КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота
на здобуття ступеня магістра
за освітньо-науковою програмою "Інформатика"
спеціальності 122 "Комп'ютерні науки"
на тему:

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ РОЗПІЗНАВАННЯ ОБРАЗІВ

Виконала
студентка 2-го курсу
Аліна БІЛОНОЖКО

Науковий керівник:
доцент, к.п.н.
Наталія РУСІНА

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань
Студентка _____

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
«08» травня 2023 р., протокол N16
Завідувач кафедри
Микола НІКІТЧЕНКО _____

Київ – 2023

РЕФЕРАТ

Загальна кількість 83 сторінки, з них 74 сторінки тексту, 6 додатків, 16 ілюстрацій, 1 таблиця, 36 джерел посилань

РОЗПІЗНАВАННЯ ОБРАЗІВ, МАШИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, МОБІЛЬНИЙ ЗАСТОСУНОК, ФРЕЙМВОРК

Об'єктом дослідження є процес розпізнавання образів.

Метою роботи є проєктування та розробка мобільного застосунку на платформі iOS, з використанням технології розпізнавання образів для людей з фізичними та сенсорними порушеннями.

Предмет: мобільний застосунок, для полегшення життя людей з фізичними та сенсорними порушеннями

Інструменти розроблення: мови програмування Python та Swift, фреймоворки Vision, Core ML, AVFoundation, UIKit, середовище програмування Xcode.

Результати роботи: розглянуто процес розпізнавання образів для людей з фізичними та сенсорними порушеннями, проаналізовано наявні застосунки на ринку, розроблено інтерфейс користувача для мобільного застосунку на платформі iOS та використано машинне навчання для забезпечення коректної роботи застосунку.

Результати дослідження можуть бути застосовані для подальшої роботи з розпізнавання образів та використання людьми з фізичними та сенсорними порушеннями.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

AI - Artificial Intelligence, штучний інтелкт.

AVCaptureSession - Audiovisual Capture Session, сесія аудіовізуального захоплення.

AVCaptureVideoDataOutput - Audiovisual Capture Video Data Output, вихід даних аудіовізуального захоплення відео.

AVFoundation - Audiovisual Foundation, аудіовізуальна основа.

AVSpeechSynthesizer - Audiovisual Speech Synthesizer, аудіовізуальний синтезатор мови.

CIImage - Core Image Image, зображення Core Image.

CMSampleBuffer - Core Media Sample Buffer, буфер зразка ядра медіа.

CNN - Convolutional Neural Network, згорткова нейронна мережа.

Core ML - Core Machine Learning, основи машинного навчання.

CVPixelBuffer - Core Video Pixel Buffer, буфер пікселів Core Video.

iOS - iPhone Operating System, операційна система iPhone.

MacOS - Macintosh Operating System, операційна система Macintosh.

ML - Machine Learning, машинне навчання.

MLModel - Machine Learning Model, модель машинного навчання.

MVC - Model-View-Controller, Модель-Вид-Контролер.

SVM - Support Vector Machine, метод опорних векторів.

TTS - Text-to-Speech, текст в мовлення.

TvOS - Apple TV Operating System, операційна система Apple TV.

UIKit - User Interface Kit, набір інтерфейсу користувача.

VNCoreMLRequest - Vision Core ML Request, запит Vision Core ML.

VNImageRequestHandler - Vision Image Request Handler, обробник запиту зображення для Vision.

VNRequest - Vision Request, запит Vision.

WatchOS - Apple Watch Operating System, операційна система Apple Watch.

YOLO – You Only Look Once, ти дивишся лише раз.

ВСТУП

Оцінка сучасного стану об’єкта розробки. Мобільні застосунки є досить ефективними рішеннями у сфері технологій для підтримки людей з різними порушеннями, зокрема фізичними та сенсорними порушеннями.

Незважаючи на наявність різних аналогів на ринку, на сьогоднішній день, застосунки постійно удосконалюються, через розвиток машинного навчання, постійного аналізу потреб для підтримки людей з обмеженою мобільністю. Це означає, що ідея мобільного застосунку має потенціал для розвитку та залучення нових користувачів. Крім того, програмний продукт може допомогти користувачам відчувати комфорт у повсякденному житті.

Актуальність роботи та підстави для її виконання. За даними WHO, у світі понад мільярд людей мають різні форми порушень, що складає приблизно 16% населення [1]. Ця аудиторія зазнає труднощів у здійсненні повсякденних рутинних задач, зокрема навігації в урбаністичному середовищі. Тому дослідження присвячено розробці мобільного застосунку, що спрямований на поліпшення якості життя людей з фізичними та сенсорними порушеннями.

Застосунок має потенціал для соціальної інтеграції людей з фізичними та сенсорними порушеннями та може зробити внесок у поліпшення якості їхнього життя. Таким чином, його розробка є актуальною та важливою задачею в сучасному світі, для того, щоб люди з фізичними та сенсорними порушеннями могли відчувати себе повноцінними членами суспільства.

На даний час 30,44% користувачів мобільних пристроїв припадає на iOS [2]. Платформа має велику кількість інструментів для розробки застосунків, включаючи мову програмування Swift, яка є однією прогресивних мов програмування, та середовище розробки Xcode, що забезпечує широкі можливості для створення якісних та стабільних додатків. Отже, вибір платформи iOS для розробки може забезпечити його успішний розвиток та популярність серед користувачів.

Мета й завдання роботи. Метою роботи є проєктування та розробка мобільного застосунку на платформі iOS, з використанням технології розпізнавання образів для людей з фізичними та сенсорними порушеннями.

Задля досягнення поставленої мети були визначені такі завдання:

- провести аналіз наявних на ринку мобільних застосунків, для розпізнавання образів;
- розглянути та дослідити теоретичні основи машинного навчання;
- розробити та реалізувати мобільний застосунок для людей з фізичними та сенсорними порушеннями;
- поглибити та закріпити навички роботи з мовою програмування Swift та середовищем програмування Xcode.

Об’єкт, методи та засоби розроблення.

Об’єктом розроблення є мобільний застосунок для розпізнавання образів. Розробка застосунку базувалась на принципах мінімалізму та простоти інтерфейсу користувача, зокрема**:**

- легкість у використанні - застосунок був розроблений з урахуванням потреб користувачів, їх зручності та простоти використання;
- модульність – мобільний застосунок був розроблений з використанням модульної архітектури, що дозволяє легко розширювати його функціональність та проводити тестування окремих модулів.

Можливі сфери застосування для полегшення комунікації, для людей з вадами зору, для людей з фізичними та сенсорними порушеннями, для навчання.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Аналіз предметної області є важливим етапом при розробці проєкту, оскільки він дозволяє зрозуміти основні аспекти, проблеми та потреби, пов'язані з конкретною сферою діяльності. У випадку нашого проєкту, якому призначено надавати підтримку людям з фізичними та сенсорними порушеннями, аналіз предметної області стає ще більш важливим.

Завданням аналізу предметної області є з'ясування основних факторів, що впливають на цільову аудиторію застосунку, їхні потреби, проблеми та вимоги. Це дає можливість розробити ефективні рішення, які відповідають унікальним вимогам цільової аудиторії та допомагають вирішувати їхні проблеми.

1.1 Ідентифікація цільової аудиторії

Даний аспект дослідження доцільно провести, оскільки зрозуміння цільової аудиторії дозволяє належно спрямувати ресурси та зусилля на розробку ефективного та зручного рішення для користувачів.

Аудиторія може включати, але не обмежується нижче представленими категоріями [3].

- Люди з обмеженою рухливістю. Це можуть бути люди зі зниженою мобільністю, які мають складнощі з ходьбою, використанням ручок, сходинками або підйомами. Вони можуть користуватися палицями, ротаторами або іншими пристосуваннями для підтримки.
- Незрячі або слабозорі особи. Це можуть бути люди з важким або помірним зниженням зору, які потребують додаткової підтримки та інформації для безпечного руху. Вони можуть користуватися палицями для сліпих, керованими сигналізаціями або електронними пристроями для навігації.
- Люди з вадами слуху. Це можуть бути люди зі зниженим чи повним втратою слуху, які мають проблеми з отриманням аудіальної

інформації. Вони можуть використовувати сурдоперекладачі, коклеарні імпланти або інші асистивні пристрої для комунікації та сприйняття інформації.

- Люди з координаційними порушеннями. Це можуть бути люди з розладами координації рухів, які можуть мати складнощі зі стабільністю та управлінням своїм тілом під час ходіння. Вони можуть потребувати підтримки, стабільного опору або допомоги в управлінні своїм рухом.
- Люди з інтелектуальними порушеннями. Це можуть бути люди зі зниженим інтелектуальним розвитком або розладами аутистичного спектра, які можуть мати складнощі з орієнтацією та мобільністю. Вони можуть використовувати спеціальні програми або застосунки для навігації та підтримки.
- Вікова група: застосунок може бути корисним для різних вікових груп, починаючи від підлітків до літніх людей. Особливо користувачі похилого віку можуть використовувати застосунок для підтримки здорового та безпечного ходіння, орієнтації у місцевості.
- Професійні групи: застосунок може зацікавити людей, що працюють у сфері фізіотерапії, реабілітації або медичної допомоги. Він може служити додатковим інструментом для їхньої роботи та сприяти покращенню результатів терапії та реабілітації пацієнтів. Також може бути корисним для осіб, які надають підтримку або догляд за людьми з фізичними та сенсорними обмеженнями, оскільки він може допомогти полегшити їхню роботу та забезпечити кращий догляд.

Отже, підсумовуючи ідентифікацію цільової аудиторії, можна зробити наступні висновки. Цільова аудиторія застосунку - люди з фізичними та сенсорними обмеженнями, які мають потребу в допомозі та підтримці. Цільова аудиторія може мати різні рівні фізичних обмежень, включаючи проблеми з рухом, балансом та сприйняттям навколишнього середовища. Цільова аудиторія може мати проблеми зі спостереженням, безпекою, мобільністю, самостійністю та соціальним включенням.

1.2 Потреби цільової аудиторії

Оскільки цільова аудиторія складається з людей з фізичними та сенсорними обмеженнями та людей з психічними порушеннями вони можуть стикатись зі специфічними потребами та проблемами, які можуть бути вирішені за допомогою застосунку. Основні потреби та проблеми цільової аудиторії включають [4]:

- потреба у допомозі під час ходіння. Люди з фізичними та сенсорними обмеженнями можуть мати проблеми з рухом та балансом, що робить, наприклад ходіння, викликом для них. Застосунок може допомогти їм у полегшенні ходьби, надаючи підказки, підтримку та мотивацію;
- потреба у доступності та адаптованості. Цільова аудиторія може мати різні рівні фізичних або сенсорних обмежень, тому важливо, щоб застосунок був доступним та адаптованим до різних потреб користувачів. Наприклад, інтерфейс повинен бути легким у використанні та містити можливість налаштування під потреби користувача, такі як розмір шрифту чи кольорова схема. Крім того, застосунок повинен підтримувати сумісність з допоміжними технологіями, такими як читачі екрану або розширені можливості вводу;
- потреба у зручності та простоті використання: Люди з фізичними та сенсорними обмеженнями можуть мати обмежену моторику рук чи зору, тому важливо, щоб застосунок був зручним та простим у використанні. Інтерфейс повинен бути інтуїтивно зрозумілим та ергономічним, зроблений з урахуванням потреб цільової аудиторії;
- потреба у аналізі навколишнього середовища. Людям з психічними порушеннями застосунок може допомогти в розвитку пам’яті, вивченні навколишнього середовища;
- потреба у допомозі в орієнтації на місцевості. Застосунок може поліпшити орієнтацію на місцевості для людей з сенсорними порушеннями, а саме людям з важким або помірним зниженням зору: підказати розташування предмету, та назвавши оточуючі предмети.

Отже, підсумовуючи, можемо сказати, що актуальність ідентифікації цільової аудиторії та їх потреб базується на наступних факторах: соціальна потреба - люди з фізичними та сенсорними порушеннями є вразливою категорією населення, яка потребує спеціальної підтримки та допомоги; технологічні можливості - завдяки розширеному використанню мобільних пристроїв та доступу до розумних технологій, створення застосунку, спрямованого на покращення мобільності та безпеки людей, стає актуальним; потенційний вплив - покращення функціональних можливостей та самостійності цільової аудиторії може мати значний вплив на їхнє самопочуття, залучення до соціального життя та загальний розвиток. Розуміння їхніх потреб та проблем становить основу для створення ефективного рішення, що може поліпшити їхнє життя.

1.3 Огляд існуючих аналогів на ринку

У сучасному світі зростає роль технологій в покращенні якості життя людей. Однією з технологічних інновацій є розробка мобільних застосунків для покращення мобільності та допомоги людям з фізичними та сенсорними порушеннями у здійсненні їхньої повсякденної діяльності. Вже існує кілька мобільних застосунків для користування людьми з фізичними та сенсорними порушеннями, які допомагають покращити якість життя цієї групи людей.

Серед таких застосунків можна виділити:

"TapToTalk" [5] - це додаток для спілкування людей з психічними порушеннями, які мають проблеми з мовленням. Він дозволяє користувачам взаємодіяти з іншими людьми за допомогою символів та іконок.

Користувачі можуть створювати власні категорії та символи, що дозволяє їм налаштувати додаток на власні потреби. Застосунок містить багато вбудованих категорій та символів, що охоплюють широкий спектр тем і ситуацій, що забезпечує більш швидку та ефективну комунікацію.

Особливістю TapToTalk є його гнучкість та адаптивність (див. рис. 1.1). Користувачі можуть вибрати різні режими відображення символів,

включаючи сітку, карту та список. Крім того, додаток можна настроїти для різних рівнів мовленнєвого розвитку, що забезпечує більш індивідуалізований підхід до кожного користувача.

TapToTalk дозволяє користувачам з психічними порушеннями вести більш активний та самостійний спосіб життя, забезпечуючи їм зручну та доступну засоби комунікації.

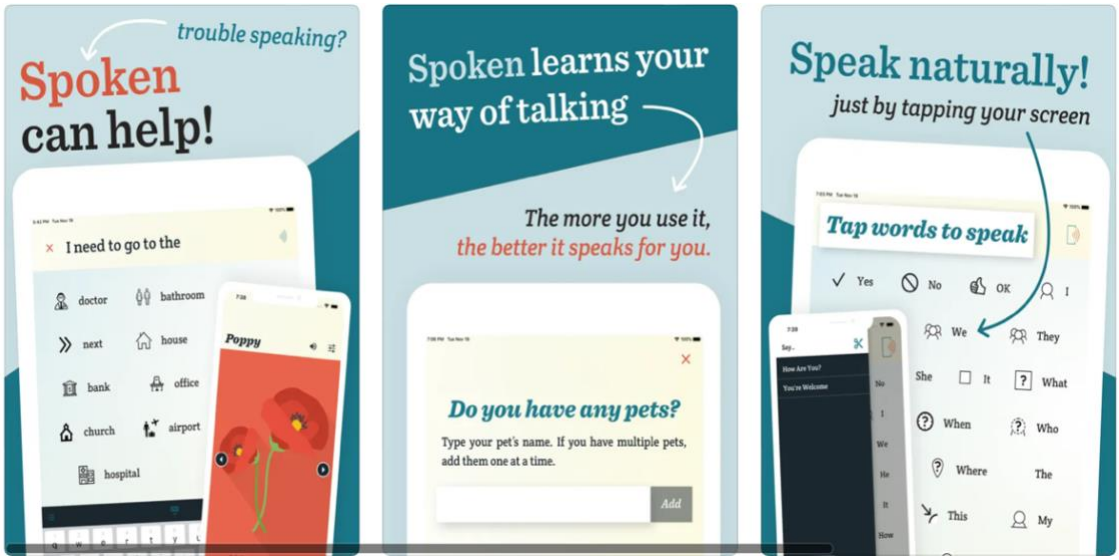


Рисунок 1.1 – Інтерфейс мобільного застосунку "TapToTalk"

"Dragon Anywhere" [6] - це мобільний додаток для диктовки тексту на мобільних пристроях під управлінням операційної системи iOS і Android. Розроблений компанією Nuance Communications, Inc., цей застосунок використовує технології ML і AI для точного та ефективного розпізнавання мовлення.

Однією з ключових функцій Dragon Anywhere є його здатність розпізнавати спеціальну мову, пов'язану з певними галузями, такими як медицина, юриспруденція та фінанси. Додаток також може автоматично форматувати текст, включаючи розділи, списки та заголовки. Крім того, він має функції редагування, які дозволяють коригувати текст шляхом голосового вводу (див. рис. 1.2).

Dragon Anywhere є платним додатком, але він пропонує безкоштовний пробний період для нових користувачів. Він також інтегрується з різними

хмарними сервісами, такими як Dropbox і Evernote, що дозволяє зберігати та збільшувати доступність текстових документів.

Загалом, Dragon Anywhere є потужним інструментом для тих, хто шукає ефективний спосіб вводити текст на мобільному пристрої. Його здатність розпізнавати спеціалізовану мову та форматувати текст автоматично робить його особливо корисним для професіоналів, які повинні працювати з документами на щоденній основі.

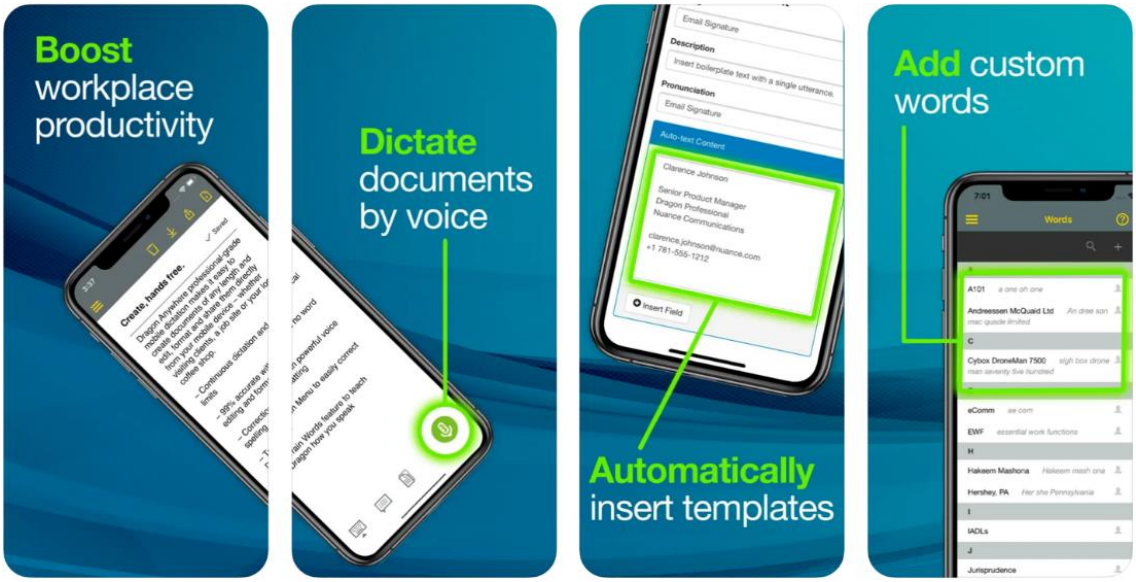


Рисунок 1.2 – Інтерфейс мобільного застосунку "Dragon Anywhere"

"Be My Eyes" [7] - це безкоштовний мобільний додаток, який допомагає людям зі зниженим зором або зовсім його втратою. Застосунок працює на платформах iOS та Android і дає можливість користувачам звернутися до волонтерів з усього світу, які можуть надати візуальну допомогу у різних ситуаціях (див. рис. 1.3). Наприклад, волонтер може допомогти з визначенням кольору одягу, зчитати етикетки на продуктах, або описати навколишнє середовище. Застосунок використовує технології відео-трансляції та чату для зв'язку між користувачами та волонтерами. Be My Eyes - це важливий інструмент для покращення якості життя людей зі зниженим зором, який допомагає їм зберігати незалежність та повноцінність у повсякденному житті.

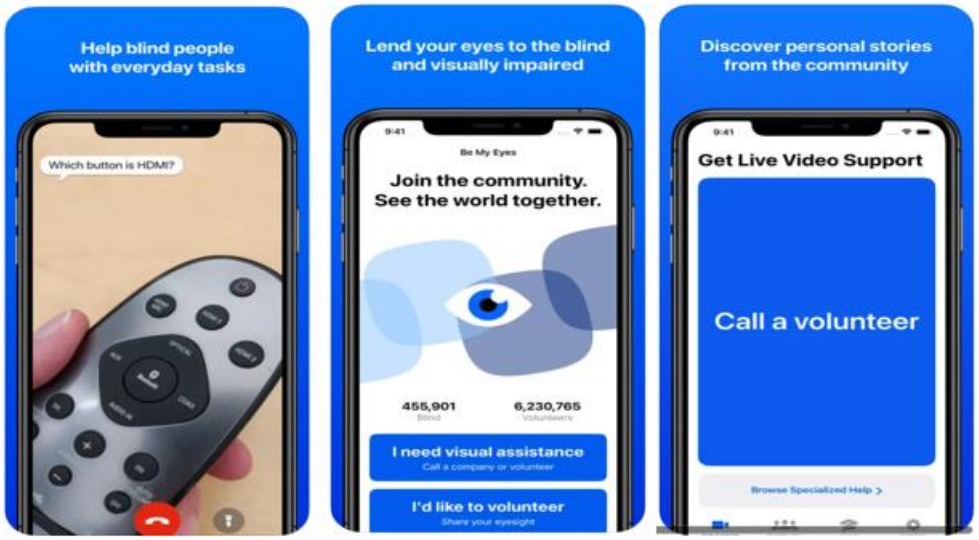


Рисунок 1.3 - Інтерфейс мобільного застосунку " Be My Eyes"

"Seeing AI" [8] - це мобільний додаток від Microsoft, який використовує технології ML для допомоги людям з візуальними обмеженнями. Застосунок дозволяє розпізнавати обличчя, текст, предмети, кольори та інші об'єкти, що допомагає користувачам зрозуміти своє оточення. Seeing AI також надає можливість розпізнавати валюту та відстань до предметів, а також розповідає, що відбувається на екрані застосунка (див. рис. 1.4). Окрім того, додаток може розпізнавати настрої людей на фотографіях та відтворювати текст, що знаходиться на зображеннях, за допомогою голосового синтезатора. Seeing AI є досить популярним застосунком серед людей з візуальними обмеженнями та отримав багато позитивних відгуків.

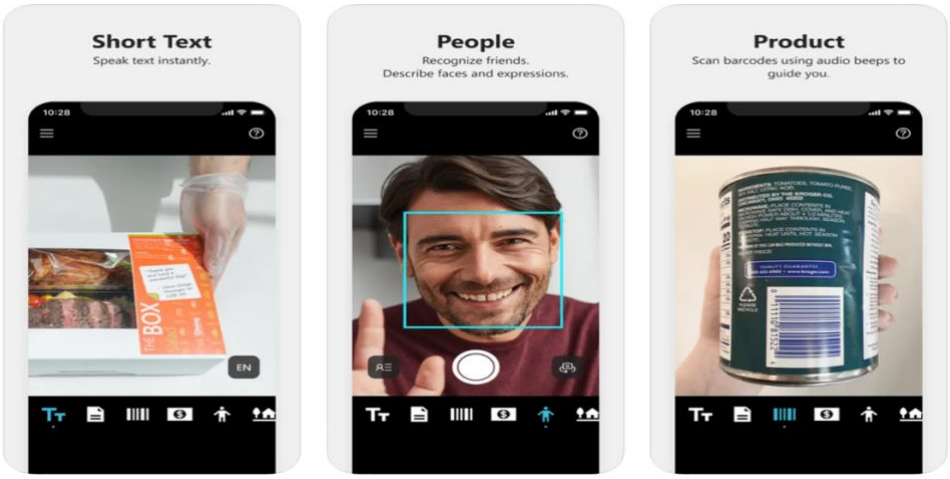


Рисунок 1.4 - Інтерфейс з основними функціями мобільного застосунку “Seeing AI”

"TapTapSee" [9] - TapTapSee є мобільним додатком для iOS та Android, призначеним для людей з візуальними порушеннями. Застосунок пропонує можливість розпізнавання образів на основі фотографій, зроблених на камеру смартфона (див. рис. 1.5). Користувач може зробити фото об'єкту, який потрібно розпізнати, а додаток поверне результат з описом того, що було знято на фото.

Технологічною основою додатку TapTapSee є комп'ютерне зорове розпізнавання, що дозволяє додатку розпізнавати різноманітні об'єкти, такі як тварини, продукти, одяг, меблі, гроші та інше. Результат розпізнавання описується аудіо-відгуком, який користувач може прослухати.

Додаток TapTapSee забезпечує більш незалежний та комфортний спосіб сприйняття навколишнього світу людьми з візуальними порушеннями. Завдяки його функціоналу користувачі можуть швидко та легко знайти інформацію про предмети навколо них без допомоги інших людей.

Рисунок 1.5 - Інтерфейс мобільного застосунку “TapTapSee”

"Aira" [10] - це додаток, розроблений для покращення життя людей з візуальними порушеннями. Застосунок використовує технології комп'ютерного зору та AI для надання допомоги людям з обмеженнями в баченні.

Користувачі можуть скористатись послугами операторів Aira, які можуть надати допомогу у багатьох різних сферах життя, таких як навігація в

незнайомому місці, читання документів, вибір та покупка товарів, а також забезпечення безпеки (див. рис. 1.6).

Aira використовує камеру смартфона або спеціальні окуляри, щоб передавати зображення на сервери Aira, де оператори здійснюють аналіз зображення та надають допомогу. Застосунок також містить функцію "Airaport", яка дозволяє користувачам замовляти послуги Aira безкоштовно в деяких аеропортах та інших громадських місцях.

Загалом, Aira дозволяє людям з візуальними порушеннями отримувати допомогу у багатьох різних сферах життя та забезпечує їм більшу незалежність та свободу.

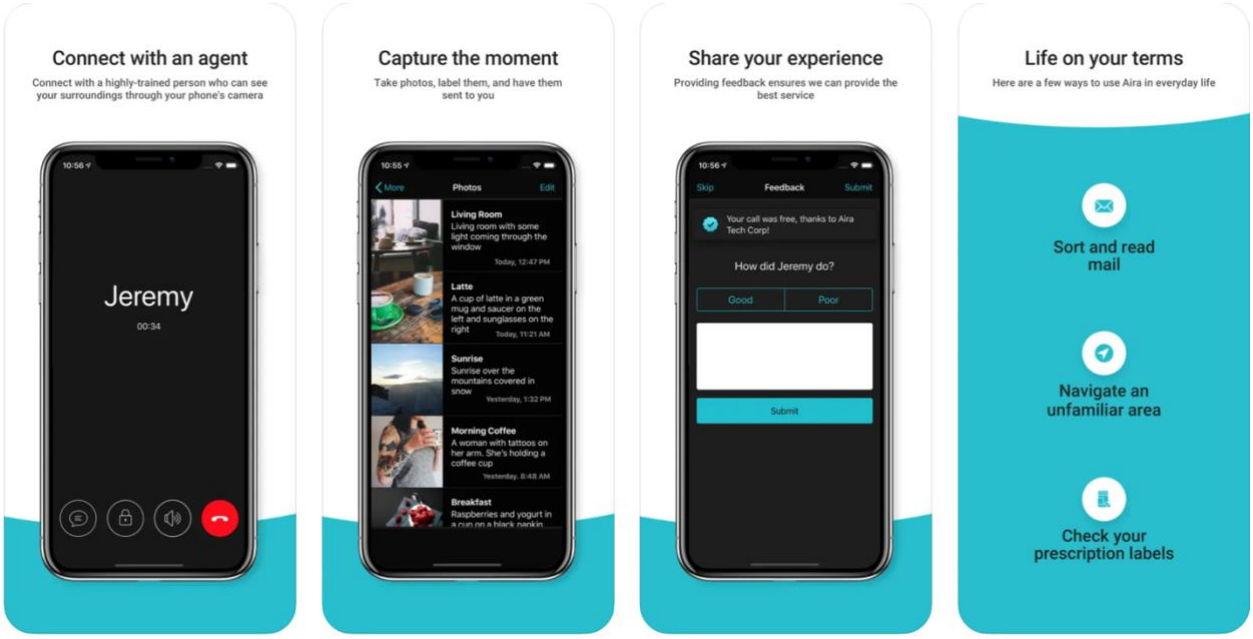


Рисунок 1.6 - Інтерфейс з функціями мобільного застосунку “Aira”

Підсумовуючи аналіз ринку, варто зазначити, що існують мобільні застосунки як безкоштовні, умовно безкоштовні та платні. Кожен з застосунків має свої переваги та недоліки. Вони допомагають покращити якість життя людей з цільової аудиторії.

РОЗДІЛ 2. ОГЛЯД ТЕХНОЛОГІЙ МАШИННОГО НАВЧАННЯ

Машинне навчання (ML) - це галузь штучного інтелекту, яка вивчає алгоритми та моделі, які дозволяють комп'ютерам самостійно вчитися та покращувати свою продуктивність на основі даних без явного програмування. Замість того, щоб написати конкретні інструкції для виконання завдання, ML дає можливість системам самостійно виявляти закономірності та робити прогнози на основі набору даних [11].

Штучний інтелект (AI) - це галузь науки, яка займається розробкою комп'ютерних систем та програм, що демонструють розумність і здатність до самостійного прийняття рішень, аналізу даних, навчання, розпізнавання образів та мови, планування та вирішення проблем. AI намагається моделювати імітацію інтелекту людини з метою досягнення різних завдань, використовуючи комп'ютери та алгоритми, які базуються на логіці, статистиці, нейронних мережах та інших методах обробки інформації [12].

Використання ML в розпізнаванні образів полягає у його потенціалі вирішувати складні завдання аналізу даних, включаючи розпізнавання об'єктів, класифікацію зображень та виявлення паттернів. Застосування ML дозволяє автоматизувати процеси аналізу великих обсягів даних та отримувати цінні висновки та інсайти.

У сучасному світі, де доступні великі обсяги даних та розвиваються нові технології обробки інформації, використання ML стає дедалі важливішим для вирішення складних завдань розпізнавання образів. Це може знайти застосування в таких галузях, як медицина, автоматизація, транспорт, безпека та багато інших.

Дослідження актуальності використання ML в розпізнаванні образів базується на роботах [11], [12], [13] авторів у галузі ML та комп'ютерного зору.

2.1 Значення машинного навчання в розробці застосунків

ML та AI відіграють важливу роль в розробці застосунків, зокрема у сфері AI для мобільних додатків. Давайте розглянемо їх значення в контексті розробки застосунків.

ML дозволяє створювати моделі та алгоритми, які можуть навчатися на основі великих обсягів даних та виконувати завдання без явного програмування. У розробці мобільних застосунків, ML може бути використане для розпізнавання образів, класифікації даних, прогнозування, автоматичного перекладу мови та багатьох інших завдань. Це дозволяє створити застосунки, які можуть адаптуватися до потреб користувачів і надавати більш персоналізований досвід [11].

AI включає в себе не тільки ML, але й інші підходи до розумової моделі, такі як логічне мислення, експертні системи, нейронні мережі, обробка природної мови та багато інших. У розробці мобільних застосунків, AI може бути використаний для створення інтелектуальних асистентів, автоматичного прийняття рішень, обробки та аналізу великих обсягів даних, розпізнавання голосу та образів та багатьох інших функцій, які покращують користувацький досвід [12].

Застосування ML та AI в розробці мобільних застосунків дозволяє створювати розумні, адаптивні та інноваційні рішення. Вони забезпечують взаємодію з користувачами дослідили значення ML та AI в розробці мобільних застосунків, і можна виокремити кілька ключових переваг цих технологій [14].

- Покращений користувацький досвід: ML та AI дозволяють створювати застосунки, які можуть навчатися з взаємодії з користувачем та адаптуватися до його потреб. Наприклад, вони можуть прогнозувати вподобання користувача, рекомендувати персоналізований контент або автоматично адаптувати інтерфейс для зручного використання.
- Автоматизація та оптимізація завдань: ML дозволяє автоматизувати рутинні та складні завдання, що збільшує продуктивність та зменшує витрати часу. Наприклад, застосунки можуть автоматично

класифікувати та обробляти дані, розпізнавати образи або вести автоматичний аналіз даних.

- Покращена точність та ефективність: ML дозволяє створювати моделі, які можуть виявляти складні залежності в даних та забезпечувати високу точність результатів. Це допомагає покращити рішення, прийняті застосунком, та забезпечує більш ефективну роботу з обробкою даних.
- Аналітика та прогнозування: ML дозволяє аналізувати великі обсяги даних та робити прогнози на основі цих даних. Застосунки можуть надавати користувачам аналітичні звіти, прогнозувати тренди та підказувати оптимальні рішення на основі статистичних даних.
- Розпізнавання образів та голосу: ML дозволяє розробляти застосунки з функціями розпізнавання образів та голосу. Це дозволяє створювати застосунки, які можуть розпізнавати обличчя, об'єкти, логотипи та інші візуальні елементи. Також, застосунки можуть розпізнавати та інтерпретувати голосові команди користувача, що дозволяє реалізувати голосовий інтерфейс та забезпечити зручну навігацію та взаємодію з застосунком.

Застосування ML в розробці застосунків дає можливість створити інноваційні та ефективні рішення, які покращують користувацький досвід, автоматизують рутинні завдання, забезпечують аналітику та прогнозування, розпізнають образи та голос, а також допомагають у прийнятті рішень. Ці технології використовуються в різних сферах, від медицини та фінансів до туризму та розваг, і мають значний потенціал для подальшого розвитку та застосування у майбутніх застосунків [14]. Завдяки AI та ML, застосунки можуть стати інтелектуальними, адаптивними та орієнтованими на користувача.

2.2 Огляд властивостей машинного навчання

ML має ряд властивостей, які роблять його потужним інструментом для аналізу даних та прийняття рішень. Вивчення цих властивостей допоможе

зрозуміти, як ML працює і як його використовувати в різноманітних областях. Основні властивості ML є ключовими аспектами цієї галузі і допомагають забезпечити його успішність у різноманітних задачах [15]. Вони включають:

1. Навчання на основі даних: ML використовує великі обсяги даних для тренування моделей. Ці дані можуть бути розміченими (з мітками) або нерозміченими (без міток). Моделі вчаться розпізнавати закономірності в даних та виробляти корисні висновки.

2. Автоматична адаптація: машинні моделі можуть автоматично адаптуватися до змін в даних або умовах оточення. Вони можуть самостійно оновлювати свої параметри та вдосконалювати свої навички, підлаштовуючись до нових вхідних даних.

3. Генералізація: ML ставить перед собою завдання генералізації, тобто здатності до правильної реакції на нові, раніше невідомі дані. Моделі, навчені на певних даних, повинні мати здатність узагальнювати свої знання і застосовувати їх до нових ситуацій.

4. Прийняття рішень: машинні моделі можуть здійснювати прийняття рішень або робити прогнози на основі вхідних даних. Вони можуть виконувати класифікацію, регресію, кластеризацію, або вирішувати інші завдання на основі навчальних прикладів. Наприклад, у медицині ML може допомагати у визначенні діагнозу на основі медичних зображень або клінічних даних.

5. Підтримка прийняття рішень: машинні моделі можуть служити інструментами для підтримки прийняття рішень. Вони можуть аналізувати великі обсяги даних, виявляти складні залежності та робити прогнози. Це може бути корисним для бізнесу, фінансів, маркетингу, громадської безпеки та багатьох інших галузей.

6. Використання інструментів ML: у розробці застосунків ML використовуються спеціалізовані інструменти, такі як бібліотеки ML, середовища для розробки моделей, платформи для обчислень та інші. Ці

інструменти допомагають у тренуванні, валідації та оптимізації моделей, а також у їх розгортанні та використанні в реальному часі.

2.3 Типи алгоритмів машинного навчання

У світі ML і AI існує широкий спектр алгоритмів, які можуть бути використані для розв'язання різноманітних завдань. Знання про різні типи алгоритмів ML є важливим для розробки застосунків, оскільки вони надають різні можливості та підходи до обробки даних і здійснення прогнозів. У цьому пункті ми розглянемо основні типи алгоритмів ML та їх особливості.

Навчання з учителем

Навчання з учителем є одним з ключових підходів до ML, де модель навчається на основі пар вхідних даних та відповідних міток чи правильних відповідей. У цьому підході модель намагається знайти залежності між вхідними змінними та вихідними значеннями, щоб зробити передбачення на нових даних.

Отже, воно передбачає навчання моделі на навчальному наборі з мітками, де кожна вхідна точка даних пов'язана з відповідною вихідною міткою. Метою навчання з вчителем є вивчення функції відображення, яка здатна точно передбачати вихідні мітки для нових, невидимих даних [15].

Основні концепції в навчанні з вчителем включають:

- Навчальний набір даних: це мітковий набір даних, який використовується для навчання моделі. Він складається з вхідних ознак (незалежних змінних) та відповідних вихідних міток (залежних змінних);
- Вхідні ознаки: це змінні або атрибути, які описують точки даних. Вони можуть бути числовими або категоріальними і використовуються як вхідні дані для моделі;
- Вихідні мітки: це цільові змінні, які ми хочемо передбачити або класифікувати. Модель вивчає асоціацію вхідних ознак з їх відповідними вихідними мітками під час процесу навчання;

- Навчання моделі: у навчанні з вчителем модель навчається шляхом надання їй набору міткових точок даних. Модель вивчає з вхідних ознак і відповідних вихідних міток, щоб налаштувати свої внутрішні параметри і оптимізувати свою продуктивність;
- Оцінка моделі: після навчання модель оцінюється на окремому наборі даних, який називається тестовим;

Це навчання застосовується у широкому спектрі задач, таких як класифікація, регресія, рекомендації та багато інших.

Лінійна регресія є одним з простих і широко використовуваних алгоритмів навчання з вчителем. Вона використовує лінійну функцію для встановлення залежності між вхідними ознаками і вихідними значеннями. Цей алгоритм може бути застосований для задач регресії, де метою є передбачення неперервних числових значень [16].

SVM є потужним алгоритмом, який використовується як для задач класифікації, так і для регресії. Він шукає границю прийняття рішень, яка максимально відділяє точки даних різних класів або найкраще підходить до регресійних значень. SVM може використовувати різні ядра для знаходження нелінійних залежностей між даними [15].

Ці алгоритми є лише кількома прикладами методів навчання з вчителем. Наявність різних алгоритмів дозволяє вибирати найбільш підходящий для конкретної задачі і типу даних.

Навчання без вчителя

Навчання без вчителя є іншим важливим типом алгоритмів ML, де модель навчається на основі безміткових даних. Навчання без вчителя є одним з підходів у ML, де модель вчиться з набору даних, який не має вихідних міток або навчального супроводу. Основна ідея в тому, щоб модель самостійно виявляла корисні структури, закономірності та складнощі у наборі даних, не отримуючи інформацію про їх відповіді або категорії [16].

Приклади алгоритмів навчання без вчителя:

- Кластеризація: алгоритми кластеризації, такі як k-середніх або ієрархічна кластеризація, дозволяють групувати схожі об'єкти разом у кластери на основі подібності або відстані між ними. Наприклад, можна застосувати кластеризацію для групування користувачів на основі спільних характеристик.

- Асоціативні правила: алгоритми дозволяють виявляти цікаві зв'язки та залежності між об'єктами в наборі даних. Наприклад, алгоритм Apriori використовується для виявлення асоціативних правил в магазинних транзакціях, де можна з'ясувати, які товари часто купуються разом.

- Рекомендаційні системи: алгоритми допомагають пропонувати користувачам релевантні рекомендації на основі їхніх попередніх взаємодій або спільних інтересів. Наприклад, алгоритм колаборативного фільтрування використовується для рекомендацій фільмів або товарів на основі спільних вподобань користувачів.

- Аналіз асоціацій: алгоритми виявляють статистичні зв'язки та закономірності між різними змінними або подіями. Наприклад, алгоритм аналізу асоціацій може виявити, що покупці, які купують певні продукти, часто також купують інші специфічні продукти.

- Визначення тематики та кластеризація текстів: алгоритми дозволяють категоризувати та групувати текстові документи за їхнім змістом або тематикою. Наприклад, алгоритми LDA або HDP використовуються для тематичного моделювання текстів.

- Розмітка даних: алгоритми допомагають розмітити або класифікувати об'єкти або дані згідно з певними категоріями або мітками. Наприклад, алгоритми кластеризації даних або класифікатори, такі як SVM або навчання DNNs, можуть бути використані для розмітки зображень або інших типів даних.

- Генеративні моделі: алгоритми дозволяють генерувати нові дані або зображення, які мають подібні характеристики до навчального набору.

Наприклад, VAE або GAN використовуються для генерації нових зображень або текстів.

Навчання без вчителя знаходить широке застосування у багатьох галузях, таких як обробка природної мови, комп'ютерне зору, аналіз даних та багато інших. Воно дозволяє виявляти складні залежності, розпізнавати шаблони та виявляти цікаві структури в даних без необхідності попереднього позначення або супроводу вчителя.

Змішані алгоритми машинного навчання

Змішані алгоритми ML комбінують підходи навчання з вчителем та без вчителя для отримання кращих результатів. Вони можуть використовуватися, коли в наявності є часткові мітки даних або коли необхідно вирішити задачу з декількома типами вихідних даних. Змішані алгоритми ML можуть бути корисними для задач, де звичайні підходи з вчителем та без вчителя не дають оптимальних результатів [17].

CNN для розпізнавання образів

CNN є потужними алгоритмами ML для обробки зображень та розпізнавання образів. Вони мають спеціалізовану архітектуру, яка дозволяє виявляти локальні особливості в зображеннях та використовувати їх для класифікації, сегментації, виявлення об'єктів та інших задач, пов'язаних з обробкою зображень.

Архітектура CNN базується на принципах конволюції, пулінгу та повторення. Вона складається з трьох основних типів шарів: конволюційних шарів, шарів пулінгу та повнозв'язаних шарів [11].

Конволюційні шари використовуються для виявлення локальних функцій або ознак на зображенні. Кожен шар складається з набору фільтрів, які здійснюють операцію конволюції над вхідним зображенням. Це дозволяє виявити різні шаблони, контури та особливості у зображенні.

Шари пулінгу використовуються для зменшення розмірності зображення та виділення найважливіших ознак. Зазвичай використовується

пулінг за максимальним значенням, де вибирається найбільше значення з кожного регіону зображення.

Повнозв'язані шари використовуються для класифікації зображення на основі знайдених ознак. Вони приймають векторизовані ознаки з попереднього шару та застосовують лінійну комбінацію та нелінійні функції активації для визначення кінцевого результату класифікації.

Основні особливості архітектури CNN, які роблять її ефективною для розпізнавання образів, включають [11]:

- Шари згортки: Це основний компонент архітектури CNN, який використовує згорткові фільтри для виявлення особливостей на зображенні. Кожен фільтр проходить по зображенню і відбирає корисну інформацію, яка допомагає в розпізнаванні образів.
- Шари пулінгу: після проходження через шари згорток, зображення зменшуються шляхом використання шарів пулінгу. Це допомагає знизити кількість параметрів і обчислювальну складність моделі, зберігаючи при цьому важливі особливості зображення.
- Повнозв'язані шари: після шарів згортки і пулінгу використовуються повнозв'язані шари, які виконують класифікацію зображення на певні категорії. Ці шари отримують векторизований вихід від попередніх шарів і здійснюють класифікацію на основі цього вектора.
- Шари активації: для введення нелінійності в мережу використовуються функції активації, такі як ReLU, Sigmoid або Tanh. Це допомагає моделі узагальнювати та вивчати більш складні залежності в даних.
- Застосування передобробки даних: перед направленням даних на вхід мережі, їх часто піддають передобробці, такі як нормалізація значень пікселів або застосування трансформацій, які поліпшують якість зображень та забезпечують кращу роботу моделі.

Ці шари співпрацюють, щоб виявити, виділити і класифікувати важливі особливості на зображеннях. Зазвичай архітектура CNN складається з декількох повторюваних блоків, кожен з яких може містити такі шари. Кожен

блок може включати один або кілька шарів згортки, пулінгу та активації, що допомагає нейронній мережі відповідати на більш складні візуальні особливості. На виході повнозв'язаного шару зазвичай використовується функція активації Softmax, яка перетворює вихідні значення на ймовірності для кожного класу.

Основна ідея застосування цих шарів в тому, що вони дозволяють нейронній мережі автоматично вчитися виявляти ієрархічні рівні абстракції в зображеннях [11]. Від простих функцій, таких як границі, до складних візуальних концепцій, які можуть представляти об'єкти або сцени.

Ці основні шари і їх взаємодія роблять архітектуру CNN ефективною для завдань розпізнавання образів, таких як класифікація зображень, виявлення об'єктів або сегментація зображень. Вони дозволяють моделі навчатися виявляти складні візуальні паттерни та зробити точні прогнози зображень з високою швидкістю та ефективністю.

РОЗДІЛ 3. ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

3.1 Мови програмування

Swift

Swift - це мова програмування, яка була розроблена компанією Apple Inc. і була представлена в 2014 році. Swift є мовою програмування високого рівня, що дозволяє розробляти застосунки для платформ iOS, macOS, watchOS та tvOS [18].

Swift є досить сучасною мовою програмування, яка має декілька важливих особливостей, що роблять її сучасною [19]:

1. Безпека: Swift має вбудовані засоби для запобігання виникненню помилок, таких як доступ до невизначених значень або падіння програми через винятки. Це можливо завдяки використанню опціональних типів даних, вбудованої підтримки винятків та валідації вхідних даних.

2. Функціональне програмування: Swift підтримує функціональне програмування, яке є дуже популярним у сучасному програмуванні. Це дозволяє програмістам писати більш компактний та елегантний код, що знижує кількість помилок та покращує ефективність програми.

3. Асинхронне програмування: Swift має вбудовану підтримку асинхронного програмування, що дозволяє програмістам створювати застосунки, які можуть обробляти багато запитів одночасно. Це є дуже важливим для розробки застосунків, які працюють з мережами та базами даних.

4. Протоколи та розширення: Swift має вбудовану підтримку протоколів та розширень, що дозволяє програмістам легко розширювати функціональність класів та структур. Це дозволяє програмістам писати більш модульний та легкий для розуміння код.

5. Playground: Swift має вбудовану функцію Playground, яка дозволяє програмістам тестувати свій код без створення окремого проєкту. Це дозволяє програмістам швидко експериментувати зі своїм кодом та дізнаватися про нові функції мови.

6. Вбудована підтримка для iOS: Swift є основною мовою програмування для розробки застосунків для платформи iOS. Це дозволяє програмістам швидко та ефективно розробляти застосунки для iPhone, iPad та інших пристроїв на базі iOS.

7. Високорівневий синтаксис: Swift має дуже зрозумілий та логічний синтаксис, що дозволяє програмістам писати код швидше та з меншою кількістю помилок. Наприклад, в Swift відсутній символ крапки з комою для кінця рядка, що дозволяє програмістам писати більш зрозумілий та легкий для читання код.

8. Відкритий код: Swift є відкритим джерелом, що дозволяє програмістам розробляти свої власні інструменти та бібліотеки на основі мови. Це стимулює розвиток мови та допомагає програмістам швидше та ефективніше розробляти свої застосунки.

9. Вбудована підтримка для ML: Swift має вбудовану підтримку для ML через фреймворк Core ML. Це дозволяє програмістам легко створювати моделі ML та використовувати їх у своїх застосунках.

10. Розвиток мови: Swift є молодою мовою програмування, яка активно розвивається та отримує оновлення. Наприклад, з'явилися нові функції у версії Swift 5.5, такі як асинхронні функції та удосконалені конкурентні операції. Це дозволяє програмістам отримувати нові можливості та вдосконалювати свої застосунки на базі Swift.

Основні недоліки мови:

1. Молодість мови: Swift є досить молодою мовою програмування, що може призвести до змін у майбутньому. Наприклад, можливо, що у майбутньому будуть введені зміни до синтаксису мови, що може спричинити проблеми для існуючого коду.

2. Недоступність для інших платформ: Swift був розроблений для розробки застосунків для платформи Apple, що обмежує його використання для інших платформ, таких як Windows чи Android.

3. Великі розміри застосунків: Swift має великі розміри бібліотек, що може при звести до збільшення розміру застосунків. Це може бути проблемою для застосунків з обмеженим простором пам'яті на пристрої.

4. Строгість типів даних: Swift має строгу систему типів даних, що може призвести до додаткового часу для програмістів на перевірку типів та зміну типів даних [20]. Це може бути проблемою для більш складних додатків, де зміна типів даних відбувається досить часто.

5. Необхідність використання Xcode: Для розробки застосунків на Swift необхідно використовувати Xcode, інтегроване середовище розробки, що доступне лише для платформи macOS. Це може бути проблемою для розробників, які використовують іншу операційну систему.

6. Обмежена підтримка документації: У порівнянні з іншими мовами програмування, Swift має обмежену кількість документації та прикладів коду. Це може призвести до складнощів для початківців у вивченні мови.

Swift є мовою програмування з багатьма перевагами, такими як безпека, швидкість, вбудована підтримка паралельного та функціонального програмування, високорівневий синтаксис, вбудована підтримка для iOS та macOS та відкритий код [18]. Проте, мова також має недоліки, такі як молодість мови, обмежена доступність для інших платформ, великі розміри застосунків, строгість типів даних, необхідність використання Xcode та обмежена підтримка документації. З урахуванням цих переваг та недоліків, Swift є сучасною мовою програмування, яка може бути вибрана для розробки застосунків на платформі iOS та macOS.

Python

Python - це інтерпретована, високорівнева мова програмування з динамічною типізацією. Вона зазвичай використовується для розробки веб-застосунків, ігор, різноманітних наукових досліджень, аналізу даних та багатьох інших задач [21].

Однією з головних переваг Python є його простота та легкість вивчення. Синтаксис мови дуже простий та зрозумілий, що дозволяє розробникам

швидко створювати прототипи та розвивати програми. Python має велику кількість бібліотек та фреймворків, що полегшує розробку та дозволяє розробникам створювати складні програми з мінімальними зусиллями.

Ще однією важливою перевагою Python є його кросплатформенність. Python працює на різних операційних системах, таких як Windows, macOS, Linux та інші. Це дозволяє розробникам створювати програми, які працюють на будь-якій платформі без необхідності відновлювати код для кожної платформи окремо.

Python також має велику кількість бібліотек та фреймворків для різних задач. Наприклад, бібліотека NumPy використовується для роботи з масивами даних, Pandas - для обробки та аналізу даних, Flask - для створення веб-застосунків, Pygame - для розробки ігор та інші.

Однак, Python має деякі недоліки, такі як менша продуктивність порівняно з деякими іншими мовами програмування, такими як C++, та менша ефективність в роботі з великими обсягами даних. Також використання інтерпретатора Python може призвести до певної втрати продуктивності, порівняно з мовами, які компілюються в бінарний код. Однак, насправді ці недоліки не є критичними для більшості застосувань, для яких використовується Python.

Python залишається однією з найпопулярніших мов програмування у світі завдяки своїм перевагам та простоті використання. Python використовується в багатьох відомих компаніях, таких як Google, Facebook, Dropbox та інші, для розробки різноманітних продуктів та застосунків. Python має активну спільноту розробників, яка постійно розвиває нові бібліотеки та фреймворки, що дозволяє розробникам зосередитися на створенні нових функцій та інновацій у своїх проєктах замість вирішення технічних проблем.

У загальному, Python є ефективною, простою та потужною мовою програмування, що забезпечує багато можливостей для розробки застосунків та програм [21]. Його простота та легкість вивчення дозволяє розробникам швидко створювати програми та прототипи, а велика кількість бібліотек та

фреймворків забезпечує широкий спектр можливостей для розробки різноманітних застосувань. З урахуванням цих переваг, Python буде важливою мовою програмування на багато років вперед.

3.2 Середовище розробки Xcode

Xcode є IDE для розробки програмного забезпечення для платформи Apple, таких як iOS, macOS, watchOS та tvOS. Xcode містить різноманітні інструменти та ресурси для розробки застосунків, включаючи редактор коду, компілятор, дебаггер, візуальний дизайнер інтерфейсів, інструменти для тестування та аналізу профілювання застосунків [22].

Однією з головних переваг Xcode є те, що він включає в себе велику кількість інструментів та ресурсів для побудови застосунків для платформи Apple. Редактор коду Xcode підтримує різні мови програмування, включаючи Swift, Objective-C, C++ та інші, що дозволяє програмістам вибирати мову, яка найбільше підходить для їх проєкту. Крім того, Xcode має вбудовану підтримку для інструментів керування версіями, що дозволяє програмістам працювати зі своїм кодом в команді та контролювати його історію змін.

Xcode також має вбудовані інструменти для дизайну та розробки інтерфейсів користувача. Інтерфейсний конструктор Xcode дозволяє програмістам візуально створювати інтерфейси користувача, що зменшує кількість коду, який необхідно написати вручну. Крім того, Xcode має вбудовані інструменти для тестування та аналізу профілювання застосунків, що дозволяє програмістам налагоджувати та оптимізувати свої застосунки.

Xcode також має велику кількість документації, прикладів коду та ресурсів для розробників, що дозволяє їм вивчати нові технології та інструменти для розробки застосунків. Крім того, Xcode має активну спільноту розробників, що дозволяє програмістам отримувати підтримку та поради від інших спеціалістів.

Однією з головних переваг Xcode є те, що він безкоштовний та доступний для завантаження на офіційному сайті Apple. Це робить його

доступним для широкої аудиторії розробників, які можуть використовувати Xcode для розробки своїх застосунків.

Однак, Xcode має деякі недоліки. Наприклад, для використання Xcode необхідно мати пристрій з операційною системою macOS, що може бути проблемою для розробників, які використовують інші операційні системи. Крім того, Xcode може бути дуже важким та споживати багато ресурсів, що може призвести до повільної роботи пристрою.

Однак, незважаючи на ці недоліки, Xcode залишається одним з найкращих інструментів для розробки програмного забезпечення для платформи Apple. Він має широку підтримку спільноти розробників та оновлюється регулярно з новими функціями та покращеннями. Крім того, Xcode безкоштовний та доступний для завантаження на офіційному сайті Apple, що робить його доступним для розробників з різних країн та різного досвіду.

Отже, Xcode - це потужний інструмент для розробки програмного забезпечення для платформи Apple, який має свої переваги та недоліки. Якщо використовувати Xcode правильно та знати його особливості, то цей інструмент може стати відмінним помічником у розробці застосунків для платформи Apple.

3.3 Фреймворки

Core ML

Core ML - це фреймворк ML, розроблений компанією Apple, який дозволяє розробникам інтегрувати моделі ML у свої застосунки для платформ iOS та macOS. Core ML працює з різними типами моделей ML, включаючи DNNs, та дозволяє розробникам використовувати готові моделі або створювати свої власні [23].

Однією з головних переваг Core ML є його інтеграція з екосистемою Apple, що дозволяє розробникам легко і швидко інтегрувати моделі ML в свої застосунки. Це забезпечує зменшення часу та зусиль, необхідних для створення застосунків з ML для платформи Apple.

Core ML також має вбудовану підтримку для багатьох моделей ML, включаючи відомі бібліотеки, такі як Keras, Caffe та TensorFlow. Це дозволяє розробникам використовувати існуючі моделі та переносити їх безпосередньо в свої застосунки.

Ще однією перевагою Core ML є його продуктивність та швидкість. Core ML працює на пристроях Apple, що дозволяє використовувати апаратну прискорення, що забезпечує швидкість та продуктивність під час роботи з моделями ML [24].

Core ML має також деякі недоліки. Однією з них є обмежена підтримка моделей ML, що може бути проблемою для розробників, які використовують моделі, які не підтримуються Core ML. Крім того, Core ML може бути дещо складним у використанні для розробників з недостатнім досвідом у ML та програмуванні.

Однак, з розвитком Core ML та розширенням його можливостей, ці недоліки можуть бути знижені. Крім того, спільнота розробників Apple продовжує активно працювати над розширенням можливостей Core ML та підтримки нових моделей ML.

До прикладів використання Core ML можна віднести розпізнавання зображень та об'єктів на зображеннях, класифікацію тексту, голосовий інтерфейс, прогнозування поведінки користувачів та багато іншого. Core ML може бути використаний в різних галузях, включаючи медицину, фінанси, автомобільну промисловість та інші.

Загалом, Core ML є потужним та ефективним фреймворком ML, який дозволяє розробникам інтегрувати моделі ML в свої застосунки для платформи Apple зі значною легкістю. Core ML має багато переваг, таких як вбудована підтримка для багатьох типів моделей ML, швидкість та продуктивність та простота використання [24]. Незважаючи на деякі недоліки, Core ML залишається одним з найбільш популярних ML фреймворків для платформи Apple.

AVFoundation

AVFoundation - це фреймворк для обробки мультимедійних даних, що входить до стандартного набору інструментів для розробки застосунків для платформи Apple. Цей фреймворк дозволяє розробникам працювати з аудіо та відео на iOS, macOS та інших платформах Apple, забезпечуючи широкий спектр можливостей для обробки та маніпулювання мультимедійними даними [25].

Однією з головних переваг AVFoundation є її спрощення процесу роботи з аудіо та відео для розробників. AVFoundation дозволяє легко інтегрувати різні формати мультимедіа в застосунки, що забезпечує ефективне та швидке зчитування, запис та редагування аудіо та відео. Крім того, AVFoundation дозволяє розробникам працювати з різними аудіо та відео форматами, включаючи AAC, MP3, H.264, MPEG-4 та інші [25].

Ще однією перевагою AVFoundation є її продуктивність та ефективність. Фреймворк AVFoundation дозволяє швидко та ефективно працювати з мультимедійними даними, що забезпечує більш швидку роботу застосунків та ефективне використання ресурсів пристроїв.

AVFoundation має також і деякі недоліки. Наприклад, для розробників з недостатнім досвідом може бути складним використання AVFoundation, оскільки цей фреймворк вимагає певних знань та досвіду у роботі з мультимедіа та іншими технологіями. Крім того, у деяких випадках можуть виникати проблеми з оптимізацією продуктивності для великих обсягів даних.

Загалом, AVFoundation є потужним та ефективним інструментом для роботи з мультимедійними даними в застосунках для платформи Apple. Цей фреймворк дозволяє розробникам ефективно та швидко працювати з аудіо та відео та значно зменшує складність роботи з мультимедіа. AVFoundation має деякі недоліки, але з розвитком технологій та розширенням можливостей фреймворку, ці недоліки можуть бути зменшені. AVFoundation залишається одним з найпопулярніших інструментів для роботи з мультимедіа на платформі Apple і може бути використаний для розробки різних застосунків,

включаючи додатки для зйомки та редагування відео, музичні застосунки та інші.

UIKit

UIKit - це фреймворк для розробки інтерфейсу користувача на платформах iOS, iPadOS, tvOS та watchOS. Цей фреймворк містить набір інструментів та класів, що дозволяють розробникам створювати графічний інтерфейс користувача для застосунків, що працюють на платформах Apple [26].

Однією з головних переваг UIKit є його простота та легкість використання. Цей фреймворк містить ряд готових елементів інтерфейсу, таких як кнопки, текстові поля, таблиці, списки та інші, що значно полегшує розробку інтерфейсу користувача. Крім того, UIKit дозволяє розробникам створювати свої власні елементи інтерфейсу, що дозволяє їм забезпечувати більш гнучкий та індивідуальний підхід до дизайну застосунків.

UIKit також має великий набір інструментів для роботи з анімацією та взаємодією, що дозволяє розробникам створювати динамічні та привабливі інтерфейси. Крім того, UIKit забезпечує велику кількість можливостей для обробки взаємодії користувача з застосунками, включаючи роботу з жестами, вводом даних та іншими.

Однак, UIKit має і деякі недоліки. Наприклад, розробка більш складних інтерфейсів може вимагати більш складних та продуманих рішень, що може призвести до складнішого коду та менш ефективного використання ресурсів. Крім того, в деяких випадках розробка на UIKit може бути несумісною з іншими платформами або технологіями.

Загалом, UIKit є надійним та потужним фреймворком для розробки інтерфейсів користувача на платформах Apple. Він дозволяє розробникам легко створювати привабливі та динамічні інтерфейси, що полегшує роботу з застосунками та забезпечує високий рівень користувацького досвіду. У разі правильного використання та належної оптимізації, UIKit може бути дуже

ефективним та продуктивним фреймворком для розробки застосунків на платформах Apple.

Vision

Vision - це фреймворк для комп'ютерного зору, розроблений компанією Apple. Він надає широкий спектр інструментів і функцій для роботи з обробкою зображень та комп'ютерним зором на платформі iOS та macOS [27].

Основні компоненти Vision:

- VNImageRequestHandler: Це клас, який використовується для обробки зображень та виконання запитів Vision. Він приймає зображення у форматі CVPixelBuffer або UIImage та дозволяє виконувати різні запити Vision на цих зображеннях.
- VNRequest: Це базовий клас для всіх запитів Vision. Він представляє конкретну операцію, яку ви хочете виконати з використанням Vision, таку як розпізнавання об'єктів, визначення облич, виявлення контуру тощо. Існують різні види запитів, такі як VNCoreMLRequest, VNDetectFaceRectanglesRequest, VNDetectTextRectanglesRequest тощо.
- VNResult: Це об'єкт, який представляє результат виконання запиту Vision. Він містить інформацію про виявлені об'єкти, облича, контури тощо. Результати можуть бути відображені, оброблені або використані для подальшого аналізу в додатку.
- VNSequenceRequestHandler: Це спеціальний обробник запитів, який дозволяє обробляти послідовності зображень, наприклад, відеопотоки. Він автоматично керує обробкою кадрів і підтримує виконання запитів Vision на кожному кадрі послідовності.

Основні можливості Vision:

- Розпізнавання об'єктів: за допомогою Vision ви можете розпізнавати об'єкти на зображеннях. Ви можете використовувати вбудовані моделі ML, такі як модель VGG16 або ResNet50, або навіть імпортувати власні моделі ML, створені з використанням фреймворків, таких як Core ML або

TensorFlow. Vision надає простий спосіб використання цих моделей для виявлення та класифікації об'єктів на зображенні.

- Виявлення облич: Vision має вбудовані функції для виявлення та аналізу облич. Ви можете використовувати Vision для знаходження облич на зображенні, визначення основних розмірів облич, виявлення особливих ознак, таких як очі, ніс, рот та інші.
- Виявлення контуру: Vision надає можливість виявляти контури об'єктів на зображенні. Ви можете використовувати цю функцію для виділення та аналізу контурів об'єктів, таких як облича, руки, тіла тощо. Це може бути корисно для розпізнавання жестів або визначення форми об'єктів.
- Виявлення тексту: Vision дозволяє виявляти та аналізувати текст на зображенні. Ви можете використовувати Vision для визначення розташування та витягування тексту з зображення, а також для розпізнавання символів і слів.
- Аналіз кольору та освітлення: за допомогою Vision ви можете аналізувати кольорову палітру та освітлення на зображенні. Це може бути корисно для визначення атмосфери або настрою зображення.

3.4 Архітектура MVC

MVC - є однією з найпоширеніших архітектурних парадигм для розробки програмного забезпечення. MVC розділяє програму на три головні компоненти: Модель, Представлення та Контролер, кожен з яких відповідає за власні функції та задачі [28].

Взаємодія трьох головних компонентів архітектури MVC: Контролера, Представлення та Моделі наведена нижче (див. рис. 3.1). Контролер зберігає стан користувача та обробляє події користувача. Представлення відображає дані для користувача та генерує події, що пов'язані з користувацьким інтерфейсом. Модель забезпечує доступ до даних та зберігає дані, які необхідні для роботи програми.

MVC дозволяє відокремлювати різні відповідальності та завдання, що дозволяє більш ефективно розробляти та підтримувати програмне забезпечення (див. рис. 3.1).

Модель - це компонент, який відповідає за управління даними та бізнес-логікою програми. Модель забезпечує доступ до даних та обробляє їх, щоб забезпечити коректну роботу програми. Модель не повинна залежати від Представлення та Контролера, а має бути незалежною компонентою програми [28].

Рисунок 3.1 - Прикладна схема архітектури MVC

Представлення - це компонент, який відповідає за представлення даних для користувача. Це можуть бути елементи інтерфейсу користувача, такі як кнопки, текстові поля, списки та інші елементи. Представлення відображає дані з Моделі та взаємодіє з Контролером для обробки користувацьких дій.

Контролер - це компонент, який відповідає за обробку користувацьких дій та взаємодію з Моделлю та Представленням. Контролер отримує вхідні дані від користувача та обробляє їх для забезпечення коректної роботи програми. Контролер також забезпечує зв'язок між Моделлю та Представленням та керує взаємодією між ними [28].

Основною перевагою архітектури MVC є те, що вона забезпечує відокремлення зв’язків між різними компонентами програми. Це полегшує

розробку та підтримку коду, оскільки кожен компонент виконує свої власні функції.

Також, MVC дозволяє розробникам змінювати один компонент програми без необхідності зміни інших компонентів [28]. Наприклад, можна змінити спосіб відображення даних без впливу на бізнес-логіку програми або змінити спосіб роботи з даними без впливу на спосіб їх відображення.

MVC також дозволяє розробникам розподілити роботу між декількома людьми або командами. Кожна команда може працювати над своїм компонентом програми, не втручаючись в роботу інших команд. Це дозволяє збільшити продуктивність та зменшити час розробки.

Однак, MVC має також деякі недоліки. Наприклад, якщо компоненти програми дуже сильно залежать один від одного, то може бути дуже складно розробляти та підтримувати програму. Також, якщо розмір програми зростає, то може бути дуже складно керувати взаємодією між різними компонентами.

У загальному, архітектура MVC є ефективним інструментом для розробки програмного забезпечення, особливо для веб-застосунків та мобільних застосунків. Вона дозволяє розробникам забезпечувати відокремлення зв’язків між різними компонентами програми, що сприяє підвищенню продуктивності та зменшенню часу розробки [29].

3.5 Алгоритм YOLO

YOLO є однією з ефективних архітектур для об'єктного визначення та локалізації на зображеннях та відео [30]. Його особливістю є висока швидкість та точність роботи, завдяки використанню CNN і побудові прогнозів у режимі реального часу.

Алгоритм YOLO використовує одну нейронну мережу для прогнозування об'єктів з різних класів та їхніх прямокутних рамок на зображенні. Він розбиває зображення на сітку, в якій кожна комірка відповідає певній області зображення. Для кожної комірки модель прогнозує клас об'єкта та координати його прямокутної рамки. Це дозволяє YOLO виявляти та локалізувати об'єкти швидше, ніж багатошарові методи.

Однією з особливостей YOLO є його здатність працювати з різними типами об'єктів та розпізнавати їх на зображенні навіть у відсутності попереднього навчання для конкретних класів. Також існують різні версії YOLO, такі як YOLOv3, YOLOv4 та YOLOv5, які покращують точність і швидкість роботи алгоритму.

YOLO [30] базується на ідеї сегментації зображення на менші зображення. Зображення розбивається на квадратну сітку розмірами S × S. Кожна клітинка сітки передбачає лише один об’єкт.

Для кожної комірки сітки:

- прогнозує межі B, і кожна коробка має один бал достовірності,
- виявляє лише один об'єкт незалежно від кількості ящиків B,
- передбачає ймовірності умовного класу C (по одній на клас для ймовірності класу об’єктів).

Кожен граничний блок містить 5 елементів: (x, y, w, h) і оцінку достовірності квадрата. Оцінка достовірності відображає, наскільки вірогідно рамка містить об’єкт (об’єктність) і наскільки точним є межова рамка. Нормалізуємо ширину обмежувальної рамки w і висоту h шириною і висотою зображення. x і y є зсувами до відповідної комірки. Отже, x, y, w і h знаходяться між 0 і 1. Кожна комірка має 20 умовних класів ймовірностей. Імовірність умовного класу – це ймовірність того, що виявлений об’єкт належить до певного класу (одна ймовірність на категорію для кожної комірки). Отже, передбачення YOLO має вигляд (S, S, B×5 + C) = (7, 7, 2×5 + 20) = (7, 7, 30) (див. рис. 3.2).

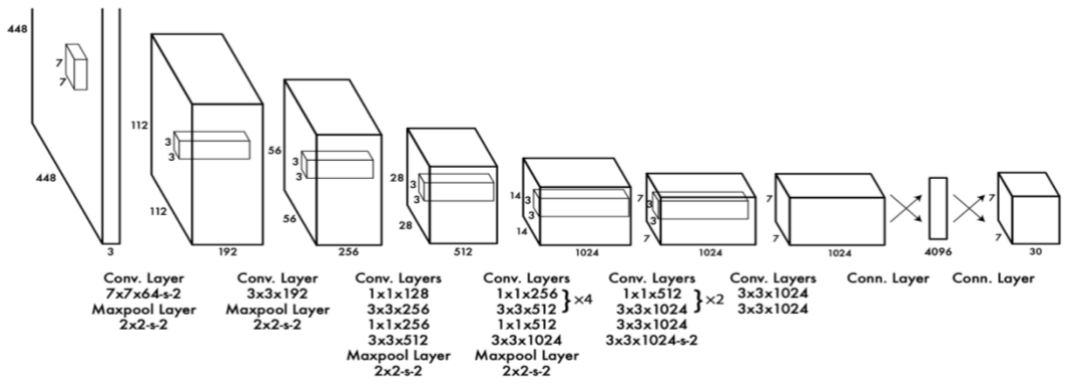


Рисунок 3.2 - Архітектура YOLO

Функція втрат

YOLO передбачає кілька обмежувальних рамок на клітинку сітки. Щоб обчислити збитки для справжнього позитиву, потрібно, щоб лише один із них відповідав за об'єкт [30]. Для цього вибирається той, який має найвищий IoU (перетин над об'єднанням) із основною правдою. Ця стратегія призводить до спеціалізації серед передбачень обмежувальної рамки. Кожне передбачення покращує прогнозування певних розмірів і пропорцій.

YOLO використовує квадратичну похибку між прогнозами та основною правдою для розрахунку втрат. Функція втрат складається з:

- Втрати класифікації.

Якщо об'єкт виявлено, втрата класифікації в кожній комірці є квадратом помилки умовних ймовірностей класу для кожного класу:

$$\sum_{i=0}^{S^2} \mathbb{I}_i^{obj} \sum_{c \in classes} (p_i(c) - \widehat{p_i}(c))^2,$$

де

$\mathbb{I}_i^{obj} = 1$, якщо об'єкт з'являється в клітинці i, інакше 0

$\widehat{p_i}(c)$, позначає умовну ймовірність класу для класу c у клітинці i

- Втрати локалізації (помилки між передбачуваним граничним блоком і правдою землі).

Втрата локалізації вимірює помилки в прогнозованому розташуванні та розмірі граничної рамки. Враховується лише коробку, яка відповідає за виявлення об'єкта.

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{obj} [(x_i - \widehat{x_i})^2 + (y_i - \widehat{y_i})^2]$$

$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\widehat{w_i}})^2 + (\sqrt{h_i} - \sqrt{\widehat{h_i}})^2 \right],$$

де

$\mathbb{I}_{ij}^{obj} = 1$, якщо j у рамці клітинки i відповідає за виявлення об'єкта, інакше 0

λ_{coord}, збільшує вагу втрати в координатах граничної рамки

Абсолютні похибки не зважуються однаково у великих і малих коробках. Тобто помилка в 2 пікселя у великій коробці є однаковою і для маленької коробки. Щоб частково вирішити цю проблему, YOLO передбачає квадратний корінь із ширини та висоти обмежувальної рамки замість ширини та висоти. Крім того, щоб приділити більше уваги точності граничної рамки, множаться втрати на λcoord (за замовчуванням: 5).

- Втрати впевненості (об'єктивність ящика).

Якщо в коробці виявлено об’єкт, втрата впевненості (вимірювання об’єктності коробки) становить:

$$\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{I}_{ij}^{obj}(\mathrm{C}_i-\widehat{\mathrm{C}_\iota})^2,$$

де

$\widehat{\mathrm{C}_\iota}$, це бал достовірності поля j у комірці i

$\mathbb{I}_{ij}^{obj}=1$, якщо j у рамці клітинки i відповідає за виявлення об’єкта, інакше 0

Якщо об’єкт не виявлено в коробці, втрата впевненості становить:

$$\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{I}_{ij}^{noobj}(\mathrm{C}_i-\widehat{\mathrm{C}_\iota})^2,$$

де

$\mathbb{I}_{ij}^{noobj}$, є доповненням до $\mathbb{I}_{ij}^{obj}$

$\widehat{\mathrm{C}_\iota}$, це бал достовірності поля j у комірці i

λ_{noobj}, зменшує втрати під час виявлення фону

Більшість ящиків не містять жодних предметів. Це викликає проблему дисбалансу класів, тобто мета навчити модель виявляти фон частіше, ніж виявляти об’єкти.

Остаточна втрата додає разом втрати локалізації, впевненості та класифікації.

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$

$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{obj} (\mathrm{C}_i - \hat{\mathrm{C}}_i)^2$$

$$+ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{noobj} (\mathrm{C}_i - \hat{\mathrm{C}}_i)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{I}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

Переваги YOLO

- Швидкість. Добре підходить для обробки в реальному часі.
- Прогнози (розташування об'єктів і класи) робляться з однієї мережі. Можна тренувати наскрізне для підвищення точності.
- YOLO є більш узагальненим. Він перевершує інші методи при узагальненні природних зображень в інших сферах, наприклад творах мистецтва.
- Методи пропозиції регіону обмежують класифікатор конкретним регіоном. YOLO отримує доступ до всього зображення, прогнозуючи межі. Завдяки додатковому контексту YOLO демонструє менше помилкових спрацьовувань у фонових областях.
- YOLO виявляє один об’єкт на клітинку сітки. Він забезпечує просторове розмаїття під час прогнозування.

У проєкті використовувалась 5 версія YOLO

Розміри моделі

YOLOv5 поставляється в 4 розмірах (s, m, l і xl) [30]. Інтуїтивно зрозуміло, що чим більша мережа, тим більше налаштовуваних параметрів, тим краща продуктивність. Але знову ж таки, більше параметрів означає

більший час навчання та час висновку. Якщо створюється система для виявлення в реальному часі, то однозначно потрібно віддати перевагу моделі малого або середнього розміру. Нижче ви можна побачити огляд чотирьох розмірів моделей (див. рис. 3.3).

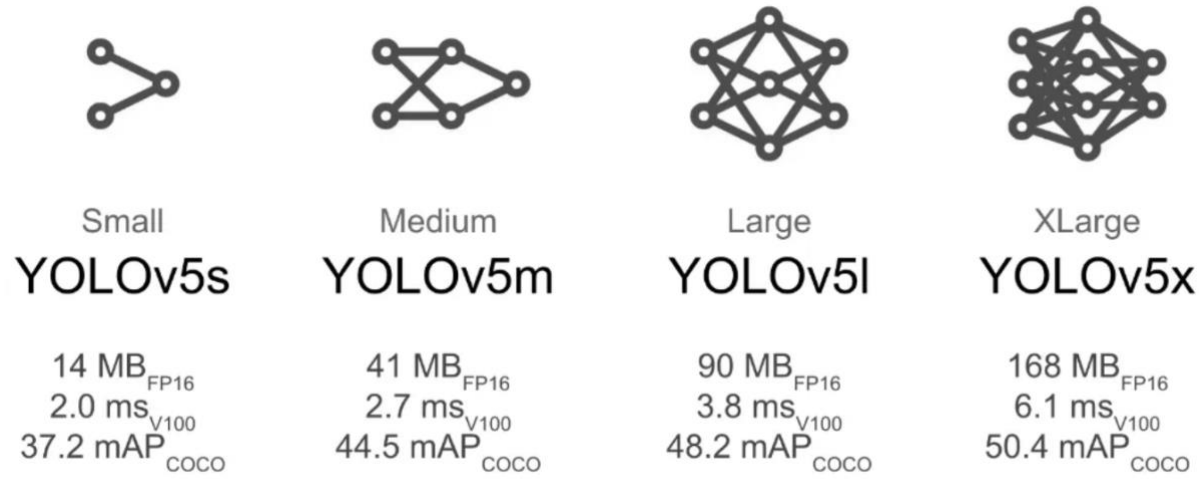


Рисунок 3.3 - Розміри YOLOv5

РОЗДІЛ 4. РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ

4.1 Вимоги до мобільного застосунку

Вимоги до застосунку базувались на вивченні потреб цільової аудиторії - людей з фізичними та сенсорними порушеннями, а також на загальних цілях проєкту. Основні вимоги включають:

1. Розпізнавання об'єктів: здатність знайти об'єкти у реальному часі за допомогою камери на пристрої. Це дозволяє користувачам з фізичними обмеженнями отримувати інформацію про предмети, які знаходяться перед ними, і сприяє їхній незалежності та самостійності.
2. Голосове повідомлення: можливість надавати повідомлення за допомогою звуку з інформацією про розпізнаний об'єкт або розташування. Це допомагає користувачам отримувати доступ до інформації, навіть якщо вони мають обмеження зору або не можуть прочитати текстову інформацію на екрані.
3. Інтуїтивний інтерфейс: простий та зрозумілий інтерфейс користувача, щоб забезпечити зручну навігацію та використання. Це особливо важливо для користувачів з обмеженими фізичними можливостями, які можуть мати складнощі з моторикою або сприйняттям інтерфейсу.
4. Підтримка доступності: застосунок має бути доступним для користувачів з різними рівнями фізичних обмежень. Це включає можливість налаштування розміру шрифту, контрастності та звуку, адаптивний дизайн, підтримку сенсорних вводів та можливість роботи зі спеціальними пристроями допомоги.
5. Інтеграція з голосовими асистентами: здатність підтримувати інтеграцію з популярними голосовими асистентами, такими як Siri або Google Assistant. Це дозволяє користувачам з фізичними порушеннями керувати застосунком голосом і виконувати різні команди за допомогою голосових інструкцій.

6. Підтримка мов: гнучка підтримка мов, щоб відповідати потребам користувачів з різних країн та культур. Це забезпечує більш широкий коло користувачів і полегшує їхнє використання застосунку.
7. Робота офлайн: використання без підключення до Інтернету. Це важливо для користувачів, які можуть перебувати в областях з обмеженим або відсутнім доступом до мережі. Робота офлайн дозволяє користувачам використовувати основні функціональні можливості застосунку, такі як розпізнавання об'єктів і отримання голосової інформації, незалежно від наявності Інтернет-підключення.

Основні функціональні вимоги застосунку:

1. Розпізнавання перешкод: можливість розпізнавати різні перепони на шляху користувача, такі як вуличні об'єкти, автомобілі, люди тощо.
2. Візуальне та аудіальне попередження: застосунок аудіально та візуально має повідомляти користувача про виявлені перешкоди, небезпеку або зміни в середовищі.
3. Навігація та маршрут: містить інструкції для користувача щодо навігації, вказувати оптимальний маршрут та допомагати користувачеві дістатися до пункту призначення.
4. Підтримка голосових команд: працювати з аудіо командами користувача для виконання певних дій або запитів.

4.2 Реалізація застосунку

Реалізація мобільного застосунку базувалась на використанні мови програмування Swift та інтеграції з бібліотеками ML та Vision.

Основна логіка застосунку в розпізнаванні образів за допомогою комп'ютерного зору та аналізі зображення, яке потім передається до моделі ML для виявлення предмета. Далі, в залежності від результатів аналізу, генерується текстове сповіщення та голосовий коментар про предмет, який було розпізнано. Крім того, в додатку було реалізовано функцію запису голосових нотаток та їх зберігання на пристрої користувача.

Експорт моделі виявлення

На початку роботи нам потрібно буде створити модель YOLOv5 Core ML, перш ніж ми зможемо запустити програму. Для цього слід клонувати репозиторій YOLOv5 з GitHub за допомогою команди:

```
git clone https://github.com/ultralytics/yolov5.git
```

Щоб отримати файли для YOLOv5s, використана така команда:

```
cd yolov5
git checkout tags/v5.0
```

Щоб завантажити вагові файли для YOLOv5s у папку yolov5/weights, потрібно виконати наступну команду:

```
bash weights/download_weights.sh
```

Для створення експортованої моделі нам знадобиться python ≥3.7 і встановити залежності з кореня репозиторію за допомогою:

Для конвертації моделі YOLOv5s у формат Core ML нам знадобляться наступні вимоги:

- Встановлений Python на комп'ютері
- Встановлена бібліотека PyTorch для роботи з моделлю YOLOv5s.
- Встановлена бібліотека CoreMLTools, яка дозволить конвертувати модель YOLOv5s у формат Core ML.

```
pip install -r requirements.txt -r equirements-export.txt
```

Після успішного встановлення вимог фактичний експорт можна запустити за допомогою:

```
python export.py –weights yolov5s.pt –img 480 --batch 1
```

У цій команді --weights yolov5s.pt вказує шлях до вагових файлів YOLOv5s, а --img 480 встановлює розмір вхідного зображення для конвертації. Можна вказати інший розмір зображення. Модель YOLOv5s буде конвертована в формат Core ML та збережена у файлі з розширенням .mlmodel.

Імпорт моделі в проєкт

Після отримання файлу yolov5s.mlmodel можемо використовувати його у застосунку. Для цього потрібно додати файл yolov5s.mlmodel до проєкту, перетягніть файл yolov5s.mlmodel у папку проєкту (див. рис. 4.1).

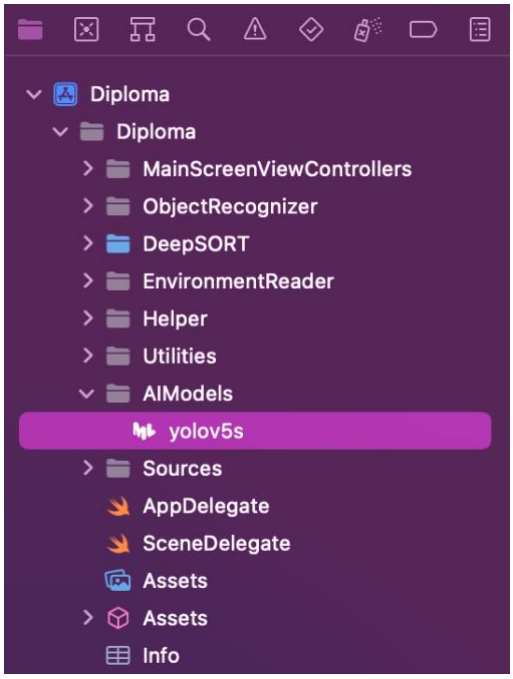


Рисунок 4.1 - Розміщений файл в проєкті

Далі на наступному екрані, можна підтвердити додавання файлу, де важливо не забувати включити файл моделі до таргетів (“Add to targets”) (див. рис. 4.2).

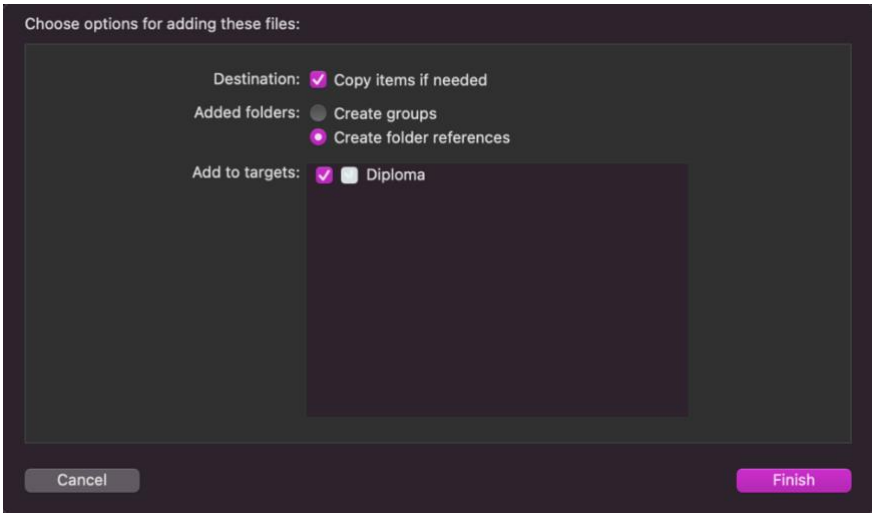


Рисунок 4.2 - Діалогове вікно додавання файлу yolov5s.mlmodel

Використання моделі

В Swift-файлі, де використовуємо модель, імпортуємо Vision та Core ML фреймворки:

```
import Vision
import CoreML
```

Створюємо клас ObjectDetectionViewController:

```
class ObjectDetectionViewController: UIViewController { }
```

Оголошуємо змінну для моделі та для запиту Vision:

```
var yolov5sModel: YOLOv5sModelClass!
var request: VNCoreMLRequest!
```

YOLOv5sModelClass є назвою класу, який буде згенерований автоматично при конвертації моделі YOLOv5s у формат Core ML. При конвертації, Core ML Tools створює клас, який представляє модель у коді. клас YOLOv5sModelClass буде мати такі основні властивості та методи:

- init(contentsOf: URL): Це ініціалізатор класу, який дозволяє створити екземпляр моделі YOLOv5s з файлу mlmodel. Він отримує URL файлу mlmodel як параметр.
- func prediction(input: YOLOv5sModelInput) throws -> YOLOv5sModelOutput: Цей метод приймає вхідні дані типу YOLOv5sModelInput і повертає результат типу YOLOv5sModelOutput. Вхідні дані представляють вхідні параметри моделі, такі як зображення, яке потрібно аналізувати. Результат містить прогнозовані значення моделі, такі як виявлені об'єкти, їх класи та координати.

Завантажуємо модель у форматі Core ML:

```
}
    \\ Обробка помилки
} catch {
    yolov5sModel = try YOLOv5sModelClass(configuration: modelURL)
do {
}
    return
    \\ Обробка помилки
                                    withExtension: "mlmodelc") else {
guard let modelURL = Bundle.main.url(forResource: "yolov5s",
```

"mlmodelc" - це компільована версія моделі YOLOv5s.

Після завантаження моделі, створюємо Vision request з використанням моделі:

```
do {
    request = try VNCoreMLRequest(model: yolov5sModel)
} catch {
    // Обробка помилки
}
request.imageCropAndScaleOption = .scaleFill
```

Vision request - це об'єкт, який представляє запит до фреймворку Vision для обробки зображень або відео [27]. Він визначає, яку операцію або аналіз потрібно виконати на вхідних даних.

В Vision є кілька типів запитів, які можна використовувати залежно від потреб:

- VNCoreMLRequest: запит дозволяє використовувати модель ML у форматі Core ML для класифікації, виявлення об'єктів, сегментації та інших завдань комп'ютерного зору.
- VNRecognizeTextRequest: запит використовується для розпізнавання тексту на зображенні або в потоці відео.
- VNDetectFaceRectanglesRequest: запит дозволяє виявляти обличчя на зображенні або в потоці відео.
- VNDetectBarcodesRequest: запит використовується для виявлення штрих-кодів на зображенні або в потоці відео.
- VNDetectTextRectanglesRequest: запит дозволяє виявляти прямокутники, які обмежують області тексту на зображенні або в потоці відео.
- Інші типи запитів: Vision також має інші типи запитів для інших завдань, таких як виявлення країв, вимірювання об'єктів, визначення орієнтації та інше.

Ми використовуємо VNCoreMLReques. Опція imageCropAndScaleOption для request, вказана, для надання інструкції як обробляти зображення.

VNCoreMLRequest використовує модель Core ML для аналізу зображення та виявлення об'єктів. При використанні VNCoreMLRequest, вхідні та вихідні дані такі [31]:

- Вхідні дані: зображення у форматі CVPixelBuffer або CIImage. Це вхідне зображення, яке буде аналізоване моделлю. VNCoreMLRequest може працювати з CVPixelBuffer, який містить пікселі зображення, або з CIImage, який представляє зображення у форматі Core Image.
- Вихідні дані: результати аналізу зображення у форматі VNRequestObservation. Конкретний тип спостереження залежить від типу моделі, яка використовується. Для задачі виявлення об'єктів, VNCoreMLRequest повертає спостереження типу VNRecognizedObjectObservation, яке містить інформацію про виявлені об'єкти на зображенні.

Звернемось до методу, де отримуємо вхідне зображення, і виконаємо наступні дії:

1. Створюємо CVPixelBuffer з отриманого зображення:

```
guard let pixelBuffer = CMSampleBufferGetImageBuffer(sampleBuffer) else
{
    // Обробка помилки
    return
}
```

CVPixelBuffer є типом даних у фреймворку Core Video, що представляє буфер зображення, який містить пікселі зображення в пам'яті. Він широко використовується для обробки та обміну зображеннями між різними фреймворками. Основні властивості CVPixelBuffer [32]:

- Формат зображення: CVPixelBuffer підтримує різні формати зображень, такі як RGBA, BGRA, YUV та інші. Формат зображення залежить від контексту його використання та джерела даних.
- Розмір та розташування пікселів: CVPixelBuffer містить розмір та розташування пікселів зображення. МИ можемо отримати ширину,

висоту та кількість каналів кольору (наприклад, RGB або RGBA) зображення з допомогою властивостей CVPixelBuffer.

- Доступ до даних пікселів: CVPixelBuffer надає можливість отримувати прямий доступ до даних пікселів зображення. Ми можемо отримати адресу пам'яті, яка містить дані пікселів, та прочитати або записати значення пікселів безпосередньо.
- Інтеграція з іншими фреймворками: CVPixelBuffer є типом даних, який широко використовується в різних фреймворках, таких як Vision, Core ML, AVFoundation та інші.

В контексті використання Vision та Core ML, CVPixelBuffer використовується для передачі вхідних зображень моделі для аналізу. Шляхи отримання CVPixelBuffer залежать від джерела даних, таких як зображення з камери або відеопотік.

2. Створюємо VNImageRequestHandler з CVPixelBuffer:

```
let imageRequestHandler = VNImageRequestHandler(cvPixelBuffer: pixelBuffer,
                                                orientation: .up)
```

VNImageRequestHandler є класом у фреймворку Vision, який дозволяє обробляти зображення або послідовність кадрів у форматі CVPixelBuffer або CIImage з використанням Vision запитів [27]. Він спрощує передачу вхідних зображень або відео для обробки фреймворку Vision.

Основні функції VNImageRequestHandler:

- Ініціалізація: VNImageRequestHandler може бути ініціалізований з вхідним зображенням або послідовністю кадрів у форматі CVPixelBuffer або CIImage;
- Виконання Vision запитів: після ініціалізації VNImageRequestHandler можна виконати один або кілька Vision запитів, передавши їх у метод perform;
- Опції обробки: VNImageRequestHandler дозволяє вказати додаткові опції обробки зображень, передаючи їх у метод init або метод perform. Можна вказати орієнтацію зображення, як відображаються вхідні

зображення, чи повинна виконуватися передача даних фреймів фоновим процесом та інші опції.

- Обробка результатів: результати обробки Vision запитів можна обробити у замиканні, як переданий у метод perform. У замиканні можна отримати та обробити результати запитів

В додатку А наведені фрагменти коду використання моделі.

Отримання та передача зображення з відеопотоку

Для захоплення кадрів з відеопотоку можна використовувати фреймворк AVFoundation.

```
import AVFoundation
```

Використовуючи AVCaptureSession, можна налаштувати захоплення відеопотоку з камери або іншого джерела. Приклад коду для налаштування захоплення відеопотоку з камери:

```
// Створення AVCaptureSession
let captureSession = AVCaptureSession()
// Вибір вхідного пристрою (камери)
guard let captureDevice = AVCaptureDevice.default(for: .video) else {
    // Обробка помилки
    return
}
// Створення AVCaptureDeviceInput з вибраним пристроєм
guard let input = try? AVCaptureDeviceInput(device: captureDevice) else {
    // Обробка помилки
    return
}
// Додавання вхідного пристрою до captureSession
captureSession.addInput(input)
// Створення AVCaptureVideoDataOutput для отримання кадрів відеопотоку
let videoOutput = AVCaptureVideoDataOutput()
videoOutput.setSampleBufferDelegate(self,
                        queue: DispatchQueue(label: "VideoOutputQueue"))
captureSession.addOutput(videoOutput)
// Запуск captureSession
captureSession.startRunning()
```

У цьому прикладі створюємо AVCaptureSession та встановлюємо вхідний пристрій (камеру) як вхід для session. Також ми створюємо AVCaptureVideoDataOutput для отримання кадрів відеопотоку та встановлюємо його делегатом з використанням черги DispatchQueue для обробки кадрів.

AVCaptureSession - це клас у фреймворку AVFoundation, який використовується для керування захопленням медіаданих (відео, аудіо, метаданих) з різних джерел, таких як камера, мікрофон або інші зовнішні пристрої [33].

Основні можливості AVCaptureSession включають:

- Вибір та налаштування вхідних та вихідних пристроїв: AVCaptureSession дозволяє вибирати та налаштовувати вхідні пристрої, такі як камери або мікрофони, а також вихідні пристрої, такі як запис на диск або передача потоку.
- Управління життєвим циклом захоплення: AVCaptureSession забезпечує методи для початку, зупинки та паузи захоплення медіаданих. Є можливість контролювати життєвий цикл сесії в залежності від потреб застосунку.
- Налаштування властивостей захоплення: є можливість налаштувати різні параметри захоплення, такі як роздільна здатність, частота кадрів, якість стиснення та інші параметри вхідних та вихідних пристроїв.
- Потокова передача даних: AVCaptureSession дозволяє передавати захоплені медіадані в реальному часі, так що ми можемо використовувати їх для відображення на екрані, обробки або передачі по мережі.

AVCaptureDevice є класом у фреймворку AVFoundation, який представляє фізичний пристрій захоплення, такий як камера або мікрофон, який може бути використаний для захоплення відео, аудіо або інших медіаданих.

AVCaptureDeviceInput - це клас, який використовується для підключення інстанції AVCaptureDevice до AVCaptureSession як вхідного пристрою. Він інкапсулює вхідний пристрій захоплення (наприклад, камеру) та надає його як вхідний сигнал для AVCaptureSession.

AVCaptureVideoDataOutput є класом у фреймворку AVFoundation, який дозволяє отримувати вихідні дані відео з AVCaptureSession у вигляді

CMSampleBuffer. Він представляється як вихідний об'єкт для захоплення відео з камери або іншого джерела та передачі його для подальшої обробки.

Основні можливості AVCaptureVideoDataOutput включають:

- Отримання вихідних даних відео: AVCaptureVideoDataOutput видає вихідні дані відео в форматі CMSampleBuffer, який містить кадр відеопотоку разом з додатковими метаданими.
- Налаштування формату вихідних даних: є можливість налаштувати формат вихідних даних, такі як роздільність, формат пікселів, компресія та інші параметри.
- Потокова передача даних: є можливість передавати вихідні дані в реальному часі для відображення на екрані, обробки або передачі по мережі.

Реалізація методів делегата AVCaptureVideoDataOutputSampleBufferDelegate: extension ViewController:

```
extension ViewController: AVCaptureVideoDataOutputSampleBufferDelegate {
    func captureOutput(_ output: AVCaptureOutput,
                       didOutput sampleBuffer: CMSampleBuffer,
                       from connection: AVCaptureConnection) {
        // Отримання CVPixelBuffer з CMSampleBuffer
        guard let pixelBuffer = CMSampleBufferGetImageBuffer(sampleBuffer)
        else {
            // Обробка помилки
            return
        }
        // Використання CVPixelBuffer для подальшої обробки або передачі в
           модель
        // ...
    }
}
```

У цьому методі, можна обробляти вихідні дані відео у форматі CMSampleBuffer, отримувати доступ до кадрів в відеопотоку та виконувати потрібні дії з ними. Наприкладі продемонстровано як передавати ці кадри в модель для аналізу або відображати їх на екрані.

Фрагмент реалізації підпункту продемонстровано у додатку Б.

Обробка результату

Для обробки кожного кадру відеопотоку, потрібно передати його у VNImageRequestHandler для виконання Vision запиту:

```
// Створення VNImageRequestHandler з CVPixelBuffer
    let imageRequestHandler =
    VNImageRequestHandler(cvPixelBuffer:pixelBuffer,
                          options: [:])

     do {
         // Виконання Vision запиту
         try imageRequestHandler.perform([request])
     } catch {
         // Обробка помилки
     }
```

VNImageRequestHandler - це клас у фреймворку Vision, який дозволяє виконувати різні запити на обробку зображень з використанням моделей Core ML [27]. Основна функція VNImageRequestHandler - це передача зображень для обробки моделлю Core ML та отримання результатів аналізу.

Основні властивості та методи VNImageRequestHandler:

1. Ініціалізація:

- init(cgImage:options:): Ініціалізує об'єкт VNImageRequestHandler з об'єктом CGImage та додатковими опціями.
- init(ciImage:options:): Ініціалізує об'єкт VNImageRequestHandler з об'єктом CIImage та додатковими опціями.
- init(cvPixelBuffer:options:): Ініціалізує об'єкт VNImageRequestHandler з об'єктом CVPixelBuffer та додатковими опціями.

2. Виконання запитів:

- perform(_:): Виконує запити Vision на обробку зображень. Приймає масив об'єктів VNRequest, які представляють запити на аналіз зображень.

3. Додаткові опції:

- options: Дозволяє встановлювати додаткові параметри для обробки зображень, такі як орієнтація, масштабування, область обрізання тощо.

У блоку замикання запиту (request.completionHandler), можна отримати та обробити результати розпізнавання об'єктів для кожного кадру відеопотоку:

```
if let observations = request.results as? [VNRecognizedObjectObservation]
{
    // Обробка результатів для кожного об'єкту
    for observation in observations {
        let boundingBox = observation.boundingBox
        let label = observation.labels.first?.identifier
        let confidence = observation.confidence
        // ...
    }
}
```

Ми отримуємо спостереження (VNRecognizedObjectObservation) - клас у фреймворку Vision, який містить інформацію про виявлені об'єкти на кадрі відеопотоку [27]. Ми можемо отримати розмір рамки (boundingBox), мітку (label) та впевненість (confidence) для кожного виявленого об'єкта:

- BoundingBox: координати рамки, яка виокремлює виявлений об'єкт на зображенні.
- Labels: список міток (labels) та ймовірностей (confidence) для виявлених об'єктів. Кожна мітка ідентифікує клас об'єкта, а ймовірність вказує на впевненість моделі щодо відповідності зображення до цього класу.

Для обробки результатів VNRecognizedObjectObservation та відображення рамки та назви об'єкта на екрані нам знадобиться відповідний фреймворк для роботи з графікою, такий як UIKit.

Наступним етапом обробки є проходження через кожне VNRecognizedObjectObservation та виконання потрібних дій (зображення рамки та мітки):

```
for observation in results {
    let boundingBox = observation.boundingBox
    let labels = observation.labels
    // Конвертація boundingBox у координати екрану
    let transformedRect =
    VNImageRectForNormalizedRect(boundingBox,
                                 Int(imageView.bounds.width),
                                 Int(imageView.bounds.height))
    // Відображення рамки на екрані
    let boundingBoxView = UIView(frame: transformedRect)
    boundingBoxView.layer.borderWidth = 2.0
    boundingBoxView.layer.borderColor = UIColor.red.cgColor
    imageView.addSubview(boundingBoxView)
    // Відображення назви об'єкта на екрані
    if let firstLabel = labels.first {
        let confidence = Int(firstLabel.confidence * 100)

        let labelText = "\(firstLabel.identifier)
                        (\(confidence)% confidence)"
        let labelView =
            UILabel(frame: CGRect(x: transformedRect.origin.x,
                                  y: transformedRect.origin.y - 20,
                                  width: transformedRect.width,
                                  height: 20))
        labelView.text = labelText
        labelView.textColor = UIColor.red
        labelView.font = UIFont.systemFont(ofSize: 12)
        imageView.addSubview(labelView)
    }
}
```

Конвертуємо координати boundingBox в координати екрану. Створюємо UIView для відображення рамки і додаємо його на imageView. Відображаємо назву об'єкта та впевненість у відповідному UILabel і також додаємо його на imageView.

Фрагмент коду наведено у додатку В.

Додавання аудіосупроводу

Для застосування аудіосупроводу у застосунку задля проговорення розташування результату нам знадобиться використати TTS функціональність. У iOS є вбудована функція Speech Synthesis, яка дозволяє озвучувати текстові рядки.

TTS є технологією, яка перетворює текстові рядки на аудіо, що озвучується комп'ютером або іншим пристроєм [34]. Ця технологія дозволяє додати голосову функціональність до додатків та систем, щоб озвучувати текст для користувачів. Основні компоненти TTS:

- TTS Engine: програмний компонент, який виконує перетворення тексту на аудіо. Існують різні двигуни TTS, які можуть мати різні голоси, мови та параметри озвучування.
- Text Analyzer: компонент аналізує текстові рядки для виявлення фонетики, інтонації, розділових знаків та інших важливих елементів, щоб визначити правильну вимову та наголос.
- Speech Synthesizer: компонент, який генерує аудіо на основі тексту та інших параметрів озвучування, таких як швидкість, гучність, тон, емоційний вираз тощо.
- Voice Library: набір доступних голосів, які можуть бути використані для озвучування тексту. Кожен голос може мати свою мову, стиль та характеристики.

Основні кроки для додавання аудіосупроводу:

1. Імпортування фреймворку AVFoundation в файл, де планується використовувати TTS.

```
import AVFoundation
```

2. Створення об'єкту AVSpeechSynthesizer, який буде відповідати за озвучування тексту.

```
let synthesizer = AVSpeechSynthesizer()
```

AVSpeechSynthesizer є класом у фреймворку AVFoundation, який використовується для озвучування тексту TTS на iOS [35]. Він надає зручний спосіб озвучувати текстові рядки з налаштуваннями параметрів озвучування.

Основні властивості AVSpeechSynthesizer:

- delegate: об'єкт, який виступає в якості делегата AVSpeechSynthesizer і отримує повідомлення про стан озвучування.
- isSpeaking: вказує, чи програється в даний момент текст.

- isPaused: вказує, чи призупинено відтворення тексту.

Основні методи AVSpeechSynthesizer:

- speak(_ utterance: AVSpeechUtterance): Починає озвучування тексту, передаючи об'єкт AVSpeechUtterance.
- stopSpeaking(at: AVSpeechBoundary): Зупиняє озвучування тексту, опціонально досягаючи межі AVSpeechBoundary.
- pauseSpeaking(at: AVSpeechBoundary): Призупиняє тимчасово озвучування тексту, опціонально досягаючи межі AVSpeechBoundary.
- continueSpeaking(): Продовжує озвучування тексту після призупинення.

3. Створення текстового рядка, який буде містити розташування результату

```
let locationText = "Розташування: \(location)"
```

4. Створення об'єкту AVSpeechUtterance, який буде містити текст, який треба озвучити.

```
let speechUtterance = AVSpeechUtterance(string: locationText)
```

AVSpeechUtterance є класом у фреймворку AVFoundation, який представляє текст, який потрібно озвучити (текстовий рядок) і містить параметри озвучування [36].

Основні властивості AVSpeechUtterance:

- speechString: текстовий рядок для озвучування.
- voice: голос, який використовується для озвучування тексту. Можна вибрати голос з наявних мов та налаштувань голосу.
- rate: швидкість озвучування тексту. Значення 1.0 відповідає нормальній швидкості, значення нижче 1.0 сповільнює озвучування, а значення більше 1.0 прискорює його.

- pitchMultiplier: показник висоти голосу. Значення 1.0 відповідає нормальній висоті голосу, значення менше 1.0 знижує висоту, а значення більше 1.0 підвищує її.
- volume: гучність озвучування тексту. Значення від 0.0 до 1.0, де 0.0 представляє беззвучний звук, а 1.0 - максимальну гучність.
- preUtteranceDelay: затримка перед початком озвучування тексту.
- postUtteranceDelay: затримка після завершення озвучування тексту.

5. Налаштування параметрів AVSpeechUtterance, які включають швидкість озвучування, голос та інші.

```
speechUtterance.rate = AVSpeechUtteranceDefaultSpeechRate
speechUtterance.voice = AVSpeechSynthesisVoice(language: "uk-UA") //
Вибір мови озвучування
```

6. Озвучування тексту за допомогою AVSpeechSynthesizer.

```
synthesizer.speak(speechUtterance)
```

Розроблений клас для аудіосупроводу наведено в додатку Г.

Ітерпретація отриманих результатів в формат тексту

Для інтерпретації результатів моделі YOLOv5s в текст для озучування кількості кроків до предмету та розташування у просторі відносно камери, нам необхідно знати відстань до предмету та позицію предмету у просторі.

Необхідні дані:

- Відстань до предмету: потрібно визначити кількість кроків до предмету, необхідно мати інформацію про відстань між камерою та предметом. Цю відстань можна виміряти, використовуючи різні методи, такі як відстань на основі глибинної карти або відстань на основі розмірів об'єкта на зображенні.
- Розташування у просторі: для опису розташування предмету у просторі відносно камери, потрібно знати позицію предмету у тривимірному просторі. Це можна досягти, використовуючи методи тривимірного виявлення об'єктів або використовуючи глибинну інформацію зображення. Ці дані можуть бути представлені у форматі координат XYZ або інших відповідних представленнях.

- Інтерпретація результатів: залежно від отриманих результатів і формату, можна інтерпретувати їх для генерації тексту, що описує кількість кроків та розташування. Наприклад, можна перетворити відстань у кількість кроків, використовуючи відомі параметри, такі як довжина кроку, або використовувати описові терміни, такі як "близько" або "далеко".

Якщо модель повертає координати XYZ для розташування предмету, то можна інтерпретувати ці координати наступним чином:

- Якщо координата X позитивна, можна сказати, що предмет знаходиться справа від камери, а якщо вона від'ємна, він знаходиться зліва.
- Якщо координата Y позитивна, предмет знаходиться вгорі, а якщо вона від'ємна, він знаходиться внизу.
- Якщо координата Z позитивна, предмет знаходиться попереду камери, а якщо вона від'ємна, він знаходиться позаду.

Комбінуючи ці терміни, щоб створити текстовий рядок, що описує положення предмету. Наприклад, "Предмет знаходиться праворуч та вгорі", "Предмет знаходиться попереду та нижче" тощо.

Приклад коду, який демонструє, як інтерпретувати отримані результати моделі YOLOv5s в текст для озучування кількості кроків до предмету та розташування у просторі відносно камери:

```
    let detections = // Отримані результати моделі YOLOv5s
    // Припустимо, що є значення довжини кроку і позиції камери у просторі
    let stepLength = 0.7 // Довжина кроку (наприклад, 0.7 метра)
    let cameraPosition = SIMD3<Float>(0, 0, 0) // Позиція камери (наприклад, в
центрі простору)
    for detection in detections {
        let classID = detection.classID
        let boundingBox = detection.boundingBox
        let objectLabel = classLabels[classID] ?? "Невідомий об'єкт"
        // Розраховуємо відстань до предмету від камери
        let objectDistance = calculateDistance(boundingBox, cameraPosition)
        // Розраховуємо кількість кроків до предмету
        let stepsToObject = Int(objectDistance / stepLength)
        // Формуємо текст для озучування
        let    speechText    =    "Підходьте    до    \(objectLabel).    Відстань
\(objectDistance) метрів, це приблизно \(stepsToObject) кроків."
    // Використовуємо AVSpeechSynthesizer для озучування тексту
        let speechUtterance = AVSpeechUtterance(string: speechText)
        speechSynthesizer.speak(speechUtterance)
    }
```

Функція calculatePosition використовується для визначення розташування об'єкту у просторі відносно камери на основі координат обмежувального прямокутника та позиції камери.

Основна ідея полягає в тому, що ми можемо визначити положення об'єкту відносно камери за допомогою різниці між центром об'єкту та позицією камери по осі X, Y та Z. Можна використовувати ці значення для визначення положення об'єкту відносно камери, наприклад, визначити, чи об'єкт знаходиться зліва або справа, вгорі або внизу.

Приклад коду, який демонструє, як визначити положення об'єкту у просторі відносно камери:

```
func calculatePosition(_ boundingBox: CGRect,
                       _ cameraPosition: SIMD3<Float>) -> String {
    let objectCenterX = boundingBox.origin.x + boundingBox.width / 2
    let objectCenterY = boundingBox.origin.y + boundingBox.height / 2

    let objectPositionX = objectCenterX - cameraPosition.x
    let objectPositionY = objectCenterY - cameraPosition.y

    var positionDescription = ""

    if objectPositionX < 0 {
        positionDescription += "ліворуч"
    } else if objectPositionX > 0 {
        positionDescription += "справа"
    }

    if objectPositionY < 0 {
        positionDescription += " та нижче"
    } else if objectPositionY > 0 {
        positionDescription += " та вгорі"
    }

    return positionDescription
}
```

У цьому прикладі функція calculatePosition використовує центр обмежувального прямокутника boundingBox для визначення положення об'єкту відносно камери. Залежно від значень objectPositionX та objectPositionY, які представляють різницю між координатою центра об'єкту та позицією камери по осі X та Y відповідно, визначається опис положення об'єкту у просторі.

Функція calculateDistance використовується для визначення відстані між камерою і об'єктом на основі координат обмежувального прямокутника та позиції камери.

Для розрахунку відстані нам потрібно врахувати розміри об'єкту на зображенні, відносну висоту об'єкта, а також кут огляду камери. Ці дані можуть бути доступні у наших результатах моделі або можуть бути задані окремо.

Приклад коду, який демонструє, як визначити відстань між камерою і об'єктом, а також кількість кроків до об'єкту:

```
func calculateDistance(_ boundingBox: CGRect,
                       _ cameraPosition: SIMD3<Float>,
                       _ stepLength: Float)
                       -> (distance: Float,
                       steps: Int) {
    let objectHeight = boundingBox.height // Висота об'єкта на зображенні
    let focalLength = 1000 // Фокусна відстань камери (наприклад, 1000
                           пікселів)
    let knownObjectHeight = 1.8 // Відома висота об'єкта у реальному світі
                                (наприклад, 1.8 метра)
    let imageHeight = // Висота зображення (пікселі)
    let distance = (knownObjectHeight * focalLength * imageHeight) /
                    (objectHeight * cameraPosition.y)
    let steps = Int(distance / stepLength)
    return (distance, steps)
}
```

У цьому прикладі функція calculateDistance отримує параметр stepLength, який представляє довжину одного кроку. За допомогою цього параметра вона розраховує кількість кроків, необхідних для пройдення відстані до об'єкту.

Функція повертає кортеж, що містить відстань у просторі між камерою і об'єктом (distance) та кількість кроків до об'єкту (steps).

Для розпізнавання образів та аналізу зображення використовувалися бібліотеки комп'ютерного зору, такі як Vision, OpenCV та Core ML. Для створення моделі ML використовувалася платформа Apple Create ML, що дозволяє створювати моделі ML на основі зображень та тексту. Для розробки інтерфейсу користувача використовувалася стандартна бібліотека UIKit, що дозволяє створювати елементи інтерфейсу користувача для iOS пристроїв.

В процесі розробки застосунку було дотримано основні принципи розробки мобільних застосунків, такі як ефективність, надійність та зручність використання. Для забезпечення надійності застосунку були використані різноманітні тестувальні методи, включаючи модульні, інтеграційні та прийомний тест.

Окремі етапи розробки мобільного застосунку можуть бути описані наступним чином:

- Аналіз вимог та розробка концепції: на цьому етапі було проведено детальний аналіз вимог та потреб користувачів з фізичними та сенсорними порушеннями. Було розроблено концепцію застосунку та його інтерфейсу користувача.
- Проєктування застосунку: після розробки концепції була розроблена архітектура застосунку та створені макети його інтерфейсу користувача.
- Розробка функціональності: на цьому етапі були реалізовані основні функціональні можливості застосунку, такі як розпізнавання образів, відтворення звуків, допомога з навігацією тощо.
- Тестування та налагодження: після реалізації функціональності було проведено тестування застосунку на різних пристроях з платформою iOS. Після виявлення помилок були проведені налагодження та виправлення проблем.

UML діаграма основних класів проєкту (див. додаток Д) та діаграма прецедентів (див. додаток Е) представлені у додатках.

Модель проєкту складається з 7 основних класів (див. табл. 4.1).

Таблиця 4.1 – Опис основних класів

№	Клас	Опис
1	MainViewController	Контролер, що містить основний інтерфейс користувача та є центральним місцем взаємодії з користувачем.
5	EnvironmentReaderViewController	Клас, який відповідає за зчитування даних про навколишнє середовище

6	SettingsViewController	Контролер, що відповідає за відображення налаштувань застосунку та їх збереження.
7	CameraViewController	Клас, що дозволяє працювати з камерою телефону для зйомки фотографій
8	SoundManager	Клас, відповідає за відтворення звукових ефектів у застосунку. Для цього він використовує технологію AVFoundation, що вбудована в iOS SDK і дозволяє працювати з аудіофайлами та звуковими потоками.
9	ObjectDetectionViewController	Клас відповідає за розпізнавання об'єктів на зображеннях, використовуючи модель ML. У ньому реалізовані методи для завантаження моделі, обробки зображення та відображення результатів розпізнавання на екрані. Крім того, в цьому класі реалізовані методи для відображення інформації про об'єкти, такої як назва, категорія та ймовірність відповідності.

Ці класи взаємодіють між собою та з різними системними класами, такими як AVFoundation для озвучення тексту та UIKit для відображення інтерфейсу користувача. Кожен клас відповідає за свою конкретну функцію в застосунку, що дозволяє зберігати код застосунку організованим та підтримувати

4.3 Інструкція користувача

Для комфортного користування мобільними пристроями люди з фізичними та сенсорними порушеннями можуть використовувати голосового помічника Siri. Нижче наведено декілька кроків, які допоможуть запустити застосунок голосом:

- Активуйте Siri: на пристрої iOS зайдіть до налаштувань, знайдіть розділ "Siri та Пошук" і включіть опцію "Дозволити Siri".
- Запитайте Siri про запуск застосунку: необхідно промовити фразу "Hey Siri" (або викликати Siri, натиснувши кнопку Home або бічну кнопку на сумісних пристроях) і поставити запитання "Запусти [назва застосунку]". Асистент спробує запустити вказаний застосунок та повідомить, якщо не зможе знайти або запустити застосунок.

Рисунок 4.1 – початковий екран з кнопками: 1 – “Navigation”, 2 – “Environment Reader”, 3 – “Settings”.

Під час запуску застосунку ми потрапляємо на основний екран з 3 можливими опціями для використання: 1 – “Navigation”, 2 – “Environment Reader”, 3 – “Settings” (див. рис. 4.1).

Якщо користувач натисне на кнопку “Settings”, то застосунок перенаправить на наступний екран, де він зможе відрегулювати швидкість мовлення голосового асистенту (див. рис. 4.2)

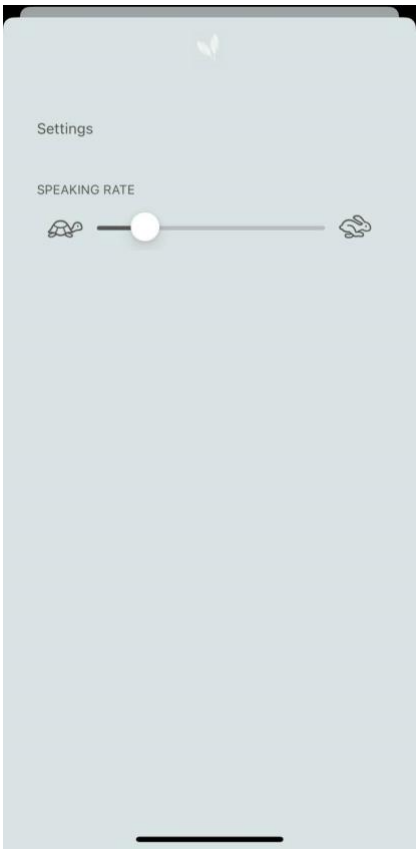


Рисунок 4.2 - Екран налаштувань застосунку, з регулятором швидкості мовлення

Повернутися до основного екрану застосунку можливо, провівши пальцем по екрану згори вниз.

Якщо користувач обере першу основну функцію проєкту, натиснувши на кнопку “Navigation” (див. рис. 4.1), то застосунок перенаправить на наступний екран (див. рис. 4.3).

Достатньо спрямувати камеру пристрою на об'єкт, також можна обрати виділений об’єкт. Застосунок використовує розпізнавання образів, щоб ідентифікувати об'єкти та надати інформацію про них. Після розпізнавання об'єкта, застосунок озвучить інформацію про нього, таку як назву і розташування, відстань до нього. Також на екрані відображені координати розташування у просторі.

У верхній частині екрану також відображається додаткова інформація:

1 – “Inference” - реальний час аналізу та розпізнавання даних, що допомагає в реалізації функціональності застосунку,

2 – “Execution” – час існування даного екрану,

3 – “FPS” - швидкість оновлення відеопотоку з камери або частоту розпізнавання образів (див. рис. 4.4).

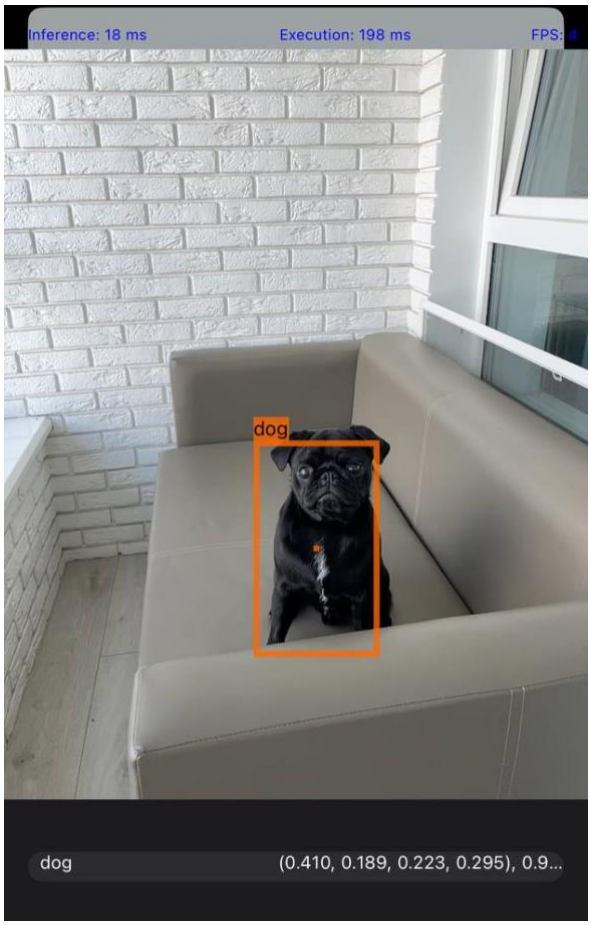


Рисунок 4.3 - Екран розпізнавання образів

Рисунок 4.4 - Фрагмент екрану розпізнавання образів з додатково інформацією 1 – “Inference”, 2 – “Execution”, 3 – “FPS”

Змінити екран можливо провівши пальцем зверху вниз. Далі користувач повертається на головний екран і може перейти до наступної основної функції

застосунку натиснувши на кнопку Enviroment reader (див. рис. 4.1). Завдяки використання розпізнавання образів відбувається виявлення перешкод на шляху користувача, таких як стовпи, стіни, двері або інші об'єкти, які можуть ускладнити рух (див. рис. 4.5).

Рисунок 4.5 - Екран оточення

Також під час. Використання функцій об’єкту відбувається аудіо озвучення інструкцій з пересування для користувача. Також внизу екрану зображено FPS, кількість кроків.

ВИСНОВКИ

Підсумовуючи, розробка мобільного застосунку для допомоги людям з фізичними та сенсорними порушеннями є актуальною і важливою задачею, що вимагає використання сучасних технологій та знань з області штучного інтелекту та машинного навчання. Відповідно до поставленої мети були виконані завдання*:*

- проведено аналіз наявних на ринку мобільних застосунків, для розпізнавання образів;
- розглянуто та досліджено теоретичні основи машинного навчання;
- розроблено та реалізовано мобільний застосунок для людей з фізичними та сенсорними порушеннями;
- поглиблено та закріплено навички роботи з мовою програмування Swift та середовищем програмування Xcode.

Для розробки мобільного застосунку були використано інструментарій: Swift, фреймворки Core ML для інтеграції машинного навчання та UIKit для розробки інтерфейсу користувача, а також середовище розробки Xcode.

Обрано спрямованість на людей з фізичними та сенсорними порушеннями, оскільки ця група користувачів часто зустрічає значні труднощі в повсякденному житті. Наприклад, люди з обмеженою рухливістю можуть мати труднощі з пересуванням, використанням звичайних побутових приладів та інших повсякденних задач. Так само, люди з сенсорними порушеннями можуть мати обмеження в сприйнятті інформації та комунікації.

Результатом роботи є мобільний застосунок на платформі iOS може стати допоміжним інструментом для покращення якості життя цієї групи користувачів, допомогти їм виконувати повсякденні завдання та забезпечити більш вільне та комфортне пересування. Це може сприяти підвищенню їхньої самостійності, підтримувати їхню незалежність та забезпечити більш повноцінну участь у суспільстві. Застосунок має інтуїтивно зрозумілий інтерфейс користувача, що дозволяє легко користуватись ним.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Disability. World Health Organization (WHO) [Електронний ресурс] – Режим доступу до ресурсу: https://www.who.int/news-room/fact-sheets/detail/disability-and-health#:~:text=Key%20facts,1%20in%206%20of%20us
2. Mobile Operating System Market Share Worldwide | Statcounter Global Stats. StatCounter Global Stats [Електронний ресурс] – Режим доступу до ресурсу: https://gs.statcounter.com/os-market-share/mobile/worldwide
3. Люди з обмеженими можливостями (інваліди) [Електронний ресурс] – Режим доступу до ресурсу: https://igov.org.ua/subcategory/1/3/situation/20#anchor1
4. Bloomberg. Bloomberg [Електронний ресурс] – Режим доступу до ресурсу: https://www.bloomberg.com/news/articles/2021-07-29/the-apps-and-technology-helping-people-with-disabilities
5. Inc S. Spoken - Tap to Talk AAC. App Store. [Електронний ресурс] – Режим доступу до ресурсу: https://apps.apple.com/us/app/spoken-tap-to-talk-aac/id1034487817
6. Communications N. Dragon Anywhere. App Store. [Електронний ресурс] – Режим доступу до ресурсу: https://apps.apple.com/us/app/dragon-anywhere/id1024652126
7. Be My Eyes - See the world together [Електронний ресурс] – Режим доступу до ресурсу: https://www.bemyeyes.com/
8. Corporation M. Seeing AI. App Store. [Електронний ресурс] – Режим доступу до ресурсу: https://apps.apple.com/us/app/seeing-ai/id999062298
9. TapTapSee - Blind and Visually Impaired Assistive Technology - powered by CloudSight.ai Image Recognition API. TapTapSee - Blind and Visually Impaired Assistive Technology - powered by CloudSight.ai Image Recognition API [Електронний ресурс] – Режим доступу до ресурсу: https://taptapseeapp.com

10. Visual Interpreting Get Live, On-demand Access to Visual Information. Aira - Visual Information On Demand. [Електронний ресурс] – Режим доступу до ресурсу: https://aira.io/
11. Goodfellow I., Bengio Y., Courville A. Deep Learning (Adaptive Computation and Machine Learning series). The MIT Press, 2016.
12. Russel S., Norvig P. Artificial Intelligence: A Modern Approach. 3rd ed. Prentice-Hall, 2009
13. Advances in neural information processing systems 25 : 26th annual conference on neural information processing / ed. by P. Bartlet et al. ; Issuing body neural information processing systems. NY : Curran Associates, Inc., Red Hook, 2013
14. Mobile Data Science and Intelligent Apps: Concepts, AI-Based Modeling and Research Directions / I. H. Sarker et al. Mobile Networks and Applications. 2020. Vol. 26, no. 1. P. 285–303.
15. Bishop C. M. Pattern Recognition and Machine Learning. NY : Springer New York, 2006.
16. Murphy K. P. Machine Learning: A Probabilistic Perspective. London : The MIT Press, 2012.
17. Syed U., Taskar B. Semi-supervised learning with adversarially missing label information. Advances in Neural Information Processing Systems 24, 10 December 2010
18. Swift - Apple Developer. Apple Developer [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/swift/
19. Swift Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/
20. Veen T. Swift in Depth. Manning Publications, 2018.
21. Welcome to Python.org. Python.org. [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/

22. Xcode 14 - Apple Developer. Apple Developer [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/xcode/
23. Core ML | Apple Developer Documentation. Apple Developer Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/coreml
24. Newnham J. Machine Learning with Core ML: An iOS developer's guide to implementing machine learning in mobile apps. Birningham: Packt Publishing, 2018.
25. AVFoundation - Apple Developer. Apple Developer [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/av-foundation/
26. UIKit | Apple Developer Documentation. Apple Developer Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/uikit/
27. Vision | Apple Developer Documentation. Apple Developer Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/vision
28. Design Patterns: Elements of Reusable Object-Oriented Software / E. Gamma et al. Addison-Welsey Professional, 1994.
29. Garcia R. F. iOS Architecture Patterns: MVC, MVP, MVVM, VIPER, and VIP in Swift. Apress, 2023.
30. You Only Look Once: Unified, Real-Time Object Detection / J. Redmon et al. Conference on Computer Vision and Pattern Recognition, Washington.
31. VNCoreMLRequest | Apple Developer Documentation. Apple Developer Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/vision/vncoremlrequest
32. Hollemans M. Core ML Survival Guide. Leanpub, 2020. 505 p.
33. AVCam: Building a Camera App | Apple Developer Documentation. Apple Developer Documentation [Електронний ресурс] – Режим доступу до ресурсу:

https://developer.apple.com/documentation/avfoundation/capture_setup/avcam_building_a_camera_app

34. Taylor P. Text-to-Speech Synthesis. Cambridge University Press, 2009. 626 p.
35. AVSpeechSynthesizer | Apple Developer Documentation. Apple Developer Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/avfaudio/avspeechsynthesizer
36. AVSpeechUtterance | Apple Developer Documentation. Apple Developer Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/avfaudio/avspeechutterance

ДОДАТОК А

Використання моделі

```
class ObjectDetectionViewController: UIViewController, ARSessionDelegate, ARSCNViewDelegate {

    var objectRecognitionModel: yolov5s {
        do {
            let config = MLModelConfiguration()
            config.computeUnits = .all
            return try yolov5s(configuration: config)
        } catch {
            print(error)
            fatalError("Cannot create YOLOv5s")
        }
    }

    // MARK: Vision
    var request: VNCoreMLRequest?
    var visionModel: VNCoreMLModel?
    var didInference = false

    // MARK: Prediction requests array
    var predictions: [VNRecognizedObjectObservation] = []

    var soundManager = SoundManager()

    lazy var videoPreview: ARSCNView = {
        let videoPreview = ARSCNView(frame: self.view.frame)
        videoPreview.clipsToBounds = true
        videoPreview.translatesAutoresizingMaskIntoConstraints = false

        return videoPreview
    }()
  // MARK: CoreML
  func setupCoreMLModel() {
      if let visionModel = try? VNCoreMLModel(for: objectRecognitionModel.model) {
          self.visionModel = visionModel
          request = VNCoreMLRequest(model: visionModel,
                                    completionHandler: didCompleteVisionRequest)
          request?.imageCropAndScaleOption = .scaleFill
      } else {
          fatalError("Fail to create vision model")
      }
  }

extension ObjectDetectionViewController {

    func predictUsingVision(pixelBuffer: CVPixelBuffer) {
        guard let request = request else { fatalError() }

        self.semaphore.wait()   /// wait(): -1, signal(): +1
        let handler = VNImageRequestHandler(cvPixelBuffer: pixelBuffer, orientation: .up)
        try? handler.perform([request])
    }
```

ДОДАТОК Б

Отримання та передача зображення з відеопотоку

```
// MARK: ARSession
func startARSession() {
    guard ARWorldTrackingConfiguration.supportsFrameSemantics([.sceneDepth]) else { return }

    /// Enable both the `sceneDepth` and `smoothedSceneDepth` frame semantics.
    let config = ARWorldTrackingConfiguration()

    config.frameSemantics = [.sceneDepth]
    videoPreview.session.run(config)
}

func pauseARSession() {
    videoPreview.session.pause()
}
```

```
// MARK: ARSession
func session(_ session: ARSession, didUpdate frame: ARFrame) {
    guard let depth = frame.sceneDepth?.depthMap else { return }
    guard let confidence = frame.sceneDepth?.confidenceMap else { return }
    let depthWidth = CVPixelBufferGetWidth(depth)  // MARK: width = 256
    let depthHeight = CVPixelBufferGetHeight(depth)  // MARK: height = 192

    CVPixelBufferLockBaseAddress(depth, .readOnly)
    CVPixelBufferLockBaseAddress(confidence, .readOnly)

    guard let depthBaseAddress = CVPixelBufferGetBaseAddress(depth) else { return }
    guard let confidenceBaseAddress = CVPixelBufferGetBaseAddress(confidence) else { return }

    /// UnsafeMutableRawPointer -> UnsafeBufferPointer
    let bindDepthPtr = depthBaseAddress.bindMemory(to: Float32.self,
                                                   capacity: depthWidth * depthHeight)
    let bindConfidencePtr = confidenceBaseAddress.bindMemory(to: Int8.self,
                                                             capacity: depthWidth * depthHeight)

    /// UnsafeMutablePointer -> UnsafeBufferPointer
    let depthBufPtr = UnsafeBufferPointer(start: bindDepthPtr,
                                          count: depthWidth * depthHeight)
    let confidenceBufPtr = UnsafeBufferPointer(start: bindConfidencePtr,
                                               count: depthWidth * depthHeight)

    let depthArray = Array(depthBufPtr)
    let confidenceArray = Array(confidenceBufPtr)

    let patternArray: [[Float32]] = Array(repeating: Array(repeating: 0, count: depthWidth),
                                          count: depthHeight)
    var iter = depthArray.makeIterator()
    let newDepthArray = patternArray.map { $0.compactMap {_ in iter.next() }}

// MARK: ARSCNView CVPixelBuffer
guard let frame = session.currentFrame else { return }
let imageBuffer = frame.capturedImage

let imageSize = CGSize(width: CVPixelBufferGetWidth(imageBuffer),
                       height: CVPixelBufferGetHeight(imageBuffer))
let viewPort = videoPreview.bounds
let viewPortSize = videoPreview.bounds.size
let interfaceOrientation = self.videoPreview.window!.windowScene!.interfaceOrientation
let image = CIImage(cvImageBuffer: imageBuffer)
let normalizeTransform = CGAffineTransform(scaleX: 1.0/imageSize.width,
                                           y: 1.0/imageSize.height)
```

```
guard let img = UIImage(ciImage: transformedImage).convertToBuffer() else { return }
if !self.didInference {
    self.didInference = true
    self.performanceMeasurement.didStartNumericMeasurement()
    self.predictUsingVision(pixelBuffer: img)
}
var minValueDictionary: [Float32 : [String]] = [:]
for prediction in predictions {
    let detectedBoundingBox = prediction.boundingBox
    let depthBounds = VNImageRectForNormalizedRect(detectedBoundingBox, depthHeight, depthWidth)  /
    var boundingBoxCoordinate: Array<Int> = []
    boundingBoxCoordinate += [Int(round(depthBounds.minX)),
                              Int(round(depthBounds.minY)), Int(round(depthBounds.maxX)),
                              Int(round(depthBounds.maxY))]
    boundingBoxCoordinate = boundingBoxCoordinate.map{ $0 < 0 ? 0 : $0 }  /
    var convertedDepth = convertToDepthCoordinate(coordinate: boundingBoxCoordinate)  /
    convertedDepth = convertedDepth.map{ $0 < 0 ? 0 : $0 }
    var minDepth: Float32 = 10.0
    var minDepthCoordinate: String = ""
    for y in convertedDepth[1] ... convertedDepth[3] {
        let slice = newDepthArray[y][convertedDepth[0] ... convertedDepth[2]]
        guard let minData = slice.min() else { return }
        if minData < minDepth {
            minDepth = minData
            minDepthCoordinate = "\(String(slice.firstIndex(of: minData)!)), \(String(y))"  /
        }
    }
    minValueDictionary[minDepth] = [minDepthCoordinate, String(prediction.label!)]
}
```

ДОДАТОК В

Обробка результатів

```
func predictUsingVision(pixelBuffer: CVPixelBuffer) {
    guard let request = request else { fatalError() }

    self.semaphore.wait()   /// wait(): -1, signal(): +1
    let handler = VNImageRequestHandler(cvPixelBuffer: pixelBuffer, orientation: .up)
    try? handler.perform([request])
}


// MARK: Post processing
func didCompleteVisionRequest(request: VNRequest, error: Error?) {
    self.performanceMeasurement.didObjectLabeled(with: "EndInference")
    if let predictions = request.results as? [VNRecognizedObjectObservation] {
        self.predictions = predictions
        DispatchQueue.main.async {
            self.boundingBoxView.predictedObjects = predictions
            self.labelsTableView.reloadData()
            self.performanceMeasurement.didEndNumericMeasurement()
            self.didInference = false
        }
    } else {
        self.performanceMeasurement.didEndNumericMeasurement()
        self.didInference = false
    }
    self.semaphore.signal()
}

func createLabelAndBox(prediction: VNRecognizedObjectObservation) {
    let labelString: String? = prediction.label
    let color: UIColor = labelColor(with: labelString ?? "N/A")

    let scale = CGAffineTransform.identity.scaledBy(x: bounds.width, y: bounds.height)
    let transform = CGAffineTransform(scaleX: 1, y: -1).translatedBy(x: 0, y: -1)
    let bgRect = prediction.boundingBox.applying(transform).applying(scale)

    let bgView = UIView(frame: bgRect)
    bgView.layer.borderColor = color.cgColor
    bgView.layer.borderWidth = 4
    bgView.backgroundColor = UIColor.clear
    addSubview(bgView)

    let label = UILabel(frame: CGRect(x: 0, y: 0, width: 300, height: 300))
    label.text = labelString ?? "N/A"
    label.font = UIFont.systemFont(ofSize: 13)
    label.textColor = UIColor.black
    label.backgroundColor = color
    label.sizeToFit()
    label.frame = CGRect(x: bgRect.origin.x,
                         y: bgRect.origin.y - label.frame.height,
                         width: label.frame.width,
                         height: label.frame.height)
    addSubview(label)
```

```
/// Create a dotView to track the object.
let dotView = UIView(frame: CGRect(x: bgRect.midX - 2,
                                   y: bgRect.midY - 2,
                                   width: 4,
                                   height: 4))
dotView.layer.borderColor = color.cgColor
dotView.layer.borderWidth = 2
dotView.backgroundColor = UIColor.clear
addSubview(dotView)

/// Store the object's center coordinates in the array, in the form of [label: [x coordinate, y coordinate]].
objectCenterCoordinates[labelString!] = [bgRect.midX, bgRect.midY]

/// Create a lineView to connect the center of the same object.
let lineView = UIView(frame: CGRect(x: bgRect.midX,
                                    y: bgRect.midY,
                                    width: 0,
                                    height: 0))
lineView.layer.borderColor = color.cgColor
lineView.layer.borderWidth = 2
lineView.backgroundColor = UIColor.clear
addSubview(lineView)

/// Update the center of the object as the object moves.
objectCenterCoordinates[labelString!] = [bgRect.midX, bgRect.midY]

/// Update the line as the object moves.
lineView.frame = CGRect(x: objectCenterCoordinates[labelString!]![0],
                        y: objectCenterCoordinates[labelString!]![1],
                        width: bgRect.midX - objectCenterCoordinates[labelString!]![0],
                        height: bgRect.midY - objectCenterCoordinates[labelString!]![1])

/// Remove the dot and the line when the object disappears.
if prediction.confidence < 0.4 {
    dotView.removeFromSuperview()
    lineView.removeFromSuperview()
}

extension VNRecognizedObjectObservation {
    var label: String? {
        return self.labels.first?.identifier
    }
}
```

ДОДАТОК Г

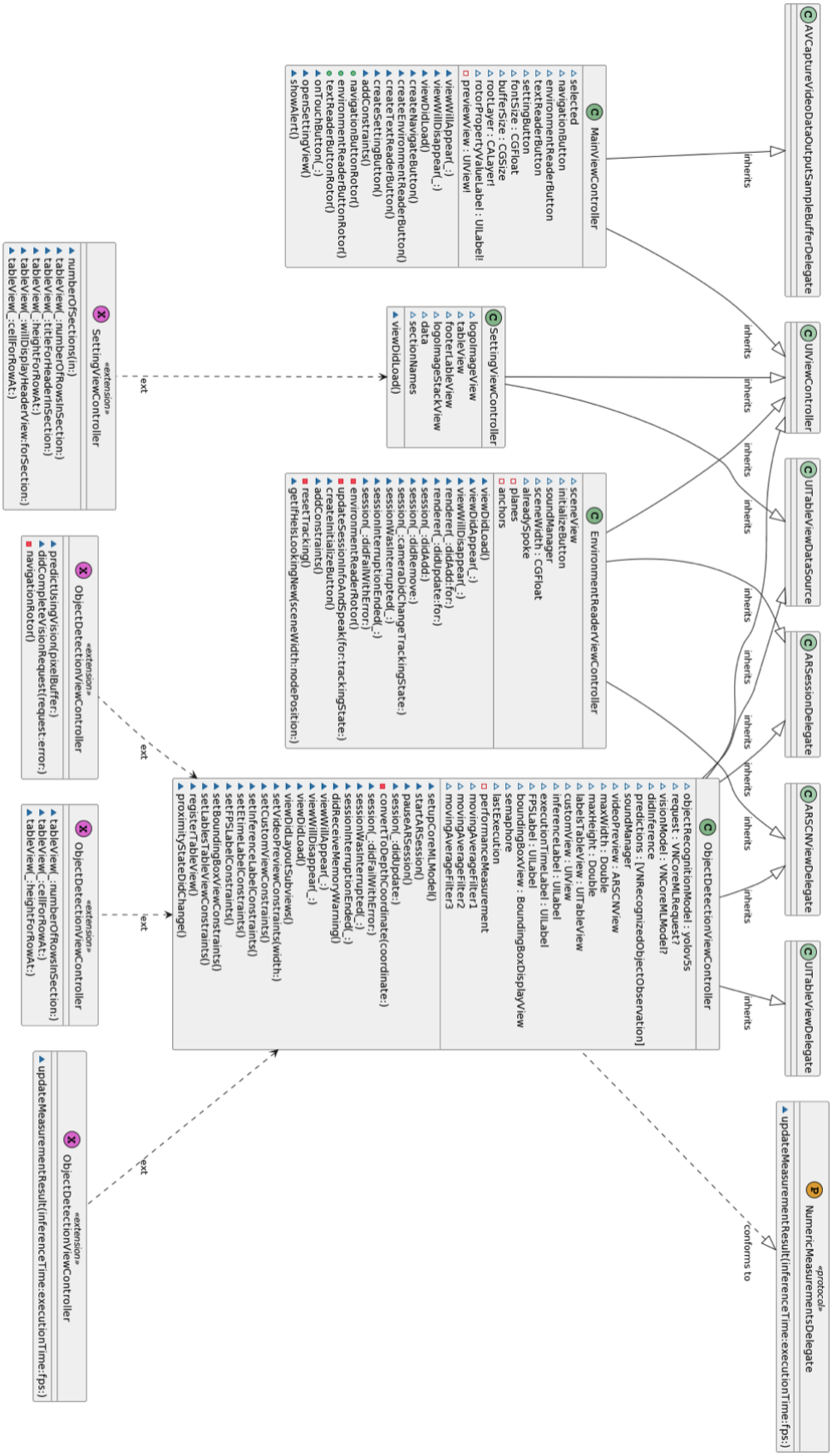
Додавання синтезу мовлення

```
class SoundManager {
    /// управління аудіо-двигуном AVAudioEngine. Це потрібно для налаштування та керування аудіопотоком.
    let audioEngine = AVAudioEngine()
    ///типовий звук-сигнал
    let defaultBeep = "defaultBeep.mp3"
    /// вузол аудіопрогравача AVAudioPlayerNode. Він використовується для відтворення звукових ефектів
    /// та аудіопотоків у застосунку.
    let audioPlayerNode = AVAudioPlayerNode()
    /// вузол аудіооточення AVAudioEnvironmentNode. Він використовується для налаштування акустичних
    /// властивостей аудіопотоку, таких як просторова розстановка звуку та об'єм.
    let environmentNode = AVAudioEnvironmentNode()
    /// синтезатор мовлення AVSpeechSynthesizer. Він використовується для озвучування текстових повідомлень або
    /// взаємодії з користувачем через аудіо.
    let synthesizer = AVSpeechSynthesizer()
    /// визначає швидкість відтворення мовлення, встановлену для синтезатора мовлення.
    var speakingRate: Float
    /// начення 1.0 відповідає повній гучності, а значення менше 1.0 зменшує гучність мовлення.
    var speakingVolume = Float(1.0)

func playBeep(x:Float, y:Float, z:Float,beepSource:String = "defaultBeep.mp3") {
    /// отримує URL-адресу файлу звуку "beep" з ресурсів проекту.
    let defaultBeepUrl = Bundle.main.url(forResource: beepSource, withExtension: nil)
    /// створює об'єкт AVAudioFile для читання звукового файлу "beep"
    let defaultBeepFile = try! AVAudioFile(forReading: defaultBeepUrl!)
    /// розкладає звуковий файл для відтворення за допомогою об'єкта audioPlayerNode.
    ///  Звуковий файл буде відтворюватись без затримок від моменту виклику функції.
    self.audioPlayerNode.scheduleFile(defaultBeepFile, at: nil, completionHandler: nil)
    /// встановлює позицію звукового відтворення в тривимірному просторі.
    self.audioPlayerNode.position = AVAudio3DPoint(x: x, y: y, z: z)
    /// запускає відтворення звукового файлу за допомогою об'єкта audioPlayerNode.
    /// Звуковий файл почне відтворюватись з заданою позицією в тривимірному просторі.
    self.audioPlayerNode.play()
}

func speak(_ string: String) {
    if UIAccessibility.isVoiceOverRunning {
        UIAccessibility.post(notification: .announcement, argument: string)
    } else {
        /// створює об'єкт AVSpeechUtterance з переданим текстом.
        /// AVSpeechUtterance визначає текст, який має бути вимовлений голосом.
        let utterance = AVSpeechUtterance(string: string)
        /// встановлює голосовий зразок для AVSpeechUtterance.
        /// У даному випадку, використовується голосовий зразок англійської мови ("en-US").
            utterance.voice = AVSpeechSynthesisVoice(language: "en-US")
        /// встановлює швидкість вимовлення голосу.
        /// Значення speakingRate є регульованим параметром, який можна налаштувати.
        utterance.rate = speakingRate
        /// встановлює гучність вимовлення голосу.
        /// Значення speakingVolume є регульованим параметром, який можна налаштувати.
        utterance.volume = speakingVolume
        synthesizer.stopSpeaking(at: AVSpeechBoundary.word)
        /// запускає голосове відтворення переданого тексту за допомогою об'єкта synthesizer.
        ///  Голосове відтворення розпочнеться з налаштованими параметрами,
        ///  такими як голосовий зразок, швидкість і гучність.
        synthesizer.speak(utterance)
    }
}
```

```
func speakByTTS(_ string: String) {
    let utterance = AVSpeechUtterance(string: string)

    utterance.rate = speakingRate
    utterance.volume = speakingVolume

    synthesizer.stopSpeaking(at: AVSpeechBoundary.word)
    synthesizer.speak(utterance)
}

func stopSpeak() {
    if (synthesizer.isSpeaking) {
        synthesizer.stopSpeaking(at: AVSpeechBoundary.immediate)
    }
}
```

ДОДАТОК Д

Діаграма основних класів



ДОДАТОК Е

Діаграма прецедентів