Міністерство освіти і науки України

Національний університет «Києво-Могилянська Академія»

Кафедра мультимедійних систем

КУРСОВА РОБОТА

за спеціальністю «Комп’ютерні науки» 122

на тему: розробка мобільного додатку під ОС Android з використанням комп’ютерного зору

Науковий керівник:

ст. викладач Борозенний С. О.

Виконав:

студент 4-го курсу Підлісний М. В.

Київ – 2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних технологій факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем

доцент, к.ф.-м.н

_____О.П. Жежерун

„____”_____2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту _____ факультету _____ ____4-го_____курсу

ТЕМА _____

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

1 Аналіз предметної області

2 Механізм розпізнавання позиції людини

3 Механізм класифікації позицій

4 Розробка з камерою в Android

5 Розробка програмного прототипу

Висновки

Список літератури

Додатки (за необхідністю)

Дата видачі „___” _____ 2022 р. Керівник _____ __

Завдання отримав _____

Календарний план виконання роботи

Тема: Розробка мобільного додатку на основі ОС Android

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу.	02.11.2021	
2.	Огляд технічної літератури за темою роботи.	15.11.2021	
3.	Аналіз використання комп’ютерного зору в мобільних додатках	25.11.2021	
3.	Проектування архітектури додатку	Грудень 2021	
4.	Програмування додатку	Січень - березень 2022	
5.	Написання текстової частини роботи.	Квітень 2022	

Студент _____

Керівник _____

“_____”_____

ЗМІСТ

Анотація

Курсова робота присвячена дослідженню використання методів комп’ютерного зору та машинного навчання в мобільних застосунках на операційній системні Android з використанням сучасних стилів, бібліотек та технологій. Особливо увагу приділено на задачі розпізнавання пози людини та її класифікації. Отриманим результатом роботи є удосконалений фітнес-додаток з функцією «розумного тренера», що дозволяє розпізнати та лічити кількість правильно виконаних фізичних вправ користувача.

Ключові слова: Android, Kotlin, ML Kit, pose estimation, , pose classification, фітнес додаток, комп’ютерний зір, машинне навчання.

Вступ

На сьогоднішній день буде складно знайти людину, яка б не чула про феномен штучного інтелекту та наскільки зараз поширене використання його можливостей. Одним з найпотужніших та привабливих видів ШІ є комп'ютерний зір.

Комп'ютерний зір — це галузь інформатики, яка зосереджується на відтворенні складних частин системи людського зору та наданні комп'ютерам можливості ідентифікувати й обробляти об'єкти на зображеннях та відео так само, як це робимо ми, люди.

Завдяки досягненням у галузі штучного інтелекту та інноваціям у навчанні з використовуванням великих об'ємів даних та нейронних мережах, ця галузь змогла зробити великі стрибки за останні роки та змогла перевершити людей у деяких завданнях, пов'язаних із виявленням та маркуванням об'єктів.

Донедавна комп'ютерний зір працював лише в обмежених можливостях та вимагав відповідного програмного та апаратного забезпечення, проте ситуація на сьогодні значно змінилася. У теперішній час для використання переваг комп'ютерного зору буде достатньою невеликого комп'ютера, який сьогодні є майже у кожної людини в кишені – сучасного мобільного телефона. Разом зі світом технологій розриваються й інші галузі життєдіяльності людини адаптуючи нові підходи та можливості. Беручи до уваги становище подій у сучасному світі люди почали більше проводи часу вдома, а ніж за межами його, й це стосується не тільки роботи чи навчання, а й такого хобі, як зайняття спортом. З'явилося безліч сервісів для зайняття спортом, таких як персональний підбір вправ, записи інструкції й їх виконання та навіть можливість дистанційного зв'язку з персональним тренером для відстеження вірного виконання вправ. Однак останній сервіс має безліч своїх недоліків,

починаючи зі складності оцінки тренера якості виконання вправ лише по відео, закінчуючи потребою одночасного підключення тренера та клієнта до мережі. Для вирішення цих проблем чудовим вибором стане використання можливостей комп’ютерного зору та машинного навчання, для оцінки якості вправ користувача та їх підрахунок.

Мета цієї роботи полягає в тому, щоб з’ясувати та ознайомитися з механізмами комп’ютерного зору та машинного навчання, а саме розпізнаванням людини та її поз, класифікацією їх та підрахунком вірних повторень в реальному часі за допомогою камери телефону.

У результаті виконання дослідження створено мобільний застосунок з функцією “віртуального тренера”, для класифікації та підрахунку вірних повторень вправ користувача.

1. Аналіз предметної області

1.1 Характеристика предметної області

За останні кілька років мобільні пристрої, програми та носимі технології змінили методи фітнесу та слідкуванням зв своїм здоров'ям. Ця галузь стає все більш популярною, і людям цікаво дізнатися більше про власне здоров'я та рівень фізичної підготовки, не залежно від лікаря чи тренера для вимірювання їхньої ефективності. Поширеність додатків для здоров'я та фітнесу пов'язана з поєднанням зростання індустрії носимих технологій та еволюції руху здоров'я. Як наслідок, споживачі все більше покладаються на мобільні додатки, щоб допомогти їм керувати своїм здоров'ям та фізичними можливостями.

Тренування стали високотехнологічними. Не так давно, якщо людина вирушила на пробіжку і захотіла дізнатися, яку відстань вона пройшла, вона б дивилися на відстань на карті або відстежували її за допомогою автомобільної GPS-системи. Сьогодні буде достатнього одного телефона з відповідним фітнес-додатком. На сьогодні частка фітнес додатків серед усіх додатків на маркеті Play Market складає цілих дев'ять відсотків й прогнозується, що ця цифра буде тільки зростати й збільшиться на 17.6 відсотків з 2022 року до 2030[1]. Разом з популярністю мобільних додатків буде й розвиваються симбіоз зі світом сучасних технологій. Вже зараз, використовуючи сучасні методи машинного навчання та комп'ютерного зору, технології для здоров'я та фітнесу більш персоналізовані в оцінці нашого тіла та тренувань.

Таким чином, основна мета цією курсової робити – це познайомитися з технологією комп'ютерного зору в межах світу мобільної розробки. Більш детально, було розглянуто задачі класифікації та визначення об'єкта серед інших на реальному прикладі та застосовано в прототипі мобільного застосунку. Розроблений функціонал додатку представляє собою функцію

«віртуального тренера», тобто можливість займатися спортом та відслідковувати правильність виконання вправ без потреби у персональному тренері. Додаток за допомогою камери телефону зчитує рухи користувача, класифікує його позицію та вправу, яку користувач виконує. Якщо вправа була виконана вірно, то користувача повідомляється звуковим та візуальним сигналом, додаючи повторення до лічильника.

1.2 Порівняння аналогів

Для повноти дослідження використання технологій комп'ютерного зору та машинного навчання в мобільних додатках також був проведений детальний аналіз відповідних сучасних застосунків на маркеті Play Market. Так як основна тема розробленого додатку спорт та здоров'я, то й відповідно на додатки з цієї категорії приділялася найбільша увага.

1) InfiGro

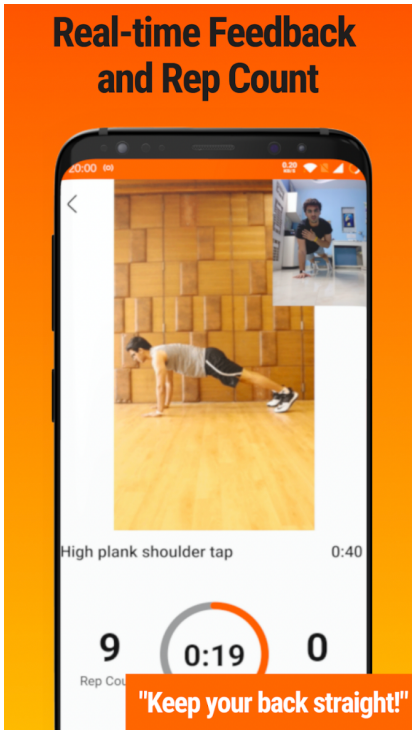

Рисунок 1.2.1 – Додаток InfiGro

InfiGro — це повністю автоматизоване програмне забезпечення цифрового персонального тренера на основі штучного інтелекту, яке використовує камеру телефону, щоб навчати, оцінювати, виправляти та надихати користувача у режимі реального часу (рисунок 1.2.1). InfiGro від Infivolve перетворює телефон у фітнес-тренажер. Щоб підтримувати своє психічне здоров'я у формі, користувач також може перевірити своє харчування та скористатися заготовленими техніками для медитації. Основні переваги та можливості додатка це - персональний помічник з фітнес-тренувань 24/7,

найкращі тренування та програми, створені досвідченими тренерами по всьому світу, відстеження вправ у реальному часі лише за допомогою камери смартфона користувача, звуковий відгук про техніку виконання вправ та продуктивність користувача, цілеспрямовані плани тренувань які забезпечують досягнення користувачами своїх цілей.

2) Gymfitty

Рисунок 1.2.2 – Додаток Gymfitty

Gymfitty — це ще один додаток-персональний тренер на базі штучного інтелекту, який використовує передові алгоритми для аналізу кожного користувача та його програми тренувань, винаходячи й адаптуючи вправи за потреби, так само, як і справжній тренер в спортивному залі(рисунок 1.2.2). Gymfitty пропонує інтелектуальні вправи, засновані на продуктивності корисутувача, а не на попередньо записаних програмах. Пристосовуючи тренування до точних можливостей користувача, додаток стає персоналізованішим щоразу, коли він або вона його використовує. Разом з тим додаток використовує сучасні дослідження та дані на кожному етапі тренування, щоб відстежувати те та як, робить користувач, щохвилини

коригуючи, щоб переконатися, що він або вона досягне своїх цілей у фітнесі. Крім того, він працює в автономному режимі, тому, коли користувач перебуває в місці з поганим зв'язком, йому не доведеться турбуватися про швидкість з|єднання чи можливі дефекти у роботі.

3) FitnessAI

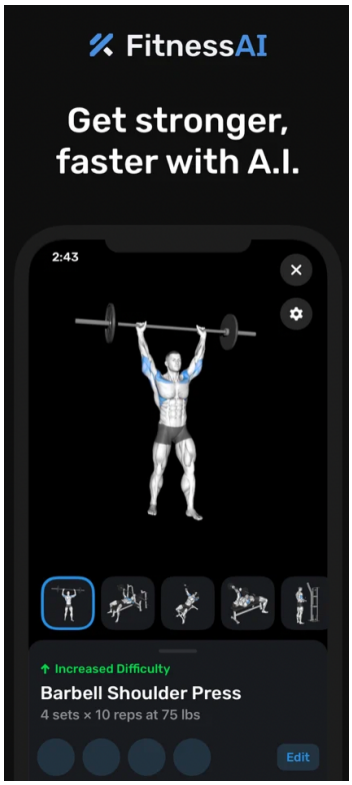

Рисунок 1.2.3 – Додаток FitnessAI

FitnessAI — це додаток, який використовує штучний інтелект для розробки персоналізованих програм тренувань для кожного користувача (рисунок 1.2.3). Ця програма починалася як додаток лише для тренажерного залу, але нещодавно він змінив швидкість, щоб задовольнити зростаючий попит на фізичні вправи вдома. Просто, щоразу, коли користувач тренується, FitnessAI оптимізує підйоми тягарів, повторення та ваги для кожного виду діяльності, підштовхуючи їх до ефективного зростання м'язів. Іншими словами, приклад цього додатка демонструє, як машинне навчання можна використовувати для створення графіка фітнесу. За словами продавця, алгоритм цього програмного забезпечення був навчений на 5,9 мільйона вправ. За три роки було зібрано 10 мільйонів підходів, повторень і ваг від понад 30 тисяч експертів важкоатлетів

і відвідувачів тренажерного залу. Вважається, що додаток містить одну з найбільших у світі баз даних з важкої атлетики і обіцяє перевершити будь-якого персонального тренера людини. Крім цього з переваг додатку це підтримка «розумних» годинників, які розширюють відстеження показників користувача та в цілому функціонал додатку.

1.3 Функціональні вимоги

Після аналізу задачі та мети дослідження для розробки застосунку було створено нижченаведений перелік функціональних вимог:

a) Можливість легкого вивантаження текстових ресурсів додатку для подальшого перекладу
b) Локалізація даних додатку
c) Адаптація екранів під різні розміри телефонів та планшетів
d) Підтримка вертикальної та горизонтальної орієнтації екрану
e) Наявність кешування даних в пам’яті пристрою
f) Можливість доступу до функціоналу додатку без постійного або зі слабким підключенням до інтернету
g) Можливість розпізнавання людини з зображення та відео
h) Можливість розпізнавання рухів людини
i) Можливість розпізнавання ключових точок тіла людини для створення візуального відстеження
j) Класифікація позиції людини
k) Перевірка правильного виконання вправи
l) Підрахунок вірних повторень вправи
m) Візуальне відтворення кількості вірних повторень вправи
n) Можливість використовувати як основну, так і фронтальну камеру для функції «віртуального тренера»

2. Механізм розпізнавання позицій людини

2.1 Загальні відомості

Оцінка та відстеження пози людини — це завдання комп'ютерного зору, яке включає виявлення, асоціацію та відстеження семантичних ключових точок тіла людини. Прикладами семантичних ключових точок є «праве плече», «ліве коліно» або інше.

Визначення пози людини з зображення та відео відіграє важливу роль у різних програмах, таких як кількісна оцінка фізичних вправ, розпізнавання мови жестів та керування спеціальними можливостями за допомогою жестів всього тіла. Для прикладу, механізм розпізнавання позиції людини може стати основою для застосування додатку для фітнесу, танців і йоги. Він також може дозволити використання цифрового вмісту та інформації поверх фізичного світу в доповненій та віртуальній реальності. Оцінка пози для фітнес-застосунків є особливо складною через широкий спектр можливих поз (наприклад, сотні асан йоги), численні ступені свободи, оклюзії (наприклад, тіло або інші об'єкти перекривають кінцівки, як видно з камери) та й звичайно різноманітність зовнішності, одягу, не кажучи вже про додаткові унікальні аксесуари та прикраси.

Разом з тим усі підходи до оцінки пози можна згрупувати у два основні види: «знизу вгору» та «зверху вниз». Методи знизу вгору спочатку оцінюють кожен суглоб тіла, а потім групують їх, щоб сформувати унікальну позу. Методи зверху вниз спочатку запускають детектор людини і оцінюють суглоби тіла в межах виявлених обмежувальних рамок.

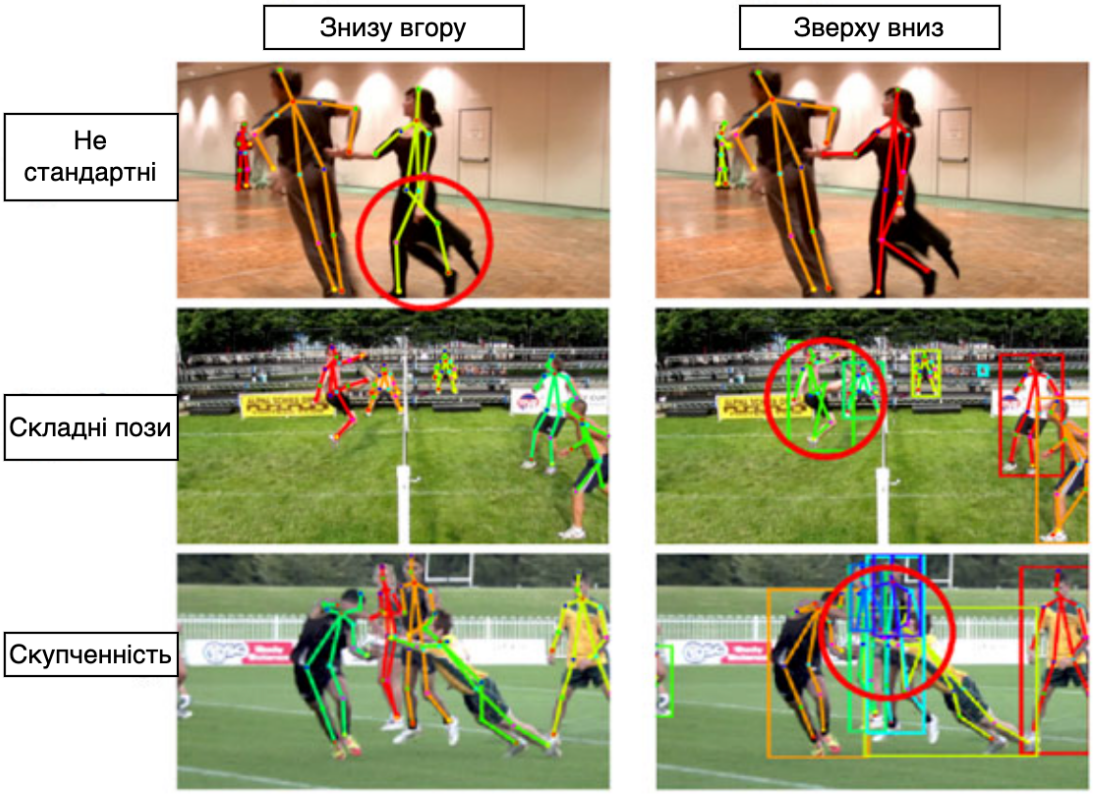

Рисунок 2.1.1 – Порівняння методів знизу вгору та зверху вниз

На рисунку 2.1.1 наведене порівняння цих двох методів в трьох різних ситуаціях й додатково варто зазначити, що метод зверху вниз в більшості ситуацій витрачає більше часу чим метод знизу вгору. Це пояснюється тим, що для вирішення задачі методом зверху вниз потрібно детектор визначення позиції на кожен результат детектора людини.

2.2 ML Kit Pose Detection API

ML Kit Pose Detection API — це легке універсальне рішення для розробників додатків, яке дозволяє виявляти позу тіла людини в режимі реального часу за допомогою безперервного відео або статичного зображення. Поза описує положення тіла в певний момент часу з набором орієнтирів скелета. Орієнтири відповідають різним частинам тіла, таким як плечі та стегна. Відносне розташування орієнтирів можна використовувати, щоб відрізнити одну позу від іншої.

ML Kit Pose Detection API використовує стандарт BlazePose, який в свою чергу використовує метод зверху вниз для оцінки позиції людини та базується на топології COCO[2]. Тобто використовує додатковий детектор обличчя людини й має наступну структуру, зображену на рисунку 2.2.1.

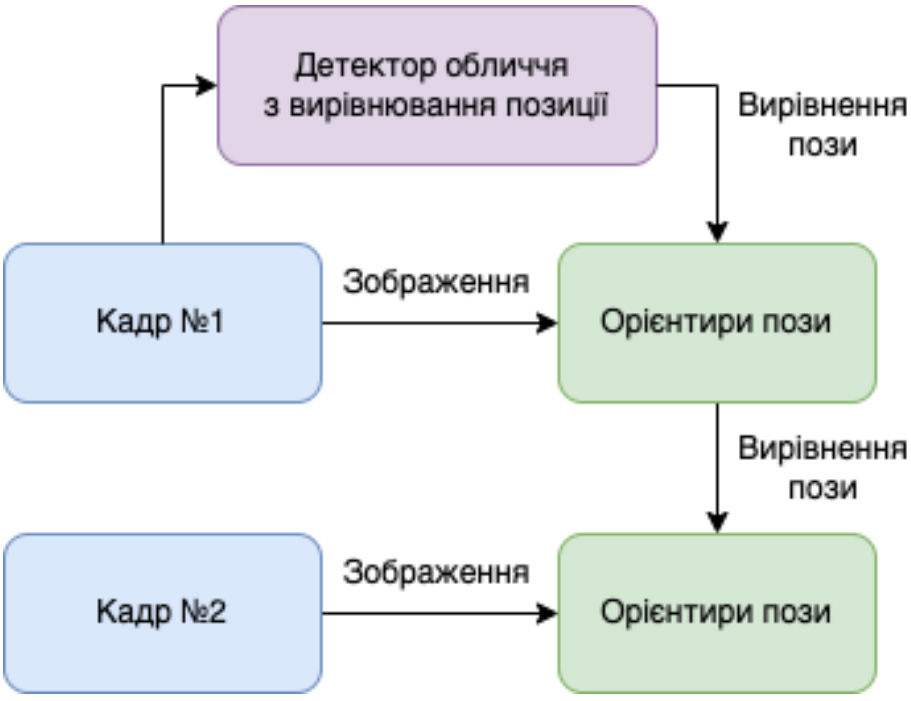

Рисунок 2.2.1 – Структура стандарту BlazePose

Детектор прогнозує координати ключової точки, присутність людини на поточному кадрі та більш уточнену область кадру, яка по суті виступає обрізом зображення. В момент, коли детектор вказує, що людини немає

запускається повторно мережа детектора на наступному кадрі та запускається перевірка по новому.

ML Kit Pose Detection створює 33-точковий скелет для всього тіла, який включає орієнтири на обличчі (вуха, очі, рот і ніс) і точки на руках і ногах. На рисунку 2.2.2 нижче показано орієнтири дзеркального відображення користувача.

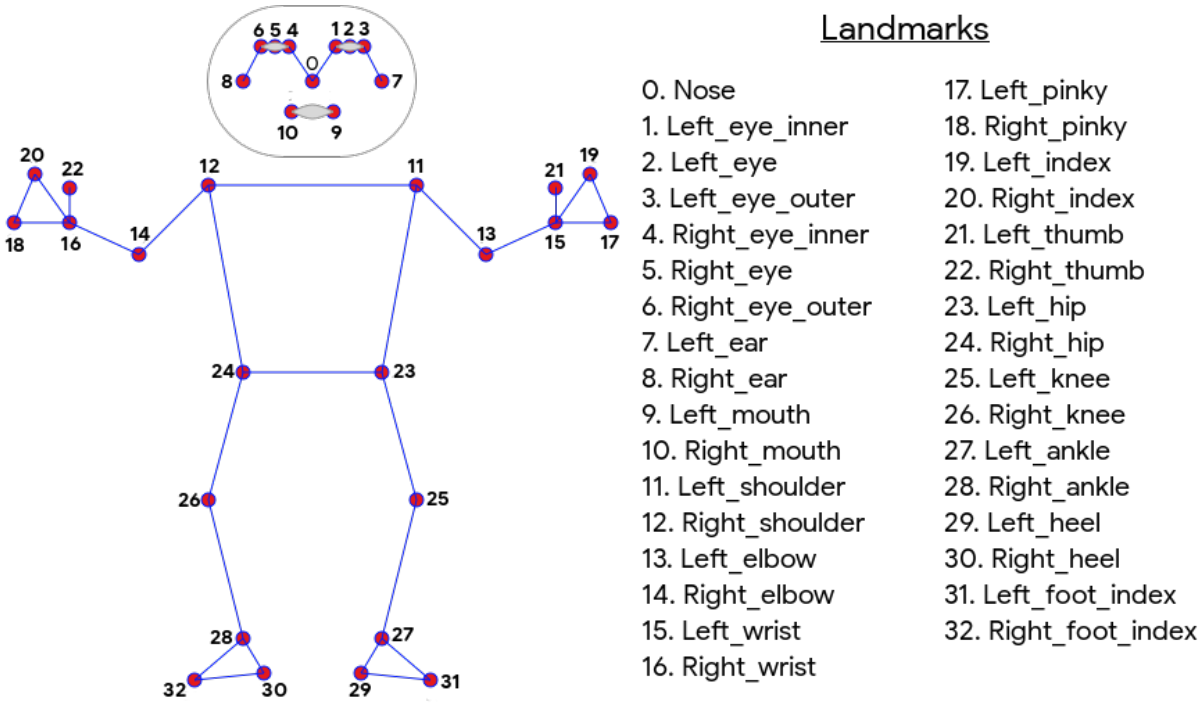

Рисунок 2.2.2 – Орієнтири пози людини

Загальноприйнятий підхід полягає в створенні теплових карт[3] для кожного з’єднання суглобів з уточнюючими зміщеннями для кожної координати. Разом з тим в той час коли цей вибір теплових карт масштабується для кількох людей з мінімальними накладними витратами, для однієї людини цей підхід робить модель значно більшою, ніж прийнятною для мобільних смартфонів ще й тим більше, в реальному часі. Саме тоді приходить на допомогу регресійний підхід, який є менш вимогливий до складних обчислень та є кращим для масштабованості. Таким чином Алехандро Ньюелл у своєму докладі «Stacked hourglass networks for human pose estimation» на конференції з комп’ютерного зору довів[4], що архітектура «Stacked hourglass» дає значне підвищення якості передбачення, навіть з меншою кількістю параметрів й саме тому вона була

використана як архітектура мережі для прогнозування теплових карт для всіх з’єднань, в якій детектор регресує безпосередньо до координат усіх з’єднань. Бо основана перевага цього підходу – це що не лише знаходимо найбільше значення в матриці, бо потрібен більш точний підхід, за допомогою цього підходу після пошуку максимального елементу поряд з ним шукають наступного по розміру сусіда та при цьому трохи рухають прогнозування в його бік. Проте ключове розуміння цього підходу полягає в тому що гілку теплової карти можна відкинути під час результату, що робить його достатньо легким для роботи на мобільному телефоні.

3. Механізм класифікації позицій

3.1 Загальні відомості

Класифікація — це визначений процес категоризації певного набору даних на відповідні йому класи. Важливо зазначити, що цей процес можна виконувати як для структурованих, так і для неструктурованих даних. Класи часто називають цільовими, мітками або категоріями. Він починається з прогнозування класу заданих точок даних. Прогностичне моделювання класифікації є завданням апроксимації функції відображення від вхідних змінних до дискретних вихідних змінних. Основна мета – визначити, до якого класу чи категорії потраплять нові дані. Так, наприклад, можна класифікувати новий лист на електронній пошті до категорії спам, чи до важливі.

Розрізняють три основні типи задач класифікації: бінарна класифікація, багато класова класифікація, класифікація з багатьма мітками. Бінарна класифікація – це тип класифікації з двома результатами, наприклад, істинним або хибним. Тобто існує лише два стани: належить класу чи ні. Яскравим прикладом алгоритму машинного навчання, який може використовуватися для бінарної класифікації є логістична регресія[5]. Багато класова класифікація – це тип класифікації з можливістю мати клас з двох або більше класів в процесі виконання, проте завжди один й тільки один в результаті. Використання цього типу допомагає у визначенні виду рослини по фотографії. Одним з популярних алгоритмів, який використовують для цього типу класифікації є метод к-найближчих сусідів. Тип класифікації з багатьма мітками відрізняється від типу багато класової класифікації тим, що кожен об’єкт класифікації може належати декільком класам одночасно. Цей тип часто використовується для класифікації багатьох об’єктів по фото, де потрібно точно визначити до якого класу належить той чи інший об’єкт. Прикладом слугує алгоритм дерева рішень з кількома мітками[6].

3.2 Процес класифікації

Визначення пози людини з зображення є задачею багато класової класифікації, бо відповідно до прикладу з рослинами, наведеним вище, кожна позиція може бути класифікована лише під один клас. Відповідно до типу класифікації було обрано алгоритм к-найближчих сусідів для перетворення вхідних даних до визначеної позиції.

Алгоритм к-найближчих сусідів - це непараметричний алгоритм машинного навчання, який використовує близькість для класифікації або прогнозів щодо групування окремої групи даних. Тобто алгоритм визначає клас об’єкта на основі найближчих зразків у навчальному наборі даних. Процес формування навчального набору даних є одним з найважливіших критеріїв точності цього алгоритму. У рамках задачі класифікації позиції для створення навчального набору даних потрібно зібрати зразки зображень під різними кутами та в кожному кінцевому стані вправи, так для прикладу з присіданнями це положення присівши та стоячи. Разом з тим, потрібно визначити орієнтири людини на зображеннях позицій. Після отримання орієнтирів, їх необхідно перетворити на формат придатний для алгоритму, а саме в вигляді вектора даних, для кожного зразка. Крім того також для роботи алгоритму к-найближчих сусідів потрібно визначити метрику обчислення найменшої відстані між двома векторами для знаходження найближчого об’єкту до тренувального зразка. Щоб перетворити орієнтири пози у вектор даних, ми використовуємо попарні відстані між попередньо визначеними списками суглобів поз (Рисунок 3.2.1), наприклад відстані між плечем і суглобом, гомілковостопним і стегновим суглобом, а також двома зап’ястями. Оскільки алгоритм спирається на відстані, усі пози нормалізуються, щоб мати однаковий розмір тулуба та вертикальну орієнтацію тулуба перед перетворенням.

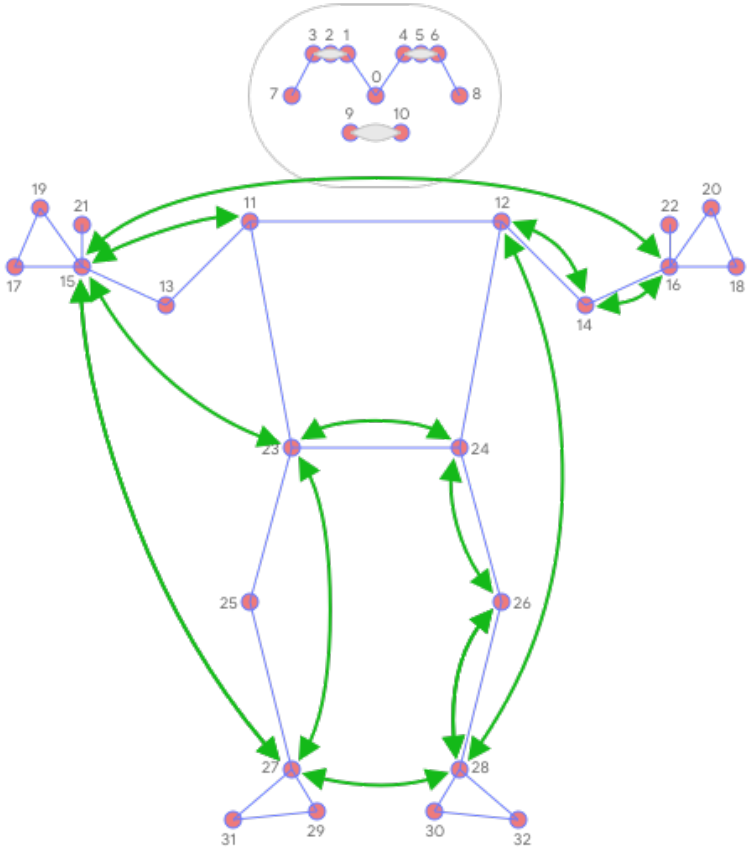

Рисунок 3.2.1 – Попарні відстані для вектора даних

Пошук к-найближчих сусідів застосовується два рази з різними показниками відстані та в кінці застосовується механізм згладжування, щоб отримати кращий результат знаходження класу до кожного елементу,. Отже, щоб відфільтрувати вибірки, які майже збігаються з цільовим, але мають лише кілька різних значень у векторі даних, мінімальна відстань за координатами використовується як метрика відстані. Тоді середня відстань за координатами використовується, щоб знайти найближчий кластер пози серед тих, що були з першого пошуку. Й зрештою, застосовується процес згладжування експоненційної ковзної середньої (EMA)[7], щоб нівелювати будь-який шум від передбачення пози або класифікації. Для цього проводиться пошук не тільки найближчого кластеру поз, а й обчислюється ймовірність для кожного з них, яка буде використовуватися для згладжування надалі.

3.3 Підрахунок повторів вправ

Для впровадження функції підрахунку повторів, застосовано алгоритм, який відстежує ймовірність кожного класу для пози. Для прикладу розглянемо присідання з термінальними станами «стоячи» і «присівши»:

- Коли ймовірність класу пози «присівши» вперше перевищує певний поріг, алгоритм позначає, що введено клас пози «присівши».
- Як тільки ймовірність падає нижче порогового значення, алгоритм позначає, що клас пози «присівши» вийшов, і збільшує лічильник.

Разом з тим для уникнення таких випадків, коли ймовірність коливається навколо порогового значення (наприклад, коли користувач робить паузу між станами «стоячи» і «присівши»), що спричиняє підрахунок, так званих фантомів, поріг, який використовується для виявлення виходу зі стану, насправді трохи нижчий, ніж той, який використовується для визначення, коли введено стан. За рахунок цього й створюється інтервал, коли лічильник й особливо клас пози («стоячи» чи «присівши») не можна змінити.

4. Робота з камерою в Android

4.1 Загальні відомості та обґрунтування використання

Застосування камери смартфона є невід'ємною частиною функціонала багатьох Android застосунків. Хоч і кожен смартфон має системний додаток камери для знімків зображень та відео й є рекомендованим для використання як окремо так і в межах інших додатків, проте зовсім часто його стає недостатньо для більш ширшого функціоналу або тим більше, якщо потрібно додати щось не типове поверх звичайних знімків.

В зв'язку з поширеною потребою імплементації особливого функціоналу поверх камери наразі представлено три основні бібліотеки для досягнення цих цілей: CameraX, Camera2, Camera. Разом з тим, варто зазначити, що бібліотека Camera на момент написання курсової роботи є застарілою та не рекомендованою до використання в нових додатках, віддавши перевагу CameraX чи Camera2.

Бібліотека Camera2 є більш розширеною ніж CameraX та надає всі можливі детальні засоби для керування у складних випадках, хоча й в той ж час є більш складною та вимагає керування багатьма окремо визначеним конфігураціями й в тому числі специфічними під різні смартфони. При цьому бібліотека CameraX підтримує переважну більшість пристроїв Android (Android 5.0 і вище) і забезпечує послідовний високорівневий інтерфейс, розроблений для основних випадків використання. Додатково CameraX включена в набір бібліотек Jetpack, які виступає головним набором стандартизації підходів у розробка мобільних додатків на платформа Android.

У результаті, розглянувши всі переваги бібліотек та врахувавши специфіку потреби використанні камери в визначенні та класифікації позиції людини було обрано CameraX, як основної бібліотеки для взаємодії з камерою.

4.2 Застосування CameraX

Бібліотека CameraX розбита на чотири частини та пропонує наступний функціонал:

- Зйомка зображення (знімання та зберігання фотографії)
- Зйомка та захоплення відео (дозволяє знімайте відео та аудіо)
- Попередній перегляд (можливість попередньо переглядати зображення камери в реальному часі без потреби його збереження)
- Аналіз зображення (набір методів аналізу зображення)

В додатку використано механізми попереднього перегляду для зображення користувача та аналіз зображення для коригування даних для відображення скелета орієнтирів за допомогою унікального елементу інтерфейсу.

Для роботи механізму попереднього перегляду необхідно додати до розмітки екрану додатковий системний елемент PreviewView та прив’язати його до життєвого циклу додатку для коректного відображення використовуючи шаблон Builder, як наведено на рисунку 4.2.1

```
fun bindPreview(cameraProvider : ProcessCameraProvider) {
    var preview : Preview = Preview.Builder()
            .build()

    var cameraSelector : CameraSelector = CameraSelector.Builder()
          .requireLensFacing(CameraSelector.LENS_FACING_BACK)
          .build()

    preview.setSurfaceProvider(previewView.getSurfaceProvider())

    var camera = cameraProvider.bindToLifecycle(this as LifecycleOwner, cameraSelector, preview)
}
```

Рисунок 4.2.1 – Прив’язка до Preview до життєвого цикла додатку

Разом з тим, механізм аналізу зображення вимагає створення окремої змінної типу ImageAnalysis, яка надає дані конкретного зображення та підтримує можливість додавання унікального класу-аналізатора.

5. Розробка програмного прототипу

5.1 Основні засоби розробки

Для інтеграції механізму визначення та класифікування поз людини було використано наступні засоби розробки:

- Android Studio – програмне середовище для розробки додатків на операційній системі Android
- Kotlin – основна мова програмування, яка використовувалася для написання програмного коду додатку
- Coroutines – бібліотека для імплементації асинхронних та багатопоточних операцій
- Room – бібліотека для організації та взаємодії з внутрішньою реляційною базою даних
- Glide – бібліотека для оптимізованого завантаження та кешування зображень
- Dagger, Hilt – бібліотеки для впровадження механізму ін’єкції залежностей
- EasyPermissions – бібліотека для організації роботи з дозволами додатку
- Guava – бібліотека з особливим набором колекції типів та підручних засобів для роботи з великими об’ємами даних
- CameraX – бібліотека для взаємодії з камерою телефона в додатку
- ML Kit Pose Detection – бібліотека для визначення основних орієнтирів пози людини

5.2 Структура додатку

Передусім проект містить три основні модулі: app, buildSrc, Gradle Scripts та складається з основних 19 директорій та 63 Kotlin та Java файлів й має наступну структуру зображену на рисунку 5.2.1.

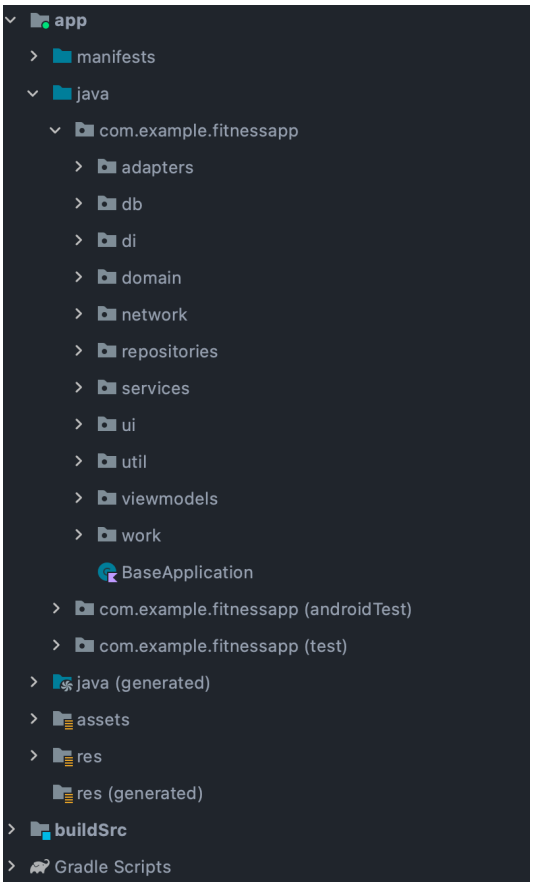

Рисунок 5.2.1 – Структура проекту в Android Studio

Основний модуль app – є модулем для розміщення програмного коду додатку та завжди містить такі директорі як “manifest” та “java”. В директорії manifest розмішений особливий конфігураційний файл у форматі .xml, який необхідний для задання поведінки додатку з системою Android та навпаки. В ньому зазначаються такі механізми як:

- Дозволи, необхідні програмі для доступу до захищених частин системи або інших програм. Він також оголошує будь-які дозволи, які повинні мати інші програми, якщо вони хочуть отримати доступ до вмісту поточного додатку.

- Конфігурації чотирьох основних компонентів[9] застосунку: Activities, Services, Broadcast Receivers, Content Providers.

В той час директорія java містить в собі всі Kotlin та Java файли в яких описується поведінка та основні механізми застосунку. Всі програмні файли структуровані за методами чистої архітектури, з розбиттям на шари даних, клієнтського інтерфейсу та домену. Разом з тим робота з представленням даних та архітектура додатку побудована враховуючи останні прийняті інструкції[8] компанією Google для побудови сучасного застосунку. Додатково варто зазначити, що структура коду додатку була побудована використовуючи зокрема такі патерни проектування[10] як Репозиторії, Адаптер, Одинак, Спостерігач та інші.

Модуль Gradle Scripts містить в собі конфігураційні файли проекту, які власне потрібні для побудови додатку в форматі .apk. В цьому модулі також наведені шляхи для основних системних директорій, підключення залежностей та опис можливих завдань роботи Gradle системи, як наприклад, очистка чи перебудова залежностей, зміна версії додатку та створення як версії для налагодження так і для випуску в маркеті Play Market.

Останній модуль buildSrc – є допоміжним модулем для більш гнучкої роботи та збереженням даних для роботи зі збіркою проекту. В ньому розміщені нетипові конфігурації проекту, зручне збереження залежностей та їх серійність для перевикористання в модулі Gradle Scripts.

5.3 Графічний інтерфейс додатку

Інтерфейс додатку розроблений у мінімалістичному стилі та зважаючи на останні інструкції дизайну від Material Design й Google щодо сучасних мобільних застосунків. Також здійснене дослідження та прийнятті міри щодо полегшення зручності використання додатку для людей з вадами зору.

Функціонал «віртуального тренера» включений в модуль тренувань та щоб до нього потрапити спочатку потрібно перейти до екрану всіх підібраних фізичних вправ на визначений день (рисунок 5.3.1), натиснувши на іконку на нижній навігаційній панелі, яка знаходиться посередині з текстом “Exercises”.

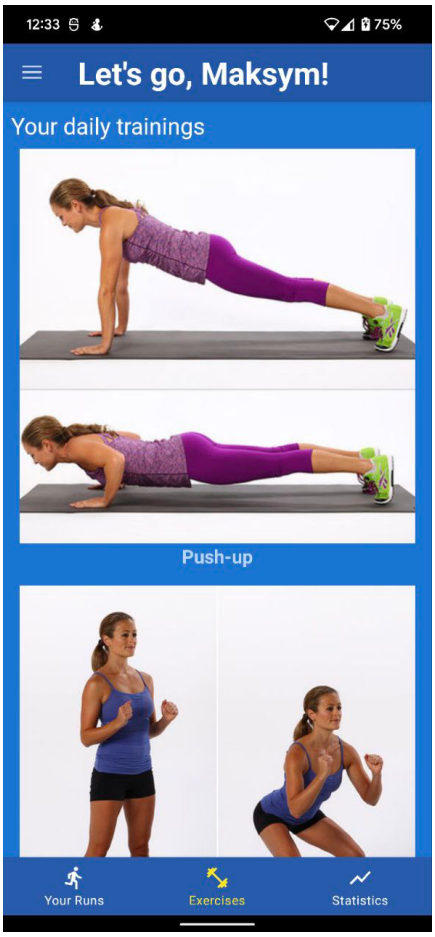

Рисунок 5.3.1 – Екран вправ

На екрані вправ зображений сформований індивідуальний список всіх підібраних вправ на день. Кожен елемент списку складається з зображення та назви вправи. Варто зазначити, що список вправ самостійно оновлюється з сервера один раз на день та використовує механізм кешування в внутрішній

реляційній базі даних додатка. Якщо раптом протягом дня не вдалося оновити список, то це також можливо при переході на цей екран з наявністю підключення до інтернету.

Натиснувши на один з елементів списку вправ користувача буде перенесено на екран з детальною інформацією вибраної вправи (рисунок 5.3.2).

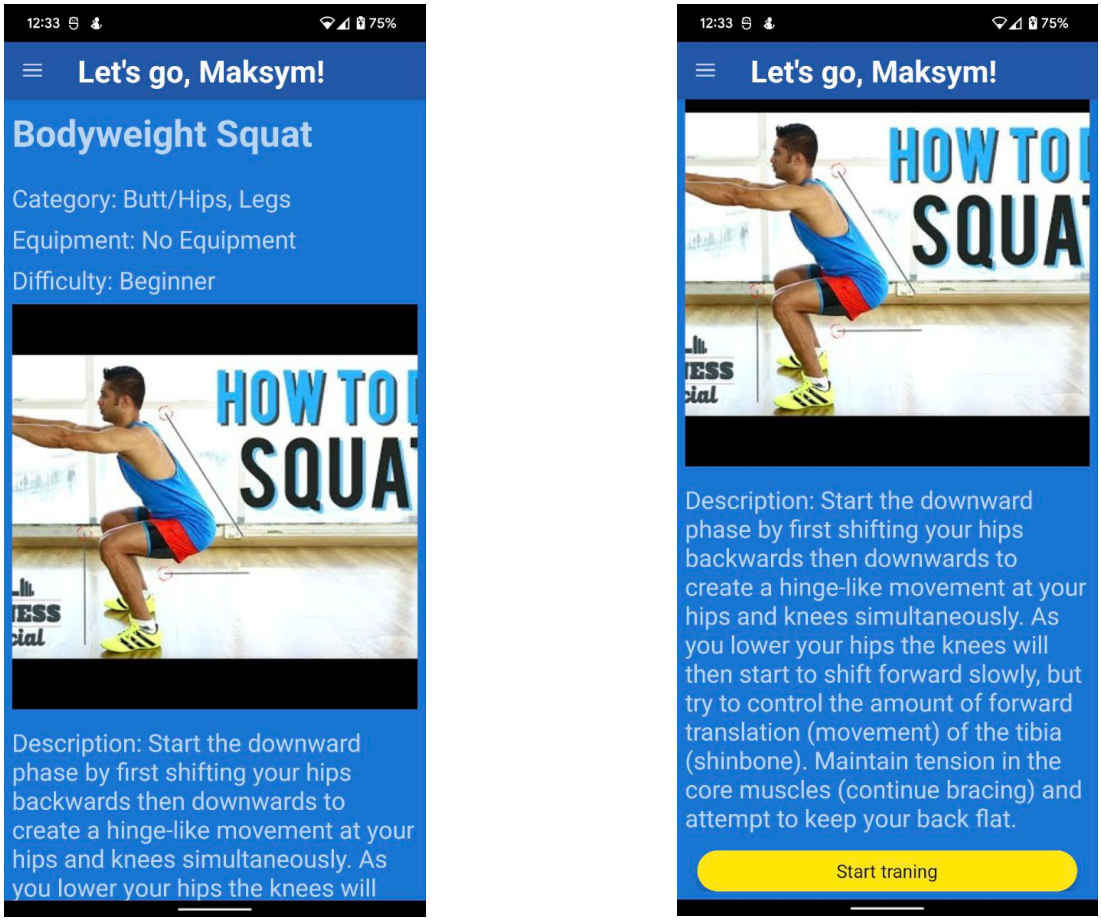

Рисунок 5.3.2 – Екран детальної інформації вправи

На цьому екрані зображені такі данні як: категорія вправи (на які м’язи вона розрахована), необхідне спорядження для її виконання (штанги, гантелі чи інше), рівень складності, детальний опис виконання та також для кожної вправи додано відео посібник, в якому професійний тренер пояснює та показує всі етапи виконання.

В самому низу екрану додано кнопку з написом “Start training”, яка ініціалізує початок роботи функції «віртуального тренера». Після її

натискання буде здійснена навігація користувача до екрану віртуального тренера (рисунок 5.3.3).

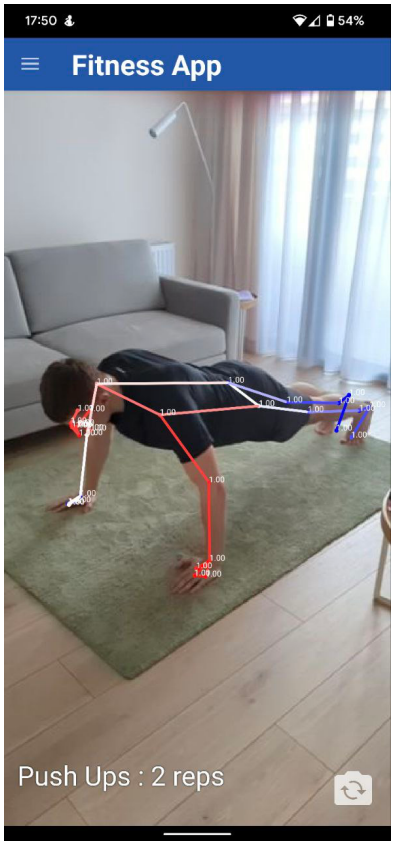

Рисунок 5.3.3 – Екран функції віртуального тренера

Як можна побачити, екран функції віртуального тренера можна розбити на три частини: попереднє зображення з камери, верхня та нижня панель. Основною частиною екрану є елемент типу Preview, саме в ньому в реальному часі відображається попереднє зображення з камери за допомогою бібліотеки CameraX та поверх нього накладається власно створений елемент графічного інтерфейсу, який є «скелетом» орієнтирів пози користувача. На нижній панелі зображена назва виконуваної вправи та кількість здійснених вірних повторень. При кожному вірному повторенні число лічильника змінюється та відбувається відповідне звукове сповіщення. Разом з тим, в лівому нижньому кутку зображена іконка зміни камери, це дає змогу використовувати як фронтальну, так і основну камеру. На верхній панелі розміщене системне меню для навігації та заголовок програми.

Висновки

В результаті виконаного дослідження було розглянуто особливі механізми комп’ютерного зору та машинного навчання, які пов’язані з розпізнаванням людини та її поз, класифікацією їх та підрахунком вірних повторень в реальному часі за допомогою камери смартфону.

Разом з тим було розроблено прототип фітнес додатка з функцією «віртуального тренера». Також варто зазначити, що в створеному застосунку було реалізовано всі функціональні вимоги, які були поставлені на початку дослідження та було проаналізовано відповідні аналоги використання механізмів комп’ютерного зору та машинного навчання у мобільних додатках.

Під час впровадження функція віртуального тренера було детально проаналізовано роботу бібліотеки ML Kit Pose Detection та ознайомлено з сучасними підходами взаємодії з камерою телефона в застосунку на операційній системі Android.

Як можливе покращення системи є адаптація додатку для інших платформ, в тому числі для операційної системи iOS та в майбутньому планується додати механізм авторизації користувача з його особистим кабінетом.

Список літератури

1. Fitness App Market Size [Електронний ресурс]. Режим доступу: https://www.grandviewresearch.com/industry-analysis/fitness-app-market
2. CCO 2020 Keypoint Detection Mask [Електронний ресурс]. Режим доступу: https://cocodataset.org/#keypoints-2020
3. The complete guide to heatmaps [Електронний ресурс]. Режим доступу: https://www.hotjar.com/heatmaps/
4. Alejandro Newell, Kaiyu Yang, Jia Deng. Stacked Hourglass Networks for Human Pose Estimation, European conference on computer vision, 2016
5. Логістична регресія [Електронний ресурс]. Режим доступу: https://stud.com.ua/139982/informatika/logistichna_regresiya
6. Multi-class Classification using Decision-Tree Model [Електронний ресурс]. Режим доступу: https://adityagoel123.medium.com/multi-class-classification-using-decision-tree-model-68e75114303
7. Exponential Moving Average (EMA) [Електронний ресурс]. Режим доступу: https://www.investopedia.com/terms/e/ema.asp
8. Guide to app architecture [Електронний ресурс]. Режим доступу: https://developer.android.com/topic/architecture
9. App Components [Електронний ресурс]. Режим доступу: https://developer.android.com/guide/components/fundamentals#Components
10. Патерни проектування [Електронний ресурс]. Режим доступу: https://refactoring.guru/uk/design-patterns

Додаток А (обов’язковий). Лістінг коду, що реалізує створення власного класу аналізатора зображення та прив’язку його до засобів CameraX.

```
private fun setUpPoseDetection() {
    if (cameraProvider == null) {
        return
    }
    if (analysisUseCase != null) {
        cameraProvider!!.unbind(analysisUseCase)
    }
    if (imageProcessor != null) {
        imageProcessor!!.stop()
    }
    val context = requireContext()
    val poseDetectorOptions = PreferenceUtils.getPoseDetectorOptionsForLivePreview(context)
    val shouldShowInFrameLikelihood =
        PreferenceUtils.shouldShowPoseDetectionInFrameLikelihoodLivePreview(context)
    val visualizeZ = PreferenceUtils.shouldPoseDetectionVisualizeZ(context)
    val rescaleZ = PreferenceUtils.shouldPoseDetectionRescaleZForVisualization(context)
    val runClassification = PreferenceUtils.shouldPoseDetectionRunClassification(context)
    imageProcessor = PoseDetectorProcessor(
        context,
        poseDetectorOptions,
        shouldShowInFrameLikelihood,
        visualizeZ,
        rescaleZ,
        runClassification,
        isStreamMode: true
    )

    val builder = ImageAnalysis.Builder()
    val targetResolution =
        PreferenceUtils.getCameraXTargetResolution(requireContext(), lensFacing)
    if (targetResolution != null) {
        builder.setTargetResolution(targetResolution)
    }
```

```
    analysisUseCase = builder.build()
    needUpdateGraphicOverlayImageSourceInfo = true

    analysisUseCase?.setAnalyzer(
        ContextCompat.getMainExecutor(requireContext()),
        ImageAnalysis.Analyzer { imageProxy: ImageProxy ->
            if (needUpdateGraphicOverlayImageSourceInfo) {
                val isImageFlipped = lensFacing == CameraSelector.LENS_FACING_FRONT
                val rotationDegrees = imageProxy.imageInfo.rotationDegrees
                if (rotationDegrees == 0 || rotationDegrees == 180) {
                    graphicOverlay!!.setImageSourceInfo(
                        imageProxy.width,
                        imageProxy.height,
                        isImageFlipped
                    )
                } else {
                    graphicOverlay!!.setImageSourceInfo(
                        imageProxy.height,
                        imageProxy.width,
                        isImageFlipped
                    )
                }
                needUpdateGraphicOverlayImageSourceInfo = false
            }
            try {
                graphicOverlay?.let { imageProcessor!!.processImageProxy(imageProxy, it) }
            } catch (e: MlKitException) {
                Timber.e( message: "Failed to process image. Error: " + e.localizedMessage)
            }
        }
    )
```

```
    cameraProvider!!.bindToLifecycle( lifecycleOwner: this,
        cameraSelector!!,
        analysisUseCase
    )
}
```

Додаток Б (обов'язковий). Лістінг коду, що реалізує механізм ініціалізації попереднього перегляду.

```
private fun bindPreviewUseCase() {
    if (!PreferenceUtils.isCameraLiveViewportEnabled(requireContext())) {
        return
    }
    if (cameraProvider == null) {
        return
    }
    if (previewUseCase != null) {
        cameraProvider!!.unbind(previewUseCase)
    }

    val builder = Preview.Builder()
    val targetResolution =
        PreferenceUtils.getCameraXTargetResolution(requireContext(), lensFacing)
    if (targetResolution != null) {
        builder.setTargetResolution(targetResolution)
    }
    previewUseCase = builder.build()
    previewUseCase!!.setSurfaceProvider(previewView!!.surfaceProvider)
    cameraProvider!!.bindToLifecycle( lifecycleOwner: this,
        cameraSelector!!,
        previewUseCase
    )
}
```

Додаток В (обов'язковий). Лістінг коду, що реалізує результат процесу класифікації позиції по її орієнтирам

```
private fun classify(landmarks: List<PointF3D?>): ClassificationResult {
    val result = ClassificationResult()
    if (landmarks.isEmpty()) {
        return result
    }

    val flippedLandmarks: List<PointF3D?> = ArrayList(landmarks)
    Utils.multiplyAll(flippedLandmarks, PointF3D.from( x: -1f,  y: 1f,  z: 1f))
    val embedding = PoseEmbedding.getPoseEmbedding(landmarks)
    val flippedEmbedding = PoseEmbedding.getPoseEmbedding(flippedLandmarks)
    val maxDistances = PriorityQueue(
        maxDistanceTopK
    ) { o1: Pair<PoseSample, Float?>, o2: Pair<PoseSample, Float?> ->
        -(o1.second!!).compareTo(o2.second!!)
    }

    for (poseSample in poseSamples) {
        val sampleEmbedding = poseSample.embedding
        var originalMax = 0f
        var flippedMax = 0f
        for (i in embedding.indices) {
            originalMax = max(
                originalMax,
                Utils.maxAbs(
                    Utils.multiply(
                        Utils.subtract(
                            embedding[i], sampleEmbedding[i]
                        ),
                        axesWeights
                    )
                )
            )
            flippedMax = max(
                flippedMax,
```

```
                Utils.maxAbs(
                    Utils.multiply(
                        Utils.subtract(
                            flippedEmbedding[i], sampleEmbedding[i]
                        ),
                        axesWeights
                    )
                )
            )
        }
        maxDistances.add(Pair(poseSample, min(originalMax, flippedMax)))
        if (maxDistances.size > maxDistanceTopK) {
            maxDistances.poll()
        }
    }
    val meanDistances = PriorityQueue(
        meanDistanceTopK
    ) { o1: Pair<PoseSample, Float?>, o2: Pair<PoseSample, Float?> ->
        -(o1.second!!).compareTo(o2.second!!)
    }
    for (sampleDistances in maxDistances) {
        val poseSample = sampleDistances.first
        val sampleEmbedding = poseSample.embedding
        var originalSum = 0f
        var flippedSum = 0f
        for (i in embedding.indices) {
            originalSum += Utils.sumAbs(
                Utils.multiply(
                    Utils.subtract(
                        embedding[i],
                        sampleEmbedding[i]
                    ),
                    axesWeights
                )
```

```
            )
            flippedSum += Utils.sumAbs(
                Utils.multiply(
                    Utils.subtract(
                        flippedEmbedding[i], sampleEmbedding[i]
                    ),
                    axesWeights
                )
            )
        }
        val meanDistance = Math.min(originalSum, flippedSum) / (embedding.size * 2)
        meanDistances.add(Pair(poseSample, meanDistance))
        if (meanDistances.size > meanDistanceTopK) {
            meanDistances.poll()
        }
    }
    for (sampleDistances in meanDistances) {
        val className = sampleDistances.first.className
        result.incrementClassConfidence(className)
    }
    return result
}
```

Додаток Г (обов'язковий). Лістінг коду, що реалізує отримання результату класифікації по позиції

```
@WorkerThread
fun getPoseResult(pose: Pose): List<String> {
    Preconditions.checkState( expression: Looper.myLooper() != Looper.getMainLooper())
    val result: MutableList<String> = ArrayList()
    var classification = poseClassifier!!.classify(pose)
    if (isStreamMode) {
        classification = emaSmoothing!!.getSmoothedResult(classification)
        if (pose.allPoseLandmarks.isEmpty()) {
            lastRepResult?.let { result.add(it) }
            return result
        }
        for (repCounter in repCounters!!) {
            val repsBefore = repCounter.numRepeats
            val repsAfter = repCounter.addClassificationResult(classification)
            if (repsAfter > repsBefore) {
                val tg = ToneGenerator(AudioManager.STREAM_NOTIFICATION, volume: 100)
                tg.startTone(ToneGenerator.TONE_PROP_BEEP)
                lastRepResult = String.format(
                    Locale.getDefault(), format: "%s : %d reps", repCounter.className, repsAfter
                )
                break
            }
        }
        lastRepResult?.let { result.add(it) }
    }
    return result
}
```