Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет __Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти ____другий (магістерський)_____
(рівень вищої освіти)
_____Система ідентифікації звуків за допомогою Machine Learning_____
_____на платформі iOS_____.
(тема)

Виконав: студент 2 курсу, групи СКСм-22-2

_____ Кравцов А.С._____
(прізвище, ініціали)

Спеціальність __123 Комп'ютерна інженерія
(код і повна назва спеціальності)

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

Освітня програма ____Спеціалізовані_____
_____комп'ютерні системи_____
(повна назва освітньої програми)

Керівник ___доц. Філіппенко І.В._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Чумаченко С. В.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

Кафедра Автоматизації проектування обчислювальної техніки

Рівень вищої освіти другий (магістерський)

Спеціальність 123_ Комп'ютерна інженерія
(шифр і назва)

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

Освітня програма Спеціалізовані комп'ютерні системи
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Кравцову Андрію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Система ідентифікації звуків за допомогою Machine Learning_ на платформі iOS
затверджена наказом по університету від _03_ _листопада_2023 р. № _1282 Ст__
2. Термін подання студентом роботи до екзаменаційної комісії 12 січня 2024 р.
3. Вихідні дані до роботи
Операційна система iOS 16.7.1
CoreML модель ідентифікації звуку
Мова програмування Swift
Середовище розробки Xcode
4. Перелік питань, що потрібно опрацювати в роботі
Аналіз предметної області та постановка завдання.
Огляд фреймворків, що будуть використовуватися у додатку.
Програмна модель додатку.
Тестування мобільного додатку.
Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) _____
Слайд-презентація – 14 слайдів

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

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

7. Дата видачі завдання 02.09.2023

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Видача теми проекту, узгодження і	02.09.2023 - 03.09.2023	
	затвердження теми		
2	Аналіз проблеми та існуючих рішень	03.09.2023 - 20.10.2023	
3	Огляд фреймворків для написання	20.10.2023 - 20.11.2023	
	мобільного додатку		
4	Розробка програмних модулів	20.11.2023 - 20.12.2023	
5	Тестування мобільного додатку	20.12.2023 - 02.01.2024	
6	Оформлення матеріалів кваліфікаційної	02.01.2024 - 08.01.2024	
	роботи		
7	Подання кваліфікаційної роботи	08.01.2024 - 12.01.2024	
	та її попередній захист		

Студент _____
(підпис)

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

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 57 сторінок, 14 рисунків, 13 джерел за переліком посилань.

ЗВУК, МОБІЛЬНИЙ ДОДАТОК, ІДЕНТИФІКАЦІЯ, СПЕКТРОГРАМА МЕЛ-СПЕКТРОГРАМА, iOS.

Метою кваліфікаційної роботи є реалізація мобільного додатку ідентифікації звуків.

У ході виконання кваліфікаційної роботи було розглянуто алгоритм перетворення звуку, фреймворки та програмну модель мобільного додатку, було проведено реалізацію та тестування мобільного застосунку.

ABSTRACT

Qualification work: 57 pages, 14 figures, 13 sources.

SOUND, MOBILE APP, IDENTIFICATION, SPECTROGTAM, MEL-SPECTROGRAM, iOS.

The goal of the qualification work is the implementation of a sound identification mobile application.

In the course of the qualification work, the sound conversion algorithm, frameworks and software model of the mobile application were reviewed, the implementation and testing of the mobile application was carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

DFT – discrete Fourier transform

FFT – fast Fourier transform

IDE – integrated development environment

iOS – iPhone OS

MFCC – Mel-frequency cepstral coefficients

ML – machine learning

OS – operating system

PCM – pulse code modulation

STFT – short-time Fourier transform

UI – user interface

дБ – децибел

ПК – персональний комп’ютер

ВСТУП

У сучасному світі розвиток технологій має значний вплив на всі сфери життя, включаючи звуковий аналіз та ідентифікацію звуків. Завдяки стрімкому розвитку областей, таких як машинне навчання і мобільні пристрої, виникають нові можливості для розробки додатків, що базуються на аналізі звукових даних. У цьому контексті кваліфікаційна робота спрямована на вивчення ідентифікації звуків з використанням методів машинного навчання на платформі iOS.

Одним із ключових елементів звукового аналізу є спектрограма, яка дає змогу візуалізувати спектральний склад звукових сигналів в залежності від часу. Проте, для більш точного інтерпретування звукових даних, спектрограму можна покращити шляхом використання мел-шкали. Мел-спектрограма дозволяє враховувати специфіку людського сприйняття звуку та забезпечує більш ефективне представлення звукової інформації.

Окрім того, для реалізації ідентифікації звуків на платформі iOS буде використано фреймворк AVFoundation для захоплення аудіо даних з мікрофона та фреймворк Core ML для інтеграції готової моделі машинного навчання.

Метою кваліфікаційної роботи є детальне дослідження та розробка системи ідентифікації звуків з використанням методів машинного навчання на платформі iOS. Буде проведено аналіз предметної області, сформульовано постановку задачі, розглянуто принципи роботи мел-спектрограми, функціональні можливості фреймворків AVFoundation та Core ML, а також реалізовано додаток для ідентифікації звуків на платформі iOS.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Аналіз предметної області

Ідентифікація звуків є актуальною темою в сучасному світі з багатьма практичними застосуваннями. Мобільний додаток може бути корисним у наступних сферах.

Музична індустрія: додаток може бути використаний для розпізнавання пісень, виконавців або музичних інструментів, що допоможе користувачам швидко визначити, яка пісня грає, і отримати про неї інформацію;

Звуковий аналіз: у галузі аналізу звуку додаток може використовуватися для аналізу звукових даних з метою виявлення особливостей звукових сигналів, може бути застосований для аналізу оточуючого середовища, визначення рівня шуму або розпізнавання звукових подій, таких як голосові команди;

Ігрова індустрія: у ігрових додатках ідентифікація звуків може використовуватися для розпізнавання звукових ефектів, музики або голосового вводу, це може створити більш інтерактивне та захоплююче ігрове середовище для користувачів;

Природознавство та зоологія: додаток може бути корисним для зоологів та дослідників при вивченні звукових сигналів тварин і птахів – може допомогти ідентифікувати різні види тварин за їх голосами або розпізнавати звукові сигнали, пов'язані з певними поведінковими паттернами;

Безпека та спостереження: у системах безпеки або системах відеоспостереження ідентифікація звуків може використовуватися для виявлення та повідомлення про важливі акустичні події, як постріли, голосні крики або аварійні сигнали.

Це лише кілька прикладів сфер застосування додатка для ідентифікації звуків на платформі iOS. Залежно від конкретних потреб і цілей, додаток може бути адаптований для роботи в різних галузях.

1.2 Постановка завдання

Метою даної кваліфікаційної роботи є розробка мобільного додатку, за допомогою якого буде проводитись класифікація певних звуків навколишнього середовища за допомогою ML.

Мобільний додаток повинен бути розроблено під операційну iOS для девайсів серії iPhone. Розробка буде проводитися у Xcode IDE з використанням мови програмування Swift. Будуть використані такі фреймворки, як UIKit, Core ML, Vision та AVFoundation.

2 ЗВУК ТА СПЕКТРОГРАМА

2.1 Звук та його параметри

Звук – це коливання повітряних молекул, які поширюються у вигляді звукових хвиль. Він може бути поданий у вигляді аналогового або цифрового сигналу. Для ефективної роботи зі звуком потрібно розуміти його основні параметри.

Основними параметрами звуку є наступні.

1) Частота – це кількість коливань за секунду. Вимірюється в герцах (Гц) і визначає висоту звуку. Чим вища частота, тим вище звук.

2) Амплітуда – це міра сили звуку. Вона визначається різницею в тиску між піком і долиною звукової хвилі. Амплітуда вимірюється в децибелах.

3) Фаза – це час, необхідний для проходження звукової хвилі від однієї точки до іншої. Фазу можна змінювати на різні значення, що може призвести до зміни звуку.

4) Тривалість – це час, який звукова хвиля займає для повного коливання від однієї точки до іншої. Тривалість може варіюватися залежно від джерела звуку.

Крім того, для роботи зі звуком також використовуються інші параметри, такі як форма хвилі, тембр і т. д.

Розуміння основних параметрів звуку важливо для розуміння того, як звук обробляється і використовується в аналізі та ідентифікації звукових сигналів.

Конкретні параметри звуку можуть використовуватися для ідентифікації звуків у різних завданнях. Наприклад, для розпізнавання мови можна використовувати частоту основної гармоніки, форманти та мел-частотні кепстральні коефіцієнти (MFCC). Для розпізнавання музичних інструментів

можна використовувати спектральні характеристики звуку, такі як спектральна ширина, спектральна динаміка та спектральний флуктуаційний коефіцієнт.

У завданні ідентифікації оточуючого середовища параметри звуку, такі як рівень шуму, тривалість та потужність, можуть використовуватися для визначення типу оточуючого середовища, наприклад, міської вулиці, парку або водоспаду.

Для ідентифікації звукових подій в аудіо-контенті (наприклад, реклами або фільмів), параметри звуку можуть використовуватися для визначення типу події, наприклад, дзвінок телефону, дверного дзвінка або звуку двигуна автомобіля.

Таким чином, параметри звуку можуть бути використані в різних завданнях ідентифікації звуків для визначення характеристик звуку та його джерела.

2.2 Спектрограма

Спектрограма – це графічне зображення спектра сигналу в залежності від часу. Вона є потужним інструментом в аналізі сигналів, зокрема у звуковому аналізі. Спектрограма дає можливість візуально відобразити залежність спектра від часу, тим самим роблячи легким визначення частотних компонентів, які входять до складу звуку.

Спектрограма є результатом розбиття сигналу на короткі фрагменти, названі фреймами. Кожен фрейм перетворюється в спектр за допомогою використання алгоритму, такого як швидке перетворення Фур'є (FFT). Після цього створюється графік, де по горизонталі відображається час, а по вертикалі – частота. Інтенсивність кольору відображається за допомогою зміни яскравості пікселів відповідно до сили кожного частотного компонента.

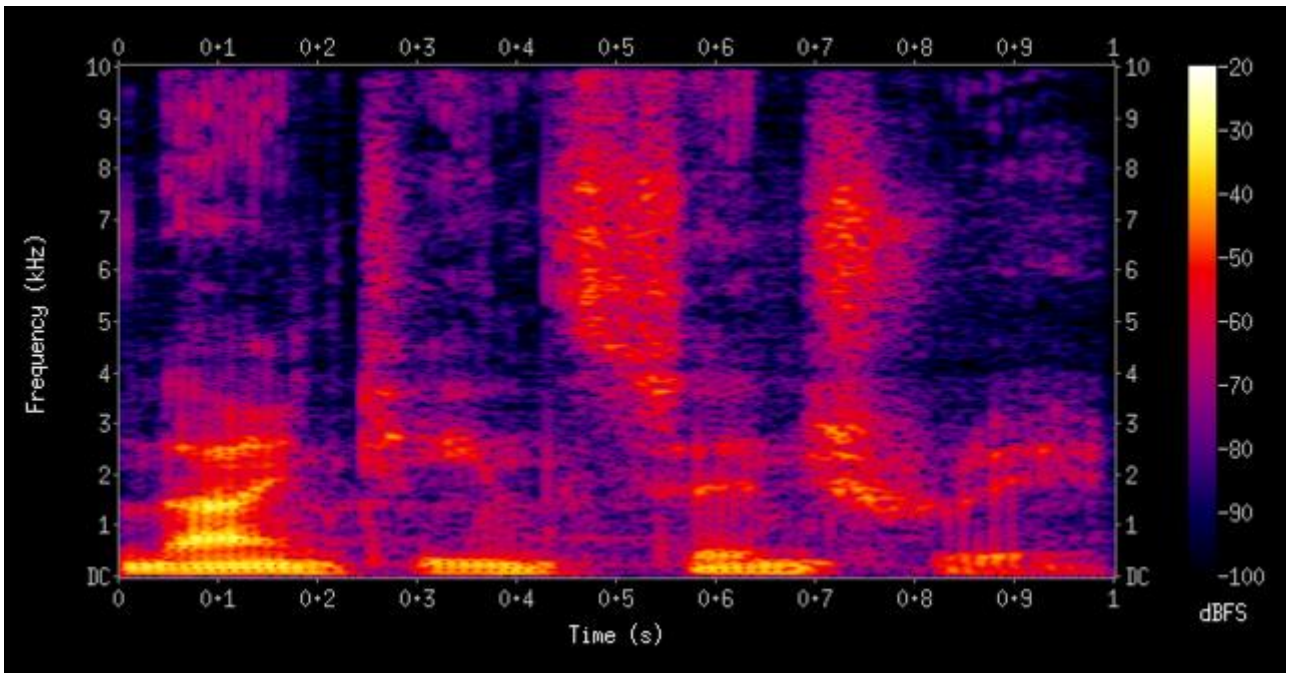

Рисунок 2.1 – Приклад зображення спектрограми

Спектрограма знаходить застосування в різних галузях, зокрема у музиці, мовленні, біомедицині, обробці звуку, розпізнаванні мови та інших. Вона дозволяє аналізувати звукові сигнали та виділяти в них особливості, такі як ритміка, мелодійність, наявність шуму та інші параметри, що є важливими для розпізнавання звуків та розуміння мовлення.

2.3 Мел-спектрограма

Мел-спектрограма – це інструмент для аналізу аудіо-сигналу, який дозволяє представити спектрограму відносно частот відтворення звуку з точки зору сприйняття людиною.

У звуковому аналізі Мел-спектрограма застосовується для виявлення різних характеристик звукового сигналу, таких як інтенсивність звуку, висота тону, зміни звукового тембру тощо. Порівняно з звичайною спектрограмою, Мел-спектрограма забезпечує кращу представленість звукового спектра з точки зору сприйняття людиною.

Мел-спектрограма формується шляхом поділу аудіо-сигналу на невеликі фрагменти та обчислення спектра для кожного фрагменту. Для отримання Мел-спектрограми кожен спектралів частоти перетворюється за допомогою шкали Мела, яка враховує особливості сприйняття людиною звуків різної висоти тона. Це дозволяє отримати спектрограму, яка краще відображає характеристики звуку з точки зору людського сприйняття.

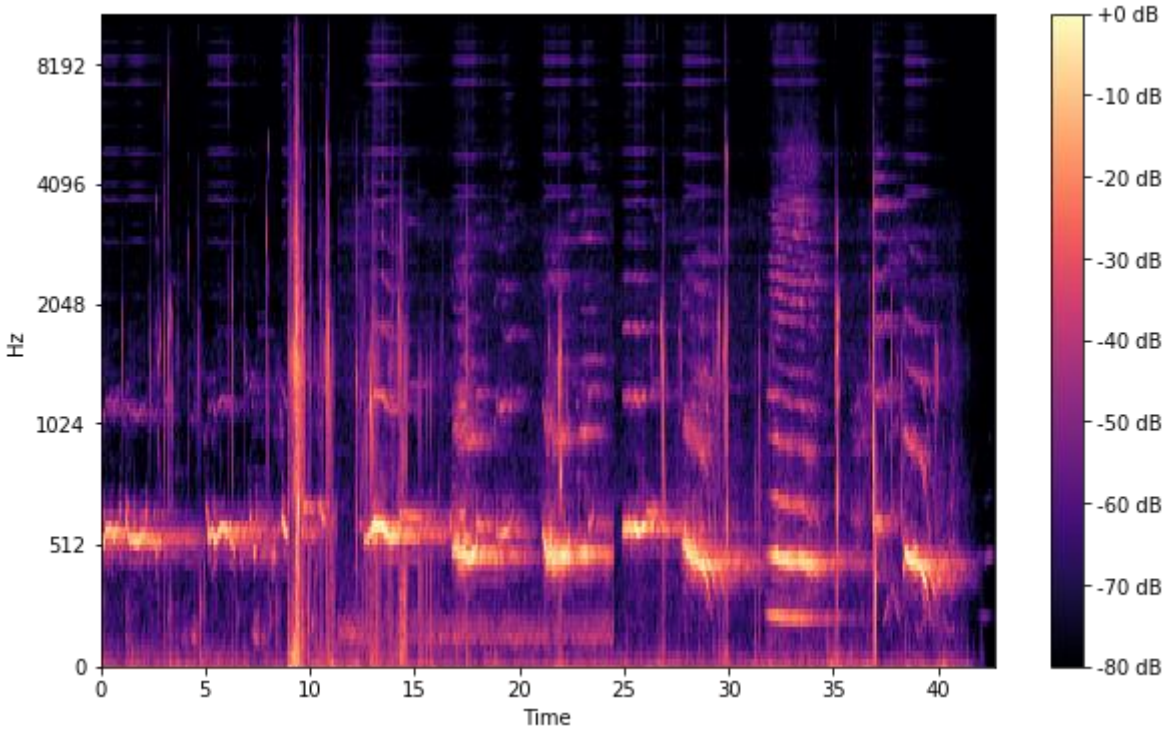

Рисунок 2.2 – Приклад зображення Мел-спектрограми

Застосування Мел-спектрограми у звуковому аналізі є різноманітним. Вона може бути використана для розпізнавання мовлення, класифікації музичних жанрів, визначення емоційного стану людини за голосом, аналізу звуків у середовищах з великою кількістю шуму та багатьох інших задач.

2.4 Алгоритм перетворення звуку в Мел-спектрограму

Для перетворення звуку в Мел-спектрограму використовується наступний алгоритм.

1. Захоплення звуку за допомогою мікрофона або завантаження файла зі звуковою доріжкою

2. Поділ на рамки, для цього аудіо-сигнал розбивається на короткі рамки фіксованої довжини. Часто використовується метод поділу на рамки з перекриванням, де кожна рамка починається з деяким зсувом відносно попередньої рамки.

3. Приведення до дискретної форми. При цьому кожна рамка аудіо-сигналу перетворюється у дискретну форму, наприклад, використовуючи формат PCM.

4. Обчислення короткочасного перетворення Фур'є. Для кожної рамки аудіо-сигналу застосовується STFT, яке перетворює сигнал з часового представлення в частотне представлення. STFT розбиває рамку на декілька накладених віконних функцій та обчислює дискретне перетворення Фур'є кожного вікна. Формулу STFT зображена на рисунку 2.3, де:

– $X(m, \omega)$ – комплексна амплітуда в часовому вікні m та частоті ω;

– x(n) – вхідний сигнал;

– $\omega(n - mR)$ – вікно, зміщене відносно поточного вікна m

– R – крок зсуву у часі;4

– n – часовий індекс;

– ω – частота.

$$X(m, \omega) = \sum_{n=-\infty}^{\infty} x(n)\omega(n - mR)e^{-j\omega n}$$

5. Обчислення амплітудного спектра виконується наступним чином. З отриманих результатів STFT виокремлюється амплітудний спектр, який представляє собою модуль комплексних значень DFT. Це відображає розподіл амплітуди сигналу на різних частотах.

6. Далі відбувається застосування мел-фільтров. Мел-фільтри використовуються для перетворення амплітудного спектра в мел-шкалу. Кожен

мел-фільтр вибирає певний діапазон частот та обчислює середню амплітуду спектра в цьому діапазоні.

7. Логарифмування та нормалізація. Отримані значення з мел-фільтрів піддаються логарифмуванню для зменшення динамічного діапазону. Також може бути застосована нормалізація для вирівнювання амплітуд між різними рамками.

8. Візуалізація. Кінцевий результат представляє собою мел-спектрограму, яка може бути візуалізована у вигляді 2D зображення, де по горизонтальній осі відображається час, а по вертикальній осі - частота. Колір або інтенсивність відображає амплітуду звуку в кожному пікселі.

3 ІНСТРУМЕНТИ РОЗРОБКИ IOS-ДОДАТКУ

3.1 Операційна система та середовище розробки

Для розробки мобільного додатку була обрана операційна система iOS, тому що вона оптимізована для створення застосунків з використанням машинного навчання.

iOS – це мобільна операційна система, що була створена і розроблена компанією Apple. Ця операційна система є другою за поширеністю для мобільних пристроїв у світі після Android.

Однією з найбільших переваг iOS перед Android є фреймворк CoreML, створений саме для використання моделей машинного навчання безпосередньо на пристроях Apple. Це дозволяє легко інтегрувати ML-функціонал в додатки, забезпечуючи високу продуктивність та ефективність обробки даних. Також багато користувачів Apple використовують оновлені версії iPhone та iPad, що сприяє більш широкій аудиторії для додатків, що використовують ML.

Процесори та апаратне забезпечення в пристроях Apple мають оптимізації, які сприяють швидкій та ефективній роботі з ML-моделями, що може забезпечити більшу швидкість та продуктивність за рахунок апаратної підтримки.

В iOS використовується ядро XNU, засноване на мікроядрі Mach і містить програмний код, розроблений компанією Apple, а також код із ОС NeXTSTEP та FreeBSD. Ядро iOS майже ідентичне ядру операційної системи Apple macOS, що була розроблена для ПК. Починаючи з першої версії, iOS працює тільки на планшетних комп'ютерах і смартфонах з процесорами архітектури ARM.

iOS повністю відрізняється від інших операційних систем мобільних телефонів через те, що ця ОС зберігає всі програми на своєму пристрої всередині захисної оболонки. Через це програми не можуть втручатися в роботу

інших програм. iOS розроблено таким чином, що вірус, який пристрій може випадково отримати, не може зашкодити іншим програмам.

Для створення мобільних додатків під iOS використовуються Xcode IDE, що розроблена компанією Apple. Xcode – це повний набір інструментів розробника для створення програм для Mac, iPhone, iPad, Apple Watch і Apple TV. Xcode об'єднує дизайн інтерфейсу користувача, кодування, тестування, налагодження та надсилання в App Store в єдиний робочий процес, підтримує як мову програмування Swift, що є основною мовою розробки для платформ Apple, так і Objective-C, яка використовується в старіших проектах. Вигляд програми наведено на рисунку 3.1.

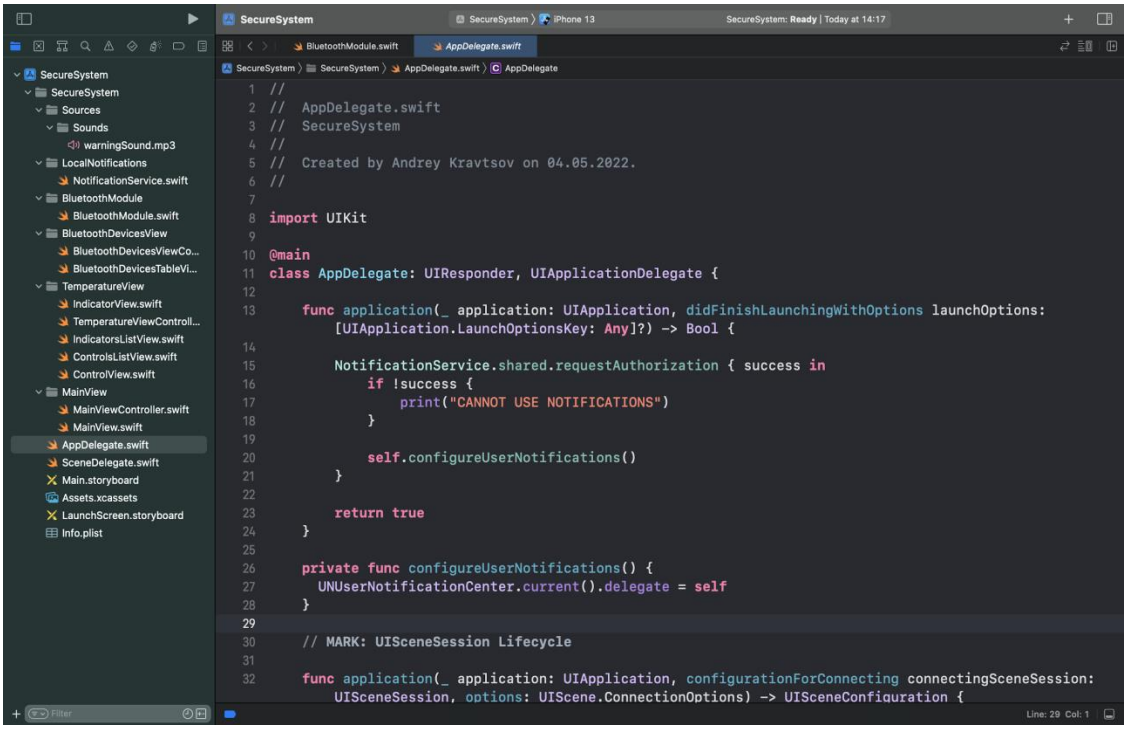

Рисунок 3.1 – Скріншот програми Xcode

Для написання мобільного додатку мовою програмування було обрано Swift. Swift – це мова програмування загального призначення, що компілюється, багатопарадигмальна, розроблена Apple Inc. та open-source спільнотою. Вперше випущений у 2014 році, Swift був розроблений як заміна попередньої мови програмування Apple Objective-C, оскільки Objective-C не

мав сучасних мовних функцій. Swift працює з фреймворками Apple Cocoa і Cocoa Touch, і ключовим аспектом дизайну Swift була можливість взаємодії з величезним набором існуючого коду Objective-C, розробленого для продуктів Apple протягом попередніх десятиліть. Також Swift має сучасний синтаксис, що сприяє покращенню безпеки та надійності програм. Він включає у себе механізми для уникнення помилок, які часто відбуваються у C та Objective-C, що допомагає уникати багатьох типових помилок програмістів. Він побудований з відкритим вихідним кодом компілятора LLVM і був включений в XCode з версії 6, випущеної в 2014 році.

3.2 AVFoundation

Мобільний додаток використовує фреймворк AVFoundation у модулі захоплення звуку для збору аудіо-даних з мікрофона на пристрої iOS та подальшого їх аналізу з метою ідентифікації звуків за допомогою машинного навчання.

AVFoundation – це фреймворк від Apple, що надає розробникам можливість працювати з медіа-файлами, відео та аудіо на платформі iOS.

AVFoundation складається з низких рівнів абстракції, які надають доступ до відповідних пристроїв вводу-виводу та дозволяють програмістам контролювати обробку аудіо- та відеоданих.

Для роботи зі звуком цей фреймворк використовується для запису, обробки та його відтворення. Залучаються такі елементи, як звукові сигнали та спектрограми. За допомогою AVCaptureSession можна збирати аудіо-дані з мікрофона та перетворювати їх у спектрограми звуку, що дозволить відокремити різні звукові сигнали та ідентифікувати їх.

Крім того, AVFoundation дозволяє отримувати інформацію про параметри звуку, такі як частота дискретизації, бітрейт, формат тощо. Ця інформація може бути корисною для аналізу звуку та створення аудіо-проектів.

Усі ці можливості дозволяють розробникам створювати різноманітні додатки, що пов'язані з обробкою звуку на платформі iOS, від музичних ігор до аудіо-редакторів.

3.3 Core ML

Для інтеграції ML-моделі у додаток використовується фреймворк Core ML у модулі аналізу Мел-спектрограми.

Core ML - це фреймворк машинного навчання від Apple, що дозволяє легко вбудовувати готові моделі машинного навчання в додатки для iOS, macOS та інших платформ Apple. Завдяки Core ML, розробники можуть додавати функціонал, пов'язаний з машинним навчанням, до своїх додатків без необхідності писати складний код для навчання та використання моделей.

Фреймворк включає в себе готові моделі машинного навчання, які можуть використовуватись безпосередньо, а також інструменти для створення власних моделей на основі зразків. Для цього використовуються спеціальні інструменти, такі як Create ML, що дозволяють навчати моделі на даних без необхідності мати глибокі знання з машинного навчання.

Одним з головних переваг Core ML є оптимізація для пристроїв Apple, що дозволяє використовувати моделі з меншим використанням енергії та швидкістю в порівнянні з аналогічними моделями, що використовуються на інших платформах.

3.4 Vision

Для роботи з ML-моделлю у модулі аналізу Мел-спектрограми використовується фреймворк Vision.

Vision – це фреймворк для розробки програмного забезпечення на платформі iOS, який надає високорівневий API для обробки зображень та машинного навчання. Його використання дозволяє розробникам створювати

додатки з розпізнаванням об'єктів, виявленням облич, аналізом тексту та багатьма іншими можливостями машинного зору.

Основною складовою Vision є VisionKit, який надає високорівневий API для розпізнавання образів та обробки зображень. Зокрема, він містить набір інструментів для розпізнавання облич, пошуку тексту на зображеннях, виявлення векторів, зіставлення зображень та інших завдань, пов'язаних з машинним зором.

Крім того, Vision включає в себе підтримку Core ML, що дозволяє використовувати моделі машинного навчання для розпізнавання об'єктів та класифікації зображень. Завдяки цьому розробники можуть легко інтегрувати свої власні моделі машинного навчання в свої додатки на iOS.

Однією з ключових переваг Vision є його ефективність. Фреймворк працює на основі високооптимізованих бібліотек, що дозволяє забезпечувати швидку та точну обробку зображень, що робить його ідеальним рішенням для додатків, які працюють з великими обсягами даних та потребують швидкого розпізнавання образів.

3.5 UIKit

Для розробки користувацького інтерфейсу (UI) було застосовано фреймворк UIKit у модулі інтерфейсу користувача. Він надає набір інструментів для побудови графічного інтерфейсу користувача, обробки подій взаємодії з користувачем та керування розміщенням елементів на екрані. UIKit є основним фреймворком для розробки iOS-додатків і забезпечує потужні можливості для створення сучасних, естетично привабливих та функціональних інтерфейсів.

UIKit включає в себе різноманітні класи, які дозволяють створювати елементи інтерфейсу, такі як кнопки, тексти, зображення, таблиці, колекції, навігаційні панелі та багато іншого. Він також надає широкі можливості для

роботи з анімаціями, змінюванням кольорів, обробкою жестів користувача та навігацією між екранами додатку.

UIKit використовує концепцію подій та делегування для обробки взаємодії з користувачем. За допомогою делегатів та вихідних методів, розробник може реагувати на події, такі як натискання кнопок, рухи по екрану, введення тексту та багато іншого. Це дозволяє створювати відзивчиві та інтерактивні додатки.

Окрім того, UIKit забезпечує можливості для роботи з різними типами медіа, включаючи зображення, звук та відео. Він підтримує відображення та обробку зображень, а також відтворення аудіо та відео файлів.

Фреймворк UIKit є важливою складовою розробки iOS-додатків і забезпечує потужні інструменти для створення інтерфейсу, який зручний та привабливий для користувачів.

4 РЕАЛІЗАЦІЯ ПРОГРАМНОЇ ЧАСТИНИ ПРОЕКТУ

4.1 Технічні вимоги до мобільного пристрою

Для завантаження та використання мобільного додатку потребується операційна система iOS, версія якої має бути не менше ніж 14.4. Це не остання версія операційної системи, проте найбільш використовувана. За статистикою, 98% користувачів iOS мають версію 14 та більше.

Для більш ефективної роботи з великими моделями важливо мати пристрій з більшою кількістю оперативної пам'яті. Оптимальний розмір оперативної пам’яті становить 4 ГБ, що дозволяє запускати вимогливі за ресурсами моделі.

Для роботи з моделями машинного навчання процесор повинен мати достатній обсяг вирахувальних можливостей для швидкої обробки моделі. Рекомендовано використання процесорів серії Apple A11 Bionic або пізніших для кращої продуктивності.

Таким чином, будь-який з сучасних iPhone, починаючи від iPhone 8 та більш нових моделей, буде підходящим для роботи з CoreML моделями машинного навчання. Однак, чим новіше пристрій, тим більше ймовірність, що він матиме більше потужності та оптимізацій для роботи з такими моделями.

4.2 Програмна модель мобільного додатку

Для реалізації мобільного додатку створено програмну модель, що складається з:

– модуля бізнес-логіки;

– модуля захоплення звуку;

– модуля трансформації звуку;

– модуля візуалізації звуку;

– модуля аналізу Мел-спектрограми

– Core ML-моделі;

– модуля інтерфейсу користувача.

Схема програмної моделі наведено на рисунку 4.1.

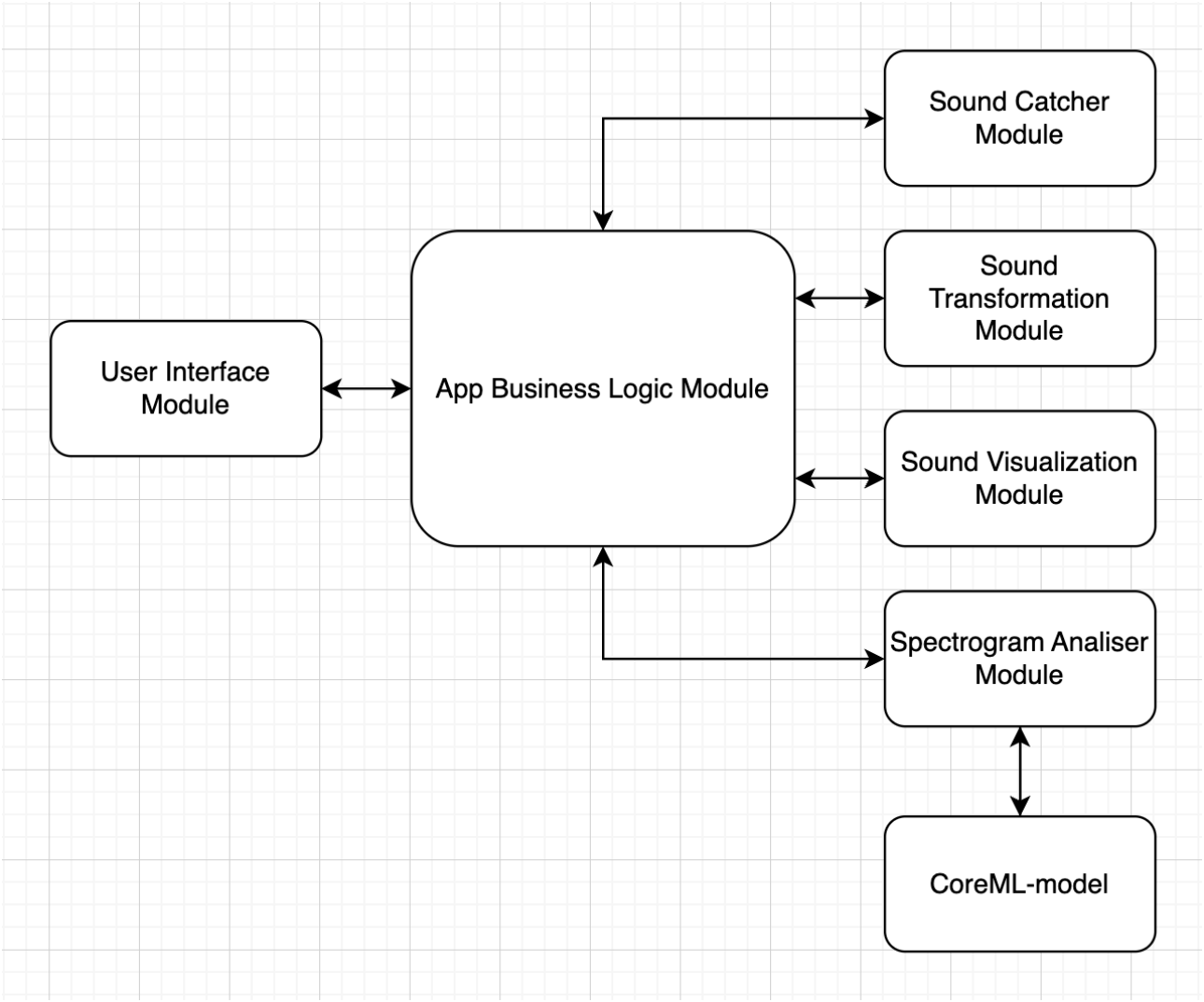

Рисунок 4.1 – Програмна модель мобільного додатку

4.3 Модуль бізнес логіки

Модуль бізнес-логіки – це основний модуль мобільного додатку. Він спілкується з іншими модулями та виконує певні операції, що обумовлені умовами додатку. За запитом користувача він ініціює запис звуку, після чого отримує від Core ML-моделі результат у вигляді пари значень класу звуку та

його ймовірності вірного розпізнавання. Далі результат відображається на екрані мобільного додатку.

Основним актором модуля бізнес логіки є класс ClappingDetector, який реалізує паттерн Mediator. Він організовує спілкування між класами-представниками інших модулів. Оскільки тільки один екземпляр классу ClappingDetector повинен існувати для коректної роботи застосунку, він також реалізує паттерн Singleton. Фрагмент коду з конструктором та полями класу ClappingDetector наведено у лістингу 4.1.

Лістинг 4.1 – Фрагмент коду з конструктором та полями класу ClappingDetector

```
final class ClappingDetector: NSObject {
    public static let shared = ClappingDetector()
    private(set) var isActive = false

    private let spectrogramAlalizer = SpectrogramAnalizer.shared
    private let audioProcessor = AudioInputProcessor()
    private let spectrogramCreator = MelSpectrogramCreator()

    private let spectrogramQueue = DispatchQueue(label:
"spectrogram",
                                                 qos: .userInitiated,
                                                 attributes:
.concurrent,
                                                 autoreleaseFrequency:
.workItem)

    override init() {
        super.init()
        commonInit()
    }

}

// MARK: - commonInit
private extension ClappingDetector {
    private func commonInit() {
        audioProcessor.delegate = self
    }
}
```

Для взаємодії з класом ClappingDetector використовується публічний інтерфейс, котрий складається з функцій start() та stop(). Код публічного інтерфейсу наведено у лістингу 4.2.

Лістинг 4.2 – Код публічного інтерфейсу ClappingDetector

```
// MARK: - public interface
extension ClappingDetector {
    func start() {
        isActive = true
        audioProcessor.start()
    }

    func stop() {
        isActive = false
        audioProcessor.stop()
    }
}
```

Після того, як користувач запустив роботу класу ClappingDetector шляхом виклика методу start(), ClappingDetector стартує виконання класу AudioInputProcessor, який є представником модуля трансформації звуку. Після цього ClappingDetector отримує оброблені аудіо дані через делегат AudioInputProcessorDelegate, реалізацію якого наведено у лістингу 4.3. Далі отримані аудіо дані передаються у модуль візуалізації звуку для створення Мел-спектрограми.

Лістинг 4.3 – Реалізація AudioInputProcessorDelegate класом ClappingDetector

```
// MARK: - confirm AudioInputProcessorDelegate protocol
extension ClappingDetector: AudioInputProcessorDelegate {
    func makeImageFrom(processedData values: [Float]) {
        spectrogramQueue.async {
            if let spectrogram =
self.spectrogramCreator.createAudioSpectrogram(from: values) {
                self.spectrogramAlalizer.analize(spectrogram)
            }
        }
    }
}
```

4.4 Модуль інтерфейсу користувача

Для взаємодії користувача та мобільного додатку використовується модуль інтерфейсу користувача. Цей модуль спроектовано за допомогою нативного фреймворку UIKit. За допомогою цього модулю користувач може розпочати або зупинити ідентифікацію звуків, отримати результат класифікації звуку від Core ML-моделі у вигляді зображення Мел-спектрограми, классу ідентифікованого звуку та ймовірності вірного розпізнання.

Головним класом модуля є SpectrogramViewController, який репрезентує головний екран мобільного додатку. Він налаштовує та розміщує на екрані такі елементи, як зображення Мел спектрограми melSpectrogram, кнопку запису recordingButton та лейбл з класом розпізнаного звуку і ймовірності його коректного розпізнавання predictionLabel. Фрагмент коду оголошення елементів користувацького інтерфейсу, їх налаштування та розміщення наведено у лістингу 4.4.

Лістинг 4.4 – Фрагмент коду оголошення елементів користувацького інтерфейсу, їх налаштування та розміщення

```
class SpectrogramViewController: UIViewController {
    private let melSpectrogram = SpectrogramLayer()
    private let recordingButton = UIButton()
    private let predictionLabel = UILabel()
    private var recordingButtonState: ActionButtonState = .start
}
// MARK: - predictionLabel
private extension SpectrogramViewController {
    private func setupPredictionLabel() {
        view.addSubview(predictionLabel)
        predictionLabel.text = "Prediction"
        predictionLabel.textColor = .black
        predictionLabel.textAlignment = .center
        predictionLabel.font = UIFont.systemFont(ofSize: 19,
weight: .black)
    }

    private func autoLayoutPredictionLabel() {
```

```
        predictionLabel.translatesAutoresizingMaskIntoConstraints
= false
        NSLayoutConstraint.activate([
            predictionLabel.leadingAnchor.constraint(equalTo:
view.leadingAnchor),
            predictionLabel.trailingAnchor.constraint(equalTo:
view.trailingAnchor),

predictionLabel.topAnchor.constraint(equalTo:view.safeAreaLayoutGu
ide.topAnchor, constant: 50),

predictionLabel.heightAnchor.constraint(equalToConstant: 44)
        ])
    }
}
// MARK: - startButton
private extension SpectrogramViewController {
    private func setupRecordingButton() {
        view.addSubview(recordingButton)

        recordingButton.backgroundColor = .black
        recordingButton.setTitleColor(.white, for: .normal)
        recordingButton.setTitle("Start recording", for: .normal)

        recordingButton.addTarget(self, action:
#selector(actionButtonHandler), for: .touchUpInside)
    }

    @objc private func actionButtonHandler() {
        updateActionButton()
        actionButtonPressed()
    }

    private func layoutRecordingButton() {
        let verticalSpacing: CGFloat = 88

        let width: CGFloat = 260
        let height: CGFloat = 44
        let x: CGFloat = (view.bounds.maxX - width ) / 2
        let y: CGFloat = view.bounds.maxY - verticalSpacing -
height
        recordingButton.layer.cornerRadius = height / 2
        recordingButton.frame = CGRect(x: x, y: y, width: width,
height: height)
    }
}
```

Зображення Мел-спектрограми та клас розпізнаного звуку SpectrogramViewController отримує від модуля аналізу Мел-спектрограми

шляхом реалізації делегату ImageDisplayer. Код реалізації делегату ImageDisplayer наведено у лістингу 4.5.

Лістинг 4.5 – Код реалізації делегату ImageDisplayer класом SpectrogramViewController

```
// MARK: - cofirm ImageDisplayer
extension SpectrogramViewController: ImageDisplayer {
    func display(_ image: CGImage, prediction:
SoundClassificationResult) {
        DispatchQueue.main.async {
            let stringPredicion = String(format:
"\(prediction.identifier): %0.4f",
                                         prediction.precision)
            self.predictionLabel.text = stringPredicion
            self.melSpectrogram.contents = image
        }
    }
}
```

Для передачі результату роботи CoreML-моделі викликається метод display(_image: CGImage, prediction: SoundClassificationResult), де image – зображення, а prediction – пара значень класу звуку та коректність його детекції.

4.5 Модуль захоплення звуку

Модуль захоплення звуку за допомогою фреймворку AVFoundation захоплює навколишній звук з мікрофону мобільного пристрою та повертає його у вигляді буфера аудіо-даних.

Головним класом модулю є AudioInputCapturePerformer. Оскільки він отримує чутливі дані з мікрофона, в край важливо забезпечити одночасне існування тільки одного екземпляру цього класу. Для цього AudioInputCapturePerformer реалізує паттерн Singleton.

Взаємодія модуля з бізнес логікою застосунку відбувається через публічний інтерфейс класу AudioInputCapturePerformer, який складається з

методів startCapture(handler: @escaping CaptureHandler) та stopCapture(). Параметром методу startCapture(_) є CaptureHandler – псевдоним функції зворотнього виклику, через яку передається буфер аудіо-даних до коду, що викликає startCapture(_). Публічний інтерфейс класу AudioInputCapturePerformer та значення CaptureHandler наведено у лістингу 4.6.

Лістинг 4.6 – Публічний інтерфейс класу AudioInputCapturePerformer та значення CaptureHandler

```
public typealias CaptureHandler = (_ buffer: CMSampleBuffer) ->
Void

// MARK: - public interface
extension AudioInputCapturePerformer {
    func startCapture(handler: @escaping CaptureHandler) {
        captureOutputHandler = handler
        sessionQueue.async {
            if AVCaptureDevice.authorizationStatus(for: .audio) ==
.authorized {
                self.captureSession.startRunning()
            } else {
                NSLog("Cannot start audio capture due to
authorization")
            }
        }
    }

    func stopCapture() {
        captureSession.stopRunning()
    }
}
```

Перед початком захоплення даних потрібно перевірити доступ до мікрофону. Якщо доступ не надано, робиться запит до системи на дозвіл доступу. Фрагмент коду з перевіркою доступу наведено у лістингу 4.7.

Лістинг 4.7 – Фрагмент коду з перевіркою доступу до мікрофону

```
private extension AudioInputCapturePerformer {
    private func checkAuthorizationForAudio() {
        switch AVCaptureDevice.authorizationStatus(for: .audio) {
```

```
            case .authorized:
                    break
            case .notDetermined:
                sessionQueue.suspend()
                AVCaptureDevice.requestAccess(for: .audio,
                                              completionHandler: {
granted in
                    if !granted {
                        NSLog("App requires microphone access.")
                    } else {
                        self.configureCaptureSession()
                        self.sessionQueue.resume()
                    }
                })
                return
            default:
            NSLog("App requires microphone access.")
        }
    }
}
```

Для виконання захоплення аудіо потрібно створити черги управління сесією та провести конфігурацію сесії захоплення аудіо-даних AVCaptureSession за викликом методу configureCaptureSession(). Код створення черг управління та функції configureCaptureSession() наведено у лістингу 4.9.

Лістинг 4.9 – Код створення черг управління та функції конфігурації сесії захоплення аудіо-даних

```
private let sessionQueue = DispatchQueue(label: "sessionQueue",
                                         attributes: [],
                                         autoreleaseFrequency:
.workItem)
private let captureQueue = DispatchQueue(label: "captureQueue",
                                         qos: .userInitiated,
                                         attributes: [],
                                         autoreleaseFrequency:
.workItem)


private func configureCaptureSession() {
        checkAuthorizationForAudio()

        captureSession.beginConfiguration()

        if captureSession.canAddOutput(audioOutput) {
            captureSession.addOutput(audioOutput)
```

```
        } else {
            NSLog("Can't add `audioOutput`.")
        }
        guard
            let microphone =
AVCaptureDevice.default(.builtInMicrophone,
                                                        for: .audio,
                                                        position:
.unspecified),
            let microphoneInput = try?
AVCaptureDeviceInput(device: microphone) else {
                NSLog("Can't create microphone.")
                return
        }

        if captureSession.canAddInput(microphoneInput) {
            captureSession.addInput(microphoneInput)
        }

        captureSession.commitConfiguration()
}
```

Під час конфігурації сесії до неї додається аудіо-вихід AVCaptureAudioDataOutput та аудіо-вхід AVCaptureDeviceInput. AVCaptureDeviceInput представляє собою мікрофон девайсу. AVCaptureAudioDataOutput робить AudioInputCapturePerformer делегатом, що отримує буфер сирих аудіо-даних з мікрофона від системи. Після отримання буферу даних, він передається у CaptureHandler. Фрагмент коду реалізації делегату AVCaptureAudioDataOutputSampleBufferDelegate наведено у лістингу 4.10.

Лістинг 4.10 – Фрагмент коду реалізації делегату AVCaptureAudioDataOutputSampleBufferDelegate

```
extension AudioInputCapturePerformer:
AVCaptureAudioDataOutputSampleBufferDelegate {
    public func captureOutput(_ output: AVCaptureOutput, didOutput
sampleBuffer: CMSampleBuffer, from connection:
AVCaptureConnection) {
        if let handler = captureOutputHandler {
            handler(sampleBuffer)
        }
     }
}
```

4.6 Модуль трансформації звуку

Після того, як було отримано буфер аудіо даних, цей буфер передається у модуль трансформації звуку. Тут буфер за допомогою алгоритму перетворення, що був описаний раніше, приводиться до вигляду, з якого буде створено зображення Мел-спектрограми.

Головним класом модулю є AudioInputProcessor. Взаємодія модуля з бізнес логікою застосунку відбувається через публічний інтерфейс, який складається з методів start() та stop(). Фрагмент коду публічного інтерфейсу наведено у лістингу 4.11.

Лістинг 4.11 – Фрагмент коду публічного інтерфейсу класу AudioInputProcessor

```
// MARK: - public interface
extension AudioInputProcessor {
    func start() {
        capturePerformer.startCapture { [weak self] buffer in
            guard let self = self,
                  let rawData = self.getRawAudioDataPtr(from:
buffer)
            else {
                NSLog("Cannot get raw audio data from
sambleBuffer")
                return
            }

            // add data to raw buffer
            self.addRawDataToBufferIfNeeded(rawData: rawData,
sampleBuffer: buffer)
            self.dispatchSemaphore.wait()

            // remove data from raw buffer and add data to
melSpectrogramValues
            self.processAudioDataFromBuffer()
            self.dispatchSemaphore.signal()
        }
    }

    func stop() {
        capturePerformer.stopCapture()
    }
}
```

Для того, щоб конвертувати буфер у спектрограму, потрібні специфічні параметри спектрограми, так як:

– кількість зразків звуку для одного фрейму аудіо даних – висота спектрограми;

– кількість буферів, що будуть відображатися – ширина спектрограми ;

– розмір стрибка, який контролює перекриття між фреймами даних і гарантує, що спектрограма не втрачає аудіо інформацію на початку та в кінці кожного зразка. Ці параметри наведені у лістингу 4.12.

Лістинг 4.12 – Фрагмент коду параметрів спектрограми

```
enum MelSpectrogramProperties {
    /// The number of audio samples per frame.
    static let sampleCount = 1024

    /// Determines the overlap between frames.
    static let hopCount = sampleCount / 4

    /// Number of displayed buffers — the width of the
spectrogram.
    static let bufferCount = 256

    /// The number of mel filter banks  — the height of the
spectrogram.
    static let filterBankCount = 64
    static let signalCount = 1

    static let maxProcessedData = 256 * 256
}
```

Буфер аудіо-даних отримується від класу AudioInputCapturePerformer через CaptureHandler. Після цього модуль конвертує буфер у зручний для обробки вигляд за допомогою методів getRawAudioDataPtr(from buffer: CMSampleBuffer) та addRawDataToBufferIfNeeded(rawData: UnsafeMutableRawPointer, sampleBuffer:CMSampleBuffer). Код конвертації буферу у зручний для обробки вигляд наведено у лістингу 4.13.

Лістинг 4.13 – Код конвертації буферу у зручний для обробки вигляд

```
// MARK: - audio data preparing
private extension AudioInputProcessor {
    private func getRawAudioDataPtr(from buffer: CMSampleBuffer) -
> UnsafeMutableRawPointer? {
        var audioBufferList = AudioBufferList()
        var blockBuffer: CMBlockBuffer?

        CMSampleBufferGetAudioBufferListWithRetainedBlockBuffer(
            buffer,
            bufferListSizeNeededOut: nil,
            bufferListOut: &audioBufferList,
            bufferListSize: MemoryLayout.stride(ofValue:
audioBufferList),
            blockBufferAllocator: nil,
            blockBufferMemoryAllocator: nil,
            flags:
kCMSampleBufferFlag_AudioBufferList_Assure16ByteAlignment,
            blockBufferOut: &blockBuffer)

        return audioBufferList.mBuffers.mData;
    }

    private func addRawDataToBufferIfNeeded(rawData:
UnsafeMutableRawPointer,

sampleBuffer:CMSampleBuffer) {

        if rawAudioBuffer.count <
MelSpectrogramProperties.sampleCount * 2 {
            let actualSampleCount =
CMSampleBufferGetNumSamples(sampleBuffer)

            let ptr = rawData.bindMemory(to: Int16.self, capacity:
actualSampleCount)
            let buf = UnsafeBufferPointer(start: ptr, count:
actualSampleCount)
            rawAudioBuffer.append(contentsOf: Array(buf))
        }
    }
}
```

Після препроцесінгу буферу у методі processAudioDataFromBuffer() береться певна кількість зразків звуку для одного фрейму. Від цих зразків віднімається розмір стрибка, який контролює перекриття між фреймами даних.

Далі починається обробка аудіо-даних для одного фрейму за допомогою функції processAudioData(data: [Int16]). Фрагмент коду функцій processAudioDataFromBuffer() та processAudioData(data: [Int16]) наведено у лістингу 4.14.

Лістинг 4.14 – Фрагмент коду функцій processAudioDataFromBuffer() та processAudioData(data: [Int16])

```
private func processAudioDataFromBuffer() {
        while self.rawAudioBuffer.count >=
MelSpectrogramProperties.sampleCount {
            let dataToProcess = Array(self.rawAudioBuffer[0 ..<
MelSpectrogramProperties.sampleCount])

self.rawAudioBuffer.removeFirst(MelSpectrogramProperties.hopCount)
            self.processAudioData(data: dataToProcess)
        }
    }

    private func processAudioData(data: [Int16]) {
        values (float). from data to timeDomainBuffer
        vDSP.convertElements(of: data,
                             to: &timeDomainBuffer)
        performForwardDFT(timeDomainValues: &timeDomainBuffer,
                                            frequencyDomainValues:
&frequencyDomainBuffer,
                                            temporaryRealBuffer:
&realParts,
                                            temporaryImaginaryBuffer:
&imaginaryParts)
        vDSP.absolute(frequencyDomainBuffer,
                      result: &frequencyDomainBuffer)
        frequencyDomainValues and filterBank. Result in
sgemmResult
        frequencyDomainBuffer.withUnsafeBufferPointer {
frequencyDomainValuesPtr in
            cblas_sgemm(CblasRowMajor,
                        CblasTrans, CblasTrans,

Int32(MelSpectrogramProperties.signalCount),

Int32(MelSpectrogramProperties.filterBankCount),

Int32(MelSpectrogramProperties.sampleCount),
                        1,
                        frequencyDomainValuesPtr.baseAddress,
Int32(MelSpectrogramProperties.signalCount),
```

```
                        filterBank.baseAddress,
Int32(MelSpectrogramProperties.sampleCount),
                        0,
                        sgemmResult.baseAddress,
Int32(MelSpectrogramProperties.filterBankCount))
        }

        var sgemm: [Float] = Array(sgemmResult)

        vDSP.convert(power: sgemm,
                     toDecibels: &sgemm,
                     zeroReference:
Float(MelSpectrogramProperties.sampleCount))
        if melSpectrumValues.count >
MelSpectrogramProperties.filterBankCount { // filterBankCount

melSpectrumValues.removeFirst(MelSpectrogramProperties.filterBankC
ount)
        }
        melSpectrumValues.append(contentsOf: sgemm)
        processedDataCounter += sgemm.count

        if processedDataCounter >=
(MelSpectrogramProperties.maxProcessedData) {
            delegate?.makeImageFrom(processedData:
melSpectrumValues)
            processedDataCounter = 0
        }
    }
```

Обробка починається з перетворення аудіо-даних до вигляду одноканального аудіосигналу в формат масиву Float чисел. Далі використовується метод performForwardDFT(), що готує та виконує пряме перетворення Фур'є, перетворюючи аудіодані з домену часу у домен частоти за допомогою FFT. Реалізацію методу performForwardDFT() наведено у лістингу 4.15.

Лістинг 4.15 – Реалізація методу performForwardDFT()

```
private extension AudioInputProcessor {
    private func performForwardDFT(timeDomainValues: inout
[Float],
                                   frequencyDomainValues: inout
[Float],
```

```
                                    temporaryRealBuffer: inout
[Float],
                                    temporaryImaginaryBuffer: inout
[Float]) {

        vDSP.multiply(timeDomainValues,
                      hanningWindow,
                      result: &timeDomainValues)

        temporaryRealBuffer.withUnsafeMutableBufferPointer {
realPtr in

temporaryImaginaryBuffer.withUnsafeMutableBufferPointer { imagPtr
in
                var splitComplex = DSPSplitComplex(realp:
realPtr.baseAddress!,
                                                   imagp:
imagPtr.baseAddress!)

                timeDomainValues.withUnsafeBytes {
                    vDSP_ctoz($0.bindMemory(to:
DSPComplex.self).baseAddress!, 2,
                              &splitComplex, 1,

vDSP_Length(MelSpectrogramProperties.sampleCount / 2))
                }
            }
        }

        temporaryRealBuffer.withUnsafeMutableBufferPointer {
realPtr in

temporaryImaginaryBuffer.withUnsafeMutableBufferPointer { imagPtr
in
                fftRealBuffer.withUnsafeMutableBufferPointer {
realBufferPtr in
                    fftImagBuffer.withUnsafeMutableBufferPointer {
imagBufferPtr in
                        var splitComplex = DSPSplitComplex(realp:
realPtr.baseAddress!,
                                                           imagp:
imagPtr.baseAddress!)

                        var bufferSplitComplex =
DSPSplitComplex(realp: realBufferPtr.baseAddress!,

imagp: imagBufferPtr.baseAddress!)

                        let log2n =
vDSP_Length(log2(Float(MelSpectrogramProperties.sampleCount)))

                        vDSP_fft_zript(fft,
                                       &splitComplex, 1,
```

```
                                        &bufferSplitComplex,
                                        log2n,

FFTDirection(kFFTDirection_Forward))
                    }
                }
            }
        }

        temporaryRealBuffer.withUnsafeMutableBufferPointer {
realPtr in

temporaryImaginaryBuffer.withUnsafeMutableBufferPointer { imagPtr
in
                var splitComplex = DSPSplitComplex(realp:
realPtr.baseAddress!,
                                                   imagp:
imagPtr.baseAddress!)

                frequencyDomainValues.withUnsafeMutableBytes { ptr
in
                    vDSP_ztoc(&splitComplex, 1,
                              ptr.bindMemory(to:
DSPComplex.self).baseAddress!, 2,

vDSP_Length(MelSpectrogramProperties.sampleCount / 2))
                }
            }
        }
    }
}
```

Для трансформування Фур’є використовуються тимчасові буфери реальних та уявних чисел, вікно Хенінга та сама функція трансформації Фур’є. Ці параметри наведені у лістінгу 4.16.

Лістінг 4.16 – Параметри трансформування Фур’є

```
private let hanningWindow = vDSP.window(ofType: Float.self,
                                        usingSequence:
.hanningDenormalized,
                                        count:
MelSpectrogramProperties.sampleCount,
                                        isHalfWindow: false)
private var fftRealBuffer = [Float](repeating: 0,
                                    count:
MelSpectrogramProperties.sampleCount / 2)
    private var fftImagBuffer = [Float](repeating: 0,
```

```
                                        count:
MelSpectrogramProperties.sampleCount / 2)
private let fft: FFTSetup = {
        let log2n =
vDSP_Length(log2(Float(MelSpectrogramProperties.sampleCount)))

        guard let fft = vDSP_create_fftsetup(log2n,
                                             FFTRadix(kFFTRadix2))
else {
            fatalError("Unable to create FFT.")
        }

        return fft
 }()
```

Після перетворення Фур’є отримуються абсолютні значення перетворення частоти в амплітуди аудіосигналу. Далі виконується множення двох матриць (представлення амплітуд частот та фільтрів). Це операція, яка застосовується для вирахування спектрограми аудіосигналу. Отримані результати перетворюються в децибели, що дозволяє краще аналізувати амплітуди звуку у вигляді логарифмічних одиниць. Значення, що відображають спектрограму звуку, оновлюються шляхом додавання нових результатів обробки до загального масиву спектральних даних. При досягненні максимальної кількості оброблених даних викликається метод makeImageFrom(processedData values: [Float]) делегата AudioInputProcessorDelegate, щоб створити зображення із отриманих спектральних даних. З викликом цього методу опрацьовані аудіо-дані направляються у модуль візуалізації звуку.

4.7 Модуль візуалізації звуку

Модуль візуалізації звуку використовується для перетворення трансформованого звуку, який репрезентується масивом значень типу даних Float , на зображення Мел-спектрограми.

Головним класом модулю є MelSpectrogramCreator. Взаємодія модуля з бізнес логікою застосунку відбувається через публічний інтерфейс, який

представлений методом createAudioSpectrogram(from values: [Float]) -> CGImage?. На вхід методу подається масив оброблених аудіо-даних, на виході з методу повертається зображення Мел-спектрограми. Код публічного інтерфейсу MelSpectrogramCreator наведено у лістингу 4.17.

Лістинг 4.17 – Код публічного інтерфейсу MelSpectrogramCreator

```
// MARK: - public interface
extension MelSpectrogramCreator {
    func createAudioSpectrogram(from values: [Float]) -> CGImage?
{

        let maxFloat =
sqrt(Float(MelSpectrogramProperties.sampleCount))
        let maxFloats: [Float] = [255, maxFloat, maxFloat,
maxFloat]
        let minFloats: [Float] = [255, 0, 0, 0]

        var melSpectrumValues = values

        melSpectrumValues.withUnsafeMutableBufferPointer {
            var planarImageBuffer = vImage_Buffer(data:
$0.baseAddress!,
                                                  height:
vImagePixelCount(SpectrogramImageSize.height),
                                                  width:
vImagePixelCount(SpectrogramImageSize.width),
                                                  rowBytes:
MelSpectrogramProperties.filterBankCount *
MemoryLayout<Float>.stride) //
MelSpectrogramProperties.filterBankCount

            vImageConvert_PlanarFToARGB8888(&planarImageBuffer,
                                            &planarImageBuffer,
                                            &planarImageBuffer,
                                            &planarImageBuffer,
                                            &rgbImageBuffer,
                                            maxFloats, minFloats,

vImage_Flags(kvImageNoFlags))

        }
        transformImageSourceWithPixels()
        rotateSourceImage()

        return try? rotatedImageBuffer.createCGImage(format:
MelSpectrogramCreator.rgbImageFormat)
    }
```

```
}
```

Після отримання масиву аудіо-даних створюється формат зображення з певними характеристиками, такими як кількість бітів на компонент, кількість бітів на піксель, простір кольорів. Далі створюються буфери зберігання зображення аудіоспектрограм у вертикальному та горизонтальному форматах. Ці буфери використовуються для маніпулювання зображенням під час його обробки. Фрагмент коду зі створенням формату зображення та буферів зберігання зображення наведено у лістингу 4.18.

Лістинг 4.18 – Фрагмент коду зі створенням формату зображення та буферів зберігання зображення

```
private static let rgbImageFormat: vImage_CGImageFormat = {
        return createImageFormat()
    }()

    private var rgbImageBuffer: vImage_Buffer = {
        return createImageBuffer(width:
SpectrogramImageSize.width,
                                 height:
SpectrogramImageSize.height)
    }()

    private var rotatedImageBuffer: vImage_Buffer = {
        return createImageBuffer(width:
SpectrogramImageSize.width,
                                 height:
SpectrogramImageSize.height)
    }()
// MARK: - fabric methods
private extension MelSpectrogramCreator {
    private static func createImageBuffer(width: Int, height: Int)
-> vImage_Buffer {
        guard let buffer = try? vImage_Buffer(width: width,
                                              height: height,
                                              bitsPerPixel:
MelSpectrogramCreator.rgbImageFormat.bitsPerPixel)
        else {
            fatalError("Unable to initialize image buffer.")
        }
        return buffer
    }
```

```
    private static func createImageFormat() ->
vImage_CGImageFormat {
        guard let format = vImage_CGImageFormat(
                bitsPerComponent: 8,
                bitsPerPixel: 8 * 4,
                colorSpace: CGColorSpaceCreateDeviceRGB(),
                bitmapInfo: CGBitmapInfo(rawValue:
CGImageAlphaInfo.first.rawValue),
                renderingIntent: .defaultIntent) else {
            fatalError("Can't create image format.")
        }
        return format
    }
}
```

Далі проводиться обробка буферу зображення шляхом конвертації планарних значень типу Float у формат ARGB8888 викликом методу transformImageSourceWithPixels(), що дозволяє представляти колір для кожного пікселя на зображенні. Код функції конвертації значень у формат ARGB8888 наведено у лістингу 4.19.

Лістинг 4.19 – Код функції конвертації значень у формат ARGB8888

```
private func transformImageSourceWithPixels() {
        vImageTableLookUp_ARGB8888(&rgbImageBuffer,
&rgbImageBuffer,
                                   nil,
                                   &ViridisColorMap.redTable,
                                   &ViridisColorMap.greenTable,
                                   &ViridisColorMap.blueTable,
                                   vImage_Flags(kvImageNoFlags))
}
```

Після цього виконується обробка кольорів буферу зображення та його обертання на 90 градусів проти годинникової стрілки викликом методу rotateSourceImage(). Фрагмент коду обертання зображення наведено у лістингу 4.20.

Лістинг 4.20 – Фрагмент коду обертання буферу зображення

```
private func rotateSourceImage() {
        vImageRotate90_ARGB8888(&rgbImageBuffer,
                                &rotatedImageBuffer,

UInt8(kRotate90DegreesCounterClockwise),
                                [UInt8()],
                                vImage_Flags(kvImageNoFlags))
}
```

Фінальним етапом є створення зображення CGImage з трансформованого буферу аудіо-даних з використанням формату зображення, який був наведений у лістингу 4.18.

4.8 Модуль аналізу Мел-спектрограми

Модуль аналізу Мел-спектрограми використовується як прошарок між модулем бізнес логіки та CoreML-моделлю.

Головним класом модулю є SpectrogramAnalizer. Для отримання спектрограми, яку потрібно проаналізувати, використовується публічний метод analize(_ image: CGImage), який наведено у лістингу 4.21 і параметром якого є зображення Мел-спектрограми.

Лістинг 4.21– Публічна функція аналізу зображення

```
public extension SpectrogramAnalizer {
    func analize(_ image: CGImage) {
            do {
                try self.makePredictions(for: image,
completionHandler: { predictions in
                  guard let prediction = predictions?.first else {
                        print("Prediction is not exist")
                        return
                    }
                    self.displayer?.display(prediction)
                    self.imageDisplayer?.display(image,
prediction: prediction)
                })
```

```
            } catch {
                print("Cannot make prediction due to:
\(error.localizedDescription)")
            }
    }
}
```

Після отримання спектрограми робиться “передбачення”. Передбаченням називається процес визначення класу звуку. Для того, щоб його провести, викликається функція makePredictions(for photo: CGImage, completionHandler: @escaping ImagePredictionHandler), яка наведена у лістингу 4.22 .

Лістинг 4.22 - Код функції проведення передбачення

```
private extension SpectrogramAnalizer {
    private func makePredictions(for photo: CGImage,
completionHandler: @escaping ImagePredictionHandler) throws {
        let imageClassificationRequest =
createImageClassificationRequest()
        predictionHandlers[imageClassificationRequest] =
completionHandler
        let handler = VNImageRequestHandler(cgImage: photo,
orientation: .up)
        let requests: [VNRequest] = [imageClassificationRequest]
        // Start the image classification request.
        try handler.perform(requests)
    }
}
```

За допомогою фреймворку Vision створюється запит на класифікацію звуку, який використовує обертку над CoreML-модель – ImageClassifier. Для цього викликається метод createImageClassificationRequest(), де потім налаштовується параметр обрізання та масштабування зображення спектрограми. Фрагмент коду зі створенням ImageClassifier та запиту на проведення передбачення наведено у лістингу 4.23.

Лістинг 4.23 – Фрагмент коду зі створенням ImageClassifier та запиту на проведення передбачення

```
static func createImageClassifier() -> VNCoreMLModel {
        let defaultConfig = MLModelConfiguration()
        let imageClassifierWrapper = try?
CustomMLModel(configuration: defaultConfig)

        guard let imageClassifier = imageClassifierWrapper else {
            fatalError("App failed to create an image classifier
model instance.")
        }
        let imageClassifierModel = imageClassifier.model
        do {
            let imageClassifierVisionModel = try
VNCoreMLModel(for: imageClassifierModel)
            return imageClassifierVisionModel
        } catch {
            print(error.localizedDescription);
            fatalError("App failed to create a `VNCoreMLModel`
instance.")
        }
    }

/// Generates a new request instance that uses the Image
Predictor's image classifier model.
    private func createImageClassificationRequest() ->
VNImageBasedRequest {
        let imageClassificationRequest = VNCoreMLRequest(model:
imageClassifier,

completionHandler: visionRequestHandler)
        imageClassificationRequest.imageCropAndScaleOption =
.centerCrop
        return imageClassificationRequest
    }

private func visionRequestHandler(_ request: VNRequest, error:
Error?) {
        guard let predictionHandler =
predictionHandlers.removeValue(forKey: request) else {
            fatalError("Every request must have a prediction
handler.")
        }
        var predictions: [SoundClassificationResult]? = nil
        defer {
            predictionHandler(predictions)
        }
        if let error = error {
            print("Vision image classification
error...\n\n\(error.localizedDescription)")
```

```
            return
        }
        if request.results == nil {
            print("Vision request had no results.")
            return
        }
        guard let observations = request.results as?
[VNClassificationObservation] else {
            print("VNRequest produced the wrong result type:
\(type(of: request.results)).")
            return
        }
        predictions = observations.map { observation in
            SoundClassificationResult(classification: observation)
        }
}
```

Далі створюється екземпляр обробника запитів класу VNImageRequestHandler який надається фреймворком Vision. У конструктор обробника передається зображення спектрограми та її орієнтація.

Створений запит передається до обробника запитів. Після того, як запит виконується, за допомогою інтерфейсу делегата ImageDisplayer результат аналізу Мел-спектрограми передається у модуль інтерфейсу користувача, де і буде відображеним. Інтерфейс ImageDisplayer наведено у лістингу 4.24.

Лістинг 4.24 – Інтерфейс ImageDisplayer

```
public protocol ImageDisplayer {
    func display(_ image: CGImage,
                 prediction: SoundClassificationResult)
}
```

4.9 Core ML-модель

Модель, яка проводить ідентифікацію звуку. Дана модель натренована на визначення одного та подвійного хлопків. Інші звуки модель навчена ідентифікувати як фон.

На вхід до моделі подається зображення Мел-спектрограми розміром 256 на 256 пікселів. Приклад вхідного зображення для ML-моделі наведено на рисунку 4.2. Після обробки зображення модель повертає словник, де ключами є класи розпізнанного звуку, а значеннями за ключем – ймовірність точного розпізнавання у виглядів чисельного значення від 0 до 1. Перелік класів ідентифікованого моделлю звуку наведено на рисунку 4.3, де Background – фон(тиша або будь-який звук, що не є хлопком), Clap – один хлопок, Double Clap – подвійний хлопок. Загальний інтерфейс CoreML-моделі наведено на рисунку 4.4.

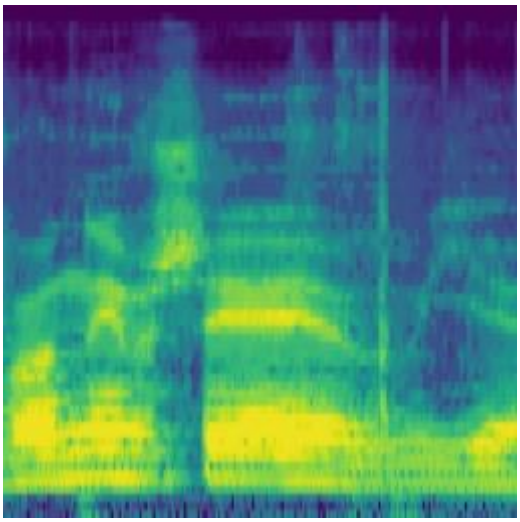

Рисунок 4.2 – Вигляд вхідного зображення для ML-моделі

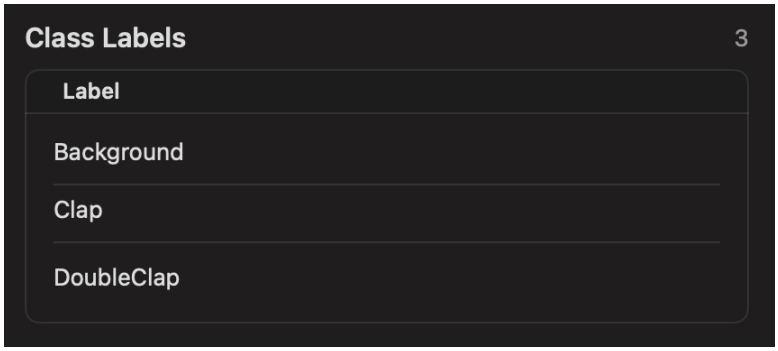

Рисунок 4.3 – Вигляд переліку класів звуку ML-моделі у Xcode

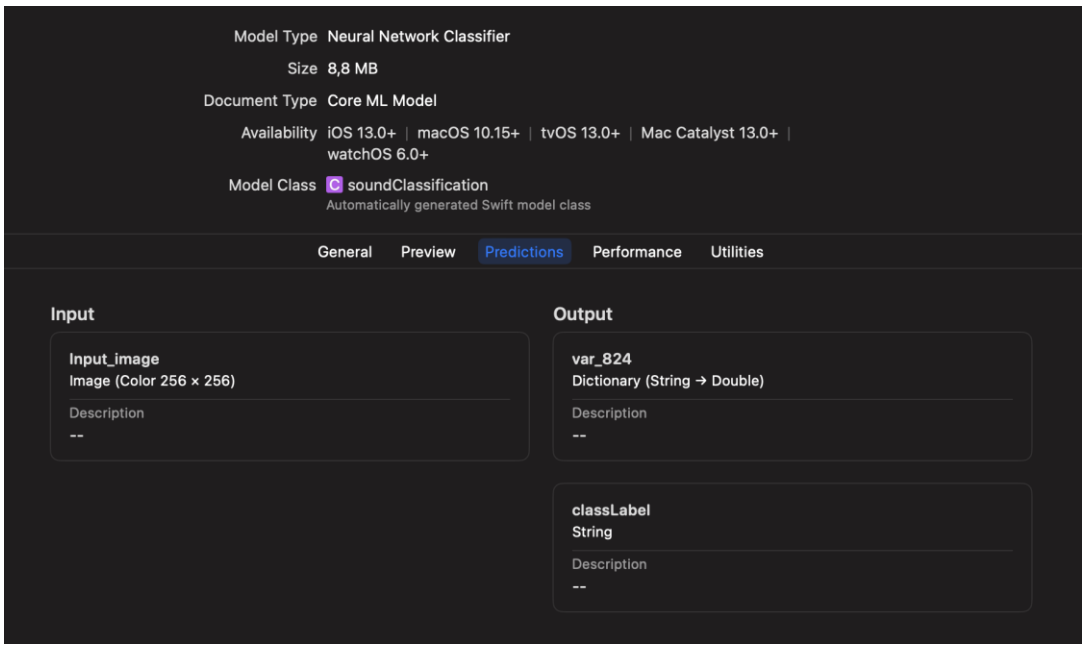

Рисунок 4.4 – Вигляд інтерфейсу ML-моделі у Xcode

5 ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ

Для початку тестування додаток було завантажено та встановлено на мобільному пристрої під керуванням операційної системи iOS версії 16.7.1.

Після запуску мобільного застосунку перед користувачем було відображено головний екран додатку. Головний екран складається з кнопки Start recording/Stop recording, зображення Мел-спектрограми, назви класу ідентифікованого звуку та ймовірності вірного визначення класу.

У головного екрану може бути два стани, що залежать від того, чи запускався процес ідентифікації звуку. Якщо запуску ідентифікації звуку ще не було, то замість назви класу звуку та ймовірності вірного розпізнавання буде відображатись “заглушка” у вигляді тексту Prediction. Зображення Мел-спектрограми буде відсутнє. Вигляд такого екрану наведено на рисунку 5.1.

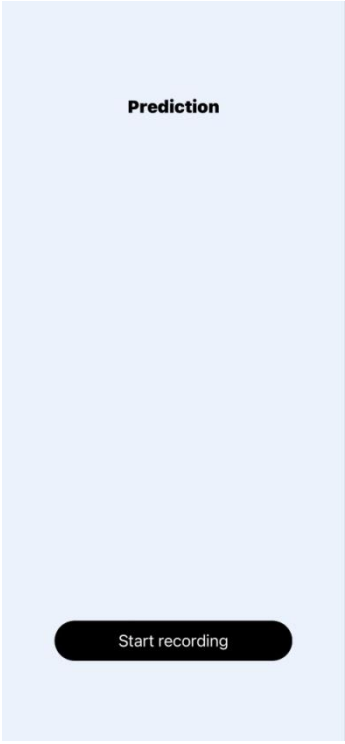

Рисунок 5.1 – Вигляд головного екрану при запуску додатку

Для початку роботи застосунку було натиснуто на кнопку Start recording, після чого почався захват та аналіз звуку з подальшим відображенням класу звуку та Мел-спектрограми.

Запуск ідентифікації звуку у додатку відбувся у кімнаті при відсутності сторонніх шумів. На рисунку 5.2 можна побачити, як на головному екрані з’явилось зображення Мел-спектрограми без яскравих жовто-зелених плям, що свідчить про тишу у навколишньому середовищі в моменті запису звуку для подальшої його обробки. Також замість “заглушки” Prediction з’явилася назва класу Background та ймовірність вірного розпізнання.

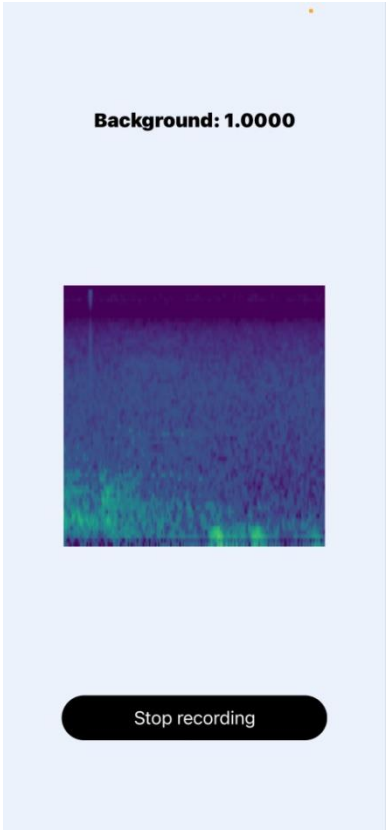

Рисунок 5.2 – Вигляд головного екрану після натискання на кнопку Start recording

Далі для тестування коректності роботи мобільного додатку було увімкнено аудіодоріжки з музикою та людською мовою. Результат розпізнавання наведено на рисунку 5.3 та рисунку 5.4

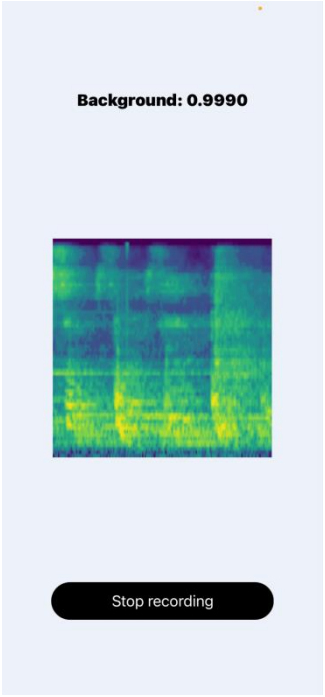

Рисунок 5.3 – Результат розпізнавання людської мови

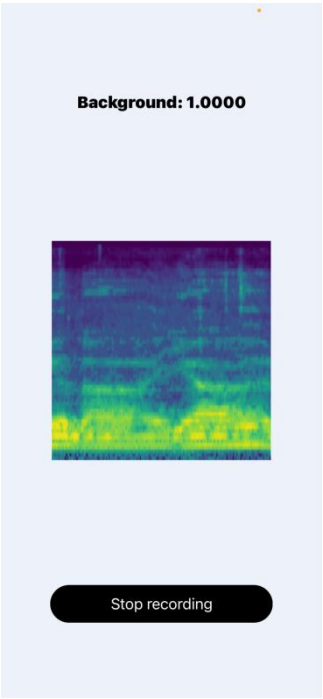

Рисунок 5.4 – Результат розпізнавання музики

Як можна побачити на рисунках 5.3 та 5.4, мова та музика були класифіковані моделлю, як Background з великою ймовірністю, тобто ці звуки не розпізнані як хлопки.

Далі було виконано почергове тестування ідентифікації одного на подвійного хлопків. На рисунках 5.5 та 5.6 наведено результат розпізнавання звуку, згідно якого можна зробити висновок, що мобільний додаток та ML-модель працюють коректно.

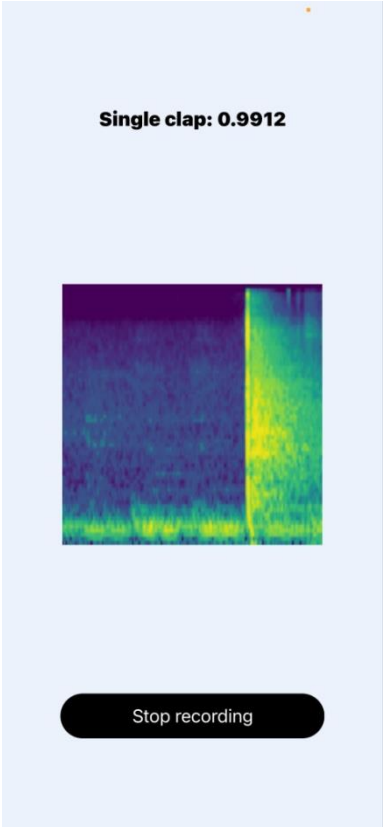

Рисунок 5.5 – Результат розпізнавання одного хлопку

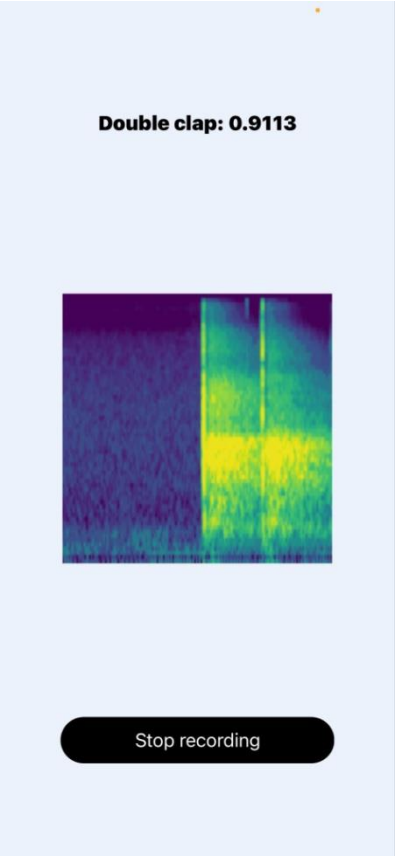

Рисунок 5.6 – Результат розпізнавання подвійного хлопку

ВИСНОВКИ

У даній кваліфікаційній роботі була розроблена система ідентифікації звуків за допомогою Machine Learning на платформі iOS. Для цього було проведено аналіз предметної області, після чого була поставлена задача по розробці мобільного додатку.

У ході роботи було розглянуто сфери застосування додатків, що проводять ідентифікацію звукових даних. Було приведене визначення звуку та проаналізовані основні параметри звуку, необхідні для його обробки та перетворення у спектральні характеристики. Також було розглянуто та порівняно спектрограму та Мел-спектрограму. Було надано огляд алгоритму перетворення звукових сигналів у Мел-спектрограму.

Процес створення додатку включав дослідження та використання інструментів розробки iOS, таких як AVFoundation, Core ML, Vision та UIKit.

Було розроблено програмну модель додатку та технічні вимоги до девайсу. Реалізація програмної частини проекту включала модулі для захоплення та обробки звуку, створення візуалізації та аналізу мел-спектрограми, модуль користувацького інтерфейсу.

Результатом роботи стало створення мобільного додатку, який, використовуючи CoreML-модель, може ідентифікувати звук навколишнього середовища, захоплений мікрофоном девайсу.

Також було проведено тестування системи. Процес тестування додатку підтвердив його ефективність та правильність роботи модулів, дозволяючи отримувати точні та надійні результати при аналізі звукової інформації.

ПЕРЕЛІК ПОСИЛАНЬ

1 How to Create & Understand Mel-Spectrograms [Електронний ресурс] / Режим доступу: www/ URL: https://importchris.medium.com/how-to-create-understand-mel-spectrograms-ff7634991056.

2 Getting to Know the Mel Spectrogram [Електронний ресурс] / Режим доступу: www/ URL: https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0.

3 Understanding the Mel Spectrogram [Електронний ресурс] / Режим доступу: www/ URL: https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53.

4 What is Spectrogram? [Електронний ресурс] / Режим доступу: www/ URL: https://emastered.com/blog/what-is-spectrogram?gad_source=1&gclid=CjwKCAiA-vOsBhAAEiwAIWR0TTFkMWt1xouUcWhHlCBmA_c2Ib7qBoHXo8btwmOXoEjVFLSg_eLUexoC4dsQAvD_BwE.

5 Core ML [Електронний ресурс] / Режим доступу: www/ URL: https://developer.apple.com/documentation/CoreML.

6 CoreML: A Quick Walkthrough [Електронний ресурс] / Режим доступу: www/ URL: https://medium.com/ibm-watson/coreml-a-quick-walkthrough-31b16b31719e.

7 Vision [Електронний ресурс] / Режим доступу: www/ URL: https://developer.apple.com/documentation/Vision.

8 Vision Framework in Swift for iOS Development [Електронний ресурс] / Режим доступу: www/ URL: https://www.bitcot.com/vision-framework-in-swift-for-ios-development/.

9 AVFoundation [Електронний ресурс] / Режим доступу: www/ URL: https://developer.apple.com/documentation/avfoundation.

10 UIKit [Електронний ресурс] / Режим доступу: www/ URL: https://developer.apple.com/documentation/uikit.

11 What is IOS operating system? [Електронний ресурс] / Режим доступу: www/ URL: https://digitaltechakshay.medium.com/what-is-the-ios-operating-system-b19c5d19f5bc.

12 Apple iOS [Електронний ресурс] / Режим доступу: www/ URL: https://www.investopedia.com/terms/a/apple-ios.asp.

13 iOS [Електронний ресурс] / Режим доступу: www/ URL: https://calvarybaptisthsv.org/wiki/IPhone_OS