МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки
Факультет Комп'ютерних наук
Кафедра Програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
рівень вищої освіти

Дослідження ML моделей для завдань розпізнавання образів на iOS

Виконав:
студент 2 курсу, групи ІПЗм-21-2
Хрипунов І. М.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова

Керівник доц. Каук В.І.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ З.В. Дудар

2023 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти другий (магістрський)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва)

Тип програми освітньо-наукова програма

Освітня програма Інженерія програмного забезпечення
(код і повна назва)

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента *Хрипунова Іллі Михайловича*
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження ML моделей для завдань розпізнавання образів на iOS* затверджена наказом по університету від « *29 » 03 20 23* р. № *302ст.*

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

3. Вихідні дані до роботи *в дослідження необхідно передбачити: використання ML моделей для розпізнавання образів у форматі CoreML та TensorFlowLite в межах мобільного додатку. Використовувати програмне забезпечення Mac OSX та iOS, середовище розробки Xcode*

4. Перелік питань, що потрібно опрацювати в роботі *вступ, аналіз предметної області та постановка задачі, моделювання мобільного додатку, реалізація мобільного додатку, порівняння зручності та якості інтеграції, висновки, перелік джерел посилання.*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів	Примітка
1	Аналіз предметної галузі	29.03 – 31.03	виконано
2	Розробка логічної та фізичної моделі бази даних	31.03 – 3.04	*виконано*
3	Проектування програми	4.04 – 7.04	*виконано*
4	Реалізація програми	8.04 – 20.04	*виконано*
5	Тестування і налагодження програми	20.04 – 24.04	*виконано*
6	Підготовка пояснювальної записки:	25.04 – 28.04	*виконано*
7	Підготовка презентації та доповіді	28.04 – 1.05	*виконано*
8	Нормоконтроль, рецензування	1.05 – 3.05	*виконано*
9	Занесення диплома в електронний архів	4.05	*виконано*
10	Попередній захист	5.05	*виконано*
11	Допуск до захисту у зав. кафедри	6.05	*виконано*

Дата видачі завдання 29 *березня* 20*23* р.

Студент _____ Хрипунов І.М.
(підпис)

Керівник роботи _____ доц. Каук В.І.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить 74 стор., 18 рис., 2 табл., 16 джерел.

МАШИННЕ НАВЧАННЯ, МОБІЛЬНИЙ ДОДАТОК, РОЗПІЗНАВАННЯ ОБРАЗІВ, CORE ML, iOS SDK, SWIFT, TENSOR FLOW LITE.

Об’єктом розробки є мобільний додаток для розпізнавання образів з використанням CoreML та TensorFlowLite моделей.

Метою роботи є дослідження використання різних типів моделей машинного навчання для розпізнавання образів на платформі iOS та порівняння їх продуктивності.

Методи рішення створені з використанням мови програмування Swift для мобільного додатку, CoreML та TensorFlowLite yolov5n моделей для розпізнавання образів.

У результаті роботи було здійснено аналіз предметної галузі, поставлена задача для дослідження, розроблена архітектура мобільного додатку, виконано цикл інтеграції відповідних моделей для розпізнавання об’єктів у iOS додаток та проведено дослідження зручності використання та інтеграції моделей пошуку та ідентифікації образів, їх продуктивності, ефективності, швидкості та якості роботи в мобільних додатках на платформі iOS, виконана апробація отриманих результатів.

Додаток доступний для використання на смартфонах, які працюють на операційній системі iOS 14 та вище.

MACHINE LEARNING, MOBILE APP, OBJECTS DETECTION, CORE ML, iOS SDK, SWIFT, TENSOR FLOW LITE.

The object of development is a mobile application for objects detection using CoreML and TensorFlowLite models.

The purpose of the development is to investigate the use of different types of machine learning models for objects detection on the iOS platform and compare their performance.

The solution methods are created using Swift programming language for mobile application, CoreML and TensorFlowLite yolov5n models for objects detection.

As a result of the work, an analysis of the subject area was carried out, a task was set for research, the architecture of the mobile application was developed, a cycle of integration of relevant models for objects recognition in the iOS application was carried out, and a research was carried out of the usability and integration of detection and identification models, their productivity, efficiency, speed and quality of work in mobile applications on the iOS platform.

The application is available for use on smartphones running iOS 14 and above.

Я, Хрипунов Ілля Михайлович, студент гр.ІПЗм-21-2, здобувач вищої освіти на другому (магістрському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження ML моделей для завдань розпізнавання образів на iOS», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

Робота виконана відповідно до вимог академічної доброчесності та пройшла перевірку на плагіат, який складає – _____7.74_____%.

ЗМІСТ

Вступ 8
1 Аналіз предметної області і постановка задачі 10
1.1 Теорія розпізнавання образів 10
1.2 Використання смартфонів для розпізнавання образів 12
1.3 Особливості CoreML 14
1.4 Особливості TensorFlow 16
1.5 Постановка задачі 19
2 Моделювання мобільного додатку 21
3 реалізація мобільного додатку 24
3.1 Реалізація мобільного застосунку 24
3.2 Інтеграція CoreML моделі 27
3.3 Інтеграція TFLite моделі 28
3.4 Алгоритм розпізнавання образів 30
3.5 Контроль за апаратним забезпеченням пристрою 33
4 Порівняння зручності та якості інтеграції 36
4.1 Розробка критеріїв порівняння 36
4.2 Порівняння за визначеними критеріями 38
5 Апробація результатів дослідження 42
5.1 Постановка практичної задачі 42
5.2 UML проектування мобільного додатку 42
5.3 Реалізація застосунку 44
5.4 Приклади найцікавіших алгоритмів та методів 47
5.5 Тестування розробленого програмного забезпечення 50
Висновки 52
Перелік джерел посилання 54
Додаток А 55
Додаток Б Звіт результатів перевірки на унікальність тексту 56
Додаток В Фрагмент лістингу коду 57
Додаток Г Апробація результатів роботи 63

Додаток Д Слайди презентації 64

Додаток Е Експертний висновок результатів Перевірки кваліфікаційної роботи на відповідність оформлення Вимоги ДСТУ 3008: 2015 74

ВСТУП

Інформаційні технології стають все більш і більш важливим компонентом будь-якої сфери життя та діяльності кожної сучасної людини. Різноманітні програмні забезпечення та мобільні додатки використовуються кожного дня практичного у всіх основних галузях діяльності людини.

Найбільш актуальними векторами розвитку інформаційних технологій на поточний момент стає розвиток штучного інтелекту та машинного навчання. Саме ці основні аспекти станом на сьогоднішній день викликають найбільший інтерес та створюють найвищий рівень попиту як серед розробників так і серед кінцевих користувачів. Дедалі більше різноманітних галузей направленні на підвищення ефективності, якості власного обслуговування та продуктивності шляхом безпосередної інтеграції вищезазначених технологій.

Останнім часом розпізнавання образів користується все більшою популярністю, адже розпізнавання зображення, тексту чи мови, а також різноманітних явищ сприяє спрощенню комунікативного зв’язку людини з комп’ютером, допомагає застосовувати різні системи штучного інтелекту, в тому числі в системах відеоспостереження. Можливість сприймати зовнішній світ у формі образів сприяє передумовам дослідження властивостей величезної кількості об’єктів завдяки ознайомленню з кінцевою їх кількістю, а об’єктивна ознака засадничої властивості образів допомагає створювати модель їх розпізнавання [4].

На сьогоднішній день дуже важко уявити життя сучасної людини без смартфону. Дослідження американського інституту інформаційних технологій станом на 15 листопада 2022 року показали, що більше 80% відсотків населення світу щодня проводить від 3 до 5 годин у власному мобільному телефоні. Фактично, смартфон стає невід’ємною частиною життя усіх людей на нашій планеті за рахунок різноманітних можливостей даних девайсів, які реалізовані у вигляді різного роду додатках та застосунках.

Вищезазначені фактори і стають опорою та відправною точкою для розвитку більш конкретного напрямку, а саме інтеграція технологій машинного

навчання з сучасними мобільними додатками. Адже, розвиток даного направлення буде сприяти підвищенню продуктивності мобільних додатків та розширенню сфери їх застосування у сучасному світі.

Метою даного проекту є дослідження можливості інтеграції моделей машинного навчання для розпізнавання образів із сучасними мобільними додатками. У зв’язку з розвитком даних окремих технологій задача їх взаємодії для отримання комплексного програмного продукту є дуже актуальною та, на поточний момент, не достатньо дослідженою.

Задача курсового проекту полягає у створенні мобільного додатку для розпізнавання образів за допомогою моделей машинного навчання. Крім того, результатом даного дослідження має бути створення низки критеріїв для оцінки ефективності та продуктивності інтеграції даних технологій з точки зору розробника та кінцевого користувача. На базі розробленого додатку та відповідних критеріїв буде проведено аналіз взаємодії даних технологій у комплексному мобільному застосунку та зроблено відповідні висновки.

Дане дослідження буде актуальним серед представників різноманітних галузей людської діяльності. Воно дасть можливість чітко зрозуміти яким чином та які переваги можна отримати від інтеграції моделей машинного навчання з мобільними додатками. Крім того, дана робота зможе окреслити шляхи вирішення задачі реалізації даної взаємодії для отримання продуктивного та привабливого для кінцевого користувача комплексного рішення для вирішення окремих бізнес задач.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАДАЧІ

1.1 Теорія розпізнавання образів

Із завданням розпізнавання образів живі системи, у тому числі і людина, стикаються постійно з моменту своєї появи. Зокрема, інформація, що надходить з органів чуттів, обробляється мозком, який у свою чергу сортує інформацію, забезпечує прийняття рішення, а далі за допомогою електрохімічних імпульсів передає необхідний сигнал далі, наприклад, органам руху, які реалізують необхідні дії. Потім відбувається зміна навколишнього оточення, і вищевказані явища відбуваються заново. І якщо розібратися, то кожен етап супроводжується розпізнаванням.

Теорія розпізнавання образів – це розділ когнітивної науки, який прагне пояснити, як люди та інші тварини сприймають і класифікують сенсорну інформацію. Він припускає, що мозок здатний ідентифікувати та витягувати значущі моделі зі складних і галасливих сенсорних вхідних даних, що дозволяє нам зрозуміти навколишній світ.

Теорія припускає, що люди народжуються з вродженою здатністю розпізнавати візерунки, тому ми можемо ідентифікувати форми, предмети та обличчя з раннього віку.

Теорія розпізнавання образів передбачає, що люди використовують різні стратегії для розпізнавання шаблонів. Ці стратегії включають виявлення особливостей, зіставлення шаблонів, зіставлення прототипів і обробку знизу вгору. Виявлення ознак передбачає ідентифікацію конкретних характеристик або особливостей візерунка, таких як форма літери або колір об’єкта. Зіставлення шаблону передбачає порівняння шаблону зі збереженим шаблоном або моделлю, щоб визначити, чи він збігається. Зіставлення прототипу передбачає порівняння шаблону із загальним прототипом або ідеалізованим представленням категорії. Обробка «знизу вгору» передбачає аналіз окремих сенсорних вхідних сигналів і формування сприйняття всієї моделі.

Теорія розпізнавання образів застосовується в багатьох галузях, включаючи комп’ютерне зір, розпізнавання мови та штучний інтелект. Його також використовували для розробки моделей навчання та пам’яті, а також для вивчення нейронних механізмів, що лежать в основі сприйняття та пізнання.

Образ являє собою певну структуру в системі класифікації, яка акцентує конкретну групу об'єктів за вибраними ознаками. Образи мають характерні властивості, які проявляються в тому, що знайомство з кінцевою кількістю явищ з одного числа множини дозволяє дізнаватися про будь-яку кількість його представників. Образи володіють об'єктивними властивостями в тому аспекті, що люди, які вчаться на різних матеріалах спостережень, переважним чином однаково та незалежно від інших поділяють одні й ті ж об'єкти за певними критеріями оцінювання. На практиці в задачі розпізнавального процесу універсальна множина ділиться на певні частини-образи. При цьому відображення будь-якого об'єкта органами сприйняття системою розпізнавання незалежно від знаходження цього об’єкту щодо цих органів, називають зображенням об'єкта, а множини цих відображень, що об'єднані схожими якостями, називають образами [6].

Розпізнавання образів – це теоретичний розділ науки штучного інтелекту, який вивчає способи класифікації об’єктів. Зазвичай об’єкт, який класифікується, являє собою образ (pattern). Образом може легко стати цифрова фотографія, наприклад, розпізнавання конкретних зображень; літера чи цифра, наприклад, у розпізнаванні символу; запис мови під час розпізнавання мови. Розпізнавання образів – задача визначення об’єкта або ідентифікація певних якостей цього об’єкта за його зображенням, так зване оптичне розпізнавання. Водночас, образом є сукупність в системі класифікації, яка з’єднує конкретну групу об’єктів за певною ознакою [5]. Способи машинного навчання та прийняття рішень – це кінцева стадія в процесі розпізнавання. Ці способи знаходяться на межі статистичної математики, способів оптимізації, а також класичних наук математики, але мають також і власну особливість, яка пов’язана з проблематикою обчислювальної продуктивності та процесом перенавчання.

Здебільшого суть навчання полягає в наступному: на базі певної навчальної вибірки з властивостями кожного класу необхідно спроектувати таку систему, завдяки якій комп’ютер зможе дати аналіз новому зображенню та визначити, який з об’єктів представлений на зображенні.

Здатність сприймати зовнішній світ у вигляді образів дозволяє з певною впевненістю розпізнати нескінченну кількість об’єктів на основі ознайомлення з їх скінченною кількістю, а об’єктивний характер основної властивості образів дозволяє нам моделювати процес їх визнання.

1.2 Використання смартфонів для розпізнавання образів

Розпізнавання образів має величезні перспективи у вирішенні широкого ряду задач у повсякденному життя. Так як розпізнавання образів потребує значних ресурсів, то певний час його можливості були обмеженні для використання пересічним користувачем. Здебільшого використовувався підхід під назвою «навчання в хмарі». Основна ідея даного підходу полягала в тому, що смартфон виступав у ролі проміжної ланки, яка збирали необхідні дані для аналізу, відправляла їх для аналізу на хмарний застосунок і обробляла отримані результати [1].

Розвиток смартфонів спричинив революцію у світі розпізнавання образів. Завдяки потужним камерам, передовим процесорам і складному програмному забезпеченню сучасні смартфони здатні розпізнавати та ідентифікувати образи в реальному часі. Від розпізнавання об’єктів до розпізнавання мови, від розпізнавання жестів до розпізнавання зображень, смартфони змінюють спосіб нашої взаємодії з навколишнім світом.

Одним із найпоширеніших застосувань розпізнавання образів на смартфонах є розпізнавання об’єктів. Це передбачає використання камери телефону та програмного забезпечення для ідентифікації об’єктів у реальному часі. Наприклад, ви можете навести свій смартфон на продукт у магазині, і він розпізнає продукт і надасть вам інформацію про нього, як-от його ціну, відгуки та характеристики. Це стало можливим завдяки вдосконаленим алгоритмам

комп’ютерного зору, які здатні ідентифікувати ключові характеристики об’єкта та зіставляти їх із базою даних відомих об’єктів.

Однією з ключових переваг використання смартфонів для розпізнавання образів є їх мобільність. На відміну від настільних комп’ютерів або навіть ноутбуків, смартфони маленькі та легкі, що робить їх ідеальними для використання в дорозі. Це означає, що ви можете використовувати свій смартфон, щоб розпізнавати образи, де б ви не знаходилися, незалежно від того, чи ви робите покупки, подорожуєте чи просто відпочиваєте вдома.

Ще однією перевагою використання смартфонів для розпізнавання образів є простота їх використання. На відміну від традиційних систем розпізнавання образів, які можуть бути складними в налаштуванні та використанні, додатки для смартфонів розроблені зручними та інтуїтивно зрозумілими. Це означає, що навіть люди з невеликим технічним досвідом або без нього можуть використовувати смартфони для розпізнавання образів.

Першим важливим аспектом є швидкість роботи застосунку. Обмін інформацією між пристроєм і хмарою може займати значний час. Швидкість роботи є дуже важливим критерієм сучасного застосунку і тому затримка більше кількох секунд може бути критичним фактором і сприяти негативному досвіду користувача.

Мобільні стосунки більшість часу доступаються до Інтернету через мобільну мережу, і користувачі зацікавлені в мінімізації її використання, оскільки це дорого, а також знижує енергоефективність. Крім того, на даний момент якісна мобільна мережа є далеко не всюди (зокрема в тунелях метро, деяких селах і на трасах між містами). Можливість роботи офлайн дасть поштовх для розвитку застосунків, що не могли бути втілені раніше в повному обсязі через необхідність стабільного підключення до Інтернету.

Енергоефективність зокрема також відіграє важливу роль у формуванні досвіду користувача, адже важлива риса мобільних пристроїв це автономність від точок живлення. Крім того, низька енергоефективність створює проблему у

довготерміновій перспективі, адже від великої кількості циклів розрядів і зарядів батарея зношується.

Надзвичайно важливим аспектом в сучасному світі є конфіденційність і безпека. Оскільки вся робота відбувається безпосередньо на пристрої, користувач уникає всіх небезпек, що пов’язані з передачею даних, зокрема їх перехоплення кіберзлочинцями. Гарантія безпеки і приватності особистих даних користувача є вимогою GDPR (General Data Protection Regulation - Загальний регламент захисту даних). Прикладом вдалого використання машинного навчання на пристрої в розрізі безпеки та приватності може слугувати розпізнавання обличчя за допомогою нейронної мережі безпосередньо на пристрої.

Також розпізнавання образів на пристрої в порівнянні з обробкою в хмарі має переваги і для розробників. Уникаючи значної централізованої обробки даних виробник застосунку може значно зекономити на серверах. Також економиться час на налаштування і підтримку серверів. До того ж знижуються вимоги до ширини каналу передачі даних [3].

Отже, розпізнавання на пристрої значно економить ресурси, гроші та час як для користувачів так і для розробників. Гіганти Google і Apple активно ведуть розробку в цьому напрямі, створюючи інструментарій для зручної розробки застосунків з машинним навчанням, додаючи все нові і нові функції.

1.3 Особливості CoreML

На WWDC'17 Apple представила новий фреймворк для роботи із технологіями машинного навчання Core ML. На основі нього в iOS реалізовані власні продукти Apple: Siri, Camera та QuickType. Core ML дозволяє спростити інтеграцію машинного навчання у додатки та створювати різні «розумні» функції. Це дозволяє створювати інтелектуальні фічі на пристрої, в тому числі й розпізнавння будь-яких об'єктів. Core ML дозволяє легко імпортувати у програму різні алгоритми машинного навчання, такі як: tree ensembles, SVMs і generalized linear models. Результати обчислень відбуваються майже миттєво.

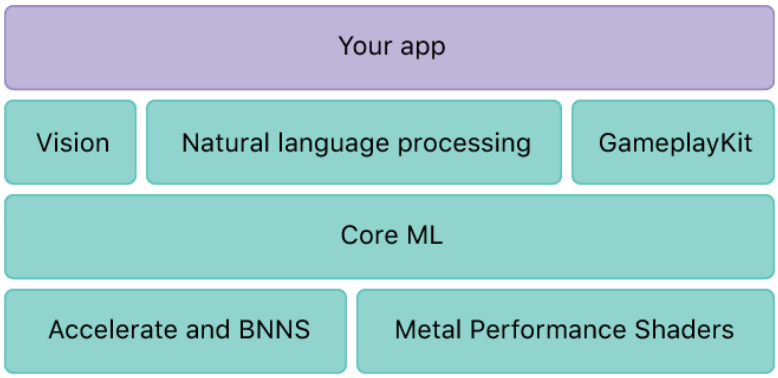

Рисунок 1.1 – Схема роботи CoreML в мобільному додатку

Фреймворк Core ML використовує низькорівневі фреймворки Metal, Accelerate та BNNS, що забезпечує високу швидкість обчислень. Ядро підтримує нейронні мережі, узагальнені лінійні моделі, деревоподібні алгоритми прийняття рішень, генерацію ознак, метод опорних векторів, пайплайн моделі.

Однією з ключових переваг CoreML є простота використання. Фреймворк надає простий API для завантаження та використання моделей машинного навчання. CoreML підтримує широкий спектр моделей машинного навчання, включаючи нейронні мережі, ансамблі дерев і опорні векторні машини. Ці моделі можна навчити за допомогою різноманітних інструментів і мов, наприклад Python, а потім експортувати у формат, сумісний із CoreML.

Core ML оптимізує продуктивність на пристрої за рахунок використання ресурсів центрального процесора, графічного процесора та Neural Engine, мінімізуючи обсяг пам'яті та енергоспоживання.

Ще однією ключовою особливістю CoreML є його ефективність. Фреймворк оптимізований для мобільних пристроїв, що означає, що він розроблений для ефективної роботи на пристроях iOS при мінімальному споживанні енергії.

Спочатку компанія Apple не представила власних технологій для створення та навчання моделей, але створила конвертер для інших популярних фреймворків: Caffe, Keras, scikit-learn, XGBoost, LIBSVM (див. рис. 1.2).

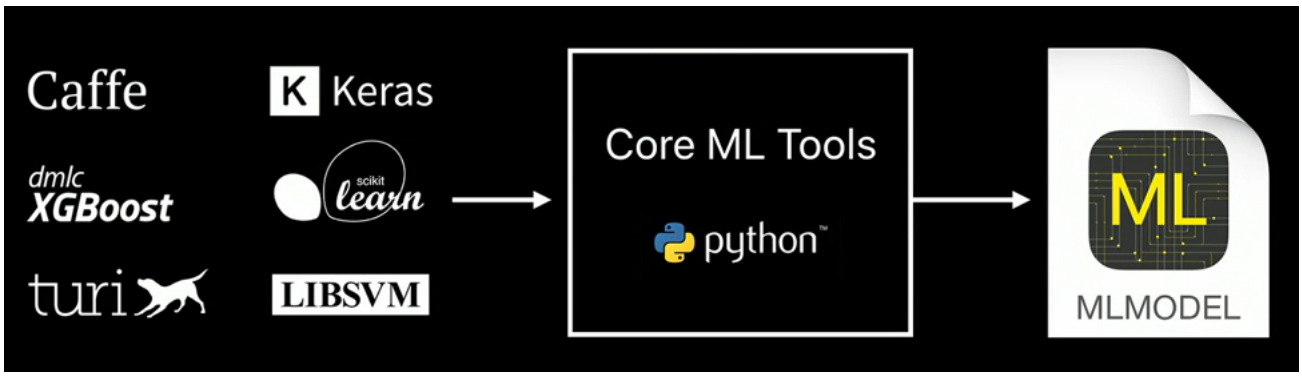

Рисунок 1.2 – Схема створення ML моделі

Використання сторонніх інструментів накладало ряд обмежень, навчені моделі мали досить великий обсяг, а саме навчання займало чимало часу. Саме тому вже у 2019 році компанія Apple представила Core ML 2 та власний інструмент для навчання моделей – фреймворк Create ML, який використовує технології Apple – Xcode та Swift. Якщо раніше для навчання моделі з використанням 20 тисяч зображень потрібно 24 години, то фреймворк Create ML дозволив скоротити цей час до 48 хвилин на MacBook Pro і до 18 хвилин - на iMac Pro. Крім того, розмір навченої моделі зменшився з 90MB до 3MB.

Незважаючи на велику кількість позитивних аспектів використання технології CoreML, в ній є також свої особливості та обмеження. Модель не стискається. Крім того, вона не шифрується. Розробникам доведеться подбати про захист даних самому.

Незважаючи на це саме ця технологія була обрана для дослідження використання моделей машинного навчання для задач розпізнавання образів, адже вона є нативною і максимально оптимізованою для використання на девайсах з операційною системою iOS.

1.4 Особливості TensorFlow

TensorFlow – це інструмент Google з відкритим вихідним кодом для паралельних обчислень, включаючи реалізацію нейронних мереж та інших методів навчання штучного інтелекту. Він розроблений, щоб спростити роботу з нейронними мережами, і розглядається як більш загальний та простий, ніж інші

варіанти. Версія Lite забезпечує виведення моделей машинного навчання на пристрої із малою затримкою.

Сама бібліотека включає безліч інструментів для різних напрямків ML, але найчастіше використовується для роботи з нейронними мережами. Це структури, натхненні пристроєм мереж нейронів у людській нервовій системі. Нейронні мережі складаються із програмних елементів-«нейронів» та зв'язків між ними, і такий пристрій дозволяє їм навчатися. TensorFlow працює зі звичайними та глибокими нейронними мережами різних типів: рекурентними, згортковими тощо.

У TensorFlow моделі представлені за допомогою графів – математичних абстракцій, які складаються з вершин та шляхів між ними [10].

TensorFlow працює з тензорами – багатовимірними структурами даних у векторному, тобто спрямованому просторі. Вони використовуються в лінійній алгебрі та фізиці. Звідси походить назва бібліотеки. За допомогою тензорів описуються шляхи графа, а вершини це математичні операції.

Обчислення TensorFlow виражаються як потоки даних через граф. Це означає, що інформація «рухається» по графу, передається шляхами від вершини до вершини.

Бібліотека може працювати на потужностях звичайного центрального процесора (CPU) або використовувати потужності графічного процесора (GPU). Режим перемикається у коді. Існує спеціальний тензорний процесор TPU, створений розробниками бібліотеки, яким можна скористатися через хмарні сервіси Google.

Бібліотека написана так, що не треба думати про технічну реалізацію абстрактних понять. Можна зосередитись на описі логіки програми та на математиці, а спосіб реалізації обчислень – завдання TensorFlow, а не програміста. Це полегшує розробку та дозволяє сконцентруватися на важливих завданнях.

На рисунку 1.3 зображено схему роботи бібліотеки на мобільномуу пристрої. Основними діючими компонентами виступають так звані «тензори».

Саме ці функціональні об’єкти і виконують основні функції, що стосуються розпізнавання вхідних даних та повернення результатів обробки.

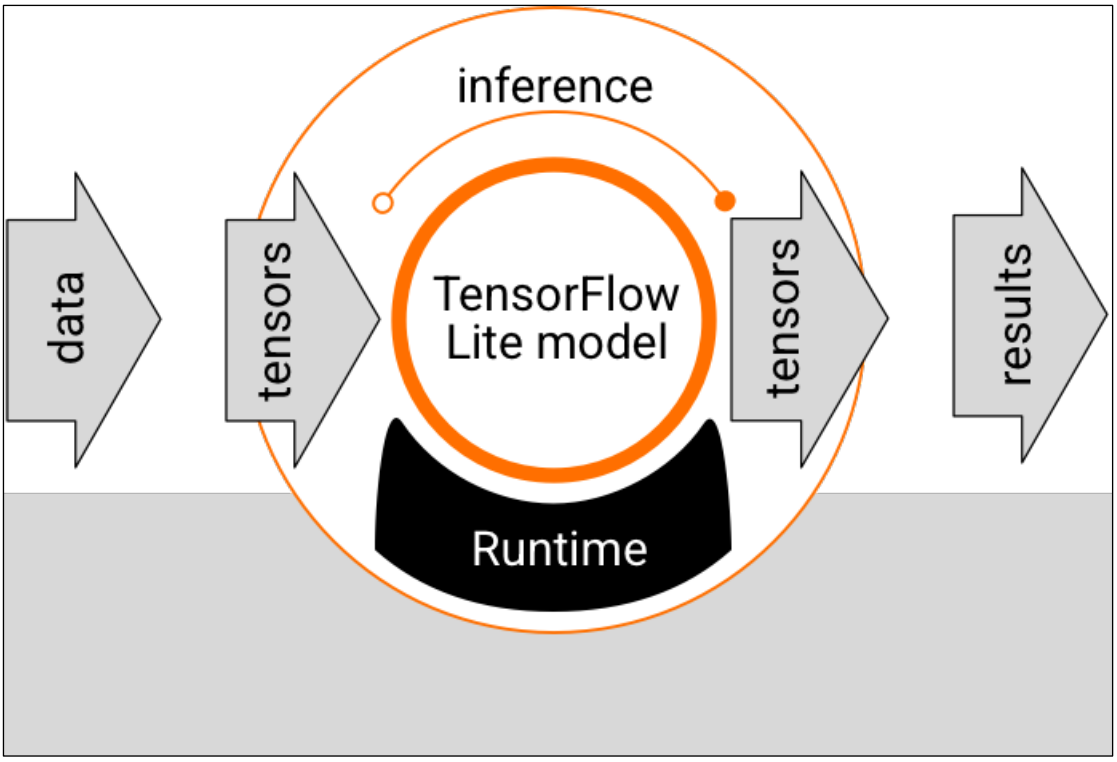

Рисунок 1.3 – Схема роботи бібліотеки в межах мобільного додатку

TensorFlow дозволяє працювати з компонентами моделі окремо і створювати її на ходу, при цьому окремо перевіряти кожен елемент. Це зручніше, ніж описувати граф як єдину монолітну структуру. Підхід робить розробку більш інтерактивною – структуру можна гнучко налаштовувати та змінювати.

TensorFlow, працює в популярних операційних системах, локально або у хмарі. Вона має розширення для мобільних пристроїв, IoT та браузерних додатків. Для мобільних пристроїв та інтернету речей можна скористатися середовищем TensorFlow Lite.

До недоліків даного фреймворку можна віднести високий рівень споживання пам’яті. Якщо TensorFlow використовується з графічним процесором, вона забирає всю його пам'ять. Це знижує продуктивність. Крім того TensorFlow – продукт Google. Компанія відома своїми стандартами для технологій. Досі в TensorFlow зустрічається неочевидна поведінка, через яку код може бути складніше налагоджувати.

Не дивлячись на це, для даного дослідження другою технологією було обрано фреймворк TensorFlow, а саме його мобільну версію TensorFlowLite, адже він є одним з найпоширеніших та найбільш популярних кросплатформерних інструментів для інтеграції моделей машинного навчання з мобільними додатками. Крім того, дана бібліотека є найшвидшою та найбільш гнучкою серед своїх конкурентів.

1.5 Постановка задачі

Метою роботи є дослідження моделей машинного навчання для розпізнавання образів на iOS. В межах проекту необхідно порівняти складність інтеграції відповідних моделей, їх ефективності та продуктивності в межах задач розпізнавання даних на зображеннях з камери девайсу. Крім того, необхідно дослідити можливості контролю над апаратним забезпеченням, що використовується моделями, для підвищення продуктивності розробленого мобільного додатку.

Для виконання даної задачі необхідно реалізувати мобільний додаток, який дасть можливість розпізнавати об’єкти за допомогою камери смартфону. Основною умовою для інтеграціє є те, що моделі, які будуть використовуватись, мають бути локальними і працездатність додатку не повинна залежати від доступу до інтернету.

Для інтеграції моделей необхідно використати фреймворки, які безпосередньо забезпечують можливість використати завчасно навчені моделі в межах функціональних особливостей застосунку. У якості бібліотек для реалізації додатку було обрано CoreML та TensorFlowLite [11], так як вони є найбільш продуктивними та популярними для реалізації задач розпізнавання образів за допомогою моделей машинного навчання.

Результатом дослідження має бути створення низки критеріїв та порівняння вищезазначених фреймворків для реалізації вищезазначених задач на платформі iOS з точки зору як розробника так і кінцевого користувача.

Використовуючи отримані результати необхідно додатково виконати їх апробацію шляхом реалізації комплексного мобільного додатку для операційної системи iOS, бізнес логіка якого має базуватись на використанні моделей машинного навчання для розпізнавання образів.

На основі проведеного дослідження буде зроблено висновок щодо можливості інтеграції моделей машинного навчання для задач розпізнавання образів в мобільні додатки та обрано найзручнішу бібліотеку для створення мобільного застосунку, що виконує поставлену задачу максимально ефективно з точки зору апаратного забезпечення девайсу, розробника та кінцевого користувача.

2 МОДЕЛЮВАННЯ МОБІЛЬНОГО ДОДАТКУ

Застосунок повинен базуватись на архітектурі «MVP» та основних прицнипів реалізації мобільних додатків для платформи iOS. Також мобільний додаток має містити в собі залежності від основних фреймворків для інтеграції моделей машинного навчання, які були обрані для даного дослідження.

На рисунку 2.1 наведено діаграму класів мобільного додатку. На діаграмі відображено усі основні компоненти, що містять логіку відображення екранів і реалізацію задачі розпізнавання образів.

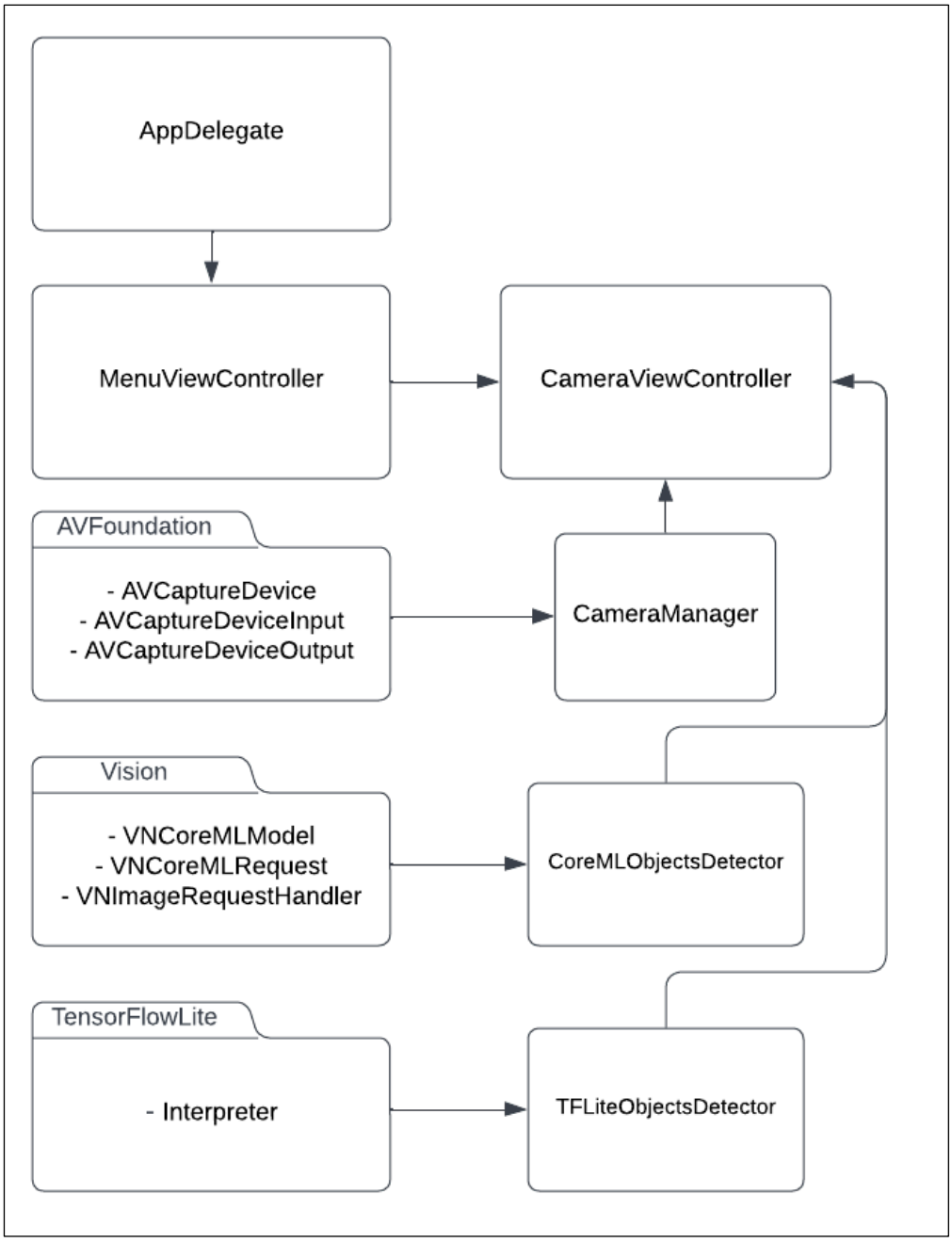

Рисунок 2.1 – Діаграма класів мобільного додатку

Відправною точкою для старту роботи мобільного додатку є клас «AppDelegate». Саме цей клас є основним керуючим компонентом та відповідає за навфгацію в середині додатку. Основною його функціональною відповідальністю є запуск першоого основного екрану, який надасть корситувачу доступ до функціоналу мобільного додатку.

Наступним важливим компонентом розробленого мобільного додатку є класи «MenuViewController» та «CameraViewController». Основна задача даних класів полягає в реалізації саме користувацького інтерфейсу. Що стосуєтсья першого класу, то він безпосерельно має надавати можливість вибирати, який саме підхід для розпізнавання образів має бути використаний – CoreML чи TensorFlowLite. Після того, як вибір зроблено має відбуватись перехід на наступний екран, а саме «CameraViewController». Що стосується даного компоненту, то його основна функціональна відповідальність полягає в надані користувачу доступу до даних з камери та відображення результатів розпізнавння отриманих кадрів у вигляді підсвічування знайдених моделлю об’єктів.

Що стосується класів з приставкою «Detector», то вони є свого роду обгорткою для моделей розпізнавання образів, які використовуються мобільним додатком. В середині себе вони інкапсулюють логіку ініціалізації моделей та виконання задач розпізнавання на відповідних зображеннях. Логіка повинна складатись з наступних етапів:

- первинна обробка вхідного зображення, яка полягає у зміні формату до такого, який необхідний моделі на вхід;
- зміна розмірів вхідного зображення до розміру, з яким працює інтегрована модель машинного навчання;
- обробка результатів отриманих від моделі після розпізнавання.

В результаті виконання усіх вищезазначених етапів класами-обгортками усі отримані дані мають бути передані до класу «CameraViewController». Даний клас має виконати певні перетворення, для того щоб правильно відобразити отримані результати. Відображення результатів має полягати у підсвіченні знайденого об’єкта за допомогою кольоровою рамки з відповідною назвою.

Класи-обгортки використовують відповідні фреймворки, які дають змогу взаємодіяти з моделями машинного навчання, що використовуються. Для «CoreMLObjectsDetector» допоміжною бібліотекою виступає «Vision» (побудований на базі CoreML), а для «TFLiteObjectsDetector» використано «TensorFlowLite» (версія технології «TensorFlow» для використання в мобільних додатках).

Для того, щоб отримати контроль над камерою девайсу і мати змогу отримувати кадри для подальшого аналізу за допомогою моделей використовується клас «CameraManager». Даний клас використовує фреймворк «AVFoundation», який є частиною iOS SDK. Ця бібліотека дає доступ розробникам до камер відповідного девайсу. Основною функціональною задачею даного класу є надання доступу до кадрів з камер девайсу, які виступають вхідними параметрами для моделей машинного навчання для подальшого розпізнавання образів [7].

Взаємодія між цими класами фактично реалізує інтеграцію мобільного додатку з відповідними моделями машинного навчання через класи обгортки з використанням камери девайсу як джерела вхідних даних для розпізнавання об’єктів.

3 РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКУ

3.1 Реалізація мобільного застосунку

Мобільний додаток реалізовано на мові програмування Swift, з використанням iOS SDK, в середовищі розробки Xcode. У якості архітектурного патерну для створення застосунку було обрано патерн «MVC» (див. рис.3.1).

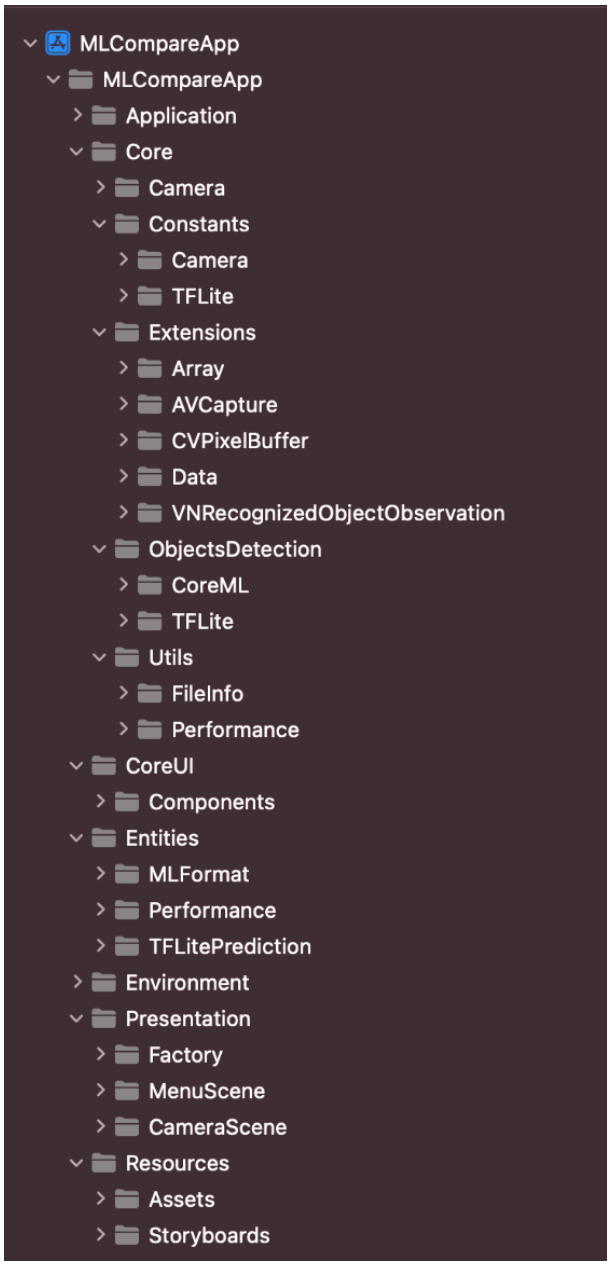

Рисунок 3.1 – Структура файлів мобільного додатку

Додаток складається з двох окремих екранів, а саме: меню та екран камери. Екран меню дає можливість обрати формат моделі, яка буде використана для подальшого розпізнавання образів на відповідному кадрі з камери.

З точки зору користувацького інтерфейсу екран представляє собою компонент для вибору варіанту, де кожен варіант відповідає за окрему модель, яка знаходиться в середині проекту. Після того, як користувач обирає варіант і натискає на кнопку «START», відбувається перехід на екран камери, де в асинхронному режимі ініціалізується об’єкт відповідного класу-обгортки для обраної моделі (див.рис.3.2).

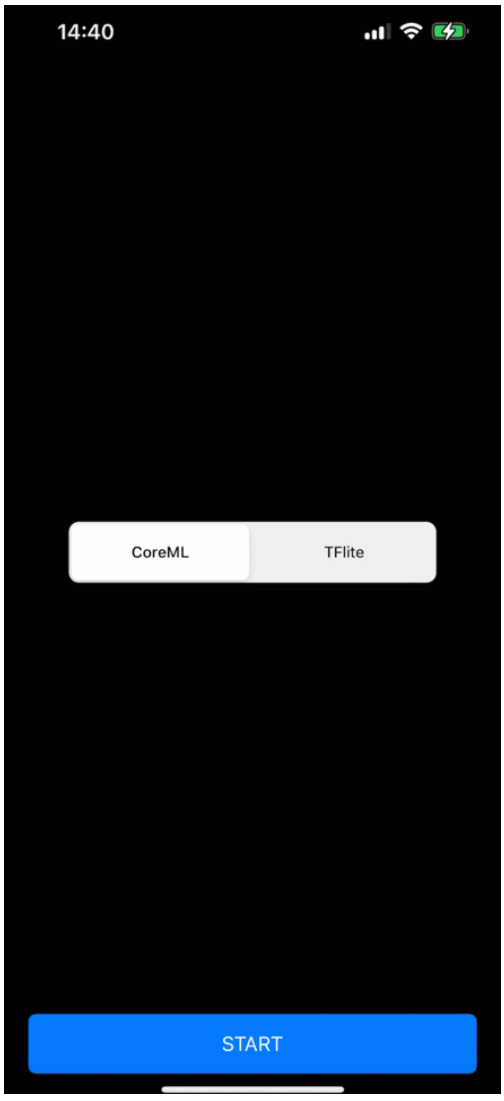

Рисунок 3.2 – Екран меню

Наступним і основним екраном мобільного додатку є екран камери. Він включає в себе відображення кадрів, які отримуються основною камерою девайсу. Головною функціональною особливістю цієї сторінки додатку є пошук та ідентифікація об’єктів на відповідних кадрах.

Основною частиною даного екрану є клас «CameraManager». Даний клас безпосередньо реалізує доступ до камери девайсу за допомогою фреймворку AVFoundation, який є частиною iOS SDK. Перш за все, об’єкт цього класу знаходить девайс, який в подальшому буде використаний для отримання інформації з камери. Далі відбувається налаштування вихідного потоку даних (див.рис.3.3).

Оскільки необхідно аналізувати дані в режимі реального часу і виконувати розпізнавання на кожному отриманому кадрі, то ми реалізуємо відповідний інтерфейс, що надає доступ до кожного з отриманих камерою буферів [2].

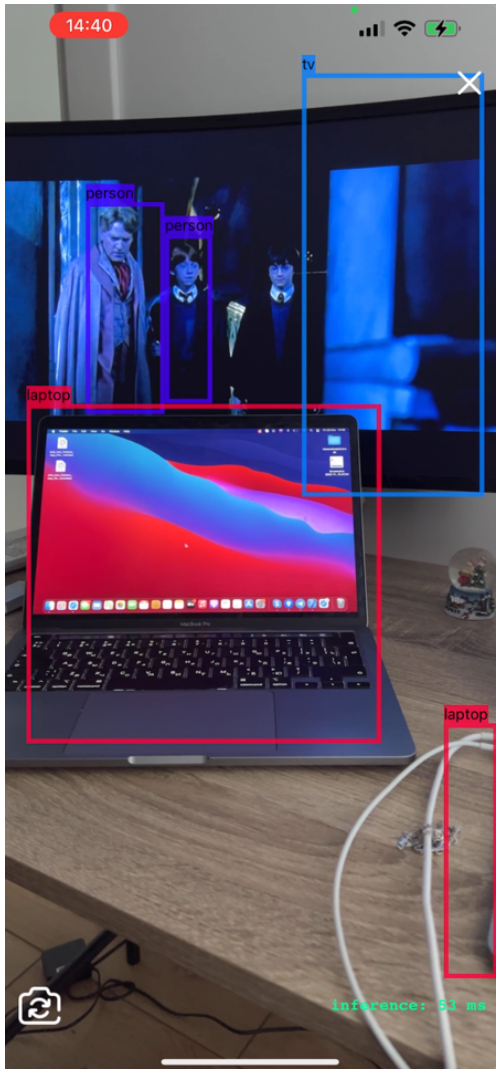

Рисунок 3.3 – Екран камери

За допомогою додаткових компонентів користувацького інтерфейсу відбувається відображення результатів розпізнавання образів на відповідному

кадрі. Коли було знайдено та ідентифіковано якийсь об’єкт, то користувач на екрані зможе побачити його межі та назву.

Також даний екран містить кнопку, яка дає можливість користувачу перемикатись з основної камери на фронтальну і навпаки. Додатково на екрані можна побачити інформацію про швидкість роботи моделі, що використовується.

Незалежно від обраної моделі для розпізнавання, користувацький інтерфейс на даному екрані залишається незмінним. Відображення результатів обробки кадрів також не відрізняється в залежності від обраної технології для пошуку та ідентифікації образів.

3.2 Інтеграція CoreML моделі

Першим етапом інтеграції CoreML моделі є додавання її файлів в проект. Для цього ми взяли файл yolov5n.mlmodel і додали його в файлову структуру проекту. Вхідні дані для даної моделі повинні бути у вигляді буферів зображень розміру 640 на 640 пікселів (див.рис.3.4).

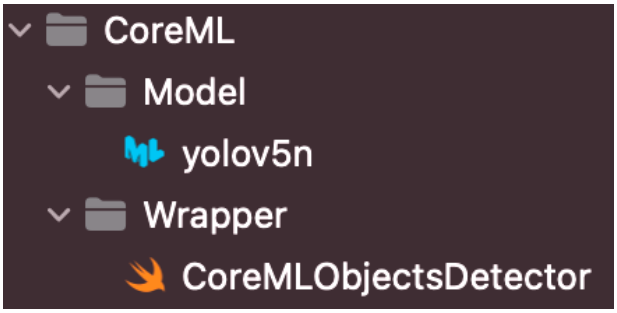

Рисунок 3.4 – Структура файлів для взаємодії з CoreML моделлю

Для більш зручної роботи з моделлю реалізовано клас «CoreMLObjectsDetector». Він містить в собі основну логіку обробки вхідних кадрів.

Реалізація функціоналу даного класу базується на використанні фреймворку Vision, який написаний на базі CoreML. Він дає можливість розробнику набагато швидше і простіше. Додатаково, використання цієї бібліотеки допомагає привести вхідні дані до необхідного формату [8].

На етапі створення об’єкту даного класу відбувається ініціалізація моделі, яка буде в подальшому використана для розпізнавання. Після цього, кожен кадр, який поступає на обробку, проходить етап трансформації до необхідного моделлю формату та розміру. Наступним кроком іде створення запиту на розпізнавання образів і відправка даного запиту в модель. Після успішного виконання запиту на розпізнавання ми отримуємо об’єкт, який містить в собі інформацію з моделі.

Наступним кроком роботи програмного застосунку є інтерпретація отриманих даних. У випадку використання CoreML дані проходять попередню обробку за допомогою вбудованого фреймворку та в результаті роботи ми отримаємо об’єкти класу «VNRecognizedObjectObservation». Отримані екземпляри містять в собі інормацію щодо назви знайденого об’єкту на вхідному зображенні та дані, які стосуються розташування знайденого образу в межах зображення.

Після успішної інтерпретації вихідних даних моделі відбувається наступний етап, а саме відмалювання отриманоє інформації на користувацькому інтерфейсі. Для цього «CameraViewController» виконує певні перетворення для того, щоб привести дані, щодо розташування знайденого образу з системи координат вхідного зображення в систему координат екрану. Після виконання даних перетвореннь ми отримуємо відповідні координати та розміри в системі координат, яка необхідна для відображення даних. Додатково відбувається підрахунок та виведення на екран кількості часу, який було використанно відовідною моделлю для пошуку даних на вхідному зображенні. Усі ці дані доступні користувачу на екрані, що дає можливість оцінити якість інтеграції.

3.3 Інтеграція TFLite моделі

Тепер розглянемо процес інтеграції моделі формату TFLite. Перш за все нам необхідно додати до проекту фреймворк «TensorFlowLite». Для того, щоб це зробити нам необхідно створити текстовий файл з назвою «Podfile» в корневій директорії і додати туди назву даної бібліотеки.

Після цього нам необхідно виконати безпосередню інтеграцію вищезазначеного фреймворка в середину мобільного додатку. Для цього необхідно виконати певний перелік команд в терміналі. В результаті виконання даних команд усі основні компоненти фреймворку будуть отримані з віддаленого ресурсу, а система в свою чергу створить окремий проект, який буде складатись з основної функціональної частини та інтегрованого фреймворку (див.рис.3.5).

```
# Uncomment the next line to define a global platform for your project
# platform :ios, '14.0'

target 'MLCompareApp' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  # Pods for MLCompareApp
  pod 'TensorFlowLiteSwift', "~> 2.3.0"
  pod 'Toast-Swift', '~> 5.0.1'

end
```

Рисунок 3.5 – Вміст файлу «Podfile»

Наступним кроком, аналогічно з попереднім варіантом, необхідно додати файли моделі в проект. В даному випадку ми додаємо два файли, а саме: yolov5n.tflite та classes.txt. Перший файл відповідає за саму модель, а текстовий файл містить в собі список ідентифікаторів об’єктів, які можуть бути розпізнані. Адже на відміну від CoreML фреймворку, дана бібліотека віддає дані в тому вигляді, в якому вони було отримані з моделі в результаті розпізнавання, тобто у вигляді багатовимірного масиву значень. Кожне вхідне значення даного массиву в свою чергу містить індекс назви знайденого об’єкту, який в подальшому використаний, для того щоб отримати відповідна дані з текстового файлу.

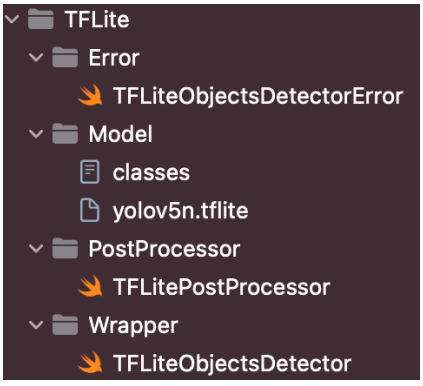

Рисунок 3.6 – Структура файлів для взаємодії з TFLite моделлю

Використовуючи файли, які ми попередньо додали відбувається створення об’єкту «Interpreter», який відповідє за обробку кадру і отримання даних з них. Однак, ініціалізація об’єкту даного класу ще не є кінцевим етапом для запуску процесу розпізнавання.

Додатково необхідно отримати дані з текстового файлу для того, щоб після отримання результатів розпізнавання знайти відповідний ідентифікатор для знайденого образу. Для цього додаток знаходить даний файл у власній структурі та зчитує усі вхідні дані у звичайний масив примітивних значень.

Для коректної інтеграції з моделлю машинного навчання даного формату необхідно створити методи, які будуть модифікувати вхідні дані до відповідних вимог моделі. В нашому випадку ми перевіряємо та модифікуємо, в разі необхідності, формат змінюємо розміри вхідного зображення до таких, які необхідні.

Після цього ми використовуємо попередньо створений об’єкт класу «Interpreter» для розпізнавання образів на вхідному кадрі. В результаті роботи ми отримаємо колекцію значень типу Float, які містять в собі інформацію про розміри, позицію та індекс ідентифікатора знайденого образу.

Для того, щоб отримати результати у форматі, який необхідний для відображення ми використовуємо клас «TFLitePostProcessor». На базі інформації про формат даних, що повертає дана модель. За допомогою методів даного класу ми конвертуємо масив даних у окремі структури. [11].

На етапі ініціалізації класу-обгортки ми отримували дані з текстового файлу. Саме ця інформація допоможе нам знайти назву для розпізнаного об’єкту за відповідним індексом. Набір фінальних структур передається на відображення в «CameraViewController».

3.4 Алгоритм розпізнавання образів

В результаті реалізації усіх основних складових частин мобільного додатку ми отримали систему взаємодій, які разом представляють собою кінцевий застосунок. Однак, ключовим аспектом з точки зору посталеної задачі, а саме

інтеграції моделі розпізнавання образів в мобільний додаток, є ефективність. Саме для підвищення даної характеристики було реалізовано певний алгоритм, який сприяє більш інтелектуальному аналізу вхідних даних моделю та безпосередньо впливає на кінцеву якість додатку.

Оскільки для розпізнавання використано камеру, то вхідною умовою для даного додатку є доступ до усіх кадрів, які отримуються пристроєм в режимі реального часу. У зв'язку з розвитком технологій та беспосерельно вдосконаленням мобільних пристроїв на сьогоднішній день користувачі мають можливість знімати відео дуже високої якості зі швидкістю від 30 до 120 кадрів за секунду в межах можливостей смартфонів на базі системи iOS. Таким чином, ми отримуємо, що у випадку прямої реалізації задачі розпізнавання образів ми стикнемось з тим, що модел буде отримувати мінімум 30 кадрів в секунду на обробку. В такому випадку ресурси будуть використовуватись на максимум і не будуть віддавати точні результати. Значить, для більш ефективної реалізації необхідно зробити так, щоб не всі кадри потрапляли на розпізнавання. На рисунку 3.7 зображено блок-схема для алгоритму розпізнавання образів за допомогою CoreML моделі машинного навчання в мобільному додатку.

Рисунок 3.7 – Алгоритм розпізнавання образів за допомогою CoreML моделі

Як ми можемо бачити на вищезазначеній діаграмі, алгоритм включає в себе додатковий підрахунок кадрів для більш оптимального розпізнавання.

Після отримання кадру з камери ми збільшуємо лічильник на одиницю. Якщо даний кадр є 15-м, то ми відправляємо його на обробку до моделі. Після отримання результатів ми відображаємо їх на екрані. Обробка кожного 15-го кадру підвищить продуктивність системи й знизить навантаження на апаратне забезпечення пристрою.

Майже аналогічним чином виглядає алгоритм розпізнавання за допомогою моделі формату TFLite (див.рис.3.8).

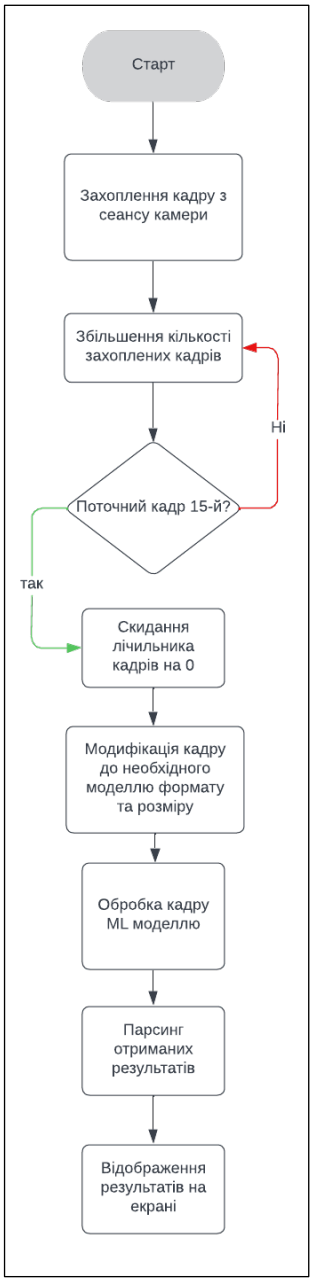

Рисунок 3.8 – Алгоритм розпізнавання образів за допомогою TFLite моделі

Основна різниця у порівнянні з CoreML полягає в тому, що ми самі виконуємо модифікацію вхідних даних до вимог моделі. Адже в попередньому варіанті цю функцію виконує Vision фреймворк самостійно. Крім того, ще одним етапом ми виконуємо перетворення отриманих результатів у структури даних для відображення на екрані і аналізу [10]. В такому випадку на етапі розробки необхідно бути дуже уважним, щоб не припуститись помилок на цих стадіях. До того ж, ці перетворення роблять алгоритм більш складним і збільшують час на інтеграцію моделі в додаток.

3.5 Контроль за апаратним забезпеченням пристрою

Досить важливим аспектом з точки зору розробника є саме правильне налаштування використання ресурсів мобільного пристрою. Адже якщо ми розглядаємо задачу розпізнаванняобразів в межах більш складних додатків з більшою кількістю функцій, то неправильне використання доступних ресурсів девайсу може призвести як до зниження ефективності роботи додатку вцілому, так і до погіршення роботу моделі машинного навчання.

Що стосується контролю за апаратним забезпеченням девайсу, яке буде використано під час роботи моделі розпізнавання, то в межах CoreML ми маємо можливість використати GPU, CPU або ANE (Apple Neural Engine)[12].

Apple Neural Engine (або ANE) – це тип NPU, що розшифровується як Neural Processing Unit. Це як GPU, але замість прискорення графіки NPU прискорює операції нейронної мережі, такі як згортки та множення матриць. Більшість нових iPhone та iPad мають Neural Engine, спеціальний процесор, який робить моделі машинного навчання дуже швидкими [1].

Коли ми створюємо екземпляр нашої моделі для подальшого використання та передбачень, ми використовуємо об’єкт MLModelConfiguration, щоб створити деякі спеціальні конфігурації. Дані конфігурації дають можливість встановити або змінити основні параметри моделі, за рахунок чого можна досягти кращих результатів розпізнавання та підвищити ефективність. Крім того, конфігурації дають можливість вказати, який пристрій буде використовувати модель для

прогнозування, наприклад графічний процесор. Додатково, в межах створення вищезазначеного об’єкту розробник має можливість обмежити модель використанням певної категорії обчислювальних пристроїв[12].

Однією з конфігурацій є MLComputeUnits, за допомогою якої ми можемо визначити набір конфігурацій процесора, який модель може використовувати для прогнозування [6]. Ось список доступних налаштувань:

– all (параметр, який дозволяє моделі використовувати всі доступні обчислювальні блоки, включаючи нейронні механізми);
– cpuOnly (параметр, який обмежує дозволяє моделі використовувати лише ЦП);
– cpuAndGPU (параметр, який дозволяє моделі використовувати обидва CPU та GPU, але не нейронний двигун);
– cpuAndNeuralEngine (параметр, який дозволяє використовувати моделі як центральний процесор, так і нейронний механізм, але не графічний процесор).

Вибір налаштування на етапі створення конфігурацій для ініціалізації моделі дає нам можливість контролювати її продуктивність моделі і використання апаратного забезпечення [3].

Тепер розглянемо даний процес при роботі з моделлю у форматі TFLite (див.рис.3.9).

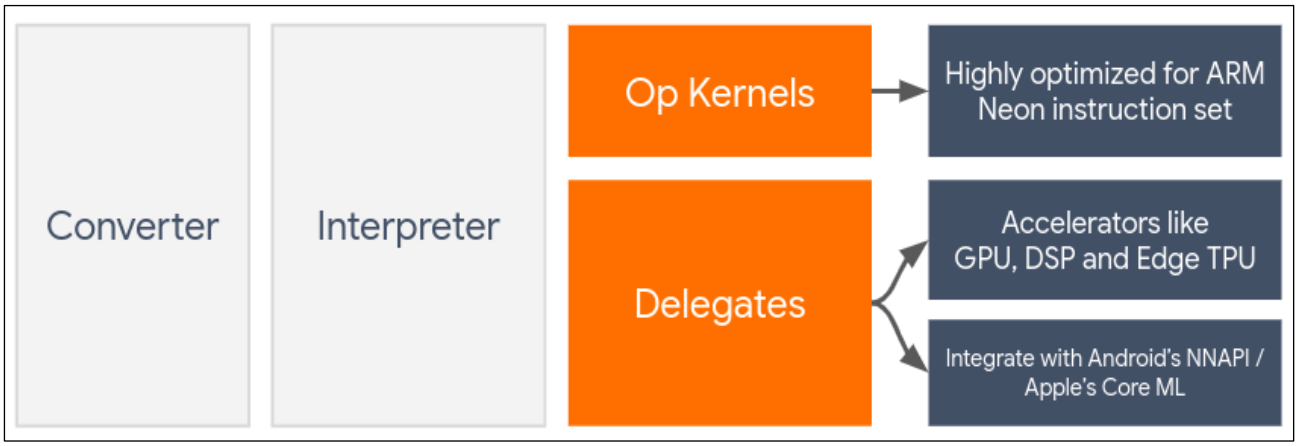

Рисунок 3.9 – Схема роботи TensorFlowLite

Контроль апаратного забезпечення за допомогою моделі TensorFlowLite ML реалізовано за допомогою делегатів. Делегати дозволяють апаратне прискорення моделей TensorFlow Lite за допомогою вбудованих прискорювачів, таких як графічний процесор і процесор цифрових сигналів (DSP). За замовчуванням TensorFlow Lite використовує ядра ЦП, оптимізовані для набору інструкцій ARM Neon. Однак ЦП – це багатоцільовий процесор, який не обов’язково оптимізований для важкої арифметики, яка зазвичай зустрічається в моделях машинного навчання.

Для контролю апаратного забезпечення, що використовує модель доступні наступні делегати:

- делегат GPU (можна використовувати як на Android, так і на iOS. Він оптимізований для запуску 32-розрядних і 16-розрядних моделей на основі плаваючої пам'яті, де є графічний процесор. Він також підтримує 8-розрядні квантовані моделі та забезпечує GPU продуктивність на рівні з їх версіями з плаваючою машиною.);
- делегат Core ML (для нових iPhone та iPad – для нових iPhone та iPad, де доступний Neural Engine, ви можна використовувати делегат Core ML, щоб прискорити роботу моделі для 32 або 16-розрядних моделей із плаваючою комою. Neural Engine доступний для мобільних пристроїв Apple з A12 SoC або вище).

Таким чином, вибір правильного делегату при роботі з TFLite дає розробнику можливість впливати на апаратне забезпечення для підвищення ефективності роботи моделі.

4 ПОРІВНЯННЯ ЗРУЧНОСТІ ТА ЯКОСТІ ІНТЕГРАЦІЇ

4.1 Розробка критеріїв порівняння

Для того, щоб порівняти зручність та якість інтеграції CoreML та TFLite моделей для розпізнавання образів в мобільному додатку, перш за все, необхідно розробити перелік критеріїв за якими і буде відбуватись подальше зіставлення.

Для початку розробимо критерії порівняння з точки зору розробника. Адже саме він є головним актором на етапі розробки додатку, який включає в себе безпосередню інтеграцію.

Оскільки першим кроком в процесі побудови мобільного додатку з використанням моделі машинного навчання є підключення відповідних бібліотек CoreML і TensorFlowLite. Саме тому, логічно першим критерієм обрати зручність підключення даних фреймворків до проекту.

На наступному кроці в процесі інтеграції ми виконуємо модифікації вхідних даних до вимог моделі. Беручи до уваги цей факт, можемо визначити наступний критерій, а саме: необхідність реалізації методів перетворення вхідних кадрів.

Після того, як ми отримали результати, нам необхідно передати їх на користувацький інтерфейс. Таким чином, ми отримуємо ще один критерій, який полягає у необхідності додаткової обробки результатів розпізнавання.

Оскільки існує різниця між методами та способами управлінням апаратним забезпеченням, що використовує модель, то нам необхідно додати ще один критерій порівняння – гнучкість контролю за архітектурними складовими пристрою, які залучені до використання.

Досить важливим аспектом для розробника на етапі інтеграції є наявність чіткої та зручної документації про вхідні та вихідні дані моделі. Саме цей фактор і додамо як один з критеріїв оцінювання.

Також в процесі розробники дуже важливо враховувати кількість часу, яка необхідна. Цей фактор також візьмемо у якості критерія порівняння.

Тепер перейдемо до визначення пунктів, за якими можемо порівняти якість інтеграції моделі машинного навчання з точки зору кінцевого користувача.

Одним з головним аспектів в цьому випадку є час роботи моделі. Адже необхідно створювати додатки таким чином, щоб вони працювали максимально швидко і не займали багато часу.

Наступним дуже важливим критерієм є якість отриманих результатів. тому що не тільки швидкість визначає високий рівень інтеграції, а і точність отриманих даних в результаті розпізнавання.

Також важливу роль відіграє кількість пам’яті яку займає додаток на смартфоні. Це також винесемо як окремий пункт якісного порівняння.

І ще одним пунктом для кінцевого користувача є енергоефективність. Додаток не повинен споживати занадто багато енергії тим самим розряджаючи пристрій. Адже це не тільки погано з точки зору досвіду користувача, а й з точки зору поганого впливу на батарею смартфону.

Таким чином, проаналізувавши основні моменти, які можуть характеризувати якість інтеграції моделі машинного навчання в мобільному додатку ми розробили низку критеріїв. З точки зору розробника маємо наступні:

- зручність підключення CoreML і TensorFlowLite фреймворків до проекту;
- необхідність реалізації методів перетворення вхідних кадрів;
- необхідність додаткової обробки результатів розпізнавання;
- гнучкість управління апаратним забезпеченням, що використовується моделлю;
- наявність чіткої та зручної документації про вхідні та вихідні дані моделі;
- кількість часу, необхідного для інтеграції.

Що стосується критеріїв зі сторони кінцевого користувача, то ми визначили три основні фактори, а саме:

- швидкість роботи моделі;
- якість результатів роботи моделі;
- кількість необхідної пам’яті;
- енергоефективність.

4.2 Порівняння за визначеними критеріями

В таблиці 4.1 зазначено порівняння зручності інтеграції CoreML та TFLite моделей з точки зору розробника мобільного застосунку.

Таблиця 4.1 – Порівняння зручності інтеграції з точки зору розробника

Критерій	CoreML	TFLite
Зручність підключення необхідних фреймворків до проекту.	CoreML та Vision є нативними фреймворками, тому жодних додактових кроків для інтеграції виконувати непотрібно.	Фреймворк доступний лише у форматі CocoaPods. Саме тому треба виконати низку дій для того, щоб підключити його до проекту.
Необхідність реалізації методів перетворення вхідних кадрів.	Фреймворк Vision бере на себе всю відповідальність за модифікацію вхідних даних. Не потрібно додактово реалізовувати якісь методи.	Необхідно додактово створити методи приведення вхідного зображення до формату та розмірів, що вимагає модель.
Необхідність додаткової обробки результатів розпізнавання	Фреймворк Vision виконує обробку отриманих результатів і повертає цілісні структурні об’єкти з даними.	Необхідно реалізувати методи перетворення отриманих результатів у структури даних для подальшого використання.
Гнучкість управління апаратним забезпеченням, що використовується моделлю.	Є можливість налаштувати даний фактор наступним чином: надати системі можливість вибирати оптимальне апаратне	Є можливість контролювати апаратне забезпечення за допомогою делегатів, а саме: CPU, GPU або ANE.

Кінець таблиці 4.1

Критерій	CoreML	TFLite
	забезпечення з доступних, вкиористовувати тільки CPU, використовувати CPU або GPU, використовувати CPU або ANE.	
Наявність чіткої та зручної документації про вхідні та вихідні дані моделі.	Xcode може представляти всю інформацію про входи та виходи конкретної моделі, детальний опис виходу та можливість тестувати модель без кодування.	Має лише зовнішню документацію з інформацією про входи та виходи.
Кількість часу, необхідного для інтеграції.	Досвідченому розробнику необхідно від 6 до 8 годин.	Досвідченому розробнику необхідно від 16 до 24 годин.

Після порівняння зручності інтеграції даних форматів моделей в мобільний додаток для розпізнавання образів, ми отримали чіткий результат. Для розробника мобільного додатку для платформи iOS більш зручним форматом моделі машинного навчання є CoreML модель. Адже необхідний для інтеграції фреймворк є нативним, інтеграція передбачає менше дій, наявна чітка і зручна документація, надає більш гнучке API для управління апаратним забезпеченням і потребує менше часу.

Перейдемо до аналізу якості з точки зору кінцевого користувача. В таблиці 4.2 зазначено результати порівняння за вище обраними критеріями. Необхідно зазначити. що для порівняння якісних характеристик було взято одну й ту саму yolov5n модель, яка була тренована на одному і тому ж наборі даних, але у двох різних форматах: yolov5m.mlmodel та yolov5n.tflite.

Таблиця 4.2 – Порівняння якості інтеграції з точки зору кінцевого користувача

Критерій	CoreML	TFLite
Швидкість роботи моделі.	Потребує від 45 до 52 мілісекунд.	Потребує від 83 до 94 мілісекунд.
Якість результатів роботи моделі	Знаходить більшу кількість образів і точно ідентифікує.	Кількість розпізнаних образів набагато менше, ідентифікація менш точна.
Кількість необхідної пам’яті	Займає 8МБ.	Займає 4МБ.
Енергоефективність	Вплив на акумулятор смартфону складає в середньому 51%.	Вплив на акумулятор смартфону складає в середньому 83%.

Результати порівняння якості показали, що CoreML в iOS додатку працює вдвічі швидше і дає біль точні результати розпізнавання. Незважаючи на те, що TFLite займає менше пам’яті, CoreML є більш енергоефективним і продуктивним.

Таким чином в результаті порівняння можемо зробити висновок, що для реалізації задач розпізнавання образів за допомогою моделей машинного навчання на iOS варто використовувати CoreML моделі. Адже цей формат є більш зручним для розробника і більш якісним з точки зору кінцевого користувача. Саме використання даної технології для реалізації вищезазначеної задачі надасть дуже гарні переваги з точки зору розробки.

По-перше, за рахунок зручності в інтеграції ми зможемо отримати перевагу в часі. Адже за рахунок дуже зручного API в межах CoreML фреймворку навіть не дуже досвідчений розробник змоде виконати інттеграцію. Крмі того, інтеграція потребує значно менше часу, ніж реалізація тіє ж самої задачі, але з використанням TensorFlowLite.

По-друге, проведені тести щодо швидеості та якості роботи відповідних моделей, які були треновані на однакових наборах даних, показали, що CoreML працює помітно швидше, та якість детекції значно краща, що полягає в кількості знайдених та ідентифікованих об’єктів.

5 АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

5.1 Постановка практичної задачі

В результаті проведеного дослідження було отримано результат, щодо інструментів, які необхідно застосовувати для реалізації задач розпізнавання образів на iOS. CoreML є найбільш зручним як з точки зору розробника так і з точки зору кінцевого користувача для створення подібних додатків.

Для того, щоб остаточно закріпити результати дослідження на практиці необхідно реалізувати комплексний мобільний додаток з використанням кількох моделей машинного навчання для розпізнавання образів.

Однією з найпопулярніших футбольних вправ, які використовуються в тренувальному циклі, є набивання м'яча. Основна ідея вправи полягає в тому, щоб підбити м’яч ногами стільки разів, скільки можливо, не впустивши м’яч на землю. Виходячи з цього основною задачую мобільного додатку є можливість розраховувати кількість набиваннь м’яча за допомогою камери та моделей машинного навчання для виявлення пози м’яча та людини.

5.2 UML проектування мобільного додатку

Для того, щоб виконати проектування даного мобільного застосунку, було обрано метод, який базується на використанні UML діаграм. Адже даний підхід дає можливість чітко візуалізувати та змоделювати майбутнє програмне забезпечення з усіх ракурсів та визначитись з кінцевими вимогами до майбутнього додатку.

У випадку реалізації мобільного додатку для підрахунку набивання шляхом інтеграції моделей машинного навчання найкращим варіантом для представлення кінцевого результату розробки є діагарама діяльності.

Блок-схеми, є типом графічного представлення, яке можна використовувати в розробці програмного забезпечення для моделювання потоку дій або процесів у системі. Вони особливо корисні для проектування складних систем з кількома видами діяльності або процесами, які необхідно координувати.

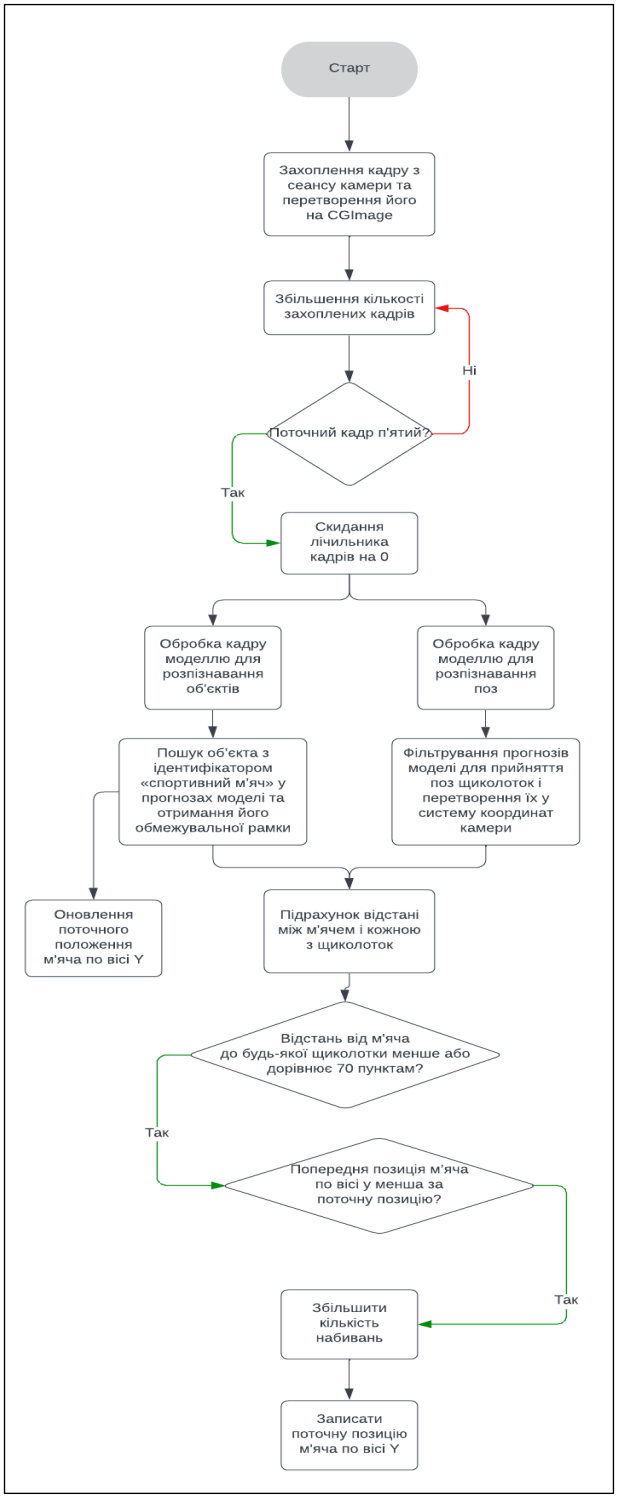

Рисунок 5.1 – Діаграма діяльності мобільного додатку

Для реалізації даного мобільного застосунку необхідно використати дві моделі машинного навчання, а саме модель для естімації поз людини та модель для розпізнавання об’єктів. Як зображено на діаграмі основна ідея полягає в тому, що одна модель розпізнає частини тіла людини в кадрі, а в нашому випадку нас

цікавить лише стопа, а друга модель шукає м’яч. На базі інформації про позиції стоп та м’яча ми можемо порахувати, чи відбувалась взаємодія між цими об’єктами та зарахувати набивання.

З точки зору реалізації даний мобільний додаток має бути реалізований для операційної системи iOS з використанням CoreML бібліотеки та відповідних моделей. Архітектурою даного застосунку має бути класична «MVC» архітектура.

Таким чином, можемо зробити висновок, що основними ключовими інструментами реалізації функціоналу даного мобільного додатку мають бути дві моделі розпізнавання образів та побудова взаємодій між ними в межах одного застосунку.

5.3 Реалізація застосунку

Даний мобільний додаток було реалізовано на мові програмування Swift в середовищі розробки Xcode. Саме ці інструменти необхідні для реалізації додатків для операційної системи iOS.

Для аналізу поз людини на відповідних кадрах з камери було обрано модель PoseNet у форматі mlmodel. Дана модель виявляє 17 різних частин тіла або суглобів: очі, вуха, ніс, плечі, стегна, лікті, коліна, зап’ястки та щиколотки. У цій версії використовується архітектура MobileNetV1 із множником ширини 0,75 і вихідним кроком 16. Використання даної моделі надає нам можливість отримати точні координати розташування будь-якої з вищеперелічених частини тіла, якщо вони присутні в кадрі.

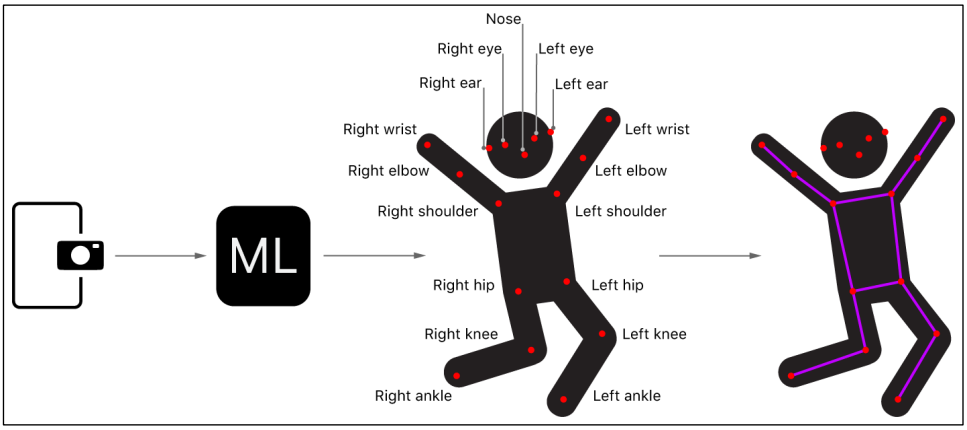

Рисунок 5.2 – Візуалізація роботи PoseNet моделі

Наступним кроком було виконано інтеграцію моделі для розпізнавання об’єктів, а саме моделі yolov5n. Що стосується даної моделі, то вона дає можливість іжентифікувати та отримати точні координати певного списку об’єктів, серед яких є м’яч. Саме можливість розпізнати даний предмет є необхідною для реалізації кінцевої задачі додатку.

На рисунку 5.3 зображена структура файлів даного мобільного застосунку.

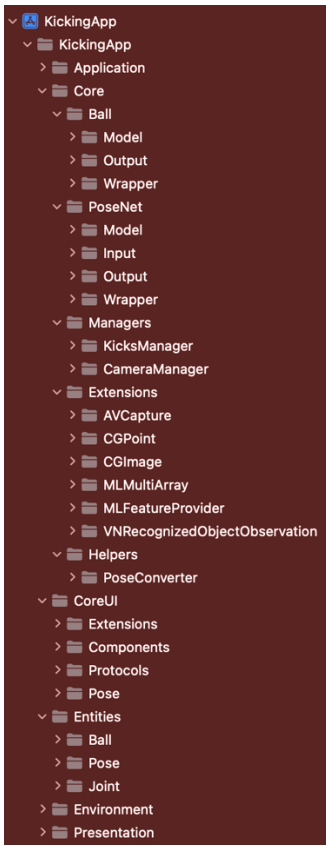

Рисунок 5.3 – Структура файлів мобільного додатку

Уся логіка, яка пов’язана з детекцією основних складових зображення, інкапусльована в межах окремих класів. Дані класи в свою чергу створюють об’єкти відповідних моделей, виконують деякі перетворення вхідних зображень, відповідають за обробку вихідних даних та передають усю цю інформацію в центральний клас, який виступає свого роду фасадом. В межах даного класу-фасаду відпувається порівняння результатів з моделей, виконуються деякі підрахунки та робиться висновок, чи було виконане успішне набивання м’яча.

З точки зору користувацького інтерефейсу додаток представляє собою простий стартовий екран та основний, на якому користувач бачить кадри з камери (див. рис. 5.4).

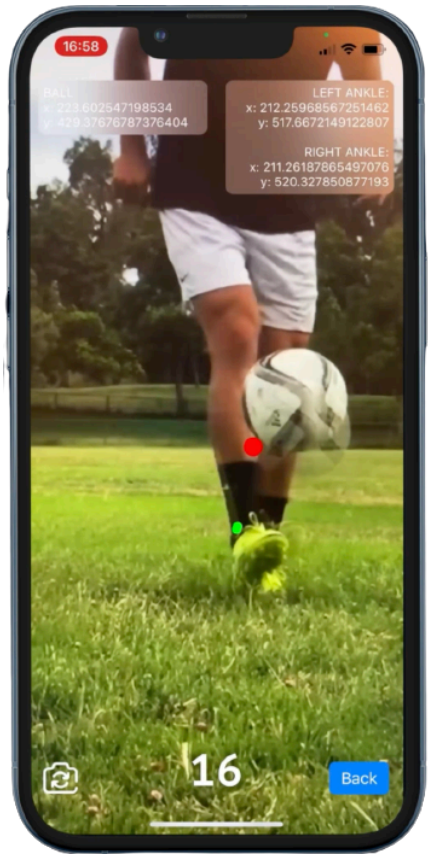

Рисунок 5.4 – Користуваціький інтерфейс мобільного застосунку

У верхній частині екрану розташовані невеличкі компоненти з текстовими даними. В лівому компоненті відображається поточна позиція м’яча а в правому – позиції лівої та правої стопи.

За допомогою невеликих індикаторів, позиція яких оновлюється в режимі реального часу, як і відповідні значення в верхні частині екрану, візуалізується поточні позиції розпізнаних образів. Червоним кольором відображається позиція м’яча, а зеленим – позиції стоп.

Що стосується нижньої частини екрану, то тут розташована кнопка зміни камери з фронтальної на основну та навпаки. Пору з даною кнопкою розташоване текстове поле, яке безпосередньо відображає кількість набиваннь. Ну і в правому нижньому кутку розташована кнопка, яка дає можливість користувачу повернутись на початковий екран.

Користувацький інтерфейс було розроблено за допомогою UIKit Framework. Цей фреймворк містить у собі спеціальні класи та компоненти, які потрібні для розробки інтерфейсу для платформи iOS.

Інтеграція моделей машинного навчання, які були обрані та побудова взаємодії з ними відбувалась за допомогою фреймворку CoreML та бібліотеки Vision. Адже базуючись на попередніх результатах дослідження саме ця комбінація інструментів показала найкращі результати у розрізі інтеграції моделй для розпізнавання образів на iOS.

Для того, щоб підвищити ефективність роботи застосунку на етапі конфігурування та створення об’єктів моделей було рознесено обробку даних кожною з моделей на окрему складову апаратного забезпечення. Модель для розпізнавання поз використовує лише можливості центрального процесора, в той час як модель розпізнавання об’єктів отримала можливість користуватись або центральним процесором або нейронним двигуном.

Результатом реалізації є мобільний додаток, який використовує дві окремі моделі машинного навчання для підрахунку кількості набивання футбольного м’яча базуючись на зображеннях з камери девайсу.

Подальшими перспективами розвитку даного мобільного застосунку є створення соціальної мережі для змагань між людьми або окремими групами людей. Додатково функціонал даного додатку можна росширити шляхом додавання інших вправ, наприклад, віджимань. Розширення функціональних можливостей не вплине на основну концепцію, а саме комлпексне використання кількох моделей машинного навчання для отримання певних спільних результатів, але й дасть можливість зробити застосунок більш цікавим для різних груп користувачів.

5.4 Приклади найцікавіших алгоритмів та методів

Ключовим алгоримом, яки представляє собою основну логіку додатку, є алгоритм обробки отриманих результатів з моделей, який робить висновок, чи відбулось набивання, чи ні:

```
func checkForKick(
      ball: CGPoint,
      leftAnkle: CGPoint,
      rightAnkle: CGPoint
) {
   guard
       !kicksOnCooldown,
       let leftAnkleDistance = calculateDistance(ball: ball,
                                                 ankle: leftAnkle),
       let rightAnkleDistance = calculateDistance(ball: ball,
                                                  ankle: rightAnkle)
   else {
       return
   }
   let minimalDistance = min(leftAnkleDistance, rightAnkleDistance)
   if minimalDistance <= kickThreshold, previousBallPosition <= ball.y
   {
       kicks += 1
       kicksOnCooldown = true
       setupCooldownTimer()
   }
   previousBallPosition = ball.y
}
```

Перш за все дана функція підраховує підстань від м’яча до кожної з ніг. Якщо відстань від м’яча до однієї з ніг менша, ніж мінімальне значення, то це означає, що відбулась взаємодія. Після цього необхідно перевірити, чи змінився напрямок руху м’яча. Для цього ми порівнюємо нове значення позиції м’яча з попереднім. Базуючись на цих основних умовах функція може збільшити лічильник набивань у разі виконання усіх необхідних умов.

Що стосується обробки отриманих даних з моделей, то в межах застосунку реалізовані відповідні функції, які виконують перетворення вхідних зображень та обробку вихідних даних. Оскільки моделі отримують на вхід зображення певного розміру, який вказаний в їх специфікації, то і результати обробки моделями ми отримуємо в системі координат вхідних зображень. А для того, щоб отримати можливість робити якісь висновки щодо взаємодії основних об’єктів, необхідно перш за все перевсти їх у єдину систему координат. Дана структура містить в собі статичну функцію, яка й виконує вищезазначену конвертацію:

```
struct PoseConverter {
   static func getPosition(
             position: CGPoint,
             frameSize: CGSize,
```

```
            viewSize: CGSize
    ) -> CGPoint {
      let yPercentage =  position.y / frameSize.height
      let xPercentage = position.x / frameSize.width
      return .init(x: viewSize.width * xPercentage,
                   y: viewSize.height * yPercentage)
    }
}
```

У якості базової системи координати, до якої буде відбуватись конвертація вихідних значень моделей, була обрана система координат екрану, адже таким чином ми не тільки отримаємо можливість порівнювати отримані значення, а й зможемо реалізувати певні аспекти користувацького інтерфейсу. За допомогою використання саме системи координат екрану з камерою ми отримали можливість підсвічувати ноги та м’яч за допомогою кольорових індикаторів.

Для підвищення якості користувацького інтерфейсу та оптимізації роботи додатку було прийняте рішення аналізувати за допомогою моделі не кожен кадр, а кожний п’ятий кадр. Це значення є найбільш зручним та оптимальним як з точки зору бізнес логіки так і з точки зору користувацького досвіду.

Однією з ключових функціональних можливостей додатку є можливість отримувати кадри з камери в режимі реального часу. Реалізація даного функціоналу також містить певні цікаві моменти, на які варто звернути увагу.

Найбільш цікавим та важливим моментом, який стосується саме камери, є первинна обробка вхідних кадрів. За реалізацію даного функціоналу відповідає наступна функція:

```
func handleOutputData(sampleBuffer: CMSampleBuffer) {
        if let pixelBuffer = sampleBuffer.imageBuffer {
            var image: CGImage?
            VTCreateCGImageFromCVPixelBuffer(pixelBuffer,
                                             options: nil,
                                             imageOut: &image)

            CVPixelBufferUnlockBaseAddress(pixelBuffer, .readOnly)

            DispatchQueue.main.async { [weak self] in
                self?.onDataOutput?(image)
            }
      }
}
```

Оскільки моделі, які використанні в межах додатку, на вхід потребують зображення у певному форматі, який не співпадає з тим форматом, який передає камера, то дана функція виконує відповідні перетворення, для того, щоб кадр далі міг бути переданий на подальшу обробку.

Дана функція також являє собою одну з основних на базі яких і будується загальна логіка додатку шляхом поєднання результатів взаємодії різних його компонентів.

5.5 Тестування розробленого програмного забезпечення

Тестування програмного забезпечення – це процес оцінювання програмного додатку або системи для виявлення дефектів, помилок або інших проблем, які можуть вплинути на їх продуктивність, надійність або функціональність. Метою тестування програмного забезпечення є переконатися, що програмне забезпечення відповідає вимогам і очікуванням його користувачів і зацікавлених сторін.

Тестування програмного забезпечення є критично важливим компонентом розробки та обслуговування програмного забезпечення. Це допомагає переконатися, що програмне забезпечення відповідає стандартам якості, виявляє помилки, зменшує ризики, забезпечує відповідність і задовольняє користувачів і зацікавлених сторін.

Під час тестування даного мобільного додатку було проведено такі види: стрес-тестування та перевірка на довге використання. Оскільки даний додаток містить в собі достатньо трудоємні задачі для мобільного пристрою, то досить важливо було провести саме ці види тестів. Адже вони допомжуть підтвердити якість інтеграції моделей машинного навчання за допомогою CoreML фреймворку, тим самим підтвердити результати дослідження.

Стрес-тестування – це тип тестування програмного забезпечення, який оцінює поведінку програмної системи або програми в умовах високого навантаження або стресу. Метою стрес-тестування є виявлення обмежень

продуктивності системи, забезпечення її стабільності та оцінка її здатності справлятися з великими навантаженнями.

Довгострокове тестування – це тип тестування програмного забезпечення, який передбачає безперервне оцінювання програмної системи або програми протягом тривалого періоду часу, як правило, тижнів або місяців. Метою тривалого тестування є оцінка стабільності, продуктивності та надійності програмного забезпечення протягом тривалого періоду часу за нормальних умов використання.

Під час тестування довгим використанням, була перевірена працездатність мобільного додатку системи витримувати довге навантаження. Результат тестування навантаженням зображено на рисунку 5.5.

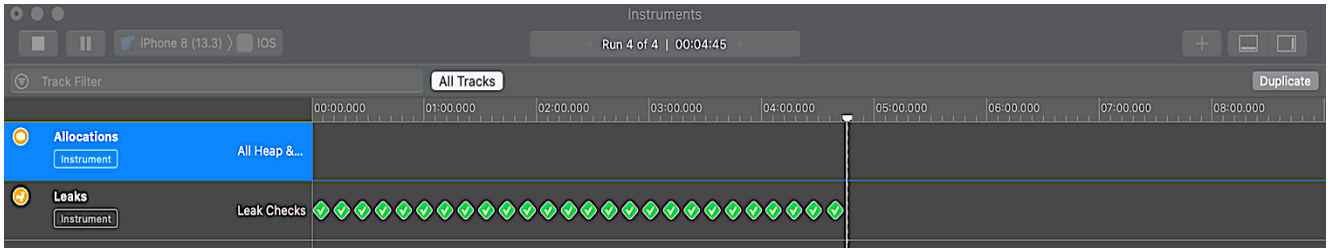

Рисунок 5.5 – Результати навантажувального тестування

Весь процес був записаний за допомогою спеціального інструменту тестування на виток пам’яті «Xcode Memory Leaks». У ході довготривалого використання не було виявлено витоків пам’яті, помилок використання або збільшення використання оперативної пам’яті девайсу.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було досліджено інтеграцію моделей машинного навчання в мобільні додатки на платформі iOS. Під час виконання роботи було детально проаналізовано предметну область, розглянуто питання теорії розпізнавання образів та використання мобільних додатків для виконання такого роду задач.

Для вибору бібліотек та форматів моделей машинного навчання для дослідження було проведено аналіз ринку наявних технологій та їх особливостей для використання в межах поставленої задачі. Результатами аналізу став вибір двох фреймворків, що використовуються для інтеграції моделей з мобільними додатками, а саме: CoreML від компанії Apple і TensorFlowLite від компанії Google. Перший було обрано як нативне рішення, яке оптимізоване спеціально для використання в межах iOS пристроїв. Що стосується другої бібліотеки, то вона є найбільш популярною серед аналогів та реалізована з використанням іншого архітектурного підходу у порівнянні з CoreML [9].

Аналіз особливостей кожної з бібліотек, що були обрані для дослідження, включає детальне дослідження можливостей, які надають дані підходи, на рівні апаратного забезпечення. Адже гнучкість в даному аспекті надає можливість підвищувати швидкість роботи фінального додатку та оптимізувати використання ресурсів мобільного пристрою. Було розглянути варіанти контролю за апаратним забезпеченням, яке використовує модель машинного навчання для виконання задач розпізнавання образів на відповідному девайсі.

Під час роботи було виконано моделювання мобільного додатку, який містить інтеграцію з обома форматами моделей машинного навчання та безпосередньо виконує задачу розпізнавання. Додаток дає можливість обирати формат моделі, яка буде використана, і виконувати операцію розпізнавання в режимі реального часу на кадрах, які отримані з камери пристрою.

Під час виконання дослідження було створено список критеріїв, за якими необхідно порівнювати зручність інтеграції моделі відповідного формату з точки

зору розробника та якість роботи інтегрованої бібліотеки з точки зору кінцевого користувача.

Дослідження включає в себе порівняння обраних технологій за попередньо розробленими критеріями. Результати порівняння з точки зору розробника показали, що CoreML модель є більш зручною, адже необхідний для інтеграції фреймворк є нативним, інтеграція передбачає менше дій, наявна чітка і зручна документація, надає більш гнучке API для управління апаратним забезпеченням і потребує менше часу. Після проведення порівнянь якості інтеграції з точки зору кінцевого користувача, то CoreML показує кращі результати у швидкості та якості розпізнавання образів, а також є біль енергоефективним з точки зору впливу на батарею мобільного пристрою. На основі проведеного дослідження та аналізу можемо зробити висновок, що CoreML моделі є найбільш зручними та якісними для виконання задач розпізнавання образів на платформі iOS.

В межах дослідження було проведено апробацію отриманих результатів шлязом реалізації комплексного мобільного застосунку, який використовує кілька моделей машинного навчання інттегрованих за допомогою CoreML фреймворку. В результаті тестування отриманих результатів було розроблено мобільний застосунок для операційної системи iOS, який виконує підрахунок кількості набивань футбольного м’яча на основі кадрів з камери, позиції окремих частин тіла та м’яча в межах відповідних кадрів.

Перспективним напрямком розвитку дослідження поставленої задачі є розширення списку технологій, що порівнюються за визначеними критеріями, й аналіз можливостей навчати моделі на рівні мобільного додатку для подальшої інтеграції.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Заяць В. М., Камінський Р. М. Методи розпізнавання образів : Навч. посіб. для студ. Львів: Нац. ун-т "Львів. політехніка", 2004. 173 с.

2. Нікольський Ю. Системи штучного інтелекту: навч. посіб. / Ю. В. Нікольський та ін.; за наук. ред. В. В. Пасічника; 2-ге вид., виправл. та доповн. Львів: Магнолія-2006, 2013. 279 с.

3. Новотарський М.А. Штучні нейронні мережі обчислення / М.А. Новотарський Б.Б. Нестеренко., 2014.

4. Основи теорії розпізнавання образів. Режим доступу: http://kist.ntu.edu.ua/textPhD/tro2.pdf (дата звернення 01 грудня 2022)

5. Ткаченко Р. Засоби штучного інтелекту: навч. посіб. / Р. О. Ткаченко та ін. Львів: Нац. ун-т «Львів. політехніка», 2014. 204 с.

6. Штучний інтелект, машинне навчання та нейронні мережі: в чому різниця і для чого їх використовують. Режим доступу: https://evergreens.com.ua/ua/articles/machine-learning-overview.html (дата звернення 11 жовтня 2022)

7. Як працює machine learning та його застосування на практиці. Режим доступу: https://nachasi.com/tech/2019/01/31/yak-pratsyuye-machine-learning/ (дата звернення 12 грудня 2022)

8. Core ML. URL: https://developer.apple.com/documentation/coreml (дата звернення 14 грудня 2022)

9. Core ML Tools Overview URL: https://coremltools.readme.io/docs (дата звернення 17 грудня 2022)

10. Nils J. Nilsson. The Quest for Artificial Intelligence. – 1. – Cambridge University Press, 2009. 578 p.

11. TensorFlow Lite URL: https://www.tensorflow.org/lite (дата звернення 18 грудня 2022).