Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів для аналізу рухів людини та реалізація мобільного застосунку на основі штучного інтелекту
(тема)

Виконав:
студент 2 курсу, групи СШІм-19-2
Кусков Р. В.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)

Керівник доц. каф. ШІ Вітько О. В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Кускову Родіону Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів для аналізу рухів людини та реалізація мобільного застосунку на основі штучного інтелекту

затверджена наказом університету від 29 березня 20 21 р. № 390 Ст

2. Термін подання студентом роботи до екзаменаційної комісії _____ _____ 20 ___ р.

3. Вихідні дані до роботи Доповіді компанії Apple, документація Swift, документація CoreML, дані з відкритих Інтернет-джерел, доповіді інших розробників мобільних застосунків

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної області та технологій
2) Вибір методу для визначення рухів людини
3) Постановка задачи
4) Обґрунтування вибора мови програмування
5) Порівняння продуктивності мов програмування
6) Проектування архітектури мобільного застосунку
7) Розробка алгоритму розпізнавання рухів
8) Розробка прототипу мобільного застосунку

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)

Рисунок 1 – Налаштування першого етапу тренування; Рисунок 2 – Результат тренуваня першої інтерації моделі; Рисунок 3 – Результат тренування першої моделі на 140 ітерація; Рисунок 4 – Результат передбачення удару; Рисунок 5 – Скелет людини у вкладці Wireframe; Рисунок 6 – Налаштування другої ітерації труннання моделі; Рисунок 7 – Результат тренувань другої ітерації моделі; Рисунок 8 – Результат передбачення бокового удару; Рисунок 9 – Налаштування третьої ітерації тренування моделі; Рисунок 10 – Результат тренувань третьої ітерації моделі; Рисунок 11 – Результат точності для кожного класу; Рисунок 12 – Результат передбачення бокового удару; Рисунок 13 – Розподіл відсотків по передбаченню удару; Рисунок 14 – Результат передбачення удару снизу.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної галузі	15.04.2021	виконано
2	Аналіз методів розпізнавання рухів людини	16.04.2021	виконано
3	Збір даних для тренування моделі	19.04.2021	виконано
4	Тренування моделі	21.04.2021	виконано
5	Аналіз отриманих результатів	22.04.2021	виконано
6	Розробка мобільного застосунку	26.04.2021	виконано
7	Написання тестів для мобільного додатку	28.04.2021	виконано
8	Написання пояснювальної записки	30.04.2021	виконано
9	Рецензування	03.05.2021	виконано
10	Подготовка презентації	05.05.2021	виконано
11	Попередній захист	14.05.2021	виконано
12	Захист перед ЕК		

Дата видачі завдання 29 03 20 21 р.

Студент _____
(підпис)

Керівник роботи _____ *доц. каф. ШІ Вітько О. В*
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 75 с., 1 табл., 37 рис., 2 дод., 17 джерел.

БОКС, МАШИННЕ НАВЧАННЯ, РОЗПІЗНАВАННЯ РУХІВ ЛЮДИНИ, ШТУЧНИЙ ІНТЕЛЕКТ, CREATEML, IOS, SWIFT.

Об’єкт розробки – програмний засіб для розпізнавання рухів людини у контексті боксу за моделі машинного навчання, яка впроваджена у мобільний застосунок.

Мета роботи – розробка моделі машинного навчаня, яка буде впроваджена у мобільний застосунок, який дозволятиме користувачам вивчати боксерські удари, удосконалювати існуючі навички та практикуватися у вільному режимі.

Результати роботи – мобільний застосунок, розрахований під платформу iOS, який дозволяє користувачу вивчати боксерські удари, удосконалювати існуючі навички та практикуватися у свободному режимі. Для того, щоб завантажити застосунок, потрібно мати iPhone X або модель новіше. Тренування будуть доступні користувачу одразу після завантаження застосунку.

Також результатом работи є розробка моделі машинного навчання, яка буде передбачати удари, які б’є користувач мобільного застосунку, за допомогою камери телефону, яка передає зображення у режимі реального часу.

Галузь застосування – продукт використовується для особистого навчання боксерської техніки та для покращення існуючих навичок.

РЕФЕРАТ

Пояснительная записка: 75 с., 1 табл, 37 рис., 2 прил., 17 источников.

БОКС, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, МАШИННОЕ ОБУЧЕНИЕ, РАСПОЗНАВАНИЕ ДВИЖЕНИЙ ЧЕЛОВЕКА, CREATEML, IOS, SWIFT.

Объектом разработки является программный инструмент для распознавания движений человека в контексте бокса при помощи модели машинного обучения, которая реализована в мобильном приложении.

Целью работы является разработка модели машинного обучения, которая будет реализована в мобильном приложении, которое позволит пользователям изучать боксерские удары, совершенствовать существующие навыки и практиковаться в свободном режиме.

Результаты работы – мобильное приложение, разработанное для платформы iOS, которое позволяет пользователю изучать боксерские удары, совершенствовать имеющиеся навыки и практиковаться в свободном режиме. Для того, чтобы скачать приложение, у пользователя должен быть iPhone X или более новая модель. Тренировки будут доступны пользователю сразу после загрузки приложения.

Также результатом является разработка модели машинного обучения, которая будет включать удары, нанесенные пользователем мобильного приложения, с помощью камеры телефона, которая передает изображения в реальном времени.

Область применения – продукт используется для персонального обучения боксерским приемам и для улучшения имеющихся навыков.

ABSTRACT

Thesis contains: 75 p., 1 tabl., 37 fig., 2 ann., 17 references.

ARTIFICIAL INTELLIGENCE, BOX, CREATEML, IOS, HUMAN ACTION RECOGNITION, MACHINE LEARNING, SWIFT.

The object of development is a software tool for recognising human movements in the context of boxing according to the machine learning model, which is implemented in the mobile application.

The purpose of the work is the development a model of machine learning, which will be implemented in a mobile application that will allow users to learn boxing punches, improve existing skills and practice in free mode.

Results of work – a mobile application designed for the iOS platform, which allows the user to learn boxing punches, improve existing skills and practice in free mode. In order to download the application, you need to have an iPhone X or a newer model. The workouts will be available to the user immediately after downloading the application.

Also the result is the development of a machine learning model, which will include the blows hit by the user of the mobile application, using a phone camera that transmits images in real time.

Scope – the product is used for personal training of boxing techniques and to improve existing skills.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень та термінів 9

Вступ 10

1 Аналіз предметної області та технологій 12

1.1 Аналіз завдання 12

1.2 Аналіз існуючих рішень 12

1.2.1 Human–action–classification 13

1.2.2 Skelemoa / ntu–x 14

1.2.3 ChengeYang / Human–Pose–Estimation–Benchmarking–and–Action–Recognition 17

1.2.4 Apple / Action Classifier with Create ML 19

2 Постановка задачі та рішення, що пропонується 21

2.1 Постановка задачі 21

2.2 Вибір методу для визначення рухів людини 22

3 Розробка прототипного рішення 24

3.1 Обґрунтування вибору мови програмування 24

3.1.1 Порівняння продуктивності мов програмування 31

3.2 Проектування архітектури мобільного додатку 35

3.3 Практичні дослідження та налаштування моделі навчання 43

3.4 Розробка прототипу мобільного додатку 56

Висновки 62

Перелік джерел посилання 63

Додаток А Код програми.....65

Додаток Б Відомість кваліфікаційної роботи.....75

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

API – application programming interface – програмний інтерфейс застосунку;

CPU – central process unit – центральний процесор;

GB – giga byte – гігабайт;

GPU – graphics processing unit – графічний процесор;

MVC – model, view, controller – модель, перегляд, контроллер;

MVP – model, view, presenter – модель, перегляд, презентер;

MVVM – model, view, viewModel – модель, перегляд, вьюМодель.

ВСТУП

Штучний інтелект неймовірно швидко розвивається в XXI столітті. У зв'язку з швидким розвитком, все більше і більше IT компаній популяризують штучний інтелект. Тому, розробка різних систем штучного інтелекту є одним з найбільш актуальним і затребуваним напрямком в наше століття.

Досить важко уявити хоча б один успішний проект, який користується високим попитом серед користувачів, в якому не було б штучного інтелекту або машинного навчання. Починаючи від самих базових додатків, такі як музичний плеєр, які здійснюють підбір музики за улюбленими для користувача жанрами, артистам і так далі, і закінчуючи складними системами, які за допомогою невеликих обсягів вхідної інформації в змозі визначити, хворі ви на певну хворобу чи ні.

Штучний інтелект популярний не тільки серед користувачів звичайних додатків, але і серед професіоналів, які користуються системами, застосунками в ході своєї роботи. Гарним прикладом може бути система, яка з високою точністю визначає наявність уражень легеневих тканин у пацієнта лише по одному рентгенівському знімку. Це далеко не перший і останній приклад складних систем, який допомагають людині в професійній діяльності. Наприклад, в державних установах штучний інтелект використовується для визначення осіб злочинців, скануючи людей на вулицях за допомогою прихованих камер, далі здійснюється пошук по базі злочинців і система автоматично знаходить людей, у яких збіги особи в процентному співвідношенні максимально велике, з особою злочинців в базі даних. Дана система не нова, відома всім як «face detection» (англ. – визначення особи).

Рішення машинного навчання і штучного інтелекту підняли розробку мобільних додатків на новий рівень. Додатки з інтеграцією технологій машинного навчання можуть успішно розпізнавати і класифікувати зображення, людські голоси і дії, виконувати розпізнавання тексту з

зображень і переводити його. Цей список можна легко продовжити. Однак як відомо, майже усі моделі зі штучним інтелектом працюють шляхом використання серверу, який спілкується з мобільним додатком шляхом різноманітних API запросів. Саме тому головною проблемою для інженерів залишається перенос на мобільний телефон величезних моделей з мільйонами підключень без втрати швидкості обробки та, що найголовніше, – якості.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ

1.1 Аналіз завдання

Завдання кваліфікаційної роботи полягає в дослідженні методів для аналізування рухів людини і розробка програми, за допомогою якої можна застосувати обраний спосіб. Для виконання поставленого завдання необхідно:

– вивчити предметну область;
– проаналізувати існуючі рішення;
– проаналізувати і вибрати оптимальний алгоритм для застосування;
– провести навчання моделі;
– впровадити алгоритм в мобільний застосунок.

Досліджувані існуючі методи повинні мати можливість визначати руху людини за допомогою вхідних параметрів, а саме відеофайлу. Методи повинні володіти великою швидкістю, наближеної до оновлень в реальному часі.

1.2 Аналіз існуючих рішень

Розглянемо декілька існуючих рішень для аналізу рухів людини за допомогою відеофайлів, які знаходяться у відкритому доступі. Більшість всіх рішень базується на тому, що передача даних між клієнт частиною (мобільним застосунком) та сервером відбувається шляхом роботи з RestAPI. Цих знань вже достатньо, щоб виділити кілька дуже вагомих мінусів:

– відсутність автономної роботи без наявності з'єднання з Інтернетом;

– неможливість виконання умови поновлення інформації та аналізування в реальному часі без єдиних затримок.

Розглянемо кілька найбільш популярних рішень для визначення рухів людини:

– dronefreak / human–action–classification;
– skelemoa / ntu–x;
– ChengeYang / Human–Pose–Estimation–and–Action–Recognition;
– Apple / Action Classifier with Create ML.

Слід зазначити, що рішення від компанії Apple – це рідний фреймворк, що розробляється під платформи macOS, iOS, tvOS, watchOS та інші операційні системи компанії Apple. Це забезпечує безболісне впровадження в мобільний застосунок.

1.2.1 Human–action–classification

Human–action–classification – алгоритм, який знаходиться у відкритому доступі в репозиторії Github, розроблений Нідерландським інженером. Сам алгоритм використовує реалізацію OpenPose з TenserFlow для виявлення поз. Для бінарної класифікації сидячих або стоячих поз використовується MobileNet, спочатку навчена на наборі даних ImageNet Large Visual Recognition Challenge. Далі модель була перенавчена на наборі даних з ~ 1500 зображень поз. Для класифікації сцен навколо людини була перенавчена архітектура Inception-v3 на наборі даних Stanford 40 (9523 зображення), який включає 40 різних повсякденних дій [1].

Автор алгоритму стверджує, що точність розпізнавання поз людини становить 94%, а точність розпізнавання сцен – 90%. Приклад роботи алгоритму зображений на рисунку 1.1.

У цьому проекті виконуються два завдання класифікації. Одна з них – класифікація поз, а інша – класифікація сцен. На рисунку 1.1 зображена людина, яка гуляє з собаками, та алгоритм розпізнав, що на зображені людина знаходится у вертикальної позиції, та передбачувана сцена – прогулка з собаками.

Також автор показує, наскільки точним є прогноз. У данному випаду, точність прогнозу є 99.6% що до дій людини, та 100% прогноз щодо пози людини. Цей показник є досить високим.

Рисунок 1.1 – Приклад роботи алгоритму

1.2.2 Skelemoa / ntu–x

Вихідний набір даних NTU містить скелет дій людини, знятий за допомогою Kinect. У цих скелетів 25 суглобів. Всі поточні найефективніші моделі здаються вузькими місцями в певних класах, які включають більш тонкі рухи на рівні пальців, такі як читання, письмо, прийом їжі і т. д [2].

Отже, новий набір даних NTU–X представляє більш детальний каркас людини з 118 суглобів для послідовностей дій набору даних NTU. Цей новий набір даних, поряд з 25 суглобами тіла, містить 42 суглоба пальців та 51 суглоби людини [2].

Приклад скелета NTU–X зображений на рисунку 1.2. Також можна помітити на рисунку 1.2, що окрім звичайного відображеня скелета людини, алгоритм відображає схематично обличчя людини, а також її кисті рук.

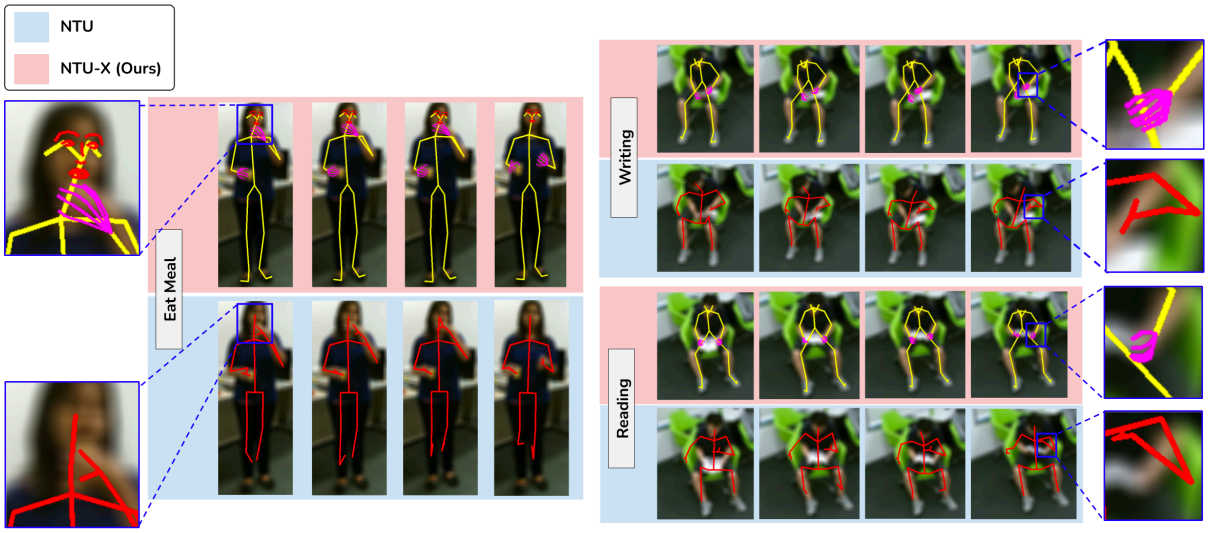

Рисунок 1.2 – Приклад скелета NTU-X

Окрім рисунків щодо роботи NTU-X, автор також запроваджує інформацію щодо функціоналу стосовно інших популярних датасетів. Ця інформація зображена на рисунку 1.3.

Comparing NTU-X to other popular action datasets.

Dataset	Body	Fingers	Face	# Joints	# Sequences	# Classes
MSR-Action 3d	✓			20	567	20
Northwestern-UCLA	✓			24	1475	10
NTU-RGB+D	✓			25	56880	60
NTU-RGB+D 120	✓			25	114035	120
NTU60-X (Ours)	✓	✓	✓	**118**	**56148**	**60**
NTU120-X (Ours)	✓	✓	✓	**118**	**113821**	**120**

Рисунок 1.3 – Порівняння функціоналу NTU-X з іншими рішеннями

Автори алгоритму стверджують, що точність визначення дій людини варіюється від 90% до 92%, що є нижчим, ніж у попереднього розглянутого

алгоритму, але все одно точність є досить високою [2].

Також на рисунках 1.4 и 1.5 зображені приклади розпізнавання дій людини, а саме: письмо і читання відповідно.

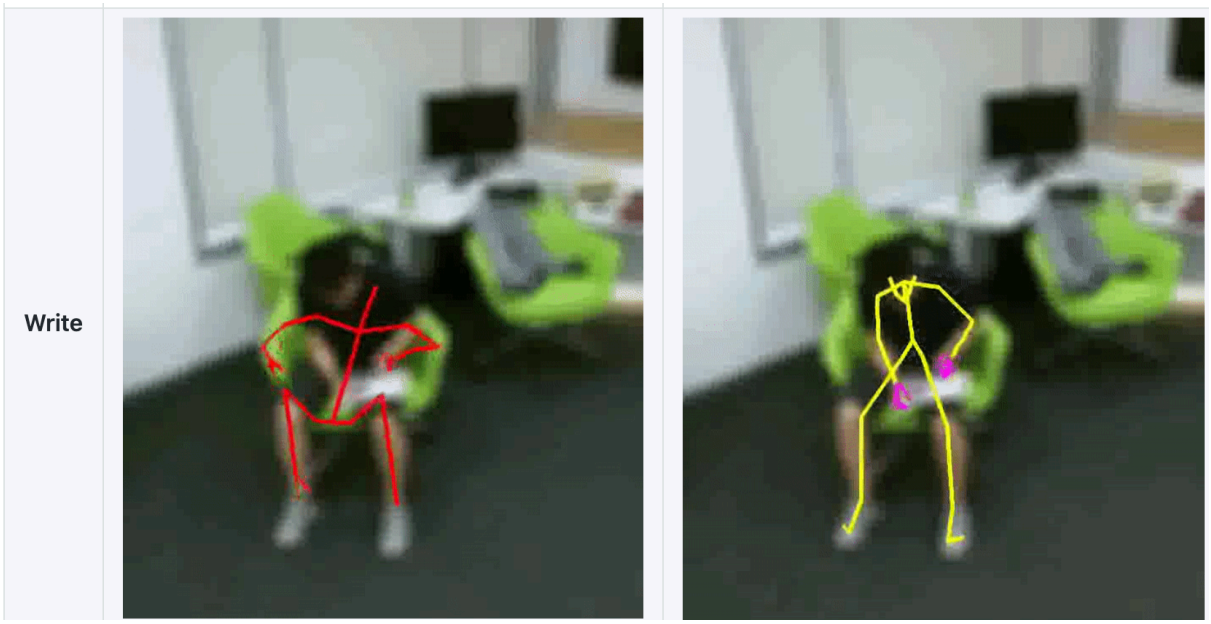

Рисунок 1.4 – Приклад розпізнавання дій людини, яка пише

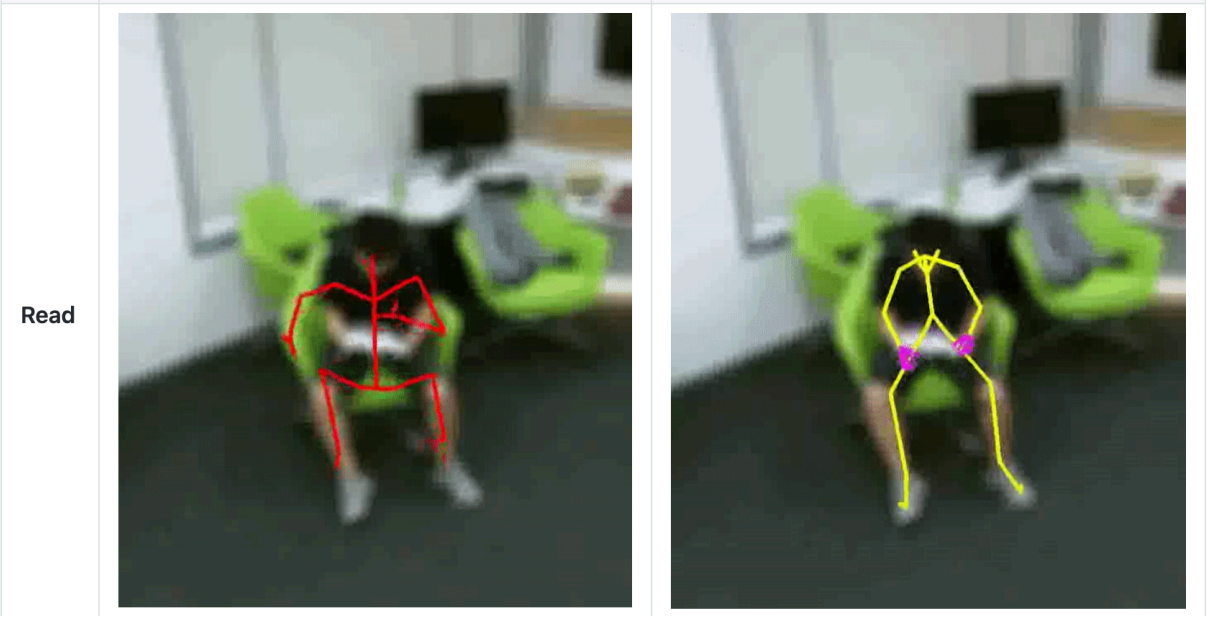

Рисунок 1.5 – Приклад розпізнавання дій людини, яка читає

1.2.3 ChengeYang / Human–Pose–Estimation–Benchmarking–and–Action–Recognition

Проект складається з кілька основних частин, які розроблялися норвезькимі розробниками взимку в 2019 році.

Першою частиною є бенчмаркінг оцінки пози людини. У цій частині розробники провели порівняльний тест на двох найсучасніших моделях оцінки пози людини – OpenPose і AlphaPose. Також протестували різні режими роботи алгоритму як для одного, так і для декількох людей. Результати першої частини зображені на рисунку 1.6 для декількох людей, а також на рисунку 1.7 для однієї людини. Варто відзначити, що обидва бенчмарка, що для однієї людини, що для кількох, проводилися за допомогою відео з роздільною здатністю в 1920x1080, в сумі 920 фреймів та 30 кадрів в секунду [3].

Бенчмарки проводилися на наступному апаратному забезпеченні:

– CPU: AMD Ryzen Threadripper 1920X (12–core / 24–thread);
– GPU: Nvidia GTX 1080Ti – 12 GB;
– RAM: 64GB;
– OS: Ubuntu 18.04.

Model	Accuracy	GPU Memory Usage	Processing speed(fps)
AlphaPose	low	60.3%	26.50
	default	60.3%	23.71
	high	60.2%	11.41
OpenPose	low	20.0%	19.59
	default	21.3%	18.77
	high	97.9%	3.47

Рисунок 1.6 – Результат бенчармків для кількох людей

Model	Accuracy	GPU Memory Usage	Processing speed (fps)
AlphaPose	low	57.1%	5.21
	default	73.4%	1.15
	high	70.1%	0.69
OpenPose	low	20.1%	19.17
	default	21.3%	18.39
	high	98.1%	3.47

Рисунок 1.7 – Результат бенчармків для однієї людини

Також розробники наводять приклад, як їх проект визначає руху декількох людей. Даний приклад зображений на рисунку 1.8.

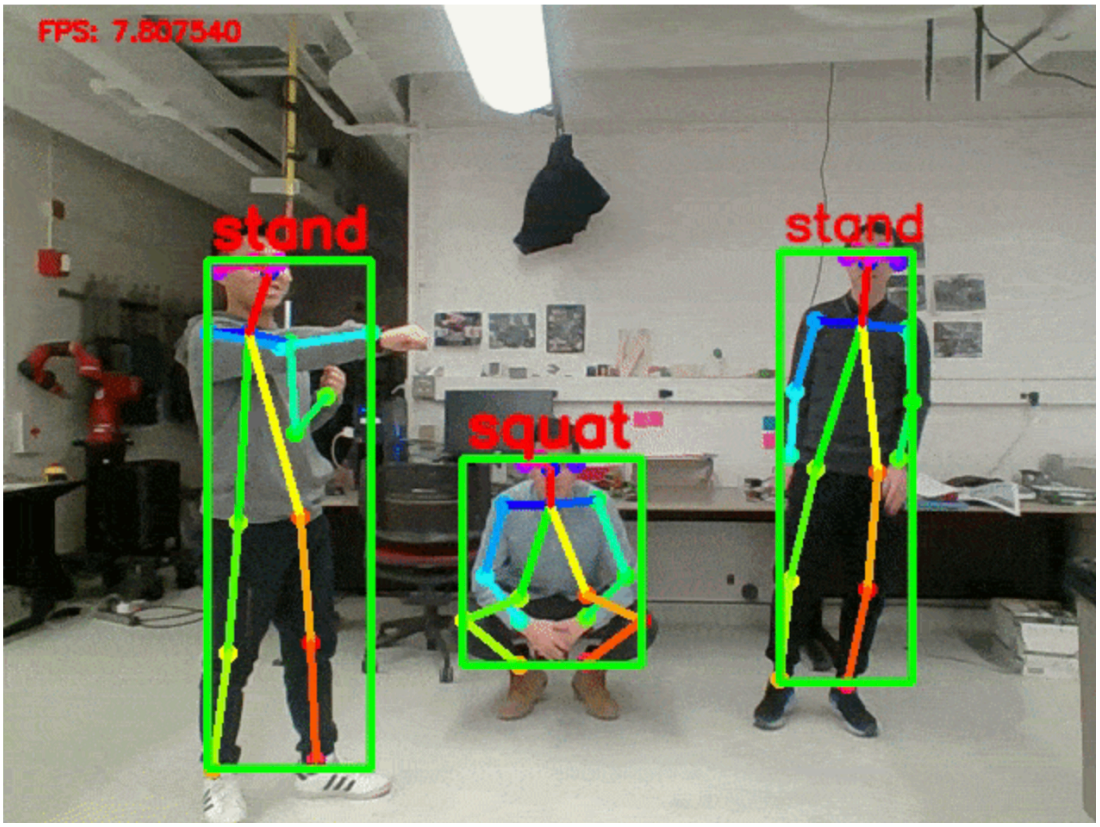

Рисунок 1.8 – Приклад роботи алгоритму, який у реальному часі визначає рухі людей

1.2.4 Apple / Action Classifier with Create ML

Останнім розглянутим рішенням для аналізу рухів людини є Фреймворк від компанії Apple – Create ML.

На щорічній конференції для розробників WWDC2018 компанія Apple представила інструменти для роботи з машинним навчанням Create ML. Навчена в Create ML модель є результатом застосування алгоритму машинного навчання для набору навчальних даних. Моделі не займають багато місця (близько 3Мб), тому їх можна зберігати в проекті. Спочатку моделі пропонувалося навчати за допомогою Playgrounds в XCode 10 (середа розробки) і підтримувалася робота з зображеннями, текстом і таблицями. При запуску Playgrounds проекту необхідно було імпортувати бібліотеку CreateML та запустити MLImageClassifierBuilder (в разі якщо б ми навчали модель для класифікації зображень) [4].

У 2019 Apple представила абсолютно новий інструмент для роботи з моделями машинного навчання, відокремивши його від Playgrounds. Стало можливим навчати моделі без єдиної строчки коду. Список типів даних поповнився звуками і активністю. Всього представлено 5 типів даних і 9 шаблонів моделей. Відповідно для даної роботи необхідно навчання моделі з рухами [4].

Великим плюсом є те, що Create ML – це рідний iOS фреймворк, з імплементацією якого не буде ніяких проблем. Даний фреймворк працює незалежно від інтернет з'єднання, також може випускати оновлення інформації в реальному часі, а також гарантує максимально можливу швидкість обробки інформації.

Також великим плюсом є те, що точність визначення рухів залежить тільки від розробника. Чим більше вхідних даних для навчання буде надано, тим точніше модель буде визначати рухи. До того ж, Create ML має можливість визначати міміку обличчя, рух пальців тощо в реальному часі.

Немає необхідності комбінувати декілька готових рішень для того, щоб відстежувати рухи пальців і рух самого тіла.

До всього цього, всередині XCode автоматично будуть зібрані різні метрики, такі як точність прогнозів, кількість ітерацій які були у ході тренування моделі та іншні. Також існує можливість додати на вхідні дані для моделі різні шуми на відеофайли, засвітки, зменшення FPS тощо.

2 ПОСТАНОВКА ЗАДАЧІ ТА РІШЕННЯ, ЩО ПРОПОНУЄТЬСЯ

2.1 Постановка задачі

Метою даної роботи є дослідження методів для аналізу рухів людини і розробка мобільного додатка на основі штучного інтелекту і машинного навчання.

Для досягнення поставленої мети необхідно виконати наступні завдання:

– складання плану роботи;
– дослідити існуючі методи;
– зібрати метрики існуючих методів та порівняти їх;
– вибрати необхідний метод, який буде застосовуватися в системі;
– зібрати дані, на яких буде тренуватися модель;
– провести навчання моделі, яка буде розпізнавати рухи людини;
– провести аналіз отриманної моделі;
– удасконалити модель шляхом додавання нових класів;
– зробити дизайн для мобільного застосунку;
– розробити мобільний застосунок під платформу iOS, який буде використовувати вибраний метод аналізу рухів та модель для розпізнання рухів людини;
– імплементувати натренеровану модель у мобільний застосунок;
– покрити застосунок необхідними Unit тестами;
– зробити тести застосунку у складних умовах;
– зібрати результуючі метрики;
– скласти проектну документацію.

В якості вхідних даних буде стрімінговий відеопотік, який буде захоплюватися з камери користувача, передаватиметься у мобільний додаток та потім до натренерованної моделі.

В свою чергу користувач буде бачити поради що до тренувань ударів.

2.2 Вибір методу для визначення рухів людини

Базуючись на інформації, яка була отримана щодо готових рішень, була зроблена наступна таблиця під номером 2.1, в якому будуть перераховані різні плюси і мінуси кожного рішення.

Таблиця 2.1 – Порівняння плюсів і мінусів рішень

Фреймворк	Інтеграция в iOS	Працює без сервера	Оновлення в реальному часі	Розмір моделі менше 3МБ	Наявність повноцінної документації
human–action–classification	–	–	–	–	–
ntu–x	–	–	–	–	–
HPEBAR	–	+	+	–	–
Create ML	+	+	+	+	+

Виходячи з таблиці 2.1 можна зрозуміти, що найбільш підходящими рішеннями для імплементації у мобільний застосунок такі рішення, як Human-Pose-Estimation-Benchmarking-and-Action-Recognition (далі HPEBAR) та Create ML від Apple. Перші два рішення не задовольняють заданим вимогам.

Вибираючи між HPEBAR і Create ML вибір падає на другий варіант. Найбільшим плюсом є можливість прямої інтеграції в iOS, як фреймворк.

Також важливим фактором є можливість роботи без серверної частини, а також підтримка оновлення інформації в реальному часі. Вирішальну роль зіграли останні два пункти – розмір моделі і наявність повноцінної документації. Розмір моделі важливий через те, що рішення

планується використовуватися в мобільному застосунку, та як відомо, розмір моделі впливає на розмір мобільного застосунку, та для користувача дуже важливо, щоб розмір мобільного застосунку був не дуже великим, а документація, в свою чергу, важлива для більш чіткого розуміння, як влаштований механізм і які взагалі у нього можливості існують. Тому, остаточним вибором є Create ML від Apple.

3 РОЗРОБКА ПРОТОТИПНОГО РІШЕННЯ

3.1 Обґрунтування вибору мови програмування

Виходячи з того, що система зі штучним інтеллектом, яка розробляються у ході цієї роботи, буде впроваджена у мобільний застосунок під платформу iOS, то мовою розробки можуть бути такі:

– React Native;

– Objective-C;

– Swift.

Усі три мови програмування є дуже добрими інструментамі у розробці, які мають свої плюси. Але кожна мова має свої мінуси також. Мова програмування обиралася по принципу якнайбільше плюсів, та найменше мінусів.

Отже, почнемо з React Native. React Native – це фреймворк, який підтримує Facebook, створений для побудови мобільних додатків. На відміну від нативних програм, які використовують різні мови програмування для кожної платформи, React Native базується на JavaScript і дозволяє користувачам спільно використовувати код між різними платформами, включаючи Android, iOS і веб-програми. До 70% коду можна розділяти між застосунками, що значно скорочує час розробки в розробці мобільних застосунків [6]. Та у React Native є дуже вагові мінуси, а саме:

– обмежений API. Хоча React Native підтримує величезну кількість API інтерфейсів, все ще існує потреба у використанні інших API через вбудовані модулі. Ці модулі в основному є частиною коду, написаного на рідною мовою, після чого інтегрованого в існуючий код. Безсумнівно, це відмінний спосіб вирішити проблему, але вам потрібно мати достатній досвід щодо рідної мови та володіти його інструментами. Можливо через цих обмежень вам все ж будуть потрібні професійні програмісти, які спеціалізуються на необхідному вам нативном мовою, щоб через мости робити зв'язки з React Native компонентами [7];

– відмінності платформ. Android та iOS використовують різні принципипроектування. Це означає, що за допомогою React Native доведеться включити безліч if-statements разом з окремим кодом, щоб укладатися в рамки розміщення графічних елементів. У доповнення до цього, створення високоякісного користувальницького інтерфейсу також виключає парадигму React, і ми були змушені писати Swift бібліотеки при розробці цієї програми. І ми вже не говоримо про те, що потрібно супроводжувати відразу кілька конфігураційних груп [7].

Також React Native має свої плюси, які перераховані нижче:

– розглядаючи React окремо, це JavaScript бібліотека для побудови призначених для користувача інтерфейсів, яка через роботу з віртуальним DOM ефективно оновлює і рендерить компоненти. Вона відмінно робить свою задачу, але в той же час нічого іншого не містить. Це як готувати ціле блюдо з одного інгредієнта. Неможливо створити застосунок, використовуючи тільки React. Безумовно, вам знадобиться webpack для збірки коду, подобу CSS для стилізації, щось схоже на Firebase для обслуговування даних і багато чого іншого. Ну а якщо ваш обсяг роботи обмежений тільки веб-розробкою, то може бути досить і одного React. Однак якщо ви також займаєтеся розробкою дизайну мобільних додатків, неможливо додати повний набір інструментів, які будуть універсальні. Ось чому React – це бібліотека, а React Native – це фреймворк [7];

– інтегрувальні рідні компоненти. Додатки React Native дозволяють використовувати рідні елементи інтерфейсів з операційної системи, одночасно підтримуючи парадигму React в JavaScript. React Native пропонує ряд інтегрованих компонентів, що дозволяє розробникам швидко і ефективно виконувати основні завдання [7].

Наступною мовою програмування є Objective-C (скорочено Obj-C). Objective-C – об'єктно-орієнтована мова програмування, яка компілюється корпорації Apple, побудований на основі мови C і парадигм Smalltalk. На відміну від C ++, мова Objective-C повністю сумісний з C (мова Objective-C є надбезліччю мови C) і код на C компілюється. Об'єктна модель

побудована в стилі Smalltalk, тобто об'єктам надсилаються повідомлення. Компілятор Objective-C входить в GCC і доступний на більшості основних платформ. Мова використовується в першу чергу для Mac OS X (Cocoa) і GNUstep – двох реалізацій об'єктно-орієнтованого інтерфейсу OpenStep. Objective-C був створений Бредом Коксом на початку 1980х в його компанії Stepstone. Ще однією з особливостей мови є те, що мова message-oriented в той час як C ++ – function-oriented. Це означає, що в мові виклики методу інтерпретуються не як виклик функції (хоча до цього зазвичай все зводиться), а саме як посилка повідомлення (з ім'ям і аргументами) об'єкту, подібно до того, як це відбувається в Smalltalk. Такий підхід дає цілий ряд плюсів – так будь-якому об'єкту можна послати будь-яке повідомлення. Об'єкт може замість обробки повідомлення просто переслати його іншому об'єкту для обробки (так зване делегування), зокрема саме так можна легко реалізувати розподілені об'єкти (тобто об'єкти, що знаходяться в різних адресних просторах і навіть на різних комп'ютерах). Прив'язка повідомлення до відповідної функції відбувається безпосередньо на етапі виконання. У мові є нормальна підтримка протоколів (тобто поняття інтерфейсу об'єкта і протоколу чітко розділені). Для об'єктів підтримується спадкування, для протоколів підтримується множинне спадкування. Об'єкт може бути успадкований від іншого об'єкта і підтримувати відразу кілька протоколів. Структура найменування файлів: файли з розширенням – h є заголовками з описом класів, функцій також як в C і C ++, файли з розширенням – m містять реалізацію класів і методів [9].

Також як і у React Native, Objective-C має свої недоліки, а саме:

– низька читаність коду, що ускладнює вивчення мови новачками [10];

– динамічна типізація передбачає можливість появи помилок у коді навіть під час компіляції. Зокрема, банальні помилки можуть серйозно затягнути процес розробки [10];

– обмежена функціональність. Цілком логічно, що мова, створена в минулому столітті, не може покрити ту функціональність, що властива сучасним конкурентам [10];

– не найвища продуктивність, викликана динамічною типізацією [10].

Та звісно, у такої мові, як Objective-C є свої плюси:

– динамічна типізація. У деяких випадках це дійсно може стати ключовою перевагою. Наприклад, спрощує створення малих програм [10];

– документація і спільноти. Більше 20 років успішного застосування мови посприяли появі великої кількості якісних ресурсів і книг. Сьогодні будь-хто, хто бажає вивчити Objective-C, без зусиль знайде відповідь на питання, що цікавить на просторах інтернету [10];

– у порівнянні з C ++, та й багатьма іншими мовами того часу, Obj-C надає розробнику куди більше гнучкості [10];

– відносна простота синтаксису. Простота полягає в першу чергу в розумінні алгоритмів і того, як працює машина [10].

Останньою розглядаємою мовою, яка дає можливість програмувати під iOS, є Swift. Swift – це сучасна мова програмування загального призначення та багатопартійної парадигми, розроблена компанією Apple для побудови своїх пристроїв, що працюють на iOS, і всієї наступної екосистеми.

Програми можуть бути розроблені для роботи також на MacOS (для комп'ютерів Apple), watchOS (AppleWatch), tvOS (цифровий мультимедійний програвач Apple TV) і, що може бути дивно, що z/OS, яка живить комп'ютери IBM Mainframe. Наразі мова поширюється на ліцензію Apache, що робить її доступною для спільноти.

Це відносно новий проект, який був запущений у червні 2014 року, через сім років після запуску iPhone. На відміну від Objective-C, в якому для кожного класу необхідно створювати файли * .h і * .m з інтерфейсом і реалізацією відповідно, в Swift потрібно створити лише один файл * .swift, в якому містяться і інтерфейс, і реалізація. Це означає, що вихідних файлів

в проекті буде в 2 рази менше, що є плюсом [12]. Та звісно у Swift є свої дуже весомі переваги над конкурентами:

– читаність. Перевагою номер один у виборі Swift, ймовірно, є його чистий синтаксис, що полегшує читання та запис. Кількість рядків коду, необхідних для реалізації функції в Swift, набагато менше, ніж для Objective-C. Причина цього полягає в тому, що Swift видаляє багато успадкованих умовностей, таких як крапки з комою до кінцевих рядків або дужок, які оточують умовні вирази всередині операторів if / else. Іншою важливою зміною є те, що виклики методів не знаходяться усередині один одного, в результаті чого виникає безлад в дужках. Замість цього, виклики методів і функцій у Swift використовують розділення комою список параметрів у дужках. В результаті, код є чистішим з спрощеним синтаксисом. Код Swift більш нагадує звичайну англійську мову, що робить написання коду більш природним, дозволяючи розробникам витрачати набагато менше часу на пошук проблемного коду [11];

– технічне обслуговування. Неможливо, щоб Objective-C розвивався без того, щоб C розвивався першим. Натомість Swift не має цих залежностей, що значно полегшує їх обслуговування. C вимагає від програмістів підтримувати два файли коду, щоб поліпшити час і ефективність створення коду, який також переноситься на Objective-C. Однак Swift відкидає цю вимогу для двох файлів, поєднуючи заголовок Objective-C (.h) і файли реалізації (.m), в єдиний файл коду (.swift). У Objective-C вам доведеться вручну синхронізувати імена методів і коментарі між файлами. У Swift програмісти можуть витрачати більше часу на створення логіки та покращення якості коду, коментарів та функцій, які підтримуються [11];

– менше коду та менше спадщини. Користуючись Objective-C існує багато проблем, які викликають збій програм. Swift надає код, який є менш схильним до помилок через його вбудовану підтримку для маніпулювання текстовими рядками та даними. Крім того, класи не розділені на дві частини; інтерфейс і реалізація. Це скорочує кількість файлів у проекті в

два рази, що робить його набагато простішим у використанні. Swift в кінцевому рахунку вимагає менших зусиль кодування при написанні повторюваних операцій або викликаючи маніпуляцію рядками. Під час роботи з Objective-C вам потрібно буде об'єднати два рядки, які роблять його довгим. За допомогою Swift потрібно лише додати знак «+», щоб об'єднати два рядки [11];

– швидкість. Swift також забезпечує різні швидкісні переваги при розробці, в свою чергу, економить на витратах. Наприклад, складний тип об'єкта буде працювати на 3,9 разів швидше, ніж реалізація того ж самого алгоритму в Python. Це також краще, ніж Objective-C, що на 2,8 рази швидше, ніж версія Python. Його продуктивність наближається до C ++, що вважається найшвидшою арифметикою алгоритму розрахунку. У грудні 2014 року лабораторія дослідників опублікувала звіт про продуктивність Swift та C ++. Apple зрозуміла, що вони присвячені підвищенню швидкості, з якою Swift може запускати логіку додатків [11];

– Swift підтримує динамічні бібліотеки. Динамічні бібліотеки – це виконувані фрагменти коду, які можна зв'язати з застосунком. Ця функція дозволяє поточним додаткам Swift зв'язуватися з новими версіями мови Swift, оскільки вона розвивається з плином часу. Динамічні бібліотеки в Swift безпосередньо завантажуються в пам'ять, зменшуючи початковий розмір програми і в кінцевому рахунку збільшуючи продуктивність програми [11];

– Playgrounds заохочує інтерактивне кодування. Playgrounds – це функція, яка дозволяє програмістам перевіряти новий алгоритм без необхідності створювати цілі програми. Компанія Apple додала виконання Inline-коду до Playgrounds, щоб допомогти програмістам створити фрагмент коду або написати алгоритм, отримуючи зворотний зв'язок. Цей контур зворотного зв'язку може поліпшити швидкість, з якою код може бути написаний за допомогою візуалізації даних. Ігрові майданчики та Swift разом вказують на спроби Apple зробити розробку програм простішим та доступнішим [11].

Та що стосується недоліків, то можна сказити, що їх майже неіснує. Можна виділити такі невагомі недоліки:

– мова ще досить молода. Swift може бути найшвидшою та навіть бути найпотужнішою мовою в світі, але ще занадто молодою. Вона має багато питань, які необхідно вирішити. Зрештою, декілька років – це занадто мало часу для того, щоб будь-яка мова дозріла, навіть якщо вона є Swift. Більше того, Swift все ще має дуже обмежену кількість «рідних» бібліотек і інструментів: багато доступних ресурсів і інструментів, присвячених попереднім версіям Swift, марні з новими релізами [12];

– низька підтримки попередніх версій iOS, які були випущені Apple. Мову програмування Swift можна використовувати лише у програмах, націлених на iOS7 та пізніші. При цьому Swift не може використовуватися для застарілих проектів, що працюють на старих версіях операційної системи. Проте, за недавніми дослідженнями, менше 5% пристроїв Apple в даний час працюють на iOS 6 або раніше.

Нижче, на рисунку 3.1 зображен графік підримки версій iOS та версій мови Swift [13]. На цьому рисунку можна побачити, що більш ніж 80% мобільних телефонів працюють під версією iOS 10 та новіше.

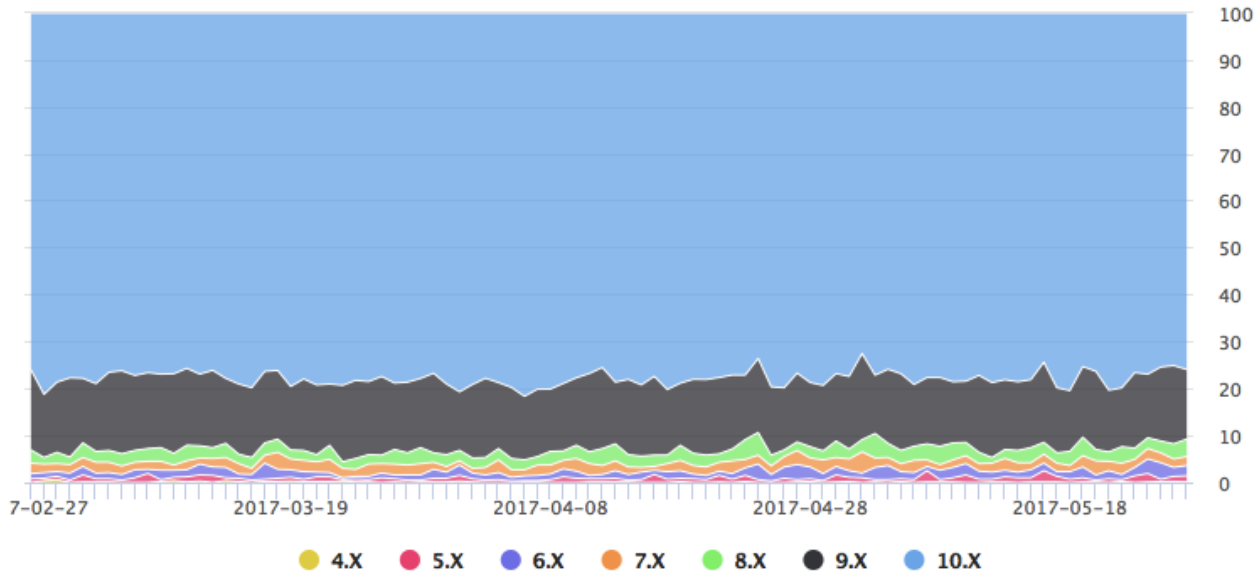

Рисунок 3.1 – Підримка версій iOS

3.1.1 Порівняння продуктивності мов програмування

Немаловажною частиною у будь-якої сучасної мові програмуванні є продуктивність (Performance). Швидкість – є дуже важливим показником у мобільних застосунках. Тому що кожна секунда прогрузки, відображення та чогось іншого впливає на користувача.

Перше порівняння продуктивності буде зроблено між Swift та React Native. Тому що саме ці дві мови є лідерами щодо швидкості роботи мобільних додатків.

За основу буде взятий мобільний застосунок, у якому є декілька дуже вимогливих до ресурсів екрана.

Для порівняння продуктивності буде виконано два тести. У першому тесті буде з'ясовано, скільки CPU ресурсів потрібно для виконання певного завдання. У другому тесті буде з'ясовано, скільки GPU ресурсів потрібно для виконання тих же завдань [5].

Перша функція вкладки «Profile» – це вхід через Facebook. У коді – це запит для зображення профілю, електронної пошти та імені, що повертається до програми з сервера Facebook. Як відомо, API запити через фреймворк Фейсбук вимагають дуже багато ресурсів. Крім API запиту потрібно буде розпарсити зображення профілю, що займе теж трохи ресурсів [5].

Завдання другої програми «To Do List» полягає в тому, щоб додати та видалити пункт «для виконання» зі списку. Ніяких запросів до серверу на цьому екрані не буде [5].

Завдання третьої вкладки «Page View» – відображеня звичайного екрану з декількома UI елементами [5].

Завдання четвертої вкладки «Maps» – після натискання на вкладку відобразити карту міста, яка автоматично збільшиться до поточної геопозіції користувача мобільного застосунку. Даний тест буде досить цікавим, так як будь-який фреймворк по роботі з картами потребує великої кількості ресурсів для відображення і обробки інформації [5].

Перше порівнення буде зроблено з кількістю використання CPU. CPU – це центральний процесор, функціональна частина комп'ютера, що призначена для інтерпретації команд. Цей тест є дуже важливий, тому що чим нижче показник використання CPU, тим вище показник автономності життя телефону від заряду батареї, відповідно, чим менше використання заряду батареї, тим краще для користувачів мобільного застосунку. На рисунку 3.2 зображен графік споживання ресурсів CPU між Swift та React Native на основі чотирьох екранів [5]. Вертикальна шкала означає відсоток використання ресурсів центрального процессору, та по горизонталі розташовані тести для двох мов.

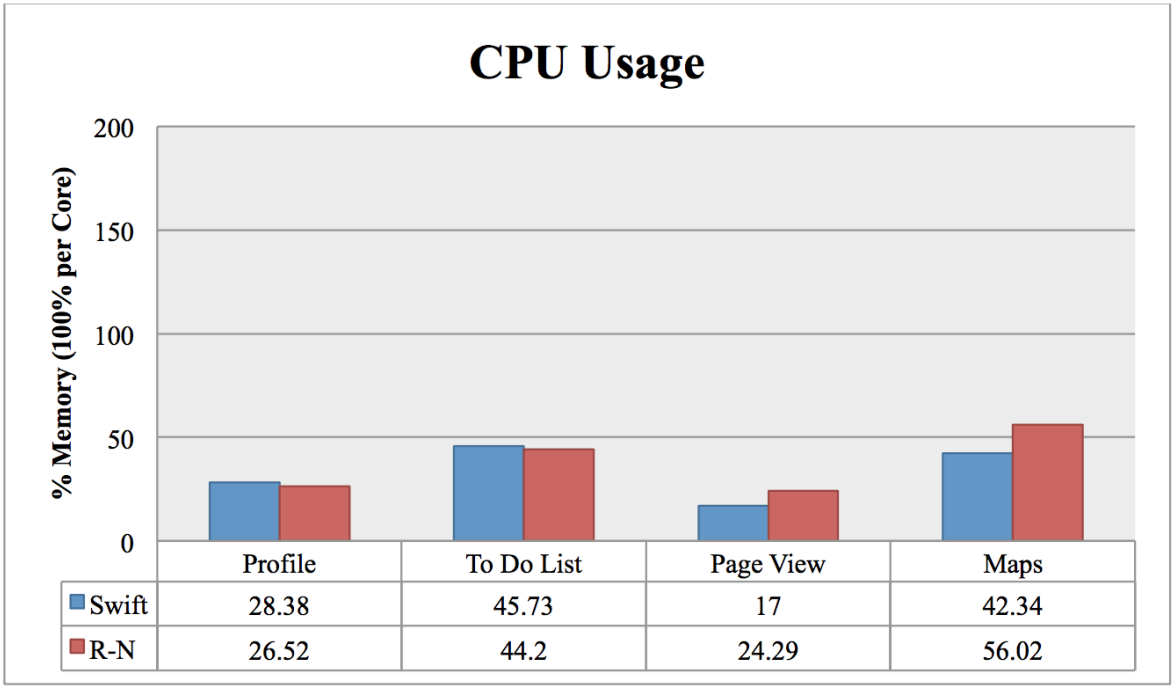

Рисунок 3.2 – Порівняння використання CPU між Swift та ReactNative

Маємо наступні результати:

– профіль: React-Native трохи виграє цю вкладку з ефективнішим використанням CPU на 1,86%. Під час виконання завдання і запису вимірювань спостерігався сплеск використання ЦП в той момент, коли тиснулась кнопка «Вхід з Facebook» [5];

– список завдань: React-Native також ледве виграє цю вкладку з ефективнішим використанням CPU на 1,53%. Під час виконання завдання і запису вимірювань спостерігалися сплески використання CP в той момент, коли додавався пункт до списку і коли його видаляли [5];

– перегляд сторінки: цього разу Swift вибив React-Native з 8,82% більш ефективним використанням CPU. Під час виконання завдання і запису вимірювань спостерігалися сплески використання CP в той момент, коли було початкове відображеня екрану [5];

– карти: Swift знову виграє цю категорію з 13,68% ефективнішим використанням CPU. Під час виконання завдання і запису вимірювань спостерігався сплеск використання CP в той момент, коли була натиснута кнопка на вкладці «Карти», що спонукало MapView знайти своє поточне розташування і виділити його синьою пульсуючою крапкою [5].

Отже, результат тесту CPU показав, що і Swift, і React Native перемогли у двух категоріях. Але якщо підрахувати проценти щодо використуваня процесорного ресурсу телефону, то можна побачити, що Swift використовував процесор на 17,58% більш ефективно.

Наступним тестом є тест використання GPU. GPU – це графічний процесор, який знаходиться на відеокарті, і призначений для обчислень із рухомою комою, чи аналогічними. Аналогічно до тесту з CPU, тест з використання GPU також є дуже важливим, тому що чим вище показники витрат GPU ресурсів процессору, тим швидше витрачає мобільний телефон заряд батареї. Слід також відмітити, що кількість FPS на екранах застосунку дуже важлива для користувача та його позитивного досвіду користування мобільним застосунком.

Будуть виконуватися всі ті ж завдання, що і для CPU тестів для кожної мови програмування. По осі Y знаходиться кількість кадрів в секунду, відповідно по осі X будуть знаходитися самі тести та результати щодо їх прохолження.

Максимальна кількість кадрів в даному тесті 60 в секунду. Кожну секунду, впродовж тесту, коли виконувалися завдання, за допомогою «Core

Animations» фреймворку буде записана інформація щодо кількості кадрів в секунду. Результати тесту зображені рисунку 3.3 [5].

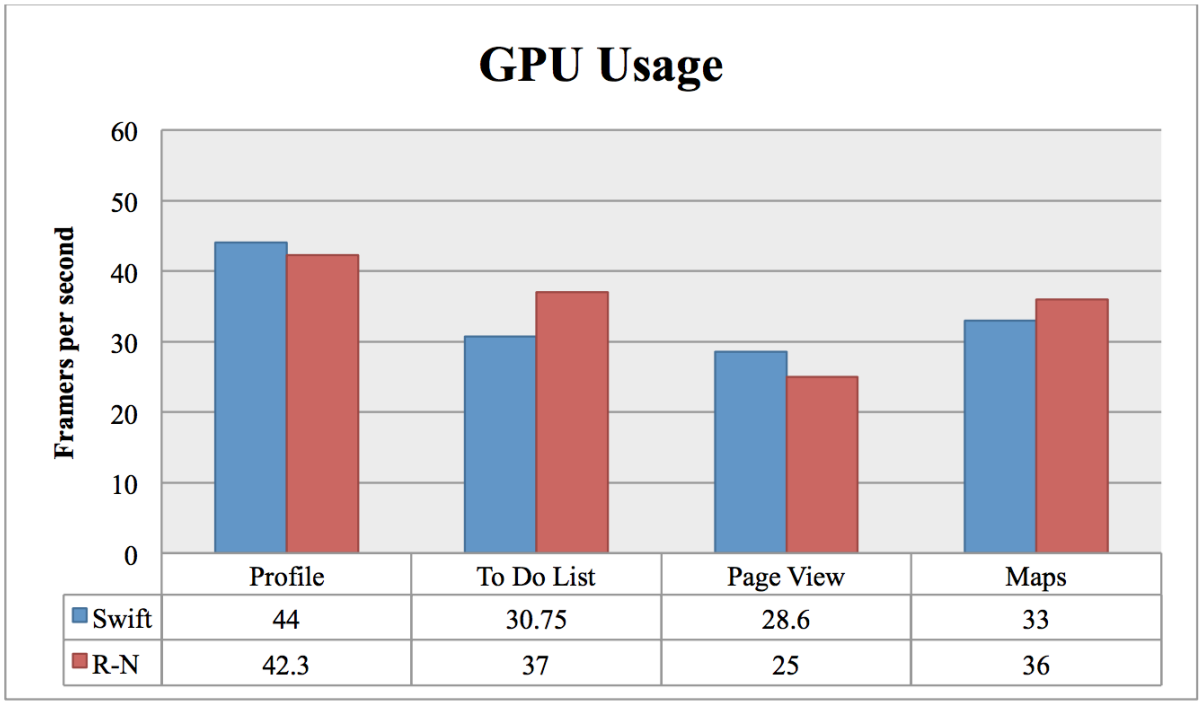

Рисунок 3.3 – Порівняння використання GPU між Swift та ReactNative

У ході тесту були отримані наступні результати:

– профіль: Swift трохи виграє цю вкладку, працюючи на 1.7 кадра / не сек вище, ніж React-Native. Під час виконання завдання і запису вимірювань спостерігався сплеск кадрів в секунду в той момент, коли натискалась кнопка «Вхід з Facebook»;

– список завдань: React-Native виграє цю категорію за 6.25 к / сек вище, ніж Swift. Під час виконання завдання і запису вимірювань спостерігався сплеск у кадрах / секундах у той самий момент, коли додавався пункт до списку і коли видалявся із списку;

– перегляд сторінки: Swift відпроцював краще, ніж React-Native у цій вкладці, працюючи на 3,6 кадрів на секунду вище. Під час виконання завдання і запису вимірювань, було помічино, що кадри / секунди знімаються до високих 50 кадрів на секунду, якщо швидко проскочувати

через сторінки;

– карти: React-Native виграє цю категорію, оскільки вона працює на 3 кадри / сек вище, ніж Swift [5].

Знову ж таки, Swift виграє дві вкладки та React-Native виграє дві вкладки. Однак React-Native виграє цю категорію загалом на 0.95 кадрів / с.

Та останнім тестом був тест на кількість використованної пам’яті у різних задачах. Першим завданням було вимірювання кількість споживаної пам'яті під час запуску програми. Обидві мови впоралися дуже добре, але є переможець. React Native використовував 57 мегабайт для того, щоб зробити перший запуск програми, а Swift в свою чергу лише 22 мегабайта. Другим завданням був запит до API. Результати були такими, що для запиту до API React Native знадобилося зарезервувати 63 мегабайта пам'яті, в той час як для мови Swift знадобилося всього лише 26 мегабайта [5].

Підводячи ітог, можна побачити, що в деяких місцях мова програмування Swift є більш швидкою та менш ресурсномісткою. Однак React Native також проявив себе досить добре у тесті з GPU. У останньому протистоянні переможець був Swift.

Варто відзначити, що в даних тестах не брав участь мова Objective-C, тому що Objective-c досить таки стара для таких порівнянь з молодими мовами програмування.

Також дуже важливим фактором є те, що використовувалася не остання версія мови Swift, а React Native був самою останньою версією. Та згідно з оновленнями у версії Swift 5.0, була дуже збільшена продуктивність і зменшені ресурсні витрати.

Виходячи з усіх плюсів, мінусів і результатів продуктивності, вибір мови програмування був зроблений в сторону Swift.

3.2 Проектування архітектури мобільного додатку

Архітектура мобільного застосунку – це неймовірно важливо. Чому варто подбати про вибір архітектури? Тому що якщо цього не зробити, то в

один прекрасний день, налагоджуючи величезний клас з десятками різних методів і властивостей, можна опинитися не в змозі знайти і виправити в ньому помилки. Природно, такий клас важко тримати в голові як єдине ціле, тому завжди будете губитися з уваги якісь важливі деталі [15].

У хорошій архітектури навіть є свої ознаки, такі як:

– розподіл обов'язків між сутностями з жорсткими ролями;
– тестованність. Зазвичай випливає з першої ознаки;
– простота використання і низька вартість обслуговування;
– безболісна модифікація поточного функціоналу.

Розподіл зменшує навантаження на мозок, коли розробник намагаємося з'ясувати, як працює та чи інша сутність. Чим більше людина буде розвиватися, тим краще мозок буде адаптуватися до розуміння складних концепцій. Але у всього є межа, і він досягається досить швидко.

Таким чином, найпростіший спосіб зменшити складність полягає в поділі обов'язків між декількома сутностями за принципом єдиної відповідальності [15].

Тестованність архітектури визначає, наскільки легко нам буде писати юніт-тести, а найчастіше – чи зможемо ми їх писати в принципі. А чи варто тестувати взагалі? Як правило, це не питання для тих, у кого завалилися юніт-тести після додавання нового функціоналу або після рефакторінга якихось тонкощів класу. Це означає, що тести вберегли розробників від виявлення проблеми на етапі запуску. Що могло б статися з застосунком вже на пристрої користувачів, а виправлення було б можливо тільки через тиждень [15].

В сьогоднішній день є дуже багато архітектурних патернів, таких як:

– MVC;
– MVVM;
– MVP;
– VIPER.

Для того, щоб полегшити собі життя, буде розглядатися три найпопулярніших патерна – MVC, MVVM, MVP.

Почнемо з MVC. У данному патерні Controller є посередником між View і Model, отже, дві останніх не знають про існування один одного. Model – це скорочення від Business Model, у данному розділу повинна бути деяка логіка, опис моделей сутністей.

У частині View знаходиться уся UI частина застосунку. View частина не повинна мати ніяку логіку, відповідає лише за відображення інформації та за взаємодію між користувачем та застосунком.

Controller має усю необхідну логіку, передає інформацію з Model до View частини. Паттерн MVC є гарним рішенням для невеликих мобільних застосунків, в яких неважливі показники покриття кода тестами.

Але у MVC є дуже багато недоліків. По-перше, проблема, яку викликає MVC, не відповідає парадигмі єдиної відповідальності і призводить до MVC. Так, MVC (Model-View-Controller) викликає MVC (Massive View Controller).

Згідно MVC, контролер є посередником між View і Model. Це змушує контролера обробляти численні обов'язки. Сучасні інтерфейси користувача розроблені таким чином, що на одному екрані передбачені певні особливості програми.

Один екран (View) складається з декількох функціональних підпрограм, для яких контролер екрану повинен інтегруватися з декількома різними складними модулями. Управління декількома функціональними підглядами та інтеграція з різними модулями – це велика відповідальність контролера. Контролер, в даному випадку, буде керувати життєвим циклом View, обробляти дії користувача, які в клієнт-серверному додатку можуть знадобитися для виклику сервера, обробки відповіді, обробки помилок, запуску таймерів оновлення екрану, обробки декількох викликів, прослуховування до сповіщень, обробляти орієнтацію View і так далі. Це робить код контролера більшим. Мабуть, керувати та розуміти код стає дуже важко. Ця проблема трохи вирішується, коли в iOS впроваджується концепція дитячих контролерів [15]. На рисунку 3.4 зображено схематично патер MVC.

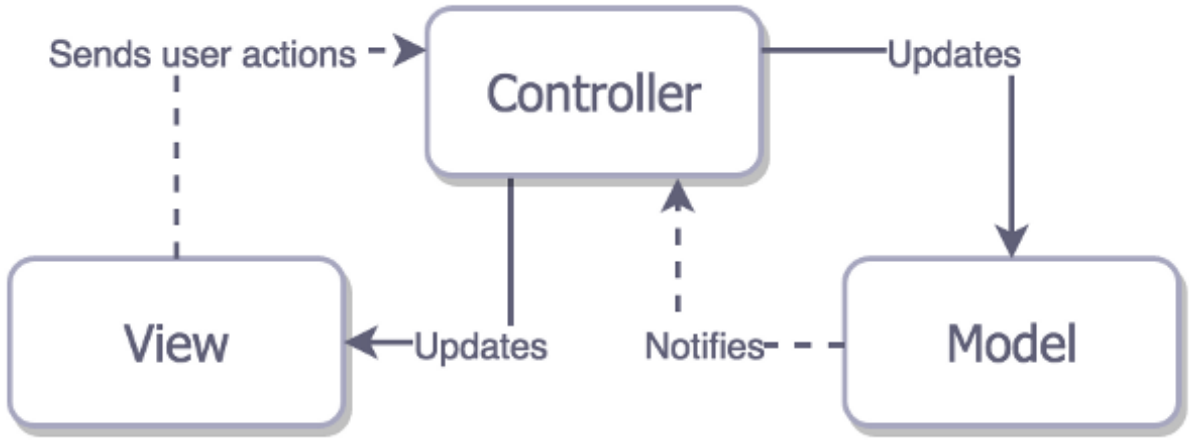

Рисунок 3.4 – Схематичний вид патерну MVC

По-друге, MVC ускладнює тестування модульних тестів. Завдяки прямому взаємозв'язку Controller-View, розробникам потрібно налаштувати та обиграти код таким чином, щоб виконувалася лише бізнес-логіка, щоб отримати дійсний результат тесту [15].

MVC все ще добре працює в багатьох випадках, коли користувальницький інтерфейс простіший, а Controller повинен поводитися з меншою відповідальністю, але він зазнає невдач у складних користувацьких інтерфейсах [15].

Наступним патерном є MVP. MVP – це розширення патерну MVC від компанії Apple, яке розшифровується як Model-View-Presenter, де відповідальність контролера перегляду розподіляється між View та Presenter.

У MVP на відміну від MVC, Controller не має ніякої логіки. View зберігає відповідальність за обробку інтерфейсу, тоді як Presenter має справу з фактичною бізнес-логікою (модель оновлення та інші речі).

Таким чином, модульне тестування простіше, оскільки всі одиниці тестового блоку записуються через Reporter, де обробка, пов'язана з переглядом, недоступна. Це знижує швидкість розробки, хоча реалізація Presenter і прив'язка його до шарів призводить до додаткової роботи. Та у лістингу 3.1 знаходиться приклад реалізації Presenter.

Лістинг 3.1 – Приклад частини Presenter у MVP патерні.

```
class TrafficLightPresenter {
    private let trafficLightService:TrafficLightService
    weak private var trafficLightViewDelegate :
TrafficLightViewDelegate?
    init(trafficLightService:TrafficLightService){
        self.trafficLightService = trafficLightService
    }
    func setViewDelegate(
        trafficLightViewDelegate:TrafficLightViewDelegate?){
            self.trafficLightViewDelegate =
trafficLightViewDelegate
    }
    func trafficLightColorSelected(colorName:(String)){
        trafficLightService.getTrafficLight(colorName:
colorName) { [weak self] traficLight in
self?.trafficLightViewDelegate?.displayTrafficLight(descriptio
n: traficLight.description)}
    }
}
```

Та на рисунку 3.5 схематично зображено патерн MVP.

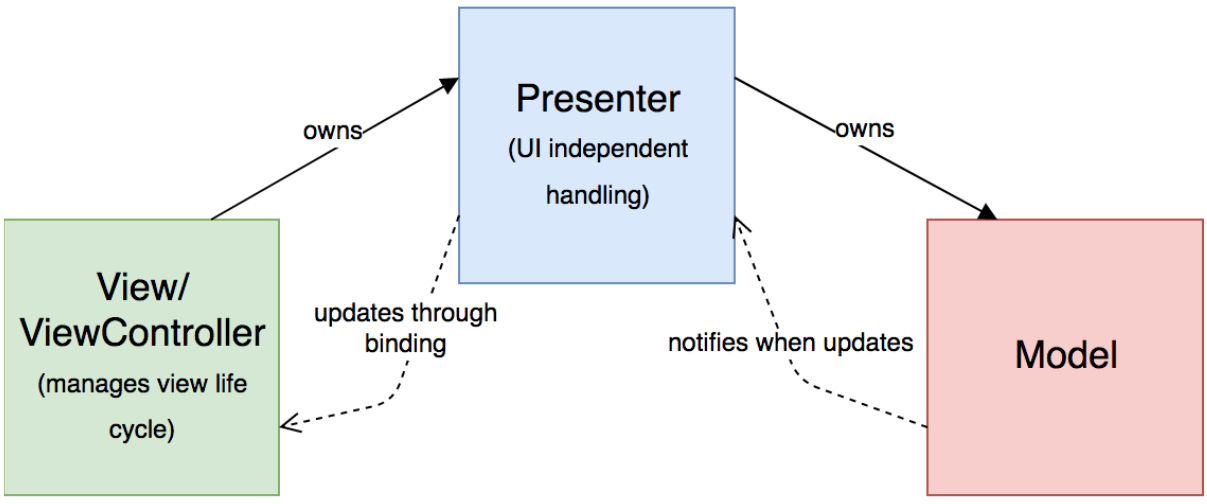

Рисунок 3.5 – Схематичний вид патерну MVP

З точки зору MVP, підкласи UIViewController насправді є View, а не Presenter. Ця різниця забезпечує чудову тестованість, яка йде за рахунок швидкості розробки, тому що ви повинні пов'язувати вручну дані і події саме між View і Presenter [15].

Останнім патерном є MVVM. Це останій патерн, що відзначається в сучасних додатках. Спочатку він був впроваджена компанією Microsoft в 2005 році, щоб полегшити програмування з керуванням подіями. Спільнота розробників для мобільних пристроїв виявила, що це також корисно для мобільних додатків. Він вийшов підходить для більшості проблем, які MVC може викликати. Як і MVP, MVVM також розглядає контролер подання як частину перегляду. Ми вже знаємо обов'язки View та Model з шаблону MVP. ViewModel – це новий компонент, який відповідає за обробку, не пов'язану з UI. У MVP перегляд конкретної логіки створення даних залишається з View. Presenter просто виступає посередником між View і Model і поділяє певну відповідальність з контролером перегляду. Через неналежне розповсюдження, контролер перегляду все ще потребує великої відповідальності в MVP. Отже, типовий функціонал в MVVM полягає в тому, що View отримує дані або з сервера, або з бази даних, надає їй ViewModel. ViewModel потім обробляє її і повідомляє через зв'язування. ViewModel зберігає складність створення репрезентативних даних з собою і, таким чином, знижує відповідальність зору (controller). ViewModel поділяє оновлення з View через прив'язку. Якщо говорити про реалізацію прив'язки в iOS, то тут не існує рідного підходу. Ми могли б реалізувати подібний механізм через KVO, делегатів або сповіщень. Це не працює так ефективно, як прив'язка до інших мов програмування. Щоб досягти обов'язкових результатів, розробники почали зв'язувати MVVM з реактивним підходом (кращим способом реалізації обов'язків, ніж через KVO або повідомлення). MVVM допомагає розподілити обов'язки і реактивний підхід допомагає прив'язувати ViewModel до View [15].

MVVM здається дуже гарною архітектурною схемою в теорії і добре працює і в практичних сценаріях. Це не найкращий вибір для простих

користувацьких інтерфейсів. У практиці паттерн MVVM, а саме ViewModel, виглядає наступним чином, як преведено у лістингу 3.2.

Лістинг 3.2 – Приклад частини ViewModel у MVVM патерні.

```
class ItemsViewController: UIViewController {
       @IBOutlet private weak var tableView: UITableView!
         private var viewModel: ItemsViewModel

         init(viewModel: ItemsViewModel) {
             self.viewModel = viewModel
             super.init(nibName: nil, bundle: nil)
         }

         override func viewDidLoad() {
             super.viewDidLoad()

             viewModel.fetch { [weak self] in
                 self?.tableView.reloadData()
             }
         }
     }

extension ItemsViewController: UITableViewDataSource {
          func tableView(_ tableView: UITableView,
numberOfRowsInSection section: Int) -> Int {
         return viewModel.items.count
     }
}
```

Виходячи з даного лістингу коду, можна побачити, що створюється екран за допомогою успадкування від UIViewController. Роль цього классу є View. На цьому екрані є таблиця, імовірно з даними. Для того, щоб відображати дані в міру необхідності, нам потрібен якийсь сигнал, який

буде повідомляти, коли варто перезавантажити таблицю і відобразити дані. У самому класі знаходиться init, який приймає параметром саму ViewModel. Далі в методі життєвого циклу йде звернення до ViewModel і викликається функція fetch. У даній функції є замикання (closure), яке поверне нам результат коли дані будуть готові. Далі у таблиці викликається метод перезавантаження і дані будуть відображені. Таким чином, View не знає нічого про дані, в чому і полягає суть MVVM [15].

Та на рисунку 3.6 зображено схематично як виглядає патерн MVVM.

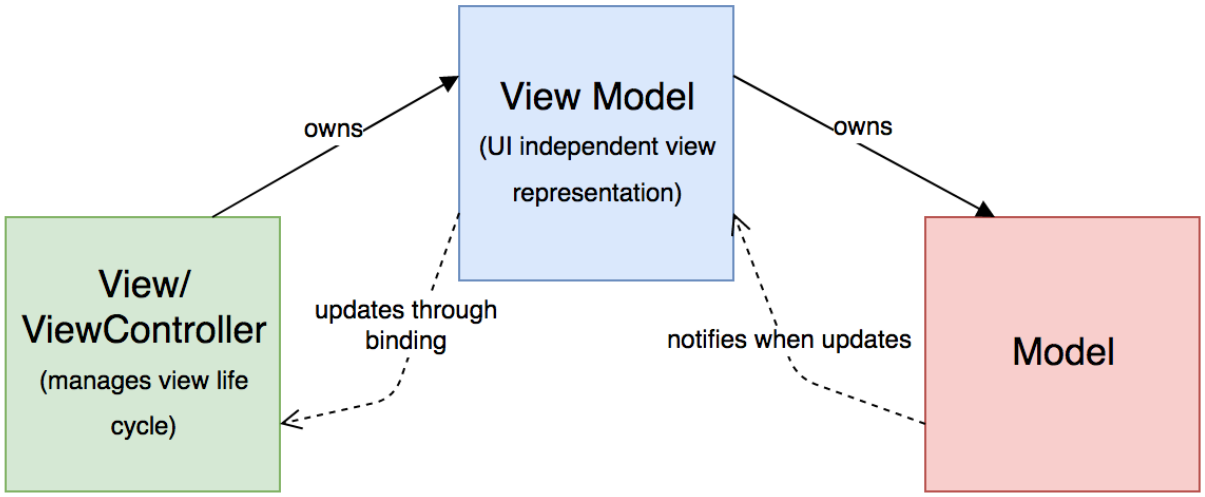

Рисунок 3.6 – Схематичний вид патерну MVVM

З цього випливає, що у центрі паттерна MVVM знаходиться ViewModel. ViewModel отримує повідомлення від Model, коли потрібно оновити дані. У свою чергу ViewModel оновлює усі UI елементи у View. Кожен незалежний View елемент має свою ViewModel. Соответсвенно кожна ViewModel має свою Model.

Роблячі висновки, дуже гарним архітектурнім рішенням є MVVM. Саме цей паттерн буде використовуватися у розробці мобільного додатка у який буде впроваджена система зі штучним інтелектом, яка буде розроблена за допомогою фреймворка від Apple – Create ML.

3.3 Практичні дослідження та налаштування моделі навчання

Як говорилося раніше, для розробки моделі машинного навчання буде використовуватися среда, надана компанією Apple, – CreateML. Головною ідеєю є розробка застосунку, який буде допомогати людям вивчати різноманітні боксерські удари, а саме: джеб, двойний джеб, хук та апперкот.

На першому етапі навчання буде відбуватися лише на двох ударах – джеб і подвійний джеб. Налаштування навчання зображено на рисунку 3.7.

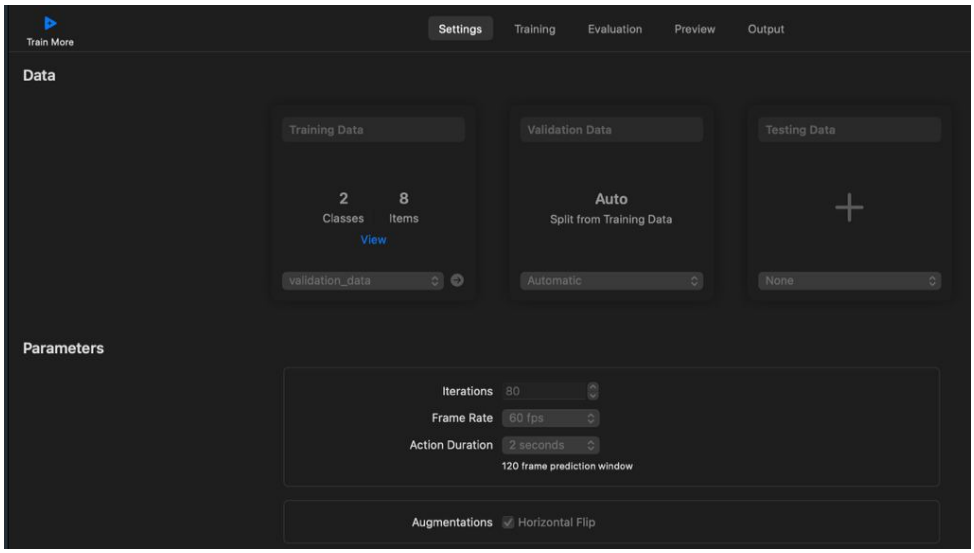

Рисунок 3.7 – Налаштування першого етапу тренування

Для кожного удару буде завантажено по 4 відео. Тренування буде проходити на 80 ітераціях, з тривалістю удару – 2 секунди і частотою 60 кадрів в секунду. Звичайно цих даних недостатньо для повноцінної системи, яка буде визначати удари.

Після закінчення тренування моделі у CreateML можна побачити результати цього тренування у вигляді графіку, та також зробити веріфікацію роботи моделі шляхом завантаження тестового відеофайлу. На рисунку 3.8 зображен результат тренування цієї моделі у вигляді графіку.

Рисунок 3.8 – Результат тренування першої ітерації моделі

Як помітно з рисунку 3.8, модель тренувалася на 80 ітераціях, та по результату точність моделі склала всього лише 61.7%. На рисунку 3.9 навчання на 140 ітераціях з додаванням відеоматеріалів для кожного класу.

Рисунок 3.9 – Продовження тренування модели на 140 ітераціях

Показник в 98% точності є вже досить таки високим. Але як говорилося раніше, це всього для двох ударів. На даному етапі можна перевірити те, як працює модель і наскільки точно. Для цього скористаємося вкладкою Preview в середовищі розробки CreateML. Для цього потрібно додати відео, за яким буде визначатися удари. На тестовому

відео чоловік б'є лівий прямий удар по мішку. Результат передбачення зображений на рисунку 3.10.

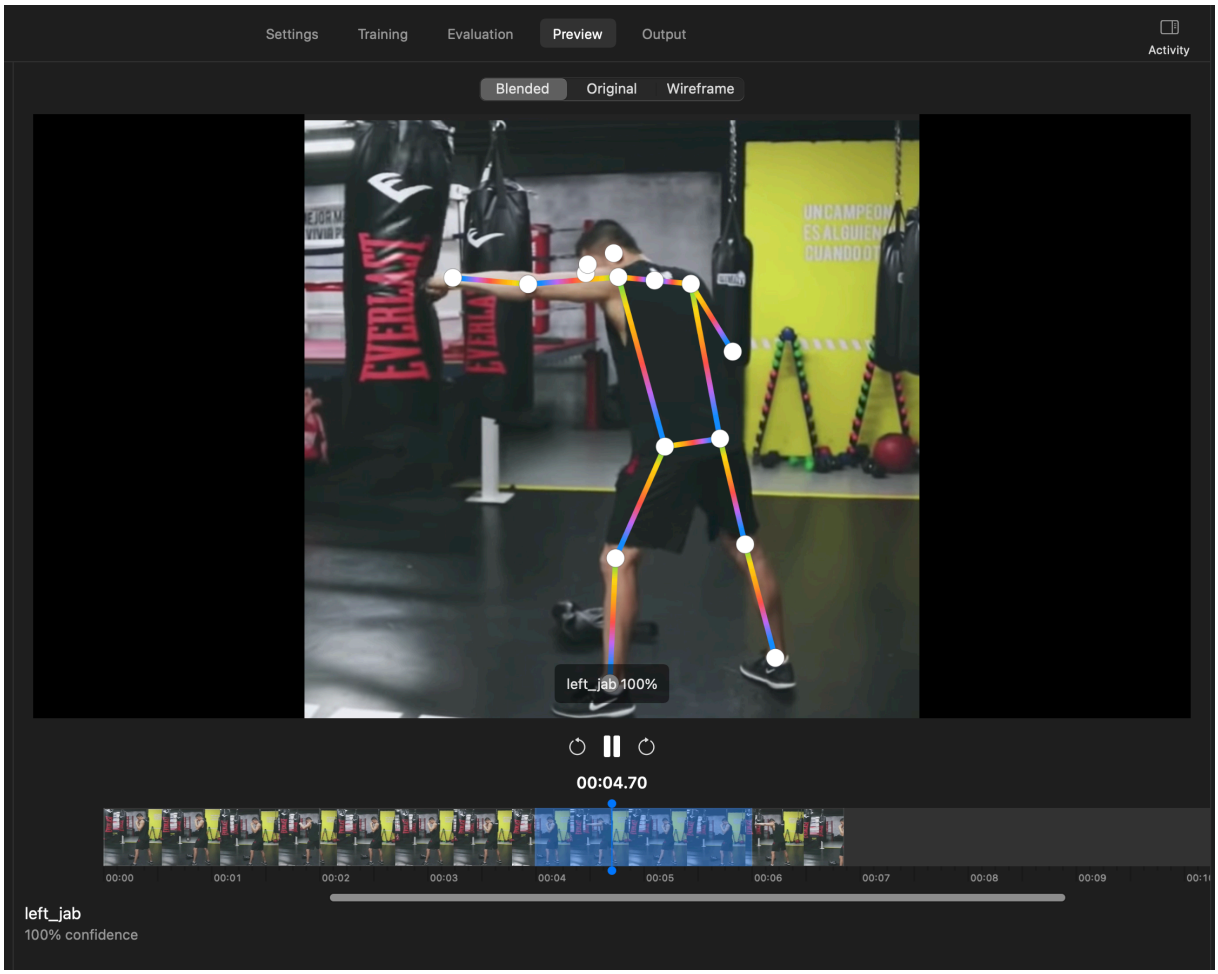

Рисунок 3.10 – Результат передбачення удару

На рисунку видно, як чоловік б'є лівий прямий удар. CreateML автоматично визначає точки тіла протягом всього відео. За результатами попереднього перегляду роботи моделі видно, що модель розпізнала лівий прямий удар зі 100% точністю.

Також CreateML має можливість відобразити тільки скелет людини, для цього потрібно переключити вкладу з Blended на Wireframe. Результат роботи Wireframe зображено на рисунку 3.11. Як видно з рисунку, CreateML дуже точно розставляє мітки на тілі людини, а також на обличчі, зокрема на носі, роті та вухах.

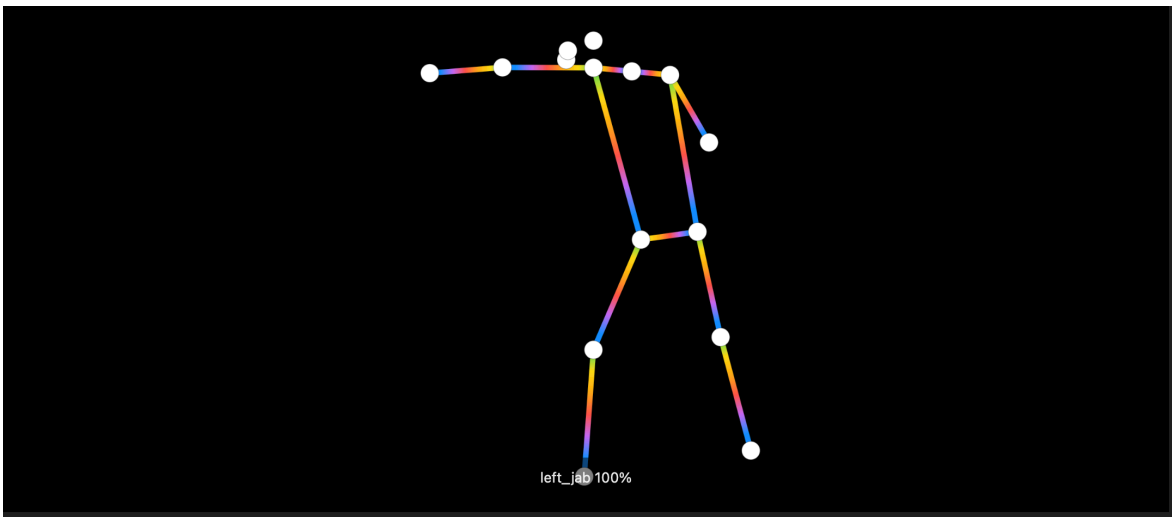

Рисунок 3.11 – Скелет людини у вкладці Wireframe

Наступним кроком є додавання нового удару. До раніше натренованої моделе буде додана нова папка з боковим ударом. Крім додавання нового класу також збільшимо кількість ітерацій до 300. Використовуватиметься 3 класу, в яких буде знаходиться по 3 тренувальних відео. Слід додати, що CreateML має можливсть автоматично генерувати дані для валідації точності праці моделі. Результат налаштувань тренування другої ітерації моделі зображений на рисунку 3.12.

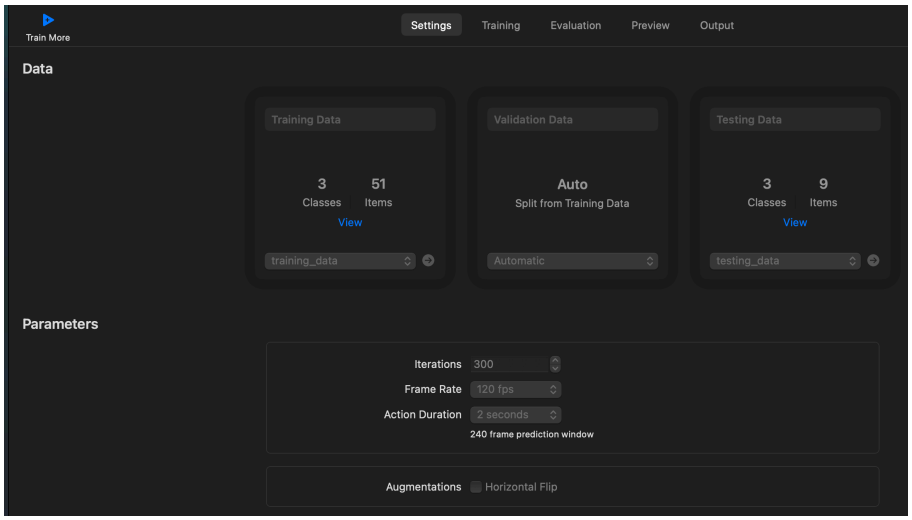

Рисунок 3.12 – Налаштування другої ітерації тренування моделі

Та на рисунку 3.13 зображено результат тренувань моделі на 300 ітераціях для 3 класів.

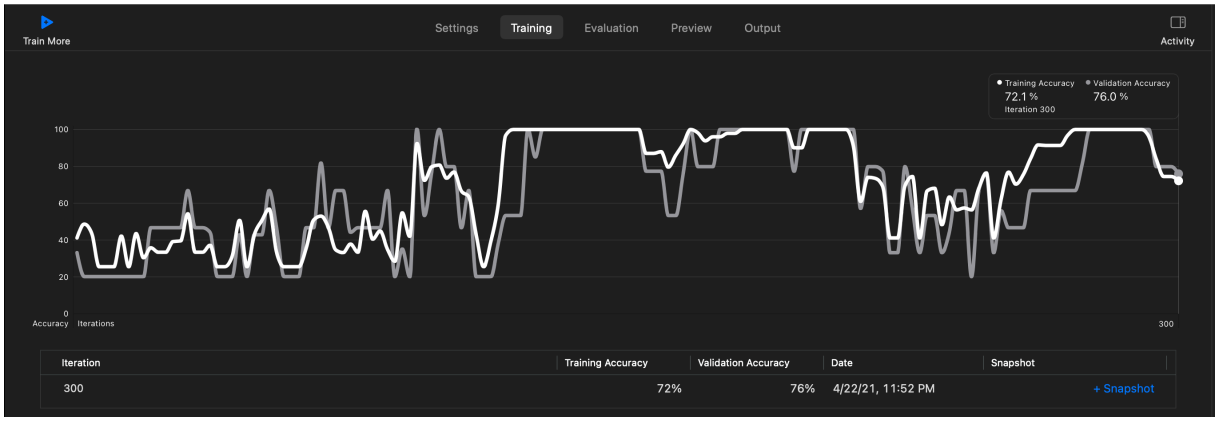

Рисунок 3.13 – Результат тренувань другої ітерації моделі

Як видно на рисунку вище, графік точності моделі почав вести себе досить нестабільно. Також впала точність моделі, з 98% до 76%. Якщо звернути увагу на точність розпізнання, то вона теж не може нас порадувати, так як становить всього 72%. Таку поведінку легко пояснити. Модель тренувалася з недостатньою кількістю вхідних даних. Незважаючи на те, що було 300 ітерацій для тренування моделі, показник у 17 тестових відео на кожен клас є замалим.

Як стверджує компанія Apple, чим більша кількість класів, тим більше треба даних для тренування моделі. Мінімально необхідне число тренувальних відео для тренування моделі становить 50 відео на кожен клас. Також слід відзначити, що скачки вниз, які відбувалися протягом тренування моделі, відбувалися через те, що велика кількість відео для тренування були зняті таким чином, що ног людини не було видно, та в відео, які були добавлені для того, щоб удосконалити алгоритм, людина як раз стоїть у весь ріст.

Спробуємо завантажити тестове відео для перегляду, наскільки точно система буде розпізнавати бічні удари. Результат зображений на рисунку 3.14.

Рисунок 3.14 – Результат передбачення бокового удару

Як видно з результату, модель не вгадала удар. Модель прийняла бічній за прямий удар. Таку поведінку теж просто пояснити. Людина на відео знаходиться в позі невідомої для моделі раніше. До того ж, вхідних даних недостатньо для точного визначення удару. Тому модель слід тренувати ще. Наступна ітерація моделі буде навчатися на 5000 ітераціях.

Останнім ударом, який залишається додати для даного прототипу є удар знизу або як його називають професіонали – аперкот. До кожного класу було додано ще кілька відео для тренування моделі. Результат налаштувань тренування моделі зображений на рисунку 3.15.

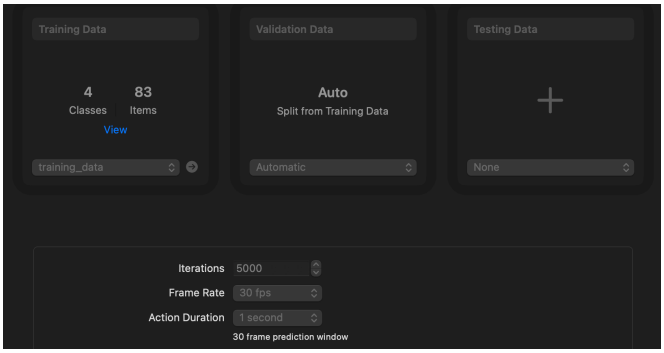

Рисунок 3.15 – Налаштування третьої ітерації тренування моделі

Після навчання моделі були отримані наступні результати, які зображені на рисунку 3.16.

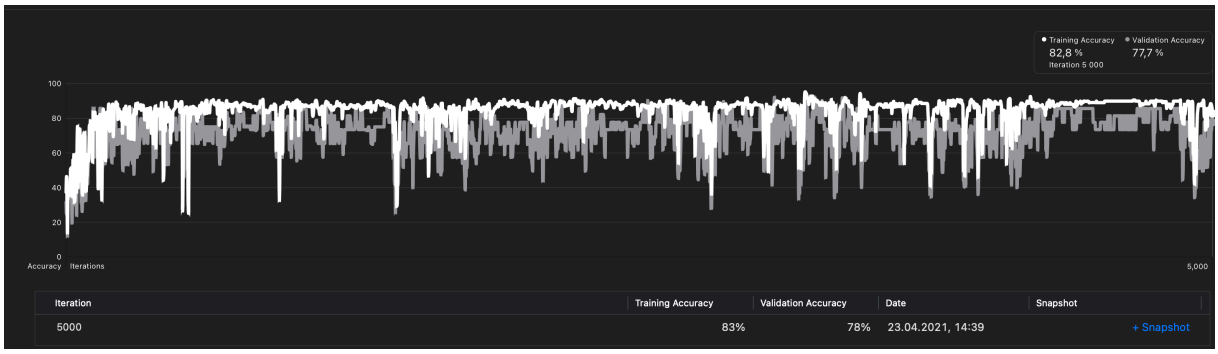

Рисунок 3.16 – Результат тренувань третьої ітерації моделі

Виходячи з отриманих результатів видно, що точність моделі стала вже набагато вище і це враховуючи те, що додався новий клас, а даних для навчання як і раніше недостатньо. Проте, результат точності моделі склав 82.8%. Якщо подивитися на вкладку Evaluation, то можна побачити, наскільки точно визначається кожен клас. Як видно з рисунку нижче, кожен клас має два поля – Precision та Recall.

Precision – це кількість справжніх позитивів, поділена на суму справжніх і помилкових позитивів.

Recall – це кількість справжніх позитивів, поділена на суму справжніх позитивних і помилково негативних.

Результат точності для кожного класу зображений на рисунку 3.17.

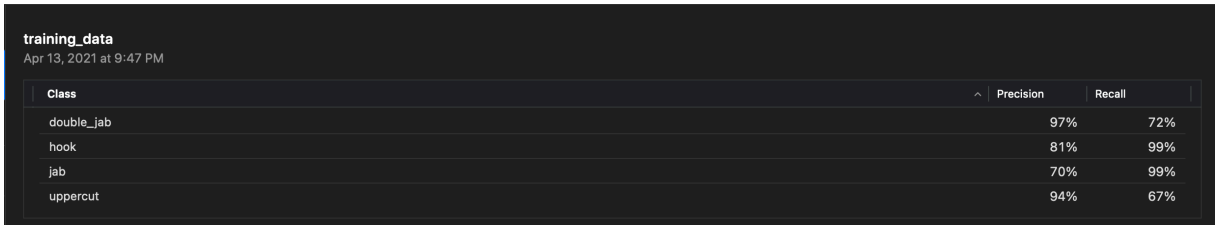

Рисунок 3.17 – Результат точності для кожного класу

Найнижчий коефіцієнт точності виявився у джеба. Він складає лише 70% precision та дивовижно великий коефіцієнт recall – 99%. Наступним є боковий удар з precision 81% та 99% recall. Та найбільша точність є в ударах подвійний джеб та апперкот, вона складає 94% та 97% відповідно з коефіцієнтом recall 67% та 72%. До кінця невідомим залишається причина такого низького показника recall для двух найбільш точних класів.

Тепер повернемося до минулого відео на якому був бічний удар, який модель так і не змогла розпізнати у зв'язку з нестачею даних та ітерацій для тренування. Завантажимо його ще раз та подивимося на результат. Цей результат зображено на рисунку 3.18.

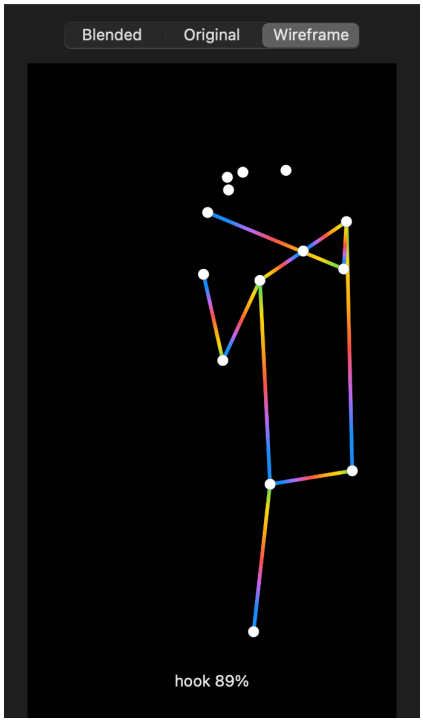

Рисунок 3.18 – Результат передбачення бокового удару

Як можна побачити з рисунку вище, модель вже відрізняє де знаходиться бічний удар. Точність в збігу становить 89%. На рисунку 3.19 можна побачити, як розподіляються решта відсотків і на які удари.

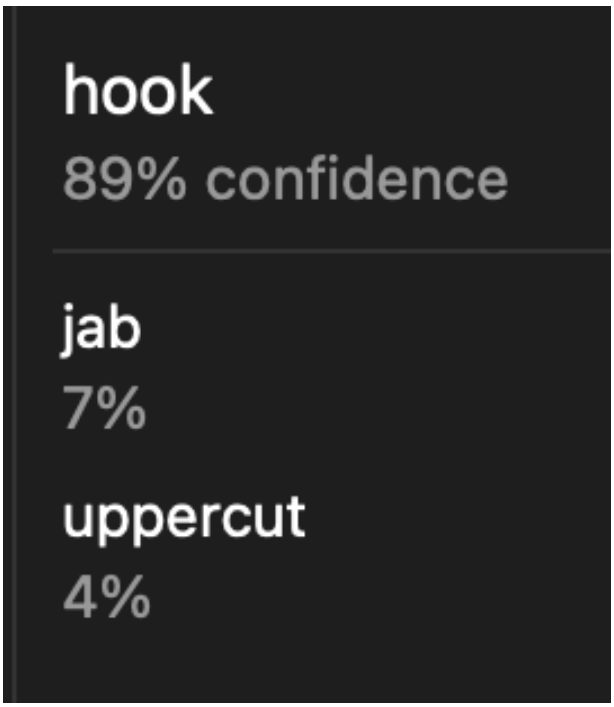

Рисунок 3.19 – Розподіл відсотків по передбачення удару

Тепер потрібно перевірити новий удар – аперкот. Для моделі цей удар є складним, тому що під різними кутами цей удар схож на прямий джеб. Результати будуть зображені на рисунку 3.20.

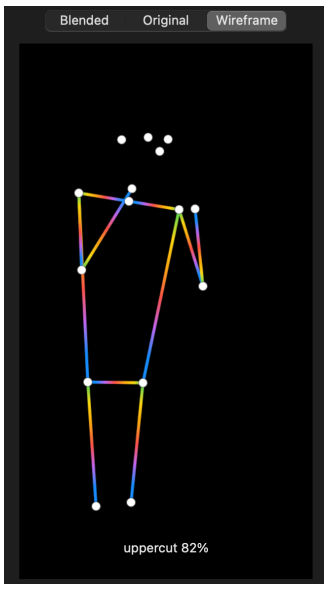

Рисунок 3.20 – Результат передбачення удару снизу

Система без проблем вгадала удар. Точність передбачення склала 82%. Варто відзначити, що залежно від кута, під яким стоїть людина, залежить точність по передбаченню.

Проблемою даної моделі є те, що коли людина стоїть і нічого не робить, модель намагається дати якийсь прогноз і робить його неправильно. Наприклад, людина стоїть в боксерській стійці, а модель видає пророкування, що це якийсь удар, що є неправильним. Для вирішення цієї проблеми потрібно додати додатковий клас, який буде позначатися як none, в якому набір даних, де люди просто рухаються, хитаються, та просто стоять на одному місці.

Другою проблемою цієї моделі є те, що модель не розуміє коли людина находиться у боксерської стійці. А враховуючи те, що модель розроблюється для розпізнання боксерьских ударів – це є досить критично. Тому окрім нового класу none буде додано новий класс з ім’ям other. Модель буде відрізняти 4 удари, боксерську стійку та бездіяльність

Для додаткового збільшення точності моделі буде завантажено ще кілька десятків відео для кожного класу. В цілому, для кожного класу буде завантажено по 40 відео даних. Навчання відбуватиметься все також на 5000 ітераціях, що займе не одну годину часу. Результат налаштувань тренування моделі зображений на рисунку 3.21.

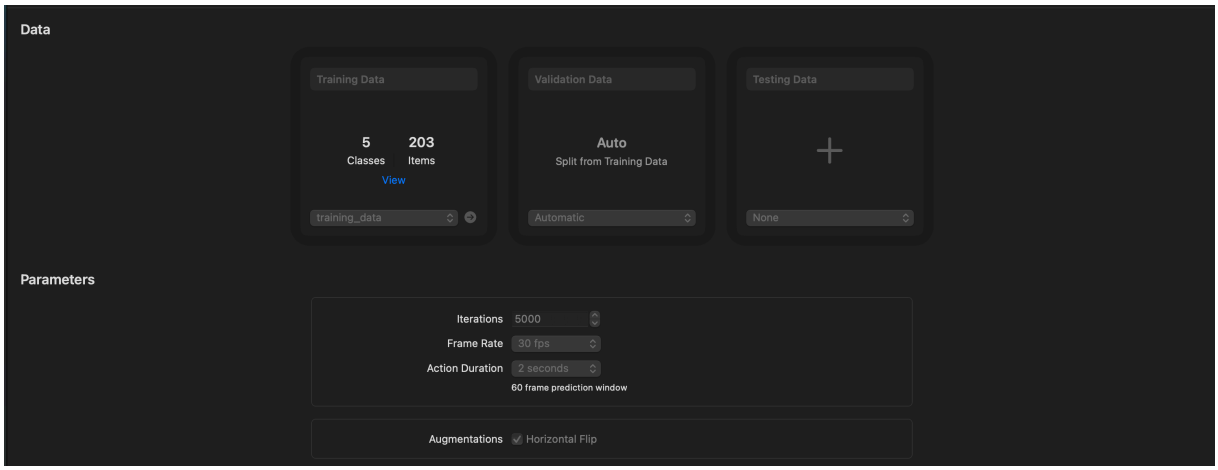

Рисунок 3.21 – Налаштування четвертої ітерації тренування моделі

Останні результати навчання моделі виглядають дуже оптимістичними. Результати тренування зображені на рисунку 3.22.

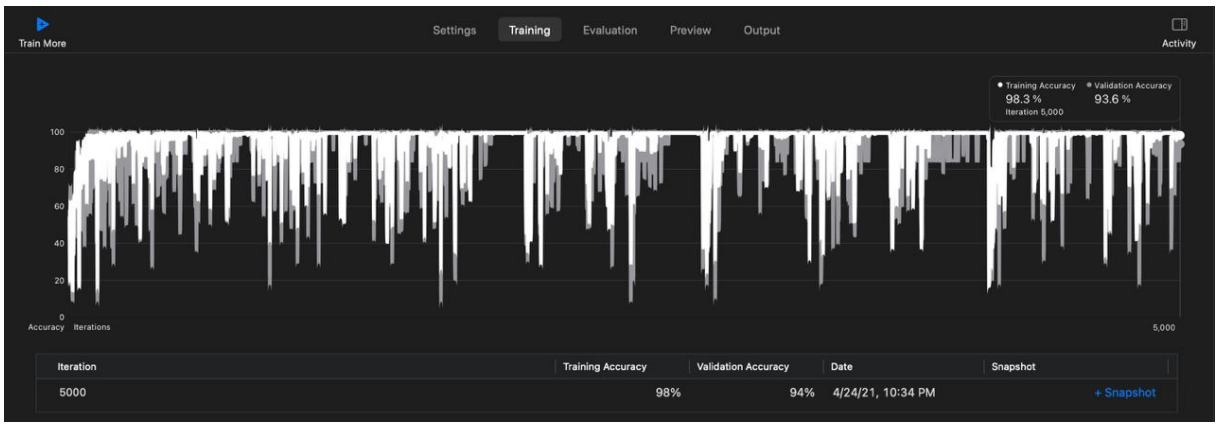

Рисунок 3.22 – Результат тренувань четвертої ітерації моделі

Виходячи з результатів видно, що модель вийшло досить таки точної, а саме 98.3%. Точність валідації вийшло теж дуже високою – 93.6%. Варто відзначити, що на графіку як і раніше видно стрибки – це все ті шуми, які позначені як клас none. Тому що їм потрібно більше ітерацій для навчання.

Якщо подивитися на точність для кожного класу, отримаємо наступні дані, які зображені на рисунку 3.23.

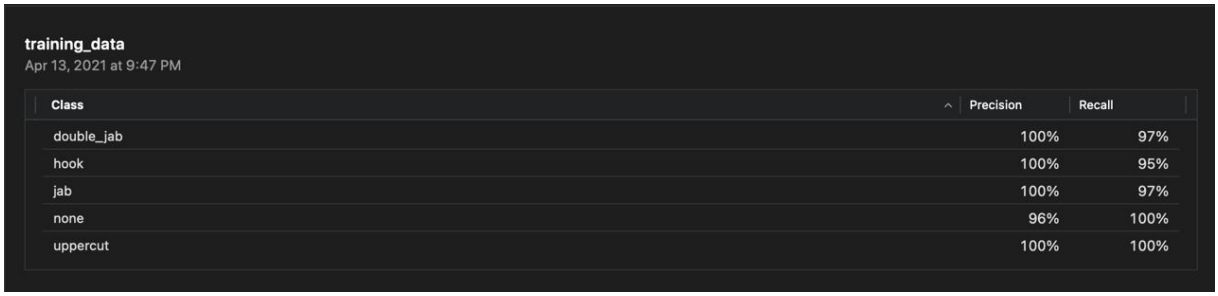

Рисунок 3.23 – Результат точності для кожного класу

Після навчання даної моделі вона буде готова до імплементації в мобільний застосунок. Як і описувалося раніше, користувач буде включати

потрібний режим тренування, після цього буде вмикатися фронтальна камера і буде відбуватися зчитування з камери в режимі реального часу.

Кожен з ударів навчився до точності 100%. Клас, в якому людина нічого не робить, навчився до точності 96%, що є теж дуже хорошим показником. Що стосується попереднього перегляду того, як працює модель, то тут все теж дуже добре. Для перевірки скористаємося складним відео, в якому людина спочатку знаходиться з руками, розташованими по швах, потім стає в стійку і через деякий час завдає бічний удар та аперкот. Результат розпізнавання зображений на рисунках 3.24 та 3.25.

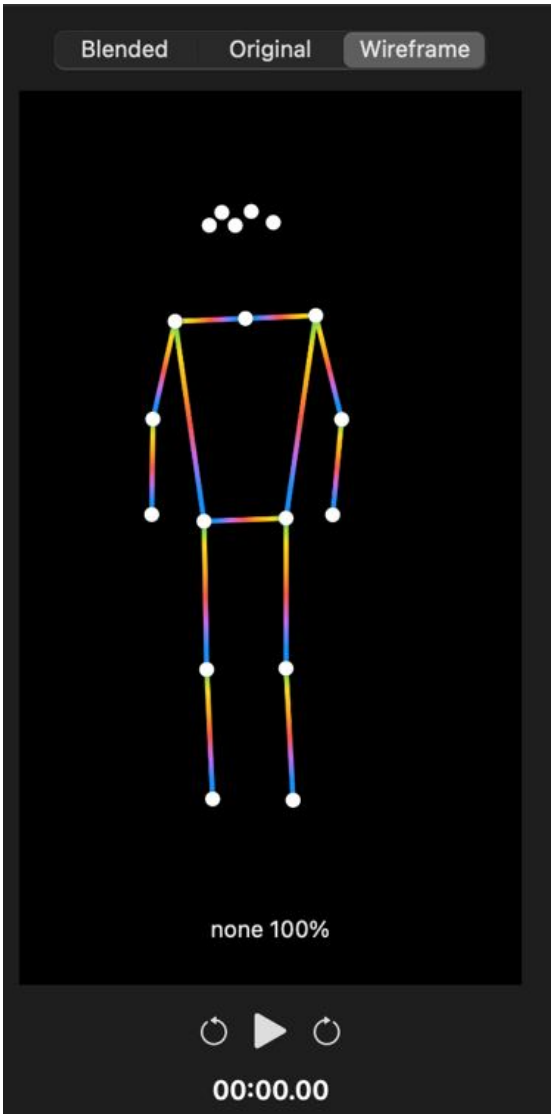

Рисунок 3.24 – Розпізнування классу none, руки по швах

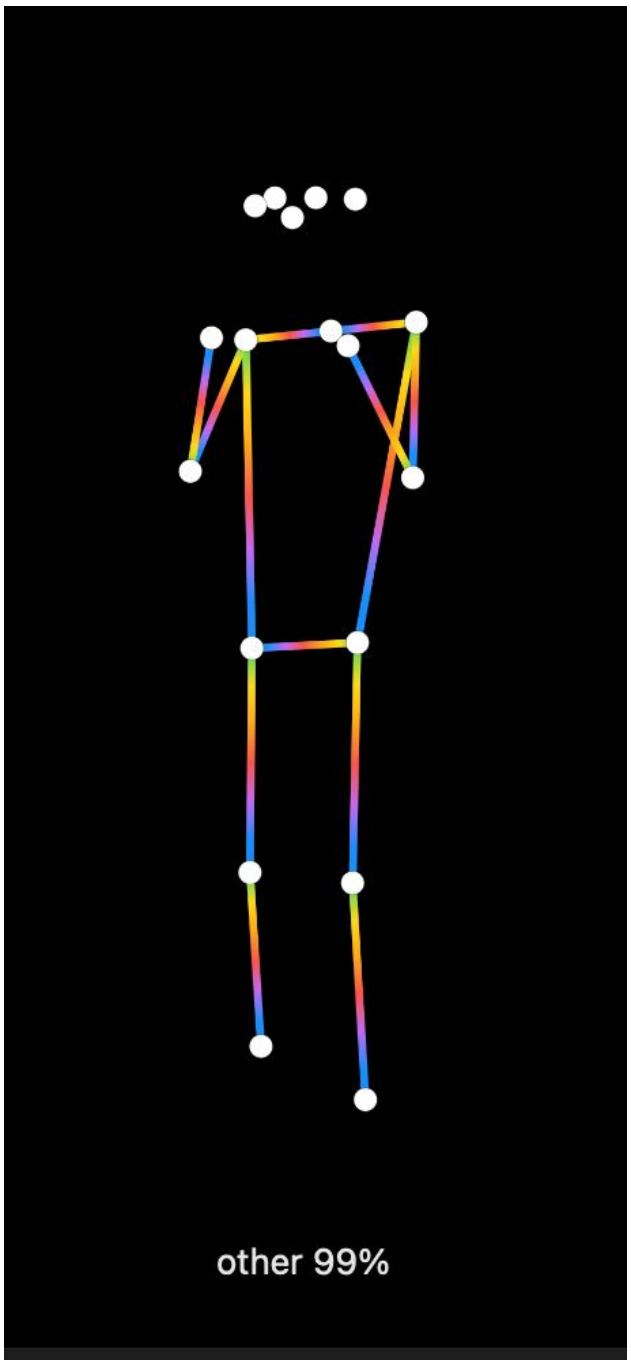

Рисунок 3.25 – Розпізнування классу other, боксерська стійка

Як видно на рисунках 3.24 та 3.25, модель з точністю 100% вгадує, що нічого не відбувається в двох випадках, коли людина просто перебуває в стоячому положенні і з руками по швах, а також коли людина знаходиться в боксерській стійці.

На даному етапі модель повністю готова до експортування та додаванню в проект мобільного застосунку.

Фінальна модель має всього навсього 4.1МБ. Вихідна конфігурація моделі зображена на рисунку 3.26. Використовувати дану модель можна буде на любому девайсі під керуванням iOS 13.0 та новіше. Також на рисунку видно додаткову інформацію що до моделі та її конфігурації.

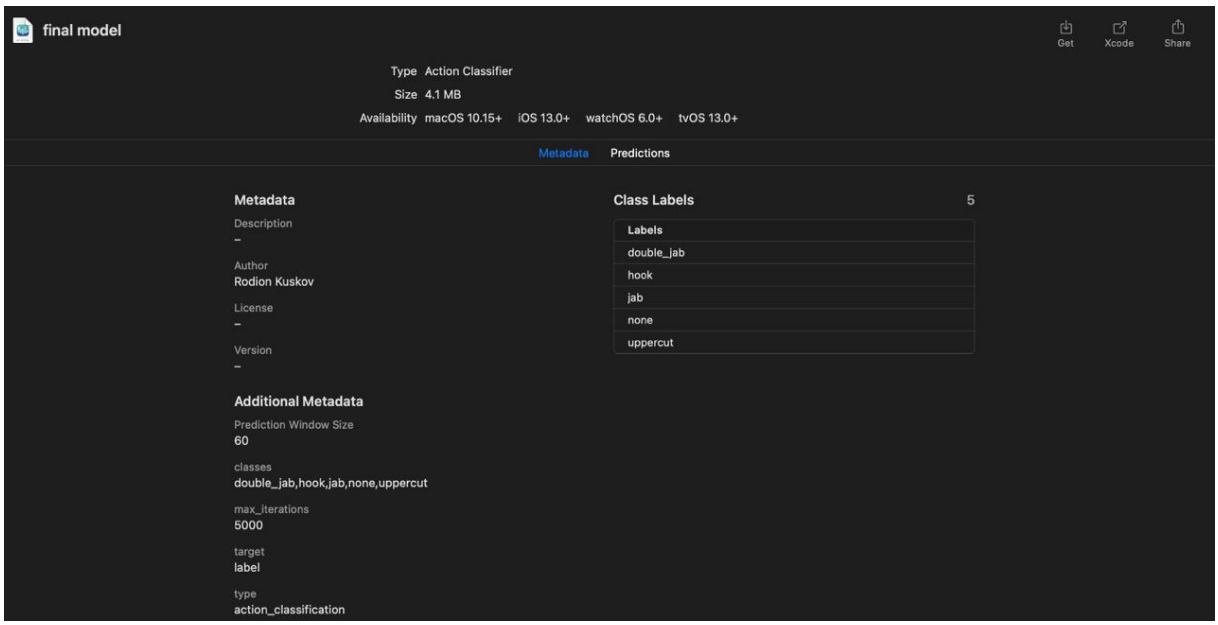

Рисунок 3.26 – Конфігурація та інформація фінальної моделі

Також слід зазначити, що при необхідності додати нові удари, це буде зробити досить таки просто. CreateML має всі необхідні механізми для цього, в тому числі і механізми для оптимізації моделі.

3.4 Розробка прототипу мобільного додатку

Як говорилося раніше, мобільний застосунок буде розроблений під платформу iOS за допомогою мови програмування Swift. Розпізнування рухів людини буде відбуватися у реальному часу за допомогою камери на айфоні. Для того, щоб зробити кастомну камеру треба скористуватися фреймворком для мови Swift – AVFoundation. Кофігурація сесії для камеризнаходиться у лістингу 3.3 та 3.4.

Лістинг 3.3 – Конфігурація сессії для камери телефону

```
private func configureCaptureSession() ->
AVCaptureVideoDataOutput? {
        disableCaptureSession()
        guard isEnabled else { return nil }
        defer { enableCaptureSession() }

        captureSession.beginConfiguration()

        let modelFrameRate = BoxModel.frameRate
        let input =
AVCaptureDeviceInput.createCameraInput(position:
cameraPosition,

frameRate: modelFrameRate)

        let output =
AVCaptureVideoDataOutput.withPixelFormatType(kCVPixelFormatTyp
e_32BGRA)
        let success = configureCaptureConnection(input,
output)
        return success ? output : nil
    }

     //MARK:- Guard if can add inputs
    private func enableCaptureSession() {
        if !captureSession.isRunning
{ captureSession.startRunning() }
    }

    private func disableCaptureSession() {
        if captureSession.isRunning
{ captureSession.stopRunning()
}
```

Лістинг 3.4 – Конфігурація з’єднання сесії для камери телефону

```
    private func configureCaptureConnection(_ input:
AVCaptureDeviceInput?,
                                            _ output:
AVCaptureVideoDataOutput?) -> Bool {

    // MARK:- Outputs
            guard let input = input else { return false }
            guard let output = output else { return false }


captureSession.inputs.forEach(captureSession.removeInput)

captureSession.outputs.forEach(captureSession.removeOutput)

    //MARK:- Guard if can add inputs
            guard captureSession.canAddInput(input) else {
                return false
            }

            guard captureSession.canAddOutput(output) else {
                return false
            }

    //MARK:- Adding inputs and outputs
            captureSession.addInput(input)
            captureSession.addOutput(output)

            guard captureSession.connections.count == 1 else {
                let count = captureSession.connections.count
                print("The capture session has \(count)
connections instead of 1.")
                return false
            }
```

Продовження лістингу 3.4

```
            // MARK:- Connection
                guard let connection =
captureSession.connections.first else {
                    print("Getting the first/only capture-session
connection shouldn't fail.")
                    return false
                }

                if connection.isVideoOrientationSupported {
                    connection.videoOrientation = orientation
                }

            // MARK:- Mirroring

                if connection.isVideoMirroringSupported {
                    connection.isVideoMirrored = horizontalFlip
                }


        }
                if connection.isVideoStabilizationSupported {
                    if videoStabilizationEnabled {
                        connection.preferredVideoStabilizationMode
= .standard
                    } else {
                        connection.preferredVideoStabilizationMode
= .off
                    }
                }

                output.alwaysDiscardsLateVideoFrames = true

                return true
            }
```

Після того, як мобільний застосунок має доступ до камери, наступним кроком потрібно зв’язати камеру та натренеровану модель, яка буде робити предбачення щодо ударів, які б’є людина.

Інформацією, яка потрібна для моделі – це SampleBuffer. Схема як працює модель з камерою зображена на рисунку 3.27 та на рисунку 3.28. На цьому рисунку помітно, що спочатку іде захват з камери телефону, наступним кроком іде зйомка відео, далі ланцюг обработки відеофайла, який передається до контроллеру, та контроллер получає інформацію щодо руху люди.

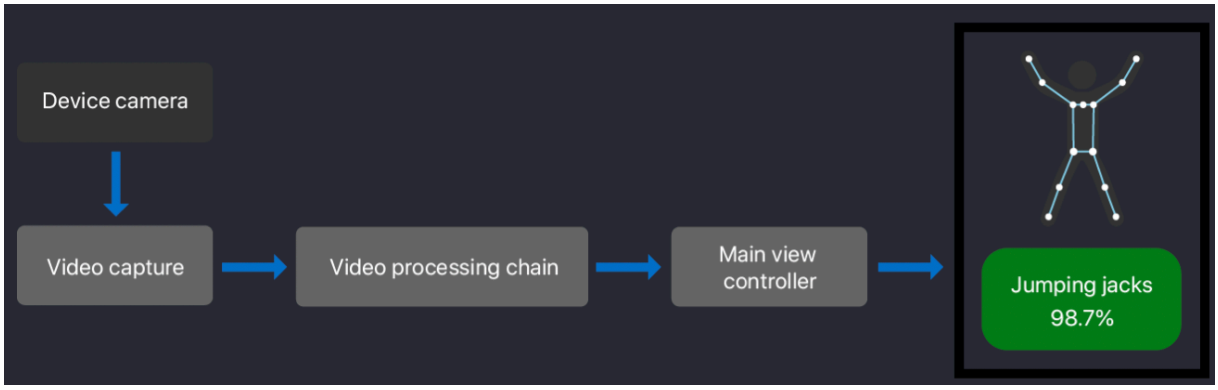

Рисунок 3.27 – Схема праці між камерою та моделью

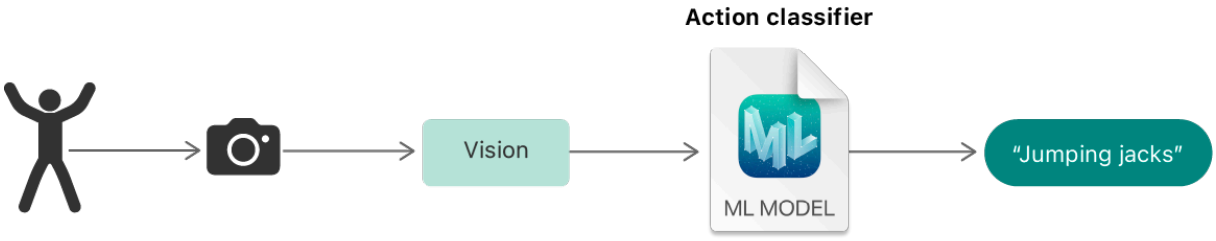

Рисунок 3.28 – Схема праці між камерою та моделью

Останнім шагом є створення UI часини, за допомогою якої користувач зможе вибрати потрібний для нього режим тренування.

Результат розробки головного екрану з режимами тренування зображен на рисунку 3.28.

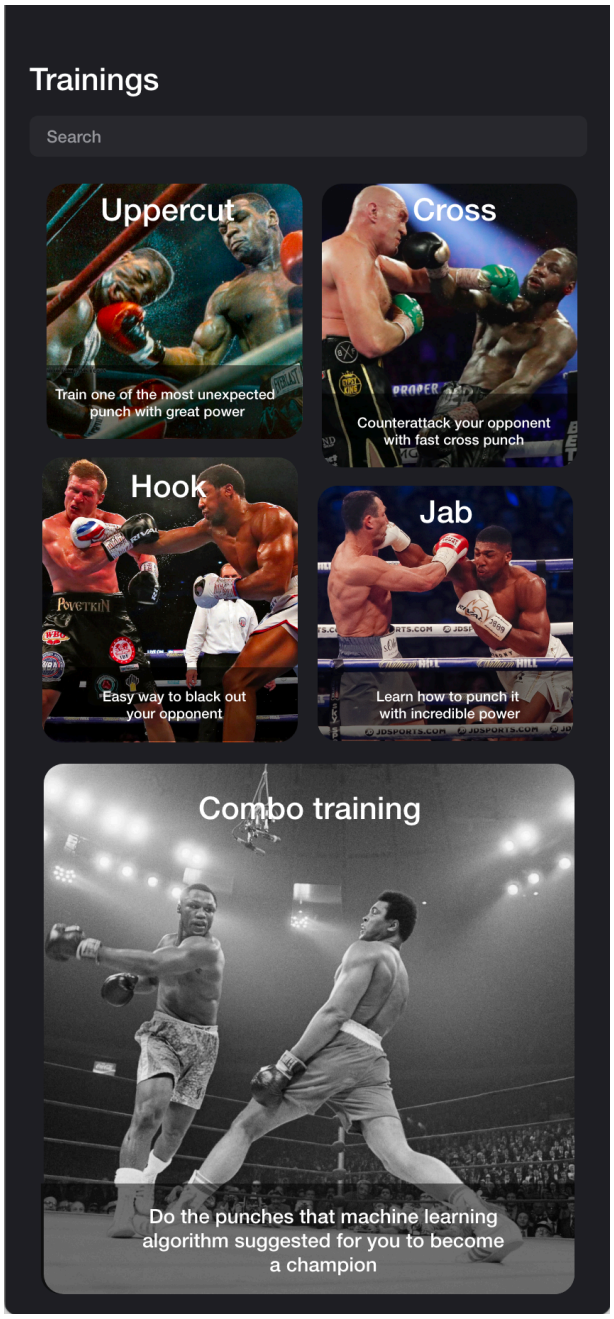

Рисунок 3.28 – Результат розробки головного екрану додатку

На етапі прототипу було додано тренування апперкоту, кросс удару, бокового, джебу та комбіноване тренування, где система пропонує удари. Комбіноване тренування показує на екрані користувача анімацію удару, та користувач намагається відтворити цей удар.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи була проаналізована предметна область за визначенням рухів людини, які обчислюються за допомогою штучного інтелекту і машинного навчання. Були знайдені чотири відповідних рішення. Одним з головних завдань був аналіз готових рішень в області аналізу та класифікації рухів людини. Було проведено велику кількість тестів для кожного рішення, шляхом завантаження різних відеофайлів з різними типами освітлення та кутом нахилу. Зваживши переваги і недоліки кожної, вибір пав на рідний фреймворк від компанії Apple – CreateML.

Також у ході виконання роботи були досягнені дві дуже важливі задачі – створити модель, яка буде прогнозувати рухи людини у реальному часу, та імплементація цієї моделі до мобільного додатку під платформою iOS.

Кінцева модель тренувалася багато разів. У остаточному варіанті точність моделі зкалала 99%, що є дуже великим показником.

Окрім теоретичних показників точності, модель була протестована у тому ж CreateML фреймворку, за допомогою складного відеофайлу, у якому людина б’є багато різноманітних ударів та просто стоїть у боксерьский стойці.

Також після успішного тренування моделі з розпізнування рухів людини у контексті бокса, був розроблен мобільний застосунок, у який була інтегрована навчена модель. Був розроблен головний екран мобільного додатку, екран камери та допоміжні UI елементи для відображення інформації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Human Action Classification. URL: https://github.com/dronefreak/human–action–classification (Last accessed: 24.04.2021);

2. NTU–X. URL: https://github.com/skelemoa/ntu–x (Last accessed: 24.04.2021);

3. Human Pose Estimation Benchmarking and Action Recognition. URL: https://github.com/ChengeYang/Human–Pose–Estimation–Benchmarking–and–Action–Recognition (Last accessed: 24.04.2021);

4. Create ML в iOS. URL: https://habr.com/ru/post/490936 (Last accessed: 24.04.2021);

5. Comparing the Performance between Native iOS (Swift) and React-Native. URL: https://medium.com/the-react-native-log/comparing-the-performance-between-native-ios-swift-and-react-native-7b5490d363e2 (Last accessed: 24.04.2021);

6. Create ML. URL: https://developer.apple.com/documentation/createml (Last accessed: 24.04.2021);

7. React Native vs. Swift – Which One To Pick When Building An iOS App? URL: https://ideamotive.co/blog/react-native-vs-swift-which-one-to-pick-when-building-an-ios-app/ (Last accessed: 24.04.2021);

8. Awesome Machine Learning demos. URL: https://github.com/applicable-ml/awesome-ml-demos-with-ios (Last accessed: 24.04.2021);

9. Плюсы и минусы платформ React Native и Real Native: сравниваем приложения. URL: https://dou.ua/lenta/articles/react-vs-real/ (Last accessed: 24.04.2021);

10. Macbug Objective-C. URL: http://macbug.ru/cocoa/objc (Last accessed: 24.04.2021);

11. Яблочное сравнение: Swift vs Objective-C. URL: https://geekbrains.ru/posts/swift_vs_obj_c (Last accessed: 24.04.2021);

12. Swift: проблемы и перспективы. URL: https://habr.com/ru/post/227243/ (Last accessed: 24.04.2021);

13. Eight Advantages of Using Swift for iOS Development. URL: https://clearbridgemobile.com/8-advantages-choosing-swift-objective-c-ios/ (Last accessed: 24.04.2021);

14. The Good and the Bad of Swift Programming Language. URL: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-swift-programming-language/ (Last accessed: 24.04.2021);

15. iOS design patterns – Part 1 (MVC, MVP, MVVM). URL: https://medium.com/swlh/ios-design-patterns-a9bd07818129 (Last accessed: 24.04.2021);

16. Build an Action Classifier with Create ML. URL: https://developer.apple.com/videos/play/wwdc2020/10043 (Last accessed: 24.04.2021);

17. iOS Swift: MVP Architecture. URL: https://saad-eloulladi.medium.com/ios-swift-mvp-architecture-pattern-a2b0c2d310a3 (Last accessed: 24.04.2021).