Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

СТВОРЕННЯ ПРОТОТИПУ СИСТЕМИ РЕКОМЕНДАЦІЙ НА ОСНОВІ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ

Текстова частина до кваліфікаційної випускної роботи

за спеціальністю «Інженерія програмного забезпечення»

Керівник кваліфікаційної роботи
асистент Франків. О.О.
_____
(підпис)
"___"_____ 2023 р.

Виконала студентка 4 р.н.
Зубрицька І.І.
_____
(підпис)
"___"_____ 2023 р.

Київ 2023

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Викладач кафедри інформатики,
канд. фіз-мат. наук, доц. _____ Гороховський С.С.
(підпис)
“___”_____ 2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на кваліфікаційну випускну роботу
студентці Зубрицькій Ірині Ігорівні
факультету інформатики 4 курсу бакалаврської програми
ТЕМА: Створення прототипу системи рекомендацій на основі розпізнавання об’єктів

Зміст ТЧ до кваліфікаційної роботи:

Індивідуальне завдання
Календарний план виконання роботи
Анотація
Вступ
1. Огляд технологій
2. Теоретичні засади для програмної реалізації
3. Опис програмного продукту

Висновки
Список джерел
Додатки

Дата видачі “___”_____2023 р. Керівник _____
(підпис)

Завдання отримала _____
(підпис)

Календарний план виконання роботи

Тема: Створення прототипу системи рекомендацій на основі розпізнавання об’єктів.

№ п/п	Назва етапу	Термін виконання	Примітка
1.	Узгодження теми та отримання завдання на кваліфікаційну роботу	01.10.2022	
2.	Огляд та аналіз технологій за обраною темою	15.10.2022	
3.	Визначення вимог до рішення	15.11.2022	
4.	Планування виконання роботи	01.12.2022	
5.	Початок розробки застосунку	01.12.2022	
6.	Кінець розробки застосунку	28.02.2023	
7.	Початок написання текстової частини	01.03.2023	
8.	Створення презентації	09.05.2023	
9.	Попередній захист	11.05.2023	
10.	Кінець написання текстової частини	18.05.2023	
11.	Фінальна корекція	20.05.2023	
12.	Подання роботи на перевірку	24.05.2023	
13.	Захист	Згідно з розкладом	

Студентка Зубрицька І.І.

Керівник Франків О.О.

“___”_____2023 р.

Зміст

Анотація

Метою цієї кваліфікаційної роботи є дослідження, як можна поєднати машинне навчання та доповнену реальність для визначення справжніх розмірів об’єктів, та створення прототипу мобільного iOS-застосунку, який рекомендує користувачу рецепти за допомогою розпізнавання інгредієнтів.

До основних функцій застосунку належать розпізнавання продуктів з камери, визначення їхніх розмірів і мас та підбір рецептів відповідно до розпізнаних продуктів з перерахунком кількості інгредієнтів.

Для розробки прототипу було використано середовище розробки Xcode, мова програмування Swift, фреймворк Vision для роботи з моделлю машинного навчання, фреймворк ARKit для використання можливостей доповненої реальності, Firebase Realtime Database для зберігання та отримання рецептів, фреймворки SwiftUI та UIKit для створення інтерфейсу користувача.

Ключові слова: мобільний застосунок, iOS, модель машинного навчання, розпізнавання, об’єкт, доповнена реальність, пристрій.

Вступ

Сучасні технології дозволяють створювати програмні продукти, які значно полегшують життя людей. Часто в основі таких програм лежить машинне навчання та доповнена реальність, які мають безліч застосувань.

Одним із прикладів застосування та поєднання вищезгаданих технологій є розробка системи підбору рецептів, яка може покращити життя людей у сфері харчування. Вона може працювати за допомогою розпізнавання продуктів та визначення їхніх розмірів, а також пристосовувати рецепт до розпізнаної кількості інгредієнтів.

Такий програмний продукт може вирішити кілька проблем. Перша проблема - нестача часу на пошук рецептів, коли людина має кілька продуктів і хоче всі з них використати у страві. Інша проблема - відсутність досвіду в приготуванні страв та одноманітний раціон. Також люди часто стикаються з проблемою невідповідності пропорцій інгредієнтів, які вони мають, та які є у рецепті, тому страва може не вдатися через порушення пропорцій. Всі ці проблеми може вирішити програма, підібравши рецепти за списком продуктів та їхніми кількостями, які є у користувача.

Отже, створення системи, що базується на розпізнаванні об’єктів та підборі рецептів, є актуальним завданням, яке може спростити та покращити якість життя. Вона може стати корисною для тих, хто бажає заощадити час на приготування їжі та урізноманітнити раціон.

Тому мета цієї роботи полягає у дослідженні, як можна поєднати та використати машинне навчання та доповнену реальність так, щоб змогти визначити реальні розміри об’єктів. За результатами досліджених методологій слід розробити прототип системи рекомендацій рецептів.

Робота складається з трьох основних розділів:

У першому розділі робиться огляд на технології, які досліджуються в цій роботі. Також він містить огляд на готові застосунки, що працюють за подібним принципом.

Другий розділ містить проблеми у розробці заданого програмного продукту та способи їх вирішення. Методи вирішення містять підбір даних для створення моделі для розпізнавання об’єктів, взаємодію з моделлю, математичні розв’язки для визначення розмірів об’єктів та їхніх мас та перерахунок кількості продуктів в рецептах.

У третьому розділі описано готовий програмний продукт та вибір технологій, визначено технічне завдання та вимоги, описані використані засоби розробки, структура програмного продукту та принцип його роботи.

1. Огляд технологій та аналіз схожих програмних продуктів

1.1. Принцип роботи машинного навчання та розпізнавання об’єктів

Машинне навчання - це галузь штучного інтелекту, яка дозволяє системі навчатися на наборах даних, використовуючи різні алгоритми, а потім передбачати результати. Зараз машинне навчання використовується у багатьох сферах, наприклад у безпілотних автомобілях для виявлення пішоходів або в медицині для виявлення захворювань, і таким чином вже допомагає людству. Однак це точно лише еволюції цієї галузі, оскільки вона продовжує активно розвиватися та покращуватись за рахунок нових методів та алгоритмів.

Одним з підходів машинного навчання є нейронна мережа. Це математична модель, яка імітує роботу мозку і складається із взаємопов’язаних нейронів. Нейронні мережі вирішують безліч задач, однією з яких є розпізнавання об’єктів. Це завдання комп’ютерного зору, ціллю якого є виявлення місцезнаходження та класифікація об’єктів. Результатом виконання цієї задачі є список об’єктів, які містять таку інформацію: мітка, ймовірність та місцезнаходження у вигляді обмежувальної рамки.

Для розпізнавання об’єктів використовуються згорткові нейронні мережі(Convolutional Neural Networks, CNN), які працюють з зображеннями як матрицями пікселів певного кольору в кілька етапів. Під час етапу згортки на зображення накладається фільтр, тобто відбувається поелементне множення значень з фільтру на значення фрагменту зображення, на які накладений фільтр, та результати додаються. Вихідна матриця містить деяку інформацію про ознаки на зображенні. Наступний етап - це пулінг(pooling), який потрібний для зменшення розміру матриці, і це дозволяє оптимізувати обрахунки. Розмір зменшується за рахунок того, що з певного квадратного фрагменту обирається максимальне або середнє

значення. Процес згортки та пулінгу може відбуватися кілька разів, щоб ще краще виокремити певні ознаки. Матриця, яку отримали на виході, перетворюється у вектор, який далі використовується у мережі нейронів.

Незважаючи на вже зроблений великий прогрес у сфері розпізнавання об’єктів, це складний процес, в процесі розробки якого виникає багато проблем. Для початку, це можуть бути різні ракурси, і завдання нейронної мережі - це вміти розпізнавати усі або хоча би більшість з них. Іншим прикладом може бути деформація, наприклад тварина, що знаходиться у різних позах. Крім того, іноді об’єкти видно не повністю, вони можуть бути перекриті іншими об’єктами, наприклад людина, яка їде на велосипеді, перекриває собою його частину. Варто теж згадати про якість зображень, часто вони бувають поганого розширення, розмиті, засвічені чи навпаки затемні. Останнє і не менш важливе - це різноманіття об’єктів, бо якщо нейронна мережа розпізнає будівлі, це може бути як магазин, так і багатоповерхівка, вони можуть бути різного кольору і так далі.

Щоб машина могла навчатися, їй треба дати початковий набір даних та очікувані результати. На основі цих результатів вона підбирає алгоритм, який потім буде використовувати для вирішення нових задач. У випадку з розпізнаванням об’єктів, на вхід даються зображення і файл з анотаціями. Дані у файлі анотацій мають мати певну структуру, що містить інформацію про назву файлу та місцерозташування об’єкту. Для останнього використовують початкову координату, ширину та висоту.

В залежності від інструментів, які використовує розробник, файл з анотаціями може мати різні формати. Одним з найпопулярніших є JSON(JavaScript Object Notation) формат.

У лістингу, наведеному нижче, можна побачити приклад частини файлу анотацій з інформацією для одного зображення, де "imagefilename" – назва файлу з зображенням, "annotation" – список анотацій. Анотація

складається з "coordinates" – місцерозташування об’єкта та "label" – мітки, або назви об’єкта.


```
{
  "imagefilename": "breakfast_0.png",
  "annotation": [
    {
      "coordinates": {
        "y": 156.062,
        "x": 195.122,
        "height": 148.872,
        "width": 148.03
      },
      "label": "Waffle"
    }
  ]
} [1]
```


Іншими популярними форматами є COCO JSON, Pascal VOC XML, TFRecord, Yaml, CSV та TXT.

Також для створення моделі машинного навчання додається набір даних для валідації. Він має відрізнятися від тренувальних даних, оскільки його метою є донавчання, і тому використання однакових даних буде неефективним. Це працює так, що система пробує сама оцінити зображення, а потім порівнює свої результати з очікуваними, і в разі потреби може вносити корективи для покращення точності. Крім того, є ще тестовий набір даних, який є завершальним етапом і використовується для оцінки остаточної ефективності моделі після її тренування та валідації. Основною відмінністю від валідаційних даних є те, що тестові не впливають на ефективність та не вносять змін у модель.

1.2. Огляд технології доповненої реальності

Технологія доповненої реальності (Augmented Reality, AR) - це технологія, яка поєднує віртуальну інформацію з реальним світом. Технічні засоби, які вона використовує, включають мультимедіа, 3D-моделювання, відстеження та реєстрацію в реальному часі, інтелектуальну взаємодію, датчики та багато іншого. Її принцип полягає в застосуванні комп'ютерно створеної віртуальної інформації, такої як текст, зображення, 3D-моделі, музика, відео і т. д., до реального світу після симуляції. Таким чином, обидва типи інформації доповнюють один одного, досягаючи покращення реального світу [2].

Доповнена реальність є часто обговорюваною темою, коли йдеться про технології, які пропонує Apple, та все більше популяризується останнім часом. Тому для цього вони розробили нативний фреймворк для роботи з доповненою реальністю, що має назву ARKit. Він поєднує відстеження руху пристрою, захоплення сцени з камери, обробку сцени та зручний дисплей, щоб спростити завдання створення AR-вражень. Використовуючи передню або задню камеру пристрою iOS, можна створювати багато різних AR-вражень за допомогою цих технологій [3].

Для відображення доповненої реальності використовується клас ARSCNView. Він надає найпростіший спосіб створення досвіду доповненої реальності, який поєднує віртуальний 3D-вміст із зображенням камери пристрою реального світу. Коли запускається сесія доповненої реальності:

- в’ю автоматично відображає відеопотік наживо з камери пристрою як фон сцени
- система координат сцени безпосередньо відповідає світовій системі координат AR, яка встановлюється конфігурацією сесії
- в’ю автоматично переміщує камеру SceneKit відповідно до руху пристрою в реальному світі

ARKit автоматично вирівнює простір координат з реальним світом, тому після розміщення віртуального контенту вашого застосунку, він зберігає ілюзію розташування у реальному світі, коли користувач переміщує пристрій [4].

1.3. Аналіз схожих програмних продуктів

Серед застосунків, що використовують доповнену реальність та машинне навчання для визначення справжніх розмірів об’єктів є “AirMeasure - AR Tape & Ruler“ [5]. Він створений у 2017 році та мав останнє оновлення у січні 2021 року. Тому, скоріш за все, він не використовує найновіші версії ARKit(останньою доступною на цей час є версія 6, а у 2020 році виходила версія 4) та CoreML і працює гірше, ніж міг би з оновленими версіями фреймворків.

Цей застосунок містить не лише виміри розпізнаних об’єктів, а й інший функціонал такий, як кисті для малювання в повітрі або на поверхнях, каталог віртуальних 3D-об’єктів, рулетка, створення плану кімнати, вимірювання росту людини та багато інших. Здається, в ньому використані всі або майже всі можливості доповненої реальності, які пропонує Apple.

Протестувати цей застосунок не вдалося через платну версію, проте розробник стверджує, що він здійснює найбільш точні виміри, і відео в AppStore переконують користувача в цьому. З іншого боку, на нього є деякі негативні відгуки, що очікувано, оскільки робота з машинним навчанням та доповненою реальністю може бути недосконалою та залежати не лише від розробників, але й користувачів, їхніх пристроїв та умов, в яких вони використовуються.

Іншим прикладом може бути системний iOS-застосунок “Мірило”, проте він дозволяє лише міряти відстані між розставленими точками. З власного досвіду його використання, він може робити значні похибки, якщо

його використовувати у несприятливих умовах, наприклад при поганому освітленні або роблячи надто різкі рухи, але за правильного використання від робить достатньо точні виміри:

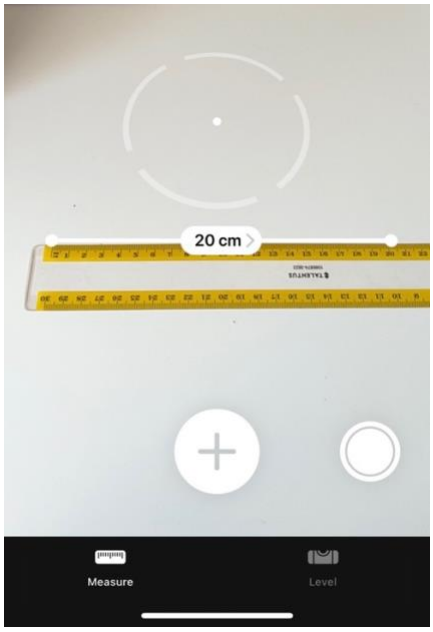


Рисунок 1.3.1 Демонстрація роботи застосунку "Мірило"

Підсумовуючи, схожих за призначенням застосунків не було знайдено, їх неможливо протестувати або вони реалізовують лише частину потрібного функціоналу.

2. Теоретичні засади для реалізації програмного продукту

2.1. Вибір датасету для розпізнавання об’єктів та тренування моделі

Для створення будь-якої моделі машинного навчання, треба правильно вибрати дані для тренування та тестування, щоб отримувати кращі результати роботи. Дані для тестування мають бути точні та різноманітні.

Оскільки створення датасету і тренування моделі не є безпосередньо задачею цієї роботи, а лише однією з її частин, було вирішено взяти готовий датасет та натренувати модель на ньому. Наразі не існує багато наборів даних, які спеціалізуються на розпізнаванні саме окремих продуктів. Вони можуть розпізнавати вже готові страви або продукти харчування можуть бути розкидані серед десятків чи сотень інших класів для розпізнавання.

Одним з найбільш підходящих датасетів є Fruit Images for Object Detection [6]. Він досить невеликий, містить всього 3 класи фруктів(яблука, апельсини та банани), але цього цілком достатньо для створення прототипу застосунку та перевірки його принципу роботи. Як показує практика, цей датасет дає досить точні результати навіть при розпізнаванні кількох різних фруктів одночасно, приклад можна побачити на зображенні нижче:

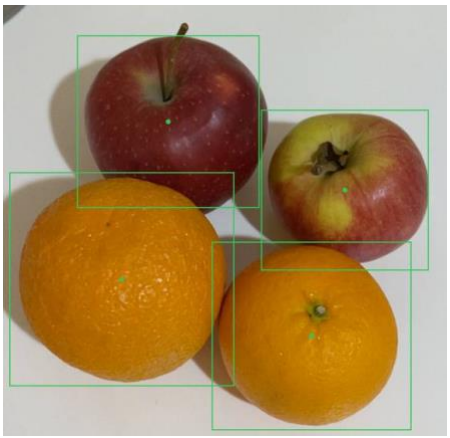


Рисунок 2.1.1 Приклад розпізнавання декількох об'єктів

Для тренування моделі, яка буде потім використовуватися в iOS-застосунку, можна вибрати один із способів - це використання програми

CreateML або створити її вручну за допомогою коду. CreateML надає зручний інтерфейс та можливість створити модель без докладання особливих зусиль. Вона дозволяє встановити деякі налаштування для тренування моделі, наприклад вибір алгоритму, параметрів обробки даних(розмір файлів, обертання чи зміщення даних), гіперпараметрів, кількості ітерацій тренування тощо. Проте якщо потрібна більша гнучкість та контроль над усіма процесами тренування, варто обрати тренування моделі через код, який дає більше можливостей для налаштувань.

У межах цієї роботи достатньо натренувати модель за допомогою CreateML, задаючи такі параметри:

- Алгоритм Full Network, що тренує мережу на основі архітектури YOLOv2
- 6000 ітерацій

На виході натреновану модель можна експортувати у форматі .mlmodel, і тоді її можна використовувати в iOS-застосунку.

2.2. Вимірювання відстані до об’єкта

Задача вимірювання відстані від камери до об’єкта є надважливою для вибраного застосунку, оскільки від відстані залежать всі подальші обрахунки. Маючи інформацію про те, як далеко знаходиться об’єкт та його обмежувальну рамку на зображенні, можна визначити його реальні розміри. Похибка у вимірюванні відстані може привезти до неправильних результатів потім, тому важливо виміряти відстань якомога точніше.

Визначення відстані до об’єкта можна здійснити за допомогою фреймворку ARKit для роботи з доповненою реальністю. Для цього можна використати метод рейкастингу, або кидання променю. Такий спосіб дозволяє перетворити двовимірну координату на екрані пристрою у тривимірний вектор, що має інформацію про розташування об’єкта в реальному світі.

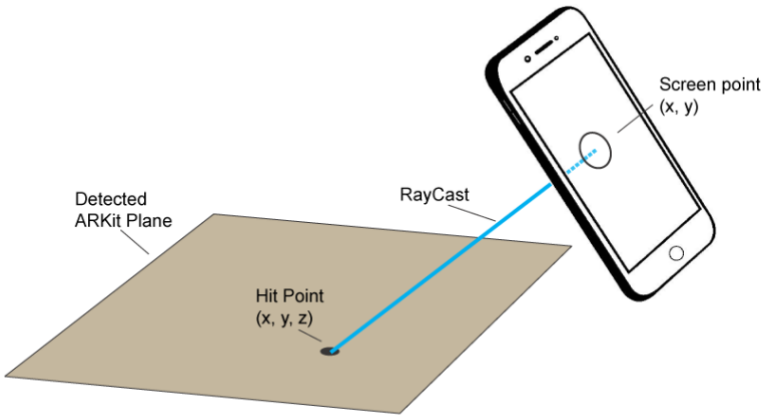


Рисунок 2.2.1 Принцип роботи рейкастингу [7]

При запуску сцени доповненої реальності, вона починає відстежувати орієнтацію камери в просторі, а також розпізнавати поверхні. Вона звертає увагу на різні кути, краї поверхонь та їхні текстури. Так можуть визначатися стіни, підлога, стіл, меблі та інші об’єкти реального світу. Цей підхід дозволяє змоделювати віртуальну 3D-модель реального світу, яка потім може використовуватися, наприклад, для розміщення в ній об’єктів доповненої реальності.

Повертаючись до рейкастингу, коли пускається промінь з камери, він має перетнути якусь розпізнану поверхню і результат рейкастингу буде містити інформацію про цю точку перетину. Може виникнути ситуація, при якій результат рейкастингу відсутній. Найчастіше це трапляється тоді, коли спробувати опустити промінь відразу після запуску сцени доповненої реальності, бо на дуже ранньому етапі застосунок не має достатньо інформації про навколишній світ. Проблема також може виникнути, коли положення камери не мінялося і застосунку важко оцінити глибину, на якій знаходяться об’єкти та їх розташування одне відносно одного.

Отож задача полягає в тому, щоб використати рейкастинг та знайти відстань до центру обмежувальної рамки розпізнаного об’єкта.

ARRaycastResult, який ми отримуємо внаслідок успішного пускання променю, має властивість worldTransform, яка є матрицею трансформації

4x4 та містить інформацію про точку перетину включно з її ротацією, координатами тощо. З відповідних елементів матриці можна отримати координати x, y, z точки перетину. Крім того, у камері є така сама матриця, тому ми можемо дістати і її координати. Маючи 2 точки у просторі, можна скористатися методом public func simd_distance(_ __x: simd_float3, _ __y: simd_float3) -> Float, який приймає на вхід 3-вимірні вектори, та отримати відстань між ними. Відстань повертається в метрах, тому для зручності можна перевести в сантиметри.

Підсумовуючи, відстань до об’єкта можна виміряти, пустивши промінь в центр об’єкта за порахувавши відстань між камерою та точкою перетину променю з поверхнею об’єкта, в цьому випадку фрукта.

2.3. Знаходження поля зору камери

Поле зору камери(field of view, FOV) - це максимальна площа зразка, яку може відобразити камера. Поле зору залежить від фокусної відстані об’єктива та розміру сенсора. Якщо розглянути приклад, коли фокусна відстань однакова, то чим більший розмір сенсора, тим більший кут поля зору [8].

При використанні фреймворку ARKit, використовується камера класу ARCamera для роботи в контексті доповненої реальності. Вона використовує справжню камеру девайсу для отримання зображення з неї, а потім використовує його для відтворення сцени, яку користувач бачить на екрані. ARCamera містить дані про її положення, орієнтацію, внутрішні параметри реальної камери пристрою, а також позиціонування об’єктів.

Доступ до цієї камери можна отримати через сесію сцени та поточний кадр:

```
let camera = sceneView.session.currentFrame?.camera
```

У неї є властивість intrinsics, що містить інформацію про оптичні параметри камери пристрою, наприклад фокусну відстань, яка потрібна для

знаходження поля зору камери. Фокусна відстань в цьому випадку вимірюється у пікселях.

Тоді горизонтальний кут поля зору камери визначається за допомогою формули:

let xFovDegrees = atan(imageResolution.width/(2 * focalLength)) * 180/Float.pi

А вертикальний кут можна знайти за допомогою горизонтального кута, помноженого на співвідношення висоти до ширини сцени:

let yFovDegrees = xFovDegrees * viewSize.height / viewSize.width

Таким чином, використавши параметри камери та розширення зображення, можна знайти кут поля зору камери, який знадобиться в подальшому.

2.4. Розрахунок розміру та маси об’єкта

Для початку варто зазначити, що в межах цієї роботи розглядатимуться предмети, що мають форму кулі або максимально наближену до неї. Зважаючи на вибраний датасет, можна порахувати об’єм яблук та апельсинів. Тоді задача знаходження розміру зводиться до знаходження радіусу кулі, через який потім можна знайти об’єм.

Щоб правильно визначити радіус, треба створити схематичний малюнок та розв’язати математичну задачу. За допомогою багатьох спроб в процесі створення малюнку вдалося зобразити найбільш точний варіант, що підходить для будь-якого випадку незалежно від розташування об’єкта на зображенні: як і в його центрі, так зміщено по осі X чи У.

Справа в тому, що промінь, що опускається з камери на поверхню об’єкта, опускається в координату в центрі обмежувальної рамки, тому він пройде через центр об’єкта. Крім того, за властивістю дотичної площини, вона перпендикулярна радіусу, що проведений до точки дотику. Тому з якого ракурсу ми б не дивилися на об’єкт, неважливо, де він знаходиться на

зображенні, малюнок виглядатиме так само та радіус можна буде порахувати за допомогою єдиної формули. Задача вирішується за допомогою прямокутних трикутників і радіус можна виразити через відстань та кут поля зору камери.

На основі цього малюнок до цієї задачі буде виглядати наступним чином:

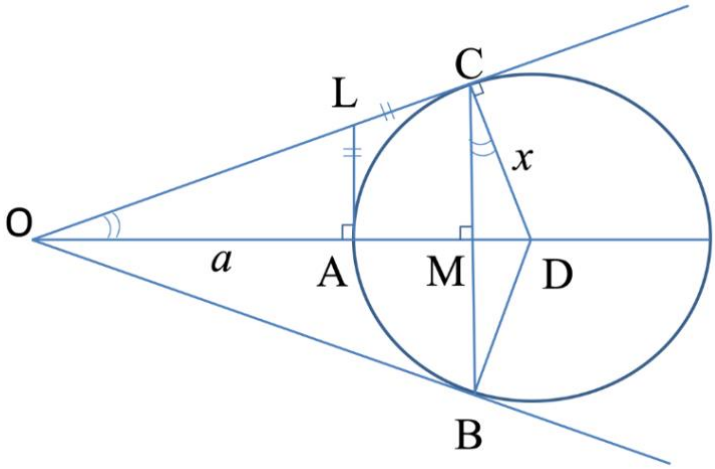


Рисунок 2.4.1 Схема до задачі знаходження радіусу об'єкта

Розв'яжемо задачу:

Дано:

OA = a – відстань від камери до об'єкта в см

CB – розмір об'єкта на зображенні в пікселях

maxY – висота зображення в пікселях

α – кут вертикального поля зору камери в градусах

Знайти:

x=*CD* – радіус об'єкта

Розв'язання

Кут, під яким видно об'єкт = $\frac{CB}{maxY}\alpha$

Тоді половина цього кута(позначена на зображенні):

$$\beta = \frac{1}{2} \cdot \frac{CB}{maxY}\alpha$$ - в градусах

Тангенс кута, який буде використовуватися для обчислень:

$$tg\beta = tg(\beta \cdot \frac{\pi}{180})$$

З ΔOAL: $tg\beta = \frac{LA}{OA}$

$LA = OA \cdot tg\beta$ – в см

Дотичні, проведені до кола з однієї точки – рівні. Тому:

$$LC = LA = a \cdot tg\beta$$

За теоремою Піфагора:

$$OL^2 = OA^2 + LA^2$$

$$OL^2 = a^2 + (a \cdot tg\beta)^2 = a^2(1 + tg^2\beta)$$

$$OL = a\sqrt{1 + tg^2\beta}$$

$$OC = OL + LC = a\sqrt{1 + tg^2\beta} + a \cdot tg\beta$$

З ΔOCD: $tg\beta = \frac{CD}{OC} = \frac{x}{OC}$

$x = OC \cdot tg\beta$

$x = a(\sqrt{1 + tg^2\beta} + tg\beta) \cdot tg\beta$ – шуканий радіус

Таким чином, можна знайти радіус об’єкта за допомогою відстані до нього та кута поля зору камери. Тепер можна вирахувати масу об’єкта.

Розрахунок маси об’єкта залежить від таких факторів, як його форма, розмір та густина. Форма була зазначена раніше та радіус вже знайдено. Густина для фруктів може залежати від різних чинників(наприклад, доспілість та сорт плоду). Оскільки такі деталі нам невідомі, треба використовувати середнє значення. Наприклад, відповідно до дослідження [9] середньостатистична густина яблука дорівнює 0.808 г/см3, а апельсина - 0.814 г/см3, які і будуть використані потім під час подальших розрахунків та розробки застосунку. Тоді маса визначається за допомогою формули $m = \rho \cdot V$, де ρ - густина, V – об’єм кулі, що дорівнює $V = \frac{4}{3}\pi r^3$.

2.5. Підбір рекомендацій за розпізнаними об’єктами

Спочатку було вирішено, що рекомендації у вигляді рецептів будуть отримуватися за допомогою запиту на сервер. Це найбільш поширений підхід в роботі мобільних застосунків, коли дані, особливо якщо вони великі, зберігаються на сервері, за логіку їхнього зберігання та модифікації відповідає бекенд, а мобільний застосунок може робити запити, щоб отримати чи змінити дані.

Враховуючи специфіку вибраного застосунку, потрібно, щоб дані мали певну структуру, з якою можна буде працювати. Як приклад можна навести наявність достатньої кількості рецептів з яблуками та апельсинами, які розпізнає застосунок. Іншим прикладом є відповідні одиниці вимірювання, в ідеальному випадку кількості продуктів мають бути вказані в грамах, щоб зручніше проводити обрахунки та не додавати зайвих операцій переведення одних одиниць в інші. Також важливо, щоб назви продуктів у відповіді сервера збігалися з назвами, які використовуються в програмі, до прикладу “apple”, а не “apples”. Інакше програма не буде давати очікуваних результатів або потрібно буде додавати логіку для обробки таких виняткових випадків, що не є дуже правильно.

Якби програмний продукт, що розробляється, був комерційним, звичайно він мав би власний бекенд з відповідними кінцевими точками(endpoints), однорідні дані та вище описаних проблем не виникало би. Проте в цій ситуації було складно знайти відкриті безкоштовні API, які би підходили для пошуку рекомендацій рецептів, через вже згадані проблеми.

Альтернативним рішенням є зберігання рецептів в базі даних, популярною є Realtime Database, яку надає сервіс Firebase, і вона була обрана для цієї роботи за рахунок зручності використання. Це NoSQL база даних, що зберігає дані у вигляді одного великого JSON-дерева, де їх можна зручно упорядковувати. Вона працює в режимі реального часу, тому вона

може часто використовуватися для месенджерів. Крім того, вона інтегрується з іншими сервісами Firebase, тому в подальшому функціонал застосунку може розширюватися за рахунок такої можливості.

Для реалізації підбору рекомендацій, треба завантажити рецепти, фільтруючи їх за списком інгредієнтів так, щоб рецепти містили всі наявні у користувача інгредієнти. Щоб не робити запит до бази даних щоразу, запит робиться тільки раз при першій рекомендації, потім рецепти зберігаються локально в межах сесії застосунку і завжди є у швидкому доступі. Це рішення ефективне для невеликого застосунку та обмеженої кількості даних, але в перспективі не варто тримати надто багато даних в сесії.

2.6. Перерахунок інгредієнтів

При перерахунку інгредієнтів є 2 основні фактори, на які треба звернути увагу:

- перерахунок відповідно до кількості наявних продуктів у користувача
- збереження пропорцій оригінального рецепту

Складність перерахунку залежить від кількості різних розпізнаних інгредієнтів. Якщо розпізнається лише 1 продукт, то задача перерахунку зводиться до звичайного множення кількості кожного інгредієнту на коефіцієнт k, де k = detectedMass / recipeMass.

Інший випадок - це коли розпізнаються 2 однакові продукти. Тоді їхні маси треба додати, вважати це одним продуктом та виконати ті самі обрахунки.

Якщо розпізнаються 2 і більше різних інгредієнтів, то підхід має бути наступним:

- дізнатися співвідношення кількості наявних продуктів до кількостей тих самих продуктів у рецепті

- перерахувати рецепт за найменшим співвідношенням

Приклад для перерахунку при розпізнаванні двох інгредієнтів:

Нехай рецепт має продукту А - 200 г, продукту В - 400 г, а користувач має продукту А - 300 г, продукту В - 300 г. Маємо такі пропорції: для продукту А - 1.5, а для В - 0.75, саме тому перерахунок має відбуватися за продуктом В. Всі продукти в оригінальному рецепті множаться на 0.75, в результаті отримуємо рецепт з 150 г продукту А і 300 г продукту В.

3. Опис реалізації програмного продукту

RecipeDetect - це iOS-застосунок, що підбирає рецепти на основі розпізнавання інгредієнтів з використанням технологій машинного навчання та доповненої реальності. Вибрано саме таку назву, бо з неї зрозуміло, що саме застосунок робить. Для нього було створено таку іконку:

Рисунок 3.1 Іконка застосунку RecipeDetect

3.1. Переваги використання мобільного пристрою для створення застосунку, який визначає справжні розміри об’єктів

Для створення програмного продукту, метою якого є не лише розпізнавання об’єктів, а й обрахунок їхніх розмірів, найкраще підходить саме мобільний застосунок.

Проблема розпізнавання об’єктів не є новою і зараз існує безліч наборів даних, аби натренувати власну модель, або можна знайти й готові натреновані моделі у різних форматах, які можна використовувати у різних програмах. Для цього не потрібно мати особливого апаратного забезпечення, варто лише вміти працювати з моделлю, а саме дати їй на вхід зображення та обробити результат розпізнавання. Проте, оскільки у цій роботі розглядаються теж розміри об’єктів, це можна зробити лише через обрахунок глибини, на якій знаходяться об’єкти, що можна зробити тільки за наявності камери.

Крім того, деякі моделі телефонів мають вбудований LiDAR(Light Detection And Ranging) сканер, використовуючи можливості якого можна визначати відстань, на якій знаходяться об’єкти, з більш високою точністю. LiDAR використовує техніку виявлення світла для обчислення глибини. Він

працює, випромінюючи імпульси інфрачервоного світла та вимірюючи, скільки часу потрібно, щоб вони повернулися після удару об предмети поблизу. Час між вихідним лазерним імпульсом і відбитим імпульсом дозволяє датчику LiDAR обчислювати відстань до кожного об’єкта на основі швидкості світла [10].

Окрім наявності потрібного апаратного забезпечення, мобільний застосунок - це про швидкість, зручність, простоту використання та доступність.

3.2. Вимоги до рішення та постановка ТЗ

Вимоги до рішення(solution requirements) - це набір вимог та характеристик, якими повинен володіти продукт, щоб відповідати потребам стейкхолдерів(зацікавлених сторін) та бізнесу. Вони діляться на 2 групи: функціональні та нефункціональні.

Функціональні вимоги визначають поведінкові якості продукту: що він робить, які має функції та властивості.

На основі аналізу технологій, предметної області та проблем користувачів визначено такі функціональні вимоги:

- Застосунок має мати вступний екран з інформацією, як ним користуватися, аби отримати найкращі результати
- Користувач може розпізнати набір продуктів з камери та відразу побачити відстань до них у вигляді 3D-тексту
- Користувач може очистити екран та спробувати виміряти відстань наново
- В разі помилки користувач має побачити відповідне повідомлення з підказкою, що варто зробити, аби уникнути помилки
- На кожному етапі має бути можливість повернутися на попередній екран або перейти відразу до камери

- Користувач може побачити список рецептів за розпізнаними інгредієнтами та відкрити деталі рецепту

Нефункціональні вимоги описують робочі якості продукту, тобто як продукт працює. До них належать ефективність роботи, зручність використання, надійність тощо.

Щоб додаток міг не лише бути функціональним, але й відповідної якості, визначено наступні нефункціональні вимоги:

- Застосунок ефективно виконує пошук та перерахунок рецептів
- Застосунок надає простий та інтуїтивний інтерфейс, що відповідає Human Interface Guidelines

На основі вимог, постановка технічного завдання наступна: створити мобільний iOS-застосунок для рекомендації рецептів, використовуючи технології, алгоритми та виведені формули під час дослідження. Застосунок має відповідати зазначеним функціональним та нефункціональним вимогам.

Кожен етап роботи має відображатися користувачу, а саме користувач має змогу побачити:

- відстань до об’єктів на екрані з камерою
- обмежувальні рамки об’єктів
- список розпізнаних об’єктів з усіма властивостями такими, як назва, розмір та маса

Під час розробки мають бути використані фреймворки Vision та ARKit.

3.3. Використані засоби розробки

Під час розробки RecipeDetect були використані різні засоби для розробки застосунку такого типу.

Roboflow [11] - це платформа з більш ніж 250 тисячами користувачів, що дозволяє створювати датасети, тренувати моделі та розгортати їх у продакшн. ЇЇ головною перевагою є те, що вона дозволяє працювати з різними форматами даних. Наприклад, вибраний для цієї роботи датасет [6] не повністю підходив для тренування моделі у CreateML через те, що анотації мають іншу структуру, ніж та, що використовується у CreateML. А за допомогою Roboflow датасет був конвертований у правильний формат, який можна було використати для подальшої роботи. Крім того, на цій платформі зручний інтерфейс користувача, з яким легко працювати з датасетами. Ось, як він виглядає на прикладі цієї роботи:

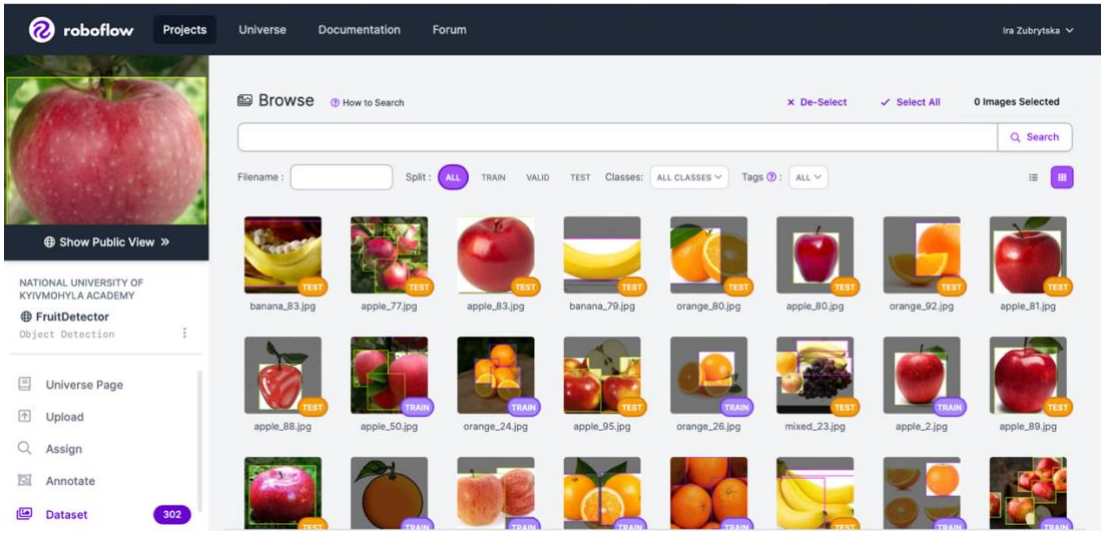


Рисунок 3.3.1 Інтерфейс платформи Roboflow

Як вже було згадано у попередніх розділах, для тренування і створенні моделі розпізнавання об’єктів, було використано програму CreateML.

Xcode - стандартне середовище розробки від Apple, яке використовується для розробки програм для iOS, macOS, watchOS та tvOS. Також була використана мова програмування Swift.

UIKit та SwiftUI - фреймворки для розробки iOS-застосунків, зокрема побудови користувацького інтерфейсу. На відміну від UIKit, SwiftUI є новішим та декларативним, а отже дозволяє розробляти застосунки швидше. Він був використаний для більшості екранів. На UIKit написаний

ARSceneViewController, а потім інтегрований у інші компоненти застосунку.

Vision - фреймворк, що дозволяє використовувати власну модель CoreML для таких завдань, як класифікація зображень чи виявлення об’єктів [12]. Саме за допомогою нього відбувається взаємодія зі створеною моделлю.

ARKit - фреймворк, що дозволяє використовувати доповнену реальність в iOS-застосунках. Зокрема у цьому застосунку він був використаний для знаходження відстані, на якій знаходяться об’єкти від камери, що дозволило оцінити реальні розміри об’єктів.

Firebase - платформа, розроблена Google, яка надає широкий спектр сервісів від баз даних до автентифікації, аналітики, поширення застосунків та інших. У RecipeDetect була використана Realtime Database для зберігання та пошуку рецептів.

3.4. Структура застосунку та його основні компоненти

Важливою частиною розробки застосунку є вміння правильно його спроектувати. Зазвичай структура базується на певній архітектурній моделі, яка розділяє функціонал застосунку на компоненти та визначає їх взаємодію.

Одним з найпопулярніших архітектурних патернів є MVVM(Model - View - ViewModel), який було використано при розробці RecipeDetect. В цьому патерні Model відповідає за дані застосунку, зазвичай це структури. View - це компонент, який відповідає за відображення даних, тобто візуальну частину. ViewModel містить логіку зв’язування даних і є комунікатором між даними та інтерфейсом. Коли користувач взаємодіє з елементами застосунку, результат взаємодії обробляється в’ю моделлю, в’ю модель може змінити дані, якщо це потрібно, а потім вона надсилає оновлені дані назад, що відобразиться в інтерфейсі.

Також під час реалізації було зроблено декомпозицію задач і створено окремі компоненти для виконання того чи іншого завдання. Далі будуть описані основні компоненти програми.

Detection - модель, що містить інформацію про розпізнаний об’єкт, а саме його назву, обмежувальну рамку та відсоток впевненості.

DistanceDetection - модель, що містить повну інформацію про розпізнаний об’єкт, додаючи до попередніх властивостей ще відстань до об’єкта, його радіус та масу.

Recipe - модель, що містить інформацію про рецепт.

ObjectDetector - клас, який працює з моделлю машинного навчання. Він робить запит до моделі, обробляє результат та повертає дані у вигляді Result<[Detection], Error>.

ARSceneViewController - один з найважливіших в’ю контролерів, який відповідає за відображення сцени доповненої реальності та в якому відбувається процес розпізнавання об’єктів та вимірювання їхніх розмірів.

ARSceneViewControllerWrapper - обгортка для ARSceneViewController, що відповідає протоколу UIViewControllerRepresentable для того, щоб цей в’ю контролер можна було використовувати у SwiftUI елементах.

ARSceneViewControllerViewModel - в’ю модель для ARSceneViewController, яка містить у собі бізнес логіку обробки розпізнаних об’єктів. У неї є стани такі як initial(початковий), data(дані) та error(помилка), відповідно до яких на контролері можна буде побачити або успішно розпізнані об’єкти, або відповідне повідомлення про помилку.

ARCameraView - SwiftUI в'ю, що містить ARSceneViewControllerWrapper і, крім того, містить додаткові елементи такі як кнопки та повідомлення про помилки.

ImageCaptureView - SwiftUI в'ю, що відображає захоплене зображення з обмежувальними рамками об'єктів та їхніми центрами, куди проводились промені.

DetectionView - SwiftUI в'ю, що відображає список розпізнаних об'єктів з усіма результатами обрахунків та *деталями* про них.

RecipeListView, RecipelistItemView, RecipeDetailsView - SwiftUI в'ю, що використовуються для відображення списку, елементу списку та деталей рецепту відповідно.

SessionDataStore - сховище для зберігання даних в межах сесії. У цьому випадку воно використовується для кешування рецептів.

FirebaseService - сервіс для роботи з Firebase Realtime Database, за допомогою якого можна звернутися до бази даних та дістати з неї рецепти.

Calculator - відповідає за усі розрахунки в рецептах та продуктах таких, як визначення маси продукту та перерахунок рецептів.

3.5. Принцип роботи застосунку

При першому запуску застосунку користувач бачить вступний екран з двома сторінками. Вони містять інформацію про те, як правильно користуватися застосунком, аби отримати кращі результати. Перша сторінка інформує, що перед початком розпізнавання варто рухати телефон навколо об'єктів, аби програма змогла дослідити середовище. Друга сторінка містить інформацію про те, що варто тримати продукти в межах рамки, де розпізнавання працює точніше (додаток А).

Також при запуску застосунку треба буде надати доступ до камери, щоб мати змогу нею користуватися.

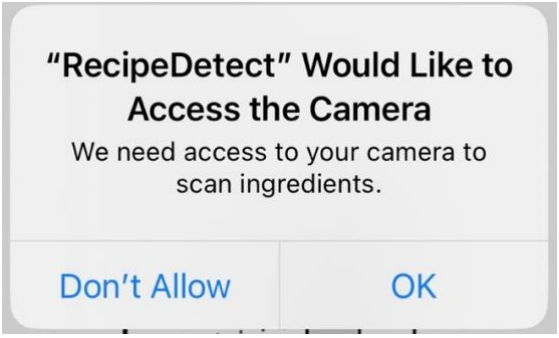


Рисунок 3.5.1 Алерт з проханням дати доступ до камери

Після вступного екрану користувачу доступний екран з відображенням вмісту з камери. Відображення камери та можливості доповненої реальності доступні за допомогою використання ARSCNView. Внизу є кнопки “Clear” для очищення екрану та кнопка для захоплення зображення (додаток Б).

Під час захоплення знімку:

- робиться снепшот сцени доповненої реальності
- в’ю модель робить запит до моделі машинного навчання з цим снепшотом, обробляє результат та повертає масив розпізнаних об’єктів
- в контролері в центр кожної обмежувальної рамки опускається промінь
- при успішному сценарії будуть обраховані радіуси та маси, інформація буде передана на наступні екрани
- під час помилки на екран буде виведене повідомлення

Якщо на знімку не було виявлено жодного об’єкта, користувач побачить повідомлення про помилку та прохання спробувати ще раз:

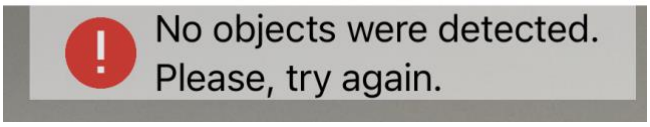


Рисунок 3.5.2 Повідомлення, що не вдалося розпізнати об’єкти

Якщо користувач проігнорував прохання рухати камеру навколо об’єктів або його рухів було недостатньо, аби вивчити середовище, він побачить помилку про те, що неможливо виміряти відстань до об’єкта і треба спробувати ще раз. Це означає, що об’єкти були виявлені, але промінь не зміг опуститись на поверхню.

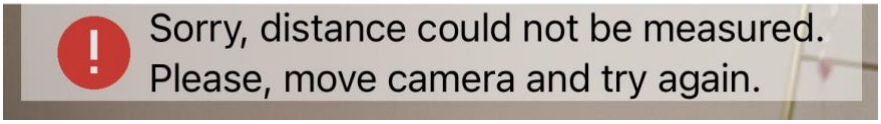


Рисунок 3.5.3 Повідомлення, що неможливо виміряти відстань

Якщо все було зроблено правильно, користувач зможе побачити відстань до об’єктів у вигляді 3D-тексту по центру кожного з них. У правому нижньому куті з’явиться кнопка зі стрілкою, по натиску на яку можна буде перейти на наступний екран (додаток В).

Екран “Bounding box” містить захоплене зображення разом з обмежувальними рамками об’єктів та їхніми центрами, куди проводились промені для виміру відстані (додаток Г).

Екран “Detection details” містить список виявлених об’єктів з повною інформацією про них:

1. Назва
2. Відсоток впевненості розпізнавання
3. Відстань в см
4. Радіус в см
5. Маса в грамах

Крім того, у навігаційній панелі є опція перейти назад до камери, якщо користувач хоче почати розпізнавання заново (додаток Ґ).

При переході на наступний екран, відбувається запит до бази даних з рецептами, які зберігаються у Firebase Realtime Database(якщо це перший перехід на цей екран). Після цього вони зберігаються локально в SessionDataStore в межах сесії і наступні рази є відразу доступними. Далі рецепти фільтруються за списком продуктів і перераховується кількість інгредієнтів відповідно до кількості виявлених продуктів зі збереженням пропорцій оригінального рецепту (додаток Д).

Список рецептів містить часткову інформацію про рецепт таку як назва, фото та час, потрібний для приготування. По натиску на елемент списку можна побачити його деталі такі як список інгредієнтів та інструкцію приготування (додаток Е).

Висновки

Результатом виконання цієї дипломної роботи є створення прототипу iOS-застосунку, який підбирає рецепти на основі розпізнавання продуктів з камери. Він може зекономити користувачу час на пошук рецептів, які будуть задовольняти його потреби, враховуючи різноманіття та кількість інгредієнтів, які є у користувача.

Під час написання було досліджено можливості поєднання машинного навчання для розпізнавання об’єктів та доповненої реальності для реалізації поставленого технічного завдання. На основі дослідження вдалося розпізнати об’єкти, визначити їхні реальні розміри та підібрати рекомендації на основі отриманих даних.

Отже, в результаті виконання роботи було повністю виконано вимоги та поставлені цілі.

Незважаючи на досить позитивні досягнені результати, завжди є місце для росту та розвитку. Наразі я бачу кілька можливостей для покращення створеного застосунку.

Перш за все, датасет, що використовувався для розробки застосунку, має лише 3 класи, які він розпізнає. Він дає досить точні результати, але їх також можна покращувати, оскільки обмежувальні рамки іноді виходили за межі об’єктів або навпаки перетинали їхні сторони. Тому першим покращенням може стати розробка власного датасету, який буде розпізнавати більшу кількість класів та давати точніші результати.

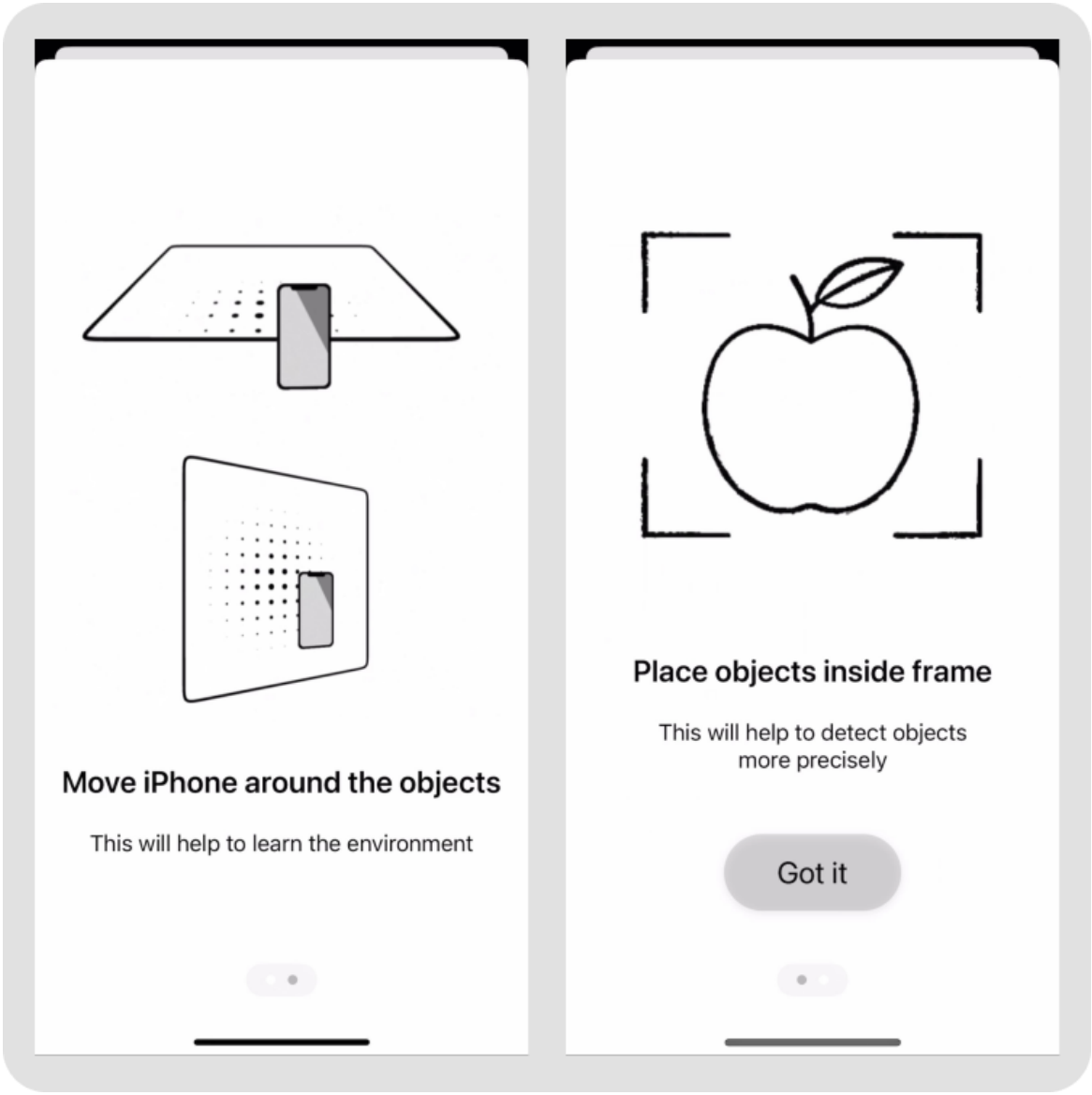
Також розроблений застосунок вміє визначати розміри об’єктів, які мають форму кулі. В подальшому його можна удосконалити та попрацювати над тим, щоб визначати розміри об’єктів інших форм.

Список джерел

1. Building an object detector data source [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/createml/building-an-object-detector-data-source.
2. An overview of augmented reality technology [Електронний ресурс] // IOP Publishing. – 2019. – Режим доступу до ресурсу: https://iopscience.iop.org/article/10.1088/1742-6596/1237/2/022082/pdf.
3. ARKit Integrate iOS device camera and motion features to produce augmented reality experiences in your app or game. [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/arkit.
4. ARSCNView A view that blends virtual 3D content from SceneKit into your augmented reality experience. [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/arkit/arscnview.
5. AirMeasure - AR Tape & Ruler [Електронний ресурс] – Режим доступу до ресурсу: https://apps.apple.com/us/app/airmeasure-ar/id1251282152.
6. Fruit Images for Object Detection [Електронний ресурс] – Режим доступу до ресурсу: https://www.kaggle.com/datasets/mbkinaci/fruit-images-for-object-detection.
7. Neil M. Design a Responsive 3D Cursor for ARKit Apps in Unity [Електронний ресурс] / Mathew Neil. – 2020. – Режим доступу до ресурсу: https://placenote.com/blog/real-world-cursor-with-arkit-hittest/.
8. Field of View and Angular Field of View [Електронний ресурс] – Режим доступу до ресурсу: https://www.princetoninstruments.com/learn/camera-fundamentals/field-of-view-and-angular-field-of-view.

9. Estimation of Volume and Mass of Axi-Symmetric Fruits Using Image Processing Technique [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: https://www.tandfonline.com/doi/pdf/10.1080/10942912.2013.831444.
10. Bragg L. What is Lidar Scanning and How Does Apple Use It? [Електронний ресурс] / Lucy Bragg. – 2022. – Режим доступу до ресурсу: https://www.mappedin.com/blog/product/indoor-mapping/what-is-lidar-scanning/.
11. Roboflow [Електронний ресурс] – Режим доступу до ресурсу: https://roboflow.com.
12. Vision Apply computer vision algorithms to perform a variety of tasks on input images and video. [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/vision.

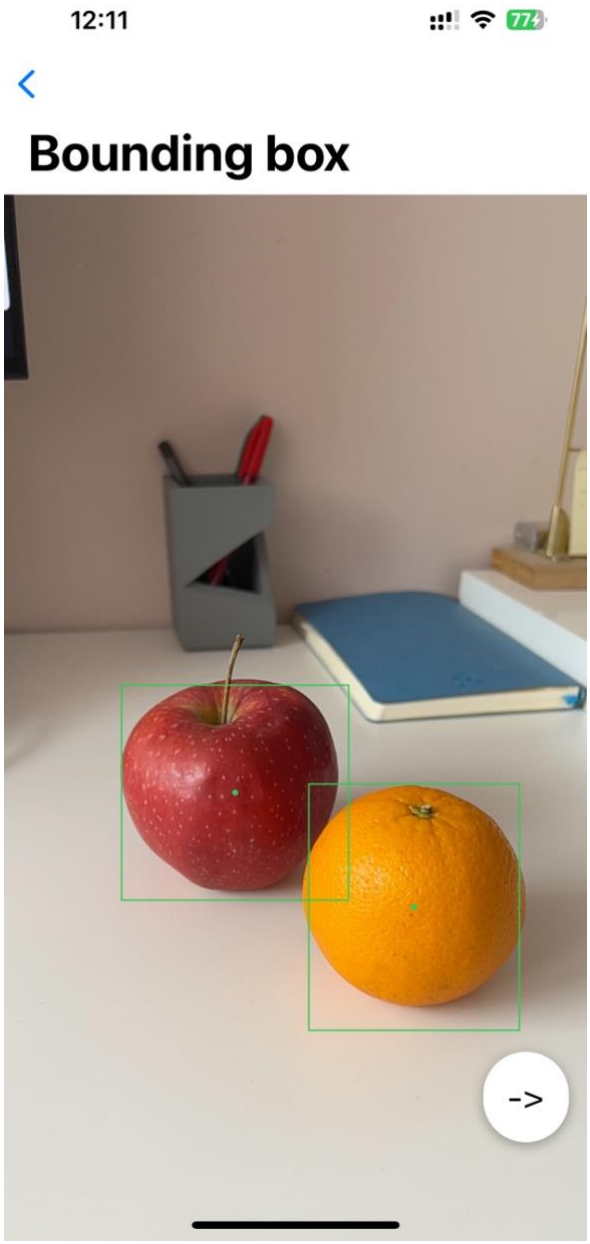
Додаток А. Вступний екран застосунку RecipeDetect



Додаток Б. Демонстрація екрану з камерою

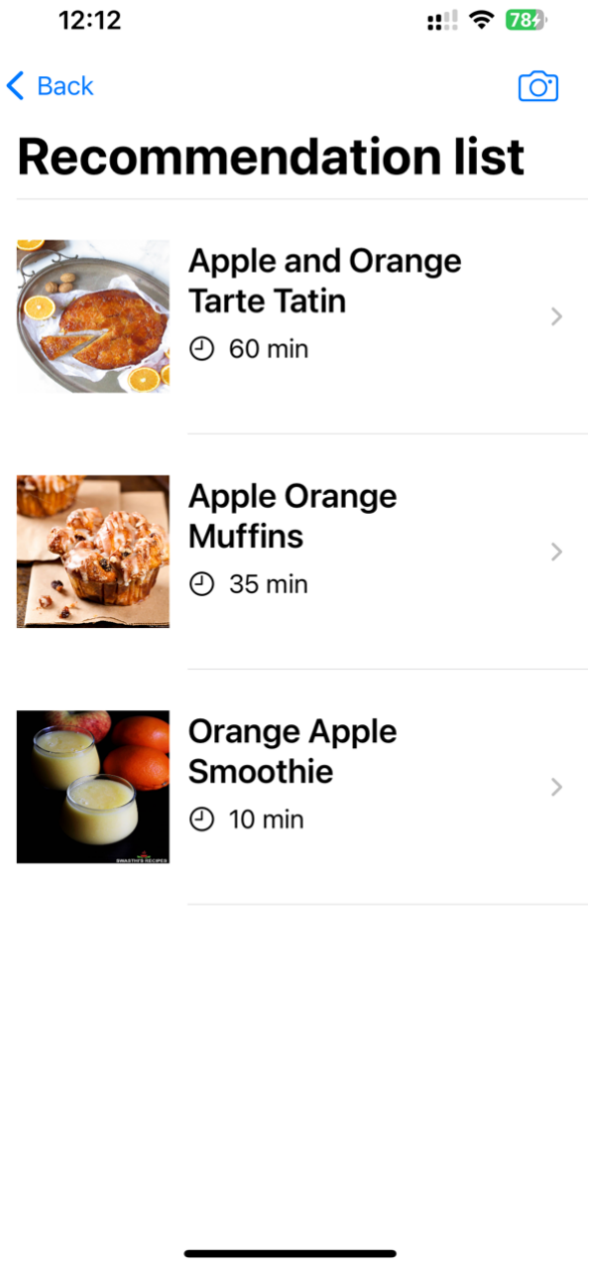
Додаток В. Демонстрація успішно виміряної відстані

Додаток Г. Екран, що зображує обмежувальні рамки розпізнаних об’єктів

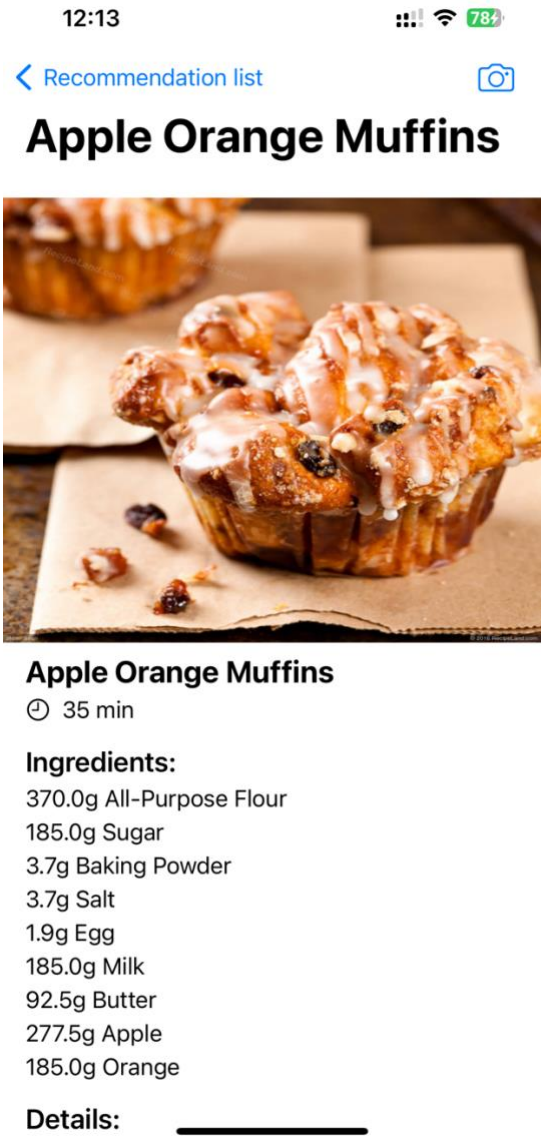


Додаток Ґ. Екран, що містить повну інформацію про розпізнані об’єкти

Додаток Д. Екран зі списком підібраних рецептів



Додаток Е. Екран з деталями рецепту



Додаток Є. Лістинг класу ObjectDetector

```
import UIKit
import Vision

/// Class that is responsible for making request to ML model, processing the results and returning object detections
final class ObjectDetector {

    typealias DetectionResult = Result<[Detection], Error>
    private var completion: ((DetectionResult) -> Void)?

    private lazy var model: VNCoreMLModel = {
        let config = MLModelConfiguration()
        let baseModel = try! FruitDetector(configuration: config)
        return try! VNCoreMLModel(for: baseModel.model)
    }()

    private lazy var detectorRequest: VNImageBasedRequest = {
        let request = VNCoreMLRequest(model: model, completionHandler: requestCompletionHandler(_:error:))
        request.imageCropAndScaleOption = .centerCrop
        return request
    }()

    func detect(image: UIImage, completion: @escaping ((DetectionResult) -> Void)) {
        self.completion = completion

        let handler = VNImageRequestHandler(
            cgImage: image.cgImage!,
            orientation: CGImagePropertyOrientation(image.imageOrientation)
        )

        do {
            try handler.perform([detectorRequest])
        } catch {
            print(error)
        }
    }
```

```
        private func requestCompletionHandler(_ request: VNRequest, error:
Error?) {
            guard let observations = request.results as?
[VNRecognizedObjectObservation] else {
                return
            }
            completion?(.success(observations.sorted(by: { $0.confidence >
$1.confidence }).compactMap { obs in
                if let label = obs.labels.first?.identifier {
                    // model returns upside down results where origin is left bottom
corner
                    // so it's necessary to flip Y coordinate and height
                    let flippedBox: CGRect = .init(
                        x: obs.boundingBox.origin.x,
                        y: 1 - obs.boundingBox.origin.y,
                        width: obs.boundingBox.width,
                        height: -obs.boundingBox.height)
                    return Detection(label: label, boundingBox: flippedBox,
confidence: obs.confidence)
                }
                else {
                    return nil
                }
            }))
        }
    }
```

Додаток Ж. Лістинг деяких методів класу ARSceneViewController

```
import UIKit
import Anchorage
import ARKit

final class ARSceneViewController: UIViewController, ARSCNViewDelegate
{
   private let sceneView: ARSCNView = ARSCNView()
   let viewModel: ARSceneViewControllerViewModel
   private var lidarSupported = false

   // MARK: Lifecycle
   override func viewWillAppear(_ animated: Bool) {
      super.viewWillAppear(animated)
      let configuration = ARWorldTrackingConfiguration()
      configuration.planeDetection = [.horizontal, .vertical]
      if lidarSupported {
         configuration.sceneReconstruction = .mesh
      }
      sceneView.session.run(configuration)
   }

   override func viewDidLoad() {
      super.viewDidLoad()

      if #available(iOS 13.4, *) {
         lidarSupported =
ARWorldTrackingConfiguration.supportsSceneReconstruction(.mesh)
      }
   }

   @objc private func capture() {
      let snapshot = sceneView.snapshot()
      viewModel.detect(snapshot: snapshot)
   }

   private func handleDetections(_ detections: [Detection]) {
      let distanceDetections: [DistanceDetection] = detections.compactMap {
detection in
         if let distance = findDistance(to: detection),
           let radius = findRadius(distance, detection.boundingBox)
```

```
            {
                return .init(
                    label: detection.label,
                    boundingBox: detection.boundingBox,
                    confidence: detection.confidence,
                    distance: distance,
                    radius: radius,
                    mass: Calculator.getMass(for: detection.label, radius: radius) ?? 0)
            }
            return nil
        }
        viewModel.handleDistanceDetections(distanceDetections, size:
sceneView.bounds.size)
    }

    // MARK: Calculations

    /// Finds radius of detected object using camera's vertical field of view.
    /// - Parameters:
    ///    - distance: `Float` distance to object
    ///    - box: `CGRect` object's bounding box
    /// - Returns: `Float?` optional radius of object.
    private func findRadius(_ distance: Float, _ box: CGRect) -> Float? {
        guard let camera = sceneView.session.currentFrame?.camera else {
            return nil
        }
        let imageResolution = camera.imageResolution
        let viewSize = sceneView.bounds.size
        let intrinsics = camera.intrinsics
        let focalLength = intrinsics[0][0]
        let xFovDegrees = atan(Float(imageResolution.width)/(2 * focalLength)) *
180/Float.pi
        let fov = xFovDegrees * Float(viewSize.height / viewSize.width)

        let heightInPixels = box.height * viewSize.height
        let widthInPixels = box.width * viewSize.width
        let sideInPixels = (heightInPixels + widthInPixels) / 2

        let boxSide = sideInPixels / viewSize.height
        let beta = fov * Float(boxSide) / 2

        let tg: Float = tan(beta * Float.pi / 180)
```

```
        let radius = distance * (sqrt(1 + pow(tg, 2)) + tg) * tg

        return radius
    }

    /// Finds distance to detected object.
    /// - Parameters:
    ///    - detection: detection from `ObjectDetector`
    /// - Returns: `Float?` distance to detection.
    private func findDistance(to detection: Detection) -> Float? {
        let centerCoordinate = center(of: detection.boundingBox)
        if let distance = distance(to: centerCoordinate) {
            return distance
        }
        return nil
    }

    /// Calculates distance to given point in centimeters
    /// - Parameters:
    ///    - box: `CGPoint` point to which distance will be measured
    /// - Returns: `Float?` optional distance to the given point
    private func distance(to point: CGPoint) -> Float? {
        guard let query = sceneView.raycastQuery(from: point, allowing:
.estimatedPlane, alignment: .any) else { return nil }
        guard let raycastResult = sceneView.session.raycast(query).first else {
return nil }

        let destinationTransform : matrix_float4x4 =
raycastResult.worldTransform
        let destinationPosition: SIMD3<Float> =
            .init(
                destinationTransform.columns.3.x,
                destinationTransform.columns.3.y,
                destinationTransform.columns.3.z
            )

        guard let camera = sceneView.session.currentFrame?.camera else {
return nil }
        let cameraTransform : matrix_float4x4 = camera.transform
        let cameraPosition: SIMD3<Float> =
            .init(
                cameraTransform.columns.3.x,
                cameraTransform.columns.3.y,
```

```
            cameraTransform.columns.3.z
        )

        let distance = simd_distance(cameraPosition, destinationPosition)
        let distanceInCm = distance * 100

        addDistanceNode(destinationTransform: destinationTransform, distance:
distanceInCm)

        return distanceInCm
    }
}
```