
4	Огляд літератури за темою дослідження	11.11.2023	26.11.2023	Виконано
5	Проходження передатестаційної практики, збір та аналіз матеріалів до КРМ	27.11.2023	23.12.2023	Виконано
6	Аналіз предметної області та розробка технічного завдання. Моделювання результатів	25.12.2023	12.01.2024	Виконано
7	Опис фахової частини КРМ, зокрема дослідження публікацій щодо розпізнавання жестів, огляд існуючих методів та технологій для вирішення поставленої задачі, реалізація обраних технологій з аналізом отриманих результатів	13.01.2024	25.01.2024	Виконано
8	Розробка спеціальної частини з охорони праці та методичної частини	26.01.2024	02.02.2024	Виконано
9	Перший попередній захист КРМ на засіданні комісії кафедри	29.01.2024	29.01.2024	Виконано
10	Корегування роботи за результатами попереднього захисту	30.01.2024	05.02.2024	Виконано
11	Доробка та остаточне оформлення КРМ	06.02.2024	11.02.2024	Виконано
12	Другий попередній захист КРМ на засіданні комісії кафедри	12.01.2024	12.01.2024	Виконано
13	Подання КРМ, її електронної копії та інших документів (відгуку, рецензії) до захисту	19.02.2024	20.02.2024	Виконано
14	Захист КРМ перед екзаменаційною комісією (ЕК)	26.02.2024	27.02.2024	Виконано

Розробила студентка ___Удовик Тетяна Олександрівна_____ _____
(прізвище та ініціали) (*підпис*)

Керівник роботи канд. фіз.-мат. наук, доцент Кулаковська Інесса Василівна__ _____
(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

«____» _____ 202_ р.

АНОТАЦІЯ

до кваліфікаційної роботи магістра
студентки групи 601 ЧНУ ім. Петра Могили

Удовик Тетяни Олександрівни

на тему: **“ІНТЕЛЕКТУАЛЬНА СИСТЕМА ПЕРЕТВОРЕННЯ УКРАЇНСЬКОЇ ЖЕСТОВОЇ МОВИ НА ТЕКСТ ТА АУДІО”**

Актуальність теми даного дослідження полягає у вирішенні проблеми спілкування жестовою мовою та поліпшення якості життя людей з вадами слуху і мовлення.

Об’єктом дослідження є процес розпізнавання жестів.

Предметом дослідження є методи та алгоритми для розпізнавання жестів та розробка інтелектуальної системи розпізнавання.

Метою дослідження є покращення комунікації людей з вадами слуху та мовлення, шляхом розробки інтелектуальної системи перетворення української жестової мови на текст та аудіо.

В результаті виконання роботи було реалізовано інтелектуальну систему перетворення української жестової мови на текст та аудіо. Розроблена система здатна розпізнавати жести та перетворювати їх на текст, здійснювати переклад жестів у режимі реального часу, а також конвертувати текст у аудіо.

Робота складається з фахової частини, спеціальної частини з охорони праці та методичної частини. Пояснювальна записка складається зі вступу, чотирьох розділів та висновків. Перший розділ присвячений аналізу та огляду досягнень у сфері перекладу жестових мов. У другому розділі розглядаються методи та інформаційні технології для вирішення задачі розпізнавання жестів. Третій розділ містить опис процесу розробки інтелектуальної системи, зокрема, розробки дизайну та архітектури програмного забезпечення. Четвертий розділ присвячений програмній реалізації, тестуванню системи та аналізу отриманих результатів.

Загальний обсяг роботи – 147 сторінок. Кваліфікаційна робота магістра містить 1 додаток, 69 рисунків, 7 таблиць і посилання на 68 літературних джерел.

Ключові слова: жестова мова, розпізнавання жестів, нейронні мережі, LSTM, MediaPipe, Python, Keras, OpenCV.

ABSTRACT

to the master's qualification work by the student of the group 601 of Petro Mohyla Black Sea National University

Udovyk Tetiana

"AN INTELLIGENT SYSTEM FOR CONVERTING UKRAINIAN SIGN LANGUAGE INTO TEXT AND AUDIO"

The relevance of the topic of this paper is to solve the problem of communicating in sign language and improving the quality of life of people with hearing and speech impairments.

The object of this research is the process of gesture recognition.

The subject of research is methods and algorithms for gesture recognition and the development of an intelligent recognition system.

The purpose of the study is to improve communication between people with hearing and speech impairments by developing an intelligent system for converting Ukrainian sign language into text and audio.

As a result of the work, an intelligent system for converting Ukrainian sign language into text and audio was implemented. The developed system is capable of recognizing gestures and converting them into text, translating gestures in real time, and converting text to audio.

The work consists of a professional part, a special part on labor protection and a methodological part. The explanatory note consists of an introduction, four chapters, and conclusions. The first section is devoted to the analysis and review of achievements in the field of sign language translation. The second section discusses methods and information technologies for solving the problem of gesture recognition. The third section describes the process of developing an intelligent system, including software design and architecture. The fourth section is devoted to software implementation, system testing, and analysis of the results.

The total volume of the work is 147 pages. The master's thesis contains one appendix, 69 pictures, 7 tables and references to 68 literature sources.

Keywords: Ukrainian sign language, gesture recognition, neural networks, LSTM, MediaPipe, Python, Keras, OpenCV.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ВООЗ	–	Всесвітня організація охорони здоров’я
МН	–	Машинне навчання
ШІ	–	Штучний інтелект
CNN	–	Convolutional Neural Networks
DT	–	Decision Tree
DNN	–	Deep Neural Networks
KNN	–	K-nearest neighbors
LR	–	Logistic Regression
LSTM	–	Long Short-Term Memory
RF	–	Random Forest
RNN	–	Recurrent Neural Network
SVM	–	Support Vector Machine
UI	–	User Interface
UX	–	User Experience

ВСТУП

В сучасному світі, що швидко розвивається технологічно та глобалізується у соціальному аспекті, важливість комунікаційного взаємодії стає ключовою для ефективного функціонування суспільства. Більше 5% населення планети зазнає труднощів, пов’язаних із вадами слуху, що створює бар'єри для ефективного спілкування та взаємодії з навколишнім світом. За таких умов важливо розглядати не лише питання доступності інформації, але й забезпечення повноцінної участі цієї великої групи населення у суспільному житті.

Українська жестова мова є способом спілкування для осіб з вадами слуху або мовлення, проживаючими в Україні та за їх межами, і відіграє важливу роль у їхній культурі та самоідентифікації. Відсутність різноманітності комунікаційних засобів та технологічних рішень для підтримки української жестової мови ускладнює життя громадян і обмежує їхню діяльність у суспільстві.

Однак у світлі зростаючих зусиль у сфері інклюзії та розвитку технологій, відкриваються нові можливості для покращення якості життя людей, які стикаються з комунікаційними бар'єрами. Розробка та впровадження спеціалізованих систем, спрямованих на підтримку української жестової мови, може стати першочерговим завданням для забезпечення їхньої повноцінної участі в усіх сферах суспільного життя. Це включає в себе розробку інноваційних технологій, які сприяютимуть легкій і ефективній комунікації, в освіті, роботі, та інших сферах. Таким чином, **актуальність** теми даної роботи полягає у вирішенні проблеми спілкування жестовою мовою та поліпшення якості життя людей з вадами слуху і мовлення.

Метою роботи є покращення комунікації людей з вадами слуху та мовлення, шляхом розробки інтелектуальної системи перетворення української жестової мови на текст та аудіо.

Для досягнення поставленої мети необхідно виконати наступні завдання:

– виконати аналіз предметної сфери;

– дослідити на проаналізувати наявні існуючи системи розпізнавання та перекладу жестових мов;

– сформувати вимоги до функціоналу розроблювальної системи;

– обрати технології та методи розробки системи перетворення жестової мови на текст та аудіо;

– створити набір даних;

– розробити та налаштувати модель глибокого навчання для розпізнавання жестів та перетворення їх на текст;

– реалізувати синтез тексту у мовлення;

– створити інтерфейс користувача;

– протестувати систему для визначення ефективності моделі.

Об’єктом дослідження даної роботи є процес розпізнавання жестів.

Предметом дослідження є методи та алгоритми для розпізнавання жестів та розробка інтелектуальної системи розпізнавання.

Технології, що мають бути використані у роботі – мова програмування Python, середовище розробки Google Colab, Jupyter, Visual Studio Code, бібліотеки OpenCV, Keras, NumPy, TensorFlow, фреймворк MediaPipe.

Дослідження було представлено на конференції «Могилянські читання-2023» 10 листопада 2023 року, тема тез «Інтелектуальна система перетворення української жестової мови на текст та аудіо».

1 АНАЛІЗ ТА ОГЛЯД ДОСЯГНЕНЬ У СФЕРІ ПЕРЕКЛАДУ ЖЕСТОВИХ МОВ

1.1 Жестова мова як засіб комунікації

Комунікація є однією з фундаментальних складових соціального життя та міжособистісних відносин. Проте, значна частина населення мають суттєві труднощі у взаємодії та спілкуванні з іншими через вади мовлення або аудіального сприйняття. За останні роки дослідники досягли значних успіхів у розробці перекладачів жестової мови в режимі реального часу, які можуть допомогти подолати цей комунікаційний розрив [1]. Ці системи покладаються на методи комп'ютерного зору та алгоритми машинного навчання для точного розпізнавання та перекладу жестів на розмовну мову. Тим не менш, точний переклад залишається складним завданням, і дослідники продовжують працювати над підвищенням точності цих систем [2].

Вади слуху – це повна або часткова нездатність чути, полягає у неправильній передачі або сприйнятті органом слуху звуків. Часткова глухота виявляється у втраті здатності чути певні частоти або розрізняти звуки з низькою амплітудою. Повна глухота означає абсолютну втрату слуху або таке його значуще зниження, коли неможливе розбірливе сприйняття мови.

За даними Всесвітньої організації охорони здоров’я (ВООЗ, анг.World Health Organization), понад 5% населення світу – або 430 мільйонів людей – потребують реабілітації для вирішення проблеми втрати слуху (включно з 34 мільйонами дітей). За прогнозами, до 2050 року майже 2,5 мільярда людей матимуть ту чи іншу ступінь втрати слуху, і щонайменше 700 мільйонів людей потребуватимуть реабілітаційних послуг у зв’язку з втратою слуху [3]. На рис. 1.1 наведено статистику, що показує передбачувану кількість людей із порушеннями слуху в усьому світі у 2019 році та прогнози на 2030, 2040 і 2050 роки [4].

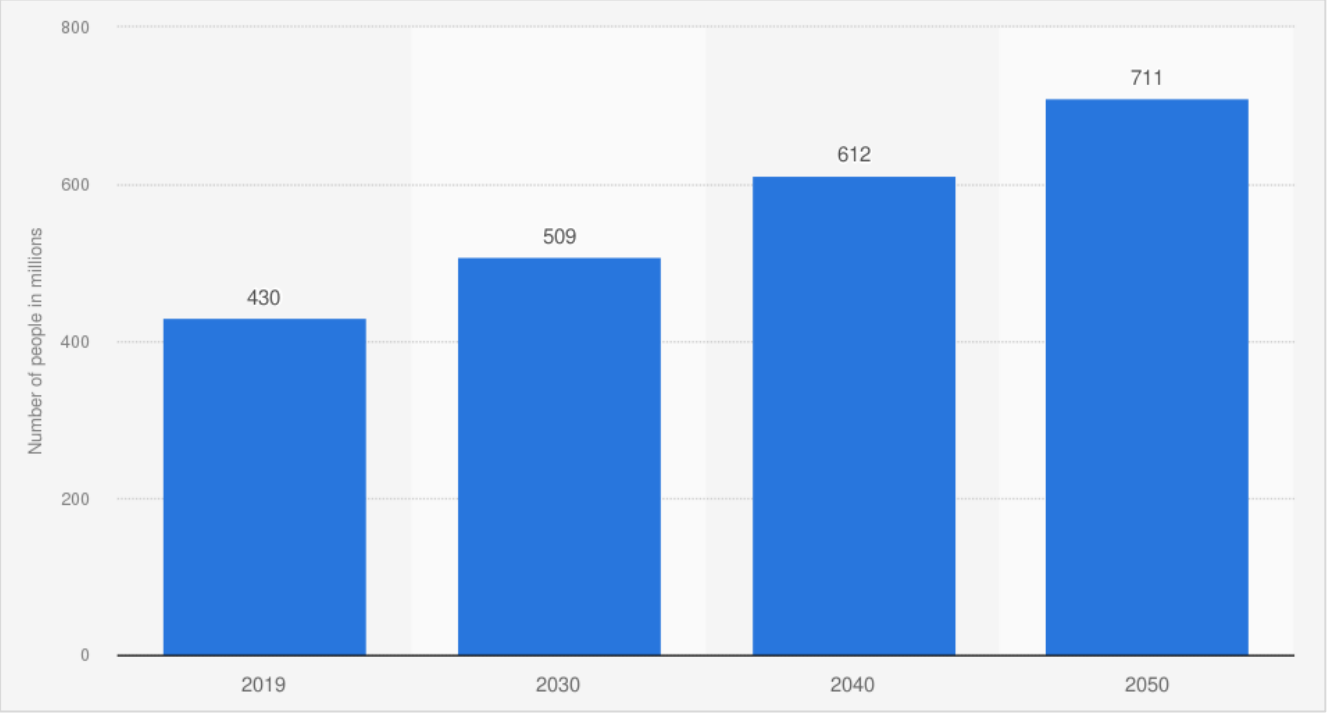

Рисунок 1.1 – Прогнозована кількість людей із порушенням слуху у світі у 2019, 2030, 2040 і 2050 роках (у мільйонах)

В Україні, де проживає близько 41 мільйона осіб, понад 2 мільйони людей стикаються з різними формами порушень слуху [3]. Ця численність не лише віддзеркалює складність проблеми, але й підкреслює необхідність вирішення питань із доступністю та рівноправністю для цієї значущої частини суспільства.

З цього числа, близько 400 000 громадян України, які проживають у різних країнах світу, спілкуються жестовою мовою. Ця мова для них стала не лише засобом спілкування, але й рідною та єдиною мовою в комунікації з навколишнім світом. Необхідність вирішення термінових потреб є постійною реальністю, з якою вони стикаються. Однак у світлі зростаючих зусиль у сфері інклюзії та розвитку технологій, відкриваються нові можливості для покращення якості життя цієї аудиторії та подолання бар'єрів, які раніше обмежували їхню активність та участь у суспільному житті.

Жестова мова – спеціальний вид мовлення, який дозволяє позначати літери алфавіту, цілі слова і вирази за допомогою конкретних жестів, що являють собою

комбінації долонь і пальців рук. Також це мовна система, яка відкриває можливості для спілкування та взаєморозуміння, незалежно від мовних бар'єрів. Вона використовується як людьми з вадами голосових зв’язок та слуху, так і людьми без таких вад. У різних країнах і культурах існують різні види жестової мови. Наприклад, американська жестова мова (англ. American Sign Language, ASL) відрізняється від британської жестової мови (англ. British Sign Language, BSL) чи жестової мови, яка використовується в Україні. Українська жестова мова походить від великої сім’ї французької жестової мови. Важливою її частиною є дактильний алфавіт – система жестів, кожен знак якої відповідає літерам українського алфавіту (рис. 1.2). В українські жестовій мові використовується ручний алфавіт із 33 знаків.

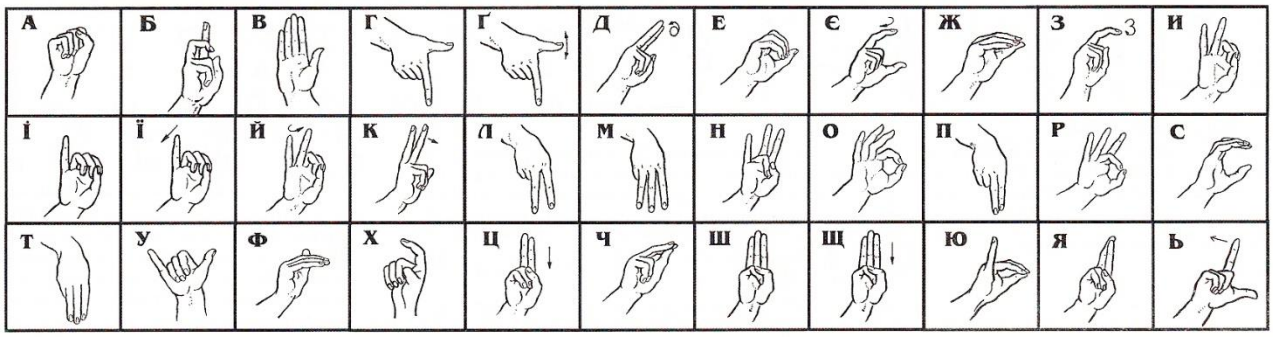

Рисунок 1.2 – Абетка української жестової мови

У світлі стрімкого розвитку інформаційних технологій та штучного інтелекту, виникають безмежні можливості в сфері розробки систем, спрямованих на поліпшення якості життя людей з вадами слуху. Однією з інноваційних напрямів в цьому контексті є розробка систем, які спроможні перекладати жестові мови у режимі реального часу, що відкриває можливість глибокої і багатопланової комунікації для частини людей, що мають вади слуху або мовлення. Ці інноваційні рішення базуються на поєднанні машинного навчання, комп'ютерного бачення та розпізнавання жестів, що дозволяє системам ефективно та точно інтерпретувати жестову мову, а потім трансформувати її в доступний текст.

Цей напрям розвитку технологій відкриває нові перспективи для інклюзивного суспільства та забезпечує можливість особам з вадами слуху активно взаємодіяти з інформаційним середовищем, вдосконалюючи якість їхнього життя та сприяючи їхній соціальній інтеграції.

1.2 Огляд існуючих систем розпізнавання та перекладу жестових мов

Системи перетворення жестової мови у текст є інноваційними технологіями, спрямованими на полегшення комунікації для людей з важкими порушеннями слуху чи мовлення. Ці системи використовують апаратне і програмне забезпечення для розпізнавання жестів та їх подальшого перекладу в текстову інформацію. Такі технології виявляються особливо корисними у ситуаціях, де звичайні засоби комунікації недоступні або обмежені.

Однією з важливих складових цих систем є алгоритми машинного навчання, які навчаються розпізнавати різні жести та їх семантичні властивості. Завдяки постійному розвитку технологій комп'ютерного зору та сенсорних платформ, такі системи стають все точнішими та чутливішими, дозволяючи їм ефективно інтерпретувати навіть тонкі відмінності в рухах. Ці інновації в сфері жестової мови сприяють інклюзивності та розширюють можливості спілкування для різних груп користувачів, роблячи їх більш доступними і самостійними.

Таким чином, на сьогоднішній день, існує чимало технологій, які створені для перетворення жестових мов на текст та мовлення. Вони різняться між собою за своїми функціональними можливостями, складністю реалізації та точністю в розпізнаванні жестів.

Система MotionSavvy UNI. Розроблений у 2012 році командою з Рочестерського технологічного інституту (Національний Технічного інституту для глухих (National Technical Institute for the Deaf) пристрій для озвучування американської жестової мови і навпаки. Команда використовувала планшет з датчиком Leap Motion. 3D-контролер Leap Motion розпізнає рухи рук, які виконує користувач над поверхнею планшета [5].

В результаті на екрані з'являється зображення жесту на екран і переводиться в текст або озвучується. Цей пристрій є двостороннім комунікаційним інструментом, який дозволяє людям з вадами слуху взаємодіяти та спілкуватися з іншими людьми з інвалідністю та без неї, які можуть не знати жестової мови [6]. Основною метою є подолання мовних бар'єрів та покращення комунікації в режимі реального часу в різних соціальних та професійних сферах. Приклад використання пристрою зображений на рис. 1.3.

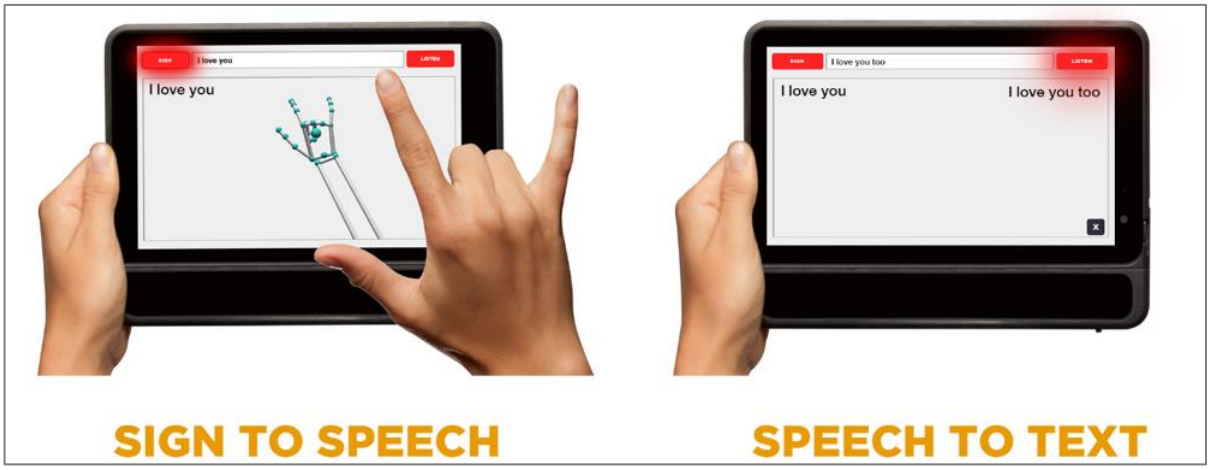

Рисунок 1.3 – Технологія MotionSavvy UNI

UNI складається з трьох частин: планшета, спеціально розробленого чохла і застосунка. Вбудований у чохол просторовий контролер Leap Motion відстежує положення рук і пальців користувача за допомогою декількох камер і датчиків. Застосунок, що функціонує на планшеті, перекладає рухи в мовлення або текст, який відображається на екрані.

Таким чином, цей пристрій в основному відстежує рухи рук людини в реальному часі. Головний екран цього пристрою складається з трьох важливих частин. По-перше, кнопка переходу від жестів до мови, яка включає базу даних жестової мови. Ця база даних розуміє жести людини і перекладає їх на розмовну англійську мову. Друга частина – це кнопка прослуховування, яка є розпізнавачем голосу. Цей розпізнавач голосу фіксує голос мовця і перетворює його на письмовий

текст. Третя частина – це візуалізатор, який відображає точні знаки рухів рук людини. UNI не тільки перекладає, а й запам'ятовує індивідуальний стиль жестів користувача, тому з часом транслює інформацію точніше.

Система має наступні переваги. Технологія розпізнавання жестової мови. MotionSavvy UNI використовує передову технологію розпізнавання жестів для точної інтерпретації та перекладу жестів на розмовну мову. Цей двосторонній переклад дозволяє людям з вадами слуху безперешкодно спілкуватися з тими, хто не розуміє мову жестів.

1. Здійснення перекладу в режимі реального часу. Однією з важливих переваг MotionSavvy UNI є його здатність здійснювати переклад у реальному часі. Ця функція має вирішальне значення для забезпечення безперешкодної та спонтанної взаємодії між користувачами жестової мови та людьми, які спілкуються вербально.

2. Висока точність розпізнавання. MotionSavvy UNI використовує передову технологію розпізнавання жестів з високою точністю, що забезпечує безперебійну взаємодію без зайвих помилок.

3. Перетворення жестової мови на текст і мовлення. Система перетворює жестову мову у зрозумілий текст та мовлення, що робить можливим сприйняття інформації для всіх учасників комунікації.

4. Портативність. Як портативний пристрій, MotionSavvy UNI дозволяє користувачам взаємодіяти з ним у будь-якому місці та в будь-який час, що підвищує його доступність.

5. Простий та зручний інтерфейс. Простота та інтуїтивно зрозумілий інтерфейс дозволяють користувачам швидко та легко оволодіти використанням системи, сприяючи ефективній комунікації.

6. Інтеграція з іншими пристроями. MotionSavvy UNI може бути легко інтегрований з іншими системами та пристроями, розширюючи можливості взаємодії і сприяючи розширенню функціоналу користувача.

Проте, значні витрати на придбання пристрою можуть ускладнити доступність цієї технології для широкого кола користувачів.

Система SignAll 1.0. Угорська компанія Dolphio Technologies створила автоматичну систему для перекладу американської жестової мови (ASL). Роботи ведуться з 2015 року. Дослідження ґрунтується на комп'ютерному зорі та обробці природної мови (NLP), використовує Kinect від Microsoft і вебкамери з датчиками глибини (датчик глибини і 3 вебкамери), під'єднані до комп'ютера. Технологія комп'ютерного зору зчитує жести і міміку глухої людини, а система обробки природної мови перетворює зібрані дані на просту фразу [7]. Розуміючи важливість руйнування бар'єрів між спільнотою глухих і суспільством загалом, Sign All 1.0 використовує передові технології для інтерпретації та перекладу жестів жестової мови в усну або письмову форму [7]. Систему має бути встановлено у фіксованому положенні (в офісі, на стійці обслуговування клієнтів, у конференц-залі тощо), де нечуюча і чуюча сторона може в будь-який час сісти і спілкуватися за допомогою екранного чату своєю рідною мовою.

На рис. 1.4 зображено використання SignAll 1.0. Ця потужна система має два монітори: один для глухого користувача, а інший для того, хто чує і говорить. Глуха людина одягає різнокольорові рукавички і показує жести перед камерою. Жестова мова переводиться в текст, який може бути прочитаний людиною, що чує. Відповідь людини, що чує, потім переводиться в текст за допомогою автоматичної системи розпізнавання мови, яку може прочитати глуха людина.

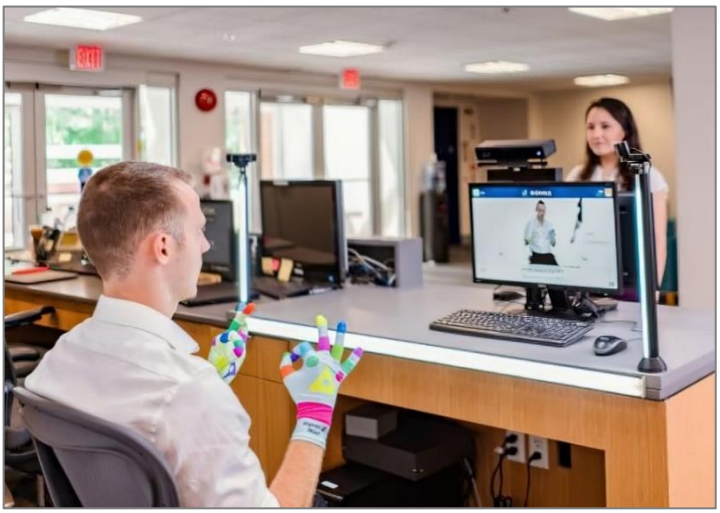

Рисунок 1.4 – Технологія SignAll 1.0

Систему SignAll можна використовувати у сфері бізнесу та освіти. Технологія підвищує доступність глухих співробітників на робочих місцях і дає змогу компаніям пропонувати краще обслуговування глухих клієнтів. Це також може допомогти друзям і сім'ям спілкуватися один з одним. Технологія SignAll 1.0 має наступні переваги.

1. Взаємодія в режимі реального часу. Система забезпечує миттєвий переклад жестів на текст, дозволяючи користувачам взаємодіяти в реальному часі без зайвих затримок.

2. Розширена комунікація. SignAll 1.0 сприяє розширеній комунікації, дозволяючи людям з вадами слуху виражати себе через мову жестів та отримувати миттєвий переклад, що збільшує їхню участь у спілкуванні.

3. Зручний інтерфейс. Система має зручний та інтуїтивно зрозумілий інтерфейс, що полегшує використання для користувачів і сприяє швидкому вивченню функціоналу.

4. Забезпечення полегшеного спілкування без необхідності використання посередника-перекладача. Система усуває потребу в посереднику-перекладачі, що дає можливість людям з вадами слуху самостійно та ефективно спілкуватися.

5. Висока точність розпізнавання жестів. Застосування технологій комп'ютерного зору та нейромереж дозволяє досягти високої точності розпізнавання жестів, забезпечуючи точний та чіткий переклад.

6. Швидкість. Робота в режимі реального часу дозволяє користувачам отримувати миттєвий переклад, що особливо важливо для людей із вадами слуху, які залежать від жестової мови.

7. Відкритий формат. Підтримка відкритого формату жестової мови дозволяє легко додавати нові жести у систему, розширюючи можливості та функціонал системи.

8. Розпізнавання жестів. Використання передової технології розпізнавання жестів дозволяє системі точно інтерпретувати нюанси рухів та виразів, що є важливим для жестової мови.

9. Кастомізація. Система розпізнає різні жестові мови, а користувачі можуть налаштовувати систему відповідно до свого діалекту жестової мови, забезпечуючи точний переклад та особисту адаптацію.

10. Спілкування в режимі реального часу. Система сприяє миттєвій комунікації та інклюзивності, надаючи можливість взаємодії в динамічному середовищі.

11. Наявність інтерфейсу чату, який відображає розмову для обох сторін, полегшує взаєморозуміння та покращує досвід користувачів.

Недоліки системи SignAll 1.0.

1. Обмеженість жестів. Технологія має обмежену кількість розпізнаваних жестів, що може обмежити можливості користувачів у вираженні себе. Це може означати, що інші системи можуть бути більш гнучкими і здатними розпізнавати більшу кількість жестів та виразів, що робить їх більш адаптивними до різних стилів комунікації.

2. Висока вартість. Система SignAll 1.0 характеризується високою вартістю, що може стати значущим обмеженням для доступу до цієї технології для багатьох користувачів та організацій.

3. Неможливість навчання новим жестам. Програма не може навчатися новим жестам та оновлювати свої алгоритми розпізнавання жестів. Це обмеження може вплинути на актуальність системи та її здатність адаптуватися до змін у мові жестів.

4. Залежність від камери. Система вимагає використання камери для розпізнавання жестів, що може обмежувати її функціональність в різних ситуаціях, таких як темні приміщення чи відкрите повітря. Інші системи можуть використовувати додаткові датчики для поліпшення точності та універсальності.

5. Необхідність встановлення спеціальних застосунків. Використання SignAll 1.0 передбачає встановлення спеціальних програм на комп'ютер чи мобільний пристрій користувача. Це може стати додатковим бар'єром для тих, хто не бажає або не може встановлювати застосунки.

Сервіс «ConnectPro». Програмне забезпечення, яке було випущено українськими розробниками, надає цілодобовий доступ до послуг сурдоперекладачів. Connect PRO дозволяє працювати у двох режимах: викликати перекладача жестової мови та автоматично конвертувати мовлення в текст та навпаки. Це особливо корисно для осіб з різними рівнями порушення слуху, які не орієнтуються в жестовій мові [8].

У застосунку реалізована можливість групового перекладу, дозволяючи одночасно взаємодіяти з перекладачем жестової мови на різних пристроях. Це особливо актуально під час організації групових заходів, навчальних сесій, конференцій та інших подій. Програмне забезпечення встановлюється на пристрій, що доступний для осіб із порушенням слуху. Для початку взаємодії з відвідувачем, який не чує, потрібно активувати жестового перекладача за допомогою опції «Виклик перекладача» у меню застосунку. Камера пристрою наводиться на відвідувача, який використовує жестову мову для спілкування з перекладачем. Усе, що говорить відвідувач, перекладач озвучує голосом, а відповіді перекладача транслюються у реальному часі за допомогою жестової мови для нечуючого. Програмне забезпечення має наступні переваги.

1. Взаємодія в режимі реального часу. Застосунок «ConnectPro» надає можливість взаємодії в режимі реального часу, що сприяє швидкому та ефективному спілкуванню між користувачами та сурдоперекладачами.

2. Професійні перекладачі зі знанням жестової мови. Система забезпечує доступ до професійних перекладачів, які володіють жестовою мовою, що гарантує якісні та точні переклади для осіб із порушеннями слуху.

3. Швидке підключення та цілодобова доступність сурдоперекладачів. ConnectPro відрізняється швидким підключенням і готовністю сурдоперекладачів протягом усіх годин доби, що забезпечує надійну та постійну доступність послуг.

4. Одним із основних недоліків є відсутність функції розпізнавання та конвертації жестів без участі сурдоперекладача. Це обмеження може призвести до залежності від наявності професіонала для здійснення комунікації.

Сервіс «ConnectWeb». Український сервіс для вебресурсів, що сприяє максимально зручній взаємодії з різними сайтами для відвідувачів із обмеженими можливостями, які користуються жестовою мовою. Для нечуючих людей, це рішення допоможе мати вільний доступ до сервісів у віртуальному просторі [8].

Алгоритм роботи з даним сервісом такий. Вебвіджет встановлюється на сайт клієнта. Використовуючи сервіс, користувач може викликати сурдоперекладача, який протягом хвилини приєднається до діалогу. Спеціаліст допоможе адаптувати текстовий контент сайту на жестову мову. Також нечуючий клієнт має можливість подзвонити до організації (телефонує сурдоперекладач, що представляє інтереси клієнта). Окрім цього, є послуга адаптації контенту на жестову мову для вебсайтів організацій та установ. Створюються спеціальні відеоматеріали, де сурдоперекладач, використовуючи жести, пояснює суть контенту. У разі потреби сурдоперекладач здійснює телефонний дзвінок від імені людини із вадами слуху, аби вирішити необхідні питання та надати офіційну відповідь.

Можна виділити наступні переваги сервісу «ConnectWeb».

1. Забезпечення безперешкодної онлайн-комунікації для осіб з вадами слуху. Connect WEB надає ефективну та доступну онлайн-комунікацію для осіб із

вадами слуху, дозволяючи їм спілкуватися з вебресурсами за допомогою жестової мови.

2. Можливість віддаленого зв’язку через сайт. Сервіс дозволяє взаємодіяти з вебресурсами без особистого візиту до магазинів чи офісів, забезпечуючи зручність та доступність віддалено.

3. Професійні перекладачі зі знанням жестової мови. Наявність професійних перекладачів, які володіють жестовою мовою, гарантує якісний та точний переклад контенту для нечуючих користувачів.

4. Швидке підключення сурдоперекладача до діалогу через відеозв’язок. Connect WEB забезпечує швидке підключення сурдоперекладача до діалогу, забезпечуючи оперативну взаємодію з вебресурсом через відеозв'язок.

5. Перетворення на жестову мову текстового контенту сайту. Спеціалізовані перекладачі адаптують текстовий контент вебсайту на жестову мову, що полегшує розуміння і сприяє включенню аудиторії із вадами слуху.

Однак, сервіс не забезпечує можливості прямої комунікації між нечуючим відвідувачем сайту та організацією чи установою без участі перекладача, що може бути обмеженням у спрощенні взаємодії та розумінні потреб користувачів.

Сервіс «Sign Language Translator». Даний сервіс надає цілодобовий доступ до сурдоперекладачів. Застосунок має інтегровану функцію здійснювати безголосові виклики служб екстреної допомоги. У випадку потреби нечуючі особо можуть запросити переклад документів на жестову мову, надіславши фотографію документа. Потім буде отримано відеопереклад на мові жестів у відповідь [8]. Сервіс має наступні переваги.

1. Цілодобовий швидкий доступ до кваліфікованих перекладачів жестової мови у будь-який час, що сприяє вирішенню комунікативних завдань користувачів.

2. Забезпечення безголосових викликів екстрених служб. Інтегрована можливість безголосових викликів служб екстреної допомоги надає нечуючим користувачам швидку і ефективну можливість звертатися за допомогою в невідкладних ситуаціях.

3. Можливість здійснювати телефонні дзвінки сурдоперекладачами від імені нечуючих користувачів. Клієнти можуть ініціювати телефонні дзвінки через сурдоперекладачів, щоб ефективно спілкуватися в реальному часі та вирішувати різноманітні справи.

Таким чином, як і програмне забезпечення «ConnectPro» та «ConnectWeb», сервіс «Sign Language Translator» не має функції розпізнавання жестів та перетворення їх на текст, що унеможливлює пряму комунікацію між нечуючим клієнтом та організацією без виклику сурдоперекладача, що може викликати деякі обмеження у взаємодії та обміні інформацією.

Порівняння оглянутих рішень. Описані системи порівняно за такими критеріями, як тип системи, функціонал, точність перекладу, переклад у реальному часі, портативність, вартість (табл.1.1). З урахуванням усіх критеріїв, вибір системи буде залежати від пріоритетів та конкретних потреб користувача. Якщо важлива портативність, висока точність перекладу та незалежність від наявності посередника-перекладача, то кращим вибором можуть стати MotionSavvy UNI.

Таблиця 1.1 – Порівняння оглянутих рішень

Критерій / Рішення	Тип системи	Функціонал	Точність перекладу	Переклад у реальному часі	Портативність	Вартість
Motion-Savvy UNI	Пристрій	Перетворення жестів на текст і голос	Висока	Так	Портативний	$200 + $20/щомісячна абонентська плата

Закінчення таблиці 1.1

SignAll 1.0	Пристрій	Перетворення жестів на текст і голос	Висока	Так	Фіксована установка	Від 1300 до понад 5000 євро
Connect-Pro	Мобільний застосунок	Доступ до перекладачів, конвертування мовлення в текст	Залежить від сурдоперекладача	Так	Портативний	Безкоштовно
Connect-Web	Вебвіджет	Онлайн-зв’язок за допомогою перекладачів	Залежить від сурдоперекладача	Так	Портативний	Безкоштовно
Sign Language Translator	Мобільний застосунок	Цілодобовий доступ до перекладачів	Залежить від сурдоперекладача	Так	Портативний	Безкоштовно

1.3 Вимоги до розроблювальної системи та постановка задачі

У результаті огляду та аналізу існуючих рішень було виявлено, що системи мають як переваги, так і недоліки. Аналіз наявних аналогів підкреслив різноманітність та важливість таких систем у полегшенні комунікації.

Однак існує потреба в розробці більшої кількості систем, які б слугували покращенню життя людей з вадами слуху. Такі системи повинні бути спрямовані на розширення можливостей комунікації, забезпечуючи надійний та ефективний зв'язок з оточуючим світом. Розробка нових інноваційних рішень у галузі перетворення жестової мови на текст та аудіо має на меті вирішення термінових потреб цієї аудиторії, сприяючи їхній активній участі у суспільному житті та розвитку. Тому метою кваліфікаційної роботи магістра є покращення комунікації людей з вадами слуху, шляхом розробки інтелектуальної системи перетворення української жестової мови на текст та аудіо з використанням штучного інтелекту.

Список вимог до функціоналу розроблювальної системи.

1. Реалізація розпізнавання жестів та перетворення їх на текст. Повинна

бути забезпечена точність розпізнавання для забезпечення високоякісного текстового виводу.

2. Здійснення перекладу в режимі реального часу. Система повинна бути здатна надавати переклад тексту, отриманого з жестів, майже миттєво.

3. Після перетворення жестів на текст, система має використовувати технології синтезу мовлення для конвертації цього тексту у звуковий формат аудіо. Це включає в себе вибір голосу та синтезування звуку. Можуть використовуватися технології синтезу мовлення, такі як текст-у-мовлення (англ. Text-to-Speech, TTS) системи. Система має забезпечувати натуральне та зрозуміле звучання синтезованого голосу.

4. Українська версія користувацького інтерфейсу. Весь інтерфейс системи повинен бути доступним українською мовою.

5. Простий та зручний інтерфейс. Забезпечення інтуїтивно зрозумілого інтерфейсу для користувачів будь-якого рівня досвіду. Мінімізація кількості кроків для виконання основних завдань. Забезпечення зручності взаємодії з системою, враховуючи можливості жестової мови.

Висновки до розділу 1

У першому розділі було проаналізовано проблематику області дослідження розпізнавання жестових мов та методи її вирішення. Аналіз предметної сфери вказав на важливість комунікації людей із вадами слуху, яка є важливою частиною соціального життя та міжособистісних відносин. Розглянуто та проаналізовано існуючі технології, що полегшують життя нечуючих та слабочуючих осіб, зокрема системи MotionSavvy UNI, SignAll 1.0, ConnectPro, ConnectWeb та Sign Language Translator. Для кожної системи наведено її переваги та недоліки, що дозволяє виявити прогалини та перспективи технологій.

Було сформовано постановку задачі для кваліфікаційної роботи магістра, яка полягає в розробці інтелектуальної системи для поліпшення спілкування людей із вадами слуху. Визначено вимоги до функціоналу системи, серед яких реалізація

розпізнавання жестів, перекладу в режимі реального часу та синтезу тексту у мовлення. Розвиток інноваційних систем у галузі перетворення жестової мови на текст та аудіо є актуальним та потребує подальших досліджень та розробок. Розробка таких систем може значно покращити якість життя людей з вадами слуху, забезпечуючи їм більш активну участь у суспільному житті та сприяючи інклюзивності.

2 МЕТОДИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ РОЗПІЗНАВАННЯ ЖЕСТІВ

2.1 Огляд алгоритмів машинного та глибинного навчання для вирішення задачі розпізнавання жестів

Існує багато різноманітних методів задля вирішення задачі розпізнавання жестів, які охоплюють класичні алгоритми машинного навчання, такі як метод опорних векторів (англ. Support Vector Machine, SVM), алгоритм k-найближчих сусідів (англ. K-Nearest Neighbors, KNN), метод логістичної регресії (англ. Logistic regression, LR), дерева рішень (англ. Decision Trees, DT), випадковий ліс (англ. Random Forest, RF), а також нейронні мережі, зокрема, згорткові (англ. Convolutional Neural Networks, CNN) та рекурентні (англ. Recurrent Neural Networks, RNN). Розглянемо перелічені методи детальніше.

Метод опорних векторів. У науковій праці [9] розробники представили спрощену методологію розпізнавання жестової мови, використовуючи фреймворк MediaPipe і Support Vector Machine. Їхня модель досягла середньої точності 99 % у кількох наборах даних, забезпечуючи точне виявлення в режимі реального часу без використання датчиків. Даний підхід став легким і економічним рішенням, що перевершує складні методи, які вимагають великих обчислювальних витрат. У іншій статті [10] дослідники продемонстрували ефективність цього підходу, розробивши модель розпізнавання цифр американської жестової мови із точністю 99,9%. Набір даних складався з 30 000 спостережень, розділених на десять класів.

SVM вибирає екстремальні вектори і точки, які допомагають у створенні гіперплощини. Опорні вектори, які використовуються для представлення цих крайніх випадків, складають основу методу SVM. Мета алгоритму SVM – визначити оптимальну межу прийняття рішення або лінію, яка може розділити n-вимірний простір на класи, щоб у майбутньому можна було швидко класифікувати нові точки даних. На рис. 2.1 показано графічне представлення алгоритму Support Vector Machine.

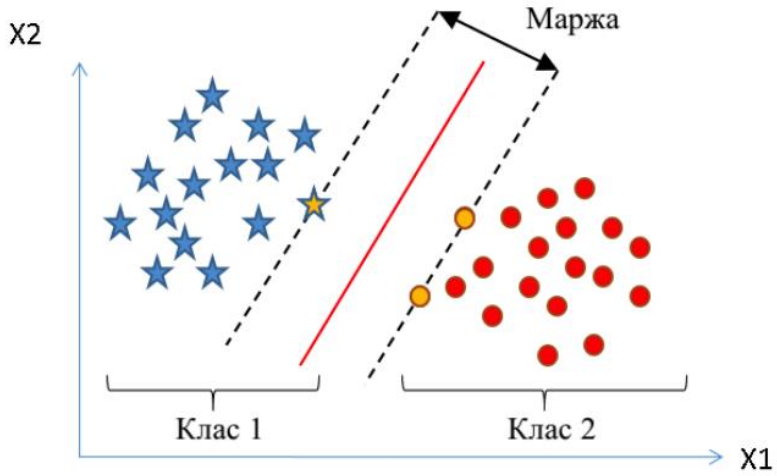

Рисунок 2.1 – Графічне зображення алгоритму SVM

Для розпізнавання жестів рук за допомогою SVM у процесі обробки спочатку знімається рух руки за допомогою вебкамери. Для ідентифікації руки на етапі попередньої обробки будується обмежувальна рамка, щоб відокремити передній план від фону. Для цього використовується різниця між послідовними кадрами. Для прискорення обчислень розмір захопленого зображення змінюється. Передній край зменшеного зображення використовується для визначення країв за методом Канні. Потім характеристики зображення витягуються з гістограми на основі градієнта (HOG). Отримані характеристики зберігаються в базі даних. SVM порівнює найбільш підходящі зображення і виводить результат на екран, якщо в базі даних є характеристики, що збігаються.

Метод k-найближчих сусідів. Для розпізнавання жестів використовується алгоритм K-Nearest Neighbor. Наприклад, в дослідженні [11] було розроблено застосунок для вивчення студентами американської жестової мови через гру. Застосунок використовує метод KNN для класифікації і досягає точності 99,44% [12]. У іншому дослідженні [13] розробники зосередились на ідентифікації мови жестів за допомогою методів машинного навчання. Об'єктом їхнього дослідження стали статичні рухи рук, що відповідають цифрам від 0 до 9. Використовуючи цифровий RGB-датчик для отримання зображень знаків, дослідники створили набір даних, що складається з 500 фотографій, по одному зображенню на кожну

цифру. Вони навчили моделі, використовуючи підходи контрольованого навчання наївного баєсівого класифікатора. та k-найближчих сусідів, і отримали середні показники точності 98,36 % і 97,79 % відповідно, причому метод k-найближчих сусідів трохи перевершив наївного баєсівого класифікатора.

Таким чином, k-найближчих сусідів використовується для класифікації, за якої класифікація точки даних визначається тим, як класифіковані її сусіди. Для пошуку найближчого сусіда в KNN використовується евклідова відстань [14]. При використанні k-найближчих сусідів класифікація виконується з використанням порогового значення, яке обчислюється як середнє значення k найближчих точок даних. Продуктивність повністю базується на відстані до найближчого сусіда, вимірюванні схожості та пороговому значенні [15]. Роботу KNN можна пояснити на основі такого алгоритму.

Етап 1. Вибирається число k сусідів.

Етап 2. Обчислюється евклідова відстань між k сусідами.

Етап 3. Береться k-найближчих сусідів відповідно до розрахованої евклідової відстані.

Етап 4. Серед цих k сусідів підраховується кількість точок даних у кожній категорії.

Етап 5. Нові точки даних відносяться до тієї категорії, для якої кількість сусідів максимальна.

Для вилучення характеристик і визначення місця розташування області руки метод k-найближчих сусідів вимагає введення параметрів. Для цього система KNN підбирає чотири параметри: співвідношення вертикальних і горизонтальних ліній, положення прямокутника, що містить середню точку та охоплює форму руки, а також параметр, який надає детальну інформацію про горизонтальний розподіл навколо середнього значення рядків у всій ділянці руки, і стандартне відхилення. Модель KNN працює, коли в полі зору камери успішно з'являється людина з однією рукою, система правильно її виявляє і потім ідентифікує жест.

Метод логістичної регресії. Логістична регресія охоплює методи збору регресії та пов'язана із зображенням зв'язку між інформативними змінними та дискретною змінною реакції. Відмінність між стандартною прямою регресією та логістичною регресією полягає в тому, що в прямій регресії змінна реакції Y є безперервною. У логістичній регресії змінна реакції є дискретною [16]. Ця відмінність проявляється при виборі меж і підозр. Робота логістичної регресії заснована на ймовірності, і для передбачення значень, що лежать між 0 і 1, використовується лінійне рівняння, через що вона відома як алгоритм прогностичного аналізу. Він використовує сигмоїдальну функцію активації, щоб результати можна було перетворити на категоріальні значення. Цю сигмоїдальну функцію також можна назвати логістичною. Її можна подати таким чином:

$$f(y) = \frac{1}{1+e^{-(y)}}, \tag{2.1}$$

де $-\infty \leq f(y) \leq +\infty$.

Суть логістичної регресії базується на ідеї “шансів” події, що визначає ймовірність її виникнення, поділену на ймовірність випадку, якщо подія не відбудеться. Так само, як у лінійній регресії, у логістичній регресії використовуються ваги, пов'язані із розмірами вхідних даних. Взаємозв'язок між ваговими коефіцієнтами та результатом моделі виражається експоненційно. Дана модель класифікації проста в реалізації та може досягати високої продуктивності у визначенні лінійно розділених класів.

Метод Decision Trees. Дерева будуються шляхом розбиття навчального набору на окремі вузли, починаючи з кореневого вузла (рис. 2.2). Один вузол дерева рішень містить всі або більшість змінних набору даних. Цей тип моделі будується з використанням рекурсивного розбиття для класифікації даних; коли члени набору даних розбиваються на підкласи на основі певних факторів, розбиті підкласи можуть бути далі розбиті, поки не буде досягнута задана глибина.

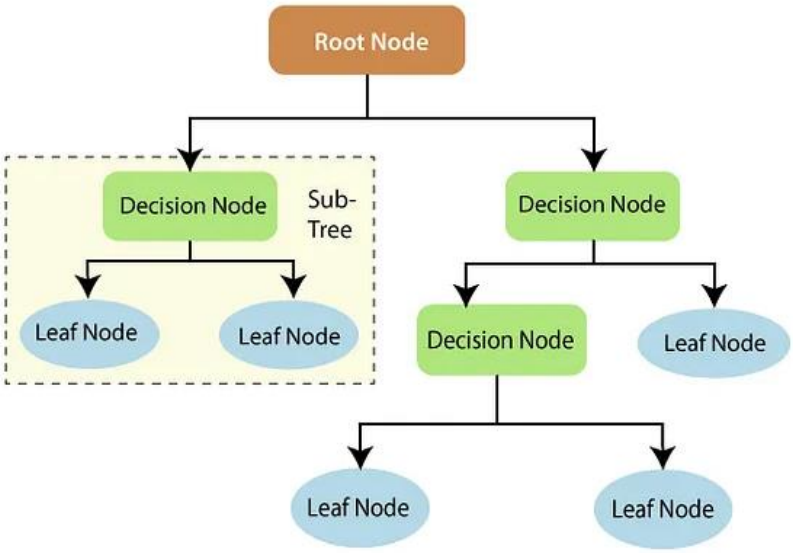

Рисунок 2.2 – Структура Decision Tree

Дерева рішень складаються з трьох основних елементів.

1. У найвищій позиції знаходиться перший компонент дерева рішень, відомий як кореневий вузол (англ. Root Node). Кореневий вузол також називають батьківським вузлом. Це вузол, з якого починається розбиття дерева.

2. Вузол рішення (англ. Decision Node) також називається дочірнім вузлом, де приймаються рішення (або) правила.

3. Листові вузли (англ. Leaf Node), які також називаються термінальним, є кінцевими вузлами і не можуть розділятися далі. Під час виклику дерева рішень для конкретних даних, листові вузли вказують, до якої конкретної категорії належать ці дані або надають передбачене числове значення. Одна з особливостей листового вузла полягає в його легкості розпізнавання, оскільки він не відгалужується на нові гілки і є кінцевою точкою рішення для даного шляху в дереві рішень.

Використання дерева рішень для жестів рук дає низку переваг, включно з масштабованістю, швидким навчанням і тестуванням, а також інтерпретованістю. Починаючи з кореня дерева та проходячи через нього до листа, дерево рішень можна використовувати для категоризації випадку. Результат тесту для кожного

нелистового вузла рішення встановлюється, а потім фокус перемикається на корінь піддерева, що відповідає цьому результату. Передбачається, що клас випадку буде тим, котрий записано на листі, коли цей метод зрештою призведе до створення листа. Для розпізнавання жестів рук кожен позначений екземпляр сегментується за допомогою сегментації з ковзними вікнами, що перекриваються. Для кожного типу вхідних даних у сегменті буде створено окремий варіант форми-кандидата. Потім метод дерева рішень обчислює відстань між кожним потенційним об'єктом форми та кожним входженням у набір навчальних даних. Найменша відстань між кандидатом та екземпляром буде використовуватись для вимірювання того, наскільки вони схожі. Потім використовується матриця відстаней зберігання значень відстаней. Немаркований екземпляр даних класифікується за допомогою вбудованого дерева рішень на основі форм.

Пробігшись по листю моделі класифікації дерева рішень, вона зупиняється, щойно задовольняється один із класів. Процедура така сама, як і під час побудови матриці відстаней: досліджуваний екземпляр спершу сегментується, щоб визначити його ідеальну відповідність місцю розташування. Відповідно до розташування листя в моделі дерева відстань кожного сегмента вимірюється за допомогою відповідної форми let. Найменша відстань використовується для визначення того, який жест руки був зафіксований камерою або будь-яким іншим пристроєм введення.

Метод Random Forest. Алгоритми випадкового лісу мають три основні гіперпараметри, які необхідно задати перед навчанням [17]. До них належать розмір вузла, кількість дерев і кількість вибірок ознак. Після цього класифікатор випадкового лісу можна використовувати для вирішення завдань регресії або класифікації. Алгоритм випадкового лісу складається з набору дерев рішень, і кожне дерево в ансамблі складається з вибірки даних, узятої з навчального набору із заміною, яка називається бутстреп-вибіркою. На рис. 2.3 показано графічне представлення даного методу.

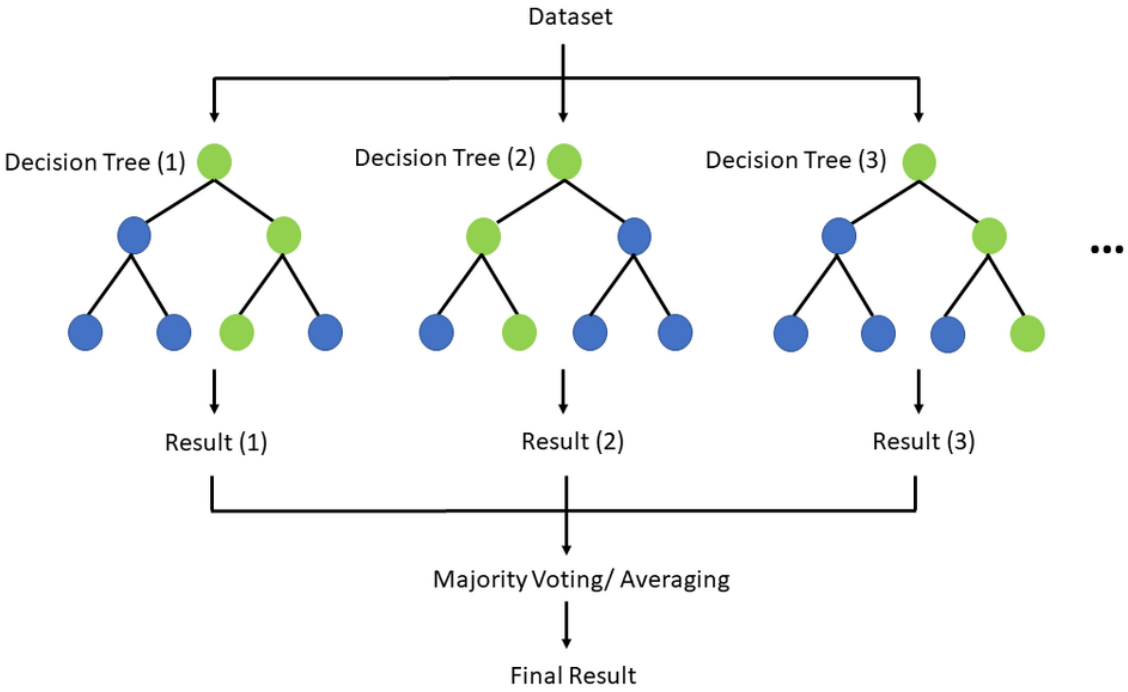

Рисунок 2.3 – Візуалізація алгоритму випадкового лісу

З цієї навчальної вибірки одна третина відкладається як тестові дані, відомі як вибірка out-of-bag. Потім за допомогою об'єднання ознак вводиться ще один приклад випадковості, що додає більше різноманітності набору даних і зменшує кореляцію між деревами рішень. Залежно від типу проблеми визначення прогнозу буде відрізнятися. Для задачі регресії окремі дерева рішень усереднюють, а для задачі класифікації прогнозований клас визначають більшістю голосів, тобто категоріальною змінною, що найчастіше зустрічається. Нарешті, вибірку out-of-bag використовують для крос-валідації, що остаточно підтверджує передбачення.

Методи випадкових лісів використовуються через такі переваги: вони добре працюють на великих наборах даних, у них немає проблеми перебору, змінні можна використовувати як числові, так і категоріальні, їх можна легко використовувати в багатокласовому середовищі, а також вони вимагають менше параметрів порівняно з іншими сучасними методами.

Згорткові нейронні мережі. Однією з найбільш вражаючих форм архітектури штучних нейронних мереж (англ. Artificial Neural Networks, ANN) є CNN, яку в основному застосовують для розв'язання складних завдань

розпізнавання образів на основі зображень. У науковій праці [18] автори застосовують згорткові нейронні мережі шляхом розробки методу класифікації зображень для американської мови жестів. Дослідники отримали 82,5% точності для жестів алфавіту і 97% точності валідаційного набору для цифр. У дослідженні [19] розробники створили систему SLR на основі глибокого навчання, використовуючи оброблені статичні зображення жестів американської жестової мови. Вони досягли середньої точності понад 90 %, навчивши Inception V3 CNN на наборі даних із 24 класів, що представляють дактилеми від A до Z, за винятком J, при цьому найкраща точність перевірки досягла 98 %. Отримавши набори даних правильно обрізаних зображень, дослідники визначили, що модель Inception V3 достатня для визначення статичної мови жестів.

CNN – це клас штучних нейронних мереж, які здебільшого використовують у сфері комп'ютерного зору, особливо для завдань, пов'язаних із розпізнаванням та аналізом зображень [20]. Також це потужний інструмент штучного інтелекту у сфері паттернів класифікації [21]. Згорткові нейронні мережі призначені для автоматичного та адаптивного вивчення закономірностей і особливостей зображень, що робить їх високоефективними для різних завдань обробки візуальних даних.

Згорткові мережі являють собою спеціалізований тип ANN з контрольованим навчанням, які обробляють свої шари, емулюючи зорову кору людського ока. Ця процедура дає змогу розпізнавати характерні патерни у вхідних даних, що дає можливість ідентифікувати об'єкти через набір прихованих шарів, які мають ієрархію і є спеціалізованими.

Це один з найбільш широко використовуваних методів, який забезпечує точну ідентифікацію жестів [22]. Хоча CNN вимагають більше обчислювальних ресурсів і даних, проте вони здатні ефективно обробляти складні і різноманітні завдання розпізнавання жестів з високою точністю і надійністю. Вони складаються з трьох шарів: перший шар згорткових фільтрів відповідає за виділення ознак зображення, пулінговий шар зменшує розміри зображення для прискорення

обчислень, а повнозв'язний шар відповідає за класифікацію результатів. Відмінною особливістю згорткових нейронних мереж є розташування нейронів у трьох вимірах: вони розподілені в ширину, глибину і висоту, що відрізняє їх від простих нейронних мереж. На рис. 2.4 представлена типова архітектура CNN [23].

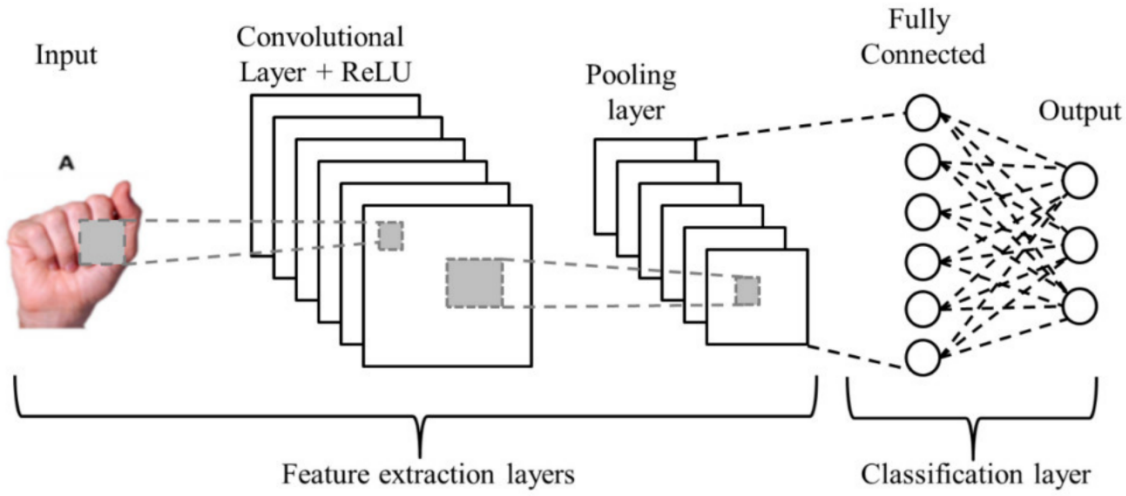

Рисунок 2.4 – Приклад архітектури згорткової нейронної мережі

Рекурентні нейронні мережі. Рекурентна нейронна мережа (англ. Recurrent Neural Network, RNN) – це один з алгоритмів глибокого навчання, який має потужні можливості в обробці послідовних даних. У традиційних моделях нейронних мереж дані зазвичай вводяться з вхідного шару, проходять через кілька прихованих шарів і, нарешті, виводяться з вихідного шару. Кожен шар повністю з'єднаний між собою, немає ніяких зв'язків всередині шару, і вони відносно незалежні один від одного [24]. Однак у рекурентній нейронній мережі всі вхідні значення в поточний момент часу 't' пов'язані з частковим станом попереднього часового кроку 't-1'. Стан мережі на попередньому часовому кроці впливатиме на стан мережі на наступному часовому кроці, що дозволяє мережі ефективно навчатися, враховуючи дані з попереднього часового кроку. Розгорнута схема рекурентної нейронної мережі [24] показана на рис. 2.5.

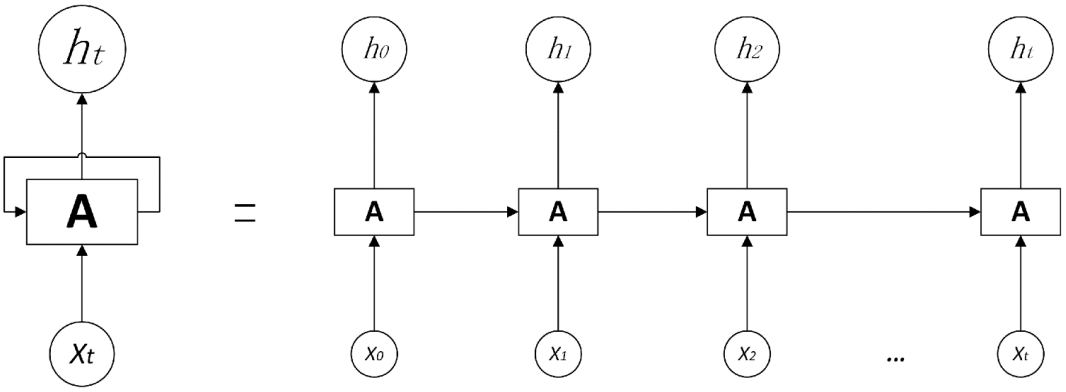

Рисунок 2.5 – Розгорнута схема рекурентної нейронної мережі

RNN страждають від короткочасної пам'яті, спричиненої проблемою градієнта, що зникає, яка переважає в нейронних мережах, що використовують методи навчання на основі градієнта і зворотне поширення. Для вирішення цієї проблеми було створено типи спеціалізованих рекурентних мереж, одним із яких є мережа довгострокової короткочасної пам’яті. Наприклад, створення моделі рекурентних нейронних мереж типу LSTM розглядається в статті [25], розробники реалізували систему класифікації жестів рук у режимі реального часу на основі попередньо оброблених сигналів електроміографії. Модель досягла точності до 99% на етапах навчання і перевірки та точності 87 ± 7% під час тестування в реальному часі [25]. У науковій роботі [26] дослідники представили заснований на комп’ютерному зорі підхід до розпізнавання алфавітів ASL з використанням фреймворку MediaPipe. Їхня система досягла точності 99 % при розпізнаванні 26 алфавітів ASL за допомогою розпізнавання жестів рук з використанням LSTM.

Довгострокова короткочасна пам’ять (англ. Long Short Term Memory, LSTM) – це популярна архітектура ШНМ, яку представили Зепп Гохрайтер і Юрген Шмідхубер як рішення проблеми зникаючого градієнта. У своїй статті [27] вони працюють над вирішенням проблеми довготривалих залежностей. LSTM використовуються у вивченні, обробці та класифікації послідовних даних, оскільки ці мережі здатні взаємодіяти з довгостроковими залежностями між часовими кроками даних.

Механізм LSTM (рис. 2.6) дозволяє рекурентній нейронній мережі зберігати вхідні дані протягом тривалого періоду часу. Це працює завдяки тому, що LSTM зберігають інформацію у своїй внутрішній пам'яті, подібно до пам'яті комп'ютера. LSTM можуть зчитувати, записувати та видаляти інформацію, розглядаючи цю пам'ять як закриту комірку, яка вирішує, чи тримати, чи видаляти дані в залежності від їх важливості. Процес присвоєння важливості відбувається через вивчені алгоритмом вагові коефіцієнти.

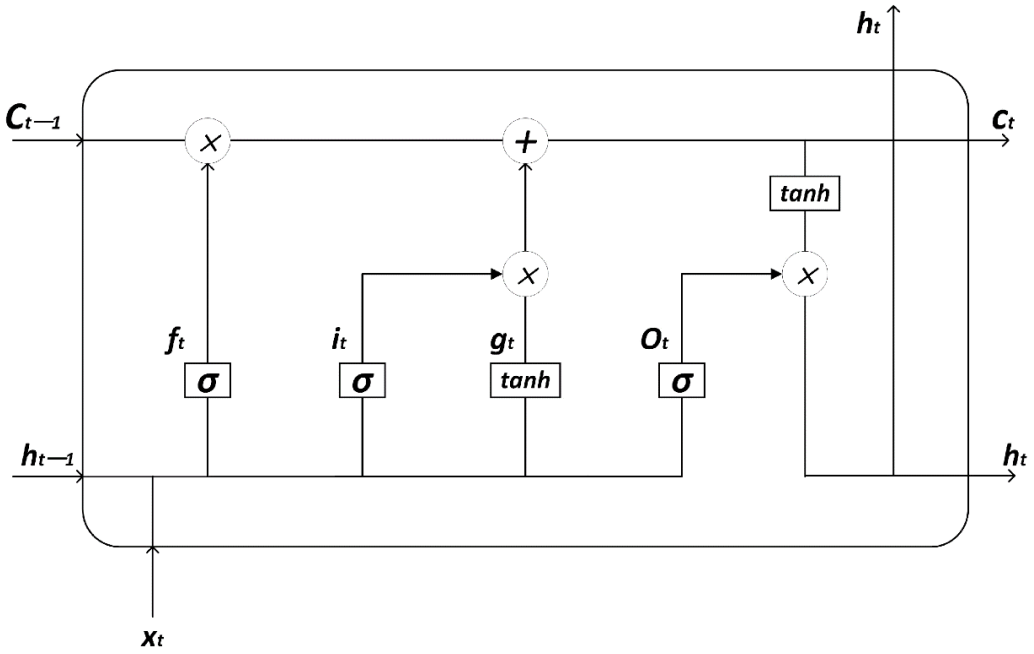

Рисунок 2.6 – Внутрішня структура нейронної мережі LSTM

Таким чином, головна особливість LSTM полягає в тому, що вони володіють здатністю запам'ятовувати та зберігати інформацію протягом тривалого періоду часу. Це досягається за рахунок використання спеціальних структур, включаючи гейти, які контролюють потік інформації в мережі. Внутрішня структура нейрона LSTM складається з чотирьох частин: вхідного гейта (англ. input gate), вихідного гейта (англ. output gate), гейта забуття (англ. forget gate) і блоку пам'яті (англ. memory unit). Гейт забуття представлений шаром sigmoid, у той час як два інших шари tanh відповідають входу і виходу блоку пам'яті [28].

Нейронна мережа LSTM має здатність видаляти або додавати інформацію в стан комірки (англ. $cell$), який контролюється і регулюється структурою gate. Структура gate складається з шару нейронної мережі sigmoid і операції точкового множення і може вибірково передавати інформацію. Вихід sigmoid шару варіюється від 0 до 1, щоб описати, скільки контенту пропускає кожен компонент. Значення 0 означає заборону на проходження вмісту, а значення 1 – дозвіл на проходження всього. Структура нейронної мережі LSTM має три таких гейти для захисту та контролю стану блоку [29]. Функції трьох гейтів наступні.

Гейт забуття (f_t): визначає, яка частина інформації в пам'яті повинна бути забута. Іншими словами, використовується для визначення інформації, відкинутої зі стану вузла. Це визначення виконується шаром $sigmoid$, який вирішує, пройти чи частково пройти, ґрунтуючись на виході попереднього часового кроку. Кожен номер виходу шару стану комірки $c_{t\text{-}1}$ між 0 і 1 визначається шляхом введення $h_{t\text{-}1}$ і x_t . Розрахунок наведено в рівнянні (2.2):

$$f_t = \sigma\big(W_f \times [h_{t-1}, x_t] + b_f\big), \tag{2.2}$$

де f_t – вихід гейту забуття для поточного часового кроку t. Цей вихід визначає, яка частина попередньої пам'яті буде забута;

σ – функція активації (зазвичай сигмоїда), вона приводить вихід гейту до значень між 0 та 1, де значення близьке до 1 вказує на те, що інформація повинна залишитися, а значення близьке до 0 вказує на те, що інформацію слід забути;

W_f – ваговий вектор для гейту забуття; це ваги, які модель вчиться під час тренування для оптимального визначення, яку частину попередньої пам'яті слід забути;

$[h_{t-1}, x_t]$ – конкатеновані вектори попереднього прихованого стану h_{t-1} та поточного вхідного вектору x_t; це створює вхід для гейту забуття, що об'єднує інформацію з попереднього стану та поточного вхідного сигналу;

b_f – зміщення (bias) для гейту забуття. Це є параметр, який модель також вчиться під час тренування.

Вхідний гейт (i_t): визначає нову інформацію, що зберігається в стані комірки, і відповідає за введення в поточній позиції послідовності. На першому етапі шар $sigmoid$, званий шаром вхідних воріт, вирішує, які значення потрібно оновити. Водночас шар $tanh$ створює новий вектор значень-кандидатів $\tilde{c}_t$ і додає його в стан комірки. Потім ці два компоненти об'єднуються для оновлення стану комірки. Розрахунки показані в рівняннях (2.3) і (2.4):

$$i_t = \sigma(W_i \times [h_{t-1}, x_t] + b_i), \tag{2.3}$$

де i_t – вихід вхідного гейту.

Інші символи мають ті ж значення, що і в формулі для гейту забуття.

$$\tilde{c}_t = tanh(W_c \times [h_{t-1}, x_t] + b_c), \tag{2.4}$$

де $\tilde{c}_t$ – вихід для оновлення пам’яті;

$tanh$ – гіперболічний тангенс.

Інші символи мають ті ж значення, що і в формулі для вхідного гейту.

Далі відбувається оновлення старого стану комірки c_{t-1} до нового стану комірки c_t. Це робиться шляхом перемноження між старим станом комірки c_{t-1} і f_t, щоб забути інформацію, визначену на попередньому кроці. Потім нове значення-кандидат c_t додається до шкали різною мірою, щоб оновити нове значення стану c_t. Розрахунок наведено в рівнянні (2.5):

$$c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t, \tag{2.5}$$

де c_t – новий стан пам’яті (комірки);

c_{t-1} – попередній стан пам’яті.

Вихідний гейт (o_t): визначає вихідний шар та регулює, яка частина пам'яті буде виведена на вихід з LSTM. Спочатку визначається вихідний стан комірки за $sigmoid$ шаром. Потім стан $cell$ масштабується між -1 і 1 за допомогою шару $tanh$ шаром. Масштабований стан осередку попарно множиться на початковий вихід, отриманий від шару $sigmoid$ шару, внаслідок чого на виході моделі отримується шукана інформація. Розрахунки показано в рівняннях (2.6) і (2.7):

$$o_t = \sigma(W_0 \times [h_{t-1}, x_t] + b_0), \tag{2.6}$$

де o_t – вихід вихідного гейту.

Інші символи мають ті ж значення, що і в формулі для вхідного гейту.

$$h_t = o_t \times \tanh(c_t), \tag{2.7}$$

де h_t – новий вихід моделі.

Інші символи мають ті ж значення, що і в формулі для вихідного гейту.

Процес взаємодії цих компонентів дозволяє LSTM ефективно вирішувати проблеми зникнення градієнту, які є типовими для звичайних рекурентних мереж.

Порівняльний аналіз перелічених підходів. Для визначення підходу задля вирішення поставленої задачі було створено таблицю 2.1, в якій порівняно описані вище алгоритми.

Таблиця 2.1 – Алгоритми для розпізнавання жестів

Критерій / Алгоритм	Передбачувана точність	Обчислювальна складність	Вимоги до даних	Виконання в режимі реального часу	Стійкість до дисперсії	Простота впровадження
SVM	Помірна	Помірна, залежить від ядра та розміру набору даних	Добре працює при помірних обсягах даних, коли кількість ознак не дуже велика	Може бути обчислювально витратним, особливо для великих наборів даних	Зазвичай має високу стійкість до дисперсії	Зазвичай вимагає досить складного налаштування гіперпараметрів
KNN	Помірна	Низька в навчанні, помірна в тестуванні	Добре працює для невеликих та середньоромірних наборів даних.	Може бути повільним, особливо при великій кількості точок у наборі даних.	Зазвичай менш стійкий до дисперсії,	Легко впроваджується, особливо для невеликих наборів даних
LR	Помірна	Помірна в навчанні, низька в тестуванні	Добре працює на великих наборах даних з числовими та категорійними ознаками	Зазвичай досить ефективний і швидкий у режимі реального часу	Зазвичай стійкий до дисперсії, особливо якщо враховані оптимізації та регуляризація	Дуже простий у реалізації та використанні

Продовження таблиці 2.1

DT	Помірна	Від низької до помірної	Підходить для різних типів даних	Може бути швидким, але складність збільшуєтьс я зі зростанням глибини дерева	Схильний до перенавчан ня та менш стійкий до дисперсії на нових даних.	Легко впроваджує ться
RF	Помірна	Висока в навчанні, помірна в тестуванні	Працює добре для різних типів даних та великих наборів.	Може бути витратним в обчисленнях через велику кількість дерев.	Зазвичай має високу стійкість до дисперсії завдяки ансамблю дерев.	Досить простий у використанні але може вимагати оптимізації параметрів
CNN	Висока	Висока, вимагає значних обчислюва льних ресурсів	Потребує великої кількості даних для більшої точності	Може бути витратним в обчисленнях	Стійкий до дисперсії, але може бути схильним до перенавчан ня при обмежених даних	Складніший у впровадженн і через потребу у великій кількості даних та обчислюваль них ресурсах
RNN	Висока	Висока, вимагає значних обчислюва льних ресурсів	Працює добре для послідовних даних	Може бути витратним в обчисленнях , але існують оптимізовані структури та архітектури.	Може бути менш стійким до дисперсії, особливо при роботі з довгостроко вими залежностя ми	Досить складний у впровадженн і через необхідність управління послідовніст ю та можливі проблеми зі зникаючим градієнтом

Закінчення таблиці 2.1

LSTM	Висока	Працює ефективно, особливо на великих наборах даних	Має певні витрати в обчислюваннях, але можливо використовувати оптимізовані структури для реального часу	Має високу обчислювальну складність	Має високу стійкість до дисперсії завдяки своєму механізму управління пам'яттю	Складний у впровадженні

Як видно із таблиці, кожен алгоритм має свої переваги та недоліки, в залежності від досліджуваної ознаки. Після ретельного аналізу особливостей даних алгоритмів для розв'язання поставленої задачі було прийнято рішення використовувати модель LSTM, адже вона має кілька явних переваг над іншими методами в контексті вирішення завдання класифікації жестів.

1. Управління довгостроковими залежностями. LSTM дозволяє враховувати та вивчати складні довгострокові залежності між рухами, що дозволяє більш точно визначати та класифікувати різні жести в залежності від їхнього контексту та послідовності.

2. Стійкість до дисперсії. Механізми управління пам'яттю та гейти дозволяють LSTM бути менш чутливим до варіацій та невизначеностей у вхідних даних, що сприяє стійкому та надійному розпізнаванню жестів навіть при невеликих змінах у виконанні.

3. Ефективність на великих обсягах даних. LSTM показує високу ефективність при роботі з великими обсягами даних, що може бути важливим для навчання моделі на різних варіаціях жестів та досягнення високої точності.

4. Можливість використання в реальному часі. Легко оптимізується для виконання в режимі реального часу, що дозволяє використовувати модель для інтерактивних систем та додатків, де важливий швидкий та точний відгук на жести.

З урахуванням цих переваг, модель LSTM виявляється найбільш придатною для досягнення високої ефективності та точності у завданні класифікації жестів.

2.2 Технології розробки системи

Мова програмування Python. Для реалізації системи було обрано досить популярну мову програмування Python. Це високорівнева, універсальна та інтерпретована мова програмування, яка набула широкої популярності завдяки своїй читабельності, простоті та легкості у використанні. Створена Гвідо ван Россумом і вперше випущена в 1991 році, Python перетворилася на потужну та надійну мову, що використовується в різних сферах, від веброзробки та науки про дані до штучного інтелекту та автоматизації.

Python має простий та зрозумілий синтаксис, що робить цю мову доступною. Це сприяє швидкому розробленню прототипів та експериментуванню з алгоритмами розпізнавання жестів. Python має широкий вибір бібліотек та фреймворків, які полегшують розробку системи зчитування жестів. Наприклад, бібліотека OpenCV, яка широко використовується в комп'ютерному зорі, має підтримку для Python та надає зручні інструменти для обробки зображень та відео. Python є популярним вибором для розробки моделей машинного навчання. Існує багато бібліотек, таких як TensorFlow, PyTorch, і Keras, які надають потужні інструменти для навчання та застосування моделей розпізнавання жестів.

Середовище розробки Jupyter Notebook. Jupyter Notebook – це оригінальний вебзастосунок для створення та обміну обчислювальними документами [30]. Він пропонує простий, оптимізований, орієнтований на роботу з документами досвід [30]. Це інтерактивне розробницьке середовище, яке дозволяє миттєво спостерігати за результатами виконання коду та його окремих частин. Його відмінність від традиційних середовищ розробки полягає в можливості розбити код на фрагменти та виконувати їх у будь-якому порядку, надаючи розробникам більшої гнучкості у вивченні та вдосконаленні коду. Jupyter Notebook буде використовуватись в даній роботі як середовище розробки під час

створення набору даних, адже даний інструмент надає широкі можливості для відображення відеопотоку з вебкамери в реальному часі безпосередньо в середовищі розробки, завдяки інтеграції з різними бібліотеками, такими як OpenCV.

Середовище розробки Google Colab. Python – крос-платформна мова програмування, що означає, що вона може працювати на різних платформах, таких як Windows, macOS і Linux, і навіть була портована на віртуальні машини Java і .NET. Це безкоштовна мова з відкритим вихідним кодом. Існує багато IDE (інтегрованих середовищ розробки) та редакторів коду, доступних відповідно до вимог розробника. IDE призначені для полегшення роботи розробника шляхом поєднання інструментів, необхідних під час розробки програмного забезпечення. Декілька популярних IDE та редакторів коду: PyCharm, Visual Studio Code, Jupyter Notebook, Google Colab.

Google Colab – це чудова платформа для розробки та запуску коду на Python без жодних інсталяцій [31]. Google Colab також надає деякі розширені функції, такі як підтримка GPU і TPU, що виявляється дуже корисним під час запуску моделей машинного навчання і глибокого навчання. Google Colab – потужний і безкоштовне інструмент, який надає багато можливостей для розробки та запуску коду Python.

1. Безкоштовний у використанні. Однією з найважливіших переваг Google Colab є те, що він абсолютно безкоштовний у використанні. Тому не потрібно платити жодних абонентських платежів або завантажувати будь-яке програмне забезпечення. Все, що треба – це обліковий запис Google, щоб отримати доступ до сервісу без будь-яких попередніх умов.
2. Хмарний. Google Colab повністю хмарний, вся обробка відбувається на серверах Google, що робить його простим у використанні на будь-якому пристрої, а також економить пам'ять комп'ютера.
3. Google Colab дозволяє легко співпрацювати з іншими в режимі реального часу. Це особливо корисно, якщо над проєктом працюють командою.
4. Підтримка графічних процесорів. Google лабораторія надає підтримку

графічних процесорів, яка буде корисною, якщо розробник працює над глибоким навчанням або іншими завданнями з інтенсивними обчисленнями. Графічний процесор можна безкоштовно використовувати до 12 годин за раз.

5. Вбудовані бібліотеки. Google Colab постачається з багатьма вбудованими бібліотеками, такими як NumPy, pandas, matplotlib та багатьма іншими, які потрібно імпортувати, щоб отримати доступ до їх функціональності. Також можна встановити додаткові бібліотеки за допомогою pip, що полегшує роботу з широким спектром інструментів.

Середовище розробки Visual Studio Code. VS Code – це спрощений, але потужний редактор вихідного коду [32], який працює на комп'ютері і доступний для Windows, macOS і Linux. Ключові аспекти та особливості Visual Studio Code.

1. Крос-платформенна підтримка. Visual Studio Code сумісний з Windows, macOS та Linux, що робить його кросплатформним редактором коду. Це дозволяє розробникам без проблем використовувати один і той самий інструмент у різних операційних системах.

2. Інтуїтивно зрозумілий інтерфейс. VS Code має чистий та інтуїтивно зрозумілий користувацький інтерфейс, в якому легко орієнтуватися. Він включає бічну панель для навігації по файлах, інтегрований термінал і гнучкий макет, який можна налаштувати відповідно до уподобань користувача.

3. Мовна підтримка. Має вбудовану підтримку JavaScript, TypeScript і Node.js, а також велику екосистему розширень для інших мов і середовищ виконання (наприклад, C++, C#, Java, Python, PHP, Go, .NET). Крім того, майже для кожної мови програмування доступні розширення, які надають розширену підтримку та можливості.

4. Розширення та маркетплейс. VS Code має багату екосистему розширень, що дозволяє розробникам розширювати функціональність редактора. Visual Studio Code Marketplace надає величезну колекцію розширень для тем, мов, інструментів налагодження, систем контролю версій і багато іншого. Користувачі можуть легко встановлювати розширення та керувати ними безпосередньо з редактора.

5. Потужні можливості налагодження. Середовище пропонує надійні засоби налагодження для різних мов програмування. Він підтримує точки зупинки, покрокове налагодження, перевірку змінних та інтегровані конфігурації налагодження для різних середовищ виконання.

6. Автоматизація завдань. Редактор дозволяє користувачам визначати і запускати завдання безпосередньо у редакторі. Ця функція корисна для автоматизації типових завдань розробки, таких як створення проектів, запуск тестів або розгортання програм.

Бібліотека OpenCV. Open Source Computer Vision Library – бібліотека комп'ютерного зору та машинного навчання з відкритим кодом [33]. OpenCV було створено, щоб забезпечити загальну інфраструктуру для програм комп'ютерного зору та прискорити використання машинного сприйняття в комерційних продуктах. Бібліотека містить понад 2500 оптимізованих алгоритмів, що включає повний набір як класичних, так і найсучасніших алгоритмів комп'ютерного зору 29 та машинного навчання. Ці алгоритми можна використовувати для виявлення та розпізнавання облич, ідентифікації об'єктів, класифікації дій людей у відео, відстеження рухів камери, відстеження рухомих об'єктів, вилучення 3D-моделей об'єктів, створення 3D-хмар точок із стереокамер, з'єднання зображень для отримання високої роздільної здатності. зображення цілої сцени, знаходити подібні зображення в базі даних зображень, видаляти червоні очі із зображень, зроблених за допомогою спалаху, стежити за рухами очей, розпізнавати пейзаж і встановлювати маркери для накладання на нього доповненої реальності тощо. OpenCV має понад 47 тисяч користувачів, оціночна кількість завантажень перевищує 18 мільйонів. Бібліотека широко використовується компаніями, дослідницькими групами та державними органами.

Бібліотека Keras. Keras – це API для глибокого навчання, написаний на Python і здатний працювати поверх JAX, TensorFlow або PyTorch [34]. Його було розроблено з розрахунком на швидке навчання. Здатність переходити від гіпотез до результатів з найменшими часовими витратами є ключем до проведення

успішних досліджень.

Бібліотека для тестування та навчання нейронних мереж з відкритим кодом, написана на Python. Дана бібліотека сумісна з TensorFlow, Microsoft Cognitive Toolkit, PlaidML, R або Theano. Вона призначена для експериментів з глибокими нейронними мережами і фокусується на зручності користування, модульності та розмірності. Окрім стандартних нейронних мереж, Keras має підтримку згорткових і рециндентних нейронних мереж. Keras потрібно використовувати, якщо потрібна бібліотека глибокого навчання, яка дає змогу легко і швидко створювати прототипи (завдяки зручності, модульності та масштабуванню), підтримує як згорткові та рекурентні мережі, так і їхні комбінації та без проблем працює як на процесорі (CPU), так і на графічному процесорі (GPU).

Бібліотека TensorFlow. Це бібліотека програмного забезпечення з відкритим вихідним кодом для програмування потоків даних [35], орієнтована на паралельні обчислення на CPU та GPU. Спочатку TensorFlow був розроблений дослідниками Google Brain, підрозділу Alphabet Inc., в основному для досліджень машинного навчання, але з тих пір він використовувався для розробки інших застосунків, таких як чат-боти, самокеровані автомобілі, обробка природної мови тощо. Використання TensorFlow продовжує швидко розширюватися, тому що філософія дизайну, що лежить в основі бібліотеки, дозволяє легко впроваджувати нові функції, зберігаючи при цьому старі функції належним чином. Зараз дослідники використовують TensorFlow у дослідженнях штучного інтелекту (англ. artificial intelligence, AI), комп’ютерного зору (англ. computer vision, CV) та глибокого навчання (англ. deep learning, DL).

TensorFlow можна використовувати для навчання моделей штучного інтелекту. Його також можна використовувати для попередньої обробки та аналізу даних перед подачею їх у моделі AI. Ця попередня обробка може включати такі речі, як зменшення шуму, нормалізація даних та виправлення зміщення. TensorFlow має багато потенційних застосувань, включаючи навчання та попередню обробку даних, оцінку продуктивності та підвищення точності.

Бібліотека NumPy. Це проєкт з відкритим вихідним кодом, який дозволяє виконувати числові обчислення за допомогою Python. Він був створений у 2005 році на основі ранніх напрацювань бібліотек Numeric та Numarray. Основний пакет для наукових обчислень на Python [36]. Ця бібліотека надає об’єкт багатовимірного масиву, різноманітні похідні об’єкти (такі як замасковані масиви та матриці), а також ряд процедур для швидких операцій над масивами, включаючи математичні, логічні, маніпуляції формою, сортування, вибір, введення/виведення. , дискретне перетворення Фур'є, базова лінійна алгебра, основні статистичні операції, випадкове моделювання та багато іншого.

Фреймворк MediaPipe. Інструмент з відкритим вихідним кодом, представлений Google [37], який допомагає створювати мультимодальні конвеєри машинного навчання. Розробник може створити прототип, не заглиблюючись у написання алгоритмів та моделей машинного навчання, використовуючи наявні компоненти. Ця структура може використовуватися для різних програм для обробки зображень та мультимедіа (особливо у віртуальній реальності), таких як виявлення об'єктів, розпізнавання облич, відстеження рук, відстеження кількох рук та сегментація волосся.

MediaPipe підтримує різні апаратні та операційні платформи, такі як Android, iOS та Linux, пропонуючи API на C++, Java, Objective-c [38]. Це середовище також здатне використовувати ресурси GPU. Mediapipe поставляється з готовими до використання моделями, де розробники можуть почати використовувати його безпосередньо або зі своїми модифікаціями. Виявлення заперечень може бути опрацьовано дуже легко, не споживаючи багато системних ресурсів. Виявлення об'єктів на основі ML з камери прямої трансляції з частотою кадрів 30 кадрів на секунду зазвичай потребує великих ресурсів і неможливе через тривалий час виведення. MediaPipe досягає цього шляхом паралельного запуску, відстеження та виявлення, тому кожен процес ніколи не буде заблокований іншим.

Бібліотека Mediapipe містить навчену ШНМ для розпізнавання рук. Долоня руки складається з суглобів, які можна подати точками, сполученими відрізками.

MediaPipe розпізнає орієнтири і витягує ключові точки з таких об'єктів, як рука, тіло або обличчя. Цей фреймворк допомагає у вирішенні поширених проблем динамічної мови жестів. З одного боку, MediaPipe може визначати форму і розташування рук і тіла. Він також вирішує питання орієнтації долоні та виразу обличчя, які є вторинними параметрами [39]. Для кожної руки MediaPipe виокремлює 21 ключову точку [40], як показано на рисунку 2.17. Ключові точки розраховуються у тривимірному просторі: X, Y та Z для обох рук. Таким чином, кількість вилучених ключових точок рук обчислюється за наступною логікою.

Ключові точки в руках × Три виміри × Кількість рук = (21×3×2) = 126 ключових точок.

Оскільки буде реалізовано розпізнавання динамічних жестів також, то класифікація жестів буде виконуватися не з одного зображення, а з їх послідовності. На відображення одного жесту виділено 30 кадрів (кадрова частота OpenCV 30 кадрів в секунду). Таким чином, жест можна представити двовимірним масивом, що складається з 30 кадрів кожен з яких містить 63 координати точок рук (21 точка помножена на 3 виміри).

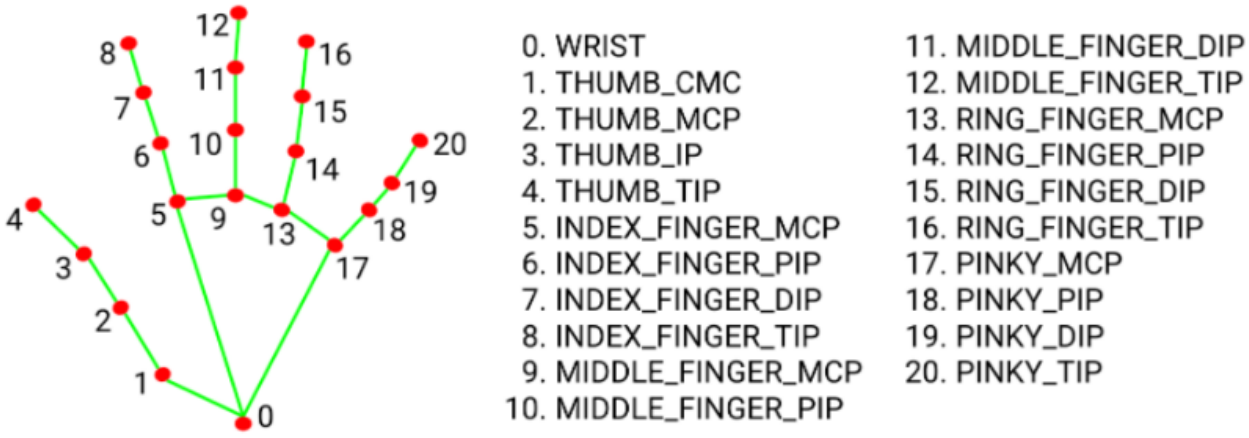

Рисунок 2.7 – Порядок і мітки для ключових точок, які існують в руках MediaPipe [41]

Висновки до розділу 2

В даному розділі було розглянуто наступні алгоритми машинного навчання: метод опорних векторів, метод k-найближчих сусідів, логістичну регресію, дерево рішень, випадковий ліс та методи глибокого навчання такі як згорткову нейронну мережу, рекурентну нейронну мережу, а також різновид RNN – довгострокову короткочасну пам’ять. Розглянуто принципи роботи кожного із методів. Встановлено їх недоліки та переваги. І як підсумок, було складено порівняльну таблицю алгоритмів за спільними ознаками. Для реалізації моделі штучної нейронної мережі обрано LSTM, адже ці мережі спроможні запам’ятовувати інформацію протягом тривалих періодів часу, через що навчання не вимагає багато часу. LSTM, при достатній кількості елементів може виконати будь-яке обчислення, тобто є універсальною; також вона добре орієнтована на вирішення проблем класифікації.

У другому підрозділі описано основні технології, які було обрано для реалізації системи, а саме – мову програмування Python, середовище розробки Google Colab, Jupyter та Visual Studio Code, бібліотеки OpenCV, Keras, NumPy, TensorFlow, фреймворк MediaPipe.

3 РОЗРОБКА ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ПЕРЕТВОРЕННЯ ЖЕСТОВОЇ МОВИ НА ТЕСТ ТА АУДІО

3.1 Розробка дизайну програмного забезпечення

Розробка дизайну програмного забезпечення є важливим етапом у життєвому циклі програмного продукту. Цей процес включає створення недеталізованого макету (ескізу) та його подальше деталізування. Нижче розглянуто обидва етапи.

Ескіз (англ. sketch) в контексті розробки програмного забезпечення відноситься до швидких та неформальних рисунків, які використовуються для візуалізації ідей та концепцій. В розробці інтерфейсів скетчі часто використовуються для створення простих макетів або сценаріїв користування. Вони можуть бути намальовані вручну або створені за допомогою графічних програм. На рис. 3.1 зображено створений недеталізований макет розроблювальної системи.

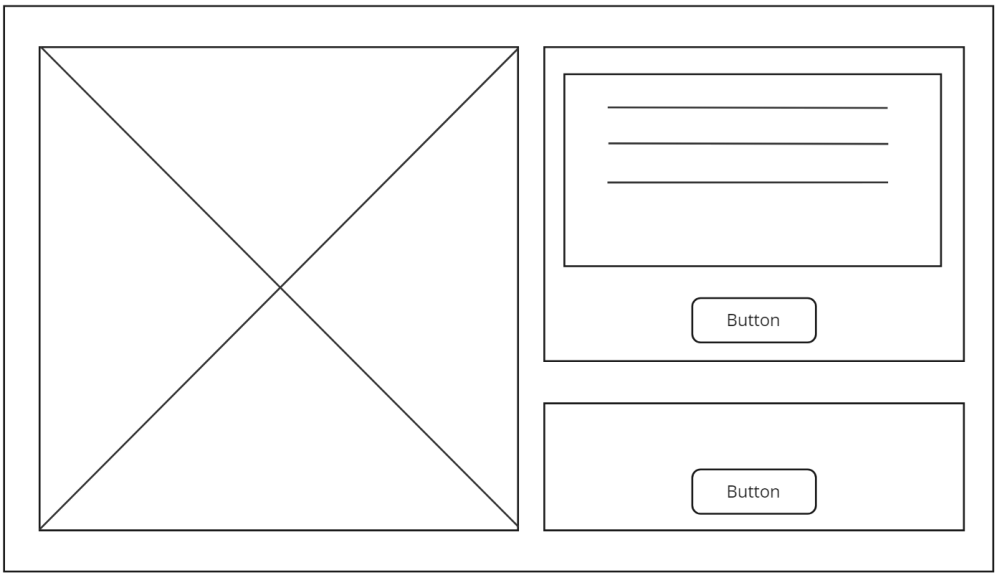

Рисунок 3.1 – Візуальне представлення концепції програмного забезпечення

Недеталізований макет передає загальну структуру та ідеї проєкту. Він слугує початковим кроком у розробці, де основна увага приділяється встановленню високорівневої архітектури системи та визначенню ключових компонентів. Цей етап допомагає зрозуміти загальний обсяг проєкту та визначити його основні функціональні вимоги.

Деталізований макет (рис. 3.2) є наступним кроком після недеталізованого та передує фазі реалізації програмного продукту (рис. 3.3). Також дозволяє розробнику зрозуміти, як конкретно реалізувати функціонал системи та як взаємодіють її компоненти. Цей етап є підготовкою до фази фактичної реалізації програмного продукту.

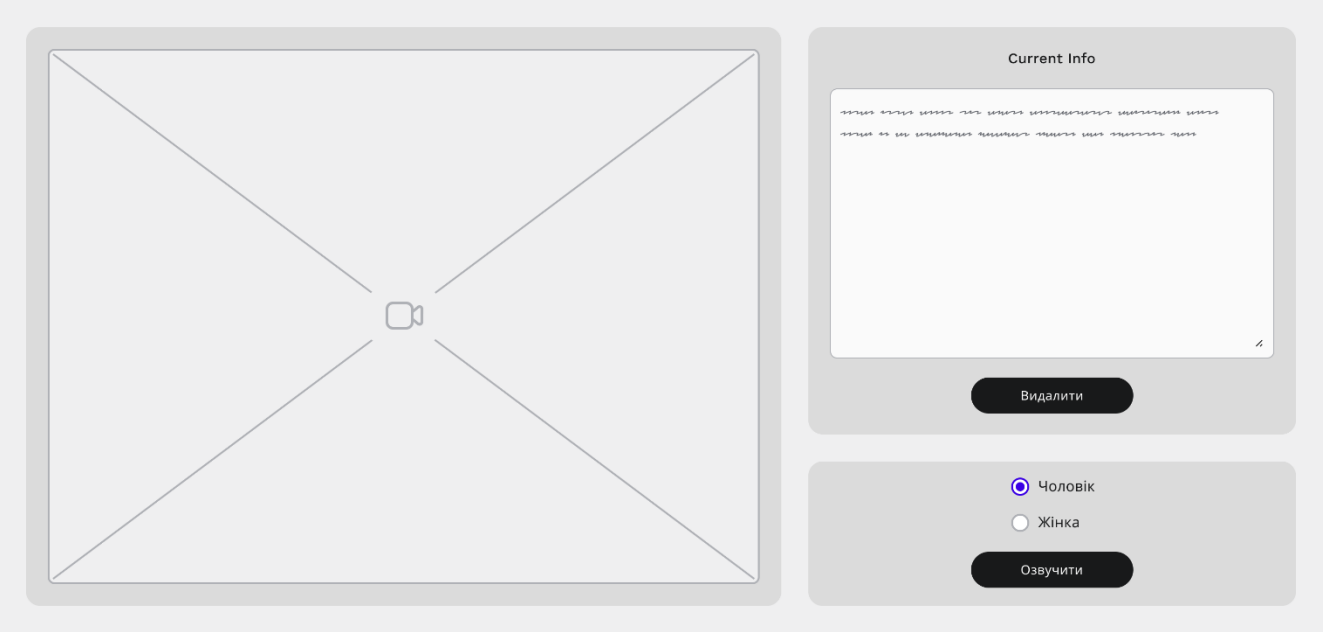

Рисунок 3.2 – Деталізований макет системи

Створення дизайну системи після етапу деталізованого макету є наступним кроком у процесі розробки програмного продукту. На цьому етапі розробники концентруються на визначенні деталей візуального вигляду та інтерфейсу програми, а також вирішенні питань, пов'язаних з ергономікою та користувацькою взаємодією. На рис. 3.3 показано розроблений дизайн користувацького інтерфейсу.

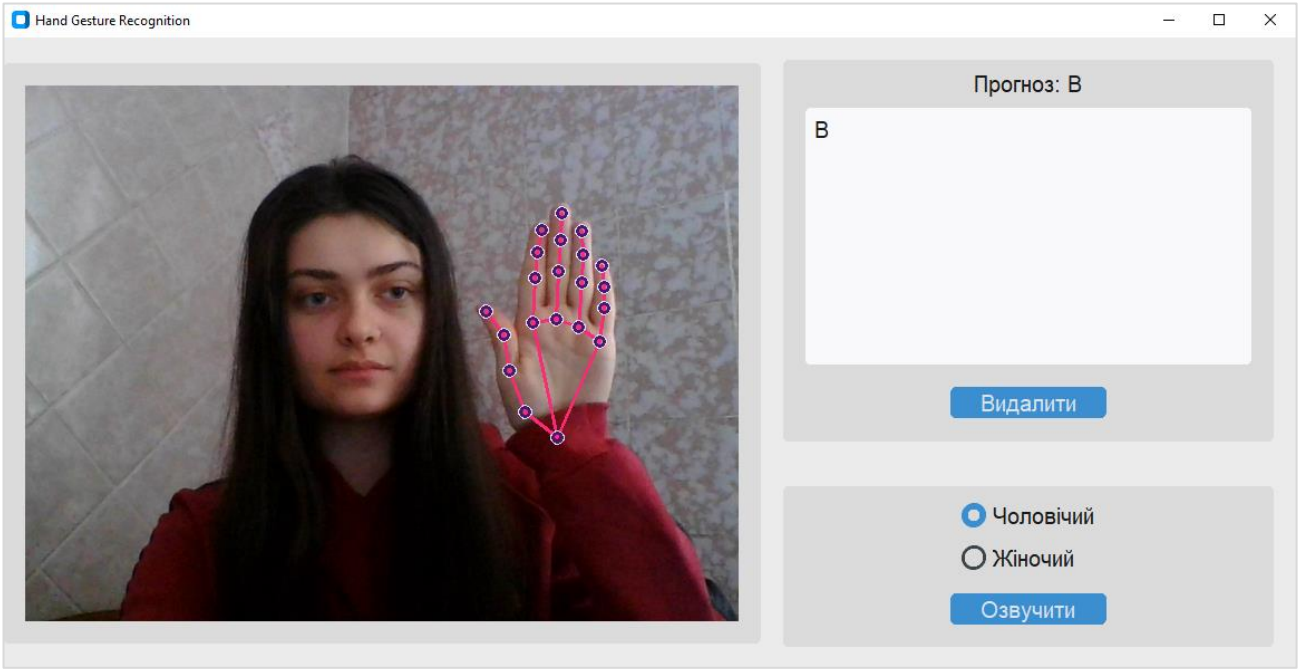

Рисунок 3.3 – Розроблений дизайн користувацького інтерфейсу

3.2 Архітектура інтелектуальної системи

UML (англ. Unified Modeling Language) є мовою графічного опису для моделювання об'єктно-орієнтованих систем. Це нотація, яка використовується для візуального представлення різних аспектів програмних систем та їхніх компонентів. UML надає засоби для опису, візуалізації, специфікації, конструювання та документування програмних систем. Елементами UML є різноманітні діаграми (в тому числі, діаграма прецендентів і діаграма послідовності) та модельні елементи, такі як класи, об'єкти, прецеденти, актори та відносини між ними. Ці компоненти використовуються для візуального представлення структури та функціональності програмних систем, що полегшує аналіз, проектування та документацію програмного забезпечення.

Діаграма послідовностей (англ. Sequence Diagram) – це один із видів діаграм в мові моделювання UML, який використовується для візуального представлення послідовності взаємодій між об'єктами чи учасниками в системі протягом конкретного сценарію чи функціональної дії. На рис. 3.4 зображено створену діаграму послідовностей для розроблювальної системи.

Основні елементи діаграми послідовностей включають:

1) об'єкти (jbjects): представлення реальних або віртуальних об'єктів, що беруть участь у взаємодії. Кожен об'єкт має свій власний життєвий цикл та взаємодіє з іншими об'єктами;

2) лайфлайни (lifelines): відображають термін існування об'єкта під час виконання конкретної операції або сценарію. Лайфлайни представляють собою вертикальні лінії, які зображують часовий проміжок, протягом якого об'єкт існує;

3) послідовність (sequence): візуальне зображення послідовності викликів методів, повідомлень та інших подій між об'єктами. Стрілки вказують напрямок виклику чи відправлення повідомлення;

4) операції (operations): показують методи чи функції, які викликаються між об'єктами у конкретний момент часу.

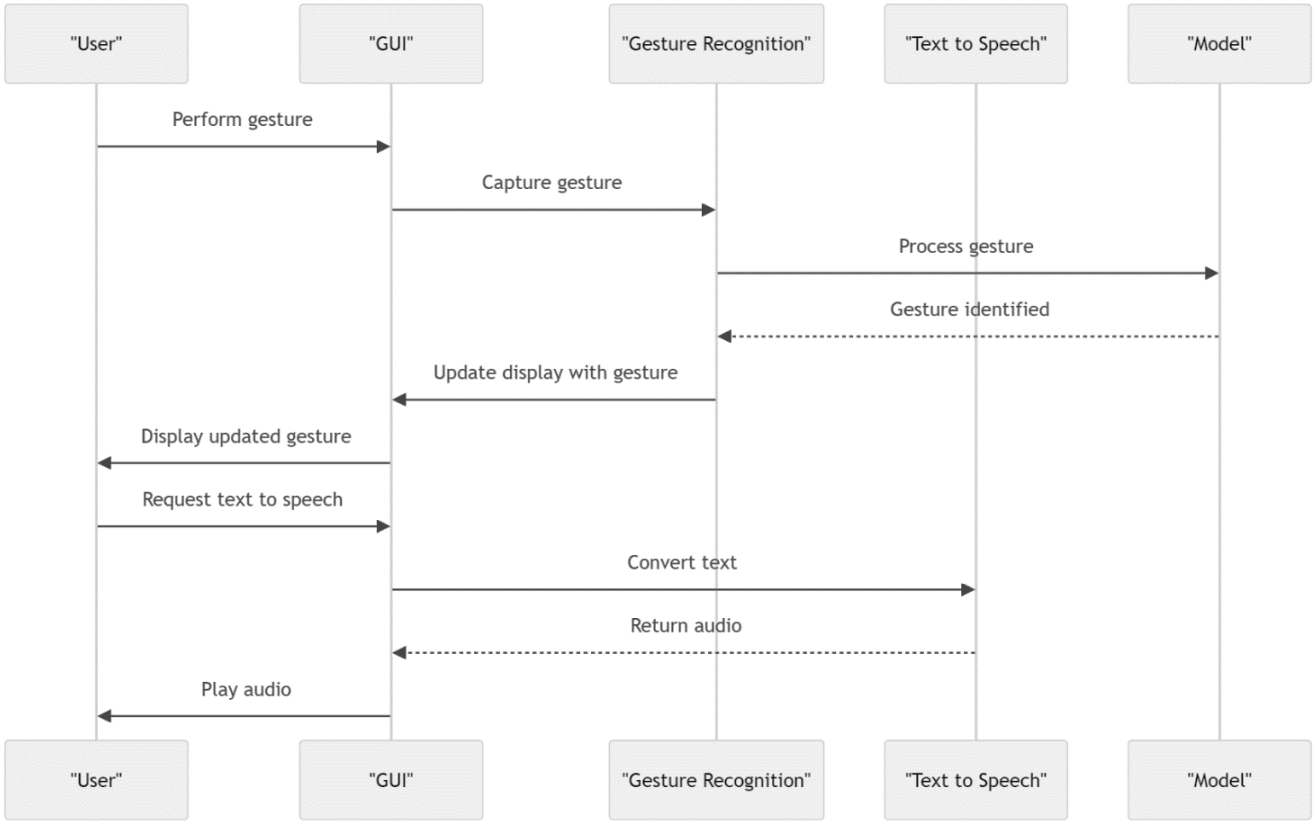

Рисунок 3.4 – Діаграма послідовностей системи розпізнавання жестової мови

Sequence Diagram візуально представляє взаємодію між різними компонентами системи, призначеної для розпізнавання жестів і перетворення тексту в мову (TTS), що слідує за жестами користувача.

1. Користувач: особа, яка взаємодіє з системою, виконує жести та отримує звуковий зворотній зв'язок.

2. Графічний інтерфейс користувача: інтерфейс, через який користувач взаємодіє з системою. Він фіксує жести та відображає оновлення.

3. Розпізнавання жестів: компонент, що відповідає за зчитування та інтерпретацію жестів користувача.

4. Модель, яка обробляє дані про жести для ідентифікації жесту.

5. Перетворення тексту в мовлення (TTS): компонент, який перетворює текст на розмовний звук, який потім відтворюється користувачеві.

Послідовність взаємодій починається з того, що користувач виконує жест, який фіксується графічним інтерфейсом. Далі графічний інтерфейс пересилає цей жест до компоненту розпізнавання жестів. Розпізнавання жестів обробляє жест і надсилає дані до Моделі для прогнозування. Модель аналізує дані про жест та ідентифікує його. Потім вона надсилає результат розпізнавання назад до компонента «Розпізнавання жестів». Отримавши ідентифікований жест, компонент розпізнавання жестів надсилає оновлення до графічного інтерфейсу, щоб відобразити розпізнаний жест користувачеві. Користувач бачить прогнозований жест у вигляді тексту.

Далі користувач може виконати перетворення тексту в мовлення для рядка прогнозованих жестів. Для цього графічний інтерфейс надсилає текст до компонента TTS для перетворення тексту в аудіо. Потім компонент TTS обробляє запит і повертає звук до графічного інтерфейсу. Нарешті, графічний інтерфейс відтворює звук користувачеві, завершуючи цикл взаємодії.

На діаграмі використовуються різні типи стрілок для представлення характеру зв'язку між компонентами:

– суцільні стрілки (->) вказують на синхронний виклик або запит, де відправник чекає на відповідь;

– пунктирні стрілки (-->) позначають відповіді або асинхронні повідомлення, які не вимагають від відправника очікування.

Дана діаграма ефективно ілюструє динамічну взаємодію і потік інформації між компонентами системи, від захоплення жестів до звукового зворотного зв'язку, висвітлюючи процес розпізнавання жестів і надання відповідного аудіовиходу користувачеві.

Діаграма прецедентів (англ. Use Case) – конкретний сценарій взаємодії користувача (або іншої системи) з програмним продуктом. Діаграма прецедентів є одним із видів діаграм у мові моделювання UML (англ. Unified Modeling Language), який використовується для визначення функціональності системи та взаємодій між різними її частинами. Зазвичай використовується для моделювання взаємодій між зовнішніми агентами (користувачами чи іншими системами) та самою системою.

Основні елементи Use Case.

1. Актори (Actors). Актори представляють суб'єктів або сутності, які взаємодіють з системою. Це можуть бути користувачі, інші системи, зовнішні сервіси, або будь-які інші сторони, які мають інтерес до системи.

2. Прецеденти (Use Cases). Прецеденти визначають конкретні функціональні можливості чи послуги, які система надає для акторів. Кожен прецедент відображає конкретний сценарій взаємодії між актором і системою.

3. Взаємодія (Interactions). Взаємодія між акторами та прецедентами описується у вигляді послідовності подій чи кроків, які відбуваються під час виконання конкретного сценарію.

4. Сценарії (Scenarios). Кожен прецедент може мати кілька сценаріїв, які представляють різні варіанти взаємодії в залежності від умов чи обставин.

Використання Use Case допомагає зрозуміти, як користувачі будуть використовувати систему та які функції вони будуть виконувати.

Це важливий інструмент для аналізу та документування вимог до системи та служить основою для подальшого проектування та розробки програмного продукту. Діаграма прецедентів використання системи розпізнавання жестів рук візуально представляє взаємодію між користувачем і функціональними можливостями системи (рис. 3.5).

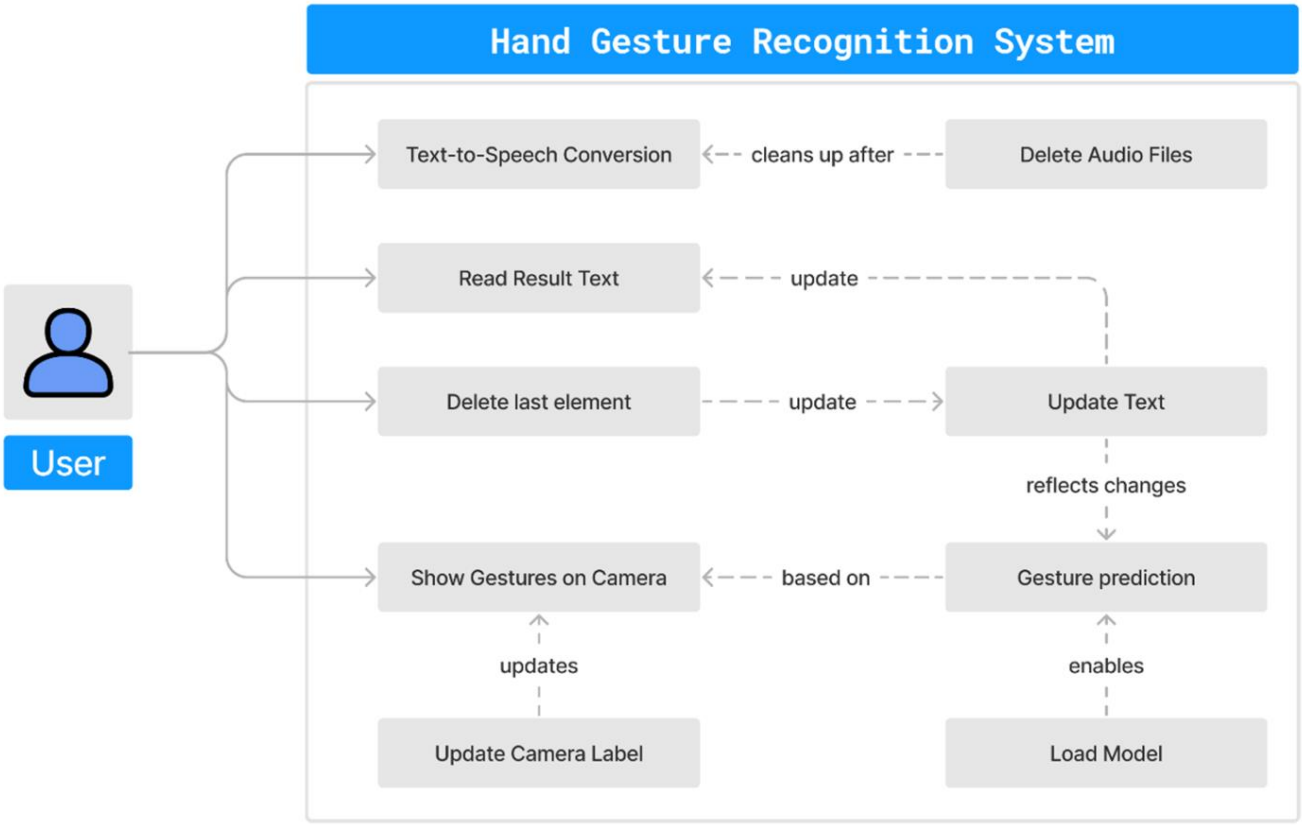

Рисунок 3.5 – Діаграма прецендентів

Діаграма визначає одного актора, Користувача, який взаємодіє з системою. Цей актор представляє будь-яку особу, яка використовує систему розпізнавання жестів рук. Система інкапсульована в прямокутнику з написом «Система розпізнавання жестів рук», що вказує на те, що всі варіанти використання є частиною цієї системи.

Усередині системи є кілька варіантів використання, кожен з яких представляє певну функціональність. Користувач може взаємодіяти з такими функціями.

1. Перетворення тексту в мовлення. Це здатність системи перетворювати текст на мовлення. Це дуже важлива функція для забезпечення доступності, що дозволяє користувачам отримувати слуховий зворотний зв'язок або інструкції.

2. Прочитати текст. Це здатність системи показати прогнозований текст, можливість для користувача отримати зворотній зв’язок із системою і перевірити правильність розпізнавання.

3. Видалити останній елемент. Цей варіант використання дозволяє користувачеві видалити останній елемент з послідовності або списку, при неправильному розпізнаванні, або очистки поля для наступного речення.

4. Показ жестів на камеру. Це основний інструмент взаємодії користувача із системою. Користувач показує жести, які потім система повинна розпізнати.

Інший функціонал використовується всередині системи.

1. Передбачення жестів. Ця важлива функція системи передбачає жести користувача на основі аналізу відео з камери. Для розпізнавання жестів використовується навчена модель глибокого навчання.

2. Завантаження моделі. цей функціонал використання стосується завантаження моделі розпізнавання жестів у систему. Це фоновий процес, необхідний для прогнозування жестів.

3. Видалення аудіофайлів. Ця функція дозволяє системі в кінці роботи видаляти аудіофайли, які були створені в процесі перетворення тексту в мову, щоб звільнити місце.

4. Оновлення камери. Цей варіант використання представляє здатність системи оновлювати зображення з камери в режимі реального часу. Він передбачає обробку зображення з камери, перетворення його у відповідний формат і відображення в інтерфейсі користувача.

5. Оновлення тексту. Ця функція дозволяє системі оновлювати текст в інтерфейсі користувача на основі введених даних або виконаних дій. Це включає зворотній зв’язок з користувачем та вивід прогнозованого результату.

6. Ця діаграма варіантів використання забезпечує високорівневий огляд функціональних можливостей системи з точки зору користувача, підкреслюючи, як користувач може взаємодіяти з системою і які дії він може виконувати.

Висновки до розділу 3

В даному розділі було розглянуто етапи розробки дизайну програмного забезпечення. Спочатку було розроблено як деталізований, так і недеталізований макети програмного продукту, щоб визначитись із функціоналом та виглядом. Далі відбулася розробка готового дизайну користувацького інтерфейсу, що включає в себе елементи інтерфейсу, які забезпечують зручність взаємодії з програмою.

У процесі дослідження архітектури розроблювальної системи для перетворення жестової мови на текст та аудіо, було ретельно розглянуто взаємодію різних компонентів цієї системи. Діаграма послідовностей створена з метою візуалізації та аналізу проходження даних та виконання функцій протягом різних етапів обробки жестів. На цій діаграмі чітко відображено, як користувацькі жести інтерпретуються та обробляються системою на кожному кроці. Починаючи з отримання жесту, система визначає його тип, після чого ініціює відповідні обчислення для генерації текстової або аудіо інформації. Це сприяє кращому розумінню та оптимізації процесів в системі.

Діаграма прецедентів деталізує основні функції та можливості системи, враховуючи потреби та очікування користувачів. Визначено різні сценарії взаємодії, включаючи перетворення тексту в мовлення, показ жестів у режимі реального часу, видалення згенерованого тексту тощо. Це забезпечує повний огляд можливостей системи та її здатностей до задоволення потреб користувачів.

Такий підхід до аналізу та проєктування архітектури системи не лише сприяє розумінню самого процесу, але і визначає основні аспекти, які впливають на якість та ефективність програмного продукту, що розробляється.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1 Створення набору даних

Для створення нейронної моделі з високим показником точності потрібна значна кількість навчальних даних. Проте, загальнодоступного датасету із необхідним форматом даних для української жестової мови – немає, тому набір було створено власноруч. Датасет представлятиме собою колекцію зібраних даних ключових точок рук під час виконання різних статичних та динамічних жестів.

Для початку потрібно імпортувати бібліотеку OpenCV для завдань комп'ютерного зору та Mediapipe. У даному випадку, будуть використовуватись модулі *holistic* та *drawing_utils*. Модуль *holistic* містить готову модель для виявлення ключових точок обличчя, тіла та рук, тоді як *drawing_utils* – корисні утиліти для ефективного відображення точок на зображенні. Далі використовується функція *mediapipe_detection* (рис. 4.1), в котрій реалізується наступна логіка. Для виявлення рук за допомогою моделі *Holistic* необхідно передати кадр з відеотрансляції OpenCV. Однак, враховуючи те, що OpenCV працює з колірним каналом у форматі BGR (Blue-Green-Red), необхідно виконати конвертацію кадру з формату BGR в RGB (Red-Green-Blue). Це робиться для оптимізації використання пам'яті, і після цієї конвертації кадр стає недоступним для запису. Після передачі конвертованого кадру на модель *Holistic* отримуємо результат розпізнавання, і кадр стає знову доступним для запису у форматі BGR.

```
def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image.flags.writeable = False
    results = model.process(image)
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    return image, results
```

Рисунок 4.1 – Обробка зображення для розпізнавання за допомогою MediaPipe

Ця послідовність дій призначена для забезпечення коректної взаємодії з бібліотекою Mediapipe. Функція *draw_landmarks* (рис. 4.2) призначена для візуалізації ключових точок на зображенні для обох рук, використовуючи результати моделі MediaPipe Holistic. Вона використовує модуль *mp_drawing*, щоб намалювати ключові точки і з'єднання між ними на зображенні.

```
def draw_landmarks(image, results):
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks,
                              mp_holistic.HAND_CONNECTIONS)
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks,
                              mp_holistic.HAND_CONNECTIONS)
```

Рисунок 4.2 – Візуалізація ключових точок обох рук

Стосовно функції *draw_styled_landmarks* (рис. 4.3), вона схожа на попередню, але надає додаткові параметри стилізації для намальованих ключових точок. Це дозволяє налаштувати колір, товщину та радіус кола для ліній та точок, які представляють ключові точки. Клас *mp_drawing.DrawingSpec* з бібліотеки MediaPipe використовується для визначення специфікацій (параметрів) стилізації, які малюються функцією *mp_drawing.draw_landmarks*. Наприклад:

- color: RGB-код кольору ліній або точок;
- thickness: товщина ліній, які з'єднують ключові точки;
- circle_radius: радіус кіл, що представляють ключові точки.

```
def draw_styled_landmarks(image, results):
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)
                              )
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                              )
```

Рисунок 4.3 – Візуалізація стилізованих ключових точок рук на зображенні

Функція *extract_keypoints* (рис. 4.4) отримує результати виявлення ключових точок за допомогою MediaPipe Holistic та витягує значення ключових точок для лівої та правої руки. Для кожної руки витягуються координати x, y та z для кожної ключової точки. Якщо виявлення не вдалося (наприклад, якщо рука не знаходиться в кадрі), повертається масив з нульовими значеннями.

```
def extract_keypoints(results):
    lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten()
    if results.left_hand_landmarks else np.zeros(21*3)
    rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten()
    if results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([lh, rh])
```

Рисунок 4.4 – Вилучення значень ключових точок

Далі реалізується логіка для налаштування структури папок для збору та зберігання даних (рис. 4.5). Визначається шлях, в якому будуть зберігатися дані у форматі NumPy arrays.

Масив міститиме 20 жестів (['І', 'А', 'В', 'Д', 'Е', 'С', 'У', 'Ц', 'Ь', 'будинок', 'дванадцять', 'добрий день', 'записатись на прийом ', 'лікар', 'сьогодні ', 'телефон', 'технологія', 'хочу', 'я', 'як ваші справи']), які будуть використовуватися для навчання моделі розпізнавання. Таким чином, модель буде розпізнавати 20 класів. Також потрібно зазначити параметри для збору даних. А саме – параметр *no_sequences* для позначення кількості відео (100) та параметр *sequence_length*, який вказує на кількість кадрів на кожне відео (30).

Після того, як створено папку для датасету та відповідні папки в ній під дані для кожного жесту, наступним етапом є збір даних у папки – запускається цикл для створення 100 папок з відео для кожного з жестів (рис. 4.6).

```
DATA_PATH = os.path.join('/content/dataset_uk/USL_Dataset')
actions = np.array(['І', 'А', 'В', 'Д', 'Е', 'С', 'У', 'Ц', 'Ь', 'будинок',
                    'дванадцять', 'добрийдень', 'записатисьнаприйом', 'лікар',
                    'сьогодні', 'телефон', 'технологія', 'хотіти', 'я', 'яквашісправи'])
no_sequences = 100
sequence_length = 30
```

Рисунок 4.5 – Налаштування папок для збору даних

```
for action in actions:
    for sequence in range(no_sequences):
        try:
            os.makedirs(os.path.join(DATA_PATH, action, str(sequence)))
        except:
            pass
```

Рисунок 4.6 – Цикл для створення структури папок для зберігання даних

Фінальним кроком створення датасету є безпосередньо реалізація процедури збору даних (рис. 4.7). Використовуючи OpenCV, встановлюється з'єднання з веб-камерою (з індексом 0, що вказує на використання вбудованої вебкамери). Створюється екземпляр класу *Holistic* з бібліотеки MediaPipe для виявлення та відстеження ключових точок рук. Параметри *min_detection_confidence* і *min_tracking_confidence* встановлюють мінімальні рівні впевненості для виявлення та відстеження ознак відповідно.

Процес збору даних реалізується у трьох вкладених циклах. Перший проходиться по класам набору даних, другий перебирає визначену кількість послідовностей *no_sequences* та третій цикл перебирає кадри всередині кожної послідовності. В середині циклів також відбувається виявлення рук, відображення ключових точок та витягування координат цих точок. Якщо це перший кадр (frame_num == 0), виводиться повідомлення про початок збору і відображається протягом 2 секунд. В іншому випадку виводиться інформація про збір кадрів. Далі витягуються значення ключових точок з результатів виявлення та зберігаються у файл формату .npy за вказаним шляхом.

```
cap = cv2.VideoCapture(0)
with mp_holistic.Holistic(min_detection_confidence=0.5,
                          min_tracking_confidence=0.5) as holistic:
    for action in actions:
        for sequence in range(no_sequences):
            for frame_num in range(sequence_length):
                ret, frame = cap.read()
                image, results = mediapipe_detection(frame, holistic)

                draw_styled_landmarks(image, results)

                if frame_num == 0:
                    cv2.putText(image, 'STARTING COLLECTION', (120,200),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4, cv2.LINE_AA)
                    cv2.putText(image, 'Collecting frames for {} Video Number {}'
                                .format(action, sequence), (15,12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,255, 0), 1, cv2.LINE_AA)
                    cv2.imshow('OpenCV Feed', image)
                    cv2.waitKey(2000)
                else:
                    cv2.putText(image, 'Collecting frames for {} Video Number {}'
                                .format(action, sequence), (15,12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,255, 0), 1, cv2.LINE_AA)
                    cv2.imshow('OpenCV Feed', image)

                keypoints = extract_keypoints(results)
                npy_path = os.path.join(DATA_PATH, action, str(sequence), str(frame_num))
                np.save(npy_path, keypoints)

                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break

    cap.release()
    cv2.destroyAllWindows()
```

Рисунок 4.7 – Реалізація збору даних

Таким чином, запускається модуль датасету, на екрані з’являється вікно з трансляцією з вебкамери (рис. 4.8). Зверху у вікні написана літера, для якої візуалізується жест, а також послідовний номер жесту. Для кожного класу записується 100 відео, кожне із яких розбивається на 30 кадрів. Зібравши 30 кадрів, трансляція зупиняється на 3 секунди, щоб встигнути змінити жест. З кожного кадру за допомогою Mediapipe витягуються координати точок рук. Всі орієнтири з зображення зберігаються у масив бібліотеки Numpy.

Після чого такі масиви з координатами орієнтирів для кожного кадра зберігаються у файли з розширенням .npy для майбутнього навчання моделі. Збір та організація даних таким чином дозволяє підготувати набір даних для подальшого використання у тренуванні моделі для розпізнавання жестів.

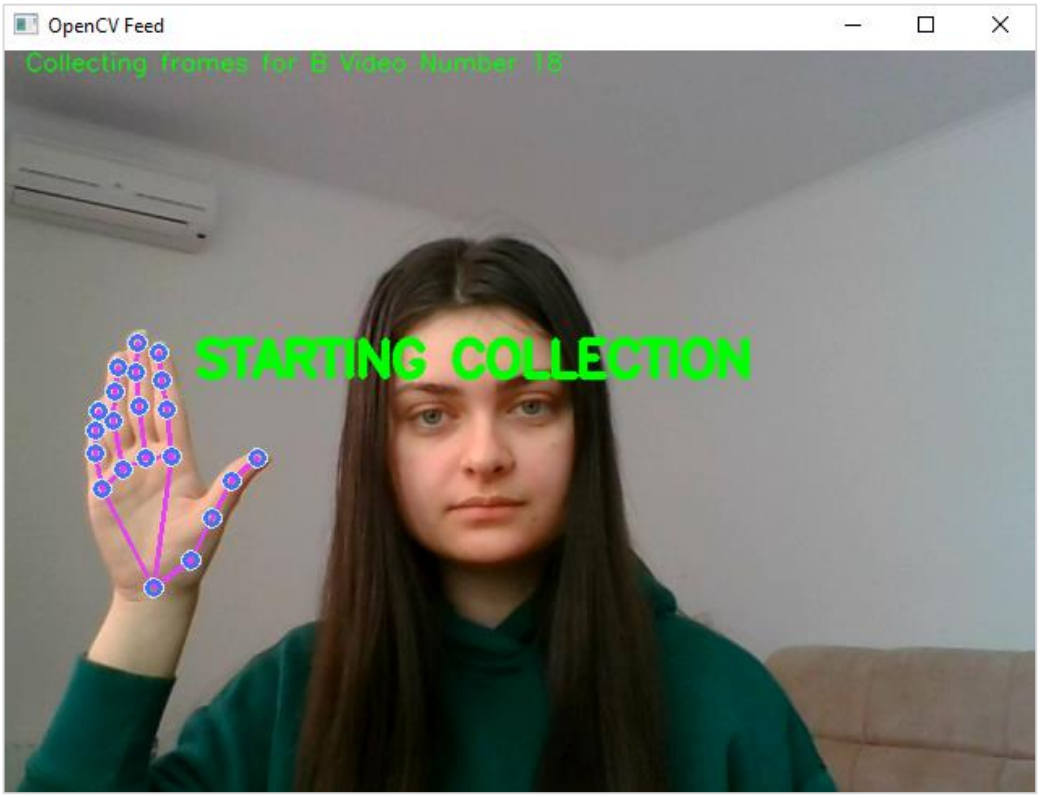

Рисунок 4.8 – Процес збору даних

Збережений датасет розташований у директорії з назвою USL_Dataset (рис. 4.9). У цьому каталозі знаходиться 20 папок, які відповідають різним класам даних. Кожна з цих папок містить 100 додаткових каталогів, представляючи різні жести. У межах кожної з цих папок розміщено 30 файлів з розширенням .npy, що містять масиви координат точок обох рук для кожного кадру.

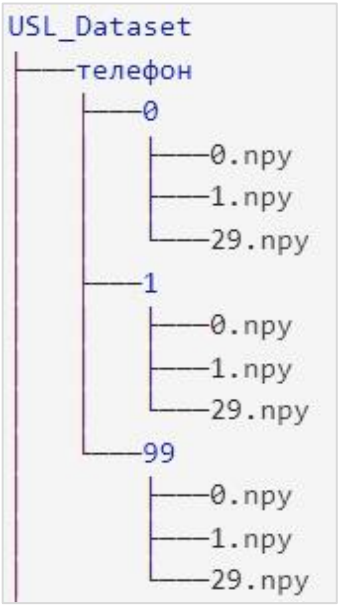

Рисунок 4.9 – Структура вмісту каталогу USL_Dataset на прикладі одного жесту

4.2 Підготовка даних для моделі розпізнавання жестів

Після завершення процесу збору даних, ключові точки, витягнуті з даних, структуруються за допомогою попередньої обробки даних. В ході даного етапу виконуються наступні операції.

1. Створення відображення міток. Використовуються бібліотеки *sklearn* та *tensorflow.keras* для підготовки даних перед навчанням моделі машинного навчання. Створює словник *label_map* (рис. 4.10), який встановлює відповідність між текстовим представленням жестів та числовими ідентифікаторами. Це важливо для присвоєння унікальної мітки кожній категорії. Мітки вказують на класи, до яких належать дані.

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

label_map = {label:num for num, label in enumerate(actions)}
```

Рисунок 4.10 – Створення відображення міток

2. Збір послідовностей кадрів та їх міток. Знову виконується цикл для кожної дії та її послідовностей, аналогічно до попереднього коду. Створюються послідовності кадрів та відповідні числові мітки (рис. 4.11).

```
sequences, labels = [], []
for action in actions:
    for sequence in np.array(os.listdir(os.path.join(DATA_PATH, action))).astype(int):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(DATA_PATH, action,
                                       str(sequence), "{}.npy".format(frame_num)))
            window.append(res)
        sequences.append(window)
        labels.append(label_map[action])
```

Рисунок 4.11 – Збір послідовностей кадрів та їх міток

3. Створення масивів. Дані структуруються таким чином, що всі масиви ключових точок кожного жесту зберігаються у вигляді масиву numpy (X), який потім зіставляється з іншим масивом міток (Y). Дана операція показана на рис. 4.12. Масив X містить вхідні дані (послідовності кадрів), а y – вихідні дані у форматі one-hot encoding (мітки класів), необхідні для тренування моделі розпізнавання жестів. Далі використовується функція *to_categorical* для перетворення Y у двійкову матрицю класів.

```
X = np.array(sequences)
y = to_categorical(labels).astype(int)
```

Рисунок 4.12 – Створення масивів для моделі

4. Поділ даних на тренувальний, валідаційний та тестовий набори. Після завершення попередньої обробки дані розділяються на навчальні, валідаційні та тестові (рис. 4.13) за допомогою функції *train_test_split*. Спочатку дані розділяються на тренувально-валідаційний (90%) та тестовий набори (10%). Потім зазначено, що 20% даних із тренувально-валідаційного набору будуть використані

для валідації, а 80% – для тренування. Такий поділ даних на тренувальний-валідаційний набір і тестовий набір з подальшим розділом тренувально-валідаційного набору на окремі тренувальний і валідаційний, відомий як двоетапний поділ (two-stage split), дозволяє здійснювати належне налаштування гіперпараметрів моделі та оцінювати її загальну ефективність. А також допомагає збалансувати процес тренування, налаштування та оцінювання моделі для забезпечення її оптимальної продуктивності на нових даних.

```
X_train_and_val, X_test, y_train_and_val,
y_test = train_test_split(X,y, train_size=0.9)
X_train, X_valid, y_train,
y_valid = train_test_split(X_train_and_val,y_train_and_val, test_size=0.2)
```

Рисунок 4.13 – Поділ даних

5. Перевірка розмірності масивів для навчального, валідаційного та тестового наборів.

Тренувальний набір (X_train): розмірність (shape): (1440, 30, 126). Це масив із 1440 елементів, кожен з яких представляє собою послідовність з 30 кадрів, кожен з яких має 126 ознак.

Валідаційний набір (X_valid): розмірність (shape): (360, 30, 126). Це масив із 360 елементів, кожен з яких також є послідовністю з 30 кадрів, кожен з яких має 126 ознак.

Тестовий набір (X_test): розмірність (shape): (200, 30, 126). Це масив із 200 елементів, кожен з яких представляє собою послідовність з 30 кадрів, кожен з яких має 126 ознак.

Мітки для тестового набору (y_test): розмірність (shape): (200, 20). Це означає, що є 200 тестових прикладів жестів, які використовуються для тестування моделі, яка буде класифікувати 20 класів.

4.3 Розробка, навчання та тестування моделі розпізнавання жестів

Створення моделі. Для розробки та навчання моделі глибокого навчання для початку потрібно імпортувати бібліотеку *Tensorflow*, модель *Sequential*, шари *LSTM* та *Dense*, інструмент для візуалізації та моніторингу тренування моделі в TensorFlow – *TensorBoard*, а також модуль *datetime* (рис. 4.14).

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import TensorBoard
import datetime
```

Рисунок 4.14 – Імпорт бібліотек

Далі визначаються необхідні параметри тренування (рис. 4.15):

– *batch_size*: кількість прикладів даних, які використовуються для одного оновлення (ітерації) в процесі тренування моделі;

– *epochs*: кількість повних проходів через весь набір даних під час тренування.

Також зазначається шлях до каталогу, де будуть зберігатися журнали TensorBoard. Створюється об'єкт *TensorBoard Callback* з передачею створеного шляху *log_dir*. Цей об'єкт буде використовуватися під час тренування моделі для збору інформації та збереження журналів, які потім можна візуалізувати за допомогою TensorBoard.

```
batch_size = 128
epochs = 100
log_dir = os.path.join('/content/Logs', datetime.datetime.now().strftime("%Y.%m.%d-%H.%M")
                       + '--batch__' + str(batch_size) + '--epochs__' + str(epochs))
tb_callback = TensorBoard(log_dir=log_dir)
```

Рисунок 4.15 – Визначення параметрів тренування та створення об’єкту TensorBoard Callback

Наступним кроком є створення моделі типу *Sequential* з бібліотеки Keras. Додаються три LSTM шари та три Dense шари з різною кількістю нейронів та функціями активації (ReLU і softmax).

Функцією активації вузла штучної нейронної мережі є функція, яка обчислює вихід вузла на основі його окремих входів та їхніх ваг [42]. ReLU (Rectified Linear Unit) – це функція активації, яка широко використовується в глибокому навчанні та нейронних мережах. Ця функція використовується для надання нелінійності моделі та активації певних нейронів у штучному нейронному шарі. Математично, функція ReLU визначається як [43]:

$$ReLU(x) = \max(0, x), \tag{4.1}$$

де x – вхідна величина або вагове сумарне значення вхідних сигналів.

Графічно ReLU виглядає як лінійна функція з нульовим відсіканням на осі абсцис у точці 0. Це означає, що функція має постійний нахил у всіх точках, крім точки 0, де відбувається відсікання (рис. 4.16).

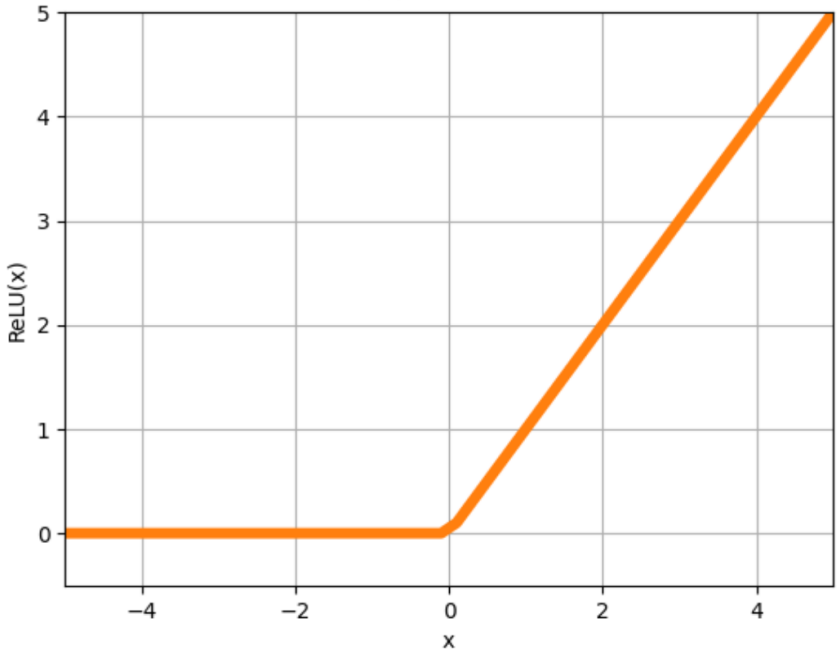

Рисунок 4.16 – Графічне представлення функції ReLU

Одна з основних переваг функції ReLU – простота обчислень та ефективність в тренуванні нейронних мереж. Функція повертає 0 для всіх від'ємних значень та саме значення для всіх додатних значень. Це призводить до того, що нейрони, які активуються, залишаються активованими з різницею відсутнього градієнту для від'ємних значень. Перевага ReLU полягає в тому, що вона сприяє уникненню проблеми зникаючого градієнту, яка виникає при використанні інших функцій активації, таких як сигмоїда або тангенс гіперболічний. Збереження додатних значень у вихідному потоці може полегшити тренування нейронних мереж, особливо глибоких моделей. Однак є й певні недоліки, такі як проблема «мертвих нейронів» (англ. dead neurons), коли нейрон завжди виводить 0 для всіх вхідних значень.

Функція активації softmax використовується для конвертації вектору числових значень в ймовірності, що додаються до 1. Ця функція часто використовується в останньому шарі нейронної мережі для класифікації, де кожен вихідний нейрон представляє ймовірність належності до певного класу. Математично, softmax визначається для вектору z з K елементів як:

$$softmax(z)_i = \frac{e^{z_i}}{\Sigma_{j=1}^{K} e^{z_i}}, \quad (4.2)$$

де e – це число Ейлера (приблизно 2.71828);

i – індекс елементу вектору.

Основна ідея softmax полягає в тому, щоб висвітлити найбільшу значущість серед усіх елементів та перетворити їх в ймовірності. Величина e^{z_i} збільшує значущість величин, а знаменник $\Sigma_{j=1}^{K} e^{z_i}$ гарантує, що ймовірності сумуються до 1.

Першим шаром нейронної мережі буде LSTM шар. Він складається з 64 нейронів, має функцію активації ReLU, вхідна форма даних шару – 30x126. Другий шар нейронної мережі – LSTM. Складається з 128 нейронів має функцію активації ReLU. Третій шар – LSTM. Складається з 64 нейронів має функцію активації ReLU.

Четвертий шар – повнозв’язний. Складається з 64 нейронів має функцію активації ReLU. П’ятий шар – повнозв’язний. Складається з 32 нейронів має функцію активації ReLU. Додається останній повнозв'язаний шар з кількістю нейронів, рівною кількості класів (20) у вхідному наборі даних. Використовується функція активації softmax для отримання ймовірностей класів. На рис. 4.17 зображено створену архітектуру моделі.

```
model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu',
               input_shape=(30,126)))
model.add(LSTM(128, return_sequences=True, activation='relu'))
model.add(LSTM(64, return_sequences=False, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(actions.shape[0], activation='softmax'))

model.compile(optimizer='Adam', loss='categorical_crossentropy',
              metrics=['categorical_accuracy'])
model.summary()
```

Рисунок 4.17 – Створена архітектура моделі

Компілюється модель з використанням оптимізатора Adam і функції втрат categorical_crossentropy для задачі багатокласової класифікації та метрики categorical_accuracy. Оптимізатори є важливим параметром нейронної мережі, оскільки вони можуть безперервно змінювати ваги та зміщення для досягнення найменшого значення функції втрат. Оптимізатор Adam – метод стохастичного градієнтного спуску. Серед безлічі методів оптимізації для глибокого навчання, оптимізатор Адама [44] є одним із найпопулярніших методів, який широко використовують для навчання різних нейронних мереж. Крім того, алгоритм має можливість обчислювати адаптивну швидкість навчання для кожного параметра, що дозволяє моделі прискорити збіжність і мінімізувати флуктуації. Висока обчислювальна ефективність і невеликий обсяг пам'яті зробили алгоритм Adam одним з найбільш часто використовуваних оптимізаторів [45, 46].

Стосовно функцій втрат, то їх мета полягає в тому, щоб обчислити величину, яку модель має прагнути мінімізувати в процесі навчання [47]. Функція втрат categorical_crossentropy часто використовується в задачах багатокласової класифікації, де кожен приклад може належати тільки одному з декількох можливих класів. Вона вимірює розрив між розподілом ймовірностей, який генерується моделлю, та фактичним розподілом класів.

Далі виводиться інформація про структуру та параметри кожного шару моделі (рис. 4.18).

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 64)            48896

 lstm_1 (LSTM)               (None, 30, 128)           98816

 lstm_2 (LSTM)               (None, 64)                49408

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 20)                660

=================================================================
Total params: 204020 (796.95 KB)
Trainable params: 204020 (796.95 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Рисунок 4.18 – Структура та параметри моделі для розпізнавання жестів

Таким чином, перший шар має 48896 параметрів, другий – 98816 параметрів, третій – 49408, четвертий – 4160, п’ятий – 2080 та останній шар має 660 параметрів.

Загальна інформація:

– *total params*: загальна кількість параметрів у моделі складає 204020 (приблизно 796.95 KB);

– *trainable params*: кількість параметрів, які можна навчити під час тренування, також 204020;

– *non-trainable params*: у даному випадку відсутні, що означає, що всі параметри моделі є навчальними.

Навчання моделі. Наступним кроком є етап тренування створеної моделі, використовуючи функцію *fit()* із бібліотеки Keras. Використання даного методу полягає у налаштуванні моделі на основі тренувальних даних, шляхом визначення оптимальних параметрів ваг і зсувів за допомогою оптимізаційних алгоритмів. Даний метод є одним із способів навчання з учителем, де використовуються мітки класів для тренування моделі. Під час навчання модель пристосовується до вхідних даних, набуває досвіду та навчається робити прогнози на основі здобутих знань. Основні налаштування функції fit() наведено нижче:

1) X_train: навчальні вхідні дані;
2) y_train: навчальні цільові дані (відповіді);
3) epochs: кількість епох тренування, тобто скільки разів модель буде навчатися на всьому наборі даних;
4) shuffle = True: вказує, чи потрібно перемішати дані перед кожною епохою тренування. Це може поліпшити якість тренування;
5) batch_size: кількість зразків, які використовуються для одного оновлення ваг моделі. Більші значення можуть пришвидшити тренування, але можуть вимагати більше пам'яті;
6) validation_data = (X_valid, y_valid): дані для валідації моделі під час тренування. Ці дані використовуються для оцінки продуктивності моделі під час кожної епохи, але не використовуються для самого тренування;
7) callbacks = [tb_callback]: список зворотних викликів (callbacks), які можуть викликатися під час тренування. У даному випадку використовується tb_callback для запису інформації для TensorBoard.

На рис. 4.19 – 4.25 представлено логи навчання, що містять інформацію, яка виводиться під час тренування моделі та служить для моніторингу її продуктивності.

Основні метрики, які виводяться під час навчання.

1. *Loss* (втрата) – числове значення, яке визначає, наскільки добре модель працю під час навчання на конкретному наборі даних. Зазвичай використовується функція втрати (loss function), яка порівнює вихід моделі з правильними відповідями та обчислює різницю. Мета під час навчання полягає в тому, щоб зменшити цю втрату.

2. *Categorical Accuracy* (категоріальна точність) – це метрика, яка визначає точність класифікації для задачі з багатьма класами. Вона вимірює відсоток правильно класифікованих прикладів.

3. *Val_loss* (валідаційна втрата) – це значення втрати на валідаційному наборі даних. Під час навчання моделі часто використовують два набори даних: тренувальний і валідаційний. Тренувальний використовується для самого процесу навчання, а валідаційний – для оцінки ефективності моделі на даних, які вона не бачила під час навчання. Val_loss вказує на те, як добре модель справляється з валідаційним набором даних.

4. *Val_categorical_accuracy* (валідаційна категоріальна точність) – це аналогічно категоріальній точності, але обчислюється на валідаційному наборі даних. Вказує на точність класифікації моделі на даних, які не брали участь у процесі навчання.

Ці метрики є ключовими для оцінки продуктивності моделі. Вони вказують на те, наскільки добре модель навчилася розпізнавати класи на тренувальних та валідаційних даних.

```
history = model.fit(X_train, y_train, epochs=epochs, shuffle=True, batch_size=batch_size, validation_data=(X_valid, y_valid), callbacks=[tb_callback])

Epoch 1/100
12/12 [==============================] - 6s 167ms/step - loss: 2.9459 - categorical_accuracy: 0.0750 - val_loss: 2.8682 - val_categorical_accuracy: 0.1139
Epoch 2/100
12/12 [==============================] - 2s 170ms/step - loss: 2.7175 - categorical_accuracy: 0.1437 - val_loss: 2.3940 - val_categorical_accuracy: 0.2306
Epoch 3/100
12/12 [==============================] - 2s 207ms/step - loss: 2.4973 - categorical_accuracy: 0.2604 - val_loss: 4.0401 - val_categorical_accuracy: 0.0694
Epoch 4/100
12/12 [==============================] - 1s 118ms/step - loss: 2.7960 - categorical_accuracy: 0.1354 - val_loss: 2.6516 - val_categorical_accuracy: 0.1389
Epoch 5/100
12/12 [==============================] - 1s 119ms/step - loss: 2.5200 - categorical_accuracy: 0.1639 - val_loss: 2.3810 - val_categorical_accuracy: 0.1556
Epoch 6/100
12/12 [==============================] - 1s 121ms/step - loss: 2.2184 - categorical_accuracy: 0.2424 - val_loss: 2.0318 - val_categorical_accuracy: 0.2361
Epoch 7/100
12/12 [==============================] - 2s 127ms/step - loss: 1.9139 - categorical_accuracy: 0.3257 - val_loss: 1.8812 - val_categorical_accuracy: 0.3278
Epoch 8/100
12/12 [==============================] - 2s 126ms/step - loss: 1.7915 - categorical_accuracy: 0.3986 - val_loss: 1.8441 - val_categorical_accuracy: 0.3861
Epoch 9/100
12/12 [==============================] - 1s 120ms/step - loss: 1.7528 - categorical_accuracy: 0.3819 - val_loss: 1.6806 - val_categorical_accuracy: 0.3083
Epoch 10/100
12/12 [==============================] - 2s 130ms/step - loss: 1.4627 - categorical_accuracy: 0.4549 - val_loss: 1.3632 - val_categorical_accuracy: 0.4611
Epoch 11/100
12/12 [==============================] - 3s 227ms/step - loss: 1.2983 - categorical_accuracy: 0.5340 - val_loss: 1.5391 - val_categorical_accuracy: 0.4306
Epoch 12/100
12/12 [==============================] - 2s 140ms/step - loss: 1.3429 - categorical_accuracy: 0.4951 - val_loss: 1.4750 - val_categorical_accuracy: 0.5083
Epoch 13/100
12/12 [==============================] - 1s 120ms/step - loss: 1.2599 - categorical_accuracy: 0.5743 - val_loss: 1.1770 - val_categorical_accuracy: 0.5833
Epoch 14/100
12/12 [==============================] - 1s 119ms/step - loss: 1.0864 - categorical_accuracy: 0.6201 - val_loss: 1.1628 - val_categorical_accuracy: 0.6167
Epoch 15/100
12/12 [==============================] - 1s 119ms/step - loss: 0.8833 - categorical_accuracy: 0.6708 - val_loss: 1.1015 - val_categorical_accuracy: 0.5944
```

Рисунок 4.19 – Логи навчання моделі (епоха 1-15)

```
Epoch 16/100
12/12 [==============================] - 1s 123ms/step - loss: 0.7404 - categorical_accuracy: 0.7194 - val_loss: 0.8509 - val_categorical_accuracy: 0.7083
Epoch 17/100
12/12 [==============================] - 1s 120ms/step - loss: 0.7353 - categorical_accuracy: 0.7472 - val_loss: 0.9903 - val_categorical_accuracy: 0.6250
Epoch 18/100
12/12 [==============================] - 2s 126ms/step - loss: 0.7963 - categorical_accuracy: 0.7271 - val_loss: 0.8102 - val_categorical_accuracy: 0.7361
Epoch 19/100
12/12 [==============================] - 2s 207ms/step - loss: 0.5786 - categorical_accuracy: 0.7861 - val_loss: 0.7923 - val_categorical_accuracy: 0.7444
Epoch 20/100
12/12 [==============================] - 2s 173ms/step - loss: 0.6123 - categorical_accuracy: 0.7958 - val_loss: 0.7592 - val_categorical_accuracy: 0.7472
Epoch 21/100
12/12 [==============================] - 1s 117ms/step - loss: 0.6193 - categorical_accuracy: 0.7861 - val_loss: 0.6279 - val_categorical_accuracy: 0.7833
Epoch 22/100
12/12 [==============================] - 1s 124ms/step - loss: 0.5300 - categorical_accuracy: 0.8160 - val_loss: 1.2121 - val_categorical_accuracy: 0.6694
Epoch 23/100
12/12 [==============================] - 1s 120ms/step - loss: 0.7186 - categorical_accuracy: 0.7563 - val_loss: 0.6653 - val_categorical_accuracy: 0.7861
Epoch 24/100
12/12 [==============================] - 1s 120ms/step - loss: 0.4771 - categorical_accuracy: 0.8347 - val_loss: 0.5834 - val_categorical_accuracy: 0.8472
Epoch 25/100
12/12 [==============================] - 1s 114ms/step - loss: 0.4198 - categorical_accuracy: 0.8347 - val_loss: 0.7538 - val_categorical_accuracy: 0.8194
Epoch 26/100
12/12 [==============================] - 1s 111ms/step - loss: 0.4349 - categorical_accuracy: 0.8438 - val_loss: 0.5048 - val_categorical_accuracy: 0.8250
Epoch 27/100
12/12 [==============================] - 2s 175ms/step - loss: 0.3195 - categorical_accuracy: 0.8958 - val_loss: 0.4782 - val_categorical_accuracy: 0.8639
Epoch 28/100
12/12 [==============================] - 3s 215ms/step - loss: 0.2755 - categorical_accuracy: 0.8972 - val_loss: 0.5606 - val_categorical_accuracy: 0.8083
Epoch 29/100
12/12 [==============================] - 1s 117ms/step - loss: 0.3127 - categorical_accuracy: 0.8868 - val_loss: 0.4276 - val_categorical_accuracy: 0.8750
Epoch 30/100
12/12 [==============================] - 1s 123ms/step - loss: 0.2820 - categorical_accuracy: 0.8917 - val_loss: 0.3925 - val_categorical_accuracy: 0.8806
```

Рисунок 4.20 – Логи навчання (епоха 16-30)

```
Epoch 31/100
12/12 [==============================] - 1s 121ms/step - loss: 0.2068 - categorical_accuracy: 0.9285 - val_loss: 0.4064 - val_categorical_accuracy: 0.8889
Epoch 32/100
12/12 [==============================] - 2s 150ms/step - loss: 0.2376 - categorical_accuracy: 0.9153 - val_loss: 0.5636 - val_categorical_accuracy: 0.8389
Epoch 33/100
12/12 [==============================] - 3s 217ms/step - loss: 0.2046 - categorical_accuracy: 0.9312 - val_loss: 0.6744 - val_categorical_accuracy: 0.7889
Epoch 34/100
12/12 [==============================] - 2s 172ms/step - loss: 0.3308 - categorical_accuracy: 0.8757 - val_loss: 0.3520 - val_categorical_accuracy: 0.8806
Epoch 35/100
12/12 [==============================] - 3s 227ms/step - loss: 0.2458 - categorical_accuracy: 0.9097 - val_loss: 0.3896 - val_categorical_accuracy: 0.8972
Epoch 36/100
12/12 [==============================] - 1s 122ms/step - loss: 0.1982 - categorical_accuracy: 0.9306 - val_loss: 0.2953 - val_categorical_accuracy: 0.9028
Epoch 37/100
12/12 [==============================] - 1s 124ms/step - loss: 0.5251 - categorical_accuracy: 0.8569 - val_loss: 0.6153 - val_categorical_accuracy: 0.8194
Epoch 38/100
12/12 [==============================] - 1s 124ms/step - loss: 0.3292 - categorical_accuracy: 0.8882 - val_loss: 0.3997 - val_categorical_accuracy: 0.8917
Epoch 39/100
12/12 [==============================] - 1s 118ms/step - loss: 0.1971 - categorical_accuracy: 0.9333 - val_loss: 0.3252 - val_categorical_accuracy: 0.9306
Epoch 40/100
12/12 [==============================] - 2s 170ms/step - loss: 0.1766 - categorical_accuracy: 0.9424 - val_loss: 0.5708 - val_categorical_accuracy: 0.8306
Epoch 41/100
12/12 [==============================] - 1s 122ms/step - loss: 0.2447 - categorical_accuracy: 0.9062 - val_loss: 0.3705 - val_categorical_accuracy: 0.9000
Epoch 42/100
12/12 [==============================] - 2s 188ms/step - loss: 0.1881 - categorical_accuracy: 0.9410 - val_loss: 0.3669 - val_categorical_accuracy: 0.9111
Epoch 43/100
12/12 [==============================] - 2s 203ms/step - loss: 0.1731 - categorical_accuracy: 0.9403 - val_loss: 0.3153 - val_categorical_accuracy: 0.9417
Epoch 44/100
12/12 [==============================] - 1s 123ms/step - loss: 0.2139 - categorical_accuracy: 0.9257 - val_loss: 0.3256 - val_categorical_accuracy: 0.9306
Epoch 45/100
12/12 [==============================] - 1s 117ms/step - loss: 0.1789 - categorical_accuracy: 0.9396 - val_loss: 0.2568 - val_categorical_accuracy: 0.9306
```

Рисунок 4.21 – Логи навчання (епоха 31-45)

```
Epoch 46/100
12/12 [==============================] - 1s 118ms/step - loss: 0.1335 - categorical_accuracy: 0.9542 - val_loss: 0.2605 - val_categorical_accuracy: 0.9583
Epoch 47/100
12/12 [==============================] - 1s 119ms/step - loss: 0.0988 - categorical_accuracy: 0.9660 - val_loss: 0.3447 - val_categorical_accuracy: 0.9583
Epoch 48/100
12/12 [==============================] - 1s 123ms/step - loss: 0.1038 - categorical_accuracy: 0.9632 - val_loss: 0.4568 - val_categorical_accuracy: 0.9083
Epoch 49/100
12/12 [==============================] - 1s 122ms/step - loss: 0.0949 - categorical_accuracy: 0.9632 - val_loss: 0.2366 - val_categorical_accuracy: 0.9500
Epoch 50/100
12/12 [==============================] - 2s 162ms/step - loss: 0.1324 - categorical_accuracy: 0.9535 - val_loss: 0.4799 - val_categorical_accuracy: 0.9056
Epoch 51/100
12/12 [==============================] - 3s 215ms/step - loss: 0.2362 - categorical_accuracy: 0.9201 - val_loss: 0.3177 - val_categorical_accuracy: 0.9306
Epoch 52/100
12/12 [==============================] - 2s 119ms/step - loss: 0.1609 - categorical_accuracy: 0.9444 - val_loss: 0.3029 - val_categorical_accuracy: 0.9417
Epoch 53/100
12/12 [==============================] - 1s 118ms/step - loss: 0.1609 - categorical_accuracy: 0.9431 - val_loss: 0.2980 - val_categorical_accuracy: 0.9250
Epoch 54/100
12/12 [==============================] - 1s 115ms/step - loss: 0.1224 - categorical_accuracy: 0.9590 - val_loss: 0.2575 - val_categorical_accuracy: 0.9444
Epoch 55/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0912 - categorical_accuracy: 0.9681 - val_loss: 0.3030 - val_categorical_accuracy: 0.9500
Epoch 56/100
12/12 [==============================] - 1s 124ms/step - loss: 0.0760 - categorical_accuracy: 0.9722 - val_loss: 0.3004 - val_categorical_accuracy: 0.9444
Epoch 57/100
12/12 [==============================] - 1s 118ms/step - loss: 0.0449 - categorical_accuracy: 0.9833 - val_loss: 0.2876 - val_categorical_accuracy: 0.9472
Epoch 58/100
12/12 [==============================] - 1s 122ms/step - loss: 0.1494 - categorical_accuracy: 0.9528 - val_loss: 0.3735 - val_categorical_accuracy: 0.9000
Epoch 59/100
12/12 [==============================] - 3s 238ms/step - loss: 0.1489 - categorical_accuracy: 0.9507 - val_loss: 0.3698 - val_categorical_accuracy: 0.9083
Epoch 60/100
12/12 [==============================] - 2s 144ms/step - loss: 0.0919 - categorical_accuracy: 0.9667 - val_loss: 0.2433 - val_categorical_accuracy: 0.9472
```

Рисунок 4.22 – Логи навчання (епоха 46-60)

```
Epoch 61/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0788 - categorical_accuracy: 0.9736 - val_loss: 0.2449 - val_categorical_accuracy: 0.9500
Epoch 62/100
12/12 [==============================] - 1s 116ms/step - loss: 0.0510 - categorical_accuracy: 0.9826 - val_loss: 0.2534 - val_categorical_accuracy: 0.9556
Epoch 63/100
12/12 [==============================] - 1s 119ms/step - loss: 0.0611 - categorical_accuracy: 0.9778 - val_loss: 0.4858 - val_categorical_accuracy: 0.9556
Epoch 64/100
12/12 [==============================] - 1s 123ms/step - loss: 0.0611 - categorical_accuracy: 0.9792 - val_loss: 0.1535 - val_categorical_accuracy: 0.9611
Epoch 65/100
12/12 [==============================] - 1s 118ms/step - loss: 0.0595 - categorical_accuracy: 0.9750 - val_loss: 0.1720 - val_categorical_accuracy: 0.9556
Epoch 66/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0749 - categorical_accuracy: 0.9750 - val_loss: 0.1908 - val_categorical_accuracy: 0.9556
Epoch 67/100
12/12 [==============================] - 3s 221ms/step - loss: 0.0424 - categorical_accuracy: 0.9847 - val_loss: 0.2106 - val_categorical_accuracy: 0.9583
Epoch 68/100
12/12 [==============================] - 2s 172ms/step - loss: 0.0759 - categorical_accuracy: 0.9736 - val_loss: 0.1613 - val_categorical_accuracy: 0.9556
Epoch 69/100
12/12 [==============================] - 1s 125ms/step - loss: 0.0615 - categorical_accuracy: 0.9778 - val_loss: 0.1860 - val_categorical_accuracy: 0.9556
Epoch 70/100
12/12 [==============================] - 1s 119ms/step - loss: 0.0348 - categorical_accuracy: 0.9854 - val_loss: 0.2168 - val_categorical_accuracy: 0.9694
Epoch 71/100
12/12 [==============================] - 1s 124ms/step - loss: 0.0330 - categorical_accuracy: 0.9861 - val_loss: 0.1840 - val_categorical_accuracy: 0.9639
Epoch 72/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0323 - categorical_accuracy: 0.9868 - val_loss: 0.3150 - val_categorical_accuracy: 0.9417
Epoch 73/100
12/12 [==============================] - 1s 117ms/step - loss: 0.0313 - categorical_accuracy: 0.9868 - val_loss: 0.1994 - val_categorical_accuracy: 0.9694
Epoch 74/100
12/12 [==============================] - 1s 118ms/step - loss: 0.0430 - categorical_accuracy: 0.9854 - val_loss: 0.3057 - val_categorical_accuracy: 0.9361
Epoch 75/100
12/12 [==============================] - 2s 193ms/step - loss: 0.0822 - categorical_accuracy: 0.9722 - val_loss: 0.2395 - val_categorical_accuracy: 0.9333
Epoch 76/100
12/12 [==============================] - 2s 191ms/step - loss: 0.0582 - categorical_accuracy: 0.9840 - val_loss: 0.2127 - val_categorical_accuracy: 0.9583
```

Рисунок 4.23 – Логи навчання (епоха 61-76)

```
Epoch 77/100
12/12 [==============================] - 1s 119ms/step - loss: 0.0622 - categorical_accuracy: 0.9806 - val_loss: 0.2535 - val_categorical_accuracy: 0.9444
Epoch 78/100
12/12 [==============================] - 1s 121ms/step - loss: 0.0636 - categorical_accuracy: 0.9778 - val_loss: 0.2239 - val_categorical_accuracy: 0.9500
Epoch 79/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0468 - categorical_accuracy: 0.9812 - val_loss: 0.2426 - val_categorical_accuracy: 0.9583
Epoch 80/100
12/12 [==============================] - 1s 119ms/step - loss: 0.0291 - categorical_accuracy: 0.9910 - val_loss: 0.2854 - val_categorical_accuracy: 0.9472
Epoch 81/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0469 - categorical_accuracy: 0.9812 - val_loss: 0.2024 - val_categorical_accuracy: 0.9528
Epoch 82/100
12/12 [==============================] - 1s 119ms/step - loss: 0.1837 - categorical_accuracy: 0.9479 - val_loss: 0.2623 - val_categorical_accuracy: 0.9556
Epoch 83/100
12/12 [==============================] - 2s 173ms/step - loss: 0.0960 - categorical_accuracy: 0.9757 - val_loss: 0.2573 - val_categorical_accuracy: 0.9694
Epoch 84/100
12/12 [==============================] - 3s 227ms/step - loss: 0.0714 - categorical_accuracy: 0.9757 - val_loss: 0.2076 - val_categorical_accuracy: 0.9611
Epoch 85/100
12/12 [==============================] - 1s 117ms/step - loss: 0.0487 - categorical_accuracy: 0.9785 - val_loss: 0.4022 - val_categorical_accuracy: 0.9222
Epoch 86/100
12/12 [==============================] - 1s 118ms/step - loss: 0.0810 - categorical_accuracy: 0.9722 - val_loss: 0.3162 - val_categorical_accuracy: 0.9250
Epoch 87/100
12/12 [==============================] - 1s 117ms/step - loss: 0.0778 - categorical_accuracy: 0.9736 - val_loss: 0.1565 - val_categorical_accuracy: 0.9611
Epoch 88/100
12/12 [==============================] - 1s 118ms/step - loss: 0.0514 - categorical_accuracy: 0.9806 - val_loss: 0.2297 - val_categorical_accuracy: 0.9528
Epoch 89/100
12/12 [==============================] - 1s 121ms/step - loss: 0.1432 - categorical_accuracy: 0.9604 - val_loss: 0.2177 - val_categorical_accuracy: 0.9306
Epoch 90/100
12/12 [==============================] - 1s 120ms/step - loss: 0.1528 - categorical_accuracy: 0.9542 - val_loss: 0.1669 - val_categorical_accuracy: 0.9556
Epoch 91/100
12/12 [==============================] - 1s 122ms/step - loss: 0.0478 - categorical_accuracy: 0.9889 - val_loss: 0.1203 - val_categorical_accuracy: 0.9861
Epoch 92/100
12/12 [==============================] - 3s 217ms/step - loss: 0.0155 - categorical_accuracy: 0.9958 - val_loss: 0.1743 - val_categorical_accuracy: 0.9694
Epoch 93/100
12/12 [==============================] - 2s 152ms/step - loss: 0.0143 - categorical_accuracy: 0.9951 - val_loss: 0.1833 - val_categorical_accuracy: 0.9778
```

Рисунок 4.24 – Логи навчання (епоха 77-93)

```
Epoch 94/100
12/12 [==============================] - 1s 116ms/step - loss: 0.0106 - categorical_accuracy: 0.9972 - val_loss: 0.2421 - val_categorical_accuracy: 0.9667
Epoch 95/100
12/12 [==============================] - 1s 120ms/step - loss: 0.0140 - categorical_accuracy: 0.9965 - val_loss: 0.1571 - val_categorical_accuracy: 0.9583
Epoch 96/100
12/12 [==============================] - 1s 124ms/step - loss: 0.0300 - categorical_accuracy: 0.9875 - val_loss: 0.1763 - val_categorical_accuracy: 0.9667
Epoch 97/100
12/12 [==============================] - 1s 115ms/step - loss: 0.0249 - categorical_accuracy: 0.9917 - val_loss: 0.1829 - val_categorical_accuracy: 0.9611
Epoch 98/100
12/12 [==============================] - 1s 118ms/step - loss: 0.0235 - categorical_accuracy: 0.9931 - val_loss: 0.1118 - val_categorical_accuracy: 0.9806
Epoch 99/100
12/12 [==============================] - 1s 119ms/step - loss: 0.0145 - categorical_accuracy: 0.9937 - val_loss: 0.0918 - val_categorical_accuracy: 0.9778
Epoch 100/100
12/12 [==============================] - 2s 192ms/step - loss: 0.0106 - categorical_accuracy: 0.9944 - val_loss: 0.1202 - val_categorical_accuracy: 0.9833
```

Рисунок 4.25 – Логи навчання (епоха 94-100)

Метод save() використовується для збереження стану навченої моделі у файл “usl_model.h5” (рис. 4.26).

```
save_model = "usl_model.h5"
model.save(save_model)
print("Model Saved into", save_model)
```

Рисунок 4.26 – Збереження моделі

Тестування та оцінювання навченої моделі. На рис. 4.27 зображено код, який використовується для оцінки продуктивності моделі на тестових даних та для генерації передбачень для 20 зразків.

```
print("Evaluate on test data")
results = model.evaluate(X_test, y_test, batch_size=1)
print("test loss, test acc:", results)

print("Generate predictions for 20 samples")
predictions = model.predict(X_test)
print("predictions shape:", predictions.shape)
```

Рисунок 4.27 – Оцінка продуктивності та прогнозування на тестових даних

На рис. 4.28 показано отримані показники втрати та точності моделі на тестових даних в результаті навчання на 50 епохах. Отримані результати свідчать про те, що модель демонструє високу точність (95.5%) та низьку втрату (0.1299) на тестових даних.

```
Evaluate on test data
200/200 [==============================] - 4s 19ms/step - loss: 0.1300 - categorical_accuracy: 0.9550
test loss, test acc: [0.1299944818019867, 0.9549999833106995]
Generate predictions for 20 samples
7/7 [==============================] - 1s 21ms/step
predictions shape: (200, 20)
```

Рисунок 4.28 – Показники втрати та точності на тестових даних

На рис. 4.29 зображено побудовані графіки точності та втрат моделі.

Рисунок 4.29 – Графіки точності і втрат моделі (при 50 епохах)

Точність навчання та валідації. Синя лінія показує точність навчання, тобто роботу моделі на навчальному наборі даних. Помаранчева лінія показує точність валідації, тобто роботу моделі на окремому наборі даних, який не використовувався під час навчання. При навчанні за 50 епох точність різко зростає, а потім стабілізується на рівні 0.9 (90%) як для навчання, так і для валідації.

Втрати при навчанні та валідації. Синя лінія показує втрати при навчанні, які вимірюють, наскільки добре працює модель з точки зору помилок на навчальному наборі даних. Менші втрати свідчать про кращу продуктивність.

Помаранчева лінія показує втрати при валідації. Втрати під час навчання різко знижуються і вирівнюються близько 0,2, а втрати при перевірці повторюють втрати під час навчання з більш значними коливаннями.

Обидва графіки є поширеними способами візуалізації процесу навчання моделі в машинному навчанні, допомагаючи визначити, коли модель навчається ефективно, коли вона може бути надмірно пристосована, або коли вона припинила вдосконалюватися.

Побудова матриці невідповідностей. Матриця невідповідностей (анг. confusion matrix) є інструментом для оцінки продуктивності моделі машинного навчання, особливо у задачах класифікації. Вона дозволяє зрозуміти, наскільки ефективно модель класифікує об'єкти на різні класи. Аналіз матриці невідповідностей допомагає зрозуміти сильні та слабкі сторони моделі та може служити основою для подальших вдосконалень. Взаємозв'язок між прогнозами позитивного та негативного класу можна представити у вигляді 2x2 матриці невідповідностей, в якій показано, до якої з чотирьох категорій належать прогнози.

1. Істинно-позитивна (англ. True Positive, TP): об'єкти, які модель правильно визнала як позитивні.
2. Істинно-негативна (англ. True Negative, TN): об'єкти, які модель правильно визнала як негативні.
3. Хибно-позитивна (англ. False Positive, FP): об'єкти, які модель помилково визнала як позитивні.
4. Хибно-негативна (англ. False Negative, FN): об'єкти, які модель помилково визнала як негативні.

По цим елементам можна обчислити різні метрики, такі як точність (accuracy), влучність (precision), повнота (recall) та F1-міра.

Точність – це відношення правильно класифікованих об'єктів (TP та TN) до загальної кількості об'єктів. Ця метрика вимірює загальну точність моделі у класифікації об'єктів на всіх класах.

$$Accuracy = \frac{\mathrm{T}P+\mathrm{T}N}{\mathrm{T}P+TN+FP+FN}, \quad (4.3)$$

де TP, TN, FP та FN – кількість випадків, коли прогнози моделі потрапляють у кожну з відповідних категорій.

Влучність є відношенням правильно класифікованих позитивних прикладів до загальної кількості прогнозів, які були визначені як позитивні. Влучність вимірює точність визначення позитивних класів моделлю.

$$Precision = \frac{\mathrm{T}P}{\mathrm{T}P+FP}, \quad (4.4)$$

де TP та FP – кількість випадків, коли прогнози моделі потрапляють у кожну з відповідних категорій.

Повнота – відношення правильно класифікованих позитивних прикладів (TP) до загальної кількості реальних позитивних об'єктів (TP + FN). Повнота вимірює ефективність виявлення всіх позитивних об'єктів моделлю.

$$Recall = \frac{\mathrm{T}P}{\mathrm{T}P+FN}, \quad (4.5)$$

де TP та FN – випадки, коли прогнози моделі потрапляють у кожну з відповідних категорій.

F1-міра (англ. F1-score) – це комплексна метрика ефективності моделі, яка об'єднує точність і повноту в єдине число, використовуючи середнє гармонійне. Ця оцінка надає збалансовану міру між точністю та повнотою, допомагаючи зрозуміти, наскільки модель вдало визначає об'єкти певного класу.

При використанні середнього гармонійного замість звичайного середнього арифметичного, F-міра надає важливість меншому значенню з двох. Це означає, що навіть якщо одна з метрик (точність або повнота) низька, F-міра буде невисокою, якщо обидві метрики не набули високого значення. Формула F-міри виглядає

наступним чином:

$$F - measure = \frac{2*precision*recall}{precision+recall} = \frac{2*TP}{2*TP+FP+FN}, \quad (4.6)$$

де $precision$ – відношення правильно визначених позитивних прогнозів до загальної кількості позитивних прогнозів;

$recall$- відношення правильно визначених позитивних прогнозів до загальної кількості реальних позитивних об'єктів.

На рис. 4.30 показано побудовану матрицю помилок.

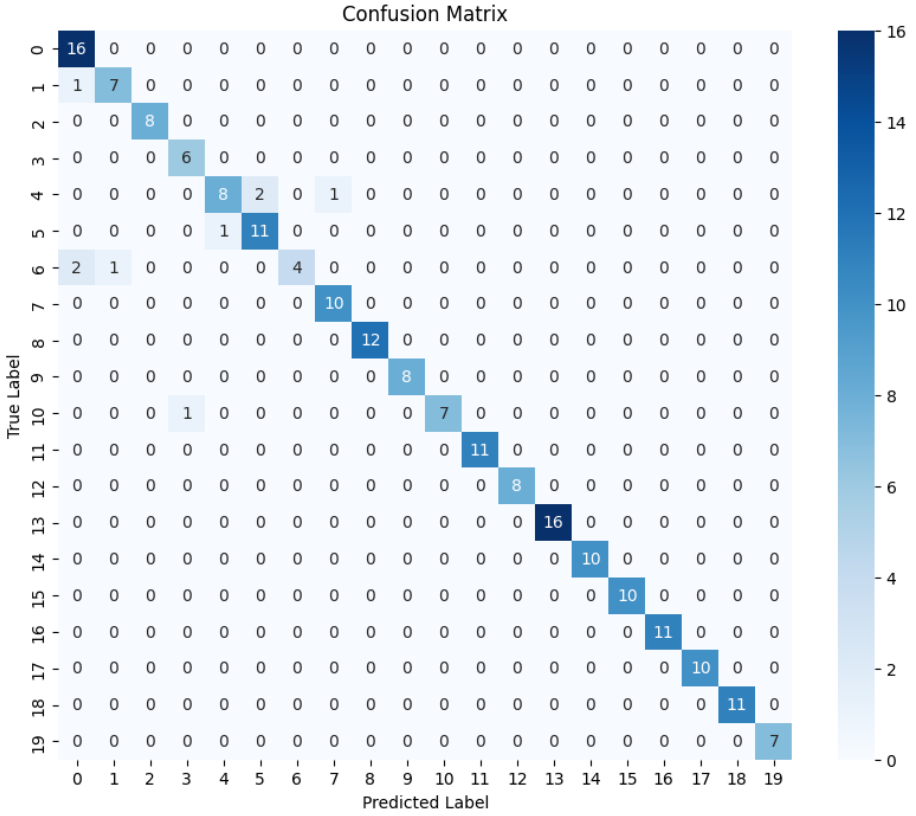

Рисунок 4.30 – Матриця невідповідностей (при 50 епохах)

Звіт про класифікацію (рис. 4.31), демонструє загальну точність моделі 95%, що є досить гарним показником. Макро- та середньозважені показники також високі. Однак для деяких класів показники precision, recall, F1-score дещо менші.

Classification Report:

	precision	recall	f1-score	support
І	0.84	1.00	0.91	16
А	0.88	0.88	0.88	8
В	1.00	1.00	1.00	8
Д	0.86	1.00	0.92	6
Е	0.89	0.73	0.80	11
С	0.85	0.92	0.88	12
У	1.00	0.57	0.73	7
Ц	0.91	1.00	0.95	10
Ь	1.00	1.00	1.00	12
будинок	1.00	1.00	1.00	8
дванадцять	1.00	0.88	0.93	8
добрийдень	1.00	1.00	1.00	11
записатисьнаприйом	1.00	1.00	1.00	8
лікар	1.00	1.00	1.00	16
сьогодні	1.00	1.00	1.00	10
телефон	1.00	1.00	1.00	10
технологія	1.00	1.00	1.00	11
хотіти	1.00	1.00	1.00	10
я	1.00	1.00	1.00	11
яквашісправи	1.00	1.00	1.00	7
accuracy			0.95	200
macro avg	0.96	0.95	0.95	200
weighted avg	0.96	0.95	0.95	200

Рисунок 4.31 – Розраховані значення метрик для кожного класу (при 50 епохах)

Спробуємо навчити модель на більшій кількості епох, наприклад, 100. Додавання графіків точності та втрат для різної кількості епох допоможе визначити, чи досягнута моделлю збалансована точність. На рис. 4.32 показано виведені показники втрат та точності на тестових даних.

```
Evaluate on test data
200/200 [==============================] - 3s 16ms/step - loss: 0.2039 - categorical_accuracy: 0.9750
test loss, test acc: [0.2038964480161667, 0.9750000238418579]
Generate predictions for 20 samples
7/7 [==============================] - 0s 16ms/step
predictions shape: (200, 20)
```

Рисунок 4.32 – Показники втрати та точності на тестових даних

Отримані результати свідчать про те, що модель демонструє високу точність (97.5%) та низьку втрату (0.2039) на тестових даних. Це може вказувати на те, що модель непогано впоралася з тестовим набором даних і добре узгоджується з новими, раніше не баченими даними. На рис. 4.33 зображено використання моделі для здійснення передбачень на тестових даних.

Також побудовано графіки точності та втрат навченої моделі (рис. 4.34).

```
res = model.predict(X_test)
actions[np.argmax(res[4])]
```

```
7/7 [==============================] - 0s 21ms/step
'будинок'
```

```
actions[np.argmax(y_test[4])]
```

```
'будинок'
```

Рисунок 4.33 – Порівняння тестових даних з прогнозованим результатом

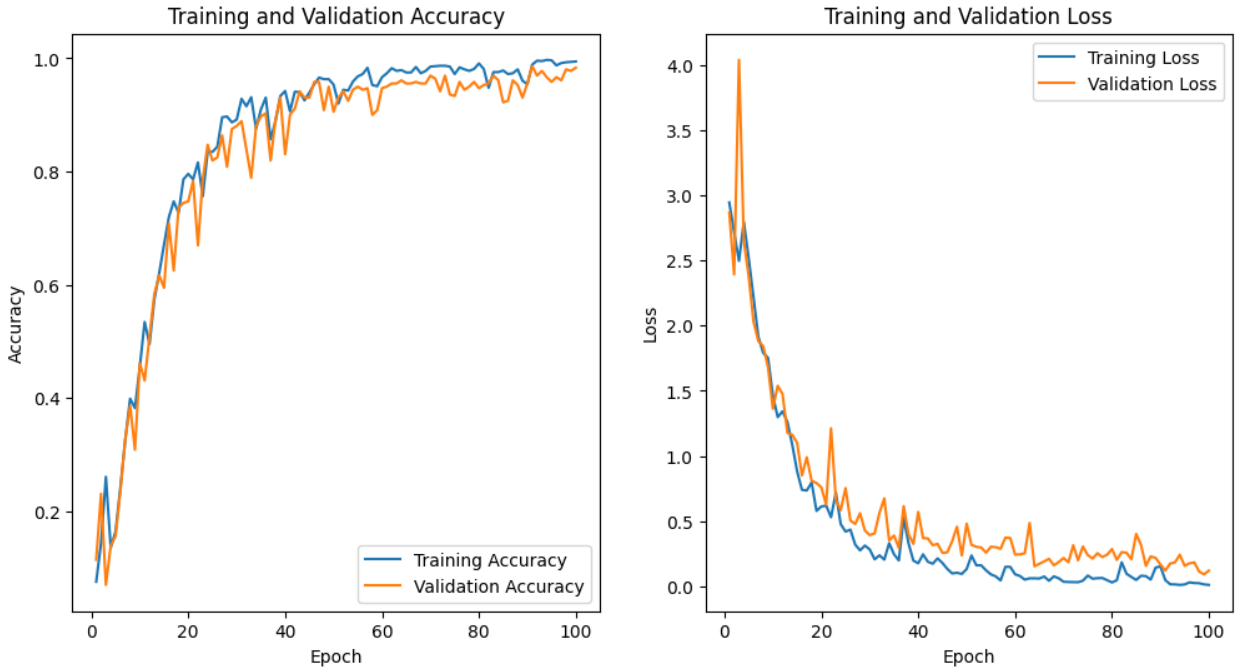

Рисунок 4.34 – Графіки точності і втрат моделі (при 100 епохах)

Аналіз графіку точності. Обидві лінії демонструють тенденцію до зростання, що свідчить про те, що здатність моделі правильно прогнозувати навчальні та валідаційні дані з часом покращується. При навчанні на 100 епохах точність також різко зростає спочатку, але плато є більш стабільним, у порівнянні із навчанням моделі на 50 епохах, особливо для точності перевірки, яка залишається близькою до точності навчання протягом усього часу. Це свідчить про хороше узагальнення без значного перенавчання.

Аналіз графіку втрат. Обидві лінії показують тенденцію до зниження, що значить те, що модель стає більш точною. Втрати при навчанні та валідації різко зменшуються на початку, а потім вирівнюються, причому втрати навчання продовжують зменшуватися трохи більше, ніж втрати валідації, зі збільшенням кількості епох. Втрати при навчанні стають дещо меншими, ніж втрати при перевірці, що вказує на те, що модель працює трохи краще на навчальних даних, ніж на невидимих перевірочних даних. При навчанні на 100 епохах як навчальні, так і валідаційні втрати зменшуються, а потім вирівнюються з меншими коливаннями, що свідчить про більш стабільний процес навчання. Втрати на валідацію також більш тісно пов'язані з втратами на навчання, що свідчить про краще узагальнення.

Розширення навчання до 100 епох, дало користь у цьому випадку, враховуючи, що для обох тренувань використовувалася однакова архітектура моделі та набір даних. Довше навчання дозволило моделі краще навчитися, про що свідчать більш стабільні та послідовні показники валідації. На рис. 4.35 зображено побудовану матрицю невідповідностей.

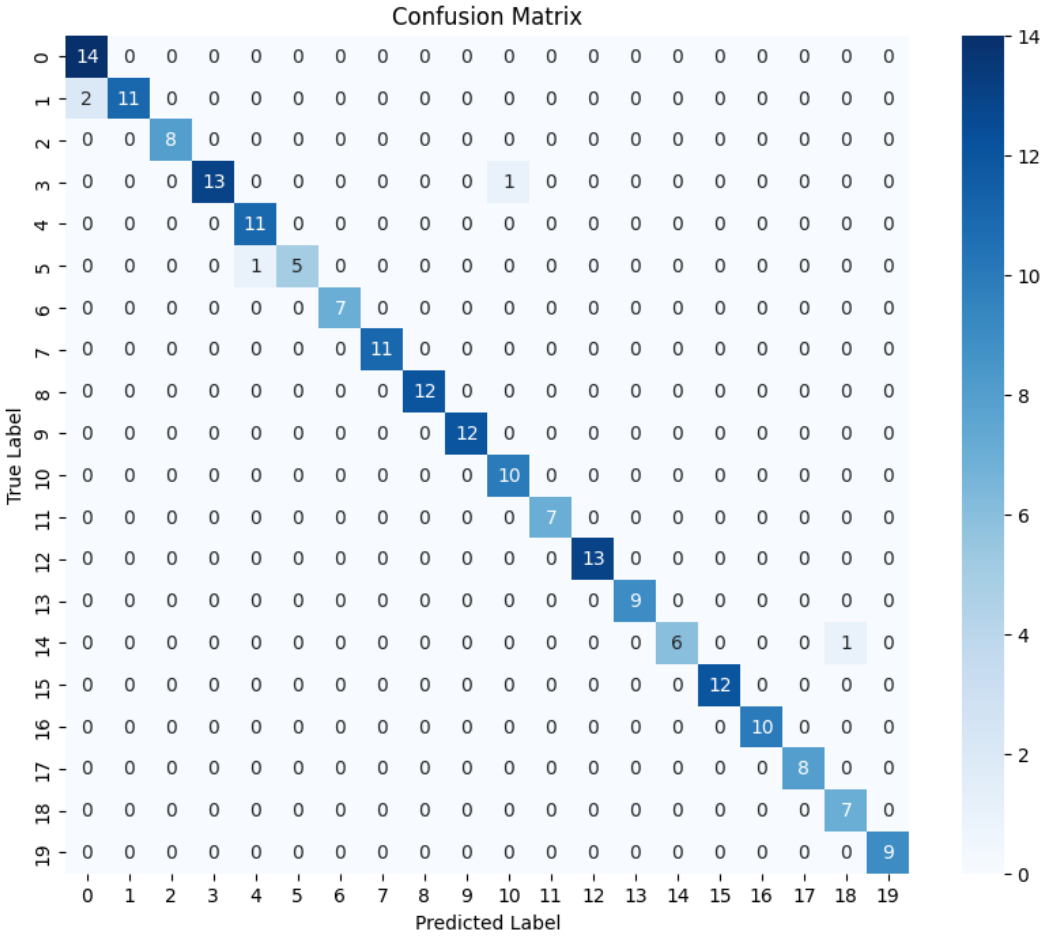

Рисунок 4.35 – Матриця невідповідностей (при 100 епохах)

Вертикальна вісь матриці представляє істинні мітки даних (англ. True label) в діапазоні від 0 до 19. Це фактичні класифікації, які модель намагається передбачити. Горизонтальна вісь представляє прогнозовані мітки (англ. Predicted Label), які модель видає на виході, також в діапазоні від 0 до 19. Кожен рядок матриці відповідає реальній мітці, а кожен стовпчик – прогнозованій мітці. Клітинки вздовж діагоналі матриці (з лівого верхнього кута до правого нижнього) представляють правильні прогнози, в яких передбачена мітка збігається з істинною міткою. Ці клітинки, як правило, підсвічуються або мають більшу кількість балів, що свідчить про правильну класифікацію, зроблену моделлю. Наприклад, клітинка для мітки 0 показує 14 правильних прогнозів, мітка 1 - 11, мітка 2 - 8 і так далі. Інтенсивність кольору кожної комірки відповідає кількості спостережень, при цьому темніші кольори означають більшу кількість спостережень. Це

підтверджується колірною смугою праворуч, яка показує шкалу значень підрахунку: від світлого для 0 до темного для найвищих значень (у цьому випадку близько 14). У даній матриці плутанини більшість діагональних елементів мають високі значення, що свідчить про те, що модель робить точні прогнози для багатьох класів. Однак є випадки неправильної класифікації (значення поза діагоналлю), наприклад, клас 1 (дактилема А) прогнозується як 0 (дактилема І), а клас 5 (дактилема С) прогнозується як 4 (дактилема Е). Проте, кількість помилкових класифікацій є відносно низькою.

Матриця невідповідностей дає змогу визначити, які мітки часто плутаються моделлю, що може бути корисно для діагностики та поліпшення роботи моделі. У цій матриці видно, що деякі мітки передбачаються дуже точно (з великими числами на діагоналі), в той час як декілька плутаються з сусідніми мітками.

Стосовно матриці помилок отриманої, в результаті навчання моделі на 50 епохах (рис. 4.30), можна помітити, що кількість випадків неправильної класифікації більша у порівнянні із результатами моделі при 100 епохах.

Таблиця 4.1 – Розраховані значення метрик для кожного класу (при 100 епохах)

Клас	**Precision**	**Recall**	**F1-score**	**Support**
І	0.88	1.00	0.93	14
А	1.00	0.85	0.92	13
В	1.00	1.00	1.00	8
Д	1.00	0.93	0.96	14
Е	0.92	1.00	0.96	11
С	1.00	0.83	0.91	6
У	1.00	1.00	1.00	7
Ц	1.00	1.00	1.00	11
Ь	1.00	1.00	1.00	12
будинок	1.00	1.00	1.00	12
дванадцять	0.91	1.00	0.95	10
добрий день	1.00	1.00	1.00	7

Закінчення таблиці 4.1

записатись на прийом	1.00	1.00	1.00	13
лікар	1.00	1.00	1.00	9
сьогодні	1.00	0.86	0.92	7
телефон	1.00	1.00	1.00	12
технологія	1.00	1.00	1.00	10
хотіти	1.00	1.00	1.00	8
я	0.88	1.00	0.93	7
як ваші справи	1.00	1.00	1.00	9
Accuracy			0.97	200
Macro avg	0.98	0.97	0.97	200
Weighted avg	0.98	0.97	0.97	200

Звіт про класифікацію (табл. 4.1) підкреслює високу загальну продуктивність моделі з високою точністю, запам'ятовуванням та показниками F1, хоча між різними класами існують певні відмінності. Макро- та середньозважені показники також високі, що свідчить про загальну високу продуктивність. Ці спостереження в сукупності свідчать про те, що модель добре працює на тестових даних. Загальна точність моделі становить 0.97, що вказує на те, що вона правильно передбачає клас приблизно в 97% випадках.

4.4 Розробка програмної частини

Для реалізації програмної частини системи було створено файл interface_modern.py. На початку програми відбувається імпорт бібліотек для роботи з графічним інтерфейсом та обробки зображень та відео. Використовуються такі бібліотеки як customtkinter для роботи з графічним інтерфейсом, PIL для обробки зображень, cv2 для роботи з відео та numpy для математичних обчислень. Крім того, імпортується бібліотека MediaPipe для виявлення ключових точок та жестів на відео, а також модуль calculation_functions для використання функцій

обчислення, і модуль tts_logic для роботи з синтезом мови, який буде описаний пізніше. Далі йде опис функцій системи.

Функція to_tts() виконує конвертацію текст-у-мовлення. Вона спочатку отримує вибрану користувачем стать та поточне речення.

Наступним кроком викликає функцію ttsfunc(), передаючи їй поточне речення та вибрану стать. Ця функція відповідає за генерацію аудіо-файлу на основі переданого їй тексту з урахуванням вибраної статі.

Функція remove_last_element() перевіряє, чи існують елементи у списку *sentence*. Якщо так, то вона видаляє останній елемент цього списку і викликає функцію «update_text()», щоб оновити відображення тексту.

Функція update_cam_label(image) призначена для оновлення відображення камери у вікні GUI. Вона приймає зображення *image*, перетворює його у формат RGBA, створює ImageTk.PhotoImage, який можна використовувати у *tkinter*, та оновлює мітку cam_label, щоб показати нове зображення.

Функція update_text(sentence) оновлює відображення тексту у віджеті тексту. Вона спочатку очищує вміст віджету, а потім вставляє очищений рядок тексту, який є об'єднаною версією всіх елементів списку sentence.

Функція update_text_info(sentence) оновлює відображення текстової інформації. Вона приймає список sentence та оновлює текстову інформацію в іншому віджеті для відображення.

Потім оголошуються змінні mp_holistic та mp_drawing, які представляють бібліотеку Mediapipe, яка використовується для виявлення та візуалізації ключових точок та жестів на зображеннях. Змінна actions містить набір можливих жестів, які програма може розпізнати.

У програмному вікні створюються віджети для відображення відео з камери та текстової інформації про розпізнані жести. Вікно має дві частини: ліву та праву. У лівій частині розташоване відео з камери, а у правій – віджети для відображення тексту та керування програмою.

Відео з камери відображається у віджеті cam_label. Цей віджет розміщений у

фреймі cam_frame, який розташований у лівій частині вікна.

Текстова інформація про розпізнані жести відображається у віджеті text_widget_info, а сам текст відображається у віджеті text_widget. Кнопка «Видалити» (remove_last_button) дозволяє видалити останній розпізнаний жест.

Ці віджети розташовані у фреймі group_frame, що знаходиться у правій частині вікна. У фреймі group_frame2 також розміщені радіокнопки для вибору гендеру та кнопка «Озвучити» (tts_button), яка запускає озвучення тексту за допомогою функції to_tts().

У функції get_frame() програма отримує кадр з потоку відео камери. Глобальні змінні, такі як sequence, sentence, predictions і threshold, використовуються для зберігання інформації про послідовність ключових точок рук, речення, передбачення жестів та порогу вірогідності.

Після отримання кадру програма викликає функцію для обробки кадру та визначення розташування ключових точок рук і жестів. Результати виявлення зберігаються у змінній results, а зображення з маркерами – у змінній image. Якщо руки виявлено на кадрі, їх координати витягуються для подальшого аналізу. Знайдені ключові точки додаються до sequence, яка містить останні 30 кадрів. Якщо довжина sequence дорівнює 30, вона передається в модель машинного навчання для передбачення жестів.

Модель повертає вектор ймовірностей для кожного класу жестів. Якщо максимальна ймовірність перевищує встановлений поріг threshold, програма визначає передбачений жест і додає його до речення sentence. Якщо sentence порожнє, то до нього додається перший жест. Після цього програма оновлює відображення текстової інформації з передбаченими жестами.

Крім того, якщо максимальна вірогідність не перевищує поріг, програма відображає повідомлення про невдачу в розпізнанні жесту. Якщо руки не були виявлені на кадрі, програма також відображає відповідне повідомлення.

Оновлене зображення кадру відображається у вікні програми, яке регулярно оновлюється кожні 10 мілісекунд, щоб відображати поточний стан камери. Після

цього програма знаходиться в безкінечному циклі, який закінчується лише при закритті вікна інтерфейсу користувача.

Далі програма ініціалізує алгоритм Mediapipe для виявлення жестів та ключових точок на зображеннях з відеопотоку камери. Параметри min_detection_confidence і min_tracking_confidence встановлюють мінімальні пороги впевненості для виявлення об'єктів та відстеження їх рухів відповідно.

Після ініціалізації алгоритму, програма ініціалізує змінні sequence, sentence та predictions для зберігання даних про послідовність ключових точок рук, речення та передбачення жестів відповідно. Змінна threshold встановлює поріг вірогідності для визначення, коли програма повинна реагувати на передбачені жести.

Далі програма завантажує попередньо навчену модель машинного навчання з файлу mymodel.h5. Після цього викликається функція get_frame(), яка починає отримувати та аналізувати кадри з камери.

Останнім кроком у цьому блоку коду є виклик методу mainloop(), який запускає безкінечний цикл обробки подій для вікна програми. Це дозволяє користувачеві взаємодіяти з програмою та переглядати результати в реальному часі. Після завершення роботи програми всі створені аудіофайли видаляються з тимчасової папки.

Після перетворення жестів на текст, система використовує технології синтезу мовлення для конвертації цього тексту у звуковий формат аудіо. Дана операція реалізована у файлі tts_logic.py. Цей фрагмент коду відповідає за синтез мовлення тексту за допомогою українського модулю TTS.

Ukrainian-TTS представляє собою чудовий інструмент для озвучування тексту українською мовою [48]. Його основою є відкрита платформа ESPnet, що дозволяє ефективно перетворювати текст на мовлення з високою точністю та природністю. Система підтримує кілька голосів, що забезпечує різноманітність тембрів та стилів мовлення, а також дозволяє користувачам вибирати найбільш підходящий голос для конкретних завдань. Однією з ключових особливостей

Ukrainian-TTS є здатність автоматично встановлювати наголоси в тексті, використовуючи комбінацію словникових даних та моделей машинного навчання.

Таким чином, спочатку імпортуються необхідні бібліотеки, такі як *ukrainian_tts* для роботи з синтезом мови, *playsound* для відтворення аудіофайлів, а також модуль *os* для операцій з файловою системою та *time* для роботи з часом.

Перед використанням TTS створюється каталог audios, якщо його не існує, де будуть зберігатися аудіофайли. Потім ініціалізується об'єкт TTS з використанням CPU (центрального процесора). За потреби можна використовувати інші пристрої, такі як GPU або MPS. Функція ttsfunc() використовується для синтезу мовлення з заданого тексту та гендеру. Після синтезу аудіофайл зберігається у каталозі audios з унікальним ім'ям, яке включає час створення файлу (рис. 4.36). Залежно від обраного гендеру використовується відповідний голос для синтезу мовлення. Після цього аудіофайл відтворюється за допомогою playsound().

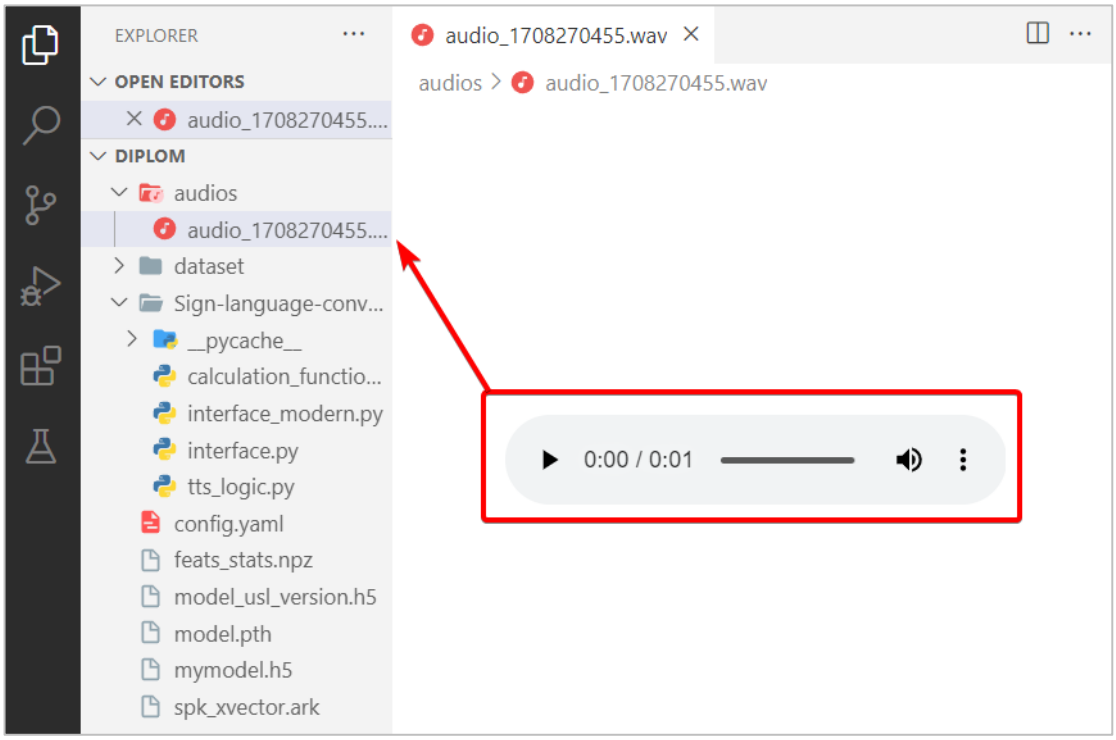

Рисунок 4.36 – Згенерований аудіофайл

Функція delete_audio_files() призначена для видалення всіх аудіофайлів у заданому каталозі. Вона перебирає всі файли у каталозі та видаляє тільки файли з розширенням .wav.

Цей код потрібен для видалення всіх створених аудіофайлів після виконання роботи програми, щоб не зайняти всю пам’ять користувача в процесі тривалого використання програми.

4.5 Тестування розробленої системи та аналіз результатів

Під час запуску програми виводиться вікно інтерфейсу, що складається із області для трансляції вебкамери (рис. 4.37, позначка 1), області для виводу прогнозованого тексту (рис. 4.37, позначка 2) та області для вибору голосу для синтезу мовлення (рис. 4.37, позначка 3).

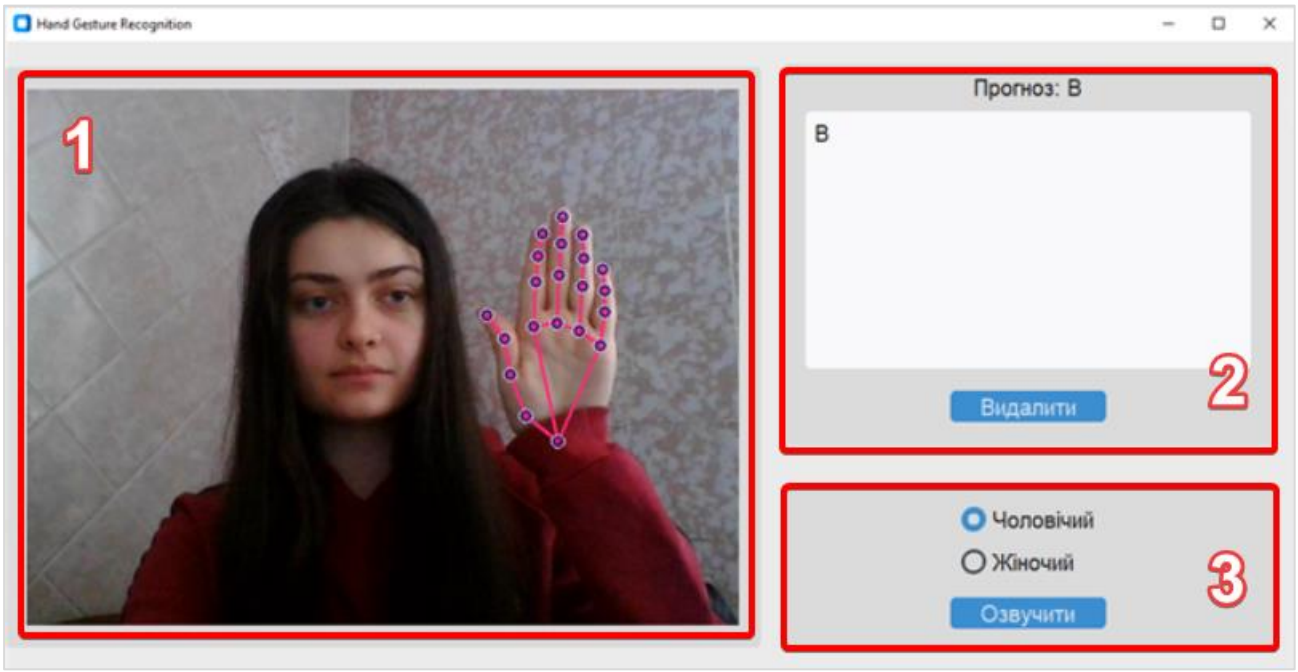

Рисунок 4.37 – Компоненти інтерфейсу розробленої системи

Якщо в області трансляції вебкамери не виявлено ні однієї руки, то виводиться повідомлення «Руку не виявлено» (рис. 4.38).

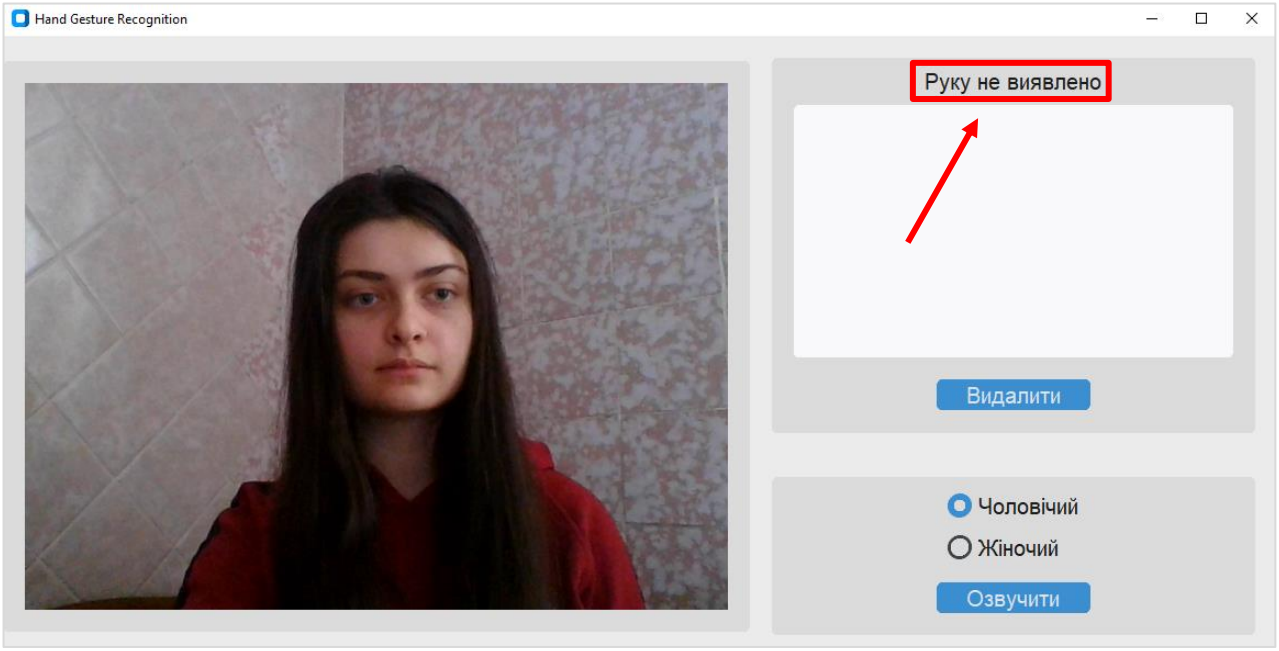

Рисунок 4.38 – Вивід повідомлення «Руку не виявлено»

Якщо довжина послідовності ключових точок рівна 10, то виводиться повідомлення «Прогнозування…» (рис. 4.39), яке говорить користувачеві про те, що зараз відбувається процес прогнозування жесту на основі накопичених даних.

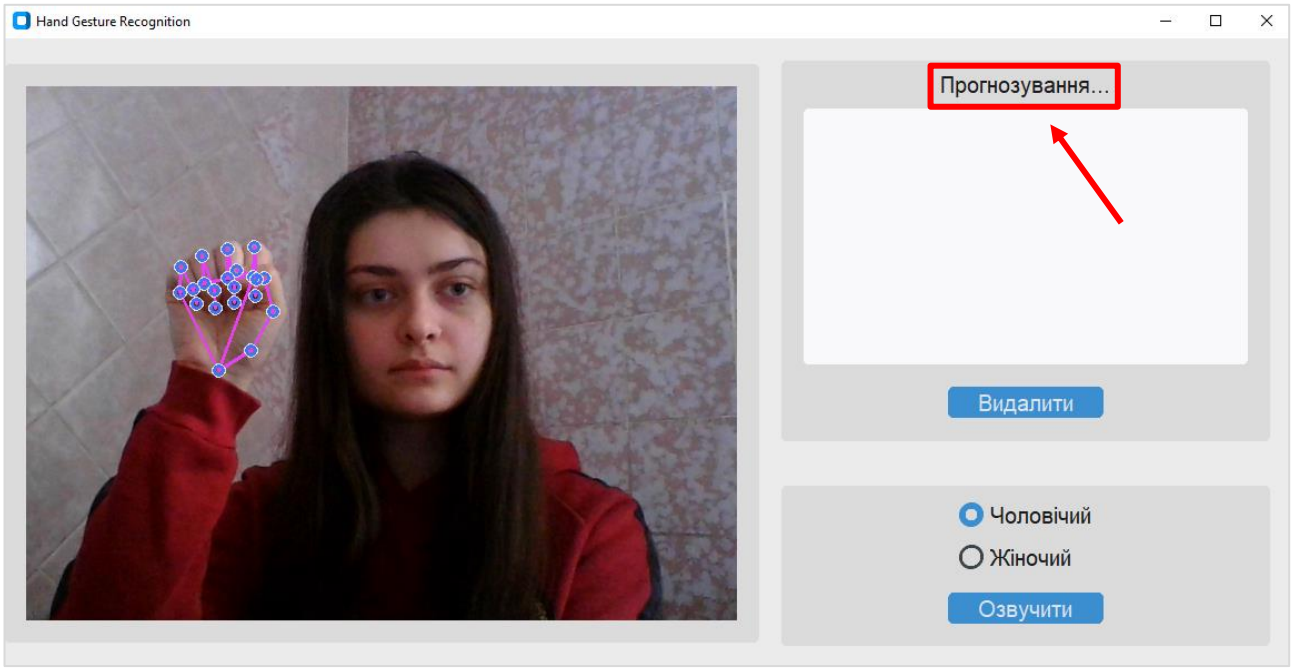

Рисунок 4.39 – Вивід повідомлення «Прогнозування…»

Якщо довжина послідовності ключових точок рівна 30, викликається модель для прогнозування. Якщо точність прогнозування жесту більша за поріг threshold, то виводиться повідомлення «Прогноз» з відповідною назвою класу (рис. 4.40). Також назва класу виводиться в текстове поле.

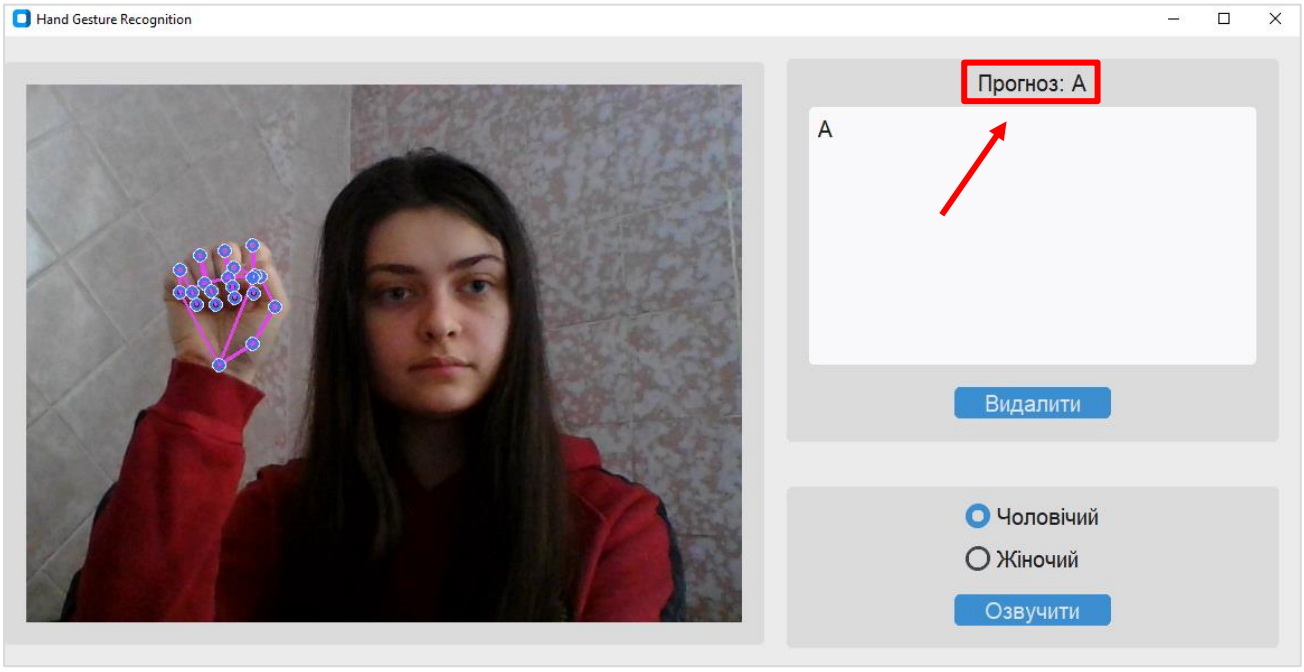

Рисунок 4.40 – Вивід прогнозованої літери «А»

У випадку, якщо точність прогнозування жесту менша за вказаний поріг, то виводиться повідомлення «Жест не розпізнано» (рис. 4.41).

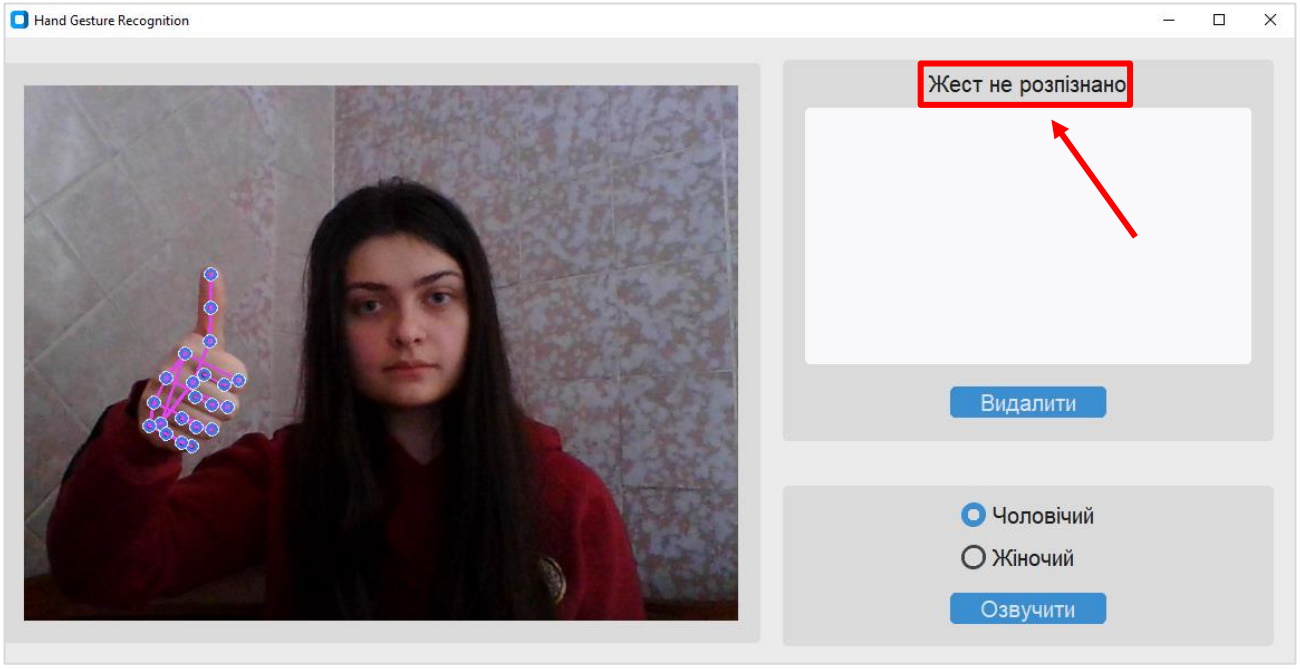

Рисунок 4.41 – Вивід повідомлення «Жест не розпізнано»

Розроблена система здатна прогнозувати літери (рис. 4.40), слова (рис. 4.42) та вирази (рис. 4.43, рис. 4.44).

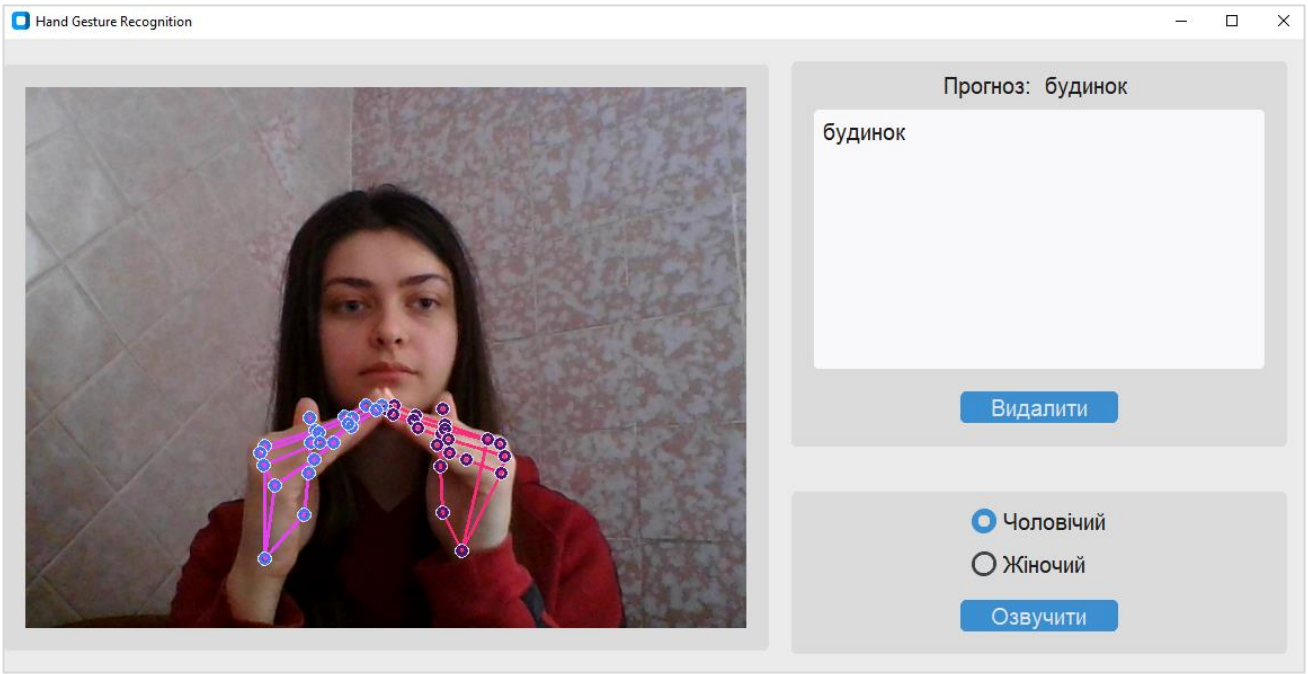

Рисунок 4.42 – Вивід прогнозованого слова «будинок»

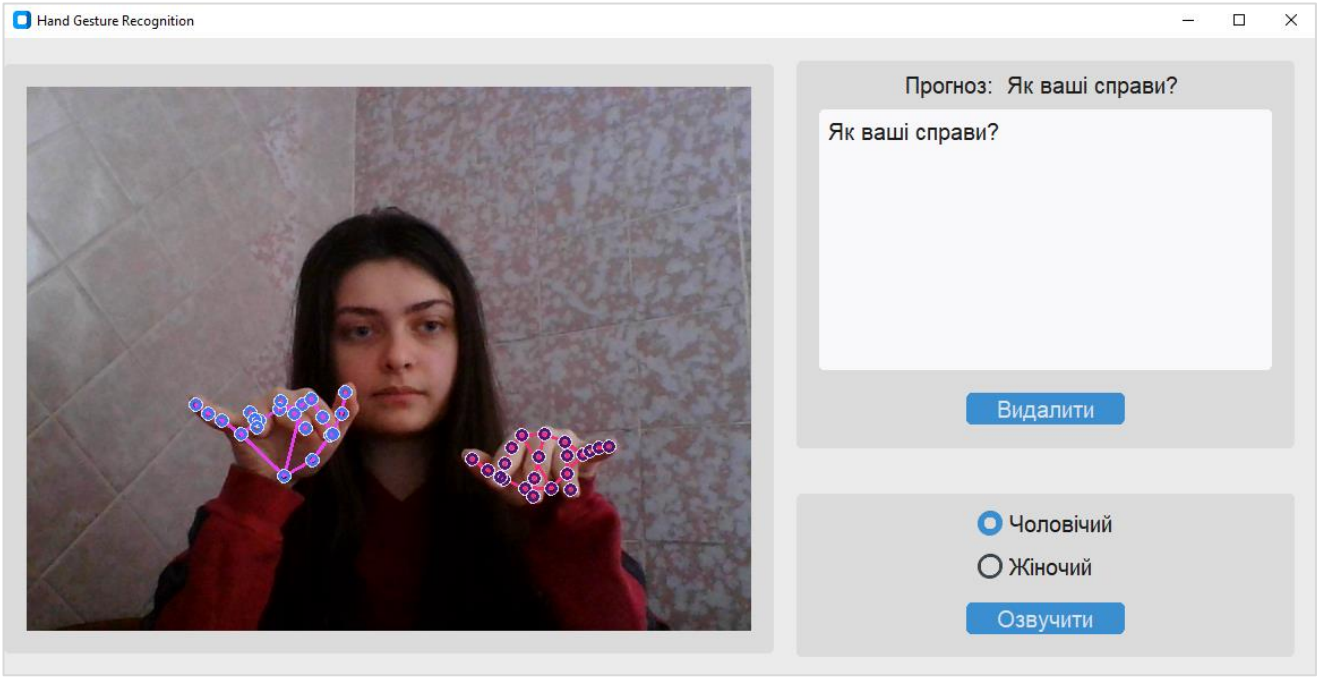

Рисунок 4.43 – Вивід прогнозованого виразу «Як ваші справи?»

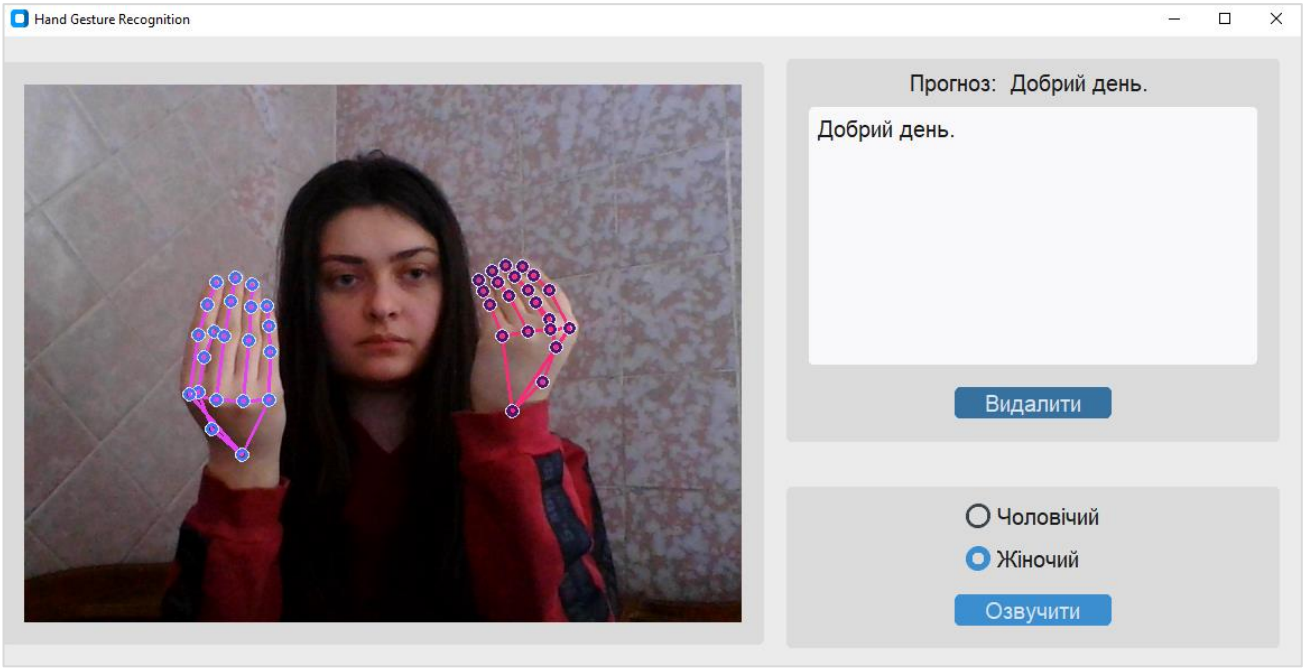

Рисунок 4.44 – Вивід прогнозованого виразу «Добрий день»

Також користувач може видалити останнє згенероване слово або літеру, натиснувши відповідну кнопку «Видалити». Після перетворення жестів на згенерований текст, юзер може вибрати відповідний голос (чоловічий або жіночий)

для синтезування звуку та натиснути на кнопку «Озвучити». Система використовує технології синтезу мовлення для конвертації цього тексту у звуковий формат аудіо. Описану логіку принципу роботи інтелектуальної системи проілюстровано на блок-схемі (рис. 4.45).

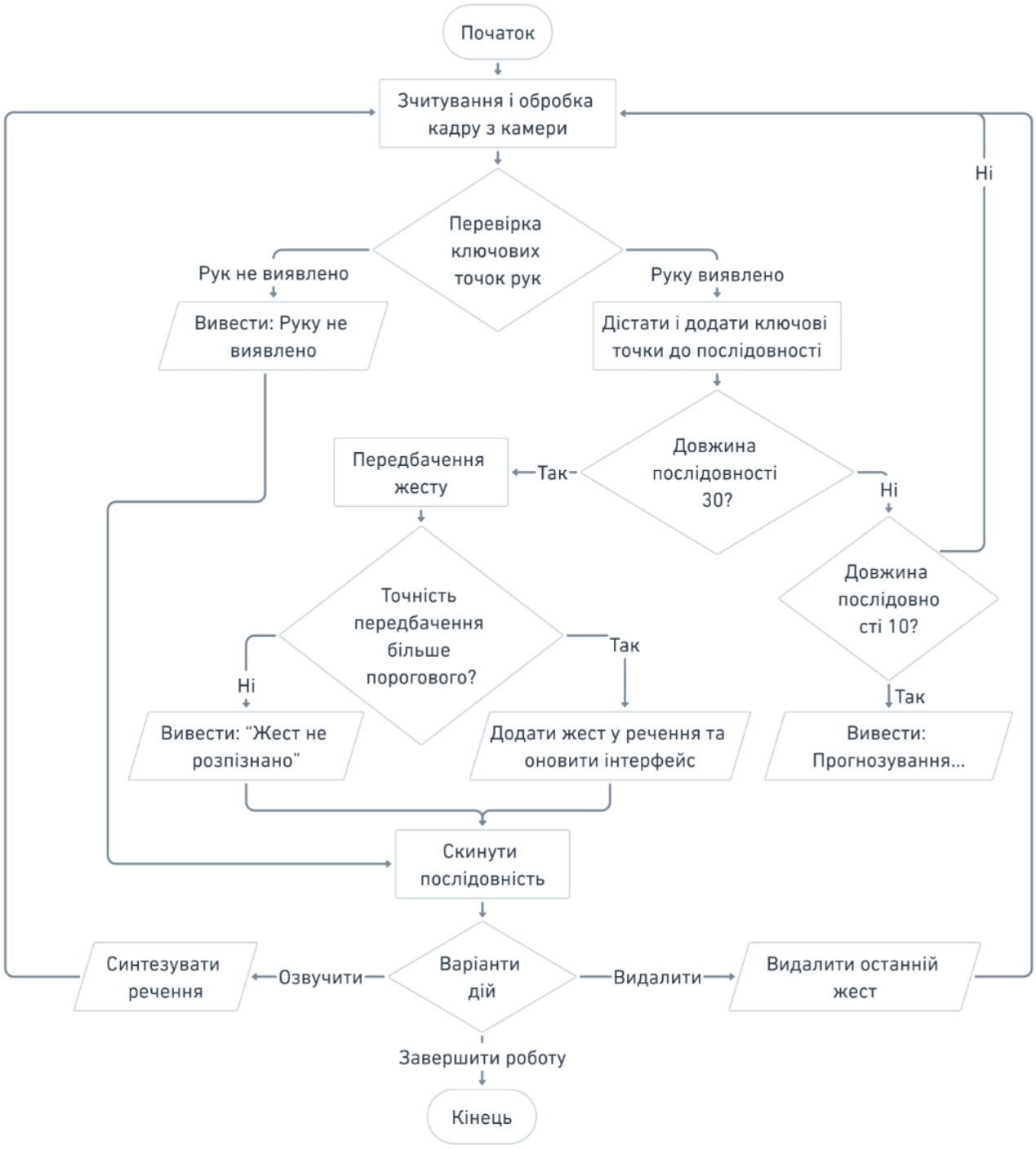

Рисунок 4.45 – Блок-схема принципу роботи розробленої системи

За допомогою прямої трансляції показано розпізнавання статичних та динамічних жестів української жестової мови та перетворення їх на текст. Тестування в реальному часі було проведено для кожного класу жестів. Отримані результати тестування системи занесено у табл. 4.2.

Таблиця 4.2 – Отримані показники точності для кожного класу під час тестування системи на реальних даних

Клас		**Точність прогнозу (accuracy)**
Дактилеми	І	0.8416
	А	0.9808
	В	0.9999
	Д	0.7687
	Е	0.9993
	С	0.6057
	У	0.9999
	Ц	1.0000
	Ь	0.9997
Слово/фраза	будинок	0.9624
	дванадцять	0.9942
	добрий день	0.9929
	записатись на прийом	0.9603
	лікар	1.0000
	сьогодні	1.0000
	телефон	0.9904
	технологія	1.000
	хотіти	0.9545
	я	0.9999
	як ваші справи?	1.0000

Отримані результати тестування системи розпізнавання жестів показують високу точність в більшості випадків. Однак, для певних класів (наприклад, «І»,

«Д» та «С»), показник точності нижчий (0.8416, 0.768, 0.6057 відповідно). Це може вказувати на те, що деякі жести можуть бути більш складними для системи в розпізнаванні через їхню схожість з іншими жестами, що впливає на надійність прогнозування.

У цілому, система демонструє гарну ефективність, забезпечуючи високу точність в більшості випадків. Отримані результати можуть служити важливою основою для подальшого вдосконалення та розвитку системи.

Висновки до розділу 4

В даному розділі було описано програмну реалізацію системи перетворення української жестової мови на текст та аудіо. Описано процес створення набору та підготовки даних для моделі розпізнавання жестів. Створено, навчено та протестовано розроблену нейронну модель. Також описано розроблену архітектуру нейронної мережі, яка складається із трьох LSTM шарів та трьох Dense шарів з різною кількістю нейронів та функціями активації. У процесі навчання моделі була використана функція втрат categorical_crossentropy та оптимізатор Adam. Побудовано графіки точності і втрат та матриці невідповідностей задля проведення порівняльного оцінювання продуктивності моделі при різній кількості епох при навчанні. Для кожного класу статичних і динамічних жестів отримано значення точності (accuracy), влучності (precision), повноти (recall) та F1-міри. Найкраща точність моделі досягнута при навчанні на 100 епохах та становить 97%, що говорить про досить хорошу прогнозовану здатність. У даному розділі було описано процес розробки програмної частини системи. Результати проведеного тестування системи розпізнавання жестів свідчать про високу точність у більшості сценаріїв.

ВИСНОВКИ

У результаті виконання даної кваліфікаційної роботи магістра було успішно розроблено інтелектуальну систему, спрямовану на перетворення жестів української жестової мови на текст та аудіо. Для моделі розпізнавання жестів була обрана архітектура довготривалої короткочасної пам’яті (LSTM), яка виявилася досить ефективною у вирішенні поставленої задачі.

Під час виконання роботи було створено власний набір даних, що включає в себе 20 класів жестів, охоплюючи літери українського дактильного алфавіту, цілі слова та вирази. Кожен клас містив 100 записів, розподілених на 30 кадрів, що загалом складало 3000 кадрів для кожного класу для навчання моделі.

Нейронна модель, розроблена для реалізації даної системи, включає три LSTM шари та три Dense шари з різною кількістю нейронів та функціями активації (ReLU і softmax). У процесі навчання використовувалися функція втрат categorical_crossentropy та оптимізатор Adam. Найкраща точність моделі досягнута при навчанні на 100 епохах та становить 97%. Результати тестування підтверджують високу ефективність системи в різноманітних сценаріях.

У роботі використовувалися такі технології, як мова програмування Python та середовища розробки Google Colab, Jupyter та Visual Studio Code, а також бібліотеки OpenCV, Keras, NumPy, TensorFlow та фреймворк MediaPipe. Розроблена система може бути успішно впроваджена в сфери освіти, медицини, комунікацій, обслуговування клієнтів та у повсякденну діяльність, сприяючи активнішій участі цільової аудиторії у різних сферах їхнього життя. При необхідності, можна розширити функціональність програми, аби вона розпізнавала більшу кількість жестів, що сприятиме більшій універсальності та адаптованості системи до різноманітних використань.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Wu, Y.; Huang, T. Vision-Based Gesture Recognition: A Review. In *Gesture-Based Communication in Human-Computer Interaction*; Springer: Berlin/Heidelberg, Germany, 1999. 103–115 с.

2. Starner, T.; Mann, S.; Rhodes, B.; Levine, J.; Healey, J.; Kirsch, D.; Picard, R.W.; Pentland, A. Augmented Reality through Wearable Computing. *Presence Teleoperators Virtual Environ.* 1997. 386–398 с.

3. World Health Organization. Deafness and hearing loss: вебсайт. URL: https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss (дата звернення: 14.12.2023).

4. Statista: Projected number of people with disabling hearing loss worldwide in 2019, 2030, 2040, and 2050: вебсайт. URL: https://www.statista.com/statistics/888569/number-of-people-with-hearing-loss-worldwide-projections/#statisticContainer (дата звернення: 14.12.2023).

5. Tvoroshenko Iryna, Kharchenko Anastasiia: Some aspects of modern development for sign language recognition systems. *Trends in the scientific development*. 2021, 352 с.

6. Opalka, A. and Miller-Jacobson, B. Motionsavvy: вебсайт. URL: https://www.motionsavvy.com/ (дата звернення: 19.12.2023).

7. Robotka, Z., Rovnyai, J., Gerlis, S., Retek, D., Pintr, M., and Croasmun, D. Signall media kit: вебсайт. URL: https://www.signall.us (дата звернення: 19.12.2023).

8. Цифрові рішення для людей з порушенням слуху: вебсайт. URL: https://se.org.ua/cifrovi-rishennya/ (дата звернення: 19.12.2023).

9. Halder, Arpita, and Akshit Tayade. Real-time vernacular sign language recognition using mediapipe and machine learning: вебсайт. URL: https://www.ijrpr.com/uploads/V2ISSUE5/IJRPR462.pdf (дата звернення: 20.12.2023).

10. C. Keskin, F. Kirac, Y. E. Kara, and L. Akarun, “Real time hand pose

estimation using depth sensors," 2011: вебсайт. URL: https://yekara.com/pub/keskin2013cdc4cv.pdf (дата звернення: 20.12.2023).

11. Lee, C.; Ng, K.K.; Chen, C.H.; Lau, H.; Chung, S.; Tsoi, T. American Sign Language Recognition and Training Method with Recurrent Neural Network: вебсайт. URL: https://doi.org/10.1016/j.eswa.2020.114403 (дата звернення: 19.12.2023).

12. Papatsimouli, M.; Sarigiannidis, P.; Fragulis, G.F. A Survey of Advancements in Real-Time Sign Language Translators. Technologies: вебсайт. URL: https://doi.org/10.3390/technologies11040083 (дата звернення: 19.12.2023).

13. A. K. Sahoo, "Indian Sign Language Recognition Using Machine Learning Techniques," Macromolecular Symposia, 2021. вебсайт. URL: https://doi.org/10.1002/masy.202000241 (дата звернення: 12.01.2024).

14. Wahid, M.F.; Tafreshi, R.; Al-Sowaidi, M.; Langari, R. Subject-independent hand gesture recognition using normalization and machine learning algorithms. *J. Comput. Sci.* 2018, *27*, 69–76.

15. Bhushan, S.; Alshehri, M.; Keshta, I.; Chakraverti, A.K.; Rajpurohit, J.; Abugabah, A. An Experimental Analysis of Various Machine Learning Algorithms for Hand Gesture Recognition. *Electronics*, 2022: вебсайт. URL: https://doi.org/10.3390/electronics11060968 (дата звернення: 19.12.2023).

16. Gajowniczek, K.; Grzegorczyk, I.; Ząbkowski, T.; Bajaj, C. Weighted Random Forests to Improve Arrhythmia Classification. *Electronics* 2020, 9, 99.

17. Bheda, V., and Radpour, D. Using deep convolutional networks for gesture recognition in American sign language. 2017. вебсайт. URL: https://arxiv.org/ftp/arxiv/papers/1710/1710.06836.pdf (дата звернення: 21.12.2023).

18. Alsharif, Bader, Ali Salem Altaher, Ahmed Altaher, Mohammad Ilyas, and Easa Alalwany. 2023. "Deep Learning Technology to Recognize American Sign Language Alphabet": вебсайт. URL: https://doi.org/10.3390/s2318797 (дата звернення: 21.12.2023).

19. A. Das, S. Gawde, K. Suratwala, and D. Kalbande, "Sign Language Recognition Using Deep Learning on Custom Processed Static Gesture Images". 2018:

вебсайт. URL: https://ieeexplore.ieee.org/document/8537248 (дата звернення: 19.12.2023).

20. Al-Obodi, A.H.; Al-Hanine, A.M.; Al-Harbi, K.N.; Al-Dawas, M.S.; Al-Shargabi, A.A. A Saudi Sign Language recognition system based on convolutional neural networks. *Build. Serv. Eng. Res. Technol.* 2020, *13*, 3328 с.

21. Hand gesture recognition using Machine Learning: вебсайт. URL: https://medium.com/@gdsc_32043/hand-gesture-recognition-using-machine-learnign-a92b5a456374 (дата звернення: 19.12.2023).

22. Fregoso, J.; Gonzalez, C.I.; Martinez, G.E. Optimization of Convolutional Neural Networks Architectures Using PSO for Sign Language Recognition. *Axioms* : вебсайт. URL: https://doi.org/10.3390/axioms10030139 (дата звернення: 21.12.2023).

23. Kaviani, S.; Sohn, I. Application of complex systems topologies in artificial neural networks optimization: An overview. *Expert Syst. Appl.* 2021, *180*, 115073: вебсайт. URL: https://doi.org/10.1016/j.eswa.2021.115073 (дата звернення: 21.12.2023).

24. Ye, D.; Wen, J.; Zheng, S.; Zhong, Q.; Pei, W.; Jia, H.; Zhou, C.; Gong, Y. Prediction of Key Parameters of Wheelset Based on LSTM Neural Network. *Appl. Sci.* 2023, *13*, 11935: вебсайт. URL: https://doi.org/10.3390/app132111935 (дата звернення: 21.12.2023).

25. Toro-Ossaba, A.; Jaramillo-Tigreros, J.; Tejada, J.C.; Peña, A.; López-González, A.; Castanho, R.A. LSTM Recurrent Neural Network for Hand Gesture Recognition Using EMG Signals. Appl. Sci. 2022, 12, 9700. вебсайт. URL: https://doi.org/10.3390/app12199700 (дата звернення: 12.01.2024).

26. Sundar, B., & Bagyammal, T. (2022). American Sign Language Recognition for Alphabets Using MediaPipe and LSTM. вебсайт. URL: https://doi.org/10.1016/j.procs.2022.12.066 (дата звернення: 12.01.2024).

27. Sepp Hochreiter, Jürgen Schmidhuber: Long short-term memory [Electronic resource]. – 1997. – Access mode: https://www.bioinf.jku.at/publications/older/2604.pdf (дата звернення 16.11.2023).

28. Bicheng, H.; Langxing, T.; Yu, Z. Short-term power generation load forecasting based on LSTM neural network. J. Phys. Conf. Ser. 2022: вебсайт. URL: https://www.researchgate.net/publication/359908802_Shortterm_power_generation_load_forecasting_based_on_LSTM_neural_network (дата звернення: 21.12.2023).

29. Song, K.; Jang, J.H.; Shin, S.J.; Moon, I.C. Bivariate Beta-LSTM. In Proceedings of the AAAI Conference on Artificial Intelligence, 2020: вебсайт. URL: https://doi.org/10.1609/aaai.v34i04.6039 (дата звернення: 21.12.2023).

30. Jupyter Notebook: вебсайт. URL: https://jupyter.org/ (дата звернення 16.11.2023).

31. Google Colab: вебсайт. URL: https://research.google.com/colaboratory/faq.html (дата звернення 16.11.2023).

32. Microsoft. Visual Studio Code - Code Editing. Redefined. Visual Studio Code: вебсайт. URL: https://code.visualstudio.com (дата звернення: 21.12.2023).

33. Realtime Computer Vision with OpenCV: Mobile computer-vision technology will soon become as ubiquitous as touch interfaces. вебсайт. URL: https://dl.acm.org/doi/10.1145/2181796.2206309 (дата звернення: 21.12.2023).

34. A multi-framework deep learning API Keras 3. About Keras 3: вебсайт. URL: https://keras.io/about/ (дата звернення: 21.12.2023).

35. Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen and others. TensorFlow: A system for large-scale machine learning: вебсайт. URL: https://arxiv.org/abs/1605.08695 (дата звернення: 21.12.2023).

36. NumPy documentation: вебсайт. URL: https://numpy.org/about/ (дата звернення: 22.12.2023).

37. Remiro, M.Á.; Gil-Martín, M.; San-Segundo, R. Improving Hand Pose Recognition Using Localization and Zoom Normalizations over MediaPipe Landmarks. *Eng. Proc.* 2023: вебсайт. URL: https://doi.org/10.3390/ecsa-10-16215 (дата звернення: 22.12.2023).

38. Mediapipe official documentation: вебсайт. URL: https://developers.google.com/mediapipe (дата звернення: 22.12.2023).

39. Samaan, G.H.; Wadie, A.R.; Attia, A.K.; Asaad, A.M.; Kamel, A.E.; Slim, S.O.; Abdallah, M.S.; Cho, Y.-I. MediaPipe's Landmarks with RNN for Dynamic Sign Language Recognition: вебсайт. URL: https://doi.org/10.3390/electronics11193228 (дата звернення: 22.12.2023).

40. Lugaresi, C.; Tang, J.; Nash, H.; McClanahan, C.; Uboweja, E.; Hays, M.; Zhang, F.; Chang, C.L.; Yong, M.G.; Lee, J.; et al. Mediapipe: A framework for building perception pipelines, 2019.

41. Zhang, F.; Bazarevsky, V.; Vakunov, A.; Tkachenka, A.; Sung, G.; Chang, C.L.; Grundmann, M. Mediapipe hands: On-device real-time hand tracking,2022.

42. Hinkelmann, Knut. University of Applied Sciences Northwestern Switzerland. "Neural Networks": вебсайт. URL: https://web.archive.org/web/20181006235506/http://didattica.cs.unicam.it/lib/exe/fetch.php?media=didattica:magistrale:kebi:ay_1718:ke-11_neural_networks.pdf. (дата звернення: 22.12.2023).

43. Keras 3 API documentation. Layer activation functions: вебсайт. URL: https://keras.io/api/layers/activations/ (дата звернення: 23.12.2023).

44. Kim, K.-S.; Choi, Y.-S. HyAdamC: A New Adam-Based Hybrid Optimization Algorithm for Convolution Neural Networks. 2021: вебсайт. URL: https://doi.org/10.3390/s21124054 (дата звернення: 22.12.2023).

45. Chen, C.-H.; Lai, J.-P.; Chang, Y.-M.; Lai, C.-J.; Pai, P.-F. A Study of Optimization in Deep Neural Networks for Regression: вебсайт. URL: https://doi.org/10.3390/electronics12143071 (дата звернення: 24.12.2023).

46. Diederik P. Kingma, Jimmy Ba. Cornell University. Adam: A Method for Stochastic Optimization: вебсайт. URL: https://arxiv.org/abs/1412.6980 (дата звернення: 25.12.2023).

47. Keras 3 API documentation. Losses: вебсайт. URL: https://keras.io/api/losses/ (дата звернення: 23.12.2023).

48. Paniv Y. Robinhad/ukrainian-tts: Ukrainian TTS (text-to-speech) using ESPNET. GitHub: вебсайт. URL: https://github.com/robinhad/ukrainian-tts (дата звернення: 05.02.2024).

ДОДАТОК А

Лістинг коду системи

Файл make_dataset.py

```
!pip install tensorflow tensorflow-gpu opencv-python mediapipe sklearn matplotlib
!pip install mediapipe
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp

#Keypoints using MP Holistic
mp_holistic = mp.solutions.holistic # Holistic model
mp_drawing = mp.solutions.drawing_utils # Drawing utilities

def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image.flags.writeable = False
    results = model.process(image)
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    return image, results

def draw_landmarks(image, results):
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS) # Draw left hand connections
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS) # Draw right hand connections

def draw_styled_landmarks(image, results):
    # Draw left hand connections
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
```

```
                              mp_drawing.DrawingSpec(color=(121,22,76),
thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(121,44,250),
thickness=2, circle_radius=2)
                              )
    # Draw right hand connections
    mp_drawing.draw_landmarks(image,          results.right_hand_landmarks,
mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(245,117,66),
thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(245,66,230),
thickness=2, circle_radius=2)
                              )

#Extract Keypoint Values
def extract_keypoints(results):
    lh    =    np.array([[res.x,    res.y,    res.z]    for    res    in
results.left_hand_landmarks.landmark]).flatten()                          if
results.left_hand_landmarks else np.zeros(21*3)
    rh    =    np.array([[res.x,    res.y,    res.z]    for    res    in
results.right_hand_landmarks.landmark]).flatten()                         if
results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([lh, rh])

#Setup Folders for Collection
# Path for exported data, numpy arrays
DATA_PATH = os.path.join('D:\\USL_Dataset')
actions = np.array(['І', 'А', 'В', 'Д',  'Е', 'С', 'У', 'Ц', 'Ь', 'будинок',
' дванадцять ', 'Добрий день', 'записатись на прийом', 'лікар', 'сьогодні',
'телефон', 'технологія', 'хочу', 'я', 'Як ваші справи'])
# Thirty videos worth of data
no_sequences = 100
# Videos are going to be 30 frames in length
sequence_length = 30
# Folder start
#start_folder = 0
```

```
for action in actions:
    #dirmax      =      np.max(np.array(os.listdir(os.path.join(DATA_PATH,
action))).astype(int))
    for sequence in range(no_sequences):
        try:
            os.makedirs(os.path.join(DATA_PATH, action, str(sequence)))
        except:
            pass

#Collect Keypoint Values for Training and Testing
cap = cv2.VideoCapture(0)
# Set mediapipe model
with                    mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as holistic:

    # NEW LOOP
    # Loop through actions
    for action in actions:
        # Loop through sequences aka videos
        for sequence in range(no_sequences):
            # Loop through video length aka sequence length
            for frame_num in range(sequence_length):
                # Read feed
                ret, frame = cap.read()

                # Make detections
                image, results = mediapipe_detection(frame, holistic)

                # Draw landmarks
                draw_styled_landmarks(image, results)

                # NEW Apply wait logic
                if frame_num == 0:
                    cv2.putText(image, 'STARTING COLLECTION', (120,200),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4,
cv2.LINE_AA)
```

```
                    cv2.putText(image, 'Collecting frames for {} Video
Number {}'.format(action, sequence), (15,12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,255, 0), 1,
cv2.LINE_AA)
                    # Show to screen
                    cv2.imshow('OpenCV Feed', image)
                    cv2.waitKey(2000)
                else:
                    cv2.putText(image, 'Collecting frames for {} Video
Number {}'.format(action, sequence), (15,12),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,255, 0), 1,
cv2.LINE_AA)
                    # Show to screen
                    cv2.imshow('OpenCV Feed', image)

                # NEW Export keypoints
                keypoints = extract_keypoints(results)
                npy_path = os.path.join(DATA_PATH, action, str(sequence),
str(frame_num))
                np.save(npy_path, keypoints)

                # Break gracefully
                if cv2.waitKey(10) & 0xFF == ord('q'):
                    break

    cap.release()
    cv2.destroyAllWindows()
```

Файл tts_logic.py

```
from ukrainian_tts.tts import TTS, Voices, Stress
from playsound import playsound
import os
import time

if not os.path.exists("audios"):
    os.makedirs("audios")
```

```
tts = TTS(device="cpu") # can try gpu, mps

def ttsfunc(sentence, gender):
    audio_file = f"audios/audio_{int(time.time())}.wav"
    with open(audio_file, mode="wb") as file:
        if gender == "Чоловічий":
            _,   output_text   =   tts.tts(sentence,   Voices.Dmytro.value,
Stress.Dictionary.value, file)
        else:
            _,   output_text   =   tts.tts(sentence,   Voices.Tetiana.value,
Stress.Dictionary.value, file)
    playsound(audio_file)

def delete_audio_files(directory):
    for file in os.listdir(directory):
        if file.endswith(".wav"):
            os.remove(os.path.join(directory, file))
```

Файл interface_modern.py

```
import customtkinter as ctk
from PIL import Image, ImageTk
import cv2
import numpy as np
import mediapipe as mp
import calculation_functions as calc
from tts_logic import ttsfunc, delete_audio_files

def to_tts():
    selected_gender = gender_var.get()
    current_sentence = ' '.join(sentence)
    ttsfunc(current_sentence, selected_gender)

def remove_last_element():
    if len(sentence) > 0:
        sentence.pop()
        update_text(sentence)
```

```
def update_cam_label(image):
    cv2image = cv2.cvtColor(image, cv2.COLOR_BGR2RGBA)
    img = Image.fromarray(cv2image)
    imgtk = ImageTk.PhotoImage(image=img)
    cam_label.imgtk = imgtk
    cam_label.configure(image=imgtk)

def update_text(sentence):
    cleaned_sentence = ''.join(sentence).strip()
    cleaned_sentence = cleaned_sentence.split()
    text_widget.delete(1.0, ctk.END)
    text_widget.insert(ctk.END, cleaned_sentence)

def update_text_info(sentence):
    text_widget_info.configure(text=' '.join(sentence))

mp_holistic = mp.solutions.holistic
mp_drawing = mp.solutions.drawing_utils

actions = np.array(['I', 'A', 'B', 'Д', 'E', 'C', 'У', 'Ц', 'Ь', ' будинок
', ' дванадцять ', ' Добрий день. ', ' записатись на прийом ', ' лікар ', '
сьогодні ', ' телефон ', ' технологія ', ' хочу ', ' я ', ' Як ваші справи?
'])

### program window ###
window = ctk.CTk()
window.title("Hand Gesture Recognition")

ctk.set_appearance_mode("light")

left_frame = ctk.CTkFrame(window)
left_frame.pack(side=ctk.LEFT)

right_frame = ctk.CTkFrame(window, fg_color="transparent")
right_frame.pack(side=ctk.RIGHT)

cam_frame = ctk.CTkFrame(left_frame)
```

```
cam_frame.pack(padx=20, pady=20)

cam_label = ctk.CTkLabel(cam_frame, text="")
cam_label.pack()

group_frame = ctk.CTkFrame(right_frame)
group_frame.pack(padx=20, pady=20)

text_widget_info = ctk.CTkLabel(group_frame)
text_widget_info.pack()

text_widget = ctk.CTkTextbox(group_frame, wrap="word", height=130,
width=400)
text_widget.pack(padx=20)

remove_last_button = ctk.CTkButton(group_frame, text="Видалити",
command=remove_last_element)
remove_last_button.pack(padx=20, pady=20)

group_frame2 = ctk.CTkFrame(right_frame)
group_frame2.pack(padx=20, pady=20, fill="both", expand=True)

gender_var = ctk.StringVar()
gender_var.set("Чоловічий")
man_radio = ctk.CTkRadioButton(group_frame2, text="Чоловічий",
variable=gender_var, value="Чоловічий")
woman_radio = ctk.CTkRadioButton(group_frame2, text="Жіночий",
variable=gender_var, value="Жіночий")

man_radio.pack(pady=10)
woman_radio.pack()

tts_button = ctk.CTkButton(group_frame2, text="Озвучити", command=to_tts)
tts_button.pack(padx=20, pady=20)

def get_frame():
    global sequence, sentence, predictions, threshold
```

```
ret, frame = cap.read()
if ret:
    image, results = calc.mediapipe_detection(frame, holistic)
    calc.draw_styled_landmarks(image, results)

    if results.left_hand_landmarks or results.right_hand_landmarks:
        keypoints = calc.extract_keypoints(results)

        sequence.append(keypoints)
        sequence = sequence[-30:]
        if len(sequence) == 30:

            res = model.predict(np.expand_dims(sequence, axis=0))[0]
            predictions.append(np.argmax(res))
            if np.max(res) > threshold:
                print(np.max(res))
                action = actions[np.argmax(res)]
                if len(sentence) > 0 and action != sentence[-1]:
                    sentence.append(action)
                    update_text(sentence)
                elif len(sentence) == 0:
                    sentence.append(action)
                    update_text(sentence)
                update_text_info(["Прогноз:", action])
            else:
                update_text_info(["Жест не розпізнано"])
            sequence = []
            #update_text(sentence)
        if len(sequence) == 10:
            update_text_info(["Прогнозування..."])

        # else:
        #     update_text_info([])
    else:
        update_text_info(["Руку не виявлено"])
        sequence = []
    update_cam_label(image)
```

```
    cam_label.after(10, get_frame)

cap = cv2.VideoCapture(0)

with                    mp_holistic.Holistic(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as holistic:
    sequence = []
    sentence = []
    predictions = []
    threshold = 0.5
    model = calc.load_model("USL_Model_new.h5")
    get_frame()
    window.mainloop()

delete_audio_files("audios/")
cap.release()
```