Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: «**Розробка імерсивного перекладача на iOS та visionOS для людей з особливими потребами (Development of an Immersive Translator on iOS and visionOS for People with Special Needs)**»

Виконав: студент 4-го року навчання,

Спеціальності
122 Комп’ютерні науки

Деркач Кирило Геннадійович

Керівник Франків О.О._____
старший викладач

Рецензент _____
(прізвище та ініціали)

Кваліфікаційна робота захищена з оцінкою _____

Секретар ЕК _____
«____» _____ 20___ р.

Київ – 2024

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,

к.ф.-м.н., доц. Гороховський С.С

_____

(підпис)

«___» _____ 2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студенту 4 р.н. бакалаврської програми Комп'ютерні науки

Деркачу Кирилу Геннадійовичу

Розробити програмний застосунок імерсивного перекладача на iOS та visionOS для людей з особливими потребами.

Зміст текстової частини до кваліфікаційної роботи:

Зміст

Анотація

Вступ

Розділ 1. Створення прикладної моделі машинного навчання

Розділ 2. Розробка застосунку для пристроїв Apple

Висновки

Список використаних джерел

Дата видачі «____»_____ 2023 р.

Керівник _____ (підпис)

Завдання отримав _____ (підпис)

Тема: Розробка імерсивного перекладача на iOS та visionOS для людей з особливими потребами (Development of an Immersive Translator on iOS and VisionOS for People with Special Needs)

Календарний план виконання роботи:

№	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на кваліфікаційну роботу	02.11.2023	
2.	Аналіз матеріалів за темою дипломної роботи	16.11.2023	
3.	Тренування глибокої нейронної мережі	07.01.2024	
4.	Розробка програмного застосунку	27.01.2024	
5.	Написання текстової частини	15.03.2024	
6.	Надання роботи на перевірку керівником	20.04.2024	
7.	Корегування роботи на основі результатів перевірки	27.04.2024	
8.	Оформлення роботи та презентації до попереднього захисту	08.05.2024	
9.	Корегування роботи за результатами попереднього захисту	18.05.2024	
10.	Захист кваліфікаційної роботи	30.05.2024	

Студент _____

Керівник _____

«____» _____ _____р.

ЗМІСТ

ЗМІСТ 4
АНОТАЦІЯ 6
ВСТУП 7
РОЗДІЛ 1. СТВОРЕННЯ ПРИКЛАДНОЇ МОДЕЛІ МАШИННОГО НАВЧАННЯ 9
1. Використання машинного навчання для розпізнавання жестів 9
1.1. Історія української жестової мови 9
1.2. Види жестової мови 9
1.3. Машинне навчання 11
1.4. Нейронні мережі 13
1.5. Глибоке навчання та типи глибоких мереж 14
1.6. Опис шарів згорткової нейронної мережі 16
2. Опис реалізації 19
2.1. Обраний інструмент 19
2.2. Тренування власної моделі 24
2.3. Деталі реалізації 34
3. Висновки розділу 36
РОЗДІЛ 2. РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ПРИСТРОЇВ APPLE 37
1. Технології імерсивного досвіду 37
1.1. Доповнена реальність 37
1.2. VisionOS 38
1.3. ARKit 40
2. Опис реалізації 41

2.1. Розробка додатку під iOS 41
2.2. Перенесення на visionOS 45
3. Висновки розділу 48
ВИСНОВКИ 50
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 52

АНОТАЦІЯ

У даній роботі було досліджено розробку глибоких нейронних мереж та проведено тренування власної моделі за допомогою спеціального інструменту MediaPipe, розробленого компанією Google. Після цього було запропоновано реалізацію застосунку для пристроїв Apple, а саме таких систем, як iOS та visionOS, що дозволяє перекладати жести української дактильної абетки в текст та виводити це належним чином у тривимірному просторі у вигляді бульбашки з текстом над головою співрозмовника, що спілкується жестовою мовою. Це досягається у програмі шляхом поєднання технологій машинного навчання та доповненої реальності за допомогою спеціально розробленого алгоритму розпізнавання та виводу, що служить для оптимізації роботи програми та сегрегації введення жестів.

ВСТУП

Постійно актуальною та важливою є проблема складності спілкування людей, що не знають жестової мови, з людьми, що з деяких причин здатні спілкуватися лише жестовою мовою. В Україні для близько 44 тисяч осіб жестова мова є єдиним засобом комунікації та для понад 200 тисяч вона є рідною [1].

У сучасному світі технології стрімко розвиваються, проникаючи в різні аспекти життя людей для полегшення їх повсякденної діяльності. Велика кількість досліджень та рішень наразі створюється в галузі машинного навчання. Не менш актуальною та важливою є технологія доповненої реальності: у лютому 2024 компанією Apple було випущено у продаж гарнітуру доповненої реальності Vision Pro, що потенційно є наступним кроком у розвитку технологій доповненої реальності.

Відповідно до викладеного вище, для врахування потреби в полегшенні зазначеного виду комунікації, вирішено запропонувати власне програмне рішення з імплементацією технологій машинного навчання та доповненої реальності.

Поставлена задача вимагає реалізації завдання класифікації жестової мови, тому у даній роботі буде розглянуто поширені підходи до побудови нейронних мереж та описано реалізацію згорткової нейронної мережі, що використовується для тренування моделі у фреймворку MediaPipe, що було обрано для імплементації у застосунок. В процесі цього дослідним шляхом буде знайдено найефективніший для конкретної моделі, а саме моделі алфавіту української жестової мови, список параметрів, а також описано та протестовано тестову та практичну ефективність створених моделей.

По завершенню створення власної моделі машинного навчання, що здійснює розпізнавання української дактильної абетки, потрібно розробити застосунок, що буде поєднувати в собі технології доповненої реальності від Apple та створену модель для розпізнавання відповідних жестів у кадрі.

Для цього необхідно розробити алгоритм, за яким буде відбуватися розпізнавання цих жестів у кадрі з наступним виведенням розпізнаного тексту в просторі, розробити додаток на базі iOS з імплементацією потрібного функціоналу, а також дослідити можливість перенесення цього рішення на платформу visionOS.

В результаті буде створено відповідний додаток, призначений для використання на системах iOS та visionOS, що буде виконувати динамічне розпізнавання української жестової мови та дозволить спростити комунікацію з людьми з особливими потребами, та описано ключові особливості та структуру цього програмного рішення.

РОЗДІЛ 1. СТВОРЕННЯ ПРИКЛАДНОЇ МОДЕЛІ МАШИННОГО НАВЧАННЯ

1. Використання машинного навчання для розпізнавання жестів

1.1. Історія української жестової мови

Українська жестова мова існує вже понад два століття, з'явившись у спільнотах людей з вадами слуху та спеціальних школах для дітей, що мали вади слуху. Вона розвивалася та культивувалася в родинах людей з особливими потребами з покоління в покоління, створюючи унікальну комунікативну систему для тих, хто не має можливості спілкуватися усно.

Історія досліджень показує, що хоча визнання жестової мови прийшло досить пізно, вона отримала достатню кількість уваги від відомих педагогів, які розуміли важливість її використання в навчальному процесі. Іван Соколянський і Лев Виготський були серед тих, хто висловлював позитивне ставлення до використання даного виду комунікації в навчанні людей з вадами слуху.

Також варто зазначити роботу українських вчених, таких як Рудольф Краєвський, які спробували лінгвістично описати українську жестову мову, відкинувши переконання в тому, що вона є примітивною або застарілою формою комунікації. Українські експерти та вчителі людей з особливими потребами зараз активно працюють над інтеграцією жестової мови у навчальний процес, розробляючи навчальні матеріали та законодавчі документи для досягнення цієї мети [2].

1.2. Види жестової мови

Жестова мова є системою комунікації, яка використовує рухи рук, тіла та обличчя для передачі інформації. Вона є природною мовою, незалежною від звукової мови, і має свою власну мовну організацію.

Вона складається з трьох основних компонентів:

- кінетики (рухи)

- лексики (жести)
- граматики (правила використання жестів)

Жестова мова використовується для вираження різних понять, від предметів до абстрактних ідей, і залежить від просторової організації інформації навколо промовця. Вона є системною і строго впорядкованою, з властивостями, які відрізняють її від вербальної мови, включаючи одночасність передачі інформації рухами.

Українська жестова мова є національною мовною системою, що використовується для комунікації людей з вадами слуху, і не тільки, на території України. Вона походить від французької жестової мови, проте має свої особливості, що є подібними до особливостей писемної та розмовної української мови. З граматичних особливостей можна виділити те, що слова у реченнях зазвичай розташовані у порядку підмет-присудок-означення, а також у порядку означення-підмет-присудок [4].

Однією із допоміжних систем української жестової мови є дактильна абетка. В ній кожній літері української мови відповідає конкретний жест ведучої руки, зігнутої в лікті. Даний спосіб комунікації в базовому розумінні використовується для вираження понять, що не мають жестових позначень, а також для пояснення значень окремих, незрозумілих співбесіднику, слів [5].

Рис. 1.1 Українська дактильна абетка

Такий спосіб комунікації є найбільш універсальним, і тому саме його було обрано для використання в даній роботі. Відповідно, вхідними даними слугуватиме зображення, що може як містити, так і не містити відповідний жест із української дактильної абетки, а вихідними даними буде набір позначок (label) текстового представлення відповідного позначення та відсотків впевненості його розпізнавання.

1.3. Машинне навчання

Машинне навчання – це напрям в галузі штучного інтелекту, що концентрується на вивченні статистичних алгоритмів, що дозволяють навчати нейронні мережі й отримувати з наявних даних нові, невідомі до цього, дані.

Його застосування широко використовується в багатьох сферах, таких як: обробка природного мовлення, економіка, комп'ютерний зір, медицина, розробка автопілотів для транспортних засобів та інші. Головною перевагою машинного навчання є здатність обробляти дуже великі обсяги даних більш чутливо, ніж людина, знаходячи приховані закономірності та надаючи корисну інформацію для прийняття майбутніх рішень [6].

Машинне навчання поділяється на три базові парадигми. Першою з них є парадигма керованого навчання, або навчання з "учителем", що характеризується одночасним наданням як екземплярів вхідних даних, так і правильних вихідних даних з метою встановити явну закономірність, за допомогою якої ці вихідні дані отримуються.

Друга парадигма – некероване навчання, або навчання без "учителя", особливістю якого, на відміну від попереднього виду, є відсутність прикладів вихідних даних у відповідність до конкретних вхідних, що дозволяє виявити неявні закономірності в даних.

І третьою парадигмою є так зване "навчання з підкріпленням", в якому програмі надається список параметрів, значення яких вона повинна максимізувати в процесі взаємодії з середовищем у процесі власного навчання.

Кожна із зазначених парадигм широко використовується для вирішення різних задач, проте жодна не є застосовною для вирішення будь-якої задачі. Проте важливо зазначити, що їх комбінування між собою дозволяє вирішувати все більш комплексні проблеми.

Існує кілька найпоширеніших алгоритмів машинного навчання: лінійна регресія, логістична регресія, кластеризація та дерева рішень. Нейронні мережі моделюють роботу людського мозку за допомогою великої кількості вузлів, що зв’язані між собою. Перевагою нейронних мереж є те, що вони здатні добре розпізнавати закономірності та відіграють ключову роль у задачах розпізнавання зображень, розпізнавання мови та створенні тексту чи зображень.

Лінійна регресія використовується для прогнозування числових значень на основі побудови лінійних відношень між різноманітними параметрами. Найпоширенішим прикладом використання цього алгоритму є будь-яке прогнозування майбутніх фінансових значень, наприклад цін на будинки або чисельності населення в регіоні.

Логістична регресія є алгоритмом навчання з “учителем”, що здійснює прогнози для категорійних змінних, наприклад класифікація спаму в листах чи перевірка якості на виробництві.

Кластеризація ж використовує навчання без “учителя” для визначення закономірностей в даних з наступним їх самостійним групуванням. В основному вона використовується для виявлення відмінностей між даними, що важко відрізнити неозброєним оком.

Такий алгоритм, як дерева рішень позиціонується як універсальний алгоритм, що дозволяє виконувати як прогнозування числових значень (регресію), так і класифікацію даних на категорії. Їх особливістю є наявність послідовності рішень, які можливо представити за допомогою деревовидної діаграми. Від нейронних мереж їх відрізняє можливість перевірки та аудиту, на відміну від прихованості обчислень, що характерна нейронним мережам.

У випадкових лісах прогнозується конкретне значення чи категорія за допомогою поєднання результатів, отриманих в декількох деревах рішень [7].

Кожен з алгоритмів є ефективним для свого класу задач, проте особливу увагу в даній роботі приділено нейронним мережам, тому що саме вони є орієнтованими на задачу класифікації зображень та широко застосовуються в завданнях комп'ютерного зору. Для реалізації моделі машинного навчання, використаної для розпізнавання жестів рук на зображенні використовується згорткова нейронна мережа, більш детальний опис якої відбувається в наступних підрозділах.

1.4. Нейронні мережі

Штучна нейронна мережа (далі – нейронна мережа) – це математична модель, яка намагається імітувати структуру та функції біологічних нейронних мереж мозку. Основним структурним компонентом кожної нейронної мережі є штучний нейрон, проста математична модель (функція), що також відома під терміном перцептрон. Він вперше був запропонований Френком Розенблаттом у 1957 році. Така модель має три прості набори правил: множення, сумування та активація.

На вході штучного нейрона вхідним даним надається вага, що означає, що кожне вхідне значення множиться на індивідуальну вагу. В серединній його частині є функція сумування, що підраховує суму всіх зважених вхідних даних та зміщень. На виході ж обчислена сума проходить через функцію активації, яка також називається передавальною функцією. Дуже часто функція активації робить відображення дійсних чисел на інтервали (-1, 1) або (0, 1) [8].

Задача передавальної функції полягає в тому, щоб нейрон активувався або, навпаки, залишився неактивним, залежно від величини обчисленої суми. Якщо результат сумування перевищує певний поріг, нейрон активується і передає сигнал далі по мережі. У протилежному випадку, якщо результат не досягає порогу, нейрон залишається неактивним і не передає сигнал. Цей механізм

дозволяє нейронам реагувати на вхідні дані та визначати, коли вони повинні активуватися або вимикатися.

Перцептрони здатні вивчати та розрізняти складні шаблони вхідних даних шляхом коригування їх вагових коефіцієнтів під час навчання. Цей процес включає в себе подання великої кількості прикладів даних, аналіз їх виходів та адаптацію вагових коефіцієнтів для досягнення бажаної реакції на вхідні дані [9].

Хоча перцептрони можуть виконувати прості завдання класифікації та регресії, вони є основою для більш складних штучних нейронних мереж, таких як багатошарові перцептрони та глибокі нейронні мережі.

Загальна структура нейронної мережі визначається її топологією, або способом, яким вузли та шари мережі з'єднані між собою. Топологія мережі може варіюватися від простих структур, таких як одношаровий перцептрон, до складних архітектур, таких як згорткові нейронні мережі або рекурентні нейронні мережі. Обрана топологія визначає можливості та обмеження мережі в розв'язанні конкретних завдань, наприклад кількість шарів та нейронів у них впливає на здатність навчання складнішим функціям, однак на перевагу цьому створюються ризики перенавчання та збільшення споживання ресурсів.

1.5. Глибоке навчання та типи глибоких мереж

Глибокі нейронні мережі – це спеціальний клас штучних нейронних мереж, які складаються з багатьох шарів нейронів, які взаємодіють між собою. Основна відмінність глибоких нейронних мереж від звичайних одношарових або двошарових мереж полягає в тому, що вони мають багато прихованих шарів між вхідним та вихідним шарами.

У глибоких нейронних мережах кожен шар нейронів отримує вхідні дані від попереднього шару та генерує вихід, який передається наступному шару. Кожен нейрон у прихованих шарах обчислює лінійну комбінацію вхідних сигналів з використанням вагових коефіцієнтів та застосовує нелінійну функцію активації до отриманого результату [10].

Глибокі нейронні мережі здатні автоматично вивчати складні закономірності вхідних даних та виконувати завдання класифікації, регресії, розпізнавання образів та інші завдання машинного навчання. Вони виявляються особливо ефективними при аналізі великих обсягів даних зі складними структурами, такими як зображення, аудіо, текст чи відео. Глибокі мережі можуть містити десятки, сотні або навіть тисячі шарів та мільйони параметрів, які налаштовуються під час навчання. Даний клас алгоритмів активно використовується в багатьох сферах, включаючи комп'ютерний зір, обробку природної мови, медичну діагностику, фінансовий аналіз та багато інших.

Отже, які типи глибоких нейронних мереж використовуються найбільш широко? Одними з яскравих представників є згорткові нейронні мережі, що використовуються переважно для обробки зображень, а також рекурентні нейронні мережі, що використовуються для обробки тексту та аудіо.

Згорткові нейронні мережі (Convolutional Neural Networks, CNNs) – це спеціальний тип глибоких нейронних мереж, що володіють властивістю автоматичного виявлення локальних шаблонів у зображеннях, таких як границі, форми та текстури. CNN складається з кількох шарів, включаючи згорткові, підсумування, нормалізації, випадання та повнозв’язані шари. Згорткові шари використовують фільтри для виявлення ознак у різних частинах зображення, підсумовуючи результати і передаючи їх наступному шару для подальшої обробки. Наступний підрозділ якраз і буде присвячено даному виду нейронної мережі, адже вона за своїми властивостями є найбільш пристосованою до виконання потрібної для роботи класифікації зображень.

Рекурентні нейронні мережі (Recurrent Neural Networks, RNNs) – тип нейронних мереж, які існують для обробки послідовних даних. Основна відмінність RNN полягає в тому, що вона має зв’язки, що вказують на попередні стани, що дозволяє враховувати контекст та залежності між елементами в послідовності. RNNs використовуються для таких завдань, як: машинний

переклад, генерація тексту та інші задачі, де важлива взаємодія між елементами послідовності.

Окрім згорткових та рекурентних, існують також інші поширені архітектури нейронних мереж. Наприклад, повнозв’язані нейронні мережі (Fully Connected Neural Networks) використовуються для задач класифікації та регресії, де кожен нейрон у кожному шарі з'єднаний з кожним нейроном у наступному шарі. Також існують гібридні моделі, які поєднують в собі різні типи нейронних шарів та архітектурні підходи для досягнення кращої ефективності у вирішенні конкретних завдань.

Не менш важливу роль відіграють параметри нейронної мережі, які необхідно налаштувати під час її навчання. Основні параметри включають ваги зв’язків між нейронами, порогові значення, швидкість навчання та функції активації. Ваги зв’язків і порогові значення постійно коригуються під час навчання з метою оптимізації роботи мережі. Швидкість навчання визначає, наскільки швидко або повільно мережа адаптується до навчальних даних.

Під час навчання нейронна мережа використовується для аналізу навчальних прикладів та пошуку оптимальних параметрів, які дозволяють мережі правильно відповідати на вхідні дані. Цей процес називається зворотнім поширенням помилки і включає в себе ітеративну корекцію параметрів мережі на підставі різниці між очікуваним та фактичним виходом мережі. Після завершення процесу навчання нейронна мережа може бути використана для розв’язання певної задачі, що включає в себе аналіз нових даних та генерацію відповідей або прогнозів [11].

1.6. Опис шарів згорткової нейронної мережі

Як було зазначено раніше, в даній роботі використовується екземпляр згорткової нейронної мережі, тому в даному підрозділі більш детально описано основні відомості про шари, що можуть бути наявні у таких нейронних мережах.

Опис шарів згорткових нейронних мереж варто почати з найголовнішого – згорткового шару (Convolutional Layer). Він є ключовим елементом у глибоких нейронних мережах для обробки зображень. Проявлення його роботи є в тому, щоб обраховувати локальні зважені суми вхідних даних за допомогою фільтрів. Кожен фільтр проходить по вхідному зображенню, виконуючи операцію згортки, яка виокремлює різноманітні патерни та ознаки. Результати згортки утворюють мапи ознак, які подаються на вихід шару для подальшого аналізу.

Згорткові шари також можуть містити параметри, які підлягають навчанню разом з іншими параметрами мережі. Ці параметри визначають внутрішні зв’язки між ознаками та дозволяють мережі вчитися виявляти специфічні ознаки, які корисні для розв’язання конкретної задачі. Глибокі згорткові архітектури можуть містити декілька згорткових шарів, кожен з яких виконує згортку з різними фільтрами, що дозволяє моделі виявляти більш абстрактні та складні ознаки.

Цей тип шарів також має параметри, які визначають крок згортки (зазвичай 1) та розмір вхідних та вихідних зображень. Величина фільтрів та їх кількість також можуть бути налаштовані під час конструювання моделі з метою оптимізації її продуктивності та точності.

Наступним видом шарів згорткової нейронної мережі, використаної в даній роботі є шар пакетної нормалізації (Batch Normalization) – це дуже важливий компонент нейронних мереж, що служить для стабілізації навчання та покращення швидкості збіжності. Основна ідея цього типу шарів полягає в нормалізації вихідних значень вузлів в межах кожної партії даних.

Це допомагає уникнути проблем вибуху градієнту чи навпаки його зникання у процесі навчання, що може спричинити повільну збіжність або нестабільність моделі. Цей шар також створює можливість більш швидкого навчання та покращує загальну ефективність мережі.

Попередня обробка даних нормалізаційним шаром також може допомогти уникнути перенавчання та знизити зайву чутливість моделі до змін в

навчальному наборі даних. В результаті шар пакетної нормалізації стає важливим інструментом для покращення стабільності та швидкості навчання глибоких нейронних мереж.

Не менш важливою є наявність активаційної функції, в даному випадку ReLU. Цей вид активаційної функції широко використовується в глибоких нейронних мережах. Вона виконує операцію активації, яка забезпечує нелінійність та неоднорідність в мережі. Основна ідея ReLU полягає в тому, що вона повертає 0 для всіх від'ємних значень та відповідно саме значення для всіх позитивних вхідних даних [12]:

$$f(x) = max(0, x)$$

Формула 1. Функція активації ReLU

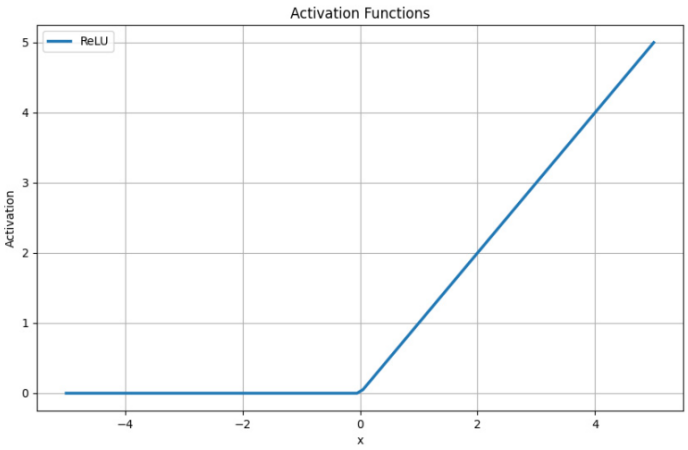

Рис. 1.2 Функція активації ReLU

Це дозволяє активувати лише деякі нейрони, що дозволяє дуже ефективно боротися з нестабільністю навчання моделі через зникання градієнтів. Рівно з цим немає також і проблеми вибуху градієнту, яка властива деяким іншим функціям, таким як сигмоїда або гіперболічний тангенс, і тому дозволяє забезпечити стабільне та ефективне навчання мережі.

Використання ReLU може допомогти підвищити швидкість навчання та покращити загальну продуктивність мережі. Ця активаційна функція дозволяє

створювати більш складні та потужні моделі, які можуть ефективно вирішувати різноманітні завдання машинного навчання.

Після обробки даних нейронами попередніх шарів, відбувається виключення (Dropout) – техніка регуляризації, яка використовується для запобігання перенавчанню в процесі створення моделі. Її головна задача полягає в тому, щоб з випадковою ймовірністю викидати деякі нейрони та їхні зв’язки під час кожного кроку навчання.

Під час навчання, цей шар випадково вимикає деякі нейрони та зв’язки в мережі, змушуючи інші нейрони взяти на себе більше відповідальності за передачу інформації. Це допомагає уникнути перенавчання шляхом зміцнення роботи нейронів, зменшення взаємозалежності та забезпечення більшої рівноваги між ними, оскільки модель не може надмірно підлаштовуватися під конкретні особливості навчального набору даних.

В результаті, модель стає більш універсальною та стійкою, що дозволяє їй краще проводити узагальнення до нових вхідних даних. Це забезпечує більш ефективне використання ресурсів та прискорює процес навчання моделі. Шар виключення може бути особливо корисним для моделей з великою кількістю параметрів та невеликим обсягом даних.

2. Опис реалізації

2.1. Обраний інструмент

Переходячи до опису реалізації задачі створення моделі машинного навчання з метою розпізнавання української дактильної абетки для подальшої імплементації в застосунок, варто звернути увагу на досить потужний інструмент під назвою MediaPipe від Google.

Ця бібліотека уможливлює реалізацію поставленої в даній роботі задачі та є дуже точним інструментом комп’ютерного зору для аналізу зображень з

розпізнаванням ряду передбачених створеними завданнями (task) на них об’єктів.

Як вже було зазначено, MediaPipe – це кросплатформенна бібліотека для обробки медіа-даних в реальному часі, розроблена компанією Google, що дозволяє створювати та налаштовувати моделі для різних завдань машинного навчання, окрім цього вона також надає ряд власних готових простих рішень конкретних задач, як, наприклад, розпізнавання жестів для гри “камінь-ножиці-папір”.

Ця бібліотека написана мовою програмування C++, що забезпечує високу продуктивність та ефективність в обробці медіа. Вона підтримується на різних платформах, таких як Android, iOS, Windows, macOS та Linux, що робить її доступною та універсальною для багатьох задач. Також варто зазначити, що вона містить відкритий вихідний код, тому існує можливість контрибуції в її розробку та адаптування її коду під власні потреби розробника.

Основною функціональністю MediaPipe є використання глибоких нейронних мереж для вирішення різноманітних завдань комп’ютерного зору, для чого використовуються два основних інструменти:

- MediaPipe Tasks: набір інструментів для швидкої інтеграції потрібних завдань в додатки на різних платформах з наявними класичними натренованими моделями.
- MediaPipe Model Maker: інструмент для кастомізації існуючих моделей машинного навчання власними датасетами, в якому наявний вичерпний набір параметрів для потрібного керування навчанням.

На GitHub проєкту його перша релізна версія датується 10 липня 2019 року. За офіційними заявами анонс перших п’яти завдань (англ. tasks; так було названо програмні рішення машинного навчання) було запущено у грудні 2022 року. До них входили: розпізнавання жестів, розмітка частин кисті руки, класифікація зображень, знаходження об’єктів та класифікація тексту. 11 травня 2023 року під

час події під назвою “Google I/O 2023” було анонсовано запуск ще дев’яти нових завдань [13].

Як було зазначено вище, MediaPipe Tasks – це ключовий компонент даної бібліотеки, який надає набір утиліт та API для розгортання швидких та ефективних рішень зі зручним інтерфейсом, що дозволяє легку імплементацію з невеликою кількістю коду.

Однією з ключових її відмінностей від інших інструментів та переваг над ними є те, що бібліотекою використовуються технології апаратного прискорення на CPU, GPU та TPU, що дозволяє значно збільшити її ефективність використання в реальному часі на пристроях різних платформ [16].

Також у завданнях реального часу використовуються спеціальні оптимізаційні алгоритми, що дозволяють знизити навантаження на пристрій шляхом відстежування розпізнаного об’єкту за попередніми кадрами, що зменшує накладні витрати на розпізнавання усього кадру заново.

Серед рішень MediaPipe Tasks можна виокремити перелік реалізацій рішень машинного навчання, таких як: виявлення об’єктів в кадрі, класифікація зображень, сегментація зображень на області для ідентифікації об’єктів чи застосування візуальних ефектів, а також інтерактивна сегментація зображень, розпізнавання жестів, розмічування ключових точок рук, створення числових представлень зображень, виявлення облич та тіла з розпізнаванням їх ключових точок, а також створення зображень облич у різних художніх стилях, що може бути використано для створення віртуальних аватарів користувачів.

Для кожного з названих завдань надаються готові моделі машинного навчання, а також для таких завдань, як виявлення об’єктів, класифікація зображень, розпізнавання жестів та стилізація зображень є можливість створення власної кастомної моделі.

Ключовим в цій роботі є завдання розпізнавання жестів (Gesture Recognition), що дозволяє виявлення та ідентифікацію рухів рук у реальному

часі. Ця технологія дозволяє розпізнавати конкретні жести, використовуючи геометрію рук, що дозволяє досягти більшої точності в цьому напрямі за рахунок обчислень з урахуванням відповідних координат рук.

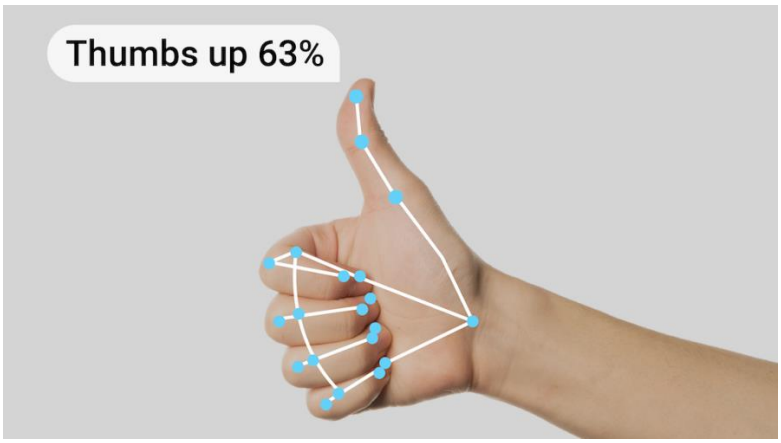

Рис. 1.3 Ілюстрація роботи MediaPipe Gesture Recognition від Google

Результатом роботи моделі є набір ключових точок руки в координатах зображення, інформація про ліву або праву руку та категорії рухів для кожної руки.

Для реалізації цього завдання використовується одразу два пакети моделей: один для виявлення ключових точок руки, а інший для класифікації рухів за ними. Перший пакет визначає геометрію руки, включаючи позицію пальців і структуру долоні, тоді як другий визначає тип руху, наприклад, кулак чи розтягнута долоня.

В прикладі надано готову модель, що працює за допомогою вищеназваної технології для розпізнавання восьми популярних жестів, в якості демонстрації роботи завдання. Проте головною функцією є можливість створення та використання власних моделей для реалізації завдань класифікації різної направленості [21].

Також варто зазначити про завдання розмітки ключових точок рук. Як було описано раніше, воно нерозривно використовується в завданні розпізнавання

жестів і є його фундаментом. Завдання розпізнавання ключових точок рук дозволяє отримати ключові точки рук людини.

Для його створення було використано одразу дві моделі: виявлення долонь та виявлення ключових точок на них. Завдання здатне працювати як із зображеннями, так і потоком в реальному часі. Результатами роботи даної моделі є локації 21 координати ключових точок рук, а також інформація про їх ймовірні сторони (ліва чи права).

Ця модель була навчена на приблизно 30 тисячах реальних зображень, а також на кількох відмальованих комп'ютером синтетичних моделях рук, накладених на різні фони [22].

Другим потужним інструментом бібліотеки є MediaPipe Model Maker. Це інструмент для кастомізації існуючих моделей машинного навчання для роботи з власними даними, необхідними для створення власної моделі машинного навчання для вирішення окремої задачі. Цей інструмент доступний для використання як швидка альтернатива створенню та навчанню нової моделі, якщо завдання не є надто специфічним та не вимагає більш тонкого підходу до проєктування нейронної мережі [14].

Цей інструмент використовує техніку машинного навчання, що називається передачею навчання, яка перенавчає моделі, що вже існують з новими даними. Ця техніка використовує значну частину наявної логіки моделі, що означає, що навчання займає менше часу, ніж навчання нової моделі, і може бути здійснено з меншою кількістю даних.

Model Maker працює з різними типами моделей, включаючи виявлення об'єктів, розпізнавання жестів або класифікатори зображень, тексту чи аудіоданих. Інструмент перенавчає моделі, видаляючи останні кілька шарів моделі, які класифікують дані на конкретні категорії, і перебудовує ці шари за допомогою нових даних, які надає проєктувальник. Model Maker також

підтримує деякі опції для налаштування шарів моделі для покращення точності та продуктивності, про що буде описано нижче.

Навчання моделі за допомогою Model Maker'а, як правило, зменшує розмір моделі, особливо якщо відбувається перенавчання нової моделі для розпізнавання меншої кількості речей. Це означає, що його можна використовувати для створення більш спрямованих моделей, які краще підходять для конкретного додатка. Інструмент також включає в себе застосовування технік машинного навчання, таких як квантизація, що сприяє тому, що модель використовує менше ресурсів і працює ефективніше.

При потребі в навчанні моделі з новими даними, рекомендується мати хоча б приблизно 100 прикладів даних вибірки для кожного класу, що підлягає навчанню, щоб досягти достатньої точності моделі. Не виключається можливість створити ефективну модель навіть із меншою кількістю даних на категорію, проте більший набір даних, як правило, покращує точність моделі. Проте потрібно мати на увазі, що значний дизбаланс в кількості даних може призвести до перенавчання моделі та сильного зниження точності розпізнавання даних з і без того малою вибіркою.

З наявних відомостей також варто зазначити, що при створенні власного навчального набору даних, ці дані розділяються під час процесу перенавчання, зазвичай у пропорції: 80% на навчання, 10% на тестування і решта на валідацію [15].

2.2. Тренування власної моделі

Наступний зміст практичної частини роботи відбувається шляхом ітеративного викладу досвіду виконання автора з усіма інструкціями по обраному методу виконання роботи, послідовним викладом відповідного способу вирішення поставленого завдання та згодом аналітикою долання проблем, що виникають на шляху виконання.

Перед початком роботи над створенням власної моделі, пропонується демонстрація роботи розмічування ключових точок розпізнаних на зображенні рук, щоб наочно продемонструвати з чим буде проводитися робота в подальшому, а опісля буде наведено ключові деталі, які буде враховано при тренуванні моделі розпізнавання жестів з використанням функціоналу бібліотеки MediaPipe.

Спершу, необхідно встановити для середовища виконання Python саму бібліотеку шляхом виконання в терміналі команди “python -m pip install mediapipe” та імпортувати необхідні завдання розмітки ключових точок рук з даної бібліотеки.

Далі відбувається імпорт ще однієї потрібної для відображення в середовищі зображення бібліотеки під назвою matplotlib, і з її допомогою відмальовується завантажене з інтернету зображення людини для демонстрації.

Рис. 1.4 Демонстрація тестового зображення

Після цього ініціалізується заздалегідь завантажене завдання розмітки ключових точок рук та за допомогою функції draw_landmarks_on_image створюється нове зображення з нанесеними на нього результатами розпізнавання.

Рис. 1.5 Демонстрація роботи MediaPipe Hand Landmarker на тестовому зображенні

Як можна побачити на зображенні (Рис. 1.5), модель достатньо чітко справляється як із задачею отримання координат положення руки, так і з класифікацією її сторони (ліва чи права), для довільного зображення, взятого з інтернету.

Отже, наступним кроком потрібно більш детально розглянути яким чином відбувається розпізнавання ключових точок рук та за якими параметрами буде відбуватися навчання (перенавчання) глибокої нейронної мережі при створенні моделі класифікатора жестів.

Дуже поширеним у великій кількості інших моделей машинного навчання для класифікації жестів є спосіб, що відбувається за наступним алгоритмом: нормалізація зображення шляхом утискання в квадратну область відносно невеликого розміру та розбиття на деяку кількість параметрів, кожен з яких характеризує яскравість відповідного пікселя на зображенні. В такого способу є очевидний недолік: модель навчається за допомогою знаходження патернів у секвенціях все ще яскравостей пікселів, що при погіршенні освітлення чи якості камери, при недостатньому розмірі навчального датасету чи недостатній різноманітності варіацій зображень, починає значно втрачати в точності.

З наведеного вище випливає наступне: для тренування глибоких моделей машинного навчання, що орієнтуються на класичне розпізнавання яскравості пікселів вхідного зображення, потрібно мати дуже великий об’єм навчальних даних. Дуже часто кількість дуже різноманітних зображень повинна перевищувати 30 тисяч для досягнення високої точності при тестуванні [28].

Як вже було сказано раніше, в завданні розпізнавання жестів від MediaPipe використовується збірка з двох моделей: розмітки ключових точок рук та класифікації жестів. Вхідне зображення для перенавчення в першу чергу проходить через першу з цих моделей для отримання 21 координатної точки розпізнаної руки, а опісля відбувається аналіз патернів, що присутні серед усіх зображень, віднесених до конкретного класу.

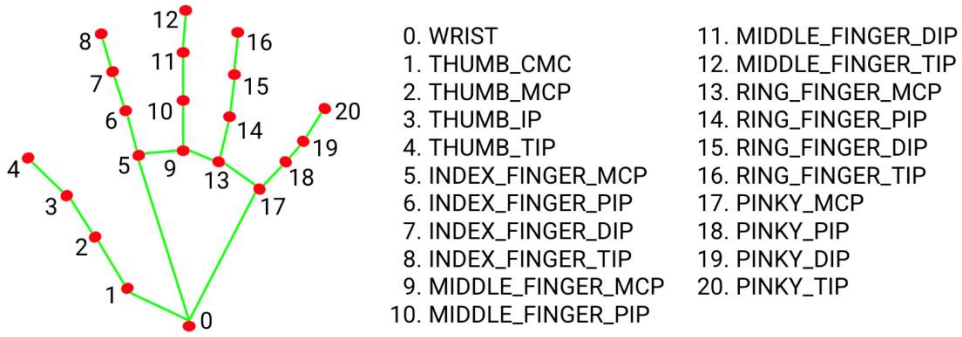

Рис. 1.6 Структура ключових точок, що розпізнаються завданням MediaPipe Hand Landmarker

В загальному при зведенні вхідних даних до розмітки ключових точок руки в результаті отримується 128 різних числових значень, що відповідають наступним параметрам:

- Сторона руки, що відповідає значенню, яке вказує на те, ліва чи права рука була розпізнана.
- Розташування, що складаються з 21 розпізнаної точки, перелік яких можна побачити на зображенні вище. Кожне розташування має координати x, y та z, проте зі спеціальними точками відліку: x та y описують відносне розташування точки на зображенні відносно його

ширини та висоти, проте ці значення нормалізується відносно проміжку [0.0, 1.0]; координата z позначає глибину конкретної мітки відносно зап’ястя на зображенні (чим менше значення, тим ближче до камери знаходиться точка).

- Світові розташування, що позначають координати тривимірного простору (x, y, z) реального світу в метрах з точкою відліку у геометричному центрі руки.

Після розмітки кожне зображення в результаті передається на вхід згортковій нейронній мережі та відбувається перенавчення моделі.

Наступним кроком в роботі є створення вхідних даних для навчання нової моделі. Нижче наведено таблицю кількостей зроблених фотографій, що містять літери абетки української жестової мови:

Таблиця 1 – Кількості фотографій жестів в початковому датасеті

Буква	Кількість	Буква	Кількість	Буква	Кількість	Буква	Кількість	Буква	Кількість
А	25	Є	9	Л	43	С	17	Ч	11
Б	23	Ж	13	М	60	Т	94	Ш	14
В	13	З	14	Н	22	У	14	Щ	10
Г	16	И	13	О	26	Ф	16	Ю	18
Д	10	І	13	П	162	Х	16	Я	23
Е	14	К	14	Р	13	Ц	13	Ь	14

Проаналізувавши таблицю, можна побачити, що для більшості жестів було достатньо зробити усього декілька фотографій, але для деяких з них кількість зображень значно переважає середню з усього набору. Для цього є вагома причина: через велику схожість деяких з жестів, мережі було важко навчитися їх чітко розрізняти. Детальніше про цю проблему буде написано у наступному підрозділі.

В процесі створення датасету було вирішено збільшити кількість фотографій для навчання, з метою збільшити точність роботи моделі, шляхом генерації синтетичних даних.

Так як усі фотографії було зроблено з правою рукою, першим кроком було створення зображень, дзеркально відображених відносно вертикальної осі, для чого було створено функцію horizontal_flip_images, що якраз і відповідає за генерування дзеркально відображених фотографій у вказану папку.

Наступним кроком було створити варіації зображення з різним кутом повороту ліворуч та праворуч, тому що жести можуть бути показані з кутовою похибкою. Завдяки цьому рішенню робиться спроба мінімізувати вплив цього типу похибки на класифікацію.

Реалізація цієї задачі відбувається за допомогою функції rotate_images, в якій створюються повороти для усіх фотографій на 2, 4, 6, 8 і 10 градусів в ліву і праву сторони: для кожного зображення створюються матриці обертання в обидві сторони з центром обертання в центральній точці зображення, кутом обертання у відповідну сторону та рівнозначним масштабом. Потім до зображень застосовуються матриці трансформації та модифіковані зображення зберігаються у вказану папку.

Після завершення роботи зі створення набору даних, можна перейти до етапу тренування моделі розпізнавання жестів на власному датасеті. Для початку роботи необхідно перейти у Google Colaboratory, тому що наразі можна зіткнутися з чималою кількістю проблем сумісності версій бібліотек при спробі відтворити експеримент на власному пристрої.

Після підключення до серверу, спершу потрібно оновити завантажувальник пакетів та встановити на машину пакет mediapipe-model-maker. Далі відбувається імпорт потрібних для роботи бібліотек.

При завантажені набору вхідних даних в даному випадку було використано Google Drive, що дозволяє легко передати необхідні файли на сервер. Для початку

взаємодії з особистим хмарним сховищем потрібно імпортувати відповідну бібліотеку, що входить в пакет google.colab, та розархівувати датасет. Після чого можна записати його місцерозташування в змінну для подальшої роботи.

Тепер потрібно перетворити вхідні дані в датасет, що є зрозумілим для бібліотеки mediapipe та в процесі перетворення в який усі зображення, де запакована за замовчуванням модель розпізнавання рук їх не знаходить, відкидаються та не враховуються в майбутньому. Для цього використовується функція gesture_recognizer.Dataset.from_folder, і потім, згідно рекомендацій, дані розбиваються у згаданих раніше пропорціях: 80% тренувальних даних, 10% для валідації та 10% для тестування.

Варто зазначити, що клас під назвою HandDataPreprocessingParams, екземпляр якого створюється і передається при створенні mediapipe датасету має дві опції, що можна налаштувати, а саме:

- Булеве значення shuffle, що позначає чи перемішувати вхідні дані, має значення true за замовченням.
- Число з плаваючою крапкою min_detection_confidence, що набуває значень від 0 до 1 та відповідає за те, з якою впевненістю розпізнавання дані будуть проходити фільтрацію та залишатися в наборі.

Після того, як дані успішно завантажені та відфільтровані, відбувається тренування моделі, але передусім потрібно зазначити, що процес перенавчення контролюється рядом гіперпараметрів, що тим чи іншим чином впливають на точність результуючої моделі. Контроль цих гіперпараметрів відбувається за допомогою класу GestureRecognizerOptions, при створені екземпляру якого використовуються два екземпляри інших класів: ModelOptions, що відповідає за налаштування самої моделі, та HParams, що слугує для налаштування параметрів, напряму пов’язаних з процесом тренування.

Серед налаштовуваних параметрів ModelOptions:

1) dropout_rate напряму відповідає за згортковий шар Dropout, позначаючи, яка частина вхідних одиниць буде випадково вимкненою в процесі навчання. Використання цієї функції допомагає запобігти перенавчанню шляхом зменшення залежностей між нейронами та узагальнюючи модель.
2) layer_widths визначає список, кожен елемент якого описує ширину кожного додаткового шару, що буде автоматично створено у моделі. До кожного прихованого шару додаються згорткові шари BatchNorm (нормалізація даних), Dropout (виключення нейронів) та ReLU (передавальна функція). Додавання прихованих шарів може допомогти моделі навчатися складнішим закономірностям у даних, що може підвищити її ефективність, але це може збільшити час тренування та ризики перенавчання.

Для контролю над процесом навчання у гіперпараметрах класу HParams представлені:

1) learning_rate (швидкість навчання): визначає, наскільки швидко модель буде навчатися. Висока швидкість може призвести до стрімкого збільшення втрат, в той час як низька швидкість – до повільного навчання та затримки збіжності.
2) batch_size (розмір пакету): характеризує скільки зразків даних використовується для одного оновлення ваг моделі. Збільшення розміру пакету може покращити якість знаходження закономірностей у даних, але занадто велике значення може призвести до погіршення якості моделі шляхом зменшення кількості знайдених закономірностей.
3) epochs (кількість епох): як сказано у назві, визначає скільки разів модель зробить переогляд усіх даних. Збільшення цього значення – можливість покращення точності, проте ризик перенавчання.

4) steps_per_epoch (кількість кроків на епоху): опціональне значення для визначення кількості оновлень моделі на епоху. Якщо не вказувати явно – визначається автоматично діленням кількості даних на розмір пакету.
5) shuffle (перемішування): відповідає за перемішування даних перед кожною епохою. Його увімкнення дозволяє моделі розглядати більш різні між собою залежності.
6) lr_decay (сповільнення навчання): цей параметр визначає наскільки швидкість навчання зменшується з кожною епохою. Він є дуже корисним для збереження стабільності навчання з часом та зменшення проблеми перенавчання.
7) gamma (параметр гамма в функції фокальної втрати): даний параметр відповідає за зміну уваги моделі до рідкісних класів у випадку незбалансованості класів. Він впливає на те, як швидко зменшується втрата для добре класифікованих прикладів (передбачення правильного класу з високою ймовірністю) та як швидко зростає втрата для важких у класифікації прикладів (коли висока ймовірність визначення неправильного класу). Встановлення цього значення меншого за одиницю дозволяє моделі краще розрізняти класи між собою, а також звертати більшу увагу на відокремлення класів з більшою кількістю прикладів [29].
8) export_dir (каталог експорту): шлях, куди зберігати файли контрольної точки моделі та експортовані файли-завдання.

Таким чином, евристичним шляхом, спираючись на наявні дані та інформацію про вплив гіперпараметрів на процес навчання, було натреновано та перевірено більше 20 різних моделей. Після цього було обрано найкращу з них за точністю та протестовано на різноманітних прикладах з інтернету.

По-перше, розгляньмо обрані значення гіперпараметрів та надамо їм аргументовані пояснення. Нижче наведено код, що ініціалізує відповідні налаштування та запускає процес тренування нової моделі.

```
hparams = gesture_recognizer.HParams(learning_rate=0.001,
                                     shuffle=True, gamma=0.4,
                                     batch_size=20, epochs=100,
                                     export_dir="exported_model_v18")
model_options = gesture_recognizer.ModelOptions(dropout_rate=0.001, layer_widths=[31, 31, 31, 80])
options = gesture_recognizer.GestureRecognizerOptions(model_options=model_options, hparams=hparams)
model_2 = gesture_recognizer.GestureRecognizer.create(
    train_data=train_data,
    validation_data=validation_data,
    options=options
)
```

Рис. 1.7 Лістинг коду ініціалізації та запуску тренування моделі

Значення швидкості навчання було обране за замовчуванням, 0.001, тому що воно вважається оптимальним для балансу між хорошою якістю навчання та мінімальним ризиком перенавчання. Так само й зі значенням коефіцієнту сповільнення навчання: його значення встановлене за замовчуванням та рівне 0.99, що є оптимальним протягом усього шляху навчання.

В процесі навчання відбувається перемішування, тому що це дозволило зменшити залежності між конкретними прикладами та покращити універсальність розпізнавання.

Через особливість значень деяких класів, а саме їх високу подібність, значення гамма було встановлено 0.4, що збільшило акцент на розпізнаванні відмінностей між такими екземплярами.

Розміру пакету було надано значення, що відповідає 20 екземплярам, що балансує між достатньою кількістю тренувань та достатньою кількістю розглянутих залежностей на кожному етапі.

Навчання відбувається в 100 епох, що дозволяє виявити достатню для ефективної класифікації кількість залежностей. Більше значення цього параметра могло призвести до нерівноцінного обміну: відсутність помітного впливу на навчання на противагу збільшенню ризику перенавчання.

Частоті виключення нейронів було встановлено значення 0.001, що є відмінним від рекомендованого 0.05 за замовчуванням. Це аргументовано тим,

що в процесі навчання не створюється надто велика кількість залежностей та більшість нейронів є важливими для подальшого тренування.

Розміри прихованих шарів експериментальним шляхом було визначено списком [31, 31, 31, 80], розмір 31 відповідає за додаткові вивчення закономірностей у вхідних даних, а 80 був обраний для створення більш потужного розгалуженого шару, що здатен вивчити більш складні закономірності моделі, забезпечуючи її більшою гнучкістю.

Для виконання перевірки моделі на практиці та оцінки її реальної точності було створено відповідний код, що методом тестування двох моделей на справжніх зображеннях порівнює їх точності загалом. Шляхом даного тестування й було обрано найкращу зі створених моделей, що, за оцінкою при тестуванні показала точність у 94.7%, що є достатньо високою для використання в режимі прямого потоку з достатньою ефективністю.

2.3. Деталі реалізації

Після завершення тренування моделі варто наголосити, що в процесі виконання цього завдання виник ряд проблем, обрані шляхи подолання яких описано в цьому підрозділі.

Однією з найголовніших з них була проблема подібності таких жестових позначень літер, як: П, Т, Л і М – що перешкоджало навчанню моделі розрізняти їх між собою. Через їх схожість впевненість розпізнавання кожної падала до 50%, що значно погіршувало якість моделі та впливало на рівномірність її навчання. Нижче подано зображення візуального позначення цих літер для розуміння проблематики.

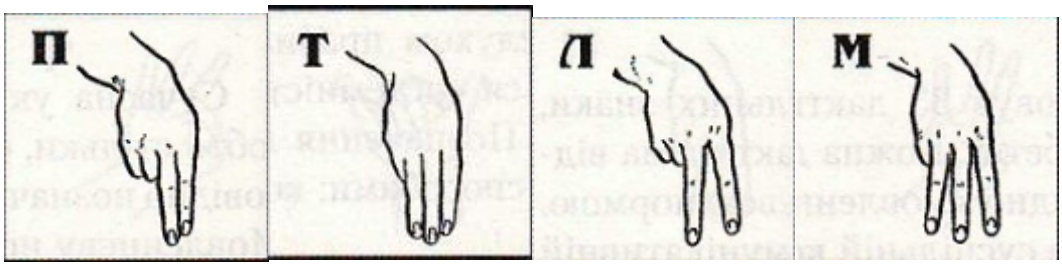

Рис. 1.8-1.11 Ілюстрації схожих букв української жестової мови

Як можна побачити, тут наявні дві проблеми для розпізнавання нейронною мережею цих позначень:

1) Дуже невелика відмінність позиціонування руки: попарно жести відрізняються лише випрямленістю безіменного пальця та наявністю простору між випрямленими пальцями.
2) Невидимість у кадрі більшості ключових точок: через те, що рука направлена донизу, у переважній більшості модель розмітки “додумує” відповідні координати.

Для вирішення цих двох проблем було прийнято один суцільний крок, що одразу поєднує в собі дві різні за спрямуванням дії: зробити більшу кількість екземплярів фотографій саме цих жестів, при цьому показуючи їх в невеликому повороті відносно камери для того, щоб розмічувальник міг більш точно позначити координати ключових точок на руці.

Загалом дана методика є чимось між техніками насичення зразками (oversampling) та видалення зразків (undersampling). В той час, як насичення зразками має за мету збільшити кількість прикладів класів з малою кількістю екземплярів, а видалення зразків спрямована навпаки на видалення прикладів класів з кількістю зразків, що переважає в наборі, це рішення спрямоване на те, щоб збільшити кількість екземплярів важко розпізнаваних класів з метою сконцентрувати більшу увагу моделі на розрізнення ознак таких класів.

У такий спосіб було досягнуто значне згладжування або навіть усунення невпевненості моделі при розпізнаванні відповідних літер. У попередньому

підрозділі було описано загальну кількість результуючих зразків для кожного класу відповідно (див. Таблицю 1).

Другою проблемою було досягнення крайньої оцінки точності моделі приблизно у 92%, що було явно недостатнім значенням, тому в результаті евристичним шляхом було проведено ряд різних експериментів з гіперпараметрами, що в результаті дозволило підвищити точність моделі до фінального значення, зазначеного в описі результуючої моделі.

3. Висновки розділу

У цьому розділі кваліфікаційної роботи було досліджено методи машинного навчання та оглянуто такий потужний інструмент як MediaPipe від Google для подальшого створення моделі розпізнавання жестів української дактильної абетки.

Було створено власний набір даних для тренування моделі машинного навчання, а також використано відповідні техніки розширення датасету шляхом синтетичної генерації зображень.

Використовуючи відповідні інструменти було проведено перенавчення моделі розпізнавання жестів MediaPipe, в процесі чого було проаналізовано та експериментальним шляхом обрано оптимальний набір гіперпараметрів для навчання необхідної глибокої нейронної мережі.

РОЗДІЛ 2. РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ПРИСТРОЇВ APPLE

1. Технології імерсивного досвіду

1.1. Доповнена реальність

Технологія доповненої реальності (AR) є однією з найбільш перспективних у світі сучасних технологій, за словами генерального директора компанії Apple, Тіма Кука. Вона дає змогу об’єднувати в собі реальний та віртуальний світи, забезпечуючи нові можливості сприйняття інформації та взаємодії з навколишнім середовищем.

Основна ідея доповненої реальності полягає в тому, щоб доповнювати реальний світ цифровими об'єктами та інформацією, що полегшує сприйняття інформації та покращує взаємодію з навколишнім середовищем. Вона може бути реалізована через різноманітні технології та пристрої, включаючи смартфони, планшети та спеціальні AR-гарнітури.

Один з основних інструментів для створення AR-досвідів в екосистемі Apple – RealityKit, який надає розробникам більше контролю над AR-додатками та дозволяє створювати інтерактивні AR-сценарії з використанням власних алгоритмів та графіки. Крім того, інструменти, такі як Reality Composer та Reality Converter, полегшують процес створення та конвертації 3D-моделей для подальшого використання в додатках доповненої реальності.

AR Quick Look є ще одним інструментом, який дозволяє швидко та просто відображати 3D-об'єкти в реальному середовищі за допомогою смартфонів та планшетів. Це дає змогу створювати і ділитися інтерактивними AR-досвідами зі споживачами.

Використання технології доповненої реальності в різних сферах, таких як виробництво, дозволяє покращити ефективність та безпеку роботи, завдяки можливості відображення інформації про об'єкти та процеси безпосередньо на реальній площині. Наприклад, AR може бути використана для діагностики

промислового обладнання або навчання персоналу, що забезпечує підвищення продуктивності та зниження ризиків неполадок [30].

1.2. VisionOS

visionOS – це операційна система для гарнітури змішаної реальності Vision Pro, яка була розроблена компанією Apple. Перше офіційне повідомлення про неї було оголошено 5 червня 2023 року під час всесвітньої конференції розробників Apple. Півроку потому, пристрій став доступним для попереднього замовлення 19 січня 2024 року, а придбати його офіційно можна було з 2 лютого 2024 року. Vision Pro позначають як “просторовий комп’ютер”, який поєднує цифрові технології з реальним світом, дозволяючи взаємодіяти з системою за допомогою різних фізичних вхідних методів, таких як рухи, погляд та мовлення [31].

Операційна система visionOS базується на каркасі системи iOS, але вона має тривимірний інтерфейс користувача та підтримку роботи з декількома задачами через вікна, що “висять” просторі навколо користувача, що сприяє зручному використанню пристрою. Також системою підтримується генерація аватарів на основі сканування обличчя користувача, що в подальшому використовується для взаємодії як з віртуальним, так і з реальним середовищем.

Для розробки додатків під visionOS використовується фреймворк SwiftUI, який надає підтримку тривимірних інтерфейсів, ефектів, іммерсивних сцен та створення кастомних жестів. При відтворенні тривимірного контенту та ефектів використовується фреймворк RealityKit, що дозволяє взаємодію з ним та інтегрується з ARKit, який надає можливості розпізнавання навколишнього простору з оцінкою площин та відстеженням об’єктів, що відкриває широкий спектр можливостей взаємодії з реальним середовищем [32].

При розробці на visionOS виділяють 12 термінів, якими оперує розробник при створенні застосунків [33]:

1) Вікна (Windows): Вікно відображає 2D або 3D контент в контейнері і може бути змінюваним за розміром.

2) Об’єми (Volumes): Об’єм відображає 3D контент, який можна переглядати з будь-якого кута, наприклад, продукти на електронному комерційному веб-сайті.
3) Орнаменти (Ornaments): Орнамент містить елементи управління або інформацію, пов’язану з вікном.
4) Панелі інструментів (Toolbars): Панель інструментів містить часто використовувані елементи керування для виконання дій на поточному екрані.
5) Панелі вкладок (Tab bars): Панель вкладок використовується для навігації між різними розділами додатка.
6) Бічні панелі (Side Bars): Бічна панель розгортається з боку вікна, коли вибрана вкладка в панелі вкладок.
7) Меню і спливаючі вікна (Menus and Popovers): Меню та спливаючі вікна можуть розширюватися за межі вікна.
8) Аркуші (Sheets): Аркуш – це модальне вікно, яке з’являється перед батьківським вікном і запитує від користувача певну інформацію або представляє просте завдання.
9) Простори (Spaces): Є загальний простір і повний простір. Вони використовуються для створення різних іммерсивних досвідів.
10) Спектр занурення (Immersion Spectrum): Складається з вікна, панорамного в’ю та оточення - дозволяє програмам плавно переходити між різними станами.
11) Прозорість (Passthrough): Прозорість дозволяє користувачам бачити своє фізичне оточення через камери пристрою.
12) Переходи та плавні анімації (Transitions & Subtle Animations): Важливо проєктувати дизайн з точки зору плавних переходів для створення неперервності між різними станами досвіду.

1.3. ARKit

ARKit – це набір інструментів розробника, що надається компанією Apple разом з рештою базових фреймворків, що входять в iOS SDK та visionOS SDK, для створення додатків доповненої реальності на платформах iOS, iPadOS та нещодавно випущеної visionOS.

За допомогою ARKit розробники можуть легко інтегрувати функціонал доповненої реальності у свої додатки, що дозволяє користувачам взаємодіяти з віртуальними об'єктами в реальному середовищі, використовуючи камеру та сенсори свого пристрою. ARKit надає можливості, такі як відстеження руху, визначення положення у просторі, відтворення 3D-об’єктів та знаходження поверхонь, що дозволяє створювати інтерактивні AR-досвіди для користувачів.

Однією з ключових функцій ARKit є відстеження поверхонь, що дозволяє додаткам відтворювати віртуальні об’єкти на реальних поверхнях, таких як підлога або стіни. Це робить можливим створення інтерактивних AR-сценаріїв, де користувачі можуть взаємодіяти з віртуальними об'єктами у реальному середовищі.

ARKit використовує таку технологію, як SLAM (Simultaneous Localization and Mapping), яка дозволяє пристрою визначати своє місцезнаходження в просторі та створювати карту навколишнього середовища шляхом аналізу камерних зображень та відстані між об’єктами. Крім того, фреймворк використовує технологію комп’ютерного зору для відстеження рухів обличчя, тіла та розпізнавання об’єктів у реальному часі.

З доступних можливостей для взаємодії ARKit з навколишнім середовищем на visionOS описують [34]:

1) Виявлення площин: дозволяє виявляти поверхні в навколишньому середовищі користувача для закріплення контенту на них.

2) Відстеження світу: шляхом визначення положення пристрою відносно оточення дозволяє розміщувати контент у просторі, що гарантує його закріплення на відповідних точках.
3) Відстеження рук: дана функція шляхом визначення позицій рук та пальців користувача дозволяє використання кастомних інтерактивних жестів для взаємодії з системою.
4) Відтворення сцени: ARKit сканує навколишнє середовище користувача та відтворює його віртуальну 3D-модель, яку можна зберегти та використовувати в подальшому.
5) Відстеження зображень: функція, що дозволяє виявляти відомі зображення в оточенні користувача та використовувати їх для закріплення в середовищі нового контенту.

2. Опис реалізації

У даному підрозділі описується шлях розробки додатку з імплементацією раніше створеної моделі машинного навчання, орієнтованої на розпізнавання абетки української жестової мови. Спочатку описується створений та готовий до використання додаток iOS з відповідними акцентами на найважливіших ключових класах, а потім – опис інструментів та досвіду, отриманого від спроби перенесення цього додатку на платформу visionOS, з висновками та аргументацією її неможливості в конкретному випадку під час написання цієї роботи.

2.1. Розробка додатку під iOS

Для початку викладення процесу розробки додатку спершу зазначаються його архітектурні особливості та потім шляхом заглиблення описується структура класів та вказуються описи їх найважливіших в даній роботі екземплярів.

Проєкт реалізовано за архітектурним шаблоном MVVM (Model-View-ViewModel), головним принципом якого є розділення класів на три підгрупи,

особливість взаємодії між якими полягає в тому, що Model описує дані, що використано в застосунку, та мінімальний функціонал для роботи з ними, View відповідає за графічне представлення інтерактивних елементів користувачу, а ViewModel поєднує роботу попередніх двох груп, вміщуючи в собі конкретний стан даних та демонструючи необхідну обчислену інформацію на графічному інтерфейсі.

Серед View частини додатку присутні усього декілька компонентів, що в загальному формують один суцільний графічний інтерфейс.

ContentView є найголовнішим компонентом та слугує для загальної демонстрації контенту, відображення зображення з камери користувача, роботи з його відсутністю та презентації відповідного попередження про це.

ContentViewController та ContentViewControllerRepresentable слугують для виконання двох основних задач: з’єднання між собою використаних в додатку Swift фреймворків графічного інтерфейсу UIKit та SwiftUI, а також для виконання ініціалізації сесії доповненої реальності з конфігурацією ARBodyTrackingConfiguration, що є однією з підтримуваних конфігурацій ARKit на iOS, яка слугує для автоматичного розпізнавання та відстеження людського тіла в кадрі з можливістю отримання тривимірних координат його ключових частин, що в подальшому буде використано в спроєктованому алгоритмі динамічного введення користувачем жестів української дактильної абетки.

Імплементація даного функціоналу відбувається наступним чином: ContentViewController запускає сесію доповненої реальності, відображає це на основному інтерфейсі та делегує управління нею класу ContentViewModel, що відповідає за взаємодію з даними.

З ключових елементів класу ContentViewModel слід акцентувати увагу на полі imageProcessor, що має тип класу USLImageProcessor. Цей клас відповідає за обробку кадрів у якості вхідних зображень та передає відповідальність далі по ланцюжку. Деталі його реалізації буде описано трохи згодом. Наразі важливо

взяти до уваги те, що при ініціалізації USLImageProcessor ContentViewModel передає йому замикання-хендлер, що відповідає за наступні зміни у середовищі: у recognizedHandsInfo записується уся отримана в результаті класифікації інформація про розпізнавання жестів у кадрі, після чого в спеціальний сервіс для обробки та виводу даних про розпізнавання на вхід передається розпізнаний на поточний момент часу жест.

Обчислювана властивість currentGesture за інформацією відслідковуваної у конкретний момент часу руки та розпізнаних жестів отримує відповідне значення жесту, що є розпізнаним наразі.

Не менш важливим структурним елементом програми є клас ARViewModel, що відповідає за усю взаємодію з доповненою реальністю та відображенням тексту, введеного користувачем у тривимірному просторі.

Даний клас зберігає в собі декілька полів: приватні поля описують ноду, що зберігає в собі об’єкт, що складає собою таку собі бульбашку з текстом, яка відображається при введені користувачем жестів, а решта – поточний розпізнаний жест, відсилку на ContentViewModel, екземпляр RecognitionOutputService, функціонал якого описується пізніше, а також відстежуваний якір розпізнаного в кадрі тіла людини.

Реалізовані у класі методи реалізують функціонал ініціалізації текстової бульбашки та її модифікації, а також метод handleHands описує алгоритм взаємодії з нейронною мережею для розпізнавання жестів наступним чином: якщо хоча б одна рука піднята (кут між передпліччям та рукою менше або дорівнює 90 градусів), то поточний кадр передається до обробника зображень USLImageProcessor, який в свою чергу запускає класифікацію та передає результуючі дані в сервіс обробки вихідного тексту.

Щодо частини Model, вона описує подання усієї необхідної для функціонування додатку інформації, включаючи переведення назв класів у літери української мови та визначення зігнутості руки шляхом обчислення кутів

між векторами з кінцями у відносних координатах плеча, ліктя та зап’ястя на вході (Рис. 2.1).

```
struct ARBodyInfo {

    static func isHandRaised(shoulderTransform: simd_float4x4, elbowTransform: simd_float4x4, wristTransform: simd_float4x4) -> Bool
        {
        let shoulder = simd_make_float3(shoulderTransform.columns.3)
        let elbow = simd_make_float3(elbowTransform.columns.3)
        let wrist = simd_make_float3(wristTransform.columns.3)

        let shoulderToElbow = normalize(elbow - shoulder)
        let wristToElbow = normalize(elbow - wrist)

        let dotProduct = dot(shoulderToElbow, wristToElbow)
        let angle = acos(dotProduct)

        let angleInDegrees = angle * 180.0 / .pi
        return angleInDegrees <= 90.0
    }
```

Рис. 2.1 Лістинг коду структури ARBodyInfo

Виклад реалізації завершується на описі спеціальних сервісів, що були створені для забезпечення функціональної взаємодії частин застосунку зі створеною моделлю машинного навчання української дактильної абетки та її імплементації в систему iOS.

Отже, як було сказано, USLImageProcessor відповідає за перетворення зображення у формат MPImage, що є частиною фреймворку MediaPipe, який було встановлено в проєкт шляхом імпорту відповідного поду CocoaPods додаванням в Podfile залежності «MediaPipeTasksVision» та запуску команди pod install в терміналі.

Після успішного створення MPImage з отриманого кадру у вигляді зображення обробник зображень передає його далі по ієрархії до USLGestureRecognizer’а.

Цей клас ініціалізує екземпляр GestureRecognizer від MediaPipe з відповідними параметрами, що містять кількість розпізнаваних в кадрі рук, вказання режиму розпізнавання (прямий потік) та шляху розташування моделі, яку використовує розпізнавач для класифікації. Також даний клас вимагає призначення делегату, що буде обробляти результати розпізнавання, тому наступним в ієрархії є USLRecognizerResultProcessor, який виконує необхідні дії

та передає оброблені дані в замикання, що вказувалося при створенні екземпляра USLImageProcessor’а в ініціалізаторі ContentViewModel.

І останнім з сервісів є безпосередньо сервіс для обробки та виводу даних про розпізнавання, що містить в собі буфер з розпізнаних протягом підняття руки жестів, поточне значення виводу тексту, а також набір методів для автоматичної обробки надходження нового вхідного жесту та отримання розпізнаної за весь останній час обробки літери.

На цьому опис процесу розробки iOS застосунку завершується. В сукупності цей набір класів утворює працездатний та корисний додаток, що цілком реалізує поставлену задачу спрощення комунікації з людьми з особливими потребами, які з тих чи інших причин здатні до спілкування лише жестовою мовою. Наступні ілюстрації демонструють роботу готового застосунку.

2.2. Перенесення на visionOS

Перш за все, варто відмітити те, що, на момент написання роботи, використана технологія відстежування рухів тіла людини, що є інструментом фреймворку ARKit, все ще є непідтримуваною на visionOS, що наразі не дає можливості відтворити такий самий алгоритм не тільки розташування тексту над людиною в тривимірному просторі, а й отримувати координати її ключових точок рук, що унеможливлює відтворення алгоритму розпізнавання, використаного у iOS частині додатку.

Одним із ключових моментів, вартих уваги є те, що iOS частина додатку була розроблена на основі технології розпізнавання скелету людини, що є частиною ARKit, проте на момент написання використаний функціонал не є доступним в поточній версії операційної системи продукту visionOS. Тому наразі цей чинник ускладнює перенесення застосунку з iOS на visionOS.

У відповідності до поданих вище обмежень було обрано інший спосіб відображення тривимірного тексту, що показує результати розпізнавання жестів.

Для цього було розроблено скелет додатку під visionOS, в якому відповідний label знаходиться на деякій відстані від користувача та є прив’язаним до руху камери.

Також при спробі імпортування фреймворку MediaPipe в систему visionOS загальноприйнятим можливим шляхом – CocoaPods, було отримано повідомлення про те, що до поду на даний момент явно не вказано підтримку цього фреймворку. Тому для подальшого перенесення було обрано шлях збірки фреймворку MediaPipe на власному пристрої.

Наступний опис дій, що було виконано автором для спроби збірки відповідної бібліотеки для платформи visionOS, показує, які кроки було зроблено для того, щоб досягнути потрібної мети, після чого приводиться необхідна аргументація неможливості її довершення в силу ряду обставин, що унеможливлюють портування застосунку на visionOS в період часу написання даної роботи з наступним описом перспектив подальшої розробки.

В загальному, для встановлення та збірки MediaPipe використовується два інструменти: Bazelisk та Tulsi.

Bazelisk – це обгортка для інструменту під назвою Bazel, написана на мові програмування Go. Вона автоматично вибирає підходящу версію Bazel залежно від поточного робочого каталогу, завантажує її з офіційного сервера та передає всі необхідні аргументи командного рядка до реального бінарного файлу Bazel.

Bazel – це інструмент для збирання та тестування програмного забезпечення, який подібний до Make, Maven та Gradle. Він використовує просту мову для опису процесу збирання проєкту та має декілька переваг, серед яких виокремлюється використання абстрактної мови, яка полегшує опис властивостей збірки проєкту, що дозволяє працювати на рівні бібліотек, виконуваних файлів, скриптів та наборів даних, знижуючи складність роботи з компіляторами та лінкерами. Інструмент працює на різних операційних системах і може створювати вихідні файли для різних платформ [37].

Tulsi – це інструмент, який використовує інформацію з файлів BUILD Bazel для створення проєктів Xcode. За допомогою нього можна компілювати та підписувати бінарні файли за допомогою Bazel, що дозволяє забезпечити однаковий результат, незалежно від того, через що відбувається компіляція: через Tulsi чи за допомогою командного рядка. Tulsi також дозволяє працювати з підмножиною вихідних файлів проєкту, що може значно зменшити час індексації файлів у Xcode, особливо для великих проектів. Крім того, цей інструмент автоматично враховує будь-які зміни, внесені до файлів BUILD, такі як додавання нових залежностей від бібліотек, під час збірки проєкту [38].

Отже, для початку відбувається встановлення Bazelisk та бібліотеки “six” для Python, а також клонування репозиторію MediaPipe на пристрій.

Потім обов’язково відбувається лінкування профілю розробника з файлом, що знаходиться в папці Mediapipe за шляхом mediapipe/mediapipe/provisioning_profile.mobileprovision. Для цього з директорії MediaPipe в терміналі прописуються відповідні команди.

Також паралельно до цього у файлі за шляхом: mediapipe/examples/ios/bundle_id.bzl – вказується префікс, що відповідає використаному профілю розробника, в полі BUNDLE_ID_PREFIX.

Наступною дією було встановлення Tulsi, що буде використано для спроби збірки фреймворку на архітектурі visionOS. Для цього спочатку відбувається клонування репозиторію Tulsi з GitHub з наступним запуском файлу сценарію оболонки Bash під назвою «build_and_run.sh».

Проте, перед запуском build_and_run.sh необхідно змінити у файлі з назвою WORKSPACE команду імпорту архіву build_bazel_rules_apple на новішу версію через несумісність застарілої версії Tulsi з новою версією Bazel (Рис. 2.2).

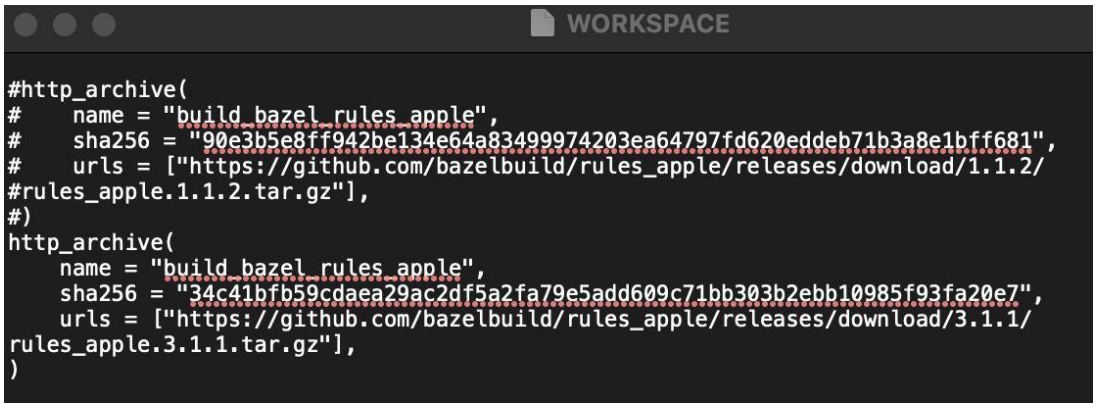

Рис. 2.2 Опис заміни застарілої залежності build_bazel_rules_apple

Ця проблема виникає через те, що інструмент Tulsi більше не підтримується та він мігрував на нову систему правил збірки проєктів XCode під назвою rules_xcodeproj, проте в MediaPipe ще використовується саме Tulsi, тому це неочевидне рішення проблеми є необхідним для того, щоб продовжити роботу в контексті фреймворку та поставленої мети.

Далі відбувається відкриття Tulsi та всередині нього відкривається файл, що знаходиться за шляхом mediapipe/Mediapipe.tulsiproj. Обирається конфігурація з назвою «MediaPipe» та нажимається кнопка Generate, після чого необхідно обрати шлях збереження згенерованого проєкту Xcode із зібраним фреймворком.

Проте збірка динамічної бібліотеки @--mediapipe-objc-mediapipe-framework-ios на симуляторі Apple Vision Pro наразі виявилася неможливою, для чого є об’єктивна причина: на момент написання роботи MediaPipe не має підтримки visionOS через несумісність залежностей. Тому MediaPipe буде доступним для інтеграції у створений додаток під visionOS за умови майбутньої появи цього функціоналу для цієї системи.

3. Висновки розділу

В даному розділі було описано процес розробки додатку для комунікації людей, що мають особливі потреби, на пристроях Apple. Було реалізовано повноцінний працездатний застосунок для платформи iOS, що вирішує

поставлену задачу та поєднує в собі технології доповненої реальності, доступні завдяки фреймворку ARKit, а також машинного навчання з використанням описаного в даній роботі потужного інструменту MediaPipe, що використовує описану та розроблену в попередньому розділі модель машинного навчання, що орієнтована на класифікацію та розпізнавання української дактильної абетки.

В процесі розробки додатку було спроєктовано та описано чітку архітектуру проєкту, що є ефективним та оптимізованим рішенням поставленої задачі, а також розроблено спеціальний алгоритм розпізнавання безперервного вводу жестів користувачем з подальшим аналізом отриманих даних та виведенням їх у просторі. А також було розглянуто перспективу майбутнього перенесення цього функціоналу на платформу visionOS для створення позитивного досвіду занурення, актуальність розробки застосунків під який наразі має стрімке зростання.

Було розглянуто та використано ряд інструментів, що слугують для збірки кросплатформених проєктів, таких як Bazelisk, Bazel, Tulsi, а також проведено спробу їх використання для додавання підтримки фреймворку MediaPipe на visionOS. Як вже було аргументовано в роботі, використані для розробки iOS частини додатку інструменти наразі не є підтримуваними системою visionOS, тому в майбутньому розглядається або можливість повноцінного портування застосунку з використанням наявних інструментів за умови появи їх підтримки, або переосмислення структури проєкту з використанням іншого списку інструментів, адже запропоноване рішення імплементує лише один з великої кількості можливих наборів інструментів, яких з часом у майбутньому буде лише більшати.

ВИСНОВКИ

У даній роботі було проаналізовано та використано ряд ефективних інструментів, що служать для реалізації таких завдань, як тренування власних нейронних мереж та імплементація цих рішень в додатках на різних платформах, а також фреймворк для розробки інтерактивних додатків з фокусом на доповненій реальності – ARKit.

Було проведено генерацію власного датасету з використанням технік генерації синтетичних даних, а також тренування власної глибокої нейронної мережі для розпізнавання української дактильної абетки з послідовним аналізом її покращень, до яких входить корегування гіперпараметрів навчання, а також аналізом проблем, що виникли в процесі навчання, та використаних технік усунення цих проблем, а саме введення керованого дизбалансу моделі в сукупності з розширенням прихованих шарів для збільшення точності розрізнення менш явних патернів серед класів. Було проведено тестування, за результатами якого обрано модель з найвищим показником точності при практичному застосуванні.

Після цього було успішно створено власне програмне рішення, що слугує для допомоги в спілкуванні та розумінні людей, що мають особливі потреби в комунікації жестовою мовою та обмежені нею з тих чи інших причин. Застосунок поєднує в собі використання технологій доповненої реальності та машинного навчання, що в сукупності з розробленим алгоритмом розпізнавання та виводу представляє собою ефективне, високотехнологічне та актуальне рішення поставленої задачі.

Окрім цього було проведено спробу перенесення додатку з архітектури iOS на архітектуру visionOS, в процесі чого було розглянуто інструменти, що використовуються в MediaPipe для збірки фреймворку на власному пристрої та використано їх у відповідності до заданої потреби. У звіті про виконання цієї частини роботи описано проблеми, що виникли на шляху досягнення цієї мети,

шляхи їх вирішення, та сформовано висновок про те, що необхідний функціонал поки що не є підтримуваним, але, за появи підтримки необхідних інструментів, реалізоване на системі iOS рішення в майбутньому має перспективу повного перенесення на архітектуру visionOS.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Міністерство соціальної політики України. 23 вересня – Міжнародний день жестових мов [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.msp.gov.ua/news/20689.html.

2. Кульбіда С. Use and Study of Sign Language in Ukraine: History of Research / докт. пед. наук Кульбіда Світлана – 2004.

3. Goldin-Meadow S. Gesture, sign, and language: The coming of age of sign language and gesture studies [Електронний ресурс] / S. Goldin-Meadow, D. Brentari // Cambridge University Press. – 2015. – Режим доступу до ресурсу: https://www.cambridge.org/core/journals/behavioral-and-brain-sciences/article/gesture-sign-and-language-the-coming-of-age-of-sign-language-and-gesture-studies/40B9B8E3C35C7005D4D588EC39E34C80.

4. Кульбіда С. Українська жестова мова як природна знакова система / докт. пед. наук Кульбіда Світлана – 2009. – С. 218-237.

5. Дактилологія / В. В. Засенко, С. В. Кульбіда // Енциклопедія Сучасної України [Електронний ресурс] / Редкол. : І. М. Дзюба, А. І. Жуковський, М. Г. Железняк [та ін.] ; НАН України, НТШ. – К. : Інститут енциклопедичних досліджень НАН України, 2007. – Режим доступу: https://esu.com.ua/article-23383

6. Crabtree M. What is Machine Learning? Definition, Types, Tools & More [Електронний ресурс] / Matt Crabtree // Datacamp. – 2023. – Режим доступу до ресурсу: https://www.datacamp.com/blog/what-is-machine-learning.

7. What is machine learning (ML)? [Електронний ресурс] // IBM. – 2023. – Режим доступу до ресурсу: https://www.ibm.com/topics/machine-learning.

8. Suzuki K. Artificial Neural Networks – Methodological Advances and Biomedical Applications / Kenji Suzuki // Artificial Neural Networks – Methodological Advances and Biomedical Applications / Kenji Suzuki. – Rijeka, Croatia: InTech, 2011. – (InTech Open). – С. 3–10. – Режим доступу: https://www.researchgate.net/profile/Kenji-Suzuki-2/publication/319316102_Artificial_Neural_Networks_-

_Methodological_Advances_and_Biomedical_Applications/links/59a42f16aca272a6461bb35e/Artificial-Neural-Networks-Methodological-Advances-and-Biomedical-Applications.pdf#page=15

9. Understanding Perceptron: The Founding Element of Neural Networks [Електронний ресурс] // AnalytixLabs. – 2022. – Режим доступу до ресурсу: https://www.analytixlabs.co.in/blog/what-is-perceptron/.

10. Deep Neural Network: What is it and how is it working? [Електронний ресурс] // DataScientist. – 2023. – Режим доступу до ресурсу: https://datascientest.com/en/deep-neural-network-what-is-it-and-how-is-it-working.

11. Нельсон Д. Що таке зворотне поширення? [Електронний ресурс] / Деніел Нельсон // Unite AI. – 2024. – Режим доступу до ресурсу: https://www.unite.ai/uk/what-is-backpropagation/.

12. Бондаренко С. Функції активації: ступінчаста, лінійна, сигмоїда, ReLU та Tanh [Електронний ресурс] / Сергій Бондаренко // robot_dreams Media – Режим доступу до ресурсу: https://robotdreams.cc/uk/blog/327-funkciji-aktivaciji-stupinchasta-liniyna-sigmojida-relu-ta-tanh.

13. Ruiz P. Introducing MediaPipe Solutions for On-Device Machine Learning [Електронний ресурс] / P. Ruiz, K. Tonthat // Google Blog. – 2023. – Режим доступу до ресурсу: https://developers.googleblog.com/en/introducing-mediapipe-solutions-for-on-device-machine-learning/.

14. MediaPipe Solutions [Електронний ресурс] // Google for Developers – Режим доступу до ресурсу: https://developers.google.com/mediapipe/solutions.

15. MediaPipe Model Maker [Електронний ресурс] // Google for Developers – Режим доступу до ресурсу: https://developers.google.com/mediapipe/solutions/model_maker.

16. MediaPipe Tasks [Електронний ресурс] // Google for Developers – Режим доступу до ресурсу: https://developers.google.com/mediapipe/solutions/tasks.

17. Object detection task guide [Електронний ресурс] // Google for Developers – Режим доступу до ресурсу: https://developers.google.com/mediapipe/solutions/vision/object_detector.

18. Image classification task guide [Електронний ресурс] // Google for Developers – Режим доступу до ресурсу: https://developers.google.com/mediapipe/solutions/vision/image_classifier.

19. Image segmentation guide [Електронний ресурс] // Google for Developers – Режим доступу до ресурсу: https://developers.google.com/mediapipe/solutions/vision/image_segmenter.

20. Interactive image segmentation task guide [Електронний ресурс] // Google for Developers – Режим доступу до ресурсу: https://developers.google.com/mediapipe/solutions/vision/interactive_segmenter.

21. Gesture recognition task guide [Електронний ресурс] // Google for Developers – Режим доступу до ресурсу: https://developers.google.com/mediapipe/solutions/vision/gesture_recognizer.

22. Hand landmarks detection guide [Електронний ресурс] // Google for Developers – Режим доступу до ресурсу: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker.

23. Image embedding task guide [Електронний ресурс] // Google for Developers – Режим доступу до ресурсу: https://developers.google.com/mediapipe/solutions/vision/image_embedder.

24. Face detection guide [Електронний ресурс] // Google for Developers – Режим доступу до ресурсу: https://developers.google.com/mediapipe/solutions/vision/face_detector.

25. Face landmark detection guide [Електронний ресурс] // Google for Developers – Режим доступу до ресурсу: https://developers.google.com/mediapipe/solutions/vision/face_landmarker.

26. Pose landmark detection guide [Електронний ресурс] // Google for Developers – Режим доступу до ресурсу: https://developers.google.com/mediapipe/solutions/vision/pose_landmarker.

27. Face stylization guide [Електронний ресурс] // Google for Developers – Режим доступу до ресурсу: https://developers.google.com/mediapipe/solutions/vision/face_stylizer.

28. Sign Language MNIST classifier [Електронний ресурс] // Kaggle – Режим доступу до ресурсу: https://www.kaggle.com/code/databeru/sign-language-mnist-classifier-acc-99-78/notebook.

29. Arora A. What is Focal Loss and when should you use it? [Електронний ресурс] / Aman Arora // Github.io. – 2020. – Режим доступу до ресурсу: https://amaarora.github.io/posts/2020-06-29-FocalLoss.html.

30. Augmented Reality, AR [Електронний ресурс] // IT-Enterprise – Режим доступу до ресурсу: https://www.it.ua/knowledge-base/technology-innovation/dopolnennaja-realnost-ar.

31. Wong A. Apple announces new VR headset Vision Pro, to launch next year [Електронний ресурс] / Aloysius Wong // CBC News. – 2023. – Режим доступу до ресурсу: https://www.cbc.ca/news/business/apple-vr-headset-announcement-1.6865872.

32. Apple Inc. Загальний опис visionOS [Електронний ресурс] // Apple Inc. – Режим доступу до ресурсу: https://developer.apple.com/visionos/.

33. Apple Vision Pro Spatial Design 101 — A Beginner's Guide [Електронний ресурс] // UX Planet. – 2023. – Режим доступу до ресурсу: https://uxplanet.org/apple-vision-pro-spatial-design-101-a-beginners-guide-6dc6f404272d.

34. Документація ARKit. ARKit у visionOS [Електронний ресурс] // Apple Inc. – Режим доступу до ресурсу: https://developer.apple.com/documentation/arkit/arkit_in_visionos.

35. Heaney D. Apple Reportedly Working On Vision Pro Full Body Tracking For Fitness Apps Post-Launch [Електронний ресурс] / David Heaney // UploadVR. – 2023. – Режим доступу до ресурсу: https://www.uploadvr.com/apple-working-on-vision-pro-full-body-tracking-fitness/.

36. MediaPipe Framework on iOS [Електронний ресурс] // Google for Developers. – 2023. – Режим доступу до ресурсу: https://developers.google.com/mediapipe/framework/getting_started/ios.

37. Intro to Bazel [Електронний ресурс] // Google LLC. – 2023. – Режим доступу до ресурсу: https://bazel.build/about/intro.

38. Документація Tulsi [Електронний ресурс] // Google LLC. – 2018. – Режим доступу до ресурсу: https://tulsi.bazel.build/docs/gettingstarted.html.