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

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ

Система розпізнавання жестів рук для управління інтерфейсом вебзастосунку

Галузь знань **12 «Інформаційні технології»**

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

Освітня програма **«Комп'ютерні науки»**

Освітній рівень: бакалавр

Виконала:
студентка 4 курсу, групи КН-41
Стукан Катерина Олегівна

Керівник:
Гайна Георгій Анатолійович,
кандидат технічних наук, професор

Випускна кваліфікаційна робота бакалавра допущена до захисту рішенням кафедри *інтелектуальних технологій*
Протокол №11 від 06.06.2022р.

зав. кафедри_____ доц. Іларіонов О.Є.

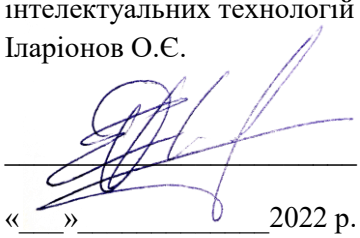
Київ – 2022

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій
Кафедра інтелектуальних технологій
Спеціальність 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ

Завідувач кафедри
інтелектуальних технологій
Іларіонов О.Є.



«___»_____2022 р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Стукан Катерині Олегівні

1. Тема проекту (роботи)
   «Система розпізнавання жестів рук для управління інтерфейсом вебзастосунку»
   затверджена протоколом засідання кафедри від «23» грудня 2021 р. №4

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

3. Вихідні дані до проекту (роботи)
   Нейромережна система розпізнавання жестів за зображенням, нейромережна система розпізнавання жестів за ключовими точками руки, програмний модуль для розпізнавання жестів у реальному часі.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
   1) Аналіз технологій та алгоритмів глибинного навчання для розпізнавання жестів
   2) Проектування системи розпізнавання жестів
   3) Опис реалізованої інформаційної технології

5. Перелік презентаційного матеріалу (з точним зазначенням обов'язкових презентацій)
   1) Вступ (тема, мета, актуальність)
   2) Огляд та обґрунтування обраних підходів та алгоритмів глибинного навчання
   3) Проектні рішення
   4) Опис реалізованої інформаційної технологій
   5) Висновки

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 15 лютого 2022 року

Керівник _____ / Гайна Г.А. /

Завдання прийняв до виконання _____ / Стукан К.О. /

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів випускної кваліфікаційної роботи	Примітка
1	Формування завдання випускної кваліфікаційної роботи	24.01.2022 – 31.01.2022	
2	Написання першого розділу	01.02.2022 – 05.03.2022	
3	Написання другого розділу	06.03.2022 – 09.04.2022	
4	Реалізація програмної технології та написання третього розділу	10.04.2022 – 10.05.2022	
5	Оформлення відповідно до вимог випускної кваліфікаційної роботи, створення презентаційного матеріалу	11.05.2022 – 28.05.2022	
6	Попередній захист	01.06.2022 – 04.05.2022	
7	Захист дипломної роботи	24.06.2022	

Студент-дипломник _____ / Стукан К.О. /

Керівник випускної кваліфікаційної роботи _____ / Гайна Г.А. /

Анотація

Стукан Катерина Олегівна виконала випускну кваліфікаційну роботу на тему «Система розпізнавання жестів рук для управління інтерфейсом вебзастосунку» за спеціальністю 122 – «Комп'ютерні науки».

У випускній кваліфікаційній роботі проведено аналіз існуючих методів розпізнавання жестів, на його основі виділено два найбільш оптимальні підходи з урахуванням особливостей функціонування вебзастосунків, визначено сценарій створення та архітектури систем розпізнавання, розроблено програмне забезпечення та проведено порівняння результатів роботи створених систем.

Ключові слова: жести рук, управління інтерфейсом, розпізнавання об'єктів, розпізнавання рухів, вебзастосунок.

Summary

Stukan Kateryna has completed the final qualifying work on the topic «Hand gesture recognition system for controlling web application interface» within the specialty 122 – «Computer Science».

In the final qualification work the existing methods of gesture recognition were analyzed, on its basis two the most optimum approaches were chosen taking into account features of web applications functioning, the scenario of creation and architectures of recognition systems were defined, software was developed and the results of the created systems were compared.

Keywords: hand gestures, interface control, object recognition, motion recognition, web application.

Зміст:

Перелік скорочень

рис. – рисунок

ВКР – випускна кваліфікаційна робота

КІ - користувацький інтерфейс

ШІ – штучний інтелект

ЗНМ – згорткова нейронна мережа

LSTM – long short-term memory, довга короткочасна пам'ять – різновид архітектури нейронних мереж

Вступ

Сьогодні люди все частіше взаємодіють з комп'ютерними інтерфейсами у різних сферах свого життя: під час роботи та навчання, для розваги та соціальної взаємодії. Розмаїття носіїв, за допомогою яких відбувається ця взаємодія, з часом тільки збільшується: модернізуються вже звичні для нас персональні комп'ютери з клавіатурами і тачпадами та смартфони з сенсорними екранами, створюються інноваційні розумні окуляри та шоломи віртуальної реальності. Вчені та винахідники з усього світу працюють над розробкою та впровадженням нових способів управління інтерфейсами, які допоможуть скоротити розрив між людиною та технологіями, будуть більш природніми та інтуїтивними. Серед них можна виділити наступні: управління жестами, голосом, управління шляхом відстеження руху очей або навіть імпульсів мозку. Усі ці підходи стали перспективними у тому числі завдяки підвищенню обчислювальної потужності апаратного забезпечення та можливості застосування алгоритмів штучного інтелекту, які її потребують.

В даній роботі буде досліджено можливість управління інтерфейсами саме жестами долоні. Крім того, що управління жестами максимально імітує взаємодію людини з предметами навколишнього середовища, така технологія також є надзвичайно актуальною під час пандемії коронавірусу в Україні та світі, коли користування сенсорними екранами в громадських місцях може бути небезпечним. Також безконтактне управління зможе полегшити робочі процеси людям багатьох професій: медикам – забезпечивши стерильність; водіям транспортних засобів – мінімізуючи їх відволікання під час руху; працівникам виробництв, на яких задіяні небезпечні речовини.

В інших випадках натискання невеликих кнопок або випадкове натискання неправильних полів інтерфейсу збільшують розчарування від використання програмних продуктів та змушують людей шукати кращий користувацький досвід у інших виробників. Розпізнавання жестів рук в режимі реального часу – це лише наступний крок у технологічній еволюції, і він ідеально підходить для сучасного споживача. Окрім використання жестів для

управління інтерфейсами, відстеження рук можна застосовувати в середовищах доповненої та віртуальної реальності, розпізнавання мови жестів, навчання, ігор та інших варіантів використання.

Сучасна людина, працюючи з комп’ютерами, проводить багато часу в веб-середовищі, відповідно більшість програмних застосунків, з якими вона взаємодіє, працюють у браузері. З погляду на це, було вирішено реалізувати систему розпізнавання жестів саме для управління вебзастосунками, враховуючи особливості та обмеження середовища у якому вони працюють.

Отже, метою даної роботи є створення системи розпізнавання жестів рук, яка зможе забезпечити зручну та інтуїтивну взаємодію з вебзастосунками та дозволить позбутися пристроїв-посередників цього процесу, наприклад, таких як мишка або сенсорний екран.

Розділ 1. Аналітичний огляд

1.1 Опис предметної області

1.1.1 Загальні відомості про інтерфейси

Загалом, поняття інтерфейс можна визначити як «точку (або місце), де зустрічаються і взаємодіють дві системи»[1].

За визначенням Оксфордського словника термінів комп'ютерних наук [2], користувацький інтерфейс (КІ) або людино-машинний інтерфейс – це засіб комунікації між користувачем-людиною та комп'ютерною системою, зокрема щодо використання пристроїв введення/виведення з допоміжним програмним забезпеченням. Іншими словами, це структура, що складається з усіх елементів, які дозволяють користувачу виконувати команди та запити до програми, а програмі – відповідно оперувати даними, які вона містить, пропонувати користувачу виконувати певні дії та надавати зворотній зв'язок щодо того, як виконуються команди та запити, який їх результат. В залежності від типу КІ можуть використовуватися різні елементи, щоб досягти вищезазначеного.

З самого початку існування комп'ютерів КІ значно еволюціонували. Щоб змусити працювати перші обчислювальні машини у 30-х роках минулого століття доводилось використовувати в якості вхідних даних перфокарти, папірці з дірочками, які поміщали в спеціальні картриджі для виконання якоїсь логічної операції. Результат логічної операції, вихід, зазвичай надходив у друк. Вся взаємодія могла тривати від годин до днів. Але збільшення можливостей обчислювальної техніки дозволило перейти спочатку до управління за допомогою командного рядка у 60-х, а потім і до звичних нам графічних інтерфейсів у 80-х. З тих пір пройшло немало часу і графічні інтерфейси, які ми бачимо зараз суттєво відрізняються від своїх попередників. Очевидними є відмінності у якості зображення та більш продуманому дизайні, але що найважливіше – це поява нових засобів вводу-виводу, що забезпечують більшу ергономічність.

1.1.2 Жести та жестові інтерфейси

Жестові інтерфейси – підмножина графічних КІ для комп’ютерів, оснащених або спеціальними пристроями введення (відмінними від клавіатури), або сенсорними екранами, які дозволяють емулювати клавіатурні команди за допомогою жестів. Основною мотивацією розробки таких інтерфейсів є поліпшення ергономічності управління, з відмовою від звичного для комп'ютерних програм меню програми.

За наведеним визначенням під жестом розуміють певний рух або фіксоване положення частини або усього тіла людини який може виконуватися у просторі, на екрані керованого пристрою, на поверхні спеціального пристрою (тачпад ноутбука) або на будь якій іншій поверхні.

В рамках цієї роботи, інтерес представляє саме можливість управління просторовими жестами рук. Просторові жести рук можна розділити на дві категорії: статичні та динамічні. В таблиці 1.1 наведені основні характеристики, які мають жести цих категорій:

Таблиця 1.1 – Характеристики статичних та динамічних жестів

Статичні жести	Динамічні жести
Положення руки у просторі відносно камери, інших об’єктів Кути згинання пальців Форма	Послідовність зміни: – Положення руки у просторі відносно камери, інших об’єктів – Кутів згинання пальців – Форми Траєкторія руху Швидкість руху Час відтворення жесту

В статичних жестах положення руки не змінюється під час відтворення жесту. Інтерпретація статичного жесту, тобто отримання значення яке він передає, залежить лише від форми жесту, яка визначається позицією руки у

просторі та кутами згинання пальців. Приклади таких жестів наведені на рис.1.1.

Рис. 1.1 – Статичні жести рук для позначення
літер американської жестової мови [3]

В динамічних жестах, навпаки, положення руки змінюється під час відтворення жесту, тобто можна вважати, що вони складаються з певної послідовності статичних жестів. Для інтерпретації таких жестів важливо знати час та швидкість їх відтворення, положення у просторі по відношенню до сенсорів пристрою введення, та правильну послідовність зміни форми та положення жесту. Приклади таких жестів наведені на рис.1.2.

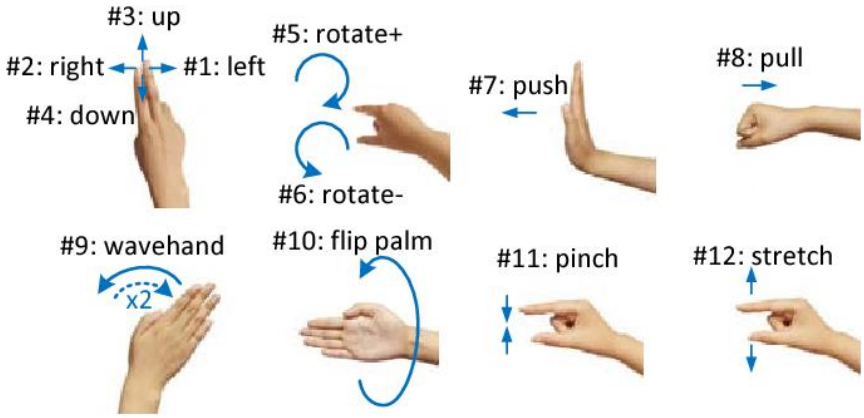


Рис. 1.2 – Зображення динамічних жестів рук [4]

Для управлінням КІ можуть бути корисними як статичні, так і динамічні жести, в залежності від дії, яку потрібно виконати. Тому краще за все

проектувати систему розпізнавання таким чином, щоб вона була здатна розпізнати обидві категорії.

1.1.4 Актуальність жестових інтерфейсів

Темпи, з якими створюється нове програмне та апаратне забезпечення, невпинно зростають, компаній-виробників стає все більше, відповідно збільшується й конкуренція між ними. Щоб завоювати прихильність користувачів, звернути їх увагу саме на свій продукт, потрібно робити його максимально зручним, надійним та безпечним. Одним з варіантів вдосконалення продукту, в залежності від сфери його застосування, може бути додавання можливості управління жестами або повний перехід на жестовий інтерфейс. Таке рішення матиме наступні позитивні наслідки:

- Підвищення якості користувацького досвіду за рахунок поліпшення ергономічності управління;
- Наближення взаємодії людини з графічними елементами КІ до взаємодії з справжніми фізичними об’єктами;
- Зменшення кількості пристроїв-посередників для введення інформації;
- Усунення необхідності прямого фізичного контакту з комп’ютером, відповідно – забезпечення стерильності при взаємодії з КІ призначеними для масового використання або для таких галузей як медицина, лабораторні дослідження, харчування.

Можна знайти безліч сфер та варіантів застосування жестових інтерфейсів. Далі наведемо декілька прикладів:

- Банківські термінали
- Каси самообслуговування в закладах громадського харчування та супермаркетах
- Мобільні додатки що використовуються водіями таксі для управління замовленнями
- Інтерактивні веб-сайти для навчання

– Комп’ютерні ігри
– Інтерактивні рекламні вітрини

Про актуальність та стрімкий розвиток даної технології також свідчить ринкова аналітика. Наприклад, у звіті американської консалтингової компанії Grand View Research[5] вказано, що зараз ключовими гравцями на ринку технологій розпізнавання жестів є такі світові гіганти, як Apple, Intel, Microsoft і Google, які активно проводять дослідження та наймають фахівців за цим напрямком. Зазначено, що Азіатсько-Тихоокеанський регіон, до якого входить багато розвинутих країн, домінував на ринку з точки зору доходів у 2017 році, і ймовірно, що регіон залишиться домінуючим щонайменш до 2025 року. На рис.1.3 наведена діаграма прогнозованого росту ринок розпізнавання жестів у Китаї, який входить у вищезазначений регіон.

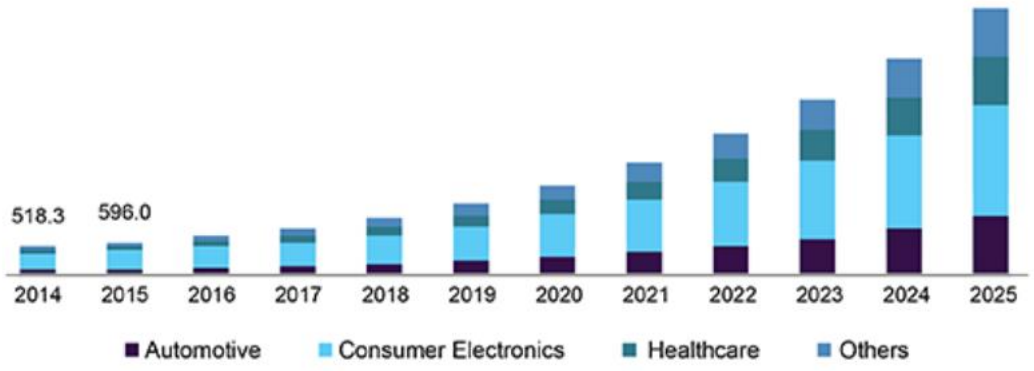


Рис.1.3 – Ринок розпізнавання жестів у Китаї, 2014–2025 рр. [5]

1.2 Огляд існуючих рішень

Ідея подібних систем не є принципово новою. Але її реалізації відрізняються у різних виробників, як за технологією розпізнавання та відстеження жестів, так і за способами використання. Розглянемо існуючі рішення.

Leap Motion Controller від Ultraleap – невеликий пристрій, що приєднується до комп’ютера через USB-порт й працює на платформах Windows, MAC OS та Linux. Пристрій має чутливий простір близько чверті

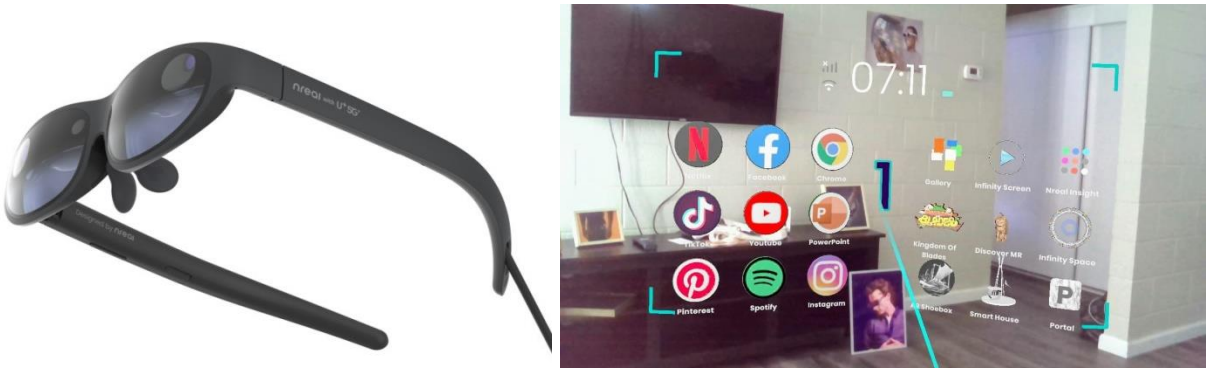
кубічного метра у вигляді перевернутої піраміди з максимальною робочою відстанню 80 см (рис.1.4а). Всередині знаходяться дві інфрачервоні камери та три потужні інфрачервоні світлодіоди. Світлодіоди підсвічують руки, а камери роблять їх захоплення передаючи зображення програмному обробнику, який використовуючи математичні алгоритми, виділяє контури рук і відстежує координати пальців. Leap Motion має два різних режими відстеження – стандартний, на столі (рис.1.4б), або як вбудована частина VR-пристроїв (рис.1.4в), таких як Oculus Rift. Ultraleap також має бібліотеки, що дозволяють створювати безконтактні жестові інтерфейси використовуючи пози рук та їх послідовності, які система може розпізнати. Але такі бібліотеки доступні лише для ігрових рушіїв Unity та Unreal Engine [6].

а) б) в)

Рис. 1.4 – Leap Motion: а) простір чутливості, б) використання з комп'ютером в) використання з VR-шоломом

Nreal Light – AR-окуляри (рис.1.5а), що дозволяють відображати як 2D-додатки Android, так і інтерактивний 3D-вміст у середовищі змішаної реальності (рис.1.5б). Управління додатками може здійснюватись за допомогою підключеного до окулярів смартфону, або ж жестами рук. Відстеження рук працює з використанням технології ClayAIR. За допомогою двох вбудованих просторових камер Nreal Light зчитується 3D-модель руки та накладається у форматі Full HD поверх фізичної руки. Зміни положень пальців і рук у реальному часі відстежуються та відображаються за допомогою найсучасніших алгоритмів глибокого навчання. Оскільки технологія відстеження рук користується великим попитом, компанією була також створена платформа

Nreal SDK для самостійної розробки AR-додатків . Налаштування, доступні для розробників, включають моделювання або візуалізацію рук, окремих пальців, обмежуючої рамки, курсору тощо. Бібліотека жестів, надана Clay AIR, включає щипки, вказування, захоплення, свайп, масштабування, а також дозволяє налаштовувати власні жести.



а) б)

Рис. 1.5 – а) Nreal Light окуляри,
б) середовищі змішаної реальності Nreal

RealSence – це ціла лінійка пристроїв компанії Intel, кожен з яких являє собою апаратно-програмний комплекс, до якого входить:

- Колірна камера – звичайна веб-камера з підтримкою FullHD відео, розширенням до 1920x1080 пікселів.
- Інфрачервоний датчик глибини – випромінює інфрачервоні проміні і заміряє, через який час вони повернулися. Таким чином прораховується відстань від камери до поверхонь об'єкта, який спостерігається.
- Графічний процесор – спеціалізований мікрочіп для первинної обробки потоку даних для таких задач як розрахунок глибини, фільтри перешкод тощо. Вбудований графічний процесор суттєво розвантажує центральний процесор комп'ютера та прискорює роботу камери загалом.

Пристрої підтримуються міжплатформним пакетом RealSenceSDK з відкритим кодом, що спрощує підтримку камер для сторонніх розробників програмного забезпечення. В пакеті надається як готовий програмний модуль відстеження рук, так і доступ до «сирих» даних, отриманих безпосередньо з датчиків (рис.1.6). Пакет не має рішення, що дозволяло б легко впроваджувати

технологію на клієнтській стороні вебзастосунків, а система розпізнавання жестів не є гнучкою та обмежена 11 базовими жестами.

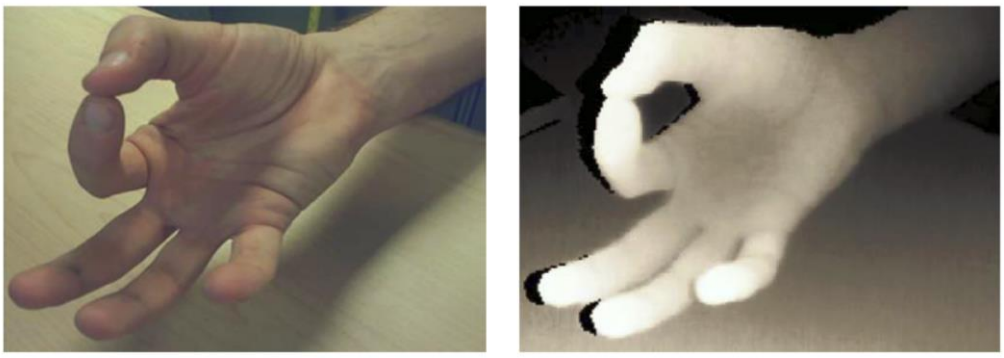


Рис. 1.6 – Кольорове зображення та зображення глибини, отримані камерами RealSence

В усіх приладах описаних вище використовуються просторові камери з інфрачервоними датчиками або комбінації таких камер з звичайними RGB-камерами. Проте існують й інші підходи до розпізнавання жестів, наприклад, з використанням провідних рукавичок. Для реєстрації за допомогою рукавичок фізичних даних таких як згинання пальців, використовуються різні сенсорні технології. Часто – це пристрій стеження за рухом, такий як магнітний або інерційний пристрій відстеження, що приєднується для збирання даних загального положення та обертання рук. Потім ці рухи інтерпретуються програмним забезпеченням, що додається до рукавички. Дорогі висококласні рукавички, також можуть забезпечувати тактильний зворотний зв'язок. Це дозволяє використовувати їх також як пристрій виводу. Перша модель рукавичок для широкого використання з'явилася майже тридцять років тому. Power Glove від Mattel Inc (рис.1.7) використовувався як ігровий пристрій керування для Nintendo. Він критикувався за низьку точність і складність у використанні. Зараз, провідні рукавички доступні лише за великою ціною та є дуже громіздкими для повсякденного використання. Проте їх часто використовуються в середовищах віртуальної реальності та у кіноіндустрії (рис.1.8).

Рис. 1.7 – Power Glove від Mattel Inc

Рис. 1.8 – Рухи Тома Хенкса, записані оптичною системою захоплення руху, для мультфільму «Полярний експрес» [7]

Інший підхід до розпізнавання жестів обрали розробники браслету Myo (рис.1.9). В залежності від того, як скорочуються м’язи, браслет розуміє, який рух ви виконуєте – стискаєте кулак або випрямляєте всі пальці, згинаєте долоню на себе або від себе, крутите рукою і т.д. Цей інноваційний пристрій під’єднується до комп’ютера, телефону або планшету через Bluetooth і використовує м’язові датчики, а також 6-осьовий датчик для відстеження вашого руху. Датчики м’язів можуть виявляти зміни положення навіть окремого пальця, і оскільки імпульси у м’язах з’являються раніше, ніж вони насправді виконують рух, функції які асоційовані з певними жестами виконуються моментально, без жодних затримок [8]. Спільнотою Myo було розроблено велику кількість програмних модулів для роботи браслета з різними платформами, у тому числі модулі для розпізнавання жестів та бібліотека для роботи у браузері MyoJS [9]. Але активність спільноти, як і підтримка бібліотек, суттєво знизилась в останні роки через рішення виробника припинити подальший випуск браслетів.

Рис. 1.9 – браслет Myo

В таблиці 1.2 наведено порівняння розглянутих технологій за властивостями, які можуть суттєво вплинути на рішення впровадження одної з них для жестового управління власним вебзастосунком.

Таблиця 1.2 – Порівняння існуючих рішень для розпізнавання жестів

	Додаткове обладнання	Можливість достатньо легко додавати нові жести	Наявн. бібліотек для роботи на клієнтській стороні вебзаст.	Доступність
Leap Motion	+	+	–	+
Nreal Light	+	+	+	+
RealSence	+	–	–	+
Провідні рукавички	+	–	–	–
Myo	+	–	+	–

1.3 Проблемні моменти

Головною проблемою розглянутих вище рішень для управління жестовими інтерфейсами є необхідність використання додаткових пристроїв. Висока вартість та обмежена доступність, в сенсі складності знаходження у магазинах та придбання, робить ці рішення непривабливими для пересічного користувача.

Проте усі сучасні комп’ютери мають вбудовану веб-камеру або ж до них легко підключається провідна веб-камера, яку можна придбати у будь-якому магазині техніки за прийнятну ціну. В той же час розпізнавання жестів за звичайними зображеннями є своєрідним викликом для розробників програмного забезпечення, адже в такому випадку:

- ми маємо набагато менше вхідних даних, на відміну від зображень глибини що несуть тривимірну інформацію;
- розпізнавання ускладнюється за поганого або, навпаки, за дуже сильного освітлення.

Тому щоб створити надійну систему, рішення для розпізнавання жестів вимагатиме, серед іншого, вирішення наступних задач:

- впровадження передових алгоритмів глибинного навчання;
- створення якісного набору даних для тренування нейронної мережі.

Іншою проблемою є відсутність в пакетах для розробників інструментів для роботи саме з вебзастосунками, тому при виборі технологій для реалізації системи розпізнавання потрібно це враховувати.

1.4 Огляд алгоритмів глибинного навчання для розпізнавання жестів

1.4.1 Згорткові нейронні мережі

Розпізнавання жестів на зображенні відноситься до задач комп’ютерного зору та є більш конкретним випадком задачі розпізнавання об’єктів. Згорткові нейронні мережі (ЗНМ) – це різновид моделей глибинного навчання, що майже повсюдно використовуються для вирішення цієї задачі [10]. Базова архітектура ЗНН повинна включати наступні три типи шарів (наприклад, як на рис.1.10): шар згортки, шар пулінгу та повнозв’язний шар. Розглянемо їх детальніше, щоб зрозуміти принцип роботи ЗНМ.

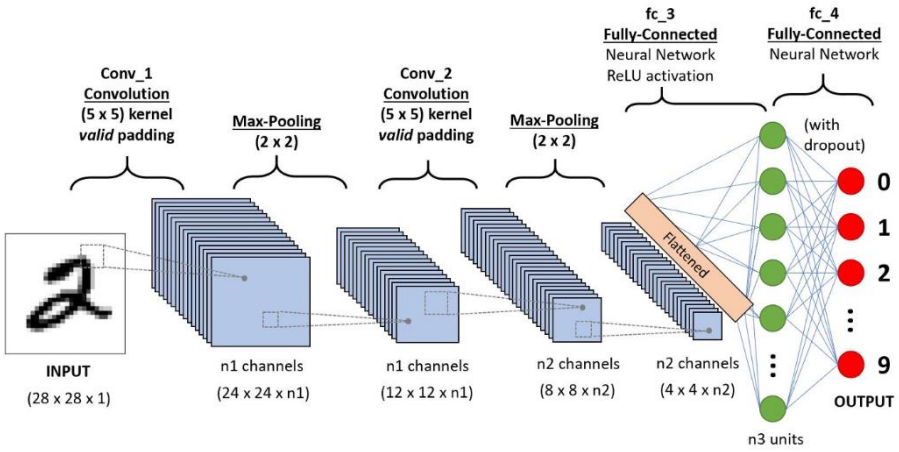


Рис.1.10 – Архітектура ЗНМ для класифікації рукописних цифр

Згорткові шари працюють на основі фільтрів (ядер), які займаються розпізнаванням локальних шаблонів. Перший шар згортки розпізнаватиме маленькі шаблони, такі як лінії різного нахилу, другий – більш великі шаблони, які складатимуться з ознак, що повертатимуться першим, і т.д. (рис.1.11) Фільтр – це звичайна матриця чисел-вагів, які коригуються під час "навчання" нейронної мережі. Він переміщається (крок переміщення за замовчуванням дорівнює одиниці) вздовж зображення і визначає, чи є певна шукана характеристика в конкретній його частині. Для отримання відповіді такого роду відбувається операція згортки, яка є сумою добутків елементів фільтра та матриці вхідних сигналів. Вихідна матриця даної операції називається картою ознак.

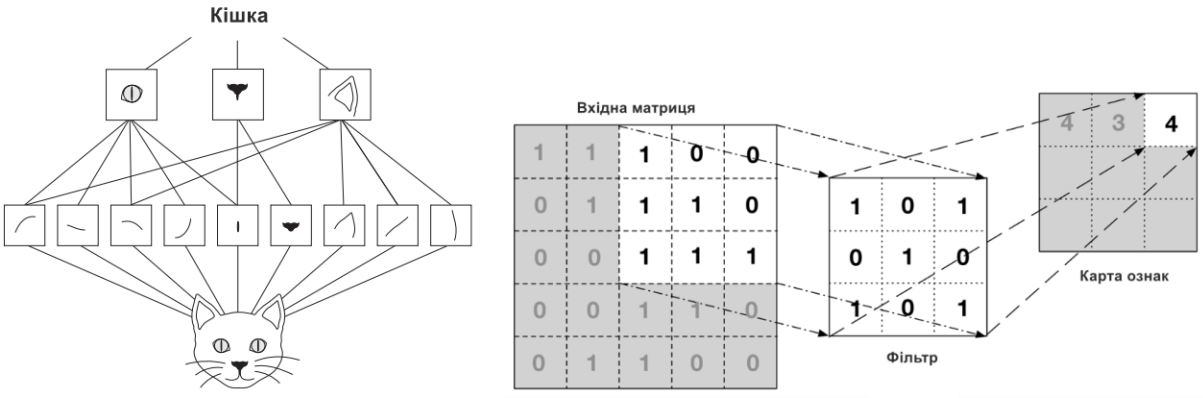


Рис.1.11 – Розпізнавання об’єктів ЗНМ та операція згортки

Після загорткових шарів розміщують шари пулінгу для проведення редукування проміжного результату, тобто зниження кількості даних із збереженням загального сенсу. Це робиться з метою прискорення процесу навчання та зменшення споживання обчислювальних ресурсів. На цьому етапі застосовується операція знаходження максимуму або середнього. Вздовж даних переміщується так зване вікно просіювання (зазвичай розміром 2×2) й з елементів, які потрапляють у його поле зору, відбирається максимальний (або розраховується середнє) і переміщається в результуючу матрицю (рис.1.12).

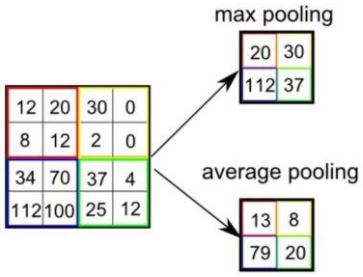


Рис. 1.12 – Операція пулінгу

Останній шар або кілька шарів ЗНМ – це повнозв’язні шари, тобто такі, з яких складається звичайний перцептрон. Їх завдання полягає в тому, щоб повідомити нам ймовірності належності об’єкта до того чи іншого класу.

1.4.2 Техніки виявлення об'єктів на зображенні

На основі ЗНМ будуються детектори для виявлення об'єктів на зображенні у реальному часі. Розглянемо кілька поширених архітектур:

1. YOLO (You Only Look Once) – одноступеневий детектор, тобто використовує одну нейронну мережу для виявлення об’єктів у реальному часі, що робить його одним з найшвидших. Вхідне зображення розділяється на квадратні області, кожна з яких відповідає за передбачення кількох обмежувальних прямокутників, для яких оцінюється достовірність присутності в них об'єктів і ймовірності приналежності до певного класу. [11]
2. SSD (Single-shot detector) – детектор подібний до попереднього, так само оцінює наявність кожної категорії об’єктів у кожному прямокутнику, але в цьому випадку можливе різне співвідношення сторін тобто мережа

налаштовує поле, щоб краще відповідати формі об’єкта. Крім того, детектор поєднує прогнози з кількох карт об’єктів з різною роздільною здатністю для обробки об’єктів різного розміру. Мережу легко навчити та інтегрувати в програмні системи, у порівнянні з іншими одноступеневими методами, SSD має набагато кращу точність, навіть з меншими розмірами вхідного зображення.[12]

3. R-CNN (Region-based Convolutional Neural Networks) – спочатку детектор намагається знайти області, які можуть містити об’єкт, об’єднуючи подібні пікселі та текстури в прямокутні рамки з використанням алгоритму жадібного пошуку. Потім згортковими шарами витягуються ознаки з знайдених областей, а SVM-класифікатор визначає, що за об’єкт на них зображено. Такий підхід простий для розуміння, проте виконує розпізнавання досить повільно порівняно з іншими, а також є дорогим з точки зору простору та часу навчання. [11]

1.4.3 LSTM-мережі для розпізнавання рухів

Раніше вже було зазначено, що розпізнавання динамічних жестів є важливим для реалізації певних взаємодій з жестовим інтерфейсом. Але для цього вже не достатньо обробки кадрів, що надходять з відеокамери, по окремості, адже щоб розпізнати та класифікувати рух потрібно зберігати інформацію про певну послідовність попередніх кадрів.

Тут у нагоді стають LSTM (англ. long short-term memory) – вид рекурентних (тобто здатних пов'язувати попередню інформацію з поточним завданням) нейронних мереж, які використовуються для класифікації чи передбачення, в умовах коли між важливими подіями існують часові затримки невідомої тривалості. Вони добре підходять для розпізнавання динамічних жестів, адже навіть одна людина може відтворювати один і той самий жест з різною швидкістю.

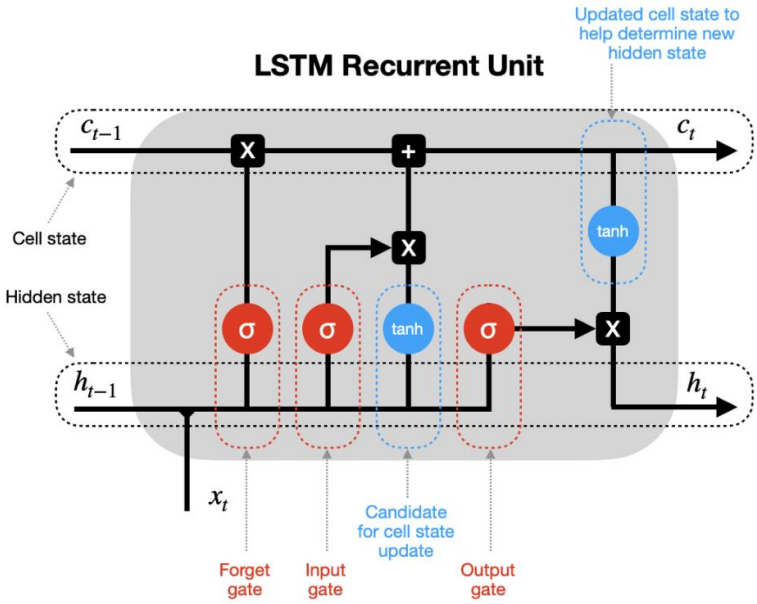


Рис. 1.13 – Нейрон LSTM мережі [13]

Принцип роботи LSTM мереж на прикладі одного нейрона (рис.1.13):

Прихований стан попереднього часового кроку (h_t-1) і вхід на поточному часовому етапі (x_t) об'єднуються перед проходженням його копій через різні вузли.

Вузол забування контролює, яку інформацію слід забути. Оскільки сигмоїдна функція знаходиться в діапазоні від 0 до 1, вона встановлює, які значення в стані комірки слід відкинути (помножити на 0), запам'ятати чи частково запам'ятати (помножити на деяке значення від 0 до 1).

Вхідний вузол допомагає визначити важливі елементи, які необхідно додати до стану комірки. Результати вхідного вузла помножуються на кандидата оновлення стану комірки, при цьому до стану комірки додається лише інформація, яку вхідний вузол вважає важливою.

Оновити стан комірки – попередній стан комірки (c_t-1) помножується на результати вузла забування, потім додається нова інформація, щоб отримати наступний стан комірки (c_t).

Оновити прихований стан – останній стан комірки (c_t) передається через функцію активації tanh і помножується на результати вихідного вузла.

Останній стан комірки (c_t) і прихований стан (h_t) рекурентно надходять до комірки і процес повторюється на часовому етапі t+1. Цикл продовжується, поки ми не дійдемо до кінця послідовності.

1.5 Відстеження рук та розпізнавання жестів

Коли ми говоримо про технології відстеження рук та технології розпізнавання жестів, їх можна легко сплутати, тому варто пояснити, що вони собою являють і чим вони відрізняються.

Обидві технології дозволяють користувачам використовувати свої руки для взаємодії з комп'ютерами без дотиків або контролерів. Задачі відстеження рук та розпізнавання жестів належать до простору задач комп'ютерного зору – області штучного інтелекту (ШІ), яка дає змогу комп'ютерам і системам отримувати значущу інформацію з цифрових зображень, відео та інших візуальних даних і виконувати дії або давати рекомендації на основі цієї інформації. Якщо ШІ дозволяє комп'ютерам думати, то комп'ютерний зір дозволяє їм бачити, спостерігати і розуміти, що вони бачать. [14]

Хоча і є спорідненими, системи розпізнавання жестів та системи відстеження рук мають одну фундаментальну відмінність – у більшості випадків система розпізнавання жестів розпізнає лише певні конкретні жести, наприклад, великий палець піднятий вгору означаючий «окей» або відкриту долоню означаючу «стоп». Кількість жестів зазвичай обмежена, в іншому випадку користувачам було б важко їх запам'ятати, але для цієї обмеженої кількості система розпізнавання зазвичай працює досить надійно.

Системи відстеження рук мають більшу кількість взаємодій, які вони можуть підтримувати. Такі системи зазвичай відстежують або об'єм руки, або окремі положення суглобів і пальців. Це дає теоретично необмежену кількість взаємодій з цифровими об'єктами, так само, як ми використовуємо наші руки для взаємодії з об'єктами в реальному світі. Але іноді можна зіткнутися з проблемою оклюзії (рис.1.14) – перекриття частин руки, внаслідок якого

ускладнюється розпізнавання. Оскільки системи розпізнавання жестів навчаються на кількох конкретних жестах, у них немає тих самих проблем, але вони натомість не мають такої ж гнучкості.

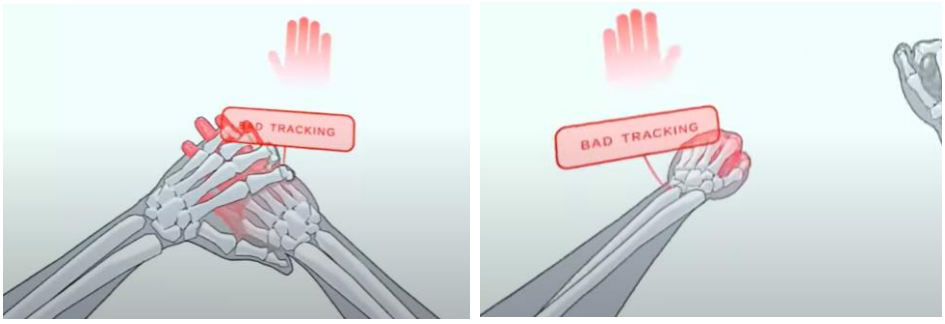


Рис. 1.14 – Приклади оклюзії: одна рука лежить на іншій, стиснутий кулак

У блозі Intel RealScence [15] наводиться аналогія, яка може допомогти зрозуміти різницю між відстеженням рук і розпізнаванням жестів: подумайте про різницю між телефоном з клавіатурою та сучасним смартфоном з сенсорним екраном. Клавіатура має певні кнопки, які дуже добре виконують певний набір взаємодій, але сенсорний екран може виконувати необмежену їх кількість, хоча іноді ви будете випадково натискати неправильну клавішу або літеру через більш тонкий характер взаємодій.

Жодна система за своєю суттю не є кращою чи гіршою за іншу, важливо вибрати оптимальну для конкретного випадку використання та потреб користувачів.

В даній роботі планується порівняти два підходи до створення систем жестового управління інтерфейсами, один з яких використовує нейронну мережу для розпізнавати жестів безпосередньо з зображення, інший же є двоступінчастим, тобто виконує розпізнавання на основі даних, отриманих системою відстеження рук.

1.6 Аналіз зацікавлених сторін

Зацікавлені сторони (або стейкхолдери) – це групи людей або окремі люди, яке прямо або опосередковано впливають на проект або мають певні очікування від нього.

1. Внутрішні зацікавлені сторони (офіційно залучені до функціонування системи):
   - ML-інженер, який безпосередньо створює систему розпізнавання жестів
   - Розробники вебзастосунків, які впроваджують систему до свого продукту
   - UX-дизайнери (дизайнери користувацького досвіду), які розробляють сценарії взаємодії з вебзастосунком та формують набір жестів
   - Замовники застосунків з жестовими інтерфейсами
2. Зовнішні зацікавлені сторони (не приймають безпосередньої участі, але можуть впливати на функціонування системи):
   - Технологічні компанії, чиє програмне забезпечення (бібліотеки, фреймворки тощо) використовується для реалізації даної системи
   - Конкурентні організації

1.7 Постановка задачі

Метою даної ВКР є реалізація системи розпізнавання жестів, яку можна буде підключити до власного вебзастосунку для забезпечення зручного та інтуїтивного управління його інтерфейсом.

Об’єкт дослідження: управління інтерфейсом вебзастосунку жестами рук.

Предмет дослідження: розпізнавання жестів рук з використанням нейромережних технологій.

Сформуємо також вимоги до розроблюваної системи.

Функціональні вимоги:

– Автоматизоване формування набору даних та його доповнення
– Перетворення кадрів відеопотоку до прийнятного системою відстеження формату
– Перетворення даних відстеженої руки до прийнятного класифікатором формату
– Коректне розпізнавання жестів
– Візуалізація розпізнавання

Нефункціональні вимоги:

– Швидке розпізнавання
– Інтуїтивність жестів
– «Легкість» моделі, враховуючи обмеження браузерів

Під час даної роботи буде реалізовано та порівняно два підходи до розпізнавання жестів, що дозволить визначити, який з них є кращим для розглянутого застосування.

Для досягнення поставленої мети було сформульовано п’ять головних задач та розбито їх на більш детальні кроки. Дерево задач наведене на рис.1.15.

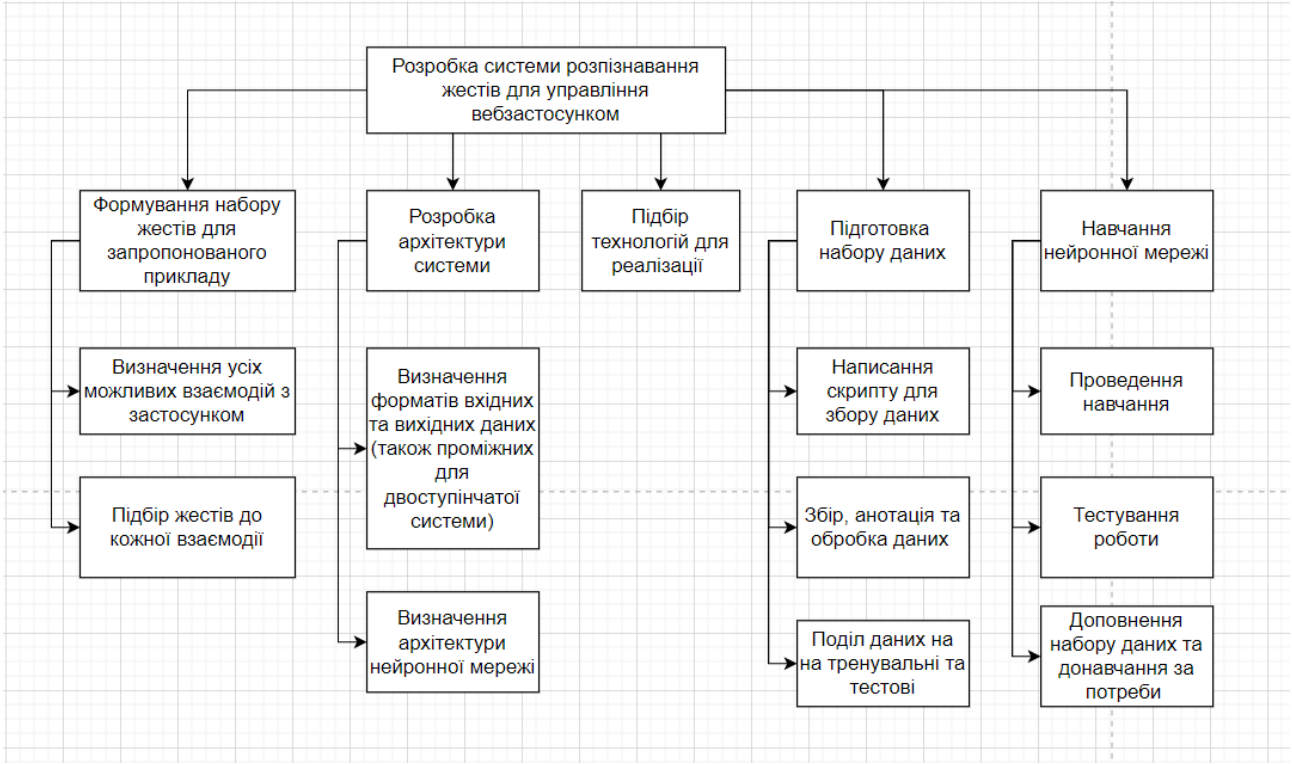


Рис. 1.15 – Дерево задач ВКР

Висновок до першого розділу

Під час роботи над першим розділом було детально проаналізовано обрану предметну область, були наведені варіанти застосування жестових інтерфейсів й аргументовано корисність та актуальність їх розробки, досліджено існуючі рішення управління інтерфейсами жестами, визначено їх переваги та недоліки. Також були описані алгоритми глибинного навчання, які застосовуються для задач розпізнавання об’єктів та рухів в режимі реального часу.

На основі проведеного аналізу було визначено основні зацікавлені сторони проекту, сформовано вимоги до системи та задачі, які потрібно виконати для досягнення мети ВКР.

Розділ 2. Проектування системи

2.1 Формування набору жестів управління

Майже кожен сучасний вебзастосунок містить в собі такі блоки, що являють собою списки однотипних сутностей (рис.2.1), наведемо приклади:

- Стрічка постів у соціальній мережі
- Стрічка новин на новинному порталі
- Список страв на сайті доставки їжі
- Плейлист з музикою в музичному веб-додатку

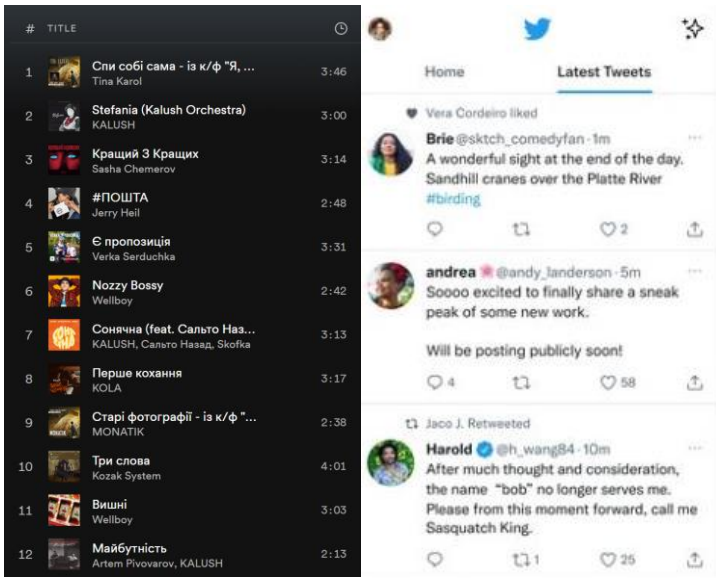


Рис. 2.1 – Приклади Spotify та Twitter

В даній роботі система буде реалізована для управління саме такими блоками інтерфейсі, на прикладі стрічки в соцмережі.

Дії які зможе виконати користувач:

- Перемикання між постами (перехід до наступного або попереднього)
- Відкриття та закриття повного тексту посту
- Можливість залишити реакцію на пост (поставити лайк), та відмінити її

Для виконання цих дій визначимо жести:

1. Піднятий вгору великий палець – поставити лайк
2. Опущений вниз великий палець – зняти лайк

3. Розімкнення долоні – відкрити повний текст (динамічний жест)
4. Стиснення руки в кулак – закриття повного тексту посту (динамічний жест)
5. Вказівний палець вверх – перейти до попереднього посту
6. Вказівний палець вниз – перейти до наступного посту

На рис.2.2 наведено ілюстрацію даних жестів.

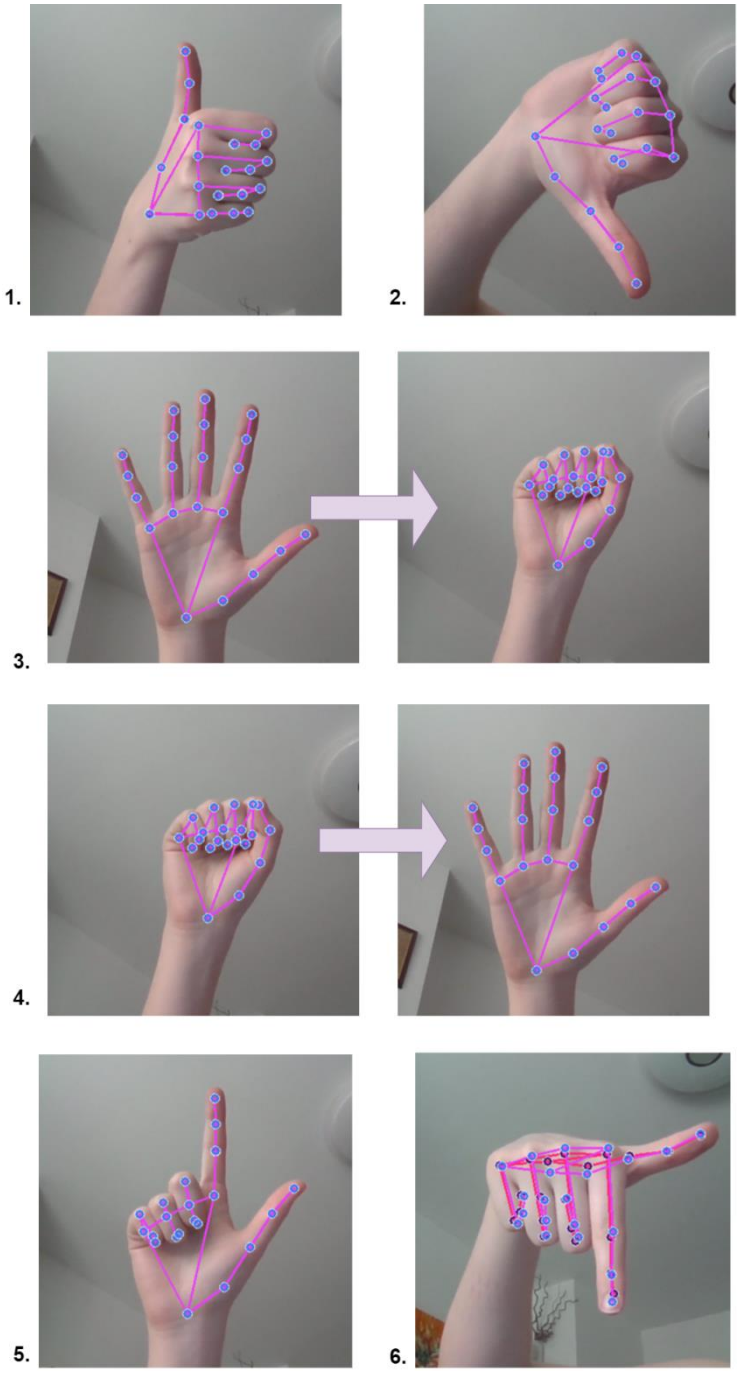


Рис. 2.2 – Набір жестів для системи розпізнавання

2.2 Моделювання процесів

Використаємо нотацію IDEF0 для графічного опису процесу розпізнавання жестів (рис.2.3). Вихідними функціональними блоками будь-якої моделі IDEF0 процесу є діяльності (activity) , які деталізуються (декомпозуються) до необхідного рівня, і стрілки (arrows). Опис стрілок системи:

- Входи (Input) – дані, що надходять в процес або використовуються процесом: RGB-зображення, отримані з вебкамери.
- Виходи (Output) – дані, що є результатом процесу: ймовірності приналежності жеста до одного з класів (якщо він наявний на зображені) та графічне зображення результатів розпізнавання у вигляді обмежувальних квадратів, виділення ключових точок руки, підпису жеста тощо.
- Управління (Control) – обмеження на виконання операцій процесу: архітектура системи, адже вона може опрацьовувати лише ті дані, для яких вона була спроектована, та набір жестів, адже аналіз буде проводитися за результатами навчання моделі на певних жестах.
- Механізми (Mechanism) – все, що застосовується для виконання процесу: сама інтелектуальна система розпізнавання та засоби графічного представлення.

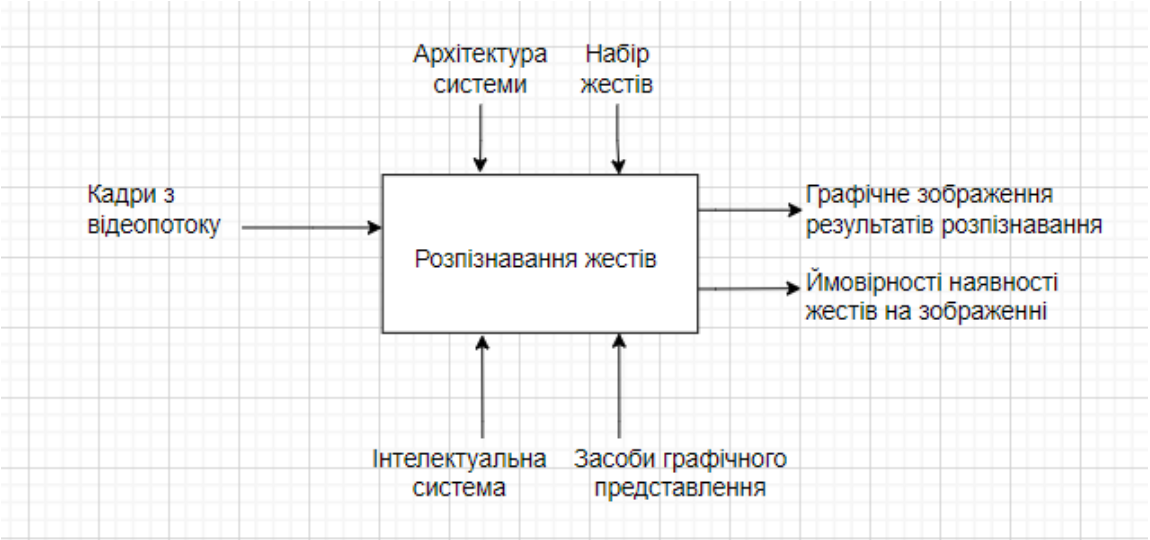


Рис.2.3 – Контекстна діаграма IDEF0

Далі буде проведено декомпозицію отриманої схеми, визначивши основні підпроцеси:

– Попередня обробка зображень та приведення до потрібного формату
– Аналіз даних
– Візуалізація результатів

Декомпозицію контекстної діаграми IDEF0 першого рівня наведено на рис.2.4.

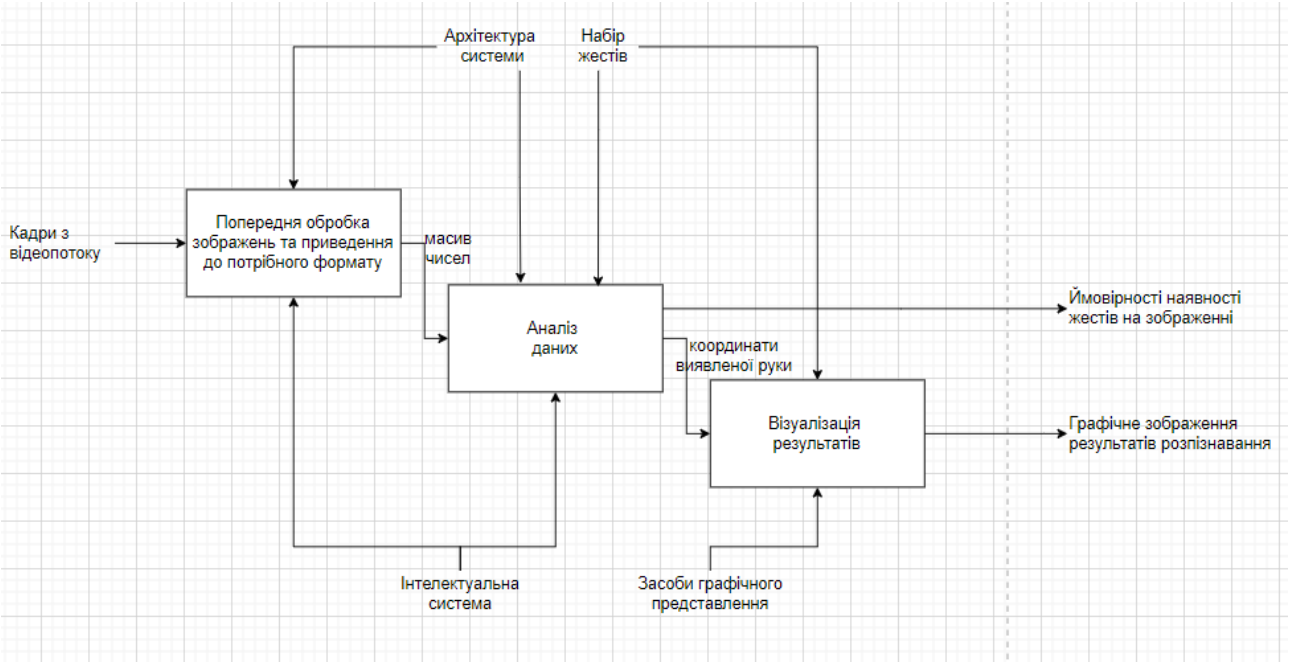


Рис.2.4 – Декомпозиція IDEF0 першого рівня

Блок «Аналіз даних» буде відрізнятися для двох підходів які розглядаються у даній роботі. Якщо для одноступінчатого підходу він буде складатися лише з отримання передбачення нейронної мережі, то для двоступінчатого підходу цей процес можна декомпозувати на наступні під процеси (рис. 2.5):

– Відстеження руки, внаслідок якого отримуємо координати ключових її точок
– Нормалізація отриманих даних
– Передбачення нейронної мережі

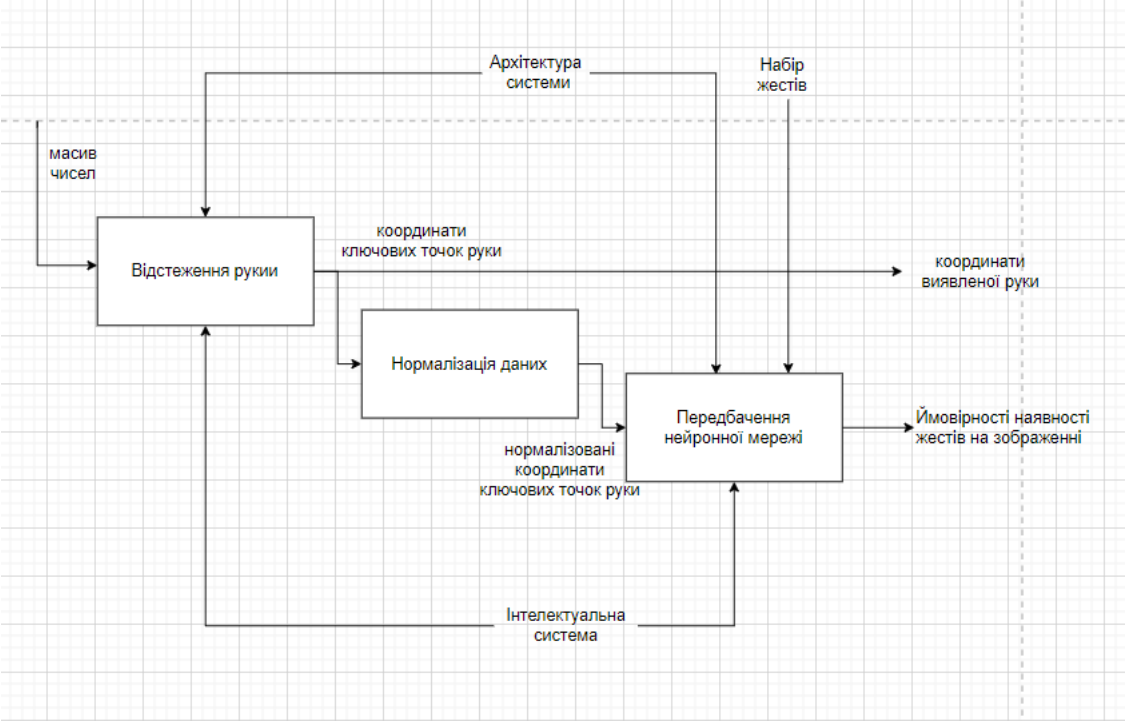


Рис.2.5 – Декомпозиція IDEF0 другого рівня для блоку «Аналіз даних» за використання двоступінчатого підходу

2.3 Необхідні програмні модулі для створення системи розпізнавання жестів

Для того щоб визначити, які програмні модулі потрібні в процесі створення системи, виділимо основні етапи цього процесу:

1. Збирання даних (фото, координати ключових точок руки)
2. Попередня обробка та анотація даних (обмежувальні рамки, нормалізація координат, форматування розмірності масивів, маркування)
3. Конфігурація нейронної мережі (визначення кількості класів, шарів, активаційних функцій тощо)
4. Тренування нейронної мережі (визначення кількості епох, умови закінчення тренування)
5. Тестування нейронної мережі (запуск системи розпізнавання у реальному часі, перевірка коректності розпізнавання)
6. Доповнення набору даних (опціонально)
7. Донавчання (опціонально)

На основі наведеного переліку, можна зробити висновок про необхідність наступних модулів:

- Модуль збирання даних
- Модуль попередня обробка та анотація даних
- Модуль тренування нейронної мережі
- Модуль візуалізації результатів розпізнавання

2.4 Одноступінчатий підхід

При дослідженні комбінованих архітектур CNN+LSTM, було знайдено приклади вдалих розробок для розпізнавання рухів [16]. В наведеній статті принцип побудови такої моделі описаний дуже загально, що ускладнює можливість її повторення. Оскільки в створений набір жестів було включено два динамічні жести, у даному випадку їх буде адаптовано: розімкнення долоні – просто відкрита долоня, стиснення руки в кулак – кулак.

2.4.1 Архітектура нейронної мережі

Для реалізації даного підходу було вирішено обрати SSD (Single-shot detector) модель, яка була розглянута в попередньому розділі, зважаючи на її переваги над іншими розглянутими моделями. Обрана архітектура – MobileNet. Загалом це невеликі моделі з низькою затримкою та малопотужними параметрами, налаштовані для роботи з обмеженими обчислювальними ресурсами у різних випадках використання, що дуже добре підходить для браузерних застосунків. Також значно меншими є розміри моделі: MobileNet потрібно лише 16–18 МБ, у той час як, наприклад, модель VGG16 може зайняти до 500 МБ дискового простору,.

MobileNet є набагато швидшою за інші моделі ЗНМ, за рахунок використання нового типу згорткового шару, відомого як глибинна роздільна згортка (depthwise separable convolution). Основна відмінність між двовимірними згортками та глибинною згорткою полягає в тому, що перші виконуються для всіх вхідних каналів зображення, тоді як у других до кожного

каналу застосовується власний згортків фільтр, а далі перетворення початкової кількості кольорових каналів для утворення нових N каналів, наприклад, на рис.2.6 п’ять каналів перетворюються у три.

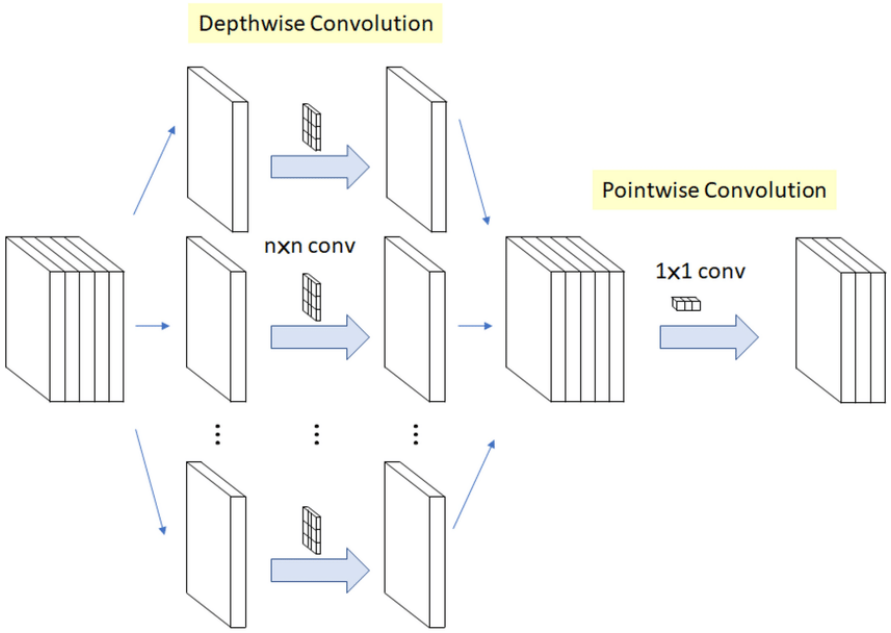


Рис. 2.6 – Глибинна роздільна згортка

Припустимо, що потрібно перетворити тензор 8x8x3 на тензор 8x8x256. Використовуючи 2D згортки загальна кількість операцій становитиме (8x8) x (5x5x3) x (256) = 1 228 800, а використовуючи глибинні роздільні згортки:

(8x8) x (5x5x1) x (3) = 3800

(8x8) x (1x1x3) x (256) = 49,152

3800+49152 = 53952

Таким чином, потрібно в 1 228 800/53 952 = 23 рази менше операцій, але зберігається висока точність розпізнавання, на що вказується в роботі розробників цієї архітектури при порівнянні з іншими відомими моделями (рис.2.7).

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Рис.2.7 – Порівняння MobileNet з іншими відомими моделями [17]

Загальна архітектура MobileNet наведена на рис.2.8.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1 Conv / s1	$3 \times 3 \times 512$ dw $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$ $14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Рис.2.8 – Загальна архітектура MobileNet [17]

2.4.1 Підготовка набору даних

Маркування даних є важливим кроком машинного навчання з вчителем. В спільноті машинного навчання кажуть, що якість навчальних даних визначає якість моделі. Те саме стосується анотацій, які використовуються для маркування даних.

Обмежувальні рамки (рис.2.9) є найбільш часто використовуваним типом анотації в комп'ютерному баченні, зазвичай в задачах виявлення та локалізації об'єктів. Обмежувальні рамки – це прямокутники, які використовуються для визначення розташування цільового об'єкта. Їх можна визначити за координатами x і y верхнього лівого кута та координатами x і y нижнього правого кута прямокутника.

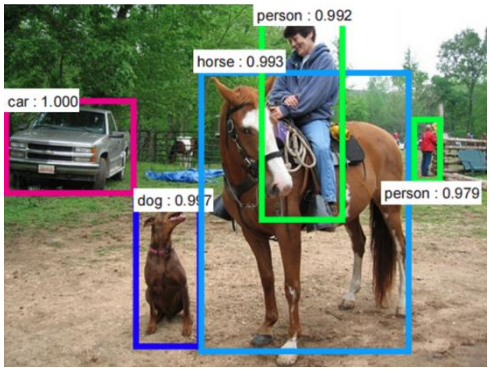


Рис.2.9 – Обмежувальні рамки на зображенні

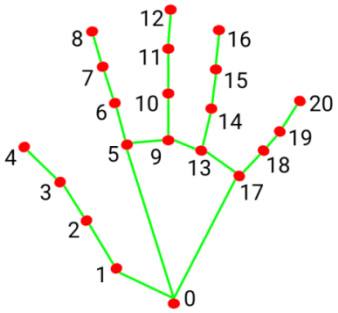
Отже, для формування набору даних, нам потрібно не лише зібрати достатню кількість зображень жестів кожного класу, а також додати до кожного зображення анотацію. Зазвичай анотацію зберігають у json або xml форматах.

2.5 Двоступінчатий підхід

2.5.1 Система відстеження рук

Для розпізнавання жестів за цим підходом, як зазначалось вище, потрібна система відстеження рук, яка зможе розпізнавати ключові точки руки. Створення такої системи є досить складним завданням, яке потребує великого набору тренувальних даних, відповідних обчислювальних потужностей для їх опрацювання, а також комбінації кількох моделей розпізнавання.[18] У даній роботі для виокремлення ключових точок руки буде застосовано фреймворк MediaPipe від Google.

MediaPipe – кросплатформна платформа з відкритим вихідним кодом для побудови конвеєрів для обробки перцептивних даних різних модальностей, таких як відео та аудіо. Цей підхід забезпечує високоточне відстеження рук і пальців, використовуючи машинне навчання, щоб визначити 21 тривимірну ключову точку (рис.2.10) руки лише з одного кадру. У той час як сучасні підходи покладаються головним чином на потужні робочі середовища, дана технологія забезпечує високу продуктивність у реальному часі навіть на смартфоні [19]. Окрім цього MediaPipe також має JavaScript Solution API, що дозволяє легко підключати та використовувати моделі для власних вебзастосунків



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Рис.2.10 – Ключові точки руки, які розпізнає MediaPipe

Розпізнавання рук доступне в рішення MediaPipe Hands та MediaPipe Holistic. Друге окрім рук також може розпізнавати точки обличчя та позу тіла людини. Ці дані не є важливими для системи розпізнавання жестів, але в даній роботі буде обрано саме його, адже на відміну від MediaPipe Hands дана модель зберігає семантичну послідовність у всьому тілі та його частинах, запобігаючи змішування лівої та правої рук або частин тіла однієї людини в кадрі з іншою. Таким чином ми зменшимо ймовірність відстеження рук інших людей, що попали у кадр випадково, та збоїв системи, які можуть через це виникнути.

2.5.1 Архітектура нейронної мережі

За даного підходу до розпізнавання жестів можна використати звичайний перцептрон, але тоді система буде здатна розпізнавати лише статичні жести. Оскільки в створений набір жестів було включено два динамічні жести буде доцільним використати розглянуту раніше LSTM мережу. У таблиці 2.1 наведено детальні характеристики створюваної нейронної мережі.

Таблиця 2.1 – Характеристики LSTM мережі

№	Параметр	Значення
1	Тип архітектури	LSTM
2	Розмірність вхідних даних	(20, 126) 20 – довжина послідовності кадрів 21 * 3 * 2 = 126 – кількість ключових точок * виміри * кількість рук
3	Кількість вихідних нейронів	6 – кількість жестів
4	Типи шарів	LSTM, LSTM, LSTM, Повнозв’язний,

		Повнозв’язний, Повнозв’язний,
5	Кількість нейронів на кожному шарі	64, 128, 64, 64, 32, 6
6	Активаційна функція вхідного шару	ReLU
7	Активаційна функція прихованих шарів	ReLU
8	Активаційна функція вихідного шару	Softmax
9	Оптимізатор	Adam
10	Функція обрахунку помилки	Categorical crossentropy

2.4.1 Підготовка набору даних

Для формування набору даних потрібно зберегти послідовності з 20 наборів координат ключових точок руки розпізнаних системою відстеження один за одним. Тобто в даному випадку не потрібно проводити анотацію вручну, адже це зробить за нас MediaPipe. Але потрібно знайти спосіб записати ці послідовності, для цього необхідно буде створити допоміжний скрипт.

Також сліз зауважити, що координати мають бути нормалізовані. Наприклад, коли камера буде захоплювати один і той самий жест в верхньому лівому кутку та в нижньому правому кутку, система не буде розуміти, що це один і той самий жест, адже координати ключових точок будуть суттєво відрізнятись. Тому усі значення координат мають бути приведені до діапазону [0, 1] або [-1, 1].

Якщо розглядати це на прикладі векторів, то на рис.2.11 видно неозброєним оком, що сині вектори однакові, але координати вони мають різні. Те саме трапляється і з ключовими точкам рук.

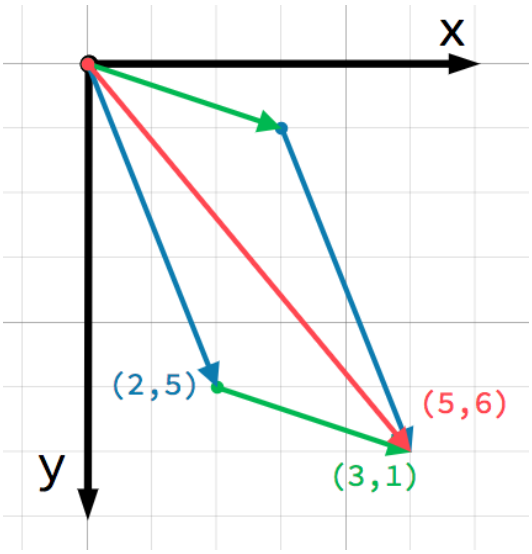


Рис.2.11 – Вектори

Висновок до другого розділу

Під час роботи над другим розділом було сформовано перелік взаємодій для наведеного прикладу та відповідний їм набір жестів; проведено моделювання процесів розпізнавання жестів для кращого розуміння потоків даних системи, що проектується; визначено етапи створення системи розпізнавання жестів та програмні модулі, які знадобляться на цих етапах; визначено архітектури нейронних мереж та принцип формування набору даних для кожного з підходів. Як наслідок було сформоване чітке розуміння того, що потрібно для реалізації проекту.

Розділ 3. Реалізація

3.1 Вибір інструментів реалізації

Для реалізації системи розпізнавання жестів було обрано мову програмування Python, вона підтримує створення широкого спектру додатків, має простий та лаконічний синтаксис, розробники вважають її чудовим вибором для проектів з використанням штучного інтелекту, машинного та глибинного навчання, адже під неї створена велика кількість відповідних бібліотек та інших інструментів, що допомагають у процесі розробки.

Бібліотеки, які будуть використані в даній роботі:

- OS – надає функції для взаємодії з операційною системою; знадобиться для написання допоміжних скриптів для збирання даних для тренування та тестування, збереження натренованої моделі та іншими взаємодіями з файловою системою проекту.
- NumPy – надає функції для роботи з багатомірними масивами.
- OpenCV – бібліотека комп’ютерного бачення та машинного навчання з відкритим кодом, в якій окрім алгоритмів наявні інструменти для перехоплення відеопотоку з камери комп’ютера та інструменти для малювання графічних елементів, що візуалізуватимуть результати роботи певного алгоритму.
- Time – забезпечує різні функції, пов’язані з часом; буде використаний у скрипті для автоматичного збирання даних.
- scikit-learn – надає прості та ефективні інструменти для аналізу даних; будуть використані функцію випадкового розділення набору даних на тренувальні та тестові, а також модуль, що реалізує декілька функцій втрати, оцінки та корисності для вимірювання ефективності класифікації.

В якості інструменту для створення моделей для системи розпізнавання було обрано TensorFlow – платформу з відкритим кодом яка має всеосяжну, гнучку екосистему інструментів, бібліотек і спільноту, які дозволяють

дослідникам використовувати найсучасніші технології машинного навчання, а розробникам легко створювати й розгортати програми, що працюють з використанням машинного навчання.

TensorFlow надає можливість як використовувати готові архітектури нейронних мереж, так і створювати власні.

TensorFlow також є ідеальним вибором, враховуючи сферу застосування розроблюваної системи – управління вебзастосунками, адже він дозволяє розгортати моделі машинного навчання у середовищах, що використовують мову програмування JavaScript, тобто у браузері та в Node.js. Для того, щоб перетворити модель до потрібного формату потрібно встановити модуль tensorflowjs та виконати конвертацію.

Весь програмний код буде написано у Jupyter Notebook – це інтерактивне обчислювальне веб-середовище для створення так званик ноутбук-документів. Документи Jupyter Notebook являють собою упорядкованы списоки комірок, які можуть містити код, текст (вводиться з використанням спеціальної розмітки), графіки та мультимедійні дані.

3.2 Одноступінчатий підхід

Додаткові інструменті для реалізації системи з одноступінчатим підходом:

TensorFlow Object Detection API – це фреймворк із відкритим кодом, побудований на основі TensorFlow, що дозволяє легко створювати, навчати та розгортати моделі виявлення об’єктів на зображеннях.

LabelImg — це графічний інструмент анотації зображень і визначення обмежувальних рамок для об’єктів на зображеннях.

3.2.1 Структура проекту

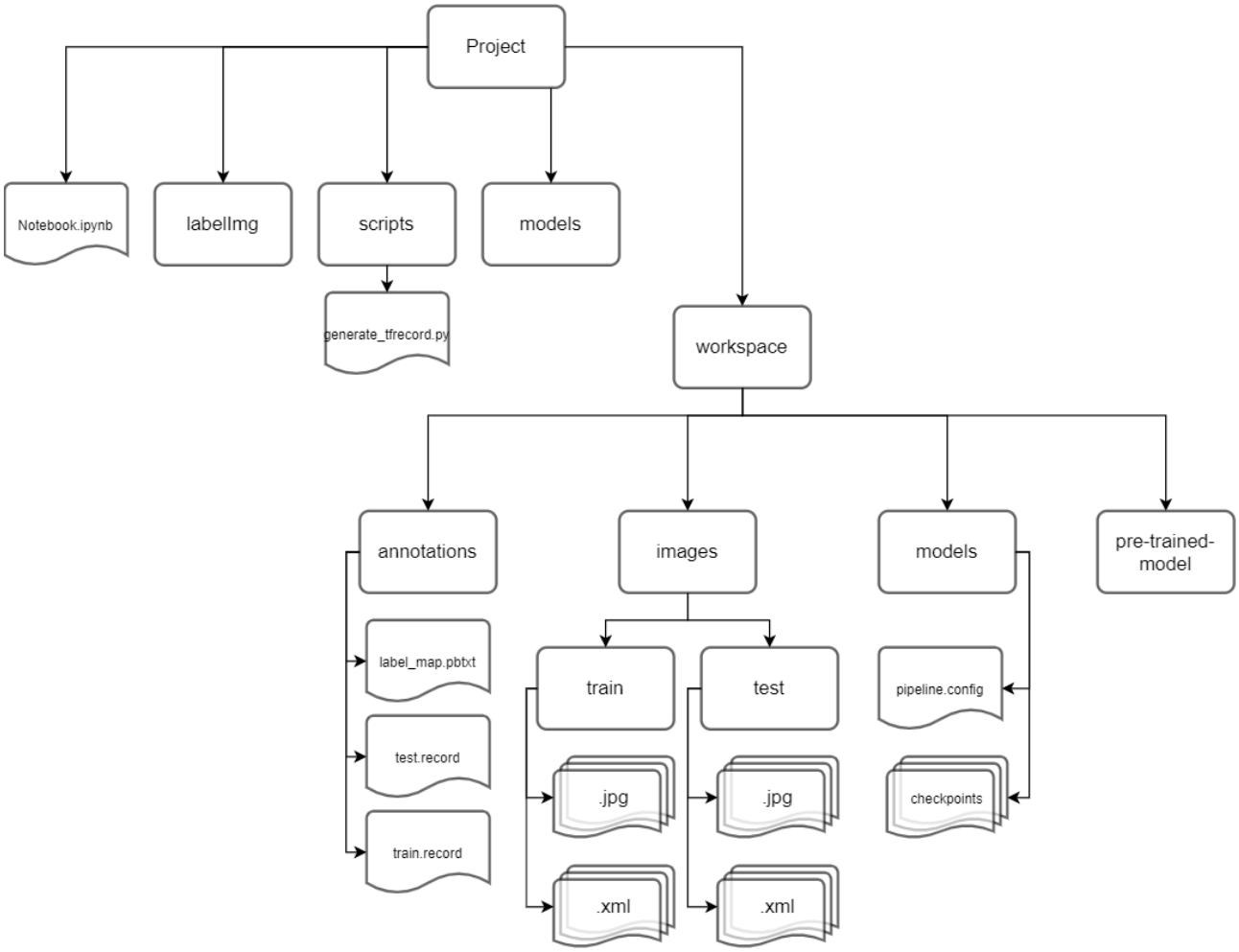


Рис.3.1 – Структура проекту реалізації одноступінчатої системи розпізнавання жестів

Опишемо вміст та призначення основних директорій та файлів проекту (рис.3.1):

Notebook.ipynb – містить весь код для створення системи.

labelImg – директорія, що містить програму LabelImg.

scripts – директорія для допоміжних скриптів.

generate_tfrecord.py – скрипт від TensorFlow Object Detection API, що допомагає згенерувати tfrecord-файли; такі файли можна розглядати як обгортку всіх окремих зразків даних, яка допомагає оптимізувати роботу з великими наборами даних.

models – директорія, в якій міститься MobileNet мережа.

workspace – директорія, в якій виконуються усі основні маніпуляції.

workspace/annotations – директорія, в якій розміщується label_map.pbtxt (представлення всіх жестів, які має розпізнавати система; включає назву та id), і tfrecord-файли для навчання моделі.

workspace/images – директорія, що містить папки train та test, з даними відповідно для навчаннц та тестування моделі, в кожній з яких зберігаються пари зображення та xml-аннотація з координатами обмежувальних рамок.

workspace/models – директорія, що містить pipeline.config з налаштування MobileNet під вирішувану задачу, та усі проміжні стани мережі між тренуваннями.

workspace/pre-trained-models – директорія, що містить попередньо натреновану модель.

3.2.2 Розробка системи

Для розробки даної системи було проведено наступні етапи:

1. Зібрані зображення для глибокого навчання за допомогою веб-камери та OpenCV. Було зібрано по 30 зображень для кожного з шести жестів.
2. Проведена анотація зображень за допомогою LabelImg (рис.3.2)

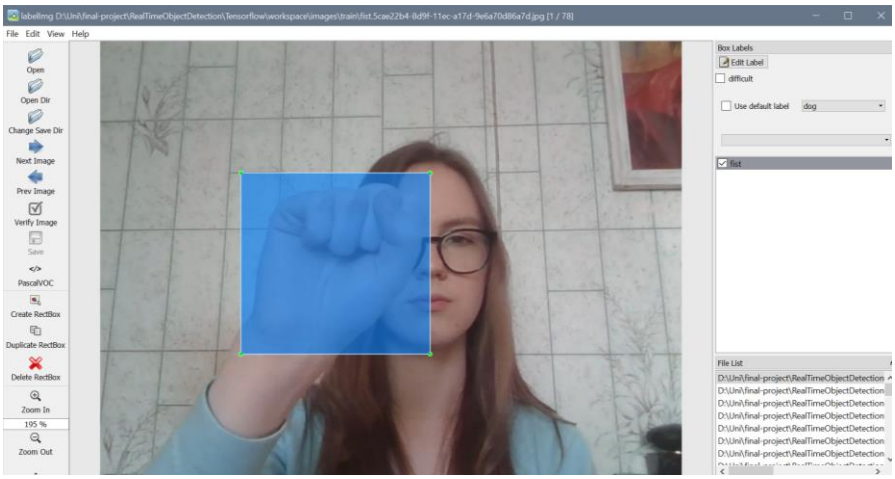


Рис.3.2 – Інтерфейс програми LabelImg

3. Налаштовано конфігурацію Tensorflow Object Detection API:

   Вирішено обрати модель SSD MobileNet V2 FPNLite 320x320 з колекції TensorFlow 2 Detection Model Zoo, попередньо навчених

моделей на наборі даних COCO 2017. Обрана модель має досить високу швидкість розпізнавання – 22 мс.

Було створено label_map (рис.3.3) та tfrecord-файли для навчання мережі.

Оновлено конфігурацію (вказано кількість класів розпізнавання – 6, тип задачі – детектування, вказано шляхи до моделі, label_map та tfrecord-файлів).

```
[{'name':'thumbsup', 'id':1},
 {'name':'thumbsdown', 'id':2},
 {'name':'openpalm', 'id':3},
 {'name':'fist', 'id':4},
 {'name':'pointup', 'id':5},
 {'name':'pointdown', 'id':6}]
```

Рис.3.3 – label_map

4. Застосувано трансферне навчання для донавчання штучної нейронної мережі

   Попередньо навчена модель – це збережена мережа, яка раніше була навчена на великому наборі даних, як правило, для великомасштабної класифікації зображень. Така модель використовуєте або як є, або проводиться трансферне навчання, щоб налаштувати цю модель для певного завдання. Ідея, що лежить в основі трансферне навчання для класифікації зображень, полягає в тому, що якщо модель навчати на великому і достатньо загальному наборі даних, ця модель буде ефективно служити загальною моделлю візуального світу. Потім можна скористатися перевагами цих моделей, не починаючи з нуля навчання власної мережі для розпізнавання об’єктів.

   Загалом було проведено 5000 епох тренування, що зайняло близько 8 годин, та отримані наступні результати точності мережі:

```
I0215 01:13:14.039462  4252 model_lib_v2.py:707] Step 5000 per-step time 1.078s
INFO:tensorflow:{'Loss/classification_loss': 0.051714897,
 'Loss/localization_loss': 0.019476729,
 'Loss/regularization_loss': 0.12571397,
 'Loss/total_loss': 0.1969056,
 'learning_rate': 0.078691795}
I0215 01:13:14.055086  4252 model_lib_v2.py:708] {'Loss/classification_loss': 0.051714897,
 'Loss/localization_loss': 0.019476729,
 'Loss/regularization_loss': 0.12571397,
 'Loss/total_loss': 0.1969056,
 'learning_rate': 0.078691795}
```

Рис.3.3 – Результати навчання мережі MobileNet

5. Отримано систему, що розпізнає жести в режимі реального часу та відображає цей процес за допомогою OpenCV.

Наведемо також перелік блоків створених в ході розробки програми:

Блок 1: Збирання набору даних

Блок 2: Встановлення шляхів до основних папок

Блок 3: Визначення label_map

Блок 4: Створення tfrecord-файлів

Блок 5: Конфігурація MobileNet

Блок 6: Завантаження моделі

Блок 7: Тестування в реальному часі

Блок 8: Збереження та конвертація в TensorflowJS

3.2.3 Тестування

Навіть при тому, що значення помилки на тестових даних невелике, в реальному часі система розпізнає жести не завжди надійно, плутає долоні з обличчям та іншими об’єктами. Результати можна виправити доповненням набору даних та донавчанням, але воно може зайняти ще більше часу, адже набір даних буде об’ємніший. Також ситуація ще більш ускладниться, якщо знадобиться додати новий жест до системи розпізнавання.

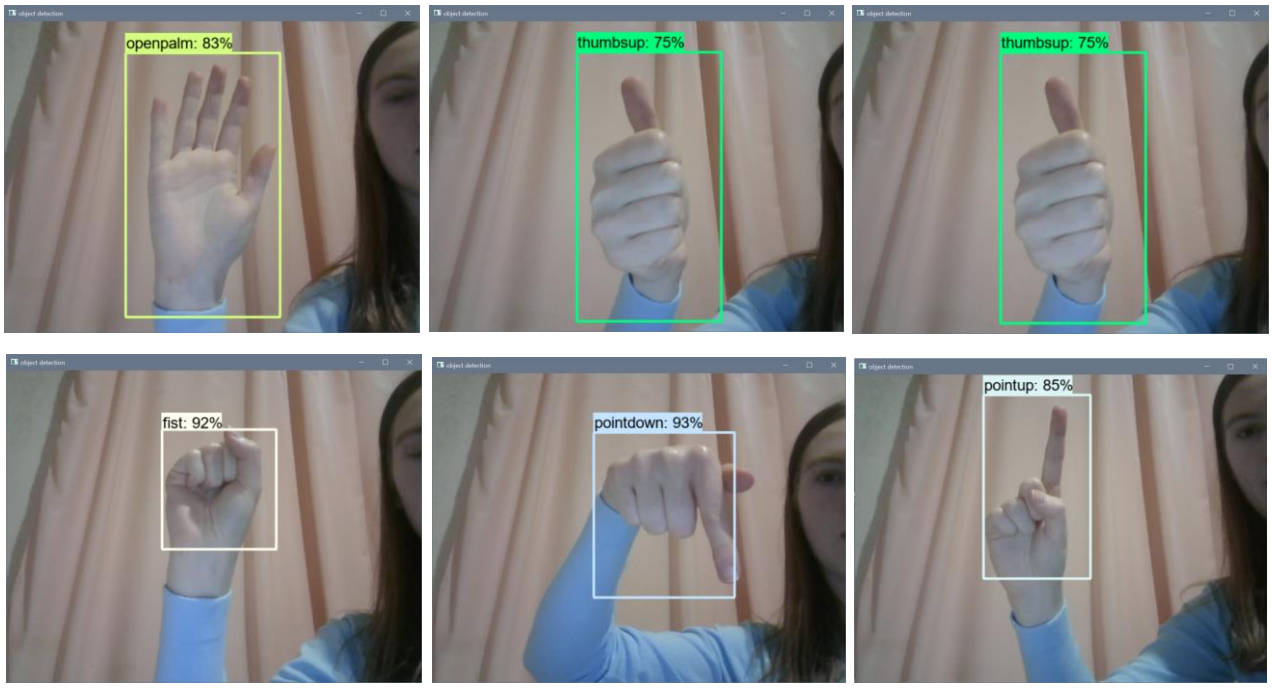


Рис.3.4 – Розпізнавання жестів мережею MobileNet в реальному часі

3.3 Двоступінчатий підхід

Для реалізації даного варіанту системи також додатково був встановлений пакет mediapipe.

3.3.1 Структура проекту

Опишемо вміст та призначення основних директорій та файлів проекту (рис.3.5):

Notebook.ipynb – містить весь код для створення системи.

Data – директорія, де міститься набір даних; складається з піддиректорій для кожного жесту, в свою чергу кожна така піддиректорія має певну кількість екземплярів даних – піддиректорій, що містять у собі 20 npy-файлів, відповідно до кількості вхідних параметрів LSTM-мережі. npy-файл – це стандартний бінарний файл для зберігання одного довільного NumPy-масиву на диску.

Logs – директорія, де зберігається інформація про тренування нейронної мережі, далі використовуючи її можна, наприклад, побудувати графіки тренування.

action.h5 – файл, в якому зберігаються ваги моделі після тренування.

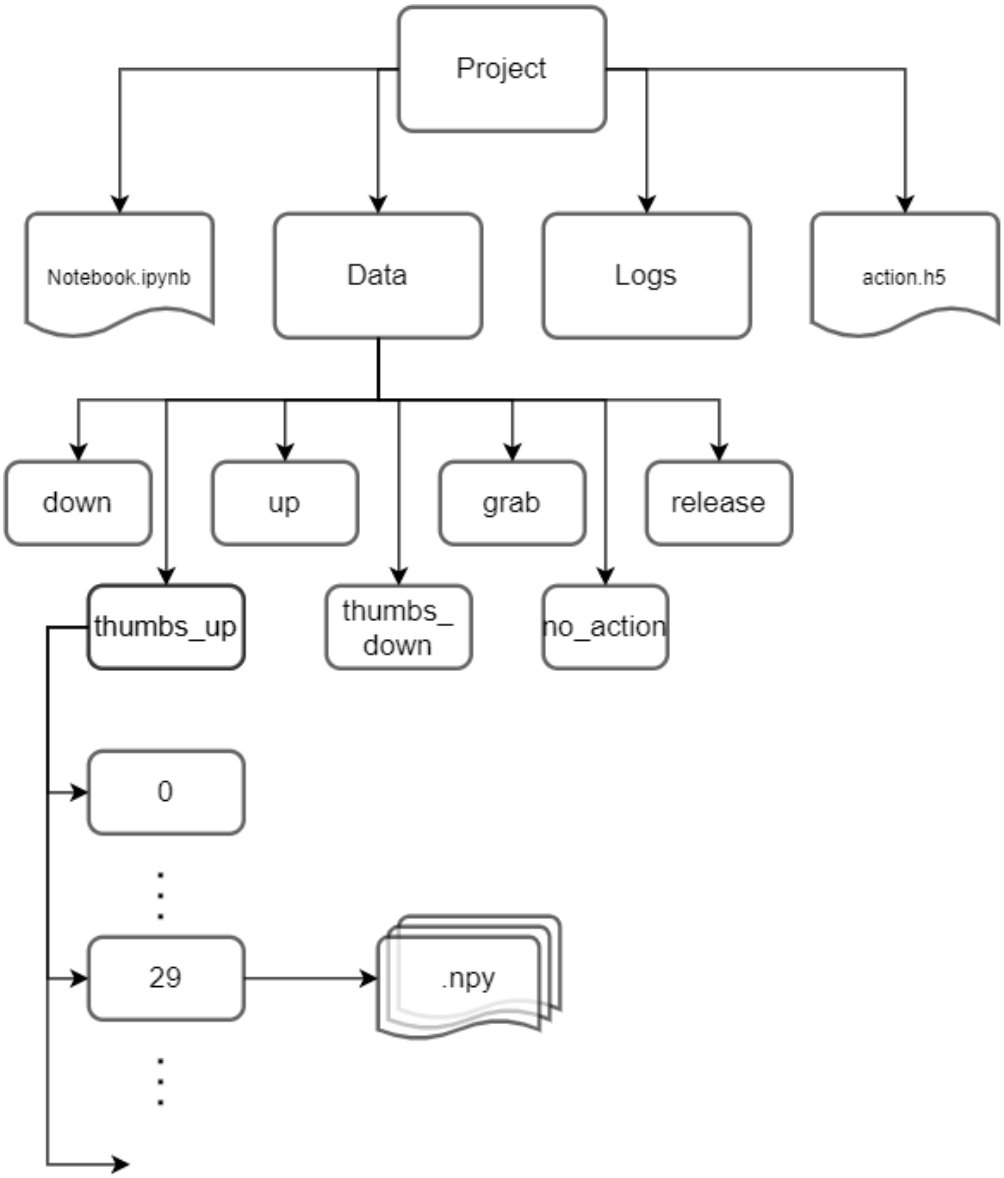


Рис.3.5 – Структура проекту реалізації двоступінчатої системи розпізнавання жестів

3.2.2 Розробка системи

Для розробки даної системи було проведено наступні етапи:

1. Спочатку було підключено mediapipe: створено функції mediapipe_detection для відстеження рук та draw_styled_landmarks для малювання ключових точок на зображені (рис.3.5).

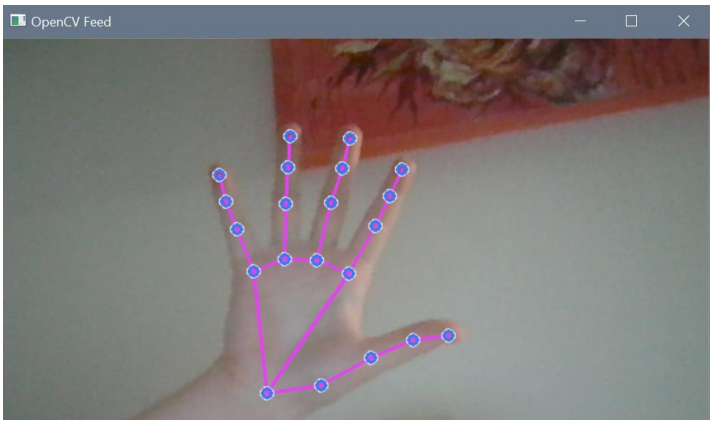


Рис.3.5 – Розпізнавання ключових точок долоні mediapipe

2. Далі, оскільки використовується MediaPipe Holistic, потрібно написати функцію extract_keypoints для виокремлення лише ключових точок рук. В результаті функція має повертати масив чисел довжиною 21*3*2=126 (рис.3.6). Також не потрібно додатково проводити нормалізацію, адже було обрано отримувати набори LEFT_HAND_LANDMARKS та RIGHT_HAND_LANDMARKS, в яких x і y нормалізуються до [0,0, 1,0] за шириною та висотою зображення відповідно, а z позначає глибину та визначається положенням точки на зап'ясті, чим менше це значення, тим ближче рука знаходиться до камери.

```
array([ 9.62735891e-01,  7.66022205e-01,  6.19830416e-07,  8.58439922e-01,
        7.28295386e-01, -3.48105356e-02,  7.71100819e-01,  6.07253611e-01,
       -4.51462492e-02,  7.16628075e-01,  5.02303600e-01, -5.34655377e-02,
        6.60742879e-01,  4.39414829e-01, -6.11573979e-02,  8.19896460e-01,
        4.09497917e-01, -1.06983688e-02,  7.92870224e-01,  2.59389818e-01,
       -3.13702933e-02,  7.77577817e-01,  1.69341356e-01, -5.19745648e-02,
        7.64757454e-01,  9.52426493e-02, -6.87692463e-02,  8.80864918e-01,
        3.86182517e-01, -1.60265416e-02,  8.52263808e-01,  2.18386546e-01,
       -3.20546925e-02,  8.33406329e-01,  1.17793411e-01, -5.16008511e-02,
        8.16964746e-01,  3.98816466e-02, -6.75681010e-02,  9.43530083e-01,
        3.93918008e-01, -2.83256695e-02,  9.24255729e-01,  2.27160856e-01,
       -5.02969399e-02,  9.11758840e-01,  1.32734418e-01, -6.89106658e-02,
```

Рис.3.6 – Фрагмент вихідного масиву функції extract_keypoints

3. Далі за допомогою спеціально написаного скрипту, веб-камери, бібліотек time та OpenCV були сформовані та заповнені директорії для збереження набору даних.
4. Зібрані дані поділені на навчальні та тестові випадковим чином.

5. Побудована LSTM-мережа за визначеними у попередньому розділі характеристиками.
6. Проведено тренування, яке було зупинено на 83 епосі (рис.3.7) через досягнення високої точності на тестових даних. Тренування зайняло близько 7 хвилин. Графік динаміки навчання нейронної мережі наведений на рис.3.8.

```
Epoch 80/250
6/6 [==============================] - 1s 90ms/step - loss: 0.0062 - categorical_accuracy: 0.9947
Epoch 81/250
6/6 [==============================] - 1s 88ms/step - loss: 0.0063 - categorical_accuracy: 0.9947
Epoch 82/250
6/6 [==============================] - 1s 91ms/step - loss: 0.0050 - categorical_accuracy: 1.0000
Epoch 83/250
6/6 [==============================] - 1s 87ms/step - loss: 0.0021 - categorical_accuracy: 1.0000
```

Рис.3.7 – Эпохи навчання LSTM-мережі

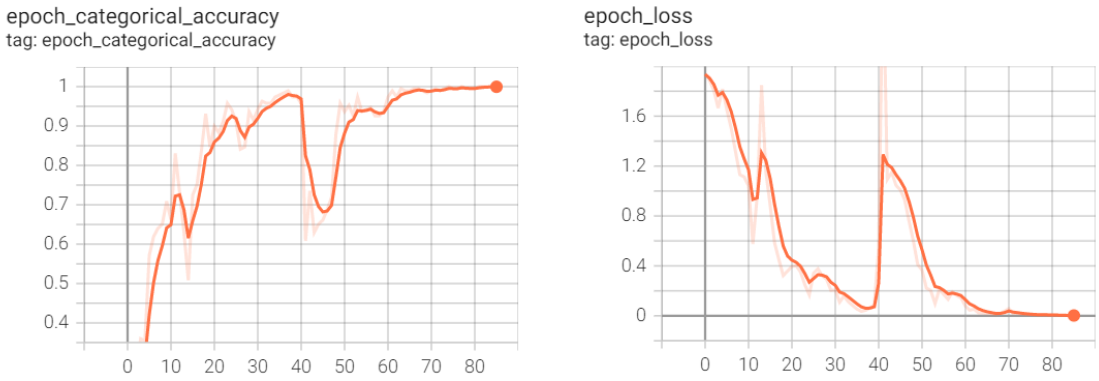


Рис.3.8 – Динаміка навчання нейронної мережі (точність та функція втрат)

7. Під час тестування роботи системи у реальному часі, розпізнавання виявилось не дуже точним, але оскільки навчається система швидко, було вирішено збільшити вдвічі набір даних, та провести донавчання (рис.3.9).

```
Epoch 80/250
6/6 [==============================] - 1s 90ms/step - loss: 0.0062 - categorical_accuracy: 0.9947
Epoch 81/250
6/6 [==============================] - 1s 88ms/step - loss: 0.0063 - categorical_accuracy: 0.9947
Epoch 82/250
6/6 [==============================] - 1s 91ms/step - loss: 0.0050 - categorical_accuracy: 1.0000
Epoch 83/250
6/6 [==============================] - 1s 87ms/step - loss: 0.0021 - categorical_accuracy: 1.0000
```

Рис.3.9 – Эпохи донавчання LSTM-мережі

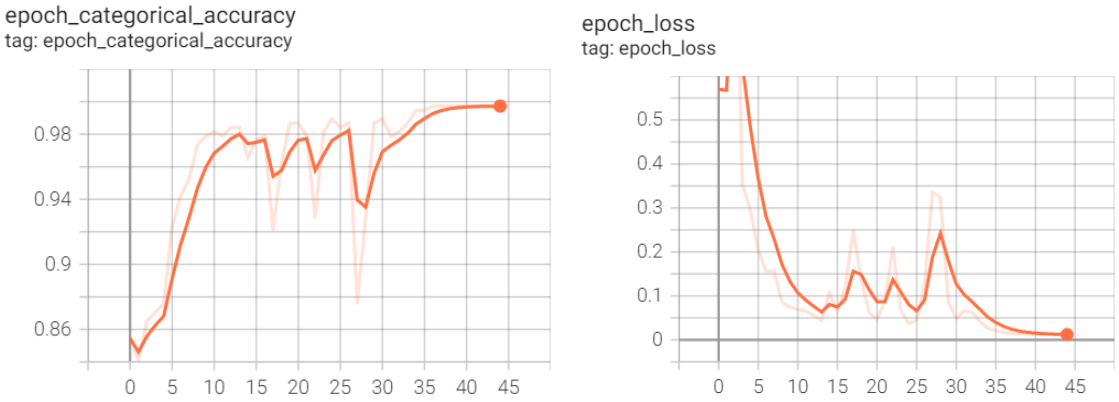


Рис.3.10 – Динаміка донавчання нейронної мережі (точність та функція втрат)

8. Також була отримано матриці помилок для кожного жесту (рис.3.11).

```
[[[36,  0],
  [ 1,  5]],

 [[38,  0],
  [ 0,  4]],

 [[34,  1],
  [ 0,  7]],

 [[37,  1],
  [ 0,  4]],

 [[33,  0],
  [ 1,  8]],

 [[36,  0],
  [ 0,  6]],

 [[36,  0],
  [ 0,  6]]],
```

Рис.3.10 – Матриці помилок для кожного жесту в послідовності ['thumbs_up', 'thumbs_down', 'up', 'down', 'grab', 'release', 'no_action'].

6. Отримано систему, що розпізнає жести в режимі реального часу та відображає цей процес за допомогою OpenCV.

3.3.2 Тестування

Для проведення тестування в режимі реального часу було написано скрипт, що забезпечує візуалізацію розпізнавання. Зліва на зображенні, що транслюється з відеокамери, розташовано діаграму, яка показує ймовірності розпізнавання того чи іншого жесту та динамічно змінюється. Якщо певний жест розпізнається з ймовірністю більшою за 0.85, то назва цього жесту записується у рядок зверху.

Після донавчання нейронної мережі, результати розпізнавання стали добрими навіть у режимі реального часу.

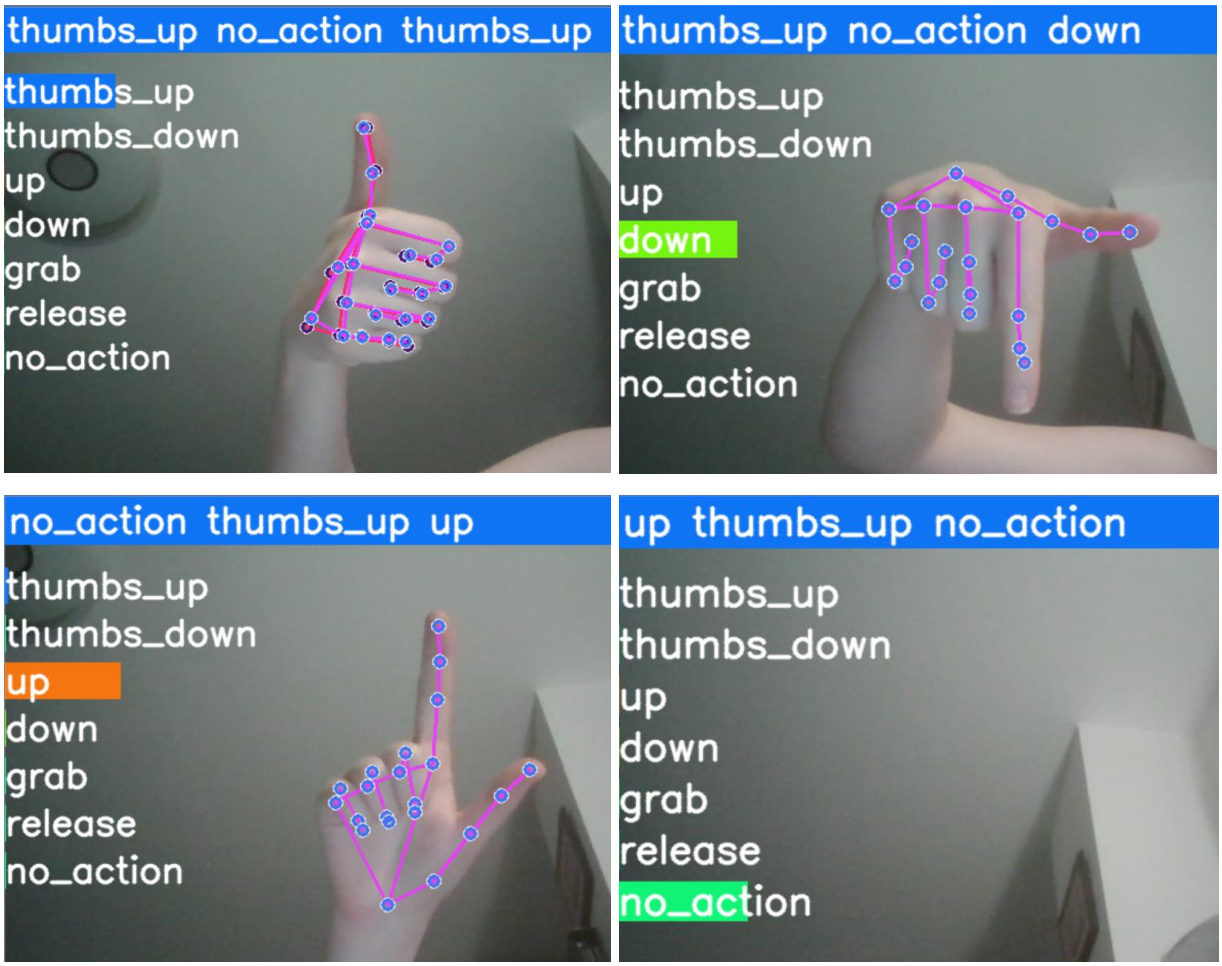


Рис.3.4 – Розпізнавання жестів за допомогою mediappipe та LSTM мережі в реальному часі

Наведемо також перелік блоків створених в ході розробки програми:

Блок 1: Візуалізація ключових точок mediappipe

Блок 2: Вилучення потрібних ключових точок

Блок 3: Збирання набору даних

Блок 4: Доповнення набору даних

Блок 5: Попередня обробка даних та маркування

Блок 6: Побудова і навчання LSTM мережі

Блок 7: Донавчання LSTM мережі

Блок 8: Оцінка моделі з Confusion Matrix

Блок 9: Тестування в реальному часі

Блок 10: Збереження та конвертація в TensorflowJS

3.4 Порівняння

Таблиця 3.1 – Порівняння створених систем

	Одноступінчатий підхід	Двоступінчатий підхід
Можливість розпізнавання динамічних жестів	Складність реалізації розпізнавання динамічних жестів, не реалізовано у даній роботі.	За допомогою mediappipe зменшується набір даних за якими проводиться розпізнавання, тому розпізнавання навіть динамічних жестів не становить проблеми.
Набір даних	Потрібний дуже великий набір даних (потрібні фото жестів різних людей, при різному освітленні, з різним оточенням).	Більш універсальний набір даних для тренування (менший вплив освітлення, не залежить від відтінку шкіри, оточення).
Тренування	Довгий час тренування (~8 годин), але все одно погано працює в режимі реального часу.	Швидке тренування (15 хвилин) з до навчанням, досить добре працює в режимі реального часу.
Швидкість	Моментальне	Трохи довше розпізнає

розпізнавання	розпізнавання.	жести, які є статичними, оскільки система проектувалась універсальною як для динамічних, так і для статичних жестів
Збирання набору даних	Потрібно збирати фото, а потім ще проводити анотацію з обмежувальними рамками, що є досить довгим процесом, враховуючи також, що даних для цього підходу потрібно більше.	Не потрібно проводити анотацію, лише записати послідовності координат ключових точок руки у правильні директорії, скориставшись створеним скриптом.
Розмір моделі	Невелика, спеціально розроблена для мобільних та вебзастосунків	Маленька

Висновок до третього розділу

В останньому розділі ВКР було описано процес створення двох систем розпізнавання жестів, що використовують різні підходи. Обґрунтовано вибір технологій реалізації, описано структуру проектів, наведено перелік отриманих блоків програм під час створення систем. Також було проведено тестування систем і порівняно їх за ключовими критеріями.

В ході порівняння було визначено, що розглянутий двоступінчатий підхід суттєво перевершує одноступінчатий. Систему, що використовує двоступінчатий підхід, набагато легше та швидше реалізувати, вона є більш універсальною і гнучкою, потребує меншого набору даних для навчання, при цьому не поступається точністю, до неї буде легше додати нові жести, а

правильно підібрані технології та невеликий розмір дозволять ефективно впровадити систему до будь-якого вебзастосунку.

Висновки

В ході виконання даної випускної кваліфікаційної роботи було спроектовано та реалізовано два варіанти системи розпізнавання жестів, які використовують різні підходи. В результаті тестування та порівняння створених систем було визначено, що найефективнішою, простішою та легшою в реалізації є система, яка розпізнає жести за ключовими точками, отриманими системою відстеження рук.

Дослідивши предметну область, а саме розвиток систем управління інтерфейсами, сучасні технології для жестового управління інтерфейсами та основні алгоритми глибинного навчання для розпізнавання об’єктів та рухів, було підібрано найкращі рішення для реалізації системи розпізнавання жестів саме для управління вебзастосунками, враховуючи особливості та обмеження середовища у якому вони працюють.

Обрані такі моделі нейронних мереж та інструменти програмування, які які зможуть забезпечити стабільну роботу, швидке реагування та коректність роботи системи у випадку її впровадження до вебзастосунку.

Також були створені допоміжні програмні модулі, наприклад, модуль для збору даних та модуль для тестування системи у реальному часі. Одною з ідей вдосконалення створеної системи є розробка у майбутньому графічного інтерфейсу для створення системи розпізнавання таких жестів, які потрібні конкретному користувачу чи проекту, без необхідності взаємодії з кодом програми. За наявності такого інструменту все більше розробників вебзастосунків можуть почати впроваджувати нову технологію до своїх проектів, що приведе до переходу на новий рівень людино-машинної взаємодії, зробить її приємнішою та зручнішою.

Література

1. Що таке інтерфейс, різновиди інтерфейсів // Онлайн-платформа Creative Practice. – 2021 [Електронний ресурс] – Режим доступу: https://cases.media/creativepractice/article/sho-take-interfeis-riznovidi-interfeisiv
2. A Dictionary of Computer Science. Seventh Edition / Andrew Butterfield, Gerard Ekembe Ngondi, and Anne Kerr – Oxford University Press, 2016. – p.258 [Електронний ресурс] – Режим доступу: https://books.google.nl/books?id=GDgICwAAQBAJ&printsec=frontcover&hl=ru#v=onepage&q&f=false
3. Sign Language MNIST: Drop-In Replacement for MNIST forHand Gesture Recognition Tasks // Kaggle. – 2017[Електронний ресурс] – Режим доступу: https://www.kaggle.com/datamunge/sign-language-mnist
4. Jian Zhao, Jingna Mao, 2017 IEEE Biomedical Circuits and Systems Conference (BioCAS) A miniaturized wearable wireless hand gesture recognition system employing deep-forest classifier [Електронний ресурс] – Режим доступу: https://www.semanticscholar.org/paper/A-miniaturized-wearable-wireless-hand-gesture-Zhao-Mao/68ddd0cca4c0da544599f9e04d3a2e7b5e6d2c70
5. Global Gesture Recognition Market Size, Industry Report, 2019-2025 (Report Overview) [Електронний ресурс] – Режим доступу: https://www.grandviewresearch.com/industry-analysis/gesture-recognition-market
6. Jim Simons, Pose Detection in Unreal [Електронний ресурс] – Режим доступу: https://docs.ultraleap.com/ultralab/pose-detection-blog.html
7. The complete guide to professional motion capture [Електронний ресурс] – Режим доступу: https://www.rokoko.com/insights/the-complete-guide-to-professional-motion-capture
8. Classification of Hand Movements Using MYO Armband on an Embedded Platform, MDPI, 2021 [Електронний ресурс] – Режим доступу:

https://www.semanticscholar.org/paper/Classification-of-Hand-Movements-Using-MYO-Armband-Javaid-Tiwana/0b8d14bbdaa75efada81257da740843150e475fa

9. Software for Myo armband [Електронний ресурс] – Режим доступу: https://github.com/balandinodidonato/MyoToolkit/blob/master/Software%20for%20Thalmic%27s%20Myo%20armband.md
10. François Chollet, Deep Learning with Python, Second Edition. – Manning, 2021 – p.148
11. Mohamed Elgendy, Deep Learning for Vision Systems. – Manning, 2020 – p.293-325
12. Object Detection in 2022: The Definitive Guide [Електронний ресурс] – Режим доступу: https://viso.ai/deep-learning/object-detection/#:~:text=on%20Viso%20Suite-,Most%20Popular%20Object%20Detection%20Algorithms,the%20single-shot%20detector%20family.
13. Saul Dobilas, LSTM Recurrent Neural Networks — How to Teach a Network to Remember the Past, 2022 [Електронний ресурс] – Режим доступу: https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-remember-the-past-55e54c2ff22e
14. What is computer vision? – IBM [Електронний ресурс] – Режим доступу: https://www.ibm.com/topics/computer-vision
15. Hand tracking and Gesture Recognition. – Intel RealScence, 2020 [Електронний ресурс] – Режим доступу: https://www.intelrealsense.com/hand-tracking-overview/
16. Human Action Recognition in Videos using a Robust CNN LSTMApproach, / Carlos Ismael Orozco, Eduardo Xamena, María Elena Buemi, Julio Jacobo Berlles – Ciencia y Tecnología, Nº 20, 2020, pp. 23-36 ISSN 1850-0870 [Електронний ресурс] – Режим доступу: https://www.palermo.edu/ingenieria/pdf2020/CyT_20_03.pdf

17. MobileNets: Efficient Convolutional Neural Networks for Mobile VisionApplications / Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. & Adam, H. – 2017 [Електронний ресурс] – Режим доступу: https://arxiv.org/pdf/1704.04861.pdf
18. On-Device, Real-Time Hand Tracking with MediaPipe / V.Bazarevsky, F.Zhang – Google AI Blog, 2019 [Електронний ресурс] – Режим доступу: https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html
19. MediaPipe Holistic – Simultaneous Face, Hand and Pose Prediction, on Device/ I.Grishchenko, V.Bazarevsky – Google AI Blog, 2020 [Електронний ресурс] – Режим доступу: https://ai.googleblog.com/2020/12/mediapipe-holistic-simultaneous-face.html

Додатки

Додаток А – Лістинг програми для створення системи розпізнавання жестів з використанням одноступінчатого підходу

```
#!/usr/bin/env python
# coding: utf-8

# In[2]:
import cv2
import os
import time
import uuid


# In[3]:
IMAGES_PATH = 'Tensorflow/workspace/images/collectedimages'


# In[4]:
labels = ['thumbsup', 'thumbsdown', 'openpalm', 'fist', 'pointup', 'pointdown']
number_imgs = 16


# In[ ]:
label = 'pointdown'
get_ipython().system("mkdir {'Tensorflow\\workspace\\images\\collectedimages\\\\' + label}")
cap = cv2.VideoCapture(0)
print('Collecting images for {}'.format(label))
time.sleep(5)
for imagenum in range(number_imgs):
    ret, frame = cap.read()
    imgname = os.path.join(IMAGES_PATH, label, label+'.'+'{}.jpg'.format(str(uuid.uuid1())))
    cv2.imwrite(imgname, frame)
    cv2.imshow('frame', frame)
    time.sleep(1)
    print(imagenum)
    time.sleep(2)

    if (cv2.waitKey(1) and 0xFF == ord('q')):
        break
cap.release()


# In[ ]:
```

```
# # 0. Встановлення шляхів до папок
```

In[1]:

```
WORKSPACE_PATH = 'Tensorflow/workspace'
SCRIPTS_PATH = 'Tensorflow/scripts'
APIMODEL_PATH = 'Tensorflow/models'
ANNOTATION_PATH = WORKSPACE_PATH+'/annotations'
IMAGE_PATH = WORKSPACE_PATH+'/images'
MODEL_PATH = WORKSPACE_PATH+'/models'
PRETRAINED_MODEL_PATH = WORKSPACE_PATH+'/pre-trained-models'
CONFIG_PATH = MODEL_PATH+'/my_ssd_mobnet/pipeline.config'
CHECKPOINT_PATH = MODEL_PATH+'/my_ssd_mobnet/'
```

1. Визначення маркувань

In[3]:

```
labels = [{'name':'thumbsup', 'id':1},
        {'name':'thumbsdown', 'id':2},
        {'name':'openpalm', 'id':3},
        {'name':'fist', 'id':4},
        {'name':'pointup', 'id':5},
        {'name':'pointdown', 'id':6}]

with open(ANNOTATION_PATH + '\label_map.pbtxt', 'w') as f:
    for label in labels:
        f.write('item { \n')
        f.write('\tname:\'{}\'\n'.format(label['name']))
        f.write('\tid:{}\n'.format(label['id']))
        f.write('}\n')
```

2. Створення TF-records

In[4]:

```
get_ipython().system("python {SCRIPTS_PATH + '/generate_tfrecord.py'} -x {IMAGE_PATH + '/train'} -l
{ANNOTATION_PATH + '/label_map.pbtxt'} -o {ANNOTATION_PATH + '/train.record'}")
get_ipython().system("python {SCRIPTS_PATH + '/generate_tfrecord.py'} -x {IMAGE_PATH + '/test'} -l
{ANNOTATION_PATH + '/label_map.pbtxt'} -o {ANNOTATION_PATH + '/test.record'}")
```

3. Завантаження моделі з Tensorflow Model Zoo

In[4]:

```
get_ipython().system('cd Tensorflow && git clone https://github.com/tensorflow/models')


# In[6]:


#wget.download('http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz')
#!mv ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz {PRETRAINED_MODEL_PATH}
#!cd {PRETRAINED_MODEL_PATH} && tar -zxvf ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz


# # 4. Копіювання конфігурації моделі до папки де буде викон. тренування

# In[2]:


CUSTOM_MODEL_NAME = 'my_ssd_mobnet'


# In[16]:


get_ipython().system("mkdir {'Tensorflow\\workspace\\models\\\\'+CUSTOM_MODEL_NAME}")
get_ipython().system("cp {PRETRAINED_MODEL_PATH+'/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/pipeline.config'} {MODEL_PATH+'/'+CUSTOM_MODEL_NAME}")


# # 5. Оновлення конфігурації НМ

# In[5]:


import tensorflow as tf
from object_detection.utils import config_util
from object_detection.protos import pipeline_pb2
from google.protobuf import text_format


# In[4]:


CONFIG_PATH = MODEL_PATH+'/'+CUSTOM_MODEL_NAME+'/pipeline.config'


# In[5]:


config = config_util.get_configs_from_pipeline_file(CONFIG_PATH)


# In[11]:
```

config

```
# In[35]:
pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(CONFIG_PATH, "r") as
f:

    proto_str =
f.read()

    text_format.Merge(proto_str, pipeline_config)


# In[36]:
pipeline_config.model.ssd.num_classes = 6
pipeline_config.train_config.batch_size = 4
pipeline_config.train_config.fine_tune_checkpoint =
PRETRAINED_MODEL_PATH+'/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/checkpoint/ckpt-0'
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
pipeline_config.train_input_reader.label_map_path= ANNOTATION_PATH + '/label_map.pbtxt'
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] = [ANNOTATION_PATH + '/train.record']
pipeline_config.eval_input_reader[0].label_map_path = ANNOTATION_PATH + '/label_map.pbtxt'
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] = [ANNOTATION_PATH + '/test.record']


# In[37]:
config_text =
text_format.MessageToString(pipeline_config)

with tf.io.gfile.GFile(CONFIG_PATH, "wb") as
f:

    f.write(config_text)


# # 6. Команда для тренування

# In[6]:
print("""python {}/research/object_detection/model_main_tf2.py --model_dir={}/{} --
pipeline_config_path={}/{}/pipeline.config --num_train_steps=5000""".format(APIMODEL_PATH,
MODEL_PATH,CUSTOM_MODEL_NAME,MODEL_PATH,CUSTOM_MODEL_NAME))


# # 7. Завантаження моделі

# In[2]:
import os
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
```

```
# In[6]:
# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(CONFIG_PATH)
detection_model = model_builder.build(model_config=configs['model'], is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(CHECKPOINT_PATH, 'ckpt-6')).expect_partial()

@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections


# # 8. Тестування

# In[7]:
import cv2
import numpy as np


# In[8]:
category_index = label_map_util.create_category_index_from_labelmap(ANNOTATION_PATH+'/label_map.pbtxt')


# In[9]:
cap = cv2.VideoCapture(0)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))


# In[ ]:
while True:
    ret, frame = cap.read()
    image_np = np.array(frame)

    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
    detections = detect_fn(input_tensor)

    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections

    # detection_classes should be ints.
    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)
```

```
    label_id_offset = 1
    image_np_with_detections = image_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
                image_np_with_detections,
                detections['detection_boxes'],
                detections['detection_classes']+label_id_offset,
                detections['detection_scores'],
                category_index,
                use_normalized_coordinates=True,
                max_boxes_to_draw=1,
                min_score_thresh=.5,
                agnostic_mode=False)

    cv2.imshow('object detection',  cv2.resize(image_np_with_detections, (800, 600)))

    if (cv2.waitKey(1) and 0xFF == ord('q')):
        cap.release()
        break


# In[42]:
detections = detect_fn(input_tensor)


# In[67]:
from matplotlib import pyplot as plt


# # 9. Збереження моделі

# In[3]:
print("""python {}/research/object_detection/exporter_main_v2.py --input_type=image_tensor --
pipeline_config_path={}/{}/pipeline.config --trained_checkpoint_dir={} --
output_directory={}export""".format(APIMODEL_PATH, MODEL_PATH,
CUSTOM_MODEL_NAME,CHECKPOINT_PATH, CHECKPOINT_PATH))


# # 10. Конвертація в TFJS

# In[4]:
get_ipython().system('pip install tensorflowjs')


# In[5]:
get_ipython().system('pip install pytest-cov')


# In[6]:
get_ipython().system('pip install pytest-filter-subpackage')
```

```
# In[7]:
get_ipython().system('pip install pytest-cov')


# In[8]:
get_ipython().system('pip install tensorflowjs')


# In[9]:
"""tensorflowjs_converter --input_format=tf_saved_model --
output_node_names='detection_boxes,detection_classes,detection_features,detection_multiclass_scores,detection_score
s,num_detections,raw_detection_boxes,raw_detection_scores' --output_format=tfjs_graph_model --
signature_name=serving_default {}export/saved_model {}converted""".format(CHECKPOINT_PATH,
CHECKPOINT_PATH)

# In[ ]:
```

Додаток Б – Лістинг програми для створення системи розпізнавання жестів з використанням двоступінчатого підходу

```
#!/usr/bin/env python
# coding: utf-8

# # 1. Встановлення залежностей

# In[7]:


get_ipython().system('pip install tensorflow==2.8.0 tensorflow-gpu==2.8.0 opencv-python mediapipe sklearn
matplotlib')


# In[1]:


import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp


# # 2. Візуалізація ключових точок Медіапайп

# In[2]:
```

```
mp_holistic = mp.solutions.holistic
"""mp_hands = mp.solutions.hands # Hands model"""
mp_drawing = mp.solutions.drawing_utils


# In[3]:


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


# In[4]:


def draw_landmarks(image, results):
    """
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(image, hand_landmarks, mp_hands.HAND_CONNECTIONS) # Draw
hands connections
    """
    # mp_drawing.draw_landmarks(image, results.face_landmarks, mp_holistic.FACEMESH_TESSELATION) #
Draw face connections
    # mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS) # Draw
pose connections
    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



# In[5]:


def draw_styled_landmarks(image, results):
    """
    # Draw hands connections
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(image, hand_landmarks, mp_hands.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 face connections
    mp_drawing.draw_landmarks(image, results.face_landmarks, mp_holistic.FACEMESH_TESSELATION,
                              mp_drawing.DrawingSpec(color=(80,110,10), thickness=1, circle_radius=1),
                              mp_drawing.DrawingSpec(color=(80,256,121), thickness=1, circle_radius=1)
                              )


    # Draw pose connections
    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(80,22,10), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(80,44,121), thickness=2, circle_radius=2)
                              )
    """

    # 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)
                              )


# In[12]:


cap = cv2.VideoCapture(0)
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    """with mp_hands.Hands(min_detection_confidence=0.5, min_tracking_confidence=0.5) as hands:"""
    while cap.isOpened():

        ret, frame = cap.read()

        image, results = mediapipe_detection(frame, holistic)
        """image, results = mediapipe_detection(frame, hands)"""
        print(results)

        draw_styled_landmarks(image, results)

        cv2.imshow('OpenCV Feed', image)

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


# In[13]:
```

```
draw_landmarks(frame, results)


# In[14]:


plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))


# # 3. Функція вилучення потрібних ключових точок

# In[15]:


len(results.left_hand_landmarks.landmark)


# In[16]:


"""pose = []
for res in results.pose_landmarks.landmark:
    test = np.array([res.x, res.y, res.z, res.visibility])
    pose.append(test)
"""


# In[17]:


# pose = np.array([[res.x, res.y, res.z, res.visibility] for res in results.pose_landmarks.landmark]).flatten() if
results.pose_landmarks else np.zeros(132)
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)


# In[36]:


def extract_keypoints(results):
    #pose = np.array([[res.x, res.y, res.z, res.visibility] for res in results.pose_landmarks.landmark]).flatten() if
results.pose_landmarks else np.zeros(33*4)
    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])
```

```
# In[19]:


result_test = extract_keypoints(results)


# In[20]:


len(result_test)


# In[21]:


np.save('0', result_test)


# In[22]:


np.load('0.npy')


# # 4. Формування папок для набору даних

# In[23]:

DATA_PATH = os.path.join('Data')


# In[24]:


actions = np.array(['thumbs_up', 'thumbs_down', 'up', 'down', 'grab', 'release', 'no_action'])

no_sequences = 60

sequence_length = 22

keypoints_length = len(result_test)

start_folder = 60


# In[55]:


"""
for action in actions:
    dirmax = np.max(np.array(os.listdir(os.path.join(DATA_PATH, action))).astype(int))
    for sequence in range(1,no_sequences+1):
```

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


# # 5. Збирання даних для тренування і тестування

# In[57]:


cap = cv2.VideoCapture(0)
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:

    for action in ['up', 'down']:
        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, 0, 255), 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, 0, 255), 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()


# In[158]:


cap.release()
cv2.destroyAllWindows()


# # 5.1 Доповнення набору даних

# In[20]:


# створення папок
for action in actions:
    dirmax = np.max(np.array(os.listdir(os.path.join(DATA_PATH, action))).astype(int))
    for sequence in range(1,no_sequences+1):
        try:
            os.makedirs(os.path.join(DATA_PATH, action, str(dirmax+sequence)))
        except:
            pass


# In[21]:


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(start_folder, start_folder+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, 0, 255), 1, cv2.LINE_AA)
                    cv2.imshow('OpenCV Feed', image)
                    cv2.waitKey(1800)
```

```
            else:
                cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 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()


# # 6. Попередня обробка даних та маркування

# In[25]:


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


# In[26]:


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


# In[27]:


label_map


# In[30]:


sequences, labels = [], []
for action in actions:
    for sequence in range(no_sequences):
        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])


# In[31]:
```

```
np.array(sequences).shape

# In[32]:

np.array(labels).shape

# In[33]:

X = np.array(sequences)

# In[34]:

X.shape

# In[35]:

y = to_categorical(labels).astype(int)

# In[36]:

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)

# In[37]:

y_test.shape

# # 7. Побудова і тренування LSTM мережі

# In[28]:

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import TensorBoard

# In[29]:
```

```
log_dir = os.path.join('Logs')
tb_callback = TensorBoard(log_dir=log_dir)


# In[30]:


model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu',
               input_shape=(sequence_length,keypoints_length)))
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'))


# In[31]:


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


# In[72]:


model.fit(X_train, y_train, epochs=250, callbacks=[tb_callback])


# In[73]:


model.summary()


# # 7.1 Донавчання

# In[45]:


model.load_weights('action.h5')


# In[46]:


model.fit(X_train, y_train, epochs=250, callbacks=[tb_callback])


# In[47]:
```

```
model.save('action2.h5')

# In[32]:

model.load_weights('action2.h5')

# # 8. Передбачення (тест)
# In[49]:

res = model.predict(X_test)

# In[50]:

np.argmax(res[4])

# In[51]:

actions[np.argmax(res[4])]

# In[52]:

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

# # 9. Збереження вагових коефіцієнтів
# In[78]:

model.save('action.h5')

# In[79]:

"""del model"""

# In[20]:

model.load_weights('action.h5')
```

```
# # 10. Оцінка моделі з Confusion Matrix

# In[53]:


from sklearn.metrics import multilabel_confusion_matrix, accuracy_score


# In[54]:


yhat = model.predict(X_test)


# In[55]:
ytrue = np.argmax(y_test, axis=1).tolist()
yhat = np.argmax(yhat, axis=1).tolist()


# In[56]:
multilabel_confusion_matrix(ytrue, yhat)


# In[57]:
accuracy_score(ytrue, yhat)


# # 11. Тестування

# In[33]:
from scipy import stats


# In[34]:
colors = [(245,117,16), (117,245,16), (16,117,245), (16,245,117), (117,245,16), (117,245,16), (117,245,16)]
def prob_viz(res, actions, input_frame, colors):
    output_frame = input_frame.copy()
    for num, prob in enumerate(res):
        cv2.rectangle(output_frame, (0,60+num*40), (int(prob*100), 90+num*40), colors[num], -1)
        cv2.putText(output_frame, actions[num], (0, 85+num*40), cv2.FONT_HERSHEY_SIMPLEX, 1,
(255,255,255), 2, cv2.LINE_AA)
    print(res)
    return output_frame


# In[35]:


plt.figure(figsize=(18,18))
#plt.imshow(prob_viz(res, actions, image, colors))
```

```
# In[37]:


sequence = []
sentence = []
predictions = []
threshold = 0.85

cap = cv2.VideoCapture(0)
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    while cap.isOpened():

        ret, frame = cap.read()

        image, results = mediapipe_detection(frame, holistic)
        print(results)

        draw_styled_landmarks(image, results)

        keypoints = extract_keypoints(results)
        sequence.append(keypoints)
        sequence = sequence[-sequence_length:]

        if len(sequence) == sequence_length:
            res = model.predict(np.expand_dims(sequence, axis=0))[0]
            print(actions[np.argmax(res)])
            predictions.append(np.argmax(res))


            if np.unique(predictions[-10:])[0]==np.argmax(res):
                if res[np.argmax(res)] > threshold:

                    if len(sentence) > 0:
                        if actions[np.argmax(res)] != sentence[-1]:
                            sentence.append(actions[np.argmax(res)])
                    else:
                        sentence.append(actions[np.argmax(res)])

            if len(sentence) > 3:
                sentence = sentence[-3:]

            image = prob_viz(res, actions, image, colors)

        cv2.rectangle(image, (0,0), (640, 40), (245, 117, 16), -1)
        cv2.putText(image, ' '.join(sentence), (3,30),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

        cv2.imshow('OpenCV Feed', image)
```

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

11. Конвертація моделі в JavaScript

In[91]:

```
get_ipython().system('pip install tensorflowjs # install converter')
```

In[ ]:

```
tensorflowjs_converter --input_format keras action.h5 js
```

In[93]:

```
newar = np.expand_dims([[1,2,3],[2,3,4],[3,4,5]], axis=0)
```

In[94]:

```
newar.shape
```

In[99]:

```
res0 = model.predict(np.expand_dims(sequence, axis=0))[0]
```

In[100]:

```
res0
```

In[ ]: