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)

Модель ідентифікації звуків з використанням Data science

(тема)

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

Єфремова А. С._
(прізвище, ініціали)

Спеціальність *123 _ Комп'ютерна інженерія*

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

Освітня програма_ *Спеціалізовані*

_комп'ютерні _системи
(повна назва освітньої програми)

Керівник роботи _доцент Філіппенко І. В._
(посада, прізвище, ініціали)

Допускається до захисту

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

2023 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)

"____" _____ 20 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові _____Єфремовій Анастасії Сергіївні_____
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту) Модель ідентифікації звуків з використанням Data science

затверджена наказом по університету від "03"____11_____2023 р. № _1282 Ст_____

2. Термін подання студентом роботи до екзаменаційної комісії ____15.12.2023_____
3. Вихідні дані до роботи (проекту)_____

операційна система MacOS

Мова програмування Python

Середовище розробки Jupyter Lab

Середовище розробки Colab

4. Перелік питань, що потрібно опрацювати у роботі _____

Існуючі методи та алгоритми ідентифікації звуків

Способи візуалізації аудіоданих

Необхідні набори даних для задачі ідентифікації звуків

Вибір операційної клієнт-системи

Процес створення та тренування ML-моделі

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 18 слайдів_____

6. Консультанти розділів роботи (проекту)

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 (проекту)	Термін виконання етапів проекту (роботи)	Примітка
1	Видача теми проекту, узгодження і затвердження теми	02.09.2023 - 03.09.2023	
2	Аналіз існуючих методів розпізнавання жестів рук	02.09.2023 - 10.09.2023	
3	Дослідження алгоритмів розпізнавання жестів рук	02.09.2023 - 17.09.2023	
4	Адаптація та розробка методів розпізнавання жестів руки з використанням нейронних мереж	02.09.2023 - 25.09.2023	
5	Програмна реалізація системи нейронної мережі для розпізнавання жестів рук	02.09.2023 - 05.10.2023	
6	Тестування системи	02.09.2023 - 10.10.2023	
7	Оформлення пояснювальної записки	02.09.2023 - 20.10.2023	
8	Перевірка виконаного проекту керівником	02.09.2023 - 20.10.2023	
9	Захист проекту	02.09.2023 - 12.01.2024	

Студент _____ _____ _____
(під

Керівник роботи _____ _____ доц. Філіппенко І.В.

РЕФЕРАТ

Пояснювальна записка містить 68 сторінок, 23 рисунків, 14 лістингів, 6 джерел за переліком посилань.

ІДЕНТИФІКАЦІЯ ЗВУКІВ, DATA SCIENCE, MACHINE LEARNING

Під час виконання кваліфікаційної роботи було проведено аналіз процесу розробки ML-моделі для ідентифікації звуків, аналіз потенційних сфер застосування даної розробки.

Також було проведено аналіз та вибір інструментів, необхідних для розробки ML-моделі. На основі аналізу процесу розробки та предметної області, було створено детальний алгоритм процесу розробки ML-моделі та її навчання.

Також під час виконання кваліфікаційної роботи була створена ML-модель, була проведена підготовка даних для аудіо-класифікації, визначена архітектура ML-моделі, були підготовлені пакети даних, а також було проведено тренування та перевірка працездатності ML-моделі.

ABSTRACT

The practice report contains 68 pages, 23 figures, 14 listings, 6 sources according to the list of links.

SOUND IDENTIFICATION, DATA SCIENCE, MACHINE LEARNING

During the performance of the qualification work, an analysis of the process of developing an ML model for sound identification, an analysis of potential areas of application of this development was carried out.

The analysis and selection of tools necessary for the development of the ML model was also carried out. Based on the analysis of the development process and subject area, a detailed algorithm of the ML model development process and its training was created.

Also, during the qualification work, an ML model was created, data was prepared for audio classification, the architecture of the ML model was determined, data packages were prepared, and the ML model was trained and tested.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – Програмне забезпечення

UI – User interface

ML – Machine learning

MacOS – Операційна система для комп'ютерів корпорації Apple

iOS – Операційна система для мобільних пристроїв компанії Apple

ВСТУП

У сучасному світі нас оточують звукові сигнали, які можуть мати різноманітні джерела – від оточуючого звукового фону та сигналів тривоги до музики. Ці джерела можуть нести в собі важливу інформацію або сигнали, тому вміння їх ідентифікувати є важливим та актуальним завданням.

У теперішній час кількість акустичної інформації, яка виробляється та передається в різних галузях життя, значно збільшилася. Це можуть бути голосові повідомлення, музичні твори, шуми в виробничих приміщеннях, звукові сигнали в медичній галузі та інше. Усі ці звукові дані можуть містити корисну інформацію, яку можна використовувати для різних цілей, але для її отримання необхідно вміти ідентифікувати кожен звуковий сигнал.

Традиційні методи ідентифікації звукових сигналів можуть бути недостатньо точними, особливо при аналізі великих обсягів даних. Таким чином, використання методів Data science та створення і тренування ML-моделей може значно підвищити точність та швидкість ідентифікації звукових сигналів.

Розвиток технологій у галузі Data science та машинного навчання відкриває нові можливості для ідентифікації звукових сигналів в режимі реального часу.

Ідентифікація звуків може бути застосована в найрізноманітніших галузях, від медицини та біології до техніки та музики. Також ідентифікація звуків може бути корисною в різних галузях безпеки. Наприклад, звукові сигнали можуть використовуватися для виявлення незвичайних подій або поведінки, які можуть вказувати на загрози безпеки. Деякі приклади застосування ідентифікації звуків в безпеці.

Отже, ідентифікація звуків з використанням Data science, створення та тренування ML-моделей є актуальною темою, яка може призвести до

розвитку нових методів аналізу звукових даних та відкриття нових можливостей в різних галузях науки та техніки.

Метою кваліфікаційної роботи є дослідження можливостей створення та тренування ML-моделей для ідентифікації звуків з використанням методів Data science. Також метою є використання результатів дослідження для реалізації ML-моделі, яка буде здатна розрізняти певний звуковий сигнал.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Аналіз предметної області

Предметна область цієї теми є досить широкою, оскільки може мати різні області застосування, пов’язані із усіма областями життя сучасної людини, а саме із такими як медицина, наука, технології, безпека, музика тощо.

Однією з найважливіших задач в області ідентифікації звуків є створення моделей, які можуть розпізнавати звукові сигнали та визначати їх джерело з достатньою точністю. Якісно навчені моделі можуть бути використані в багатьох галузях, наприклад:

– безпека: ідентифікація звуків може використовуватися в даній галузі з метою виявлення подій або поведінки, які можуть вказувати на загрози безпеки, шляхом виявлення ключових звуків, що свідчать про небезпеку;

– медицина: в даній галузі ідентифікація звуків може використовуватися для діагностики різних захворювань, наприклад, шляхом аналізу звуку дихання пацієнта;

– наука: ідентифікація звуків може використовуватися в даній галузі як один з інструментів дослідження;

– технології: ідентифікація звуків може використовуватися як засіб управління різними пристроями.

Загалом, предметна область ідентифікації звуків з використанням Data science має безліч можливостей застосування в різних галузях. Також вона дозволяє підвищити ефективність та точність розпізнавання звукових сигналів і покращити якість життя користувачів.

1.2 Огляд існуючих методів та алгоритмів ідентифікації звуків та постановка завдання

На даний момент існують різні методи та алгоритми ідентифікації звуків, які можуть використовуватися в різних областях. Одними з найбільш поширених методів та алгоритмів є наступні:

Метод головних компонент (PCA) – це метод, який використовується для зменшення розмірності даних шляхом перетворення вихідних ознак в лінійну комбінацію головних компонент. Цей метод може використовуватися для вилучення найбільш важливих ознак з аудіо сигналів.

Метод дискримінантного аналізу (LDA) – це метод, який використовується для зменшення розмірності даних шляхом пошуку лінійної комбінації ознак, яка найкраще розділяє класи звуків.

Алгоритм динамічного програмування (DTW) – це алгоритм, який використовується для порівняння двох аудіо сигналів та пошуку оптимального відповідності між ними.

Алгоритми часових рядів – це алгоритми, які включають методи, такі як ковзне середнє, ковзне вікно, дискретне перетворення Фур'є та інші. Ці методи використовуються для аналізу часових характеристик аудіо сигналів.

З виникненням технологій машинного навчання та штучного інтелекту, навколишній світ постійно змінюється, а відкриті можливості для їх використання ще не досить досліджені. Одним з цікавих напрямків є ідентифікація звуків з використанням ML-моделей. Завдяки Data Science можна створювати системи, які матимуть можливість розпізнавати звуки довільного походження, такі як голос людини, музика, транспорт та багато інших.

Метою даної кваліфікаційної роботи є аналіз процесу розробки ML-моделі для ідентифікації звуків, що включає аналіз та вибір інструментів, необхідних для розробки такої моделі. На основі аналізу процесу розробки та

предметної області, необхідно створити детальний алгоритм процесу розробки ML-моделі та її навчання.

За результатами проведеного дослідження має бути створена ML-модель, яка здатна ідентифікувати певні звукові сигнали із достатньою точністю та функціонувати у реальних умовах. Зокрема, метою є аналіз потенційних сфер застосування даної розробки.

2 ТЕХНОЛОГІЇ ІДЕНТИФІКАЦІЇ ЗВУКІВ

2.1 Ідентифікація звуків

Ідентифікація звуків – це процес визначення типу звуку на основі його акустичних характеристик. У даній роботі задача ідентифікації звуків полягає в визначенні належності звуку до одного з попередньо заданих класів звуків.

Ідентифікація звуків є ключовим аспектом в області обробки звуку та класифікації звукових шаблонів. Цей процес може включати в себе визначення походження звуку, отримання характеристик звукових сигналів, аналіз змісту сигналу.

Сучасні технології, такі як методи машинного навчання та нейронні мережі, грають важливу роль у полі ідентифікації звуків, надаючи здатність до автоматизованого навчання та адаптації до нових звукових сценаріїв. Інші підходи можуть включати в себе використання цифрових сигналів, спектрального аналізу та методів обробки сигналів для витягування характеристик звукових подій.

2.2 Складнощі у процесі ідентифікації звуків

У процесі ідентифікації звуків виникає ряд складнощів, пов'язаних з реальними умовами оточуючого середовища та властивостями звукових сигналів. Перш за все, проблема полягає у тому, що звуки, які потрібно ідентифікувати, часто з'являються на тлі різноманітних шумів і впливів, що робить їхню коректну класифікацію та розпізнавання значно важчою задачею.

Додатковим викликом є варіабельність звукових характеристик, таких як інтонація, тембр, інтенсивність та часові параметри. Ці фактори можуть змінюватися від одного звукового джерела до іншого, ускладнюючи процес

створення ефективних алгоритмів ідентифікації. Тому для створення ефективної моделі машинного навчання для ідентифікації звуків необхідно провести ретельний аналіз вимог та зібрати достатню кількість даних для навчання та тестування моделі.

2.3 Візуалізація аудіоданих

Візуалізація аудіо-даних є важливим етапом при обробці звукових сигналів, сприяючи якіснішому розумінню та аналізу.

Візуальне представлення аудіоданих у вигляді графіків часового ряду або спектрограм надає змогу швидко оцінити часові та частотні характеристики звуку. Це дозволяє ідентифікувати особливості, такі як тривалість, інтенсивність, а також визначати наявність конкретних частотних компонентів.

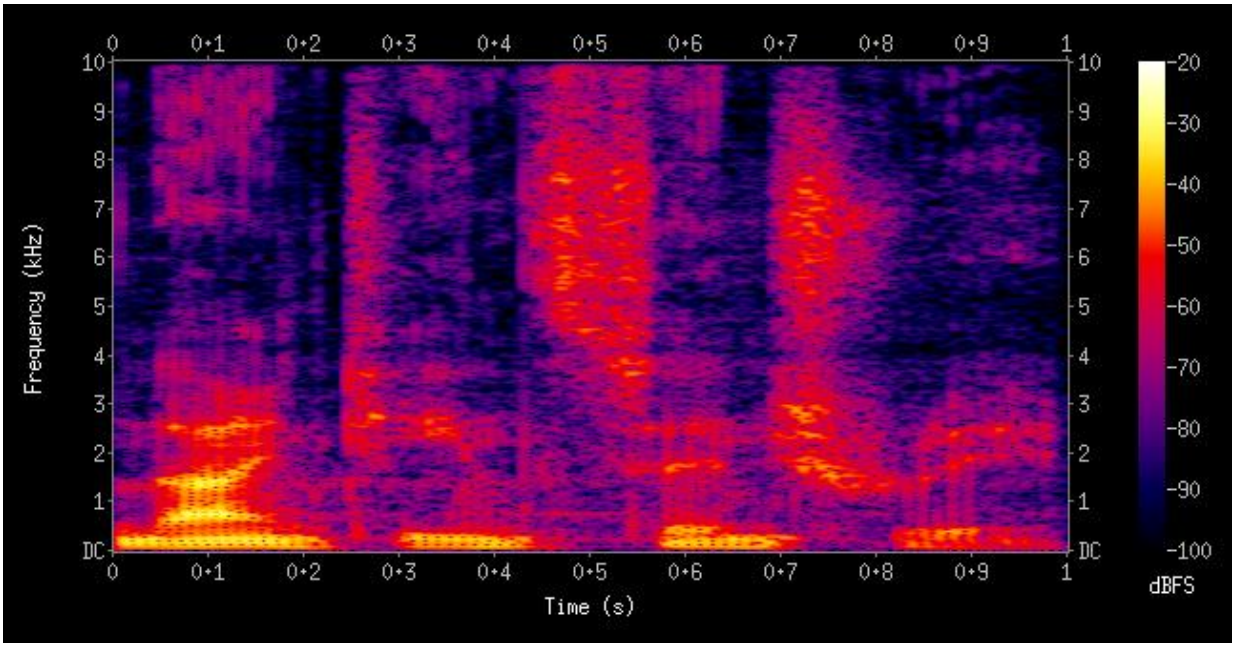


Рисунок 2.1 – Приклад візуалізації аудіоданих

Окрім цього візуалізація допомагає виявляти характеристики, які можуть бути невидимими на рівні “сирого” аудіосигналу. Це дозволяє

виявляти аномальні події, шаблони звуків, визначати особливості різних звукових джерел і враховувати їх у процесі ідентифікації.

Також аналіз візуальної динаміки аудіосигналу дозволяє виявляти та відслідковувати зміни в часі, такі як зміни в інтенсивності, частотному складі або інші зміни, які можуть бути важливими для розрізнення різних аудіо-подій.

2.3.1 Спектрограма

Спектрограма – це візуальне представлення часових та частотних характеристик аудіо сигналу. Вона дозволяє аналізувати спектральні складові звуку в залежності від часу. Спектрограма подається у вигляді 2D-зображення, де по горизонтальній осі відображається час, а по вертикальній – частота. Інтенсивність кольорів або яскравість на спектрограмі відображає амплітуду або енергію звукових компонентів.

Спектрограма може використовуватись для візуалізації звукових сигналів та отримання інформації про їх спектральний склад, а також вакористовуватись у якості інструмента аналізу звуків та мати ряд важливих застосувань:

– частотний аналіз: спектрограма надає можливість аналізувати, які частоти присутні в аудіосигналі протягом конкретного часового інтервалу. Вона візуально відображає, як змінюється частотний спектр звуку вздовж вісі часу, що дозволяє ідентифікувати характеристики звукових подій;

– розрізнення звукових джерел: спектрограма може допомагати в розрізненні звуків, які мають різні частотні характеристики;

– виявлення шумів та аномалій: спектрограма може слугувати для виявлення шумів, перешкод або інших аномалій у звуковому оточенні. Вона відображає інтенсивність частот в залежності від часу, що полегшує визначення нетипових або непередбачуваних звукових явищ;

– аналіз інтенсивності та тривалості звукових подій: аналіз спектрограми допомагає визначити тривалість та інтенсивність конкретних звукових подій, що може бути корисним у відстеженні часових параметрів аудіосигналів;

– аудіовізуальне співставлення: спектрограма може використовуватись для співставлення аудіосигналів з візуальними зображеннями, що допомагає в ідентифікації аудіо-патернів та подій.

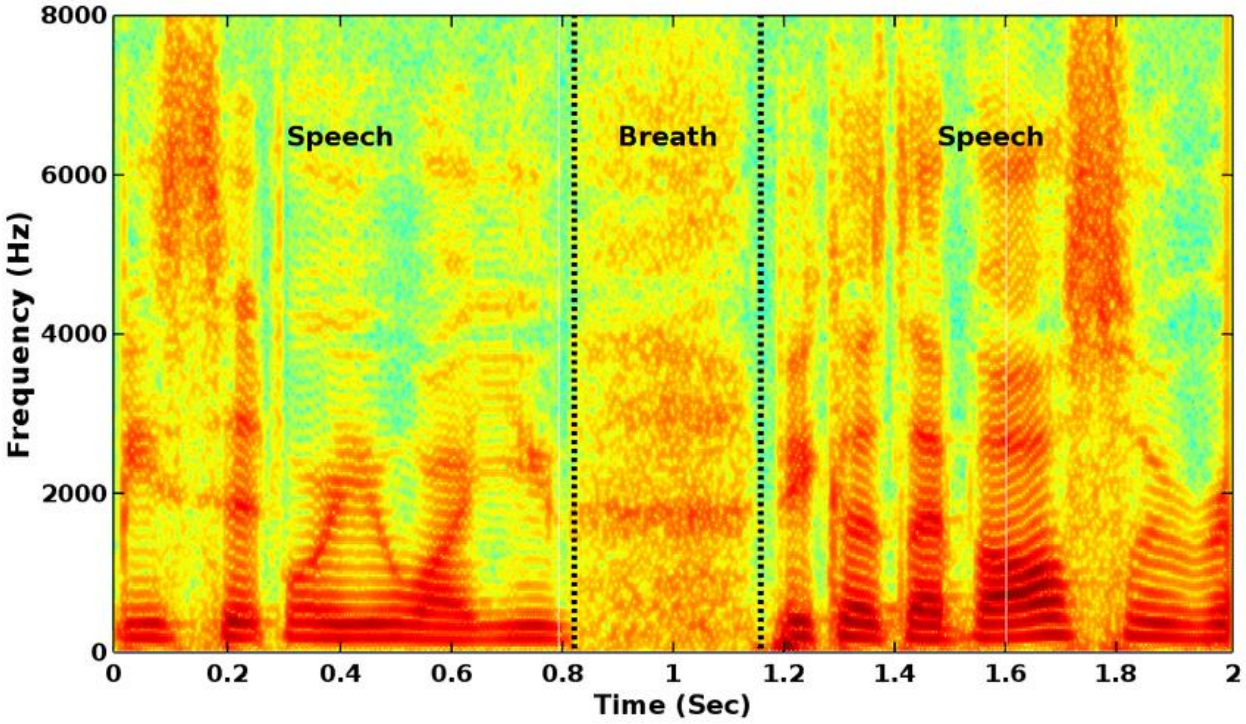


Рисунок 2.2 – Спектрограма людського мовлення та дихання

Спектрограми є важливим інструментом для аналізу звуків, виявлення патернів та відокремлення звукових класів у завданнях ідентифікації звуків з використанням методів Data Science та машинного навчання.

2.3.2 Мел-спектрограма

Мел-спектрограма – це візуальне представлення часових та мел-частотних характеристик аудіо сигналу. Вона використовує шкалу мела, яка

краще відтворює сприйняття людським вухом. Спектрограма мела використовується для аналізу аудіо даних, виявлення тонових характеристик та виокремлення важливих акустичних ознак.

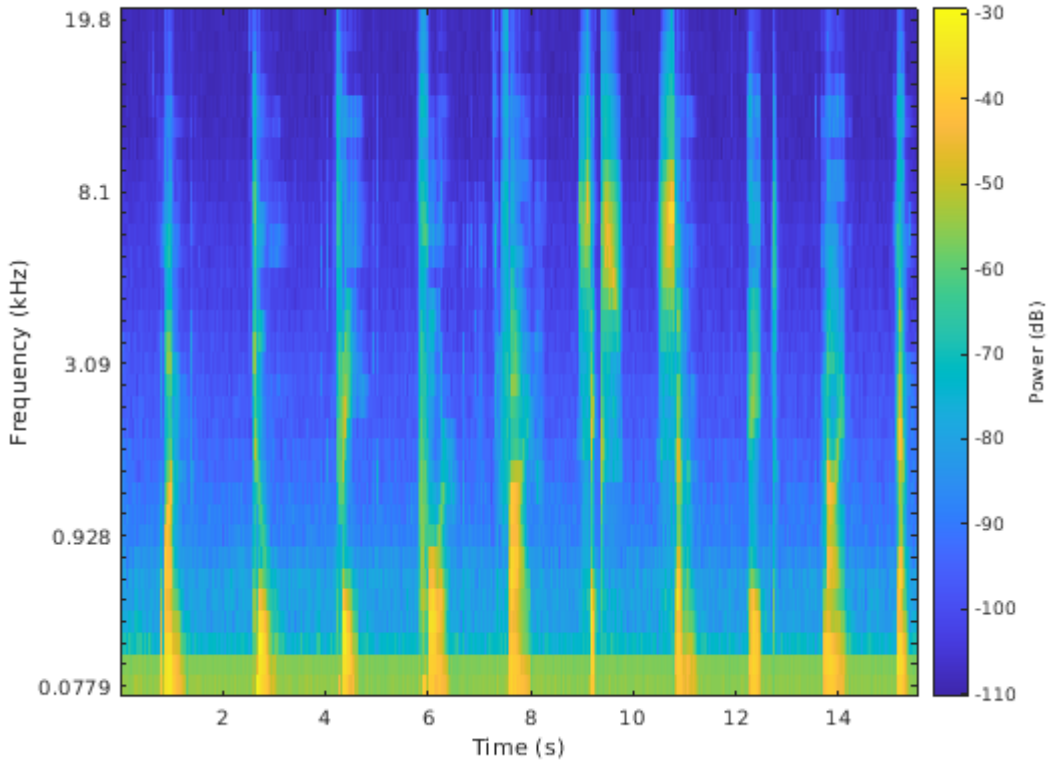


Рисунок 2.3 – Приклад вигляду спектрограми мела

Для отримання мел-спектрограми спершу аудіо сигнал розбивається на короткі фрагменти, над якими застосовується перетворення Фур'є для отримання спектру. Далі, застосовується мел-фільтрація, яка перетворює спектральні складові у шкалу мела. Потім виконується логарифмування амплітуд спектра та дискретизація на бінарні пікселі. Інтенсивність кольорів або яскравість на спектрограмі мела відображає енергію звукових компонентів у відповідних частотних діапазонах.

Спектрограма мела є потужним інструментом для аналізу звуків у багатьох областях, таких як розпізнавання мови, музичний аналіз, розпізнавання звукових подій та інші. Вона знайшла широке застосування у задачах ідентифікації звуків та покращення якості звукового сприйняття у

системах обробки звуку з використанням методів Data Science та машинного навчання.

2.3.3 Графіки хвильової форми

Візуалізація хвильової форми дозволяє відображати зміни амплітуди звукового сигналу у залежності від часу. Графіки хвильової форми дозволяють виявити основні характеристики звукового сигналу, такі як амплітуда, часова тривалість та частота.

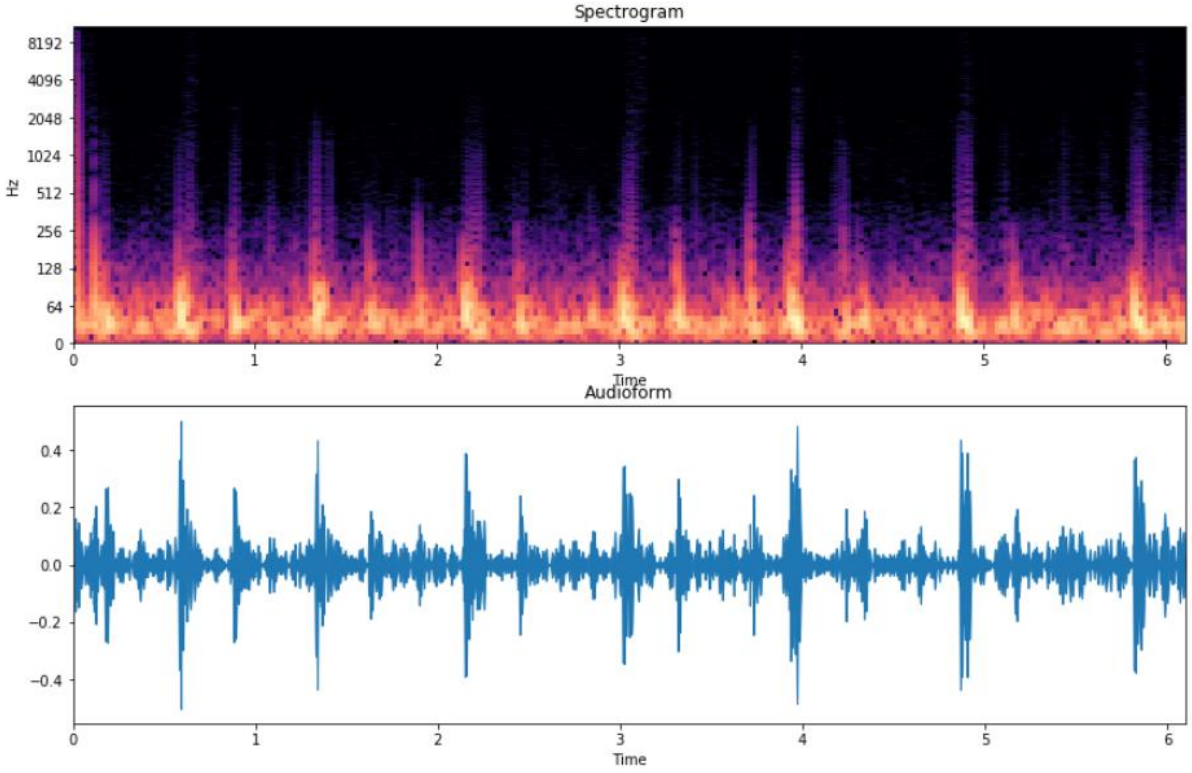


Рисунок 2.4 – Порівняння відображення звукового сигналу у форматі спектрограми та графіка хвильової форми

Графік хвильової форми показує, як змінюється амплітуда звуку с часом, у вигляді форми сигналу. Форма сигналу може мати різні види. Форма може бути симетричною або асиметричною, регулярною або нерегулярною.

Також за допомогою параметра частоти звукової форми можна визначити наявність основної частоти або частоного компоненту в звуковому сигналі.

3 РОЗРОБКА АЛГОРИТМУ АНАЛІЗУ ЗВУКІВ З ВИКОРИСТАННЯМ DATA SCIENCE

3.1 Аналіз необхідних наборів даних для задачі ідентифікації звуків

Для створення алгоритму системи необхідно провести ретельний аналіз необхідних наборів даних для задачі ідентифікації звуків з використанням Data Science.

Для створення та тренування ML-моделі для ідентифікації звуків необхідно мати в наявності достатній обсяг якісних та репрезентативних даних, що включають різноманітні типи звуків, які планується ідентифікувати.

Для початку роботи із data set необхідно визначити область звуків дослідження. Це можуть бути природні звуки, звуки транспорту, музичні інструменти або будь-яка інша область, що досліджується.

Після цього кроку необхідно провести збір аудіо даних, які мають включати в себе різноманітні варіації звуків з обраної області. Дані можна зібрати з різних джерел, таких як відкриті бази даних, записи з мікрофонів, аудіозаписи з Інтернету або спеціально створені набори даних.

Після збору даних потрібно анотувати їх, тобто надати мітки або класифікацію кожному звуку в наборі даних. Наприклад, класифікувати звуки за типом або поділити на категорії та підкатегорії за іншими характеристиками.

Після анотування необхідно провести перевірку якості даних та їх очищення. Це включає видалення артефактів, шуму або помилкових анотацій, а також видалення дублікатів або непотрібних записів. Окрім очищення даних необхідно привести звукові дані до загальної форми. Від цього кроку залежить якість ідентифікації звукових даних.

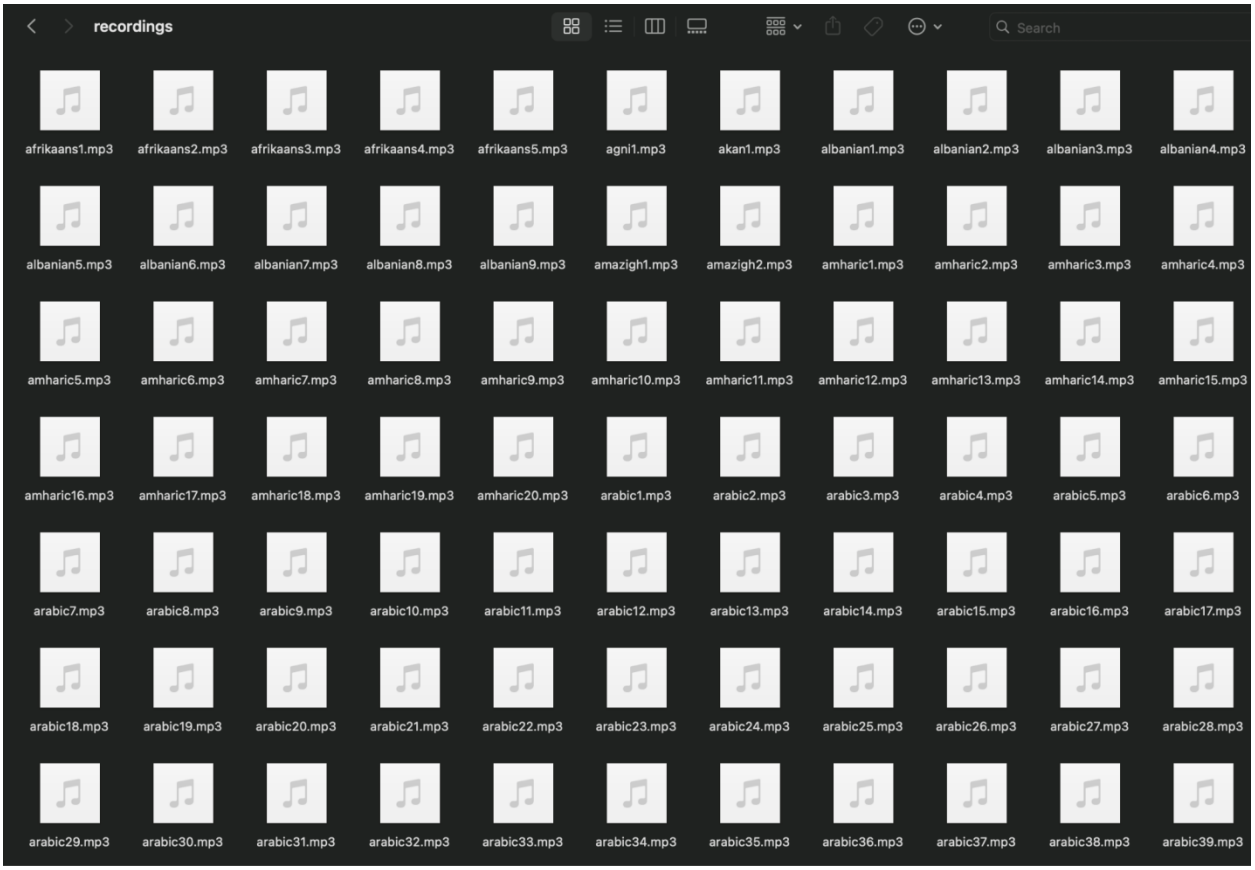


Рисунок 3.1 – Приклад зібраного data set

Для ефективного тренування та оцінки моделі наступним кроком має бути розбиття даних на тренувальний, валідаційний та тестовий набори. Зазвичай, більша частина даних використовується для тренування моделі, менша – для валідації та налаштування гіпер параметрів, а окремий набір даних залишається для оцінки фінальної ефективності моделі. Також важливо враховувати розмір тренувального та тестового наборів даних. Набір даних повинен бути достатньо великим для ефективного тренування моделі та забезпечення репрезентативності різних типів звуків.

Перед початком тренування моделі необхідно провести аналіз якості даних. Виявлення аномалій, шуму або інших проблем може допомогти покращити точність моделі.

3.2 Аналіз формату вхідних даних для ML-моделі

Для створення продуктивної системи аналізу звуків необхідно провести аналіз і визначити найкращий варіант формату даних. Модель може приймати у якості вхідних даних різні формати такі як аудіо дані або зображення спектрограми.

Аудіо формат у якості вхідних даних дозволяє працювати з оригінальними звуковими сигналами із всіма їх акустичними особливостями, а також можуть зберігати детальну інформацію про часові та частотні характеристики звуку. Проте даний формат також має свої недоліки, такі як:

– великий розмір даних порівняно із іншими форматами, що може вимагати більшої обчислювальної потужності для обробки та тренування моделі;

– даний формат потребує попередньої обробки та витрат на аудіозапис, аналіз та екстрагування акустичних ознак.

Дані у форматі зображення спектрограми навпаки є компактним форматом. Даний формат також зберігає інформацію про часові та частотні характеристики звуку. Також перевагою даного формату є той факт, що існує широкий набір інструментів та бібліотек для роботи з обробкою зображень. Недоліком даного формату є втрата невеликої деталізації та точності порівняно з оригінальним аудіо.

Після аналізу вище перелічених факторів було визначено, що дані у форматі зображення спектрограми мають більше переваг і їхня втрата деталізації є незначною. У зв'язку з цим, у якості оптимальних вхідних даних для ML-моделі був обраний формат зображення.

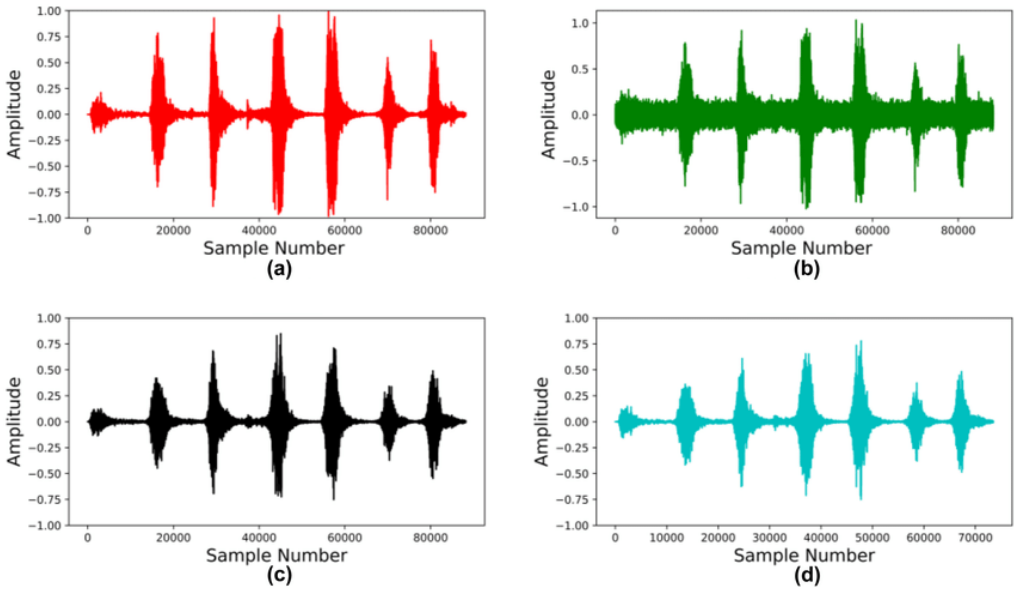


Рисунок 3.2 – Data set із даними у форматі wave form

3.3 Використання мел-спектрограми в Data Science

Спектрограма мела широко використовується в задачах машинного навчання і аналізу звуку з кількох причин:

– людське вухо сприймає звук на основі шкали мела, яка не однаково розподіляє частоти відповідно до сприйняття звуку. Використання мел-спектрограми дозволяє краще відтворити сприйняття звуку людським вухом і врахувати особливості аудіо з точки зору людського сприйняття;

– мел-спектрограма може бути представлена у вигляді 2D-зображення, що значно спрощує обробку та аналіз звуку. Це дозволяє використовувати потужні алгоритми та інструменти, розроблені для обробки зображень, для роботи з аудіо даними;

– спектрограма мел містить інформацію про часові та мел-частотні характеристики звуку, яка є важливою для багатьох задач аналізу звуку. Вона відображає розподіл енергії звукових компонентів у різних частотних діапазонах та дозволяє виявляти важливі акустичні ознаки, такі як форманти у мові або музичні характеристики у музичних записах;

– дана спектрограма широко використовуються у наукових дослідженнях та практичних застосуваннях в області обробки звуку та машинного навчання. Існує багато відкритих даних та ресурсів, які надають спектрограми мела для тренування моделей та порівняння результатів.

Загалом, використання мел-спектрограми у задачах машинного навчання дозволяє отримати компактне та інформативне представлення аудіо даних, враховуючи особливості сприйняття звуку людським вухом і використовуючи потужні методи обробки зображень.

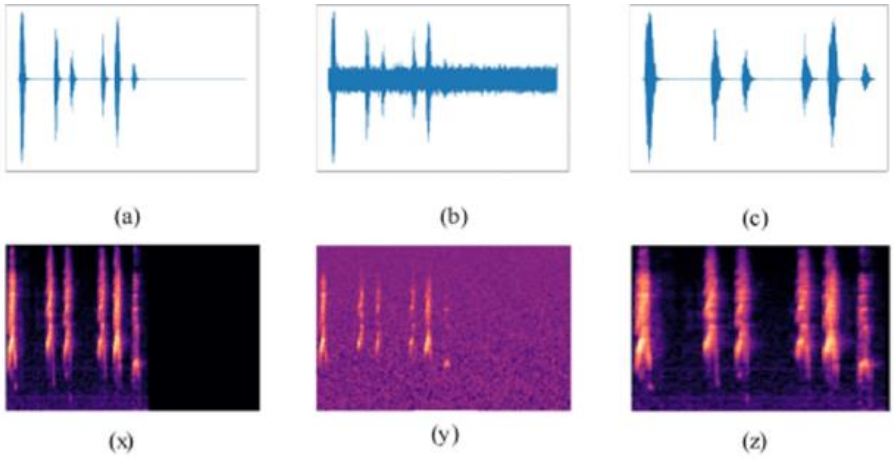


Рисунок 3.3 – Зображення неопрацьованих відображень waveform та трансформованих у мел-спектрограму

3.4 Алгоритм створення мел-спектрограми

Для перетворення аудіо-даних до виду спектрограми мела необхідно перетворити аудіо-дані у амплітудний годинниковий сигнал. Для виконання цього завдання за допомогою мови програмування Python можна використовувати вже існуючі бібліотеки.

Однією з таких бібліотек є librosa. Ця бібліотека надає різні інструменти для аналітики аудіо даних. За допомогою методу librosa.load() даної бібліотеки можна завантажити аудіофайл, який буде перетворено на амплітудний годинниковий сигнал.

Далі має бути виконано розбиття сигналу на короткі фрагменти. Аудіосигнал зазвичай розбивають на короткі фрагменти, називані рамками. Кожна рамка має визначену довжину та переміщується починаючи з початку аудіо-сигналу до його кінця. Розмір кожної рамки може бути налаштований залежно від специфікацій вашого дослідження.

Наступним кроком для кожної рамки застосовується короткочасне перетворення Фур'є, яке розкладає сигнал на його спектральні складові. Це можна зробити за допомогою функції librosa.stft(). Після застосування STFT методу отримуємо комплексне представлення спектрограми. Для отримання магнітуди спектрограми використовується функція numpy.abs(), яка обчислює амплітуду комплексних значень.

Для отримання мел-спектрограми магнітуда спектрограми замінюється на основі фільтра мел-шкали. Фільтр мел-шкали враховує специфічні особливості сприйняття звуку людським вухом. Це можна зробити за допомогою функції librosa.feature.melspectrogram().

Отриману мел-спектрограму можна піддати логарифмуванню, застосувавши логарифм до кожного значення в матриці мел-спектрограми. Це допомагає зменшити динамічний діапазон значень та поліпшити розрізненість між різними звуками.

Дані кроки дозволяють отримати мел-спектрограму, яка може бути використана як вхідні дані для моделі класифікації звуків.

У випадку інтегрування моделі у мобільний застосунок, або у програму призначену для будь-якого іншого клієнту, модулем, що відповідає за перетворення звукових сигналів у мел-спектрограму, має бути окремий актор реалізації клієнта.

Розглянемо приклад роботи моделі із застосунком для операційної системи iOS. Перенесення процесу форматування звукових сигналів на бік клієнту, у цьому випадку iOS-додатку, має суттєву перевагу.

Виконання операцій на боці пристрою дозволяє використовувати його локальні ресурси, що позитивно вплине на продуктивність і відгук програми.

Мобільні пристрої системи iOS обладнані спеціальними апаратними компонентами для обробки аудіо, і використання їх може забезпечити швидше форматування звукових даних.

Окрім цього, обробка звукових сигналів на боці iOS додатку мінімізує можливі несправності пов'язані із передачею, записаних за допомогою iOS пристрою сигналів, в інтегровану у додаток ML-модель.

Також у випадку реалізації форматування сигналів на пристрої iOS, модуль ML-моделі має єдину відповідальність: класифікація звуків, у той час, як iOS додаток відповідає за підготовлення звуків для класифікації, а саме їх збір так форматування.

3.5 Формування алгоритму аналізу звукових сигналів за допомогою Data Science

Після аналізу необхідних наборів даних для ML-моделі та їх формату було визначено, що найкращим варіантом вхідних даних є візуальне відображення звукового сигналу у вигляді мел-спектрограми. Даний спосіб візуалізації звукових даних поєднує в собі високу точність подання звуку, при цьому маючи досить низькі необхідні ресурси. Так само даний вид спектрограми дозволяє позбутися надмірних для аналізу даних через те, що мел-спектрограма відображає лише діапазон, який сприймається людськими органами слуху.

Після того як був визначений вид вхідних даних моделі, можна виконати збір аудіо-даних, на основі яких буде проводитися навчання моделі. Для збільшення даних дата сету також можна скористатися платними та безкоштовними базами дата сетів.

Наступним кроком необхідно зробити перетворення аудіо дата сету на форму дата сету з мел-спектрограмами. Для виконання даного кроку може бути застосований раніше описаний в даному проекті алгоритм.

Далі необхідно виконати розбиття даних на тренувальний та тестовий набори. Тренувальний набір використовується для навчання моделі, тоді як тестовий набір залишається для оцінки ефективності моделі після тренування.

Після завершення описаних раніше кроків необхідно виконати побудову моделі класифікації. Для побудови моделі класифікації можна використати різні алгоритми машинного навчання, наприклад, нейронні мережі, дерева рішень, метод опорних векторів (SVM) тощо. Нейронні мережі, зокрема сверточні нейронні мережі (CNN), часто показують добрі результати в розпізнаванні звуків за допомогою мел-спектрограм.

Наступним кроком має бути тренування моделі. Модель класифікації тренується на тренувальному наборі мел-спектрограм. В процесі тренування модель навчається розпізнавати характеристики різних звуків та прогнозувати їх класифікацію. Після тренування моделі необхідно оцінити її ефективність за допомогою тестового набору мел-спектрограм. Метрики, такі як точність, чутливість, специфічність та F-міра, можуть бути використані для оцінки результатів класифікації.

Після тренування та оцінки модель може бути використана для ідентифікації нових звуків. Для використання моделі необхідно завантажити новий звуковий сигнал, обчислити його мел-спектрограму та передати її до моделі для прогнозування класу звуку. Завдяки мові програмування Python та цей процес може бути автоматизованим.

3.6 Визначення умов роботи ML-моделі

Для можливості тестування сформованого алгоритму створення та тренування ML-моделі та подальшої його реалізації необхідно обрати звукові сигнали які має ідентифіковувати модель, а також клієнт-систему у яку має інтегруватися створена модель.

В якості звукових сигналів які мають бути ідентифіковані моделлю були обрані звуки одиночного клапку та подвійного. Ці звуки легко відтворити для тестування роботи ML-моделі. Також для даних сигналів може легко бути зібране обширний data source, достатній для повноцінного тренування ML-моделі.

Для вибору клієнт-системи було проведено аналіз систем доступних для реалізації застосунку із інтегрованою ML-моделлю. У якості клієнт-системи було обрано операційну систему iOS.

3.7 Вибір мобільної операційної системи

IOS Apple – це операційна система для iPhone, iPad та інших мобільних пристроїв Apple. Заснована на Mac OS, операційній системі, яка керує лінійкою настільних і портативних комп'ютерів Mac від Apple, Apple IOS розроблена для легкого безперебійного встановлення мережі між рядом продуктів Apple.

IOS має інтуїтивно зрозумілий дизайн, орієнтований на користувача, і можливість для розробників додатків створювати програми, які поширюються через магазин додатків IOS.

Одним із основних переваг розробки додатків для iPhone є високорівневий рівень безпеки, який надається системою. Це означає, що користувачі iPhone ефективно захищені від шкідливих програм, вірусів та інших загроз, які регулярно вмішуються в повсякденні операції.

IOS забезпечує захист від:

– дублювання даних;
– шифрування даних;
– крадіжки даних.

Основною аудиторією користування системи IOS є технологічна аудиторія, яка перебуває в пошуках простого, але ефективного інтерфейсу.

Перевага розробки додатків для iPhone полягає в тому, що існуючі користувачі вже звикли і задоволені синхронізованим середовищем пристроїв Apple.

Реалізація додатка із ML-моделлю для операційної системи iOS має зокрема наступну перевагу: єдина програмна реалізація може бути використана не тільки для мобільних iOS-пристроїв, а також для пристроїв iPad та MacBook із процесорами архітектури arm. Зокрема, програмна реалізація iOS-додатку може бути використана для MacBook із Intel x86-64 архітектурами за умови впровадження певних змін у кодо-базі.

4 РОЗРОБКА ML-МОДЕЛІ

4.1 Процес створення ML-моделі

Процес створення моделі машинного навчання можна поділити на чотири основні етапи, які включають збір даних, перетворення, навчання та оцінку якості передбачення.

Збір даних включає в себе визначення завдання та потреби у даних. Цей процес починається із чіткого визначення завдання, яке модель повинна вирішити, і встановлення потреб у відповідних даних для цього завдання.

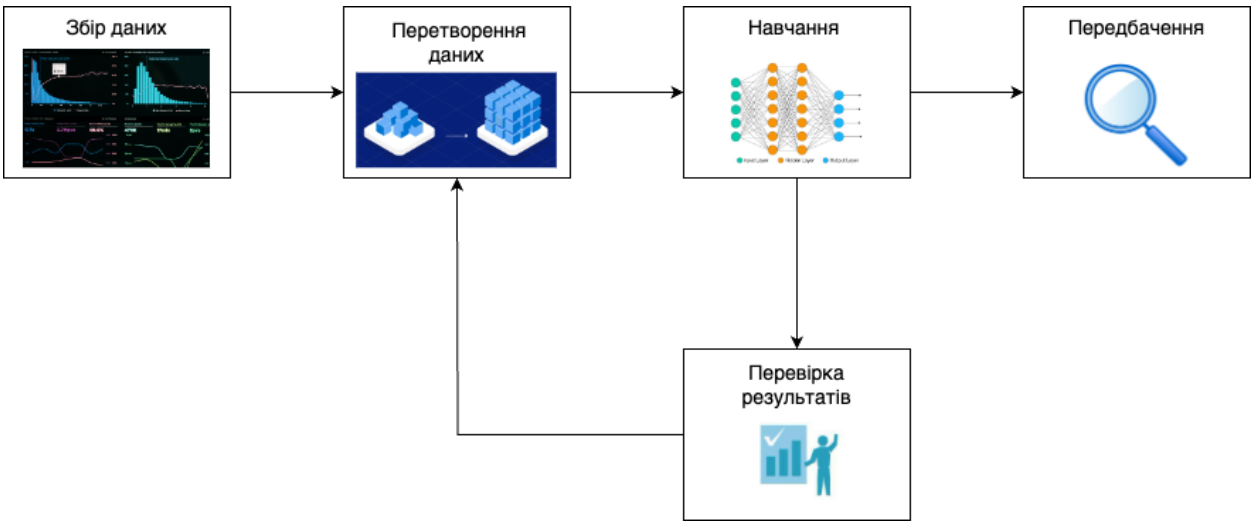


Рисунок 4.1 – Процес створення ML-моделі

Далі проводиться здійснення процесу збору необхідних даних для тренування та оцінки моделі. Це може включати в себе використання різних джерел, таких як бази даних, API, чи ручний ввід.

Перетворення включають у себе очистку та обробку даних, а саме проведення аналізу даних для виявлення аномалій, відсутніх значень або інших несправностей, а також виконання операцій їх очищення.

Далі виконується форматування – приведення даних до необхідного формату та вибір характеристик, які будуть використовуватися моделлю.

Наступним кроком є навчання моделі. Цей процес включає в себе вибір архітектури моделі відповідно до завдання, яке вирішується. Також під час навчання моделі проводиться розділення даних на тренувальний та тестовий набори для оцінки ефективності моделі.

Далі відбувається запуск процесу навчання, під час якого модель коригує свої ваги на основі тренувальних даних.

Після описаних кроків наступним іде етап оцінки якості передбачення. Даний етап поділяється на описані далі етапи. Тестування моделі, а саме, застосування моделі до тестового набору для оцінки її відповідності реальним даним. Оцінка ефективності – визначення метрик якості, таких як точність, відновлення, F-міра тощо. Підгонка та оптимізація – в необхідних випадках, внесення корекцій до моделі або її параметрів для покращення результатів.

На кроці перевірки оцінки передбачення може бути результат роботи моделі незадовільним, такий проміжуточний результат розробки моделі є нормальним, та означає те, що має відбутись повернення на минулий етап процесу розробки. Задля забезпечення задовільного результату роботи моделі, після повторного навчання, мають бути внесені певні зміни у цьому процесі. Декілька можливих кроків, які може бути вжито для покращення ефективності та точності моделі:

– ретельний аналіз даних – необхідно виконати перевірку якості та розподілу даних. Можливо, є пропуски в даних, аномалії або неспівпадіння між тренувальним та тестовим наборами;

– перегляд гіперпараметрів моделі – можна провести експерименти із значеннями гіперпараметрів моделі щоб знайти оптимальні значення;

– модифікація архітектури – необхідно переглянути архітектуру моделі. Можливо, потрібно змінити кількість шарів, розмір шарів, додати чи вилучити блоки залежно від завдання;

– збільшення обсягу даних – необхідно, за можливості, збільшити обсяг тренувального набору або використати методи аугментації даних, щоб додати різноманітності;

– зміна вагів класів – при наявності незбалансованих класів, можна розглянути встановлення вагів для класів для покращення обчислення втрат.

Загалом, можливі різні варіанти змін у моделі, при отриманні невалідного результату тренування моделі. Вони залежать від особливостей конкретної моделі.

На основі опису процесу створення моделі був побудований workflow подальшої роботи для отримання поставлених цілей роботи.

Рисунок 4.2 – Workflow створення моделі розпізнавання звуків

Результатом розробки ML-моделі має бути приведеного далі (рис. 4.2) образу система.

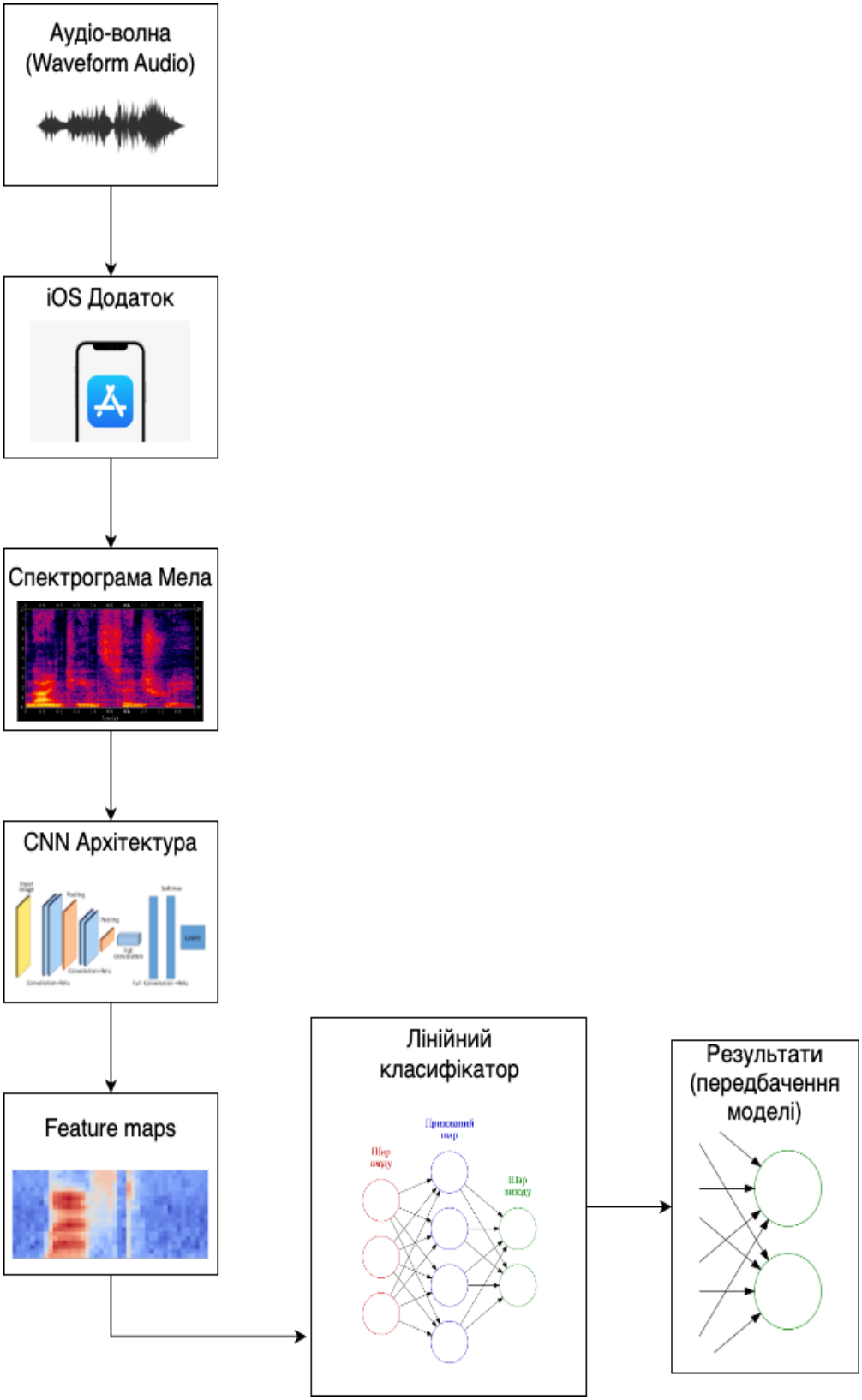


Рисунок 4.3 – Схема архітектури системи розпізнавання звуків

4.2 Підготовка даних для аудіо-класифікації

Першим кроком у створенні будь-якої ML-моделі є збір даних, на основі яких буде відбуватись навчання майбутньої моделі. Цей етап є критичним для досягнення успішних результатів. Правильно зібрані та адекватно підготовлені дані служать фундаментом для розуміння моделлю закономірностей в даних та вивчення взаємозв'язків між вхідними характеристиками та цільовою змінною.

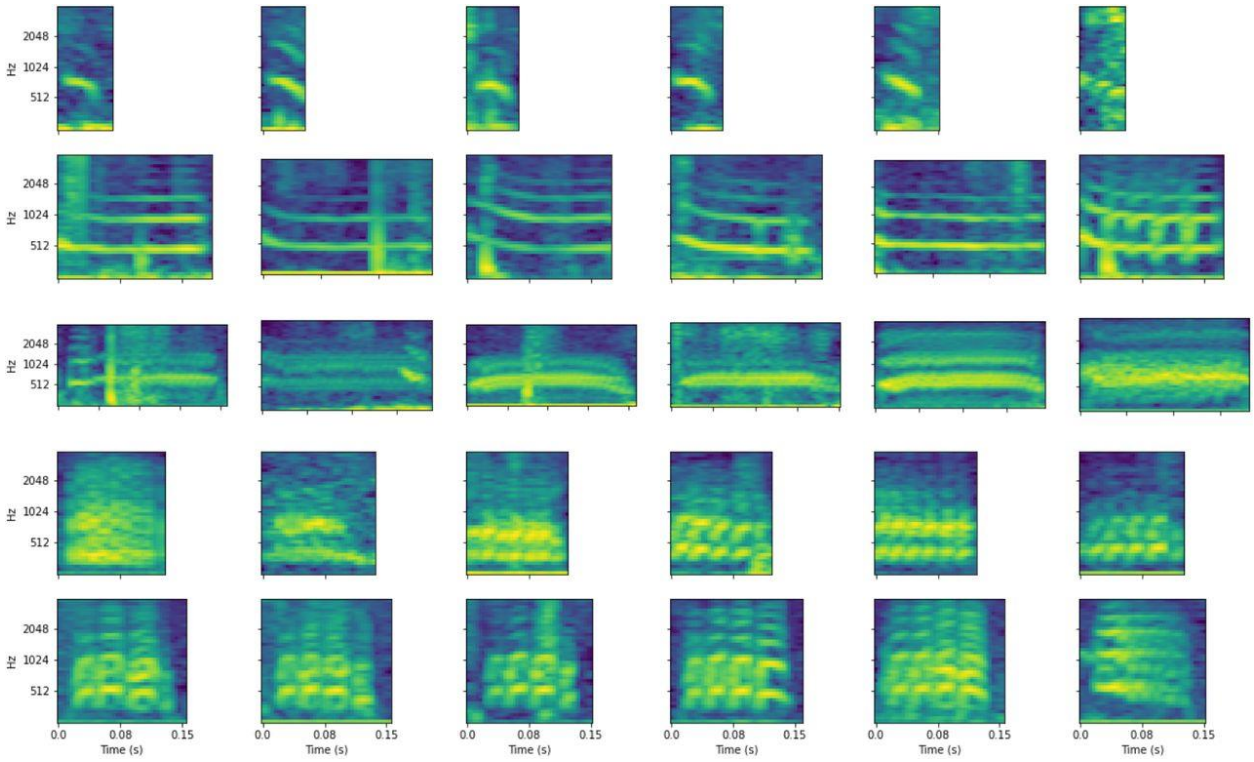
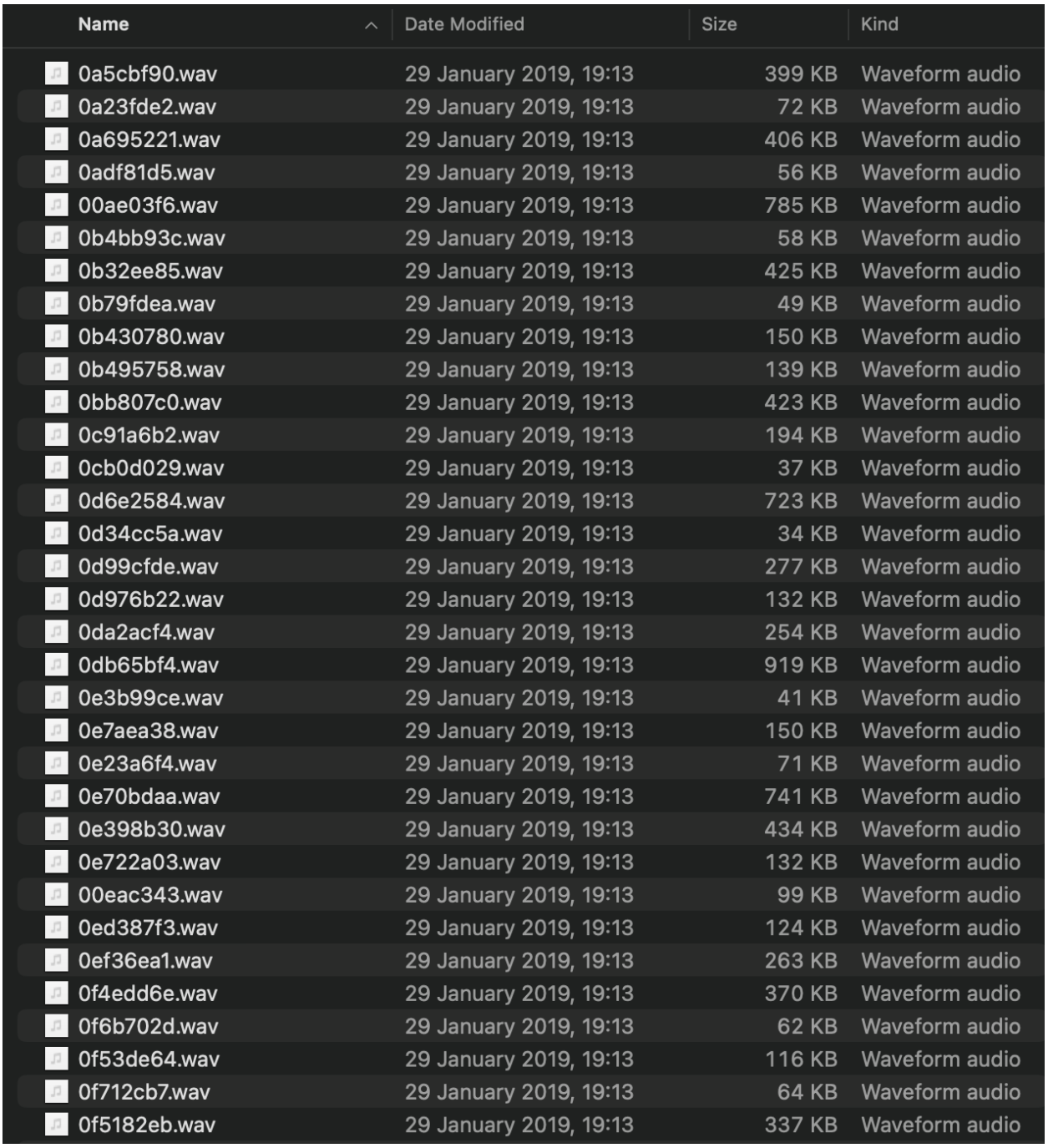


Рисунок 4.4 – Dataset мел-спектрограм

Під час збору даних важливо враховувати такі аспекти, як розмір вибірки, репрезентативність даних, узгодженість формату та якість інформації. Збір даних може включати в себе процеси обробки та очищення для усунення неточностей, аномалій та забезпечення їх придатності для моделювання.

Звуками обраними для ідентифікації були звуки подвійних хлопків та одинарних. Перевагою обраних звуків є те, що для тренування моделі легко зібрати відповідний датасет.

Для більшої репрезентативності датасет був зібраний за допомогою інтернет-ресурсів та ручного запису аудіо-сигналів.



Name	Date Modified	Size	Kind
0a5cbf90.wav	29 January 2019, 19:13	399 KB	Waveform audio
0a23fde2.wav	29 January 2019, 19:13	72 KB	Waveform audio
0a695221.wav	29 January 2019, 19:13	406 KB	Waveform audio
0adf81d5.wav	29 January 2019, 19:13	56 KB	Waveform audio
00ae03f6.wav	29 January 2019, 19:13	785 KB	Waveform audio
0b4bb93c.wav	29 January 2019, 19:13	58 KB	Waveform audio
0b32ee85.wav	29 January 2019, 19:13	425 KB	Waveform audio
0b79fdea.wav	29 January 2019, 19:13	49 KB	Waveform audio
0b430780.wav	29 January 2019, 19:13	150 KB	Waveform audio
0b495758.wav	29 January 2019, 19:13	139 KB	Waveform audio
0bb807c0.wav	29 January 2019, 19:13	423 KB	Waveform audio
0c91a6b2.wav	29 January 2019, 19:13	194 KB	Waveform audio
0cb0d029.wav	29 January 2019, 19:13	37 KB	Waveform audio
0d6e2584.wav	29 January 2019, 19:13	723 KB	Waveform audio
0d34cc5a.wav	29 January 2019, 19:13	34 KB	Waveform audio
0d99cfde.wav	29 January 2019, 19:13	277 KB	Waveform audio
0d976b22.wav	29 January 2019, 19:13	132 KB	Waveform audio
0da2acf4.wav	29 January 2019, 19:13	254 KB	Waveform audio
0db65bf4.wav	29 January 2019, 19:13	919 KB	Waveform audio
0e3b99ce.wav	29 January 2019, 19:13	41 KB	Waveform audio
0e7aea38.wav	29 January 2019, 19:13	150 KB	Waveform audio
0e23a6f4.wav	29 January 2019, 19:13	71 KB	Waveform audio
0e70bdaa.wav	29 January 2019, 19:13	741 KB	Waveform audio
0e398b30.wav	29 January 2019, 19:13	434 KB	Waveform audio
0e722a03.wav	29 January 2019, 19:13	132 KB	Waveform audio
00eac343.wav	29 January 2019, 19:13	99 KB	Waveform audio
0ed387f3.wav	29 January 2019, 19:13	124 KB	Waveform audio
0ef36ea1.wav	29 January 2019, 19:13	263 KB	Waveform audio
0f4edd6e.wav	29 January 2019, 19:13	370 KB	Waveform audio
0f6b702d.wav	29 January 2019, 19:13	62 KB	Waveform audio
0f53de64.wav	29 January 2019, 19:13	116 KB	Waveform audio
0f712cb7.wav	29 January 2019, 19:13	64 KB	Waveform audio
0f5182eb.wav	29 January 2019, 19:13	337 KB	Waveform audio

Рисунок 4.5 – Dataset waveform завантажений з Інтернету

Для ручного збору аудіо файлів використовувалась програма диктофон для клієнта MacOS.

Після підготування аудіо-датасету необхідно виконати перетворення формату waveform до формату мел-спектрограми, так як iOS клієнт буде передавати дані у модель у форматі спектрограми Мела. Для цієї мети необхідно підготувати реалізацію скрипта, який буде виконувати зазначене перетворення.

4.3 Визначення архітектури ML-моделі

Для створення моделі необхідно визначити яка архітектура має використовуватись. В контексті машинного навчання архітектура моделі описує структуру та конфігурацію моделі. Архітектура визначає, як компоненти мережі з'єднуються між собою та як вони обробляють вхідні дані для вирішення конкретного завдання.

Основні складові архітектури моделі можуть включати подані далі компоненти:

– вхідні шари визначають, як дані подаються у модель. Наприклад, для зображень це може бути двовимірний шар пікселів, а для тексту - вектори слів;

– приховані шари: містять нейрони, які виконують обчислення для вирішення завдання. Різні типи шарів можуть використовуватися в залежності від завдання, такі як повністю з'єднані шари, шари зриву та згорткові шари;

– вихідні шари: представляють результати моделі;

– функції активації: визначають, які значення передаються між нейронами;

– функції втрат: визначають, як вимірюється різниця між прогнозами моделі та фактичними значеннями;

– оптимізатори та функції вдосконалення: визначають, як модель навчається та адаптується до даних. Оптимізатори визначають процес оптимізації ваг моделі під час навчання;

– параметри та гіперпараметри: параметри моделі змінюються під час навчання. Гіперпараметри, навпаки, задаються заздалегідь і визначають структуру та поведінку моделі.

При виборі архітектури враховується мета завдання, кількість та характеристики даних, обчислювальні можливості та інші фактори. Для завдань ідентифікації звуків на основі мел-спектрограм важливо використовувати архітектури, які можуть ефективно розпізнавати патерни у часово-частотному просторі аудіосигналів. Однією з часто використовуваних архітектур для цього є Convolutional Neural Network (CNN) або комбінації CNN і Recurrent Neural Network (RNN). Є декілька можливих варіантів архітектур.

Convolutional Neural Networks (CNNs) або згорткові нейронні мережі є спеціалізованими архітектурами нейронних мереж, які широко використовуються для обробки зображень та відео. CNN може виявляти просторові патерни у мел-спектрограмі, що може бути корисно для розпізнавання звуків.

Recurrent Neural Networks (RNNs) або рекурентні нейронні мережі є типом нейронних мереж, які спеціально призначені для роботи з послідовністю даних. Вони мають здатність зберігати інформацію про попередній стан та використовувати цю інформацію для обробки нового входу.

Convolutional Recurrent Neural Network (CRNN) – це архітектура нейронної мережі, яка комбінує концепції згорткових (Convolutional) і рекурентних (Recurrent) нейронних мереж для обробки послідовних даних, таких як аудіо або зображення. Комбінація CNN і RNN може бути ефективною для розпізнавання звукових патернів в часі та частоті. CNN

може виділяти просторові функції, а RNN може враховувати часову залежність.

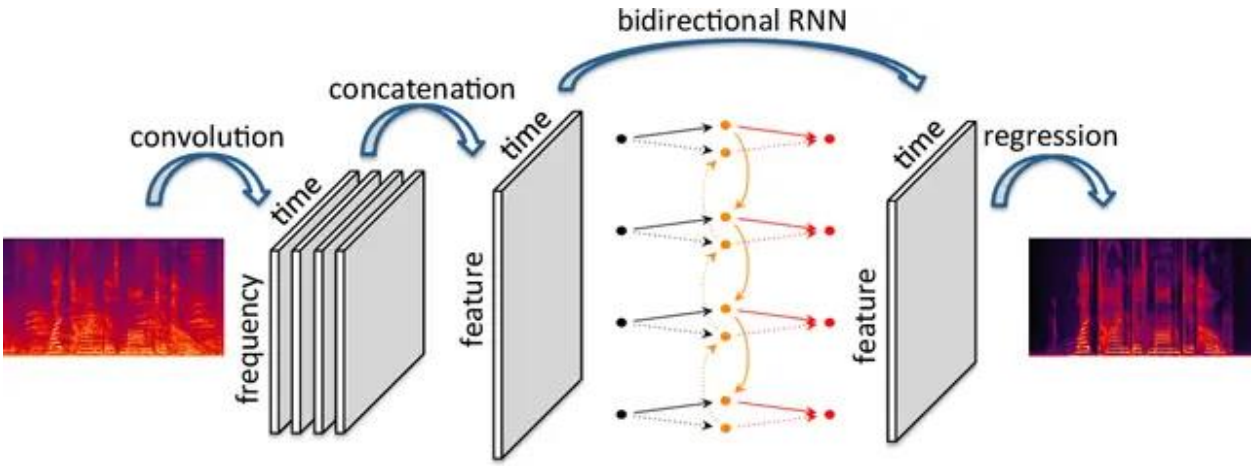


Рисунок 4.6 – Схема архітектури Convolutional Recurrent Neural Network

Deep Residual Networks (ResNets) – це архітектура нейронних мереж, яка використовує блоки з відхиленням (residual blocks) для покращення тренування глибоких нейронних мереж.

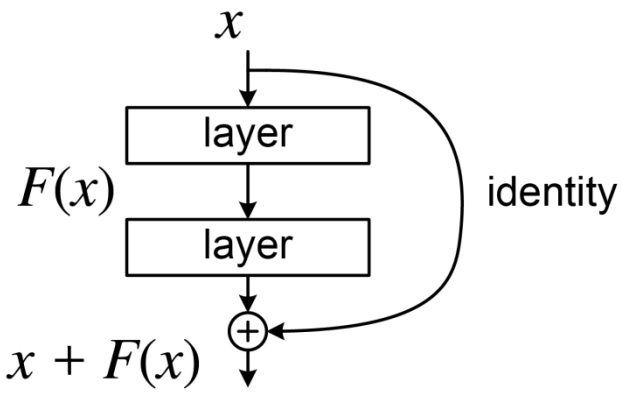


Рисунок 4.7 – Залишковий блок у глибокій залишковій мережі

Наведена архітектура була запропонована у статті "Deep Residual Learning for Image Recognition" авторами Kaiming He, Xiangyu Zhang, Shaoqing Ren, та Jian Sun, і вона стала важливим внеском в розвиток глибокого навчання.

Attention-based models використовують механізм уваги для визначення важливих частин вхідних даних, які варто виділити або на які слід зосередити увагу під час обробки. Ці моделі здатні автоматично приділяти більше уваги важливим частинам вхідного сигналу, забезпечуючи ефективніше використання інформації та поліпшення якості передбачень. Моделі з механізмами уваги можуть зосереджуватися на конкретних частинах мел-спектрограми, що може підвищити точність розпізнавання.

Transfer learning with pre-trained models є стратегією використання передньо навчених моделей для вирішення нового завдання або задачі, яка може відрізнятися від того, для чого була створена вихідна модель. Цей підхід є ефективним у випадках, коли необхідно вирішити задачу з обмеженим обсягом даних.

Використання передньо навчених моделей для аудіо, таких як VGGish або YAMNet, які навчалися на великих наборах даних аудіо, і подальше доналаштування на вашому конкретному завданні.

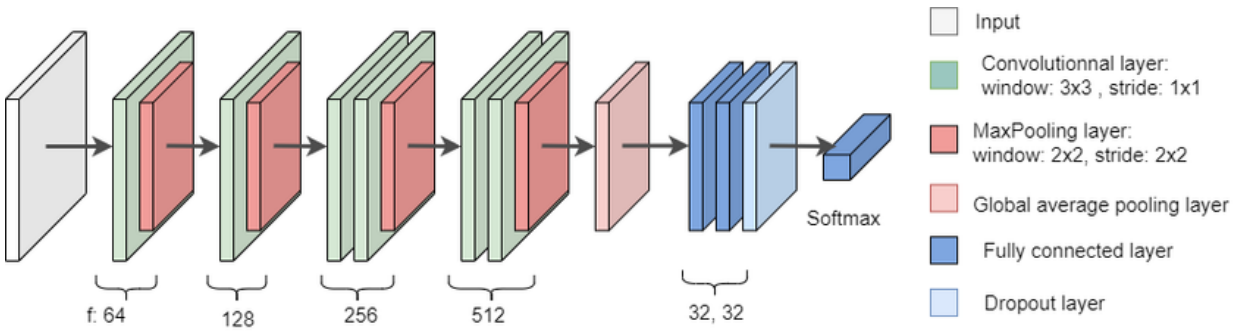


Рисунок 4.8 – VGGish-основана

Із розглянутих архітектур була обрана архітертура Convolutional Neural Networks (CNNs). CNN є потужною архітектурою для обробки послідовностей, таких як аудіосигнали на основі мел-спектрограм. Схему архітектури CNN наведено на рис. 4.9.

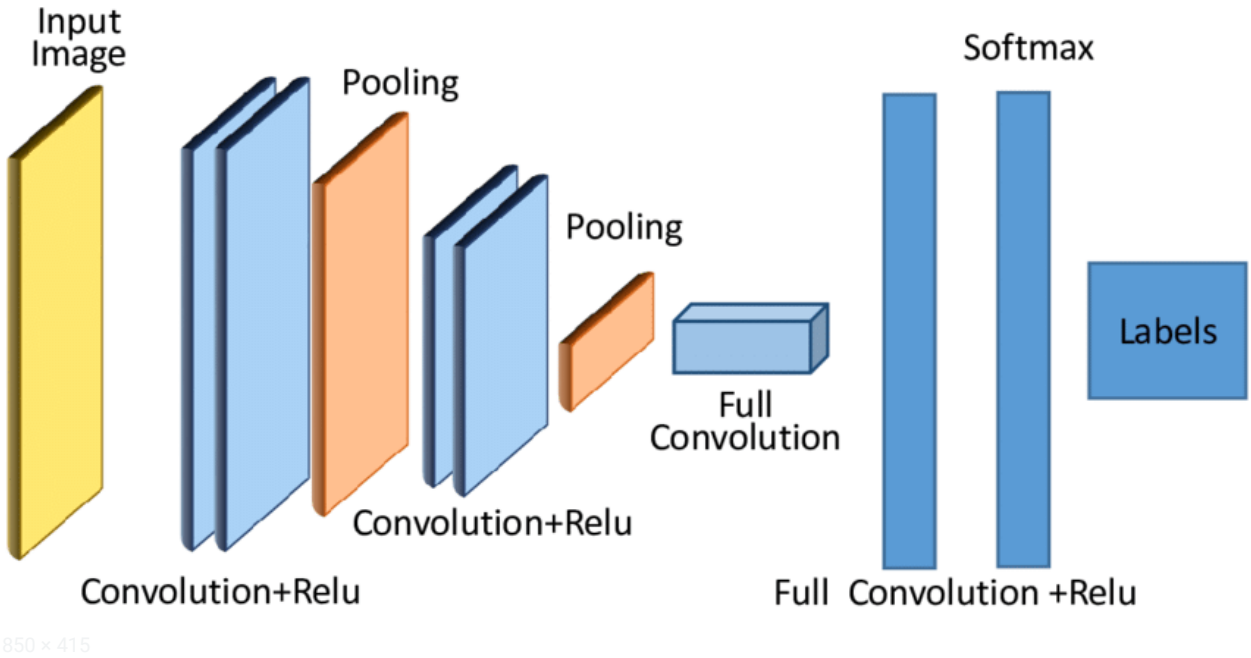


Рисунок 4.9 – Схема архітектури Convolutional Neural Network

Наведена архітектура добре підходить для завдань, пов'язаних із розпізнаванням звуків або акустичним аналізом. Ось кілька причин, чому CNN є ефективними для цих завдань:

– CNN використовує конволюційні шари для виявлення локальних патернів у вхідних даних. Це дозволяє моделі реагувати на локальні особливості в спектрограмі, такі як особливості звуку або акцентовані частоти;

– CNN може автоматично вивчати як локальні, так і глобальні залежності між частотами та часом у спектрограмі. Це важливо для ідентифікації звуків, де зміни можуть відбуватися як в дуже коротких, так і в довгих проміжках часу;

– використання пулінг-шарів дозволяє зменшити просторовий розмір спектрограми, зберігаючи при цьому важливі ознаки. Це робить модель більш ефективною у роботі з великими обсягами даних, що часто характерно для аудіозаписів;

– спектрограма, представлена у вигляді матриці, може бути оброблена за допомогою 2D-конволюційних шарів, що дозволяє моделі

автоматично вивчати просторові особливості в двох напрямках - по часу та частоті;

– застосування локальних фільтрів в конволюційних шарах допомагає зменшити кількість параметрів моделі, що робить її більш ефективною та менш схильною до перенавчання;

– CNN може працювати із спектрограмами різних розмірів, завдяки своїй адаптабельності до вхідних форм. Це особливо корисно для роботи із звуковими записами різної довжини.

CNN може бути ефективно використана для різних завдань, таких як розпізнавання мови, розпізнавання звукових подій або класифікація аудіозаписів на основі мел-спектрограм.

4.4 Розробка класу AudioPreparer

Після розбору процесу створення ML-моделі, підгодовки даних для аудіо-класифікації визначення архітектури ML-моделі можна перейти до написання коду мовою Python.

Перше, що необхідно зробити, для роботи із звуками у мові Python, це зчитати та завантажити аудіофайл у форматі ".wav". Для роботи з аудіо можна використати бібліотеки torchaudio або librosa. Так як надалі планується використовувати бібліотеку машинного навчання PyTorch, для роботи з аудіо буде використовуватись torchaudio. Torchaudio – це бібліотека для обробки звуку за допомогою PyTorch.

Лістинг 4.1 – Класс AudioPreparer

```
import math, random
import torch
import torchaudio
from torchaudio import transforms
from IPython.display import Audio

class AudioPreparer():
  @staticmethod
  def open(audio_file):
```

```
        sig, sr = torchaudio.load(audio_file)
        return (sig, sr)
```

Цей код визначає клас AudioPreparer, який надає функціонал для завантаження аудіо-файлів за допомогою бібліотеки torchaudio. В основі цього класу є статичний метод open, який приймає шлях до аудіо-файлу, завантажує його за допомогою torchaudio.load і повертає дві величини: сигнал аудіо у вигляді тензора та частоту дискретизації (sample rate).

Основні етапи виконання цього коду поділяються на три частини. На першому етапі відбувається імпорт необхідних бібліотек. Перш за все імпортуються math і random для математичних та випадкових операцій. Модуль torch необхідний для роботи з PyTorch. Модуль torchaudio для операцій над аудіо даними. Наступним імпортується модуль transforms, він необхідний для трансформацій аудіо. Також відбувається імпорт IPython.display для відтворення аудіо в ноутбуці через Audio. Хоча в даному коді не усі модулі використовуються, вони будуть використані у подальшій роботі.

Наступним етапом виконання коду є оголошення класу AudioPreparer.

Останнім етапом є визначення статичного методу open. Даний метод отримує шлях до аудіо-файлу як аргумент (audio_file). Використовуючи torchaudio.load, завантажує аудіофайл і повертає дві величини: сигнал аудіо (sig) та частоту дискретизації (sr), а також повертає кортеж значень (sig, sr).

Цей код може бути використаний для зручного завантаження аудіо-файлів у проектах, де використовується PyTorch та torchaudio для обробки аудіо-даних.

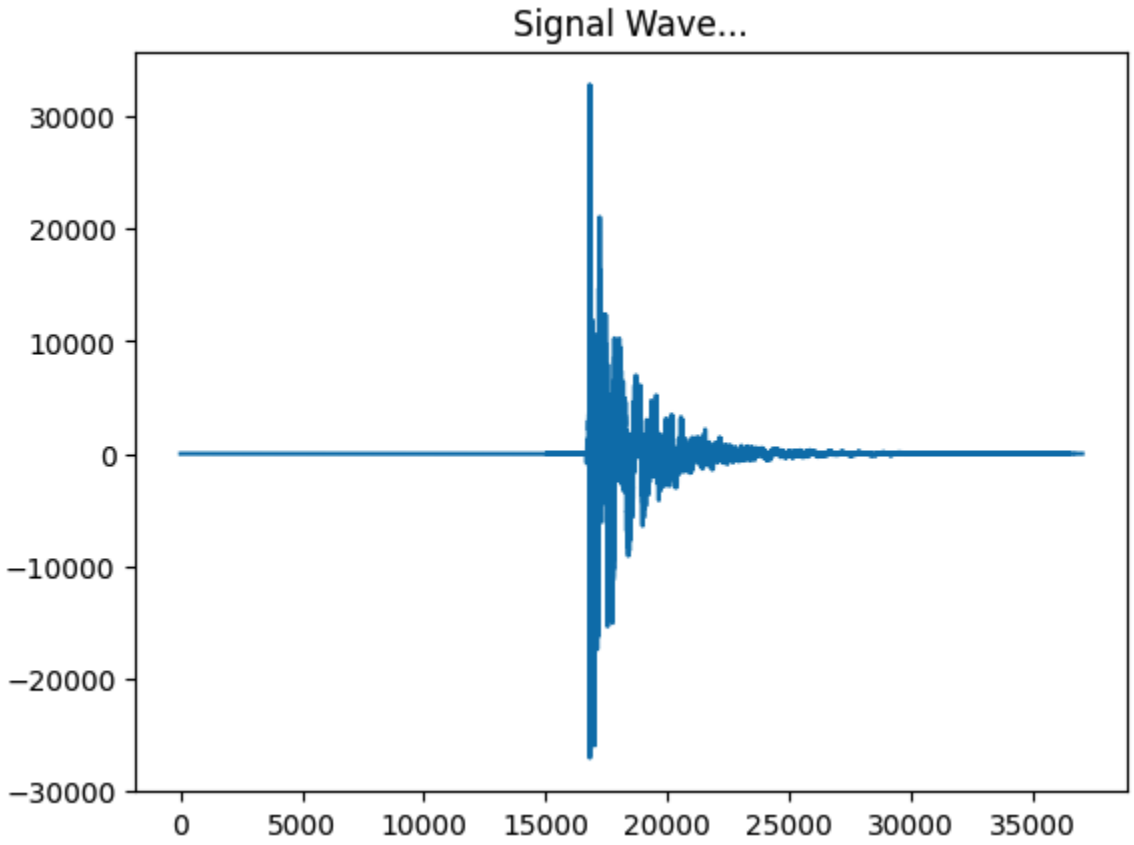


Рисунок 4.10 – Результат імпортування аудіо-файлу з одинарним хлопком

Наступним кроком у написанні коду роботи з аудіо-файлами є перетворення звукових файлів з одним каналом на два канали. Це необхідно бо деякі звукові файли у нашому dataset є моно (тобто 1 аудіоканал), тоді як більшість із них є стерео (тобто 2 аудіоканали). Оскільки наша модель очікує, що всі елементи мають однакові розміри, необхідно буде перетворити монофайли на стерео, дублюючи перший канал у другий.

Для цього було створено статичний метод change_channels.

Лістинг 4.2 – Статичний метод change_channels

```
  @staticmethod
def change_channels(audio_data, new_channels):
    """
    Change the number of channels in the audio data.

    Args:
```

```
        - audio_data (tuple): A tuple containing audio signal
and sample rate.
        - new_channels (int): The desired number of channels (1
for mono, 2 for stereo).

        Returns:
        - tuple: Modified audio data with the specified number
of channels.
        """
        signal, sample_rate = audio_data

        if signal.shape[0] == new_channels:
            return audio_data

        if new_channels == 1:
            modified_signal = signal[:1, :]
        else:
            modified_signal = torch.cat([signal, signal])

        return (modified_signal, sample_rate)
```

Перший рядок приведеного коду отримує значення сигналу (sig) і частоту дискретизації (sr) з вхідного кортежу. Наступна умова перевіряє, чи кількість каналів вже відповідає new_channel. Якщо так, метод повертає вхідний кортеж без змін. Умова (new_channel == 1) перевіряє, чи нова кількість каналів дорівнює 1 (моно). Якщо так, відбувається конвертація від стерео до моно, вибираючи тільки перший канал сигналу. У іншому випадку (коли new_channel не дорівнює 1), відбувається конвертація від моно на стерео, дублюючи перший канал сигналу. Новий кортеж (resig, sr) повертається як результат роботи методу, де resig – змінений сигнал, а sr – частота дискретизації.

Цей метод дозволяє змінювати кількість каналів аудіосигналу між моно та стерео, якщо це необхідно.

Наступним кроком є стандартизація частоти дискретизації. Необхідність стандартизації обумовлена тим, що звукові сигнали можуть мати різну частоту дискретизації, що є вадою для навчання моделі.

Лістинг 4.3 – Статичний метод resample_audio

```
    @staticmethod
    def resample_audio(audio_data, new_sample_rate):
        """
        Resample the audio data to a new sample rate.

        Args:
        - audio_data (tuple): A tuple containing audio
signal and sample rate.
        - new_sample_rate (int): The desired sample rate.

        Returns:
        - tuple: Resampled audio data with the specified
sample rate.
        """
        signal, sample_rate = audio_data

        if sample_rate == new_sample_rate:
            return audio_data

        num_channels = signal.shape[0]
        resampled_signal =
torchaudio.transforms.Resample(sample_rate,
new_sample_rate)(signal[:1, :])

        if num_channels > 1:
            resampled_channel_two =
torchaudio.transforms.Resample(sample_rate,
new_sample_rate)(signal[1:, :])
            resampled_signal = torch.cat([resampled_signal,
resampled_channel_two])

        return (resampled_signal, new_sample_rate)
```

Вхідними параметрами даного методу є параметр aud – аудіо дані, представлені у вигляді кортежу (сигнал, частота дискретизації) та параметр newsr – нова частота дискретизації для аудіо.

Перший рядок отримує значення сигналу (sig) і частоту дискретизації (sr) з вхідного кортежу. Далі виконується умова (sr == newsr), що перевіряє, чи потрібно виконувати процес перенавчання. Якщо частота дискретизації вже дорівнює newsr, метод повертає вхідний кортеж без змін.

Далі отримуємо кількість каналів у сигналі. Після чого використовується трансформація resample для перенавчання першого каналу сигналу до нової частоти дискретизації (newsr).

Умова (num_channels > 1) перевіряє, чи є більше одного каналу у сигналі. Якщо так, аналогічний процес перенавчання виконується для другого каналу, і обидва канали об'єднуються за допомогою torch.cat.

Новий кортеж (resig, newsr) повертається як результат роботи методу, де resig – змінений сигнал після перенавчання, а newsr - нова частота дискретизації.

Цей метод дозволяє перенавчати аудіосигнал до нової частоти дискретизації, якщо це необхідно.

Далі має бути виконана стандартизація розміру аудіо-файлів. Для цього має виконуватись зміна розміру усіх зразків аудіо, щоб вони мали однакову довжину. Цей процес виконується методом подовження тривалості файлу, доповнюючи його тишею, або методом його скорочення.

Лістинг 4.4 – Статичний метод pad_truncate

```
    @staticmethod
        def pad_truncate(audio_data, max_duration_ms):
            """
            Pad or truncate the audio signal to a specified
maximum duration.

            Args:
            - audio_data (tuple): A tuple containing audio
signal and sample rate.
            - max_duration_ms (int): The maximum duration
in milliseconds.

            Returns:
            - tuple: Padded or truncated audio data.
            """
            signal, sample_rate = audio_data
            num_rows, signal_length = signal.shape
            max_length = sample_rate // 1000 *
max_duration_ms

            if signal_length > max_length:
                signal = signal[:, :max_length]

            elif signal_length < max_length:
                pad_begin_length = random.randint(0,
max_length - signal_length)
                pad_end_length = max_length - signal_length
- pad_begin_length
```

```
                pad_begin = torch.zeros((num_rows,
pad_begin_length))
                pad_end = torch.zeros((num_rows,
pad_end_length))

                signal = torch.cat((pad_begin, signal,
pad_end), 1)

            return (signal, sample_rate)
```

Цей код є статичним методом класу та виконує операції з паддінгом та обрізанням аудіо-сигналу, залежно від максимальної тривалості, заданої у мілісекундах.

Вхідними параметрами методу є параметр audio_data - кортеж, що містить аудіо-сигнал та частоту дискретизації і параметр max_duration_ms який представляє собою максимальну тривалість в мілісекундах. Повертає даний метод кортеж, що представляє аудіо-дані, які можуть бути паддінговані або обрізані.

Даний метод виконує описані далі операції. Він розпаковує вхідні аудіо-дані на окремі змінні: signal та sample_rate та отримує кількість рядків та довжину сигналу. Після чого відбувається обробка обрізання або паддінгу. Якщо довжина сигналу більша за максимальну тривалість, то сигнал обрізається до заданої довжини.

Якщо довжина сигналу менша за максимальну тривалість, то виконується паддінг. Генерується випадкова довжина паддінгу на початку сигналу (pad_begin_length), обчислюється довжина паддінгу на кінці сигналу (pad_end_length), створюються тензори з нулів для паддінгу на початку та кінці сигналу. Використовується torch.cat для об'єднання паддінгу, сигналу та паддінгу на кінці.

Далі повертається результуючий сигнал та частота дискретизації. Цей метод може бути використаний для обробки аудіо-даних, забезпечуючи їхню тривалість в межах вказаного ліміту.

Далі необхідно збільшити дані необробленого аудіосигналу, застосувавши зсув у часі, щоб зсунути звук ліворуч або праворуч на випадкову величину.

Лістинг 4.5 – Статичний метод time_shift

```
    @staticmethod
        def time_shift(audio_data, shift_limit):
            """
            Apply time shift to the audio signal.

            Args:
            - audio_data (tuple): A tuple containing audio
signal and sample rate.
            - shift_limit (float): The maximum proportion
of the signal length to shift.

            Returns:
            - tuple: Time-shifted audio data.
            """
            signal, sample_rate = audio_data
            _, signal_length = signal.shape
            shift_amount = int(random.random() *
shift_limit * signal_length)

            # Apply time shift using roll
            time_shifted_signal = signal.roll(shift_amount)

            return (time_shifted_signal, sample_rate)
```

Цей код представляє собою статичний метод з назвою time_shift, який застосовує ефект зсуву в часі до аудіо-сигналу. Цей метод генерує ефект зсуву в часі для аудіо-сигналу на основі випадкового зсуву в часі, який обчислюється відсотком довжини сигналу та максимальним обмеженням.

Подальшим етапом у підготуванні аудіо-даних є перетворення розширеного аудіо-сингалу із формату waveform до спектрограми Мела. Цей формат фіксує основні характеристики аудіо і часто є найбільш підходящим способом введення аудіоданих у моделі глибокого навчання.

З цією метою був написаний код, що визначає статичний метод spectrogram, який використовує бібліотеку torchaudio для генерації звукової спектрограми.

Лістинг 4.6 – Статичний метод spectrogram

```
        @staticmethod
        def spectrogram(aud, n_mels=64, n_fft=1024,
hop_len=None):
            """
            Generate a spectrogram for the input audio
signal.

            Parameters:
            - aud (tuple): A tuple containing the audio
signal and the sample rate.
            - n_mels (int): Number of mel filters in the
spectrogram.
            - n_fft (int): Size of the FFT window.
            - hop_len (int): Hop length of the FFT window.

            Returns:
            - torch.Tensor: The spectrogram in the form of
a torch.Tensor.
            """
            sig, sr = aud
            top_db = 80

            if hop_len is None:
                hop_len = n_fft // 4

            mel_spec_transform = MelSpectrogram(sr,
n_fft=n_fft, hop_length=hop_len, n_mels=n_mels)
            amplitude_to_db_transform =
AmplitudeToDB(top_db=top_db)

            spec =
amplitude_to_db_transform(mel_spec_transform(sig))

            return spec
```

Наведений код отримує аудіосигнал у вигляді кортежу, де sig – це аудіосигнал, а sr – частота дискретизації. Далі відбувається розпакування параметрів. Параметри aud, n_mels, n_fft, і hop_len розпаковуються з кортежу або приймають значення за замовчуванням. Наступним кроком проводиться

перевірка на відсутність значення для hop_len: Якщо hop_len дорівнює None, встановлюється значення hop_len на половину.

В результаті виконання описаного раніше коду отримуємо перетворену waveform до Мел-спектрограми.

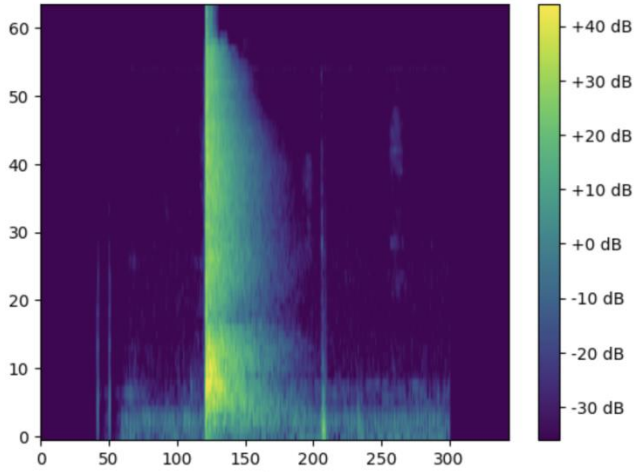


Рисунок 4.11 – Результат перетворення аудіо-сигналу одинарного хлопка до спектрограми Мела

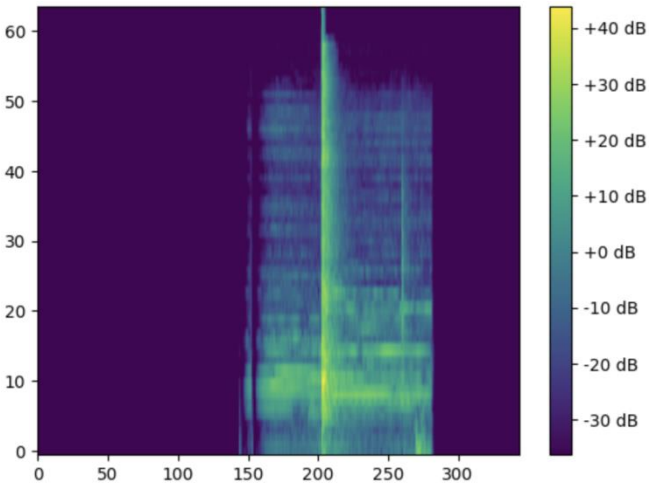


Рисунок 4.12 – Результат перетворення аудіо-сигналу одинарного хлопка із фоновими звуками до спектрограми Мела

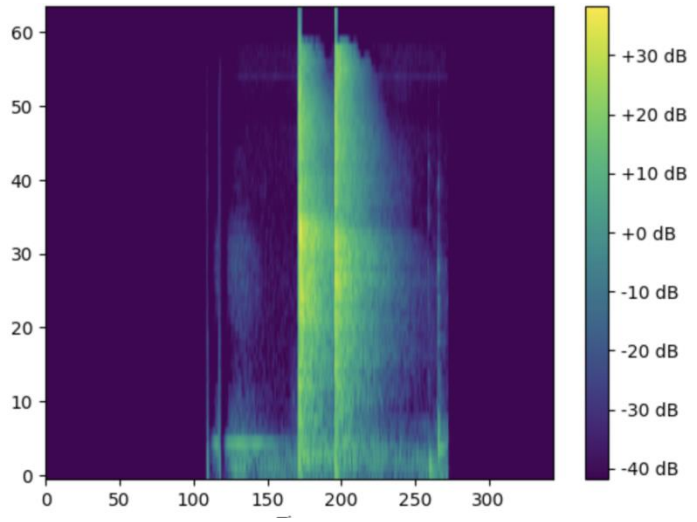


Рисунок 4.13 – Результат перетворення аудіо-сигналу подвійного хлопка до спектрограми Мела

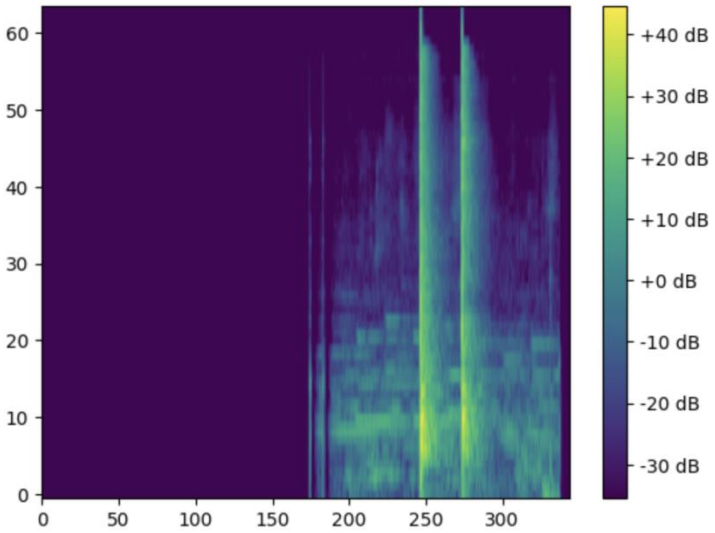


Рисунок 4.14 – Результат перетворення аудіо-сигналу подвійного хлопка із фоновими звуками до спектрограми Мела

Останнім етапом перетворення аудіо-даних є доповнення даних, а саме маскування часу та частоти. Для цього може бути використано модуль SpecAugment, який використовує методи:

– частотна маска – довільно маскує діапазон послідовних частот, додаючи горизонтальні смуги на спектрограму;

– часова маска – подібна до частотних масок, за винятком того, що випадково блокує діапазони часу зі спектрограми за допомогою вертикальних смуг.

Використання Time та Frequency Masking для тренування моделі на мел-спектрограмі має ті ж самі основні цілі і переваги, що і для будь-яких аудіосигналів загалом. Застосування Time та Frequency Masking для тренування моделі на мел-спектрограмі робить її більш гнучкою та адаптивною до різних сценаріїв, які можуть виникнути у реальних аудіоданих. Моделі, навчені з маскуванням, можуть бути більш стійкими до різних шумів, перешкод чи непередбачених змін у вхідних даних. Також маскування може поліпшити загальні властивості моделі, дозволяючи їй краще адаптуватися до різноманітних умов і залишатися ефективною на реальних даних.

Лістинг 4.7 – Статичний метод spectro_augment

```
        @staticmethod
        def spectro_augment(spec, max_mask_pct=0.1,
n_freq_masks=1, n_time_masks=1):
            """
            Apply time and frequency masking to a
spectrogram.

            Parameters:
            - spec (torch.Tensor): Input spectrogram.
            - max_mask_pct (float): Maximum percentage of
bins to mask.
            - n_freq_masks (int): Number of frequency masks
to apply.
            - n_time_masks (int): Number of time masks to
apply.

            Returns:
            - torch.Tensor: Augmented spectrogram.
            """
            _, n_mels, n_steps = spec.shape
            mask_value = spec.mean()
            aug_spec = spec.clone()

            freq_mask_param = int(max_mask_pct * n_mels)
            for _ in range(n_freq_masks):
```

```
                mask_size = random.randint(1,
freq_mask_param)
                f_mask = FrequencyMasking(mask_size)
                aug_spec = f_mask(aug_spec, mask_value)

            time_mask_param = int(max_mask_pct * n_steps)
            for _ in range(n_time_masks):
                mask_size = random.randint(1,
time_mask_param)
                t_mask = TimeMasking(mask_size)
                aug_spec = t_mask(aug_spec, mask_value)

            return aug_spec
```

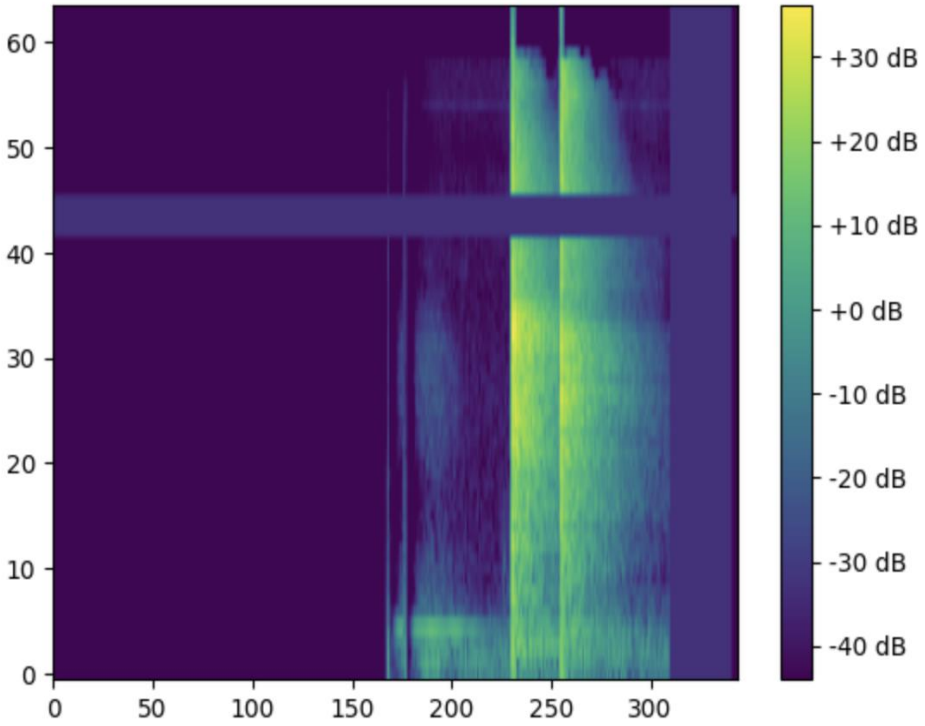


Рисунок 4.15 – Результат аугментації подвійного хлопка спектрограми Мела

Основна мета цього методу - застосувати маскування часу та частоти до спектрограми з метою аугментації даних. Метод спочатку отримує розміри спектрограми та середнє значення. Потім застосовує маскування: випадкове маскування для частот та часу, яке контролюється max_mask_pct, n_freq_masks, і n_time_masks. Отримана аугментована спектрограма повертається як результат.

4.5 Розробка класу SoundDataset

Для виконання описаних методів необхідно виконати їх виклики. Після того як були визначені всі функції трансформації попередньої обробки, визначимо спеціальний об’єкт набору даних Pytorch. Щоб передати дані в модель за допомогою Pytorch, потрібні певні об’єкти.

Для цього потрібен спеціальний об’єкт набору даних, який використовує всі аудіо-перетворення для попередньої обробки аудіо-файлу та готує по одному елементу даних. А також вбудований об’єкт DataLoader, який використовує об’єкт Dataset для отримання окремих елементів даних і упаковує їх у пакет даних.

Лістинг 4.8 – Клас SoundDataset

```
class SoundDataset(Dataset):
    def __init__(self, df, data_path):
        self.df = df
        self.data_path = str(data_path)
        self.duration = 4000
        self.sr = 44100
        self.channel = 2
        self.shift_pct = 0.4

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        audio_file = self.data_path + self.df.loc[idx,
'relative_path']
        class_id = self.df.loc[idx, 'classID']

        aud = AudioPreparer.open(audio_file)
        reaud = AudioPreparer.resample_audio(aud,
self.sr)
        rechan = AudioPreparer.change_channels(reaud,
self.channel)

        dur_aud = AudioPreparer.pad_truncate(rechan,
self.duration)
        shift_aud = AudioPreparer.time_shift(dur_aud,
self.shift_pct)
        sgram = AudioPreparer.spectro_gram(shift_aud,
n_mels=64, n_fft=1024, hop_len=None)
```

```
            aug_sgram =
AudioPreparer.spectro_augment(sgram, max_mask_pct=0.1,
n_freq_masks=2, n_time_masks=2)

            return aug_sgram, class_id
```

Цей код описує клас SoundDS, який є підкласом Dataset з бібліотеки PyTorch. Його призначення – створення набору даних для аудіо, зокрема для спеціального використання в задачах машинного навчання, які пов'язані із звуком.

Даний код містить конструктор класу, де df – це DataFrame, який містить інформацію про дані, а data_path – шлях до директорії, де знаходяться аудіофайли. Також в межах класу реалізован метод len, який повертає кількість записів в наборі даних.

А також метод getitem, який повертає елемент даних за певним індексом. Даний метод завантажує аудіофайл за індексом, використовуючи методи з класу AudioPreparer та створює спектрограму, яку повертає разом із класом аудіозапису.

Загальна ідея реалізованого класу – це підготовка даних для подальшого використання у моделі машинного навчання, наприклад, у навчанні нейронної мережі для класифікації звуків.

4.6 Підготовка пакетів даних

Після розробки класу, необхідного для підготовки даних для подальшого використання можемо використати його для підготовки пакетів даних.

Використовуємо отриманий спеціальний набір даних, щоб завантажувати функції та мітки з нашого фрейму даних Pandas і розділяти ці дані випадковим чином у співвідношенні 80:20 на набори для навчання та перевірки. Потім використовуємо їх для створення завантажувачів даних для навчання та перевірки.

Використання співвідношення 80:20 для розділу набору даних на тренувальні та валідаційні набори є загальноприйнятим підходом в машинному навчанні. Це відношення є компромісом між тим, щоб навчити модель достатньо добре на наявних даних і водночас мати достатньо валідаційних даних для оцінки її загальної ефективності та уникнення перенавчання.

Лістинг 4.9 – Клас SoundDataset

```
from torch.utils.data import DataLoader, random_split

# Create SoundDataset instance
myds = SoundDataset(df, data_path)

# Define the split ratio
split_ratio = 0.8

# Random split of 80:20 between training and validation
num_items = len(myds)
num_train = round(num_items * split_ratio)
num_val = num_items - num_train

# Use random_split directly
train_ds, val_ds = random_split(myds, [num_train,
num_val])

# Define batch size
batch_size = 16

# Create training and validation data loaders
train_dl = DataLoader(train_ds, batch_size=batch_size,
shuffle=True)
val_dl = DataLoader(val_ds, batch_size=batch_size,
shuffle=False)
```

Цей код готує дані для тренування та валідації, щоб їх можна було використовувати в навчанні та оцінці моделі машинного навчання для задачі обробки звуку. Під час його виконання створюється екземпляр SoundDataset, який використовується для створення екземпляру власного датасету SoundDataset, який, містить функціонал для завантаження та підготовки звукових даних.

Наступним кроком виконання є випадковий розподіл тренувальних і валідаційних даних. Для цього використовується функція random_split для розділення даних на тренувальний та валідаційний набори в співвідношенні 80:20.

Зокрема, у описаному коді відбувається визначення розміру пакета - задається розмір пакета для тренування та валідації (16 у цьому випадку).

Далі відбувається створення DataLoaders. Використовуючи DataLoader з PyTorch, створюються об'єкти для тренування та валідації з відповідними наборами даних, shuffle=True вказує на перемішування даних для тренування.

Коли починаємо навчання, завантажувач даних випадковим чином вибере одну партію вхідних функцій, що містить список імен аудіофайлів, і запускає аудіо-перетворення попередньої обробки для кожного аудіофайлу. Він також отримує пакет відповідних цільових міток, що містять ідентифікатори класів. Таким чином, він буде виводити одну партію навчальних даних за раз, які можуть бути безпосередньо подані як вхідні дані для нашої моделі глибокого навчання.

Розглянемо етапи трансформації наших даних, починаючи з аудіофайлу.

Аудіо з файлу завантажується в масив. Більшість аудіо дискретизується з частотою 44,1 кГц і має тривалість приблизно 4 секунди, що призводить до 44 100 * 4 = 176 400 семплів. Якщо звук має 1 канал, форма масиву буде (1, 176,400). Подібним чином аудіо тривалістю 4 секунди з 2 каналами та дискретизацією 48 кГц матиме 192 000 семплів і форму (2 192 000).

Оскільки канали та частоти дискретизації кожного аудіо відрізняються, наступні два перетворення передискретизують аудіо до стандартної частоти 44,1 кГц і до стандартних 2 каналів.

Оскільки деякі аудіозаписи можуть тривати більше або менше 4 секунд, необхідно також стандартизувати тривалість аудіо до фіксованої

тривалості 4 секунди. Тепер масиви для всіх елементів мають однакову форму (2 176 400)

Збільшення даних Time Shift тепер випадковим чином зміщує кожен аудіо-семпл вперед або назад. Форми при цьому залишаються незмінні. Доповнене аудіо тепер перетворюється на спектрограму Мела. Доповнення даних випадковим чином застосовує маски часу та частоти до спектрограм Мела.

Таким чином, кожна партія матиме два тензори: один для даних функції X, що містить спектрограми Mel, а інший для цільових міток y, що містять числові ідентифікатори класу. Партії вибираються випадковим чином із даних навчання для кожного етапу навчання. Тепер дані готові для введення в модель.

4.7 Створення ML-моделі

Оскільки підготовлені дані тепер складаються із зображень спектрограм, створюємо архітектуру класифікації CNN для їх обробки. Він має чотири згорткові блоки, які генерують карти функцій. Ці дані потім переформовуються в потрібний нам формат, щоб їх можна було ввести в лінійний класифікатор, який нарешті виводить прогнози для трьох класів (один хлопок, подвійний хлопок, фон).

Декілька деталей про те, як модель обробляє пакет даних:

– пакет зображень вводиться в модель;

– кожен шар CNN застосовує свої фільтри, щоб збільшити глибину зображення, тобто. кількість каналів. Ширина й висота зображення зменшуються в міру застосування ядер і кроків. Нарешті, після проходження чотирьох шарів CNN можна отримати вихідні карти ознак;

– дані об’єднуються та вирівнюються, а потім вводиться до лінійного шару;

– лінійний рівень виводить одну оцінку прогнозу на клас.

Для створення моделі був написаний клас AudioClassifier. Цей код визначає архітектуру моделі для класифікації аудіо.

Лістинг 4.10 – Імпортування необхідних модулів та ініціалізація класу AudioClassifier

```
import torch.nn.functional as F
from torch.nn import init

class AudioClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        conv_layers = []
        # Перший блок з конволюційним шаром, ReLU та
Batch Normalization. Використовується Kaiming
Initialization.
        self.conv1 = nn.Conv2d(2, 8, kernel_size=(5,
5), stride=(2, 2), padding=(2, 2))
        self.relu1 = nn.ReLU()
        self.bn1 = nn.BatchNorm2d(8)
        init.kaiming_normal_(self.conv1.weight, a=0.1)
        self.conv1.bias.data.zero_()
        conv_layers += [self.conv1, self.relu1,
self.bn1]

        # Другий блок з конволюційним шаром
        self.conv2 = nn.Conv2d(8, 16, kernel_size=(3,
3), stride=(2, 2), padding=(1, 1))
        self.relu2 = nn.ReLU()
        self.bn2 = nn.BatchNorm2d(16)
        init.kaiming_normal_(self.conv2.weight, a=0.1)
        self.conv2.bias.data.zero_()
        conv_layers += [self.conv2, self.relu2,
self.bn2]

        # Третій блок з конволюційним шаром
        self.conv3 = nn.Conv2d(16, 32, kernel_size=(3,
3), stride=(2, 2), padding=(1, 1))
        self.relu3 = nn.ReLU()
        self.bn3 = nn.BatchNorm2d(32)
        init.kaiming_normal_(self.conv3.weight, a=0.1)
        self.conv3.bias.data.zero_()
        conv_layers += [self.conv3, self.relu3,
self.bn3]

        # Четвертий блок з конволюційним шаром
        self.conv4 = nn.Conv2d(32, 64, kernel_size=(3,
3), stride=(2, 2), padding=(1, 1))
        self.relu4 = nn.ReLU()
        self.bn4 = nn.BatchNorm2d(64)
```

```
            init.kaiming_normal_(self.conv4.weight, a=0.1)
            self.conv4.bias.data.zero_()
            conv_layers += [self.conv4, self.relu4,
self.bn4]

            # Лінійний класифікатор
            self.ap = nn.AdaptiveAvgPool2d(output_size=1)
            self.lin = nn.Linear(in_features=64,
out_features=10)

            # Згорткові блоки
            self.conv = nn.Sequential(*conv_layers)
```

Код наведений у лістингу 4.10 виконує імпортування необхідних для моделі сторонніх модулів. Також у наведеному коді відбувається ініціалізація класу AudioClassifier.

Лістинг 4.11 – Метод forward класу AudioClassifier

```
    def forward(self, x):
            # Запуск згорткових блоків
            x = self.conv(x)

            # Адаптивний пул та розгортання для введення в
лінійний шар
            x = self.ap(x)
            x = x.view(x.shape[0], -1)

            # Лінійний шар
            x = self.lin(x)

            # Кінцевий вихід
            return x
```

Наведений код є методом forward класу AudioClassifier. Цей метод визначає обчислення проходження вперед (forward pass) для моделі. Цей метод приймає вхідні дані x та виконує низку операцій, щоб отримати вихід моделі.

Лістинг 4.12 – Ініціалізація моделі

```
    # Створення моделі та перенесення її на GPU, якщо це
можливо
    myModel = AudioClassifier()
    device = torch.device("cuda:0" if
torch.cuda.is_available() else "cpu")
```

```
myModel = myModel.to(device)
# Перевірка, що вона на CUDA
next(myModel.parameters()).device
```

Наведений у лістингу 4.12 код представляє собою ініціалізацію створеної раніше моделі.

4.8 Тренування ML-моделі

Після створення моделі, підготовки і розбиття навчальних та тестових даних можна перейти до тренування ML-моделі. Для цього необхідно визначити функції для оптимізатора, втрат і планувальника, щоб динамічно змінювати нашу швидкість навчання в міру проходження навчання, що зазвичай дозволяє зберігати навчання за меншу кількість епох.

Навчаємо модель для кількох епох, обробляючи пакет даних на кожній ітерації. Для цього відстежуємо простий показник точності, який вимірює відсоток правильних прогнозів.

Лістинг 4.13 – Метод training

```
def training(model, train_dl, num_epochs):
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(),
lr=0.001)
    scheduler =
torch.optim.lr_scheduler.OneCycleLR(optimizer,
max_lr=0.001,
steps_per_epoch=len(train_dl),
epochs=num_epochs,
anneal_strategy='linear')

    for epoch in range(1, num_epochs + 1):
        running_loss = 0.0
        correct_prediction = 0
        total_prediction = 0

        for i, data in enumerate(train_dl, 1):
            inputs, labels = data[0].to(device),
data[1].to(device)
            inputs_m, inputs_s = inputs.mean(),
inputs.std()
            inputs = (inputs - inputs_m) / inputs_s
```

```
                optimizer.zero_grad()
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                loss.backward()
                optimizer.step()
                scheduler.step()
                running_loss += loss.item()
                _, prediction = torch.max(outputs, 1)
                correct_prediction += (prediction ==
labels).sum().item()
                total_prediction += prediction.shape[0]
            avg_loss = running_loss / len(train_dl.dataset)
            acc = correct_prediction / total_prediction
            print(f'Epoch: {epoch}, Loss: {avg_loss:.2f},
Accuracy: {acc:.2f}')
    num_epochs = 2
    training(myModel, train_dl, num_epochs)
```

Цей код визначає функцію навчання training, яка відповідає за тренування моделі. Під час виконання цього коду визначаються функція втрат, оптимізатор (в даному випадку Adam) та планувальник швидкості навчання (scheduler). Планувальник використовує OneCycleLR для динамічного регулювання швидкості навчання.

Для кожної епохи проводиться проходження через всі батчі у тренувальному DataLoader. Відбувається обчислення функції втрат, зворотній прохід та оптимізація параметрів моделі. Також під час виконання наведеного коду виводиться середнє значення втрат та точність моделі на тренувальному наборі після кожної епохи.

4.9 Перевірка ML-моделі

Після тренування моделі необхідно перевірити коректність її роботи, для цього був написаний код наведений у лістингу 4.14.

Лістинг 4.14 – Метод inference

```
def training(model, train_dl, num_epochs):
    # Функція втрат, оптимізатор та планувальник
    criterion = nn.CrossEntropyLoss()
```

```
        optimizer = torch.optim.Adam(model.parameters(),
lr=0.001)
        scheduler =
torch.optim.lr_scheduler.OneCycleLR(optimizer,
max_lr=0.001,
steps_per_epoch=int(len(train_dl)),
epochs=num_epochs,
anneal_strategy='linear')
        # Повторюємо для кожної епохи
        for epoch in range(num_epochs):
            running_loss = 0.0
            correct_prediction = 0
            total_prediction = 0
            # Повторюємо для кожного пакету в тренувальному
наборі
            for i, data in enumerate(train_dl):
                # Отримуємо вхідні ознаки та цільові мітки,
і пересуваємо їх на GPU
                inputs, labels = data[0].to(device),
data[1].to(device)
                # Нормалізуємо вхідні дані
                inputs_m, inputs_s = inputs.mean(),
inputs.std()
                inputs = (inputs - inputs_m) / inputs_s
                # Обнулити градієнти параметрів
                optimizer.zero_grad()
                # Прямий прохід
                outputs = model(inputs)
                # Обчислення втрат
                loss = criterion(outputs, labels)
                # Зворотній прохід та оптимізація
                loss.backward()
                optimizer.step()
                scheduler.step()
                # Зберігаємо статистику для втрат і
точності
                running_loss += loss.item()
                # Отримуємо прогнозований клас з найвищим
балом
                _, prediction = torch.max(outputs, 1)
                # Кількість прогнозів, які відповідають
цільовій мітці
                correct_prediction += (prediction ==
labels).sum().item()
                total_prediction += prediction.shape[0]
            # Вивести статистику в кінці епохи
            num_batches = len(train_dl)
            avg_loss = running_loss / num_batches
            acc = correct_prediction / total_prediction
            print(f'Епоха: {epoch}, Втрати: {avg_loss:.2f},
Точність: {acc:.2%}')
        print('Тренування завершено')
```

```
    # Кількість епох (для демонстраційних цілей, налаштуйте
за потребою)
    num_epochs = 2
    training(myModel, train_dl, num_epochs)
```

Наведений код призначений для оцінки точності моделі на валідаційному наборі після тренування. Це важливо для визначення, наскільки добре модель справляється з новими, раніше не баченими даними.

Під час його виконання відбувається вимикання оновлення градієнтів з torch.no_grad(): Це важливий крок, оскільки нашою метою є створення інференсу, і нам не потрібно вивчати градієнти. Дялі відбувається ітерація через val_dl. Це цикл, який перебирає пакети з валідаційного набору даних (val_dl).

Вхідні дані нормалізуються за допомогою середнього значення і стандартного відхилення.

Далі отримуються прогнози моделі. Модель застосовується до вхідних даних, і отримуються вихідні прогнози. На основі цього відбувається оцінка правильності прогнозів. Для цього порівнюються прогнози з фактичними мітками, і обчислюється загальна кількість правильних прогнозів та загальна кількість прогнозів. Точність розраховується як відношення правильних прогнозів до загальної кількості прогнозів.

Останнім кроком виводиться значення точності та загальної кількості елементів.

На наведеному рисунку 4.16 можна побачити коректність роботи моделі. Модель із доволі високою впевніністю визначає три класи звуків: фоновий шум (відсутність хлопків), одинарний хлопок та подвійний хлопок.

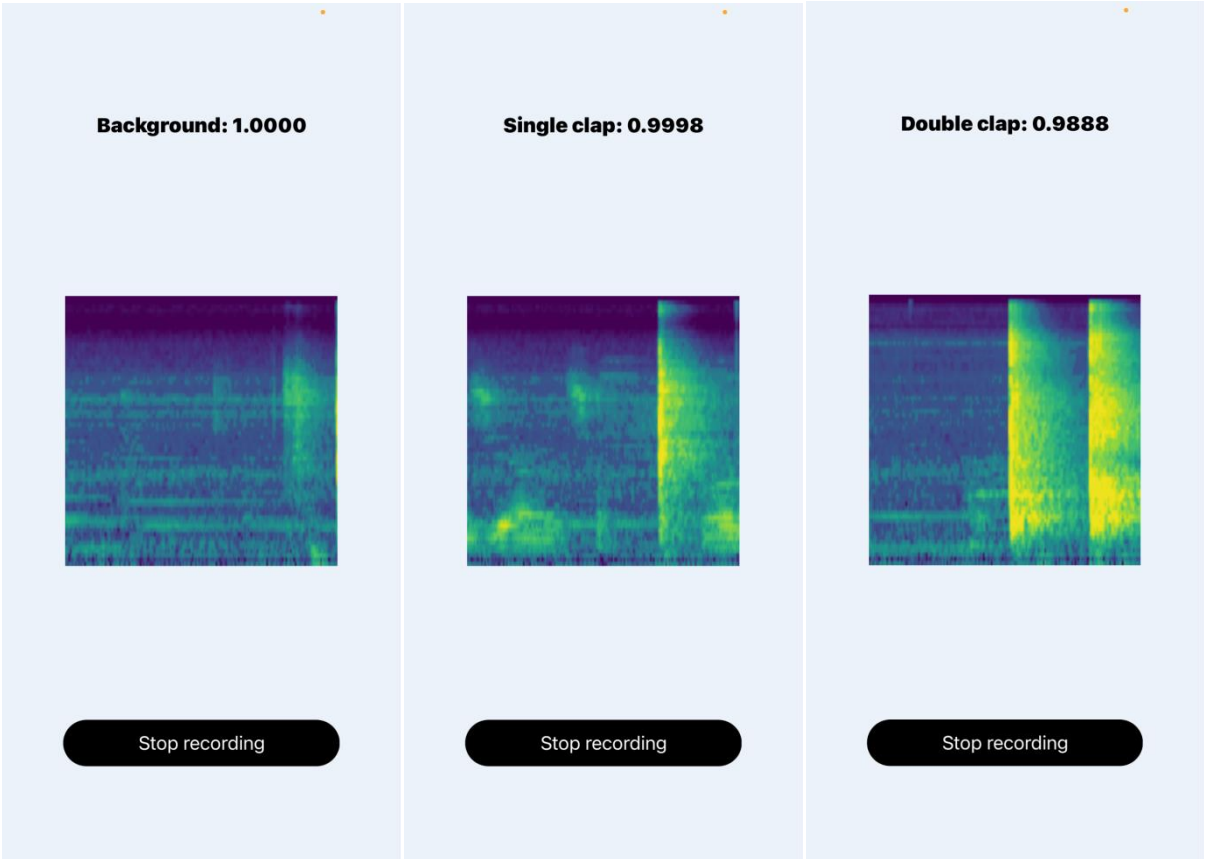


Рисунок 4.16 – Результат роботи моделі

ВИСНОВКИ

Для виконання кваліфікаційної роботи був виконан аналіз технологій ідентифікації звуків. А саме були розглянуті існуючі підходи до обробки звукових даних, було проведено аналіз існуючих методів та інструментів для обробки звукових даних, а також досліджені можливості їх застосування на практиці. Були проаналізовані можливі складнощі у процесі ідентифікації звуків. Також були розглянуті варіанти візуалізації даних, і зокрема метод візуалізації – спектрограма Мела та його, також було проведено її порівняння із графіками хвильової форми.

Було виконано аналіз процесу розробки ML-моделі для ідентифікації звуків, що включає аналіз та вибір інструментів, необхідних для розробки такої моделі. На основі аналізу процесу розробки та предметної області, було створено детальний алгоритм процесу розробки ML-моделі та її навчання. Зокрема були визначені умови роботи ML-моделі.

Після проведення перелічених етапів роботи була виконана розробка ML-моделі, що включає в себе підготовку даних для аудіо-класифікації, визначення архітектури ML-моделі, розробку супутніх класів, підготовку пакетів даних, створення ML-моделі, тренування ML-моделі та перевірку працездатності ML-моделі. Перевірка працездатності моделі дала позитивний результат та показала, що модель може бути використана для створення клієнтських застосунків з метою ідентифікації звуків. Також було проведено аналіз потенційних сфер застосування даної розробки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Xinhui Zhou, Daniel Garcia-Romero, Ramani Duraiswami, “Carol Espy-Wilson, Shihab Shamma, Linear versus mel frequency cepstral coefficients for speaker recognition”, 2011, DOI: https://ieeexplore.ieee.org/document/6163888.

2. Karlijn Willems, “Python Machine Learning: Scikit-Learn Tutorial”, 2019, URL: https://www.datacamp.com/tutorial/machine-learning-python.

3. P. Raguraman, M. R. and M. Vijayan, "LibROSA Based Assessment Tool for Music Information Retrieval Systems," 2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), San Jose, CA, USA, 2019, pp. 109-114, DOI: 10.1109/MIPR.2019.00027.

4. Code Complete / Steven C. McConnell. – United States: Microsoft Press, 1993. – 914 c..

5. Zohaib Mushtaq, Shun-Feng Su “Environmental sound classification using a regularized deep convolutional neural network with data augmentation”, 2020, DOI: 10.1016/j.apacoust.2020.107389.

6. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun “Deep Residual Learning for Image Recognition”, 12 December 2016, DOI: 10.1109/CVPR.2016.90.