НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи і методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки»

на тему: «Створення інтелектуальної системи класифікації пташиних видів на основі аналізу аудіосигналів із застосуванням алгоритмів глибокого навчання та реалізацією у мобільному додатку»

Виконав:
студент (-ка) IV курсу, групи КІ-11
Сташко Ілля Ілліч _____

Керівник:
доцент кафедри ШІ, доктор філософії,
Гуськова Віра Генадіївна _____

Консультант з економічного розділу:
доцент кафедри ЕК ФММ, к.е.н., доцент,
Рощина Надія Василівна _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри ШІ, к.т.н., доцент,
Комариста Богдана Миколаївна _____

Рецензент:
старший викладач кафедри ММСА, д.ф.,
Левенчук Людмила Борисівна _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студента _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«31» січня 2024 р.

ЗАВДАННЯ
на дипломну роботу студенту
Сташко Ілля Ілліч

1. Тема роботи «Створення інтелектуальної системи класифікації пташиних видів на основі аналізу аудіосигналів із застосуванням алгоритмів глибокого навчання та реалізацією у мобільному додатку», керівник роботи керівник роботи Гуськова Віра Генадіївна, доцент кафедри штучного інтелекту, доктор філософії, затверджені наказом по НН ІПСА від «26» травня 2025 р. №1759-с.

2. Термін подання студентом роботи «10» червня 2025 року.

3. Вихідні дані до роботи: аудіозаписи голосів птахів із публічних датасетів, анотації видів, документація фреймворків PyTorch, HuggingFace, Django, React Native.

4. Зміст роботи: дослідження предметної області, аналіз сучасних підходів до аудіокласифікації, обґрунтування вибору інструментів, навчання моделі, розробка серверної частини, реалізація мобільного застосунку та вебінтерфейсу, тестування, функціонально-вартісний аналіз.

5. Перелік ілюстративного матеріалу: презентація, скріншоти мобільного застосунку та вебінтерфейсу, матриця плутанини, графіки точності, структура архітектури системи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Надія Василівна, доцент, к. е. н.		

7. Дата видачі завдання «03» лютого 2025 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Огляд літератури за темою	21.04.2025	Виконано
2	Підготовка першого розділу	01.05.2025	Виконано
3	Підготовка другого розділу	09.05.2025	Виконано
4	Розробка програмного продукту	21.05.2025	Виконано
5	Підготовка третього розділу	26.05.2025	Виконано
6	Оформлення дипломної роботи	02.06.2025	Виконано
7	Підготовка презентації доповіді	05.06.2025	Виконано

Студент Ілля СТАШКО

Керівник Віра ГУСЬКОВА

РЕФЕРАТ

Дипломна робота: 95 с., 18 рис., 6 табл., 9 посилань, додаток.

ГЛИБИННЕ НАВЧАННЯ, КЛАСИФІКАЦІЯ АУДІОСИГНАЛІВ, WAV2VEC2, ПТАШИНІ ГОЛОСИ, МОБІЛЬНИЙ ЗАСТОСУНОК, DJANGO, REST API.

Об’єкт дослідження – аудіозаписи голосів птахів.

Предмет дослідження – методи глибокого навчання для класифікації біоакустичних сигналів.

Мета роботи – створити інтелектуальну систему для розпізнавання пташиних видів за аудіосигналами з реалізацією у мобільному додатку.

У роботі проаналізовано особливості класифікації пташиних голосів, оглянуто сучасні підходи до обробки аудіо з використанням нейронних мереж. Розроблено модель на основі Wav2Vec2, серверну частину з REST API, вебінтерфейс і мобільний застосунок. Система дозволяє користувачеві завантажити аудіо та отримати ймовірнісну класифікацію виду птаха. Проведено тестування точності (F1 ~74%), побудовано матрицю плутанини, оцінено швидкодію. Виконано функціонально-вартісний аналіз та розраховано економічну доцільність обраної архітектури.

ABSTRACT

Thesis: 95 p., 18 figures, 6 tables, 9 references, 1 appendix.

DEEP LEARNING, AUDIO SIGNAL CLASSIFICATION, WAV2VEC2, BIRD VOCALIZATIONS, MOBILE APPLICATION, DJANGO, REST API.

Object of research – audio recordings of bird sounds.

Subject of research – deep learning methods for the classification of bioacoustics signals.

Purpose of the work – to create an intelligent system for bird species recognition based on audio signal analysis, implemented as a mobile application.

The work analyzes the challenges of classifying bird vocalizations and reviews modern approaches to audio processing using neural networks. A model based on Wav2Vec2 was developed, along with a server backend using REST API, a web interface, and a mobile application. The system allows users to upload audio files and receive a probabilistic classification of bird species. Accuracy testing was performed (F1 score ~74%), a confusion matrix was generated, and system response time was evaluated. A functional cost analysis was conducted to assess the economic feasibility of the chosen architecture.

ЗМІСТ

ВСТУП

У сучасному світі зростає інтерес до вивчення, збереження та моніторингу біорізноманіття. Одним із найбільш важливих напрямів є ідентифікація та облік видів птахів, оскільки птахи є важливими біоіндикаторами стану екосистем. Традиційні методи спостереження за птахами, які ґрунтуються на візуальній ідентифікації, часто вимагають значного досвіду, часу та спеціалізованого обладнання. У той же час, голосові сигнали птахів є унікальними для кожного виду та можуть бути використані як надійний засіб для автоматизованої класифікації.

У зв’язку з активним розвитком штучного інтелекту та глибокого навчання з’являються нові можливості для створення інтелектуальних систем, здатних ефективно аналізувати біоакустичні дані. Особливу цінність мають мобільні застосунки, які дозволяють інтегрувати технології розпізнавання пташиних голосів безпосередньо у повсякденне життя користувачів. Це відкриває нові перспективи для популяризації орнітологічних знань серед широкого загалу, а також створює передумови для залучення громадськості до екологічного моніторингу.

Метою даної дипломної роботи є створення інтелектуальної системи класифікації пташиних видів на основі аналізу аудіосигналів із застосуванням алгоритмів глибокого навчання, а також реалізація функціоналу у вигляді мобільного додатку. Крім того, передбачається розробка відкритого прикладного програмного інтерфейсу (API), що дозволить використовувати результати даного дослідження в інших проєктах, сприяючи подальшому розвитку технологій біоакустичного моніторингу.

У роботі розглядаються теоретичні основи обробки аудіосигналів, сучасні методи глибокого навчання, архітектури нейронних мереж, специфіка біоакустичних сигналів, а також практичні аспекти реалізації

програмного забезпечення. Особливу увагу приділено аналізу предметної області, огляду існуючих рішень, а також технічним викликам, які виникають у процесі створення подібних систем.

РОЗДІЛ 1 ОГЛЯД СФЕРИ РОЗПІЗНАВАННЯ ПТАШИНИХ ГОЛОСІВ

1.1 Актуальність задачі розпізнавання пташиних видів за аудіосигналами

Збереження біорізноманіття є пріоритетним завданням для сучасного суспільства. Птахи, як одна з найбільш вивчених груп живих організмів, відіграють ключову роль у функціонуванні екосистем. Вони слугують біоіндикаторами стану навколишнього середовища та реагують на антропогенні зміни першими. Визначення видового складу птахів у певному регіоні дозволяє здійснювати оперативний екологічний моніторинг, а також оцінювати наслідки змін клімату та господарської діяльності людини.

Традиційні методи ідентифікації птахів потребують високої кваліфікації спостерігача та значних затрат часу, а також мають обмеження при дослідженні важкодоступних територій. У цьому контексті зростає значущість автоматизованих систем розпізнавання пташиних голосів на основі аудіоаналізу. Такі системи відкривають нові можливості як для наукових досліджень, так і для освітніх ініціатив, залучаючи громадськість до активного спостереження за природою.

Крім наукового і природоохоронного значення, важливо зазначити й зростаючий інтерес з боку широкого кола користувачів. Багато людей у побуті часто чують пташиний спів і бажають дізнатися, який саме птах видає той чи інший звук. Відсутність доступних та простих у використанні інструментів для ідентифікації таких звуків створює потребу в інтуїтивному рішенні. Розробка мобільного застосунку, що дозволяє швидко та точно визначити вид птаха за його голосом, сприяє не лише задоволенню особистої зацікавленості користувачів, а й формуванню екологічної свідомості та просвітницькій діяльності.

Запропонована система покликана вирішити цю проблему – вона забезпечує зручний інтерфейс для розпізнавання птахів за аудіосигналами,

орієнтована на використання широким колом осіб та супроводжується публічним API для подальшої інтеграції в екологічні, освітні та наукові проєкти.

1.2 Огляд існуючих систем розпізнавання пташиних голосів

З розвитком алгоритмів машинного навчання та зростанням обчислювальних можливостей мобільних і серверних платформ, з'явилася низка систем, що забезпечують автоматизоване розпізнавання пташиних голосів. Серед найбільш відомих і поширених варто виділити Merlin Bird ID, BirdNET, Warblr, ChirpOMatic, Song Sleuth, а також проєкти з відкритим кодом, такі як OpenSoundscape і Kaleidoscope.

Merlin Bird ID. Розроблений Корнелльською лабораторією орнітології, Merlin Bird ID є одним із найпопулярніших додатків для ідентифікації птахів за співом або зовнішнім виглядом. Додаток використовує глибокі нейронні мережі, натреновані на багатомільйонних датасетах, у тому числі з джерел Macaulay Library та eBird. Merlin (рис 1.1) працює в онлайн-режимі та дозволяє користувачам визначати види птахів на основі записаного аудіофрагмента, який автоматично аналізується на сервері.

Рисунок 1.1 – Merlin Bird Ai Preview

Значною перевагою Merlin є висока точність та інтеграція з глобальною екосистемою даних, однак доступ до API обмежений, а сам алгоритм є закритим, що ускладнює його повторне використання в інших проєктах

BirdNET. Ще один продукт Корнелльської лабораторії, створений у співпраці з Технологічним інститутом у Кемніці (Німеччина), – BirdNET – є відкритим дослідницьким проєктом. Він базується на глибокій нейронній мережі, що використовує спектрограми аудіосигналів для класифікації. BirdNET надає веб-інтерфейс і мобільні додатки для Android та iOS, а також має відкритий API у вигляді експериментального веб-сервісу.

BirdNET підтримує розпізнавання понад 3 тисяч видів птахів у різних регіонах світу. Серед його сильних сторін – точність, підтримка великої кількості мов, візуалізація ймовірностей класифікації, а також наукова відкритість. Водночас система залежить від підключення до Інтернету та централізованого сервера, що не завжди зручно для польових умов.

Warblr і ChirpOMatic. Ці системи були розроблені переважно для європейського ринку та мають менший обсяг видів у порівнянні з глобальними рішеннями. Warblr – британський стартап, що використовує алгоритми машинного навчання для розпізнавання звуків близько 200 видів птахів Великобританії. ChirpOMatic орієнтований на офлайн-роботу, що дозволяє використовувати його без підключення до мережі, але обмежує складність обробки та обсяг моделі.

Song Sleuth. Розробка для американського ринку, що дозволяє записувати та класифікувати пташині голоси з використанням локальної нейронної мережі. Має вбудовану бібліотеку аудіоприкладів та можливість редагувати результати. Основними обмеженнями є підтримка лише близько 200 видів та необхідність ручного втручання користувача для уточнення класифікації.

OpenSoundscape, Kaleidoscope, Wildlife Acoustics. Ці інструменти призначені здебільшого для професійного біоакустичного моніторингу, зокрема в дослідницьких і природоохоронних організаціях. Вони надають

інструменти для обробки великих масивів аудіоданих, фільтрації, анотації та автоматичної класифікації. Проте вони не орієнтовані на звичайних користувачів і потребують спеціалізованих знань.

В результаті маємо, що більшість існуючих систем мають ті чи інші обмеження – або у відкритості та інтеграції, або у масштабованості, локалізації чи доступності для непрофесійної аудиторії. Відсутність кросплатформного мобільного рішення з відкритим API, локалізованого під потреби українських користувачів, актуалізує розробку нової системи, яка б об'єднувала високий рівень автоматизації, доступність для широкого загалу та технічну відкритість для подальшого розвитку.

1.3 Особливості аудіосигналів пташок як об’єкта класифікації

Акустичні сигнали птахів відіграють фундаментальну роль у їхній життєдіяльності, слугуючи основним засобом комунікації. За допомогою звуків птахи маркують свою територію, приваблюють партнерів, сповіщають про небезпеку та підтримують соціальну структуру популяцій. Ці сигнали, як правило, є видовими маркерами, що дозволяють здійснювати точну ідентифікацію виду без потреби візуального контакту. Саме тому аналіз пташиних голосів уже давно є важливим інструментом у прикладній орнітології, біоакустичному моніторингу та екологічних дослідженнях.

Водночас аудіосигнали птахів є складним об’єктом для автоматизованої класифікації. Їх характерною рисою є висока варіативність, яка проявляється як на міжвидовому, так і на внутрішньовидовому рівнях. Географічні діалекти, індивідуальні особливості, вікові та статеві відмінності, а також сезонні зміни у вокалізації – все це суттєво ускладнює завдання ідентифікації. Навіть голоси одного й того ж виду можуть суттєво різнитися залежно від регіону чи особистих характеристик особини. Для

моделей глибокого навчання така варіативність означає необхідність навчання на великих обсягах репрезентативних даних, що охоплюють різноманіття реальних акустичних варіацій.

Пташині сигнали мають складну часово-частотну структуру. Вони зазвичай складаються з коротких елементів тривалістю від кількох сотих секунди до кількох секунд, часто містять тональні компоненти з різними гармоніками, модуляціями частоти та складними звуковими патернами. Частотний діапазон голосів багатьох видів птахів коливається в межах 1–10 кГц, іноді досягаючи ще вищих значень. У таких умовах традиційні підходи до аналізу аудіо, як правило, є недостатніми, і виникає потреба у використанні перетворень на спектрограму, мел-спектрограму або MFCC (Mel-Frequency Cepstral Coefficients), схема якої зображена на рисунку 1.2, що дозволяють виділити ознаки, релевантні для подальшої класифікації.

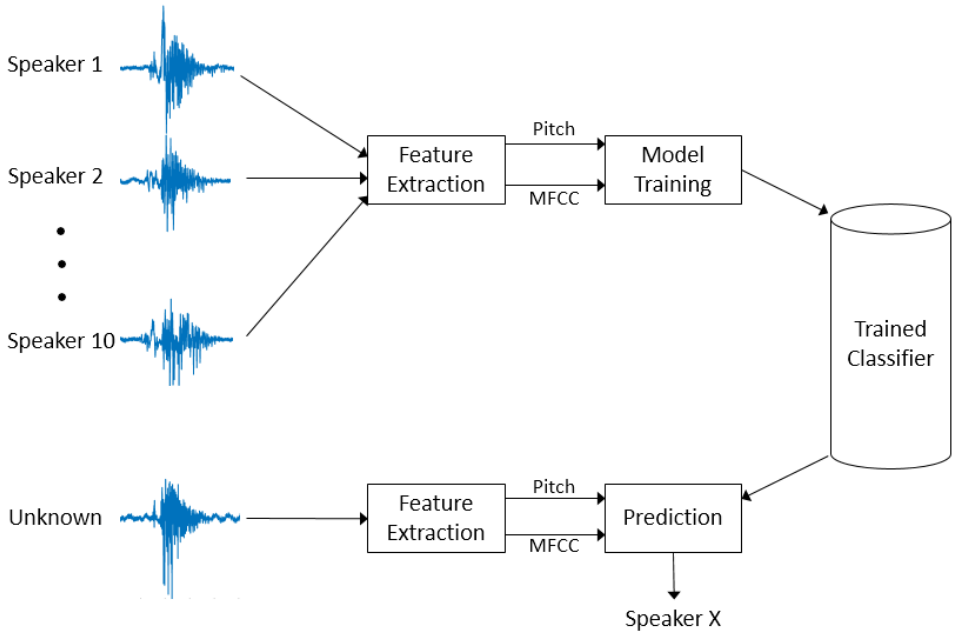

Рисунок 1.2 – Використання MFCC для ідентифікації по голосу[1]

[1]зображення з роботи Jaddu, N.S., Shashank, S.R.S., Suresh, A. (2023). Voice Emotion Detection: Acoustic Features Extraction Using Multi-layer Perceptron Classifier Algorithm. In: Gupta, D., Khanna, A., Hassanien, A.E., Anand, S., Jaiswal, A. (eds) International Conference on Innovative Computing and Communications. Lecture Notes in Networks and Systems, vol 492. Springer, Singapore. https://doi.org/10.1007/978-981-19-3679-1_49

Суттєвою складністю при роботі з біоакустичними даними є вплив фонового шуму. Оскільки записи птахів здебільшого здійснюються у відкритому природному середовищі, аудіосигнали нерідко містять додаткові звуки – спів інших птахів, шум вітру, дощу, комах, людей, транспорту тощо. Такий акустичний шум знижує співвідношення сигнал/шум, що негативно впливає на точність класифікації. Тому сучасні системи автоматичного розпізнавання повинні включати етапи попередньої обробки даних, як-от фільтрація, нормалізація або спектральна вирівнюваність. Крім того, бажаною є стійкість моделі до зашумлених сигналів, що часто досягається шляхом спеціального навчання з додаванням шумів до навчального набору.
Ще однією важливою проблемою є поліфонія – ситуація, коли одночасно чути кількох птахів, які можуть належати до різних видів. У таких випадках аудіофрагмент містить змішані звукові сигнатури, що ускладнює задачу класифікації, особливо при використанні одноетикеткових моделей. Рішенням може бути застосування підходів мультиетикеткової класифікації (multi-label classification) або виділення джерел звуку через source separation. Водночас ці підходи значно підвищують обчислювальну складність системи.

Додатковим фактором, що впливає на характеристику аудіосигналів, є сезонність і добові ритми. Як правило, найвища вокальна активність птахів припадає на весняно-літній період, особливо на ранкові години – так зване «ранкове хорове співання». Це слід враховувати при побудові систем розпізнавання, оскільки характеристики записів можуть суттєво змінюватися залежно від часу і сезону. Моделі повинні бути здатними до адаптації до таких змін, ідеально – шляхом постійного оновлення даних і донавчання.

Таким чином, пташині аудіосигнали є цінним джерелом інформації для екологічного моніторингу та вивчення біорізноманіття, але водночас становлять складний об’єкт класифікації з погляду технічної реалізації. Висока варіативність, зашумленість, поліфонія та сезонні зміни вимагають використання сучасних методів глибокого навчання, здатних до узагальнення та роботи в реальних умовах.

1.4 Проблеми та виклики при класифікації біоакустичних сигналів

Автоматична класифікація пташиних аудіосигналів – це, на перший погляд, досить приваблива задача: уявіть, що достатньо просто підняти телефон до неба, і за кілька секунд ви знаєте, хто саме із пернатих щойно заспівав у найближчому кущі. Але за цією уявною простотою ховається ціла низка викликів – технічних, концептуальних, навіть філософських.

Першою і, мабуть, найбільш очевидною проблемою є завадність аудіоданих. Природа – не студія звукозапису. Пташині голоси рідко звучать у вакуумі. У лісі чи парку звукове тло формується хаотично: тріскіт гілок, шелест листя, гул міста, а часом і спів одразу кількох видів птахів – усе це створює складну поліфонічну сцену. Для людського вуха це, можливо, чарівна симфонія, але для нейронної мережі – надскладний набір переплетених спектральних образів, де один звук накладається на інший, змазує контури, а іноді повністю зникає під маскою шуму. Навіть новітні архітектури глибокого навчання, здатні виявляти патерни в мільйонах прикладів, не завжди "розуміють", що саме в цих звуках є релевантним.

Інша складність – висока варіативність вокалізацій одного й того ж виду. Наприклад, жайворонок польовий у Франції і той же вид десь під Харковом можуть співати по-різному – настільки, що для непідготовленого слухача це будуть два різні птахи. А для нейромережі – це два спектрограми, між якими мало спільного, якщо вона не навчена розуміти внутрішню логіку варіацій. Ця проблема частково вирішується за рахунок великих датасетів, але тут виникає інша дилема – брак якісно розмічених записів.

У публічних архівах на кшталт Xeno-Canto чи Macaulay Library дійсно є тисячі аудіозаписів. Проте їх якість і точність маркування часто залишає бажати кращого. Іноді одна й та сама аудіодоріжка містить голоси кількох птахів, хоча вказаний лише один. А буває, що запис зроблений на дешевий мікрофон з постійними сторонніми шумами. Такі неточності у даних – це, як

для системи розпізнавання обличчя, намагатися вивчити портрети людей, намальовані від руки.

До того ж, не всі птахи взагалі охоче співають. Деякі види проявляють вокальну активність лише кілька тижнів на рік, або ж співають переважно вночі. Є птахи, голос яких ледь вловимий для більшості мікрофонів – він губиться серед інших частот. Іншими словами, акустичний слід виду – не завжди стабільний і доступний. Це означає, що навіть найкраща модель не зможе "впізнати" того, кого в неї ніколи не було у прикладах.

Ще один, більш технічний аспект – дизайн самої моделі. Не кожна архітектура нейронної мережі однаково добре працює з аудіо. Наприклад, класичні згорткові мережі (CNN) чудово ловлять локальні патерни на спектрограмі, але часто втрачають часову динаміку співу. Рекурентні мережі (RNN), навпаки, розуміють послідовність, але важко масштабуються. Гібриди типу CRNN – спроба об’єднати сильні сторони обох, проте вони вимогливі до ресурсів і складні в налаштуванні. Тут нема "срібної кулі", і кожен варіант – це компроміс між точністю, швидкістю і реалістичністю застосування, особливо в мобільних умовах.

І наостанок – людський фактор. Система класифікації – це не лише набір кодів і моделей, це взаємодія з користувачем. А користувач, як відомо, не завжди читає інструкції, може записати пташку з кишені або під час грози, а потім запитати: "Чому нічого не працює?". У цьому сенсі, точність моделі – це лише половина успіху. Інша половина – це UX, тобто як система справляється з помилками, як вона дає зворотний зв’язок, як "чесно" повідомляє про невпевненість у результаті. Бо в реальному житті краще дати відповідь "не знаю", ніж вигадати птаха, якого не існує**.**

1.5 Огляд сучасних підходів до обробки аудіо з використанням глибокого навчання

Коли ми говоримо про глибоке навчання в контексті звуку, перше, що спадає на думку – це мовлення, розпізнавання голосу, Alexa або Siri. Але музика, навколишнє середовище, і особливо – біоакустика – це зовсім інша історія. Тут правила гри диктують не люди, а природа, яка не дотримується логіки синтаксису або фонем. Пташиний спів – це фрагментарний, ритмічно нестабільний, нерівномірний сигнал. І саме це створює унікальний набір викликів – і дає змогу оцінити силу сучасних моделей глибокого навчання.

Один із базових принципів у роботі з аудіо – це перетворення сигналу у форму, яку "розуміє" нейронна мережа. Сирий аудіосигнал, тобто хвиля у часовому просторі (waveform), хоч і є технічно повним представленням звуку, зазвичай непридатний для прямого подання в модель. Його складно "прочитати", бо в ньому відсутні очевидні просторові ознаки, які так люблять CNN (згорткові нейронні мережі). Тому у сучасних підходах використовують спектральні подання – спектрограми, мел-спектрограми, або MFCC (Mel-Frequency Cepstral Coefficients). Уявіть, що ми беремо звук і «фотографуємо» його розвиток частот у часі – отримаємо такий собі "знімок звуку", з яким уже можна працювати як із зображенням.

Саме тут і вступають у гру згорткові нейронні мережі (CNN), які вміють виявляти локальні шаблони – наприклад, характерну форму "свисту" чи "трелі" конкретного виду птаха. Ці мережі вдало використовуються у багатьох задачах, зокрема й у роботах типу BirdCLEF Challenge (організованих спільнотою LifeCLEF), де щорічно аналізується ефективність моделей у класифікації пташиних голосів у реальних аудіозаписах.

Проте CNN мають одну очевидну слабкість: вони не завжди "відчувають" час. Для них спектрограма – це просто картинка, а не серія подій, що розгортаються у часовому просторі. А у пташиному співі час –

критичний. Порядок нот, тривалість пауз, ритм – усе це може бути ключем до класифікації. Саме тому дослідники почали активно використовувати рекурентні нейронні мережі (RNN) та їх покращені версії, зокрема LSTM (Long Short-Term Memory) або GRU (Gated Recurrent Units), які пам’ятають попередні стани та можуть враховувати послідовність звучання.

Проте RNN – це завжди компроміс між ефективністю і гнучкістю. Вони хороші у відстеженні часової динаміки, але працюють повільно, погано масштабуються і часто мають проблеми з паралелізацією обчислень. Щоб об'єднати переваги CNN та RNN, були запропоновані гібридні архітектури – CRNN (Convolutional Recurrent Neural Networks). Вони обробляють спектрограму згортками (виділяючи ознаки), а потім передають ці ознаки в рекурентну частину, яка аналізує їх часову структуру. Ці моделі вже використовуються в реальних проєктах для моніторингу дикої природи, наприклад, у системах BirdNET, Warblr та ChirpOMatic – кожна з яких має свою стратегію балансування між точністю та швидкістю.

І, звісно, не можна оминути трансформери (Transformers). Хоча їх спершу створили для обробки природної мови, трансформер-архітектури поступово завойовують і звукову сферу. Моделі типу AST (Audio Spectrogram Transformer) або Wav2Vec 2.0 показують надзвичайні результати в задачах класифікації та сегментації аудіо. Вони не просто враховують час – вони будують залежності між усіма точками сигналу одразу, створюючи цілісну картину. Але й тут є нюанс: такі моделі вимогливі до обчислювальних ресурсів і не завжди підходять для мобільних пристроїв – тому досі залишаються більш науковим інструментом, ніж рішенням "у кишені".

Ще один аспект, який часто недооцінюють – це аугментація аудіо. Як і у випадку з зображеннями, штучне "збільшення" набору даних за допомогою шумів, зсувів у часі, змін висоти тону або гучності, дозволяє створити більш стійку до змін модель. У задачах біоакустики це майже обов’язкова практика, адже реальний світ – непередбачуваний.

1.6 Можливості використання мобільних застосунків у біоакустичному моніторингу

Мобільні застосунки, що працюють з аудіо, вже не є дивиною. Існують продукти, орієнтовані на виявлення звуків природи, аналіз музики, а також на задачі екологічного моніторингу. У випадку з орнітологією, де головним джерелом інформації є саме звуковий сигнал, потенціал смартфона як інструмента виявлення, класифікації і навіть збереження звуків складно переоцінити. Наприклад, популярний додаток Merlin Bird ID, розроблений Cornell Lab of Ornithology, дає змогу розпізнавати голоси птахів у режимі реального часу – і це вже не просто «гаджет для орнітологів», а частина глобального проєкту залучення громадськості до спостереження за дикою природою (citizen science).

Однак не лише зручність робить мобільні застосунки важливими. Вони розширюють географію дослідження. Замість дорогого стаціонарного обладнання – лише смартфон, який можна взяти в джунглі, степ, міський парк. У цьому й криється революційна перевага: доступність. І тут уже важко відділити науку від культури. Людина, яка вперше дізналася, що «той дивний звук за вікном» – це, скажімо, спів вівчарика чорноголового (Sylvia atricapilla), не просто отримала факт. Вона встановила зв’язок із природою. У певному сенсі, мобільні додатки здатні перетворити пересічного міського жителя на свідомого спостерігача.

З іншого боку, варто визнати і технологічні обмеження. Мікрофони смартфонів, хоча й стали набагато якіснішими, не завжди спроможні захопити тихі або високочастотні звуки з необхідною точністю. Проблеми також виникають із фоновим шумом, змінами погоди, різним рівнем гучності, накладенням звуків. Це все – виклики, які потребують ретельної обробки сигналів, адаптивних моделей і стратегії роботи з нечистими даними. Тим не менш, уже існують підходи, що частково долають ці

недоліки. Наприклад, системи попередньої фільтрації шумів або аугментації даних у процесі навчання моделей.

Окрема увага – користувацькому інтерфейсу. Що простіший і інтуїтивніший додаток, то вища ймовірність, що ним скористаються не тільки ентузіасти, а й звичайні користувачі. Тобто не потрібно знати, що таке мел-спектрограма або RNN – достатньо натиснути "записати", і за кілька секунд отримати відповідь. У цьому й полягає головна перевага мобільного підходу – він опускає поріг входу в складну наукову дисципліну.

Висновок до розділу 1

Підсумовуючи проведене дослідження предметної області, варто зазначити, що завдання автоматизованої класифікації пташиних голосів на основі аудіосигналів належить до міждисциплінарної сфери, що охоплює біоакустичний аналіз, цифрову обробку сигналів, штучний інтелект і мобільні технології. Актуальність теми підтверджується не лише екологічними й науковими потребами, а й зростаючим інтересом з боку звичайних людей, які прагнуть глибше пізнати навколишнє середовище та дізнатися більше про птахів, що їх оточують.

Огляд сучасних систем виявив, що, попри наявність досить ефективних і навіть комерційно успішних рішень, таких як Merlin Bird ID або BirdNET, переважна більшість із них зосереджена на користувачах з певним рівнем технічної чи орнітологічної підготовки. Це створює нішу для застосунків, які поєднують простоту, доступність та високоточні алгоритми – зокрема, на основі глибокого навчання.

Проаналізувавши особливості пташиних аудіосигналів, стає зрозуміло, що їх структура є вкрай варіативною та багатошаровою. Пісня птаха – це не просто звук, а носій складної інформації: про видову приналежність,

емоційний стан, територіальність, шлюбну активність. Це підкреслює складність задачі класифікації й водночас відкриває широкі горизонти для алгоритмічного аналізу.

Проблеми, з якими стикається класифікація біоакустичних даних, – фоновий шум, перекриття сигналів, обмеженість навчальних вибірок – змушують дослідників постійно шукати нові підходи. І глибоке навчання, зокрема моделі CNN, CRNN, RNN і трансформери, демонструє сьогодні найбільшу ефективність у розв’язанні таких задач.

Особливо перспективним виглядає поєднання цих технологій із мобільними платформами. Смартфон перетворюється на інструмент, який дозволяє не лише розпізнавати голоси птахів, а й, потенційно, зберігати аудіоархіви, виводити статистику спостережень, брати участь у краудсорсингових проєктах та сприяти збереженню біорізноманіття. Таким чином, мобільний застосунок – це не лише зручний інтерфейс, а міст між високотехнологічними обчисленнями та природним середовищем.

РОЗДІЛ 2 ПІДХОДИ ДО РЕАЛІЗАЦІЇ ЗАСТОСУНКУ ДЛЯ РЕКОМЕНДАЦІЙ

2.1 Основи цифрової обробки аудіосигналів

Цифрова обробка аудіосигналів є фундаментальною складовою будь-якої системи, що працює зі звуковою інформацією, включно з проєктами біоакустичного моніторингу. Розпізнавання пташиних голосів базується саме на здатності точно представити, зберегти, аналізувати й інтерпретувати звук у цифровому вигляді. Звук як фізичне явище являє собою механічні коливання середовища, найчастіше повітря, які можна представити у вигляді аналогового сигналу – безперервної функції часу. Проте для аналізу його комп'ютерними алгоритмами необхідне попереднє перетворення цього сигналу в дискретну форму – тобто в послідовність чисел.

Цей процес називається аналого-цифровим перетворенням (АЦП) і включає два ключові етапи: дискретизацію та квантування. Дискретизація – це вимірювання значення сигналу у фіксовані моменти часу. Частота дискретизації (sampling rate), що визначає кількість таких вимірів за секунду, напряму впливає на здатність системи точно передати високочастотні компоненти звуку. Наприклад, для людської мови чи музики зазвичай використовується частота 44.1 кГц, однак для біоакустичних задач, де пташині сигнали можуть мати елементи у високочастотному діапазоні, доцільним є використання частот 48 або навіть 96 кГц. Квантування ж означає округлення значень сигналу до обмеженої кількості рівнів – це зумовлює точність відображення амплітуди звуку.

Після перетворення в цифрову форму аудіосигнали зазвичай обробляються з використанням різноманітних алгоритмів. Найпоширенішими операціями є фільтрація шумів, нормалізація гучності, обрізання або сегментація сигналів, а також побудова спектральних представлень. Ці процедури мають на меті зменшення впливу фонових

перешкод, стандартизацію даних для навчання моделей та виділення ознак, які найкраще репрезентують акустичний образ досліджуваного об’єкта.

Окремо варто зупинитися на форматах зберігання аудіоданих. Найбільш поширеними у дослідженнях є формати WAV та FLAC – безстратні, тобто такі, що зберігають первинну якість сигналу. Використання стиснутих форматів, як-от MP3, може бути допустимим для попереднього ознайомлення, проте небажаним для подальшого аналізу, оскільки втрата високочастотних компонентів, характерна для таких форматів, може унеможливити розпізнавання специфічних звуків, що притаманні деяким видам птахів.

У контексті систем глибокого навчання цифровий аудіосигнал здебільшого розглядається як тимчасовий ряд або трансформується у спектральне зображення, яке вже може бути передано на вхід нейронної мережі. Це означає, що ще до фази навчання система має мати чітко структуровані, чисті та добре описані дані – і саме цифрова обробка на попередньому етапі відіграє тут ключову роль.

Отже, основи цифрової обробки аудіосигналів забезпечують ту необхідну технічну базу, яка дозволяє переходити від аналогового, складного та хаотичного звуку до структурованого, придатного до аналізу цифрового представлення. Саме ця трансформація відкриває можливість застосування сучасних інтелектуальних алгоритмів, що лежать в основі автоматизованого розпізнавання пташиних голосів.

2.2 Перетворення Фур’є, спектрограми та мел-спектрограми

У більшості систем класифікації аудіосигналів первинний часовий ряд (waveform) не використовується безпосередньо, оскільки він є надто «сирим» і слабо інформативним для нейронних мереж. Замість цього сигнал зазвичай

перетворюється у частотну область, яка дозволяє виявити спектральні характеристики, що є ключовими для ідентифікації джерела звуку. Найбільш поширеним математичним інструментом для такого перетворення є перетворення Фур'є.

Неперервне перетворення Фур'є (Continuous Fourier Transform, CFT) описується рівнянням:

$$F(\omega) = \int_{-\infty}^{+\infty} x(t) \cdot e^{-i\omega t} dt,$$

де $F(\omega)-$ спектральне представлення функції

$f(t)$ – початкова функція у часовій області,

ω – кругова частота (в радіанах за секунду);

Для цифрових сигналів використовується дискретне перетворення Фур'є (DFT):

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-i\frac{2\pi}{N}kn}, \quad k = 0,1,2,\dots,N-1,$$

де $x[n]$ – вхідний дискретний сигнал (у часовій області), який складається з N точок,

$X[k]$ – спектр сигналу (в частотній області),

N – кількість точок (довжина сигналу),

k – індекс частоти.

На практиці застосовується його ефективна реалізація – швидке перетворення Фур'є (FFT), що знижує обчислювальну складність з $O(N^2)$ до $O(N \log N)$.

Проте звичайне перетворення Фур'є втрачає інформацію про час. Для аналізу нестаціонарних сигналів, таких як голоси птахів, де частотна

структура змінюється з часом, використовують віконне (або короткочасне) перетворення Фур'є (STFT):

$$X(t,f) = \int_{-\infty}^{+\infty} x(\tau) \cdot \omega(\tau - t) \cdot e^{-2\pi i f t} d\tau,$$

де $\omega(\tau - t)$ – віконна функція, що обмежує ділянку сигналу в околі моменту часу t. Результатом такого перетворення є спектрограма – матриця, що відображає інтенсивність частот у функції часу. У ній вісь x – час, вісь y– частота, а кольорова шкала позначає амплітуду або енергію сигналу.

Для візуалізації зазвичай застосовують логарифмічне масштабування амплітуди (в децибелах):

$$S_{dB}(t,f) = 10 \cdot \lg|X(t,f)|^2.$$

Однак людське (і, у певному сенсі, «біологічне») сприйняття звуку не є лінійним по частоті. Щоб відобразити частотну чутливість наближено до того, як її інтерпретує слух, вводиться мел-шкала. Вона побудована таким чином, що нижні частоти мають високу роздільну здатність, а вищі – нижчу, що відповідає особливостям сприйняття.

Перехід від частоти f у герцах до мел-частоти здійснюється за формулою:

$$m = 2595 \cdot \lg\left(1 + \frac{f}{100}\right).$$

Мел-спектрограма, отже, є спектрограмою, де частоти перетворено за мел-шкалою, а енергії згладжено відповідно до фільтрбанку з трикутними фільтрами. У результаті отримується двовимірне зображення, яке одночасно містить часову і частотну інформацію, оптимізовану під сприйняття.

Для систем глибокого навчання, зокрема згорткових нейронних мереж (CNN), мел-спектрограми виступають «зображеннями», які подаються на вхід моделі подібно до фотографій. Їх структура дозволяє ефективно виділяти ознаки, що притаманні голосу певного виду птаха, – наприклад, характерну гармонійну структуру або амплітудну модуляцію.

На практиці найбільш уживаними форматами спектрограм є:

- STFT спектрограми – для аналізу загальної частотної структури;
- Mel Spectrogram – для подачі на вхід нейромереж;
- MFCC (Mel-Frequency Cepstral Coefficients) – як компактне представлення аудіо.

Підсумовуючи перетворення Фур'є та його похідні не лише відіграють роль математичного фундаменту в цифровій обробці аудіосигналів, а й забезпечують міст між «сирим» звуком і тією формою, яка придатна до навчання глибоких моделей класифікації.

2.3 Основи глибокого навчання та штучних нейронних мереж

Глибоке навчання, як підмножина машинного навчання, є основною рушійною силою сучасних систем автоматичної обробки складних структурованих даних – зображень, мови, аудіосигналів, відео. Його ефективність зумовлена архітектурами з багатьма шарами нелінійних перетворень, які дають змогу поступово виділяти дедалі абстрактніші ознаки з вхідної інформації. Особливо це актуально для задач класифікації, де необхідно розпізнати складні патерни, – у нашому випадку це характерні акустичні профілі голосів птахів.

У центрі глибокого навчання лежить концепція штучної нейронної мережі, схема якої зображена на рисунку 2.1, – обчислювальної моделі натхненної принципами роботи біологічного мозку.

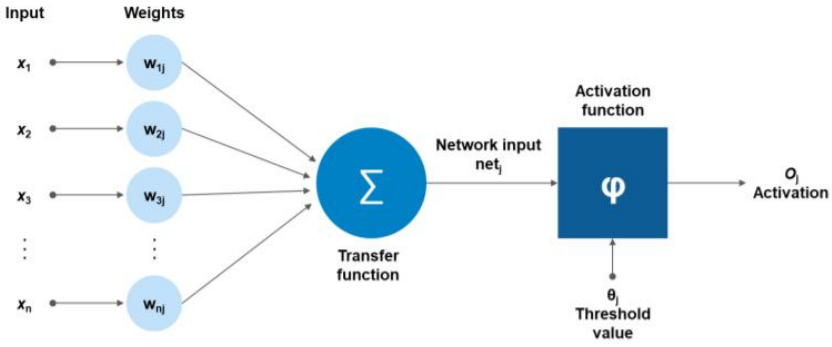

Рисунок 2.1 – Схема штучного нейрону[2]

Хоча аналогія є досить умовною, загальна ідея залишається близькою: одиничні обчислювальні елементи, які називаються нейронами, з'єднані між собою і обмінюються сигналами. Кожен нейрон отримує вхідні значення, зважує їх, підсумовує і передає результат через деяку активаційну функцію.

Найпростіший варіант такого обчислення описується формулою:

$$y = \phi\left(\sum_{i=1}^{n} \omega_i x_i + b\right),$$

де x_i – вхідні значення,

ω_i – вагові коефіцієнти,

b – зміщення,

ϕ – активаційна функція, яка додає нелінійність,

y – вихід нейрона

Активаційні функції можуть бути різними: класична сигмоїда, гіперболічний тангенс, або ж сучасна ReLU (Rectified Linear Unit), що зображена на рисунку 2.2, яка визначається як $\phi(x) = \max(0, x)$. Вибір функції суттєво впливає на ефективність навчання мережі.

[2] Джерело зображення: https://towardsdatascience.com/wp-content/uploads/2021/11/1sEB2NV2j364C1zMHGXXONg.png

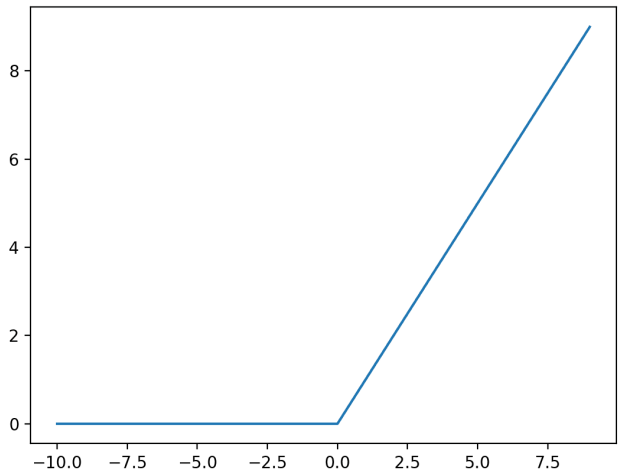

Рисунок 2.2 – Графік ReLU[3]

Мережі, що складаються з кількох таких шарів, називаються глибокими. Типова глибока нейронна мережа охоплює:

- вхідний шар, який приймає дані (наприклад, спектрограми);
- кілька прихованих шарів, де відбувається обробка ознак;
- вихідний шар, що генерує рішення – наприклад, до якого виду належить аудіозапис.

Процес навчання нейронної мережі полягає у налаштуванні ваг ω_i так, щоб мінімізувати похибку між реальними та передбаченими відповідями. Це здійснюється шляхом зворотного поширення помилки (backpropagation) у поєднанні з методом градієнтного спуску (gradient descent). Функція втрат (loss function), яку оптимізує мережа, залежить від конкретної задачі. Для класифікації зазвичай використовується крос-ентропія:

$$L = -\sum_i y_i \cdot \log \hat{y}_i,$$

де y_i – істинне значення (one-hot encoding), $\hat{y}_i$ – передбачена ймовірність.

[3] Джерело зображення: https://docs.google.com/document/d/1SrII8UHhath-gqq2VBG7RhjAcq_Bfjwd/edit#heading=h.to5ejobm3jw2

Окрім традиційних повнозв’язних мереж (Fully Connected), сучасне глибоке навчання активно використовує спеціалізовані архітектури, оптимізовані під тип даних. Для обробки аудіо це, зокрема, згорткові (CNN), рекурентні (RNN) та комбіновані (CRNN) мережі – про них буде докладніше у наступному підрозділі. Застосування таких архітектур дає змогу ефективно виділяти як просторові (частотні), так і часові залежності у звуковому потоці.

Варто зазначити, що сила глибоких моделей не лише у їхній складності, а й у здатності до автоматичного навчання ознак – замість ручного виділення дескрипторів (як це було раніше), модель сама знаходить ті патерни, які є релевантними для класифікації. Це і є ключовим зрушенням у парадигмі машинного навчання останніх десятиліть.

Таким чином, глибоке навчання надає потужний інструментарій для побудови систем, здатних розпізнавати види птахів на основі їхнього голосу, – навіть за наявності фонових шумів, неоднорідної тривалості сигналів та інших складних акустичних умов. Успіх таких систем безпосередньо залежить від правильного вибору архітектури мережі, якості вхідних даних та обчислювальної дисципліни при їхньому тренуванні.

2.4 Архітектури нейронних мереж для обробки аудіо: CNN, RNN, CRNN

Обробка аудіосигналів у задачах класифікації видів птахів вимагає поєднання просторової та часової чутливості. Іншими словами, модель має не лише виявити характерні спектральні патерни – скажімо, певну гармонічну структуру або шумовий «почерк» конкретного виду, – а й врахувати їх послідовність у часі. Саме тому класичні повнозв’язні мережі не дають достатньої гнучкості для подібних задач. У цій частині розглянемо три ключові архітектури, що використовуються для аналізу аудіо: згорткові

нейронні мережі (CNN), рекурентні нейронні мережі (RNN) та їх комбінацію – CRNN.

CNN були спочатку розроблені для обробки зображень, однак їхній принцип чудово переноситься на спектрограми, які є візуальним представленням звуку. Основною операцією у CNN є згортка (convolution) – локальне фільтрування входу з метою виявлення специфічних ознак. Наприклад, при обробці мел-спектрограми згортковий фільтр може «впіймати» частотну послідовність, яка характерна для певного виду птаха. Хід роботи згортки зображений на рисунку 2.3.

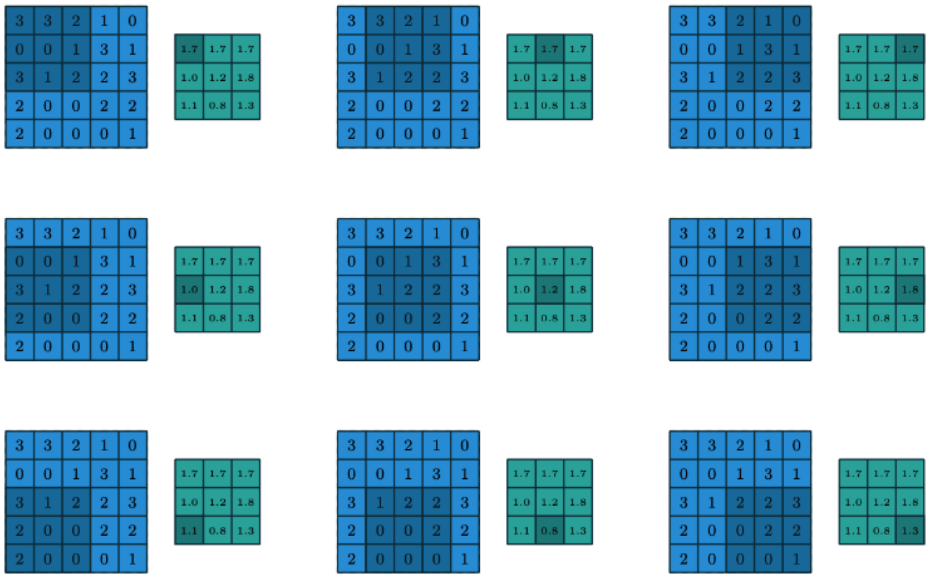

Рисунок 2.3 – Хід роботи згортки[4]

Формально, згортка між сигналом xта ядром ω у 2D-просторі визначається як:

[4] Зображення з роботи Dumoulin, Vincent & Visin, Francesco. (2016). A guide to convolution arithmetic for deep learning. 10.48550/arXiv.1603.07285.

$$s(i,j) = (x * \omega)(i,j) = \sum_m \sum_n x(i+m, j+n) \cdot \omega(m,n).$$

Основна перевага CNN полягає у інваріантності до зсувів та виділенні локальних патернів, що є критично важливим для аналізу коротких звукових «фраз» або візерунків, які повторюються з варіаціями, CNN дозволяють суттєво зменшити кількість параметрів у моделі, що позитивно впливає на продуктивність при роботі на мобільних пристроях.

Однак CNN не можуть моделювати послідовні залежності – наприклад, зміну тону або ритмічну структуру крику птаха протягом кількох секунд. Для цього використовують RNN – архітектуру, яка має «пам'ять» про попередні стани. Кожен її елемент отримує не лише вхід x_t у момент часу t, а й прихований стан h_{t-1} з попереднього кроку:

$$h_t = \phi(W_{xh}x_t + W_{hh}h_{t-1} + b_h).$$

Цей підхід дає змогу RNN моделювати часові залежності, зберігаючи інформацію про попередній контекст. Проте класичні RNN мають труднощі з довгими послідовностями – вони схильні до проблем зникнення градієнта, що ускладнює навчання. Для подолання цього використовуються більш складні варіанти: LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit), які мають спеціальні механізми контролю «забування» та «оновлення» інформації у станах.

У випадках, коли важливо одночасно мати просторову і часову чутливість, застосовуються CRNN (Convolutional Recurrent Neural Networks). Ця архітектура поєднує переваги CNN та RNN: спочатку спектрограма подається до згорткових шарів, які виділяють локальні ознаки, а результати передаються до рекурентного блоку (зазвичай LSTM або GRU), що аналізує їх у часовому розрізі.

Такий підхід особливо ефективний для біоакустичних задач, де звуки мають як складну спектральну структуру, так і розгорнутий у часі контекст. Наприклад, початкове «привітання» з коротким свистом, за яким слідує тремтіння або щебетання – типовий патерн у пташиному співі, який CRNN може розпізнати, коли інші архітектури «втрачають логіку».

У практичних реалізаціях використовуються й інші покращення: ***dropout*** для боротьби з перенавчанням, ***batch normalization*** для стабілізації градієнтів, ***attention-механізми***, які дозволяють мережі «зосереджуватися» на важливих ділянках сигналу. Також поширене застосування ***residual-блоків*** у CNN, які забезпечують кращу передачу градієнтів при глибокій структурі мережі.

2.5 Підготовка аудіо для навчання моделі: нормалізація, розмітка, augmentation

Успішність глибокої моделі – не лише у правильній архітектурі чи алгоритмі оптимізації. Як показує практика, якість і спосіб підготовки вхідних даних здатні вплинути на кінцеву точність навіть більше, ніж обрана мережа. Особливо це стосується аудіосигналів, де сировина – звук – надзвичайно варіативна за якістю, гучністю, тривалістю, кількістю шумів. Тому процес передобробки аудіоданих можна порівняти з «очищенням об’єктива», крізь який модель спостерігає за реальністю. У цьому розділі розглянемо три ключові етапи підготовки: нормалізацію, розмітку та аугментацію.

Перший крок – нормалізація сигналу. Звуки, записані у полі, можуть бути надто гучними або надто тихими; навіть у межах одного виду птаха енергія сигналу часто коливається залежно від відстані, напрямку мікрофона,

акустики середовища, щоб усі зразки були однаково сприйнятними для моделі, виконується нормалізація амплітуди:

$$x_{norm}(t) = \frac{x(t)}{\max|x(t)|}.$$

Також застосовується нормалізація спектрограм за середнім значенням та дисперсією, особливо перед подачею в CNN:

$$x' = \frac{x - \mu}{\sigma}.$$

Наступний важливий аспект – розмітка аудіо. На перший погляд, достатньо лише вказати, якому виду належить запис, однак все складніше. Часто у фрагменті присутні голоси кількох птахів, або сам спів займає лише невелику частину треку. Тому важливо точно визначити часові межі вокалізації, уникати «порожніх» або змішаних сегментів, і створити балансовані класи (особливо коли одні види трапляються частіше за інші).

Існує кілька стратегій:

- Clip-level labeling – вся аудіодоріжка позначається одним ярликом;
- Frame-level labeling – мітки ставляться для кожного часово-фреймового сегмента;
- Weak labeling – лише ймовірність присутності певного виду без точної локалізації (для великих датасетів).

Проте навіть маючи якісні та чисті дані, ми все ще стикаємось із викликом – обмежена кількість прикладів, особливо для рідкісних видів. Тут на допомогу приходить аугментація – штучне збагачення набору даних шляхом варіації наявних зразків.

Для аудіо аугментація є надзвичайно гнучким інструментом. Серед поширених методів:

- Time-stretching – зміна темпу без зміни висоти;
- Pitch-shifting – зміщення тону (імітація іншого голосу того ж виду);
- Additive noise – додавання фонових звуків: шумів лісу, вітру, комах;
- Random gain – варіація гучності;
- Mixup – накладання кількох сигналів (інколи навіть різних видів, у weak-supervision режимі);
- SpecAugment – маскування частин спектрограми по частоті або часу (часто використовується в нейронних мережах на рівні спектрограм).

Ці методи не тільки збільшують загальний об’єм даних, а й допомагають моделі навчитися робастності до змін середовища: якщо ми навчаємо її на чистому сигналі, а на практиці запис матиме шурхіт листя – точність значно впаде. Аугментація дає змогу частково зняти це обмеження.

2.6 Метрики оцінки якості класифікації

Машинне навчання – це, по суті, постійне змагання між очікуваним і реальним результатом. Ми навчаємо модель розпізнавати пташині голоси, але як переконатися, що вона справді «чує» правильно? Просто перевірити, скільки вона вгадала – недостатньо. В реальних умовах, особливо коли йдеться про багатокласову або навіть multi-label класифікацію, потрібні точні, гнучкі й осмислені способи оцінки.

Розглянемо базові. У задачах з одним класом на зразок (тобто кожен аудіофрагмент відповідає одному виду птаха) найпростіша метрика – Accuracy:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}.$$

де TP – істинно позитивні відповіді, FP – хибнопозитивні, TN – істинно негативні, FN – хибнонегативні.

Але accuracy легко вводить в оману, коли класи незбалансовані. Наприклад, якщо 90% записів – це жайворонки, модель, що завжди обирає "жайворонок", матиме 90% точності, але жодної користі для решти видів.

Тому також використовують precision (точність), recall (повнота) та їх гармонійне середнє – F1-міру:

$$Precision = \frac{TP}{TP + FT},$$

$$Recall = \frac{TP}{TP + FN},$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}.$$

У багатокласових задачах ці метрики можна усереднювати:

- Macro-averaging: середнє значення для кожного класу, без врахування частоти;
- Weighted-averaging: середнє з урахуванням частоти кожного класу;
- Micro-averaging: розрахунок на рівні всіх зразків.

Ще один корисний інструмент – матриця невідповідностей (confusion matrix). Вона показує, як часто модель плутає один клас з іншим. Для задачі класифікації пташиних видів це особливо інформативно: наприклад, якщо

модель часто плутає синицю велику з блакитною, це може вказувати на схожість їхніх вокалізацій або проблеми в розмітці.

Коли в аудіо одночасно присутні кілька видів птахів (а так буває часто), застосовуються multi-label метрики. Наприклад, hamming loss (частка помилкових міток):

$$\text{H}amming\ Loss = \frac{1}{N \cdot L}\sum_{i=1}^{N}\sum_{j=1}^{L}\mathbb{I}[y_{ij} \neq \widehat{y_{ij}}].$$

Ще одна тонка деталь – вибір порогу. Моделі часто видають ймовірності, а не чіткі передбачення. Вибір порогу для кожного класу може суттєво вплинути на F1 або Recall. У деяких випадках доцільно динамічно підлаштовувати ці пороги, наприклад при deployment на мобільних пристроях, де бажано уникати хибних спрацювань.

Висновки до розділу 2

У цьому розділі було розглянуто фундаментальні математичні та технічні основи, що лежать в основі системи класифікації пташиних видів за аудіосигналами. Цифрова обробка звуку, спектральні перетворення (зокрема, перетворення Фур'є та мел-спектрограми), архітектури глибоких нейронних мереж (CNN, RNN, CRNN) – усі ці компоненти формують єдину структуру обчислювального слуху, який здатен ідентифікувати птаха за голосом.

Особливу увагу було приділено підготовці даних, адже саме на цьому етапі закладається потенціал навчання. Нормалізація, розмітка та аугментація формують основу, без якої жодна модель не зможе досягти стабільної та узагальненої роботи. Завершує технічний цикл система метрик, що дозволяє

об'єктивно оцінити якість класифікації, з урахуванням як практичних, так і статистичних аспектів.

Усе це створює надійне підґрунтя для реалізації повноцінного інтелектуального застосунку – не лише технічно ефективного, а й корисного для користувача. У наступних розділах увага буде зосереджена на архітектурі системи, особливостях мобільної реалізації та інтеграції з API.

РОЗДІЛ 3 ПРОЄКТУВАННЯ, РОЗРОБКА, ІНТЕГРАЦІЯ ТА АНАЛІЗ СИСТЕМИ

3.1 Постановка задачі та вимоги до системи

Задача, що вирішується в межах даної роботи, лежить на перетині кількох технічних і прикладних галузей – біоакустики, глибокого навчання, мобільної розробки та вебтехнологій. Основна ідея полягає в тому, щоб створити інтелектуальну систему, здатну класифікувати пташині голоси за аудіозаписами та робити це швидко, точно й у доступний для користувача спосіб. Модель, здатна розпізнавати голоси птахів – це лише частина загальної задачі. Не менш важливо – зробити її використання настільки простим, щоб будь-хто, почувши невідомий спів, міг отримати відповідь без знання ні біології, ні технологій.

На рівні формального формулювання передбачено створення системи з клієнт-серверною архітектурою, яка включає такі ключові компоненти:

- сервер, на якому розгорнута модель класифікації аудіо за пташиним голосом;
- відкритий API для прийому аудіозапитів та повернення результатів;
- вебінтерфейс для демонстрації чи тестування функціоналу;
- мобільний застосунок для кінцевих користувачів.

Кінцевий функціонал системи може виглядати просто: користувач обирає або записує аудіо, надсилає його, отримує список птахів з імовірностями розпізнавання. Але насправді за цією схемою ховається досить складна технічна конструкція. Важливо забезпечити точність класифікації, швидкість відповіді, стабільність серверу та коректну взаємодію між усіма компонентами.

Основні функціональні вимоги до системи такі:

- прийом аудіофайлу (у підтримуваних форматах, наприклад .mp3, .wav);
- обробка звуку серверною моделлю на основі глибокого навчання;
- генерація прогнозу: список імовірних пташиних видів із відповідними значеннями ймовірності;
- повернення результату у форматі, зручному для клієнта (JSON);
- простий інтерфейс взаємодії для мобільного користувача й вебвідвідувача.

Крім того, існують нефункціональні вимоги, що значною мірою визначають стабільність та практичність системи:

- масштабованість – можливість обробляти кілька запитів одночасно без втрати продуктивності.
- надійність – обробка помилок у даних або мережі без аварійного завершення.
- швидкодія – час відповіді сервера не має перевищувати кількох секунд навіть при середньому навантаженні.
- простота використання – ніяких облікових записів, складних форм або технічних термінів.
- відкритість – API має бути доступним не лише для мобільного застосунку, але й для зовнішніх розробників чи дослідників.

Окремою підзадачею постає питання обґрунтованого вибору інструментів – фреймворків, бібліотек, архітектурних рішень, від цього залежить не лише швидкість розробки, а й можливість підтримки системи в майбутньому.

У межах цієї роботи проєктується не просто класифікатор, а цілісна цифрова екосистема, що поєднує в собі складну глибоку модель і просту зовнішню оболонку. Саме тому в наступних підрозділах докладно розглянуто вибір кожного з її компонентів, від способу підготовки даних – до мобільного UX.

3.2 Вибір технологій і програмних інструментів

Проєктування будь-якої прикладної системи, що поєднує штучний інтелект із мобільною чи вебінфраструктурою, передбачає ухвалення серії технічно виважених рішень. Обрані інструменти не лише визначають можливості реалізації, але й формують обмеження, в яких функціонуватиме кінцевий продукт. У даному випадку – йдеться не просто про технологічну «зв'язку», а про забезпечення надійної взаємодії між глибокою нейромережею, сервером і двома клієнтськими інтерфейсами.

У якості моделі глибокого навчання було обрано архітектуру ***Wav2Vec2***, що розроблена компанією Facebook AI (тепер Meta AI), архітектура якої зображена на рисунку 3.1.

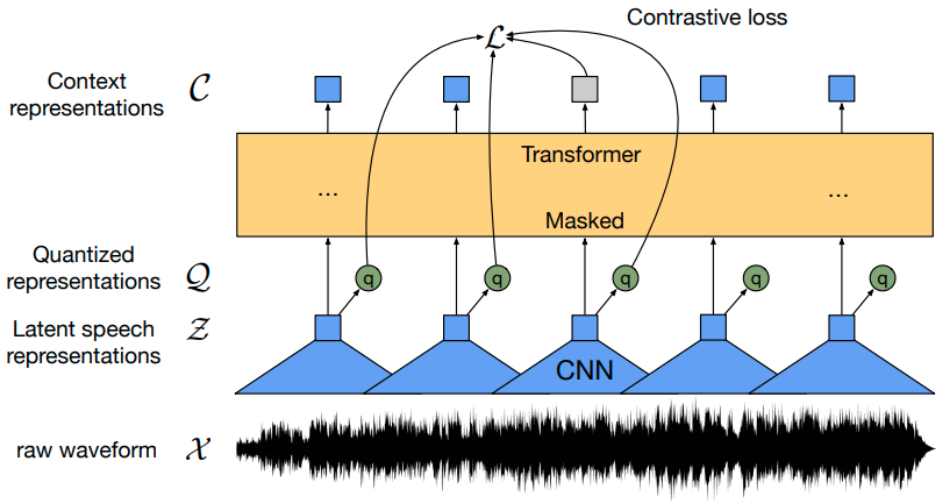

Рисунок 3.1 – Wav2vec2[5]

Ця модель показала виняткові результати у задачах розпізнавання мовлення, а згодом – і в загальніших аудіозавданнях. Її сильна сторона –

[5] https://www.researchgate.net/publication/378967143/figure/fig2/AS:11431281243192249@1715652372866/Wav2Vec-20-Architecture-Baevski-et-al-2020.png

здатність працювати без класичної попередньої обробки, такої як побудова мел-спектрограм: модель "чує" звук у сирому вигляді (waveform) і самостійно витягує ознаки. Водночас, завдяки відкритим реалізаціям у репозиторії HuggingFace Transformers, Wav2Vec2 може бути легко адаптована до задачі багатокласової класифікації, що й було реалізовано в цій бакалаврській роботі.

Для тренування та розгортання моделі використано екосистему HuggingFace (бібліотеки transformers, datasets, evaluate), а також PyTorch як базову фреймворк-платформу. HuggingFace дозволяє безболісно адаптувати готову архітектуру, змінити вихідний шар (classification head), а також організувати процес тренування за допомогою класу Trainer, який автоматизує багато рутинних кроків: збереження чекпоінтів, оцінку якості, логування, тощо. Перевагою такого підходу є зменшення часу розробки при збереженні повного контролю над структурою мережі.

На стороні сервера було вирішено використовувати Django – один із найстабільніших фреймворків для веброзробки на Python. Його інтеграція з Django REST Framework дала змогу швидко реалізувати API-ендпоінти для прийому аудіофайлів і передачі результатів класифікації у форматі JSON. Django також забезпечує зручну адмінпанель, яку в майбутньому можна використати для модерації або розширення функціоналу (наприклад, відстеження запитів чи додавання авторизації). Особливо важливе те, що реалізація бекенду мовою Python забезпечила легку інтеграцію з моделлю, що також була натренована в PyTorch.

Що стосується мобільного клієнта, вибір було зроблено на користь React Native з використанням середовища Expo. Це дало змогу швидко створити кросплатформний інтерфейс для Android та iOS без необхідності писати окремі нативні модулі. Expo надає вбудовані бібліотеки для доступу до мікрофона, файлової системи, HTTP-запитів, що істотно спростило реалізацію функцій запису або вибору аудіофайлів та їх передачі на сервер. Бібліотека axios була використана для організації запитів до API.

Окремо було реалізовано прототип вебінтерфейсу (на Django), який дозволяє завантажити аудіофайл через браузер, отримати й відобразити результати класифікації. Хоча цей інтерфейс не є основним засобом взаємодії, він дозволяє швидко тестувати API без мобільного пристрою.

Вибір усіх інструментів ґрунтувався на таких критеріях:

- відкритість і підтримка спільнотою;
- відповідність вимогам продуктивності та простоти;
- можливість швидкої інтеграції;
- підтримка сучасних стандартів розгортання (контейнеризації та хмарного хостингу).

Отже маємо сформований та узгоджений технологічний стек, що охоплює всі рівні – від моделі до користувацького інтерфейсу. У наступних підрозділах розглядатимуться особливості обробки даних, архітектура самої моделі та практичні аспекти її інтеграції в повноцінну систему.

3.3 Аналіз і підготовка датасетів для навчання

Якість навчання будь-якої моделі глибокого навчання критично залежить від вихідних даних. Особливо це стосується задач класифікації аудіо – пташині голоси можуть суттєво відрізнятися між собою за гучністю, тривалістю, фоновими шумами та навіть регіональними варіаціями. Усе це накладає додаткові вимоги до вибору джерела даних, способу попередньої обробки та структурування вибірки перед навчанням.

Для реалізації моделі у цій роботі було обрано комбінацію аудіозаписів з відкритих джерел, зокрема Xeno-canto – один із найбільших громадських архівів голосів птахів, та ресурсів, підготовлених у межах змагання BirdCLEF (серія конкурсів у рамках LifeCLEF, які фокусуються на автоматичному розпізнаванні біологічних звуків). Основною перевагою цих джерел є

наявність великої кількості реальних, «польових» записів, що дають змогу тренувати модель у максимально наближених до практичних умов обставинах.

Втім, використання таких датасетів також пов'язане з низкою викликів:

- ***неоднорідність тривалості записів***: у межах одного виду одні файли тривають кілька секунд, інші – хвилини.
- ***шуми навколишнього середовища***: вітер, комахи, людський голос, інші птахи – усе це ускладнює виділення цільового сигналу.
- ***нерівномірна кількість прикладів на клас***: одні види представлені сотнями зразків, інші – одиничними.

У зв'язку з цим було застосовано декілька рішень, які покращать роботу моделі та її якісь класифікації:

- ***очищення вибірки*** – ***автоматично*** відфільтровано записи з надто низьким рівнем гучності або некоректним форматом;
- ***сегментація*** – довгі ***аудіофайли*** було нарізано на фрагменти по 10 секунд. Це дозволило уніфікувати довжину навчальних прикладів і створити більш однорідні батчі;
- ***нормалізація*** – звукові амплітуди було масштабовано до фіксованого діапазону, що спрощує вхід до моделі;
- ***анотація класів*** – усім фрагментам надано числові ID відповідно до виду птаха.

Дані було перетворено у формат, сумісний із datasets.Dataset (бібліотека HuggingFace), що дозволяє ефективно управляти великою кількістю аудіофайлів, виконувати мапінги, розбиття на тренувальні та валідаційні підвибірки, а також інтегруватися з пайплайнами тренування моделі.

Крім технічної підготовки, важливим міркуванням при виборі датасетів була відкритість ліцензії – і BirdCLEF, і Xeno-canto надають дані на умовах, що дозволяють їх вільне використання в дослідницьких і освітніх проєктах,

за умови належного цитування. Це дало змогу уникнути обмежень у розповсюдженні, повторному використанні або публічному тестуванні системи.

Зібрана вибірка задовольняє головним критеріям: достатній обсяг, репрезентативність, структурованість і ліцензійна прозорість. Отриманий підготовлений датасет слугував основою для тренування моделі Wav2Vec2, про яку йтиметься у наступному підрозділі.

3.4 Архітектура глибокої моделі класифікації аудіо

У центрі реалізованої системи розпізнавання голосів птахів лежить модель глибокого навчання, побудована на базі ***Wav2Vec2*** – однієї з провідних архітектур для обробки звукових сигналів у сирому вигляді. Її ключова особливість полягає у відмові від традиційних інженерних ознак на кшталт спектрограм чи MFCC: модель працює безпосередньо з waveform-даними, тобто із "необробленим" звуком. Це дозволяє виявляти такі структури в аудіо, які складно виявити класичними методами, особливо в умовах шуму чи варіативного тембру.

Архітектурна основа. Wav2Vec2 – це трансформерна модель, що поєднує два основні блоки:

1. ***Feature encoder*** – згорткова нейромережа, яка перетворює сирий аудіосигнал у послідовність латентних ознак (зменшених у часовому вимірі).
2. ***Transformer encoder*** – багатошарова трансформерна мережа (на основі self-attention), яка моделює довгострокові залежності в отриманих ознаках та формує високорівневе уявлення про аудіо.

На відміну від моделей, що працюють із спектрограмами, Wav2Vec2 оперує аудіо як часовим рядом, що забезпечує гнучкішу адаптацію до

особливостей конкретного сигналу. Попередньо модель була натренована на величезних масивах мовних даних у самонавчальному режимі (self-supervised), що дозволило їй сформувати універсальні репрезентації – такі, які можуть бути корисними і поза межами розпізнавання мовлення.

Адаптація до задачі класифікації пташиних видів. Для потреб цієї роботи модель було модифіковано: з неї було знято вихідний шар, що відповідав за розпізнавання мовних одиниць, і замінено на класифікаційний шар (classification head), налаштований на кількість класів, що відповідає кількості видів птахів у підготовленому датасеті. Така модифікація є стандартною практикою transfer learning – використання великої попередньо навченого моделі як бази, до якої додається новий вузькоспеціалізований шар.

З технічного погляду це виглядає як заміна останнього шару на лінійну проєкцію:

$$\hat{y} = softmax(W \cdot h + b),$$

де h – вихід із трансформера, W– ваги нового класифікаційного шару, b – зміщення, $\hat{y}$ – ймовірності для кожного класу.

Кількість параметрів моделі після модифікації залежить переважно від кількості видів птахів (наприклад, 100 або 114 класів), але основна частина обчислювальної складності припадає на трансформерну частину, що залишилась незмінною.

Навчання та оцінка. Для навчання було використано інструментарій HuggingFace Transformers та API Trainer, який дозволяє автоматизувати більшість процесів: від логування до збереження чекпоінтів.

Було прийнято рішення використати наступні параметри тренування:

- Batch size: 2;
- Epochs: 15;
- Mixed precision (fp16);

- Learning rate: адаптивний (AdamW);
- Warmup: 10% кроків;
- Early stopping: за відсутності покращення F1.

Оцінка точності здійснювалась за метриками Accuracy та F1-мірою, а також будувалась матриця невідповідностей для виявлення класів, які часто плутаються. У результаті модель досягла стабільної точності на валідаційних даних – з розумним балансом між чутливістю та специфічністю, навіть попри фоновий шум і нерівномірність класів, яке детальніше розглянемо у наступному розділі.

Збереження та деплой. Навчена модель зберігається у форматі, сумісному з HuggingFace .from_pretrained(), що забезпечує швидку інтеграцію на сервері. Також можлива її подальша оптимізація (наприклад, через TorchScript або ONNX), якщо буде потрібно прискорити інференс у майбутньому.

3.5 Результати навчання моделі та їх аналіз

Після завершення етапу тренування адаптованої моделі Wav2Vec2 було проведено оцінку її точності на відкладеній валідаційній вибірці. Загальна ***точність класифікації становила 74,7%***, що є досить чудовим результатом з огляду на складність задачі, кількість класів (понад 50 пташиних видів) та наявність фонових шумів у багатьох аудіофрагментах.

Оцінювання здійснювалось за трьома метриками:

- ***Precision*** – точність передбачення позитивного класу;
- ***Recall*** – повнота (ймовірність знайти всі приклади класу);
- ***F1-міра*** – гармонічне середнє precision і recall.

Середнє по всіх класах (macro average) склало:

- Precision: ***74,2%***;

- Recall: ***72,1%***;
- F1-score: ***70,3%***.

Ці значення свідчать про помірну варіативність у здатності моделі однаково добре працювати з усіма видами. Варто зазначити, що weighted average, тобто з урахуванням кількості прикладів на клас, демонструє дещо кращі результати – близько ***75,5% precision*** і ***72,5% F1-score***, що вказує на впевненість у найбільш представлених видах.

Матриця несумісності. Графік, який зображений на рисунку 3.2, показав, що для більшості видів передбачення співпадає з реальним класом – головна діагональ матриці є яскраво вираженою. Проте існують локальні області, де спостерігається значна кількість помилкових класифікацій.

Наприклад, модель нерідко плутає:

- Bearded Guan із Andean Guan
- Brushland Tinamou з іншими представниками родини Tinamou
- White-throated Tinamou демонструє низький recall (13.3%), що може свідчити про нестачу або низьку якість прикладів

Узагальнено, спостерігається закономірність: класи з меншою кількістю зразків або з менш вираженим вокальним патерном, особливо тихі або короткі сигнали, дають гірші результати, які зображені на рисунку 3.3.

Найбільш «впізнаваними» виявилися:

- Spotted Nothura – $F1 \approx 97.8\%$;
- Highland Tinamou – $F1 \approx 98\%$;
- Cinereous Tinamou, Grey-headed Chachalaca – $F1$ понад 93%.

Натомість, найгіршими є:

- Variegated Tinamou має $F1 = 0.0$;
- Blue-throated Piping Guan – $F1 \approx 23\%$;
- Bearded Guan – $F1 \approx 17\%$, незважаючи на ідеальний precision (1.0), що свідчить про надмірну обережність моделі.

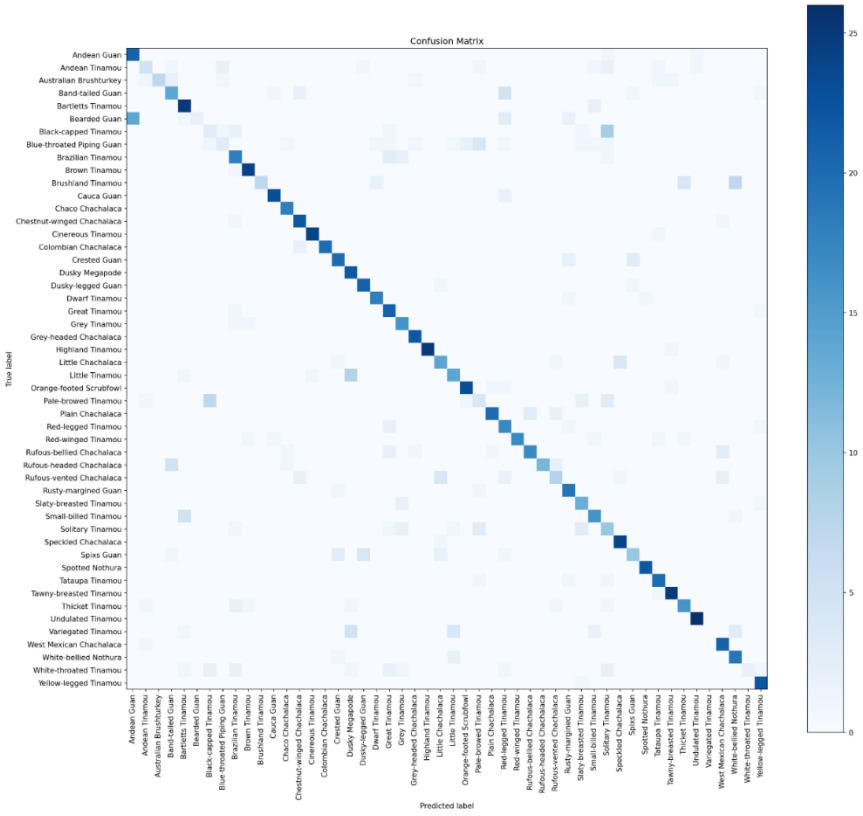

Рисунок 3.2 – Матриця невідповідністей

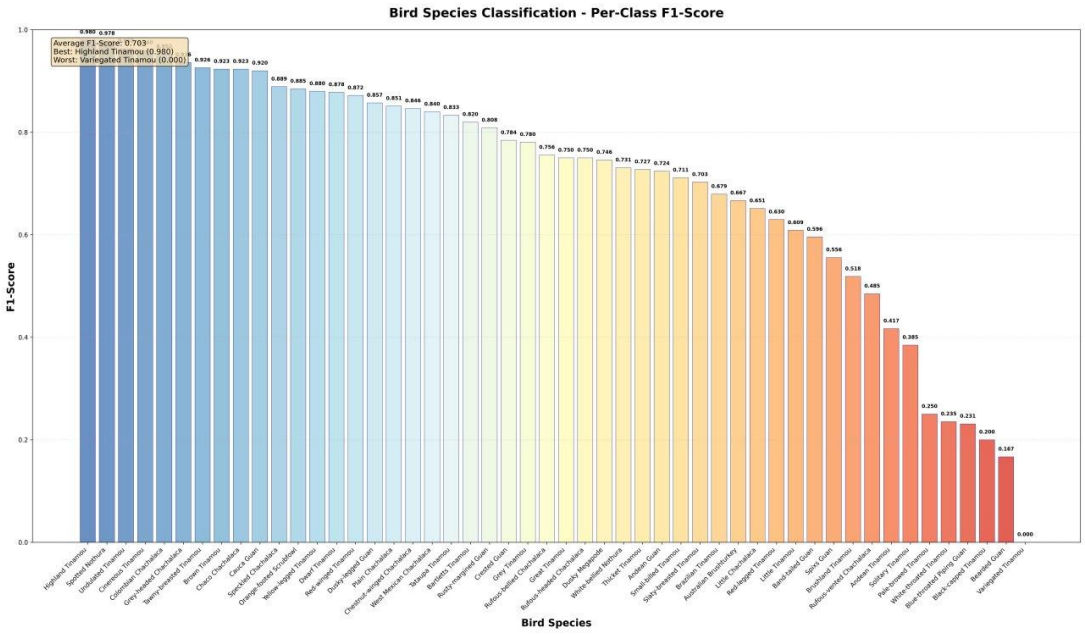

Рисунок 3.3 – Результат моделі по $F1$

Це вказує, що деякі види мають більш характерні вокалізації, які модель вчиться розпізнавати з високою точністю.

Матриця плутанини (рис. 3.2) наочно ілюструє співвідношення вірних і помилкових передбачень. Яскравість головної діагоналі – позитивний індикатор. Водночас, поява щільних плям поза нею – сигнал для подальшого аналізу помилок. Особливо у випадках, де два види часто взаємно плутаються – це може бути не лише проблемою моделі, але й наслідком схожості самих аудіосигналів.

3.6 Розробка серверної частини на Django

Серверна частина системи виконує роль зв’язуючого ланцюга між моделлю глибокого навчання та кінцевими клієнтами – мобільним застосунком і вебінтерфейсом. Її основне завдання – отримати аудіофайл, передати його в обробку, згенерувати прогноз і повернути результат у зручному форматі. З технічного боку, реалізація такого функціоналу вимагає мінімуму графічного інтерфейсу, але максимуму стабільності, масштабованості й безпеки.

Для реалізації серверної частини було обрано Django – перевірений вебфреймворк на Python, який поєднує простоту, потужність і багатий екосистемний набір інструментів. Зокрема, для створення REST API використано бібліотеку Django REST Framework (DRF), яка забезпечує зручну декларативну побудову ендпоінтів, обробку запитів, валідацію, серіалізацію та повернення відповідей у форматі JSON.

Основна логіка серверу. Ключовий сценарій роботи сервера виглядає так:

1. Користувач надсилає POST-запит з аудіофайлом (multipart/form-data);

2. Сервер тимчасово зберігає файл у файлову систему;
3. Аудіофайл передається на обробку: виконується нормалізація, ресемплінг (за потреби), конвертація до тензору;
4. Модель, завантажена з диску (torch.load або .from_pretrained()), виконує інференс (передбачення);
5. Результати (ймовірності для кожного класу) сортуються, форматуються й повертаються як відповідь клієнту.

Усі ці дії обгорнуті в окремий APIView (у DRF), який виконує мінімальне логування та обробку типових помилок (відсутність файлу, неправильний формат, технічна помилка моделі). У разі неуспіху сервер повертає HTTP 400 або 500 з описом проблеми.

Інтеграція з моделлю. Серверна частина реалізована у вигляді REST-сервісу на основі фреймворку Django у поєднанні з Django REST Framework (DRF). Вона відповідає за приймання аудіофайлів, виклик інференсу моделі, обробку результатів класифікації та повернення відповідей у форматі JSON. Реалізація організована модульно, з чітким поділом відповідальностей.

Основні компоненти серверу:

- ***сервіс класифікації*** (BirdClassificationService) – окремий клас для інкапсуляції логіки завантаження моделі, обробки аудіо та генерації передбачень;
- ***API-ендпоінти***:
  - /classify/ – приймає аудіофайл і повертає топ-5 передбачених птахів;
  - /species/ – повертає список усіх підтримуваних видів птахів;
  - /health/ – індикатор працездатності API.

Завантаження моделі. Модель Wav2Vec2 та відповідний feature_extractor завантажуються при ініціалізації сервісу. Якщо локальна копія моделі не знайдена, використовується попередньо натренована версія з HuggingFace Hub. Конфігурація (id2label) зберігається для мапінгу номерів класів у назви птахів. Модель завантажується один раз при запуску серверу й

переводиться в режим оцінювання (eval()), що дозволяє зменшити споживання ресурсів та уникнути непотрібних градієнтних обчислень.

Обробка запиту. Коли на ендпоінт «/classify/» надходить POST-запит з аудіофайлом:

1. Аудіофайл перевіряється за розширенням, розміром і наявністю ключа audio.
2. Файл тимчасово зберігається в локальну файлову систему.
3. Модуль torchaudio використовується для завантаження, ресемплінгу до 16 кГц, зведення до моно та обрізання або паддінгу до 10 секунд.
4. Аудіо конвертується у NumPy-масив і передається у feature_extractor.
5. Генерується logits, обчислюється softmax, і повертаються топ-5 класів з найбільшими ймовірностями.
6. Файл автоматично видаляється після обробки.

Формат відповіді. Повертається JSON-об’єкт із деталями файлу, передбаченнями та найвірогіднішим результатом (рис. 3.4).

Захист і обробка помилок. API містить базові перевірки :

- валідація розширення файлу (підтримуються .mp3, .wav, .m4a, .flac, .ogg);
- перевірка розміру файлу (максимум 50 МБ);
- обробка нештатних ситуацій: невдале завантаження, помилки моделі, внутрішні виключення – усе логуються та супроводжуються відповідями з HTTP-кодами 400/500.

Ця реалізація дозволяє клієнтам швидко та без авторизації взаємодіяти з інтелектуальною системою класифікації пташиних голосів. У наступному підрозділі розглядатиметься мобільний клієнт, що реалізує подібну логіку з боку користувача.

```
{
  "success": true,
  "file_info": {
    "filename": "birdsong.mp3",
    "size_bytes": 102400,
    "size_mb": 0.1
  },
  "predictions": [
    {
      "bird_species": "Dusky-legged Guan",
      "confidence": 0.7412,
      "class_id": 15
    },
    ...
  ],
  "top_prediction": {
    "bird_species": "Dusky-legged Guan",
    "confidence": 0.7412,
    "class_id": 15
  },
  "total_predictions": 5
}
```


Рисунок 3.4 – Приклад відповіді серверу

3.7 Розробка мобільного застосунку на React Native

Мобільний застосунок є ключовою точкою взаємодії користувача з системою класифікації пташиних звуків. Його мета – забезпечити максимально простий і зручний доступ до функціоналу, не перевантажуючи користувача технічними деталями. Застосунок дає змогу обрати аудіофайл, відправити його на сервер і отримати результат у кілька торкань екрана.

Технологічна основа. Для реалізації було обрано React Native з використанням Expo – середовища, що значно спрощує розробку та розгортання кросплатформних мобільних застосунків (Android / iOS). Переваги такого підходу очевидні:

- мінімізація нативного коду,
- швидке прототипування,
- підтримка модулів для взаємодії з файлами та мережею.

Для HTTP-запитів використовується бібліотека axios, а для вибору файлів – стандартні засоби Expo (DocumentPicker, expo-file-system).

Функціонал і UX. Застосунок містить лише базовий, але повністю функціональний інтерфейс. На головному екрані розміщено такі елементи (рис. 3.5):

- **API Connection Status** – перевірка працездатності сервера (/health), відображає статус, кількість підтримуваних видів, версію моделі, IP-адресу сервера;
- **Select Audio File** – кнопка для вибору аудіофайлу зі сховища телефону; підтримуються формати MP3, WAV, M4A, FLAC, OGG розміром до 50 МБ;
- **Classify Bird Sound** – основна кнопка для запуску запиту. Після надсилання файл передається через API, отримується відповідь і відображається топ-класифікація;
- **View Supported Species** – окремий запит до «/species», що повертає перелік усіх птахів, які може розпізнавати система;
- **Check API Health** – повторна перевірка працездатності.

Кольори кнопок підібрано з урахуванням асоціацій користувача: червона – для запуску, жовта – перегляд, зелена – статус. У нижній частині передбачено підказки для розробника: як оновити адресу API (API_BASE_URL), як дізнатися IP сервера тощо.

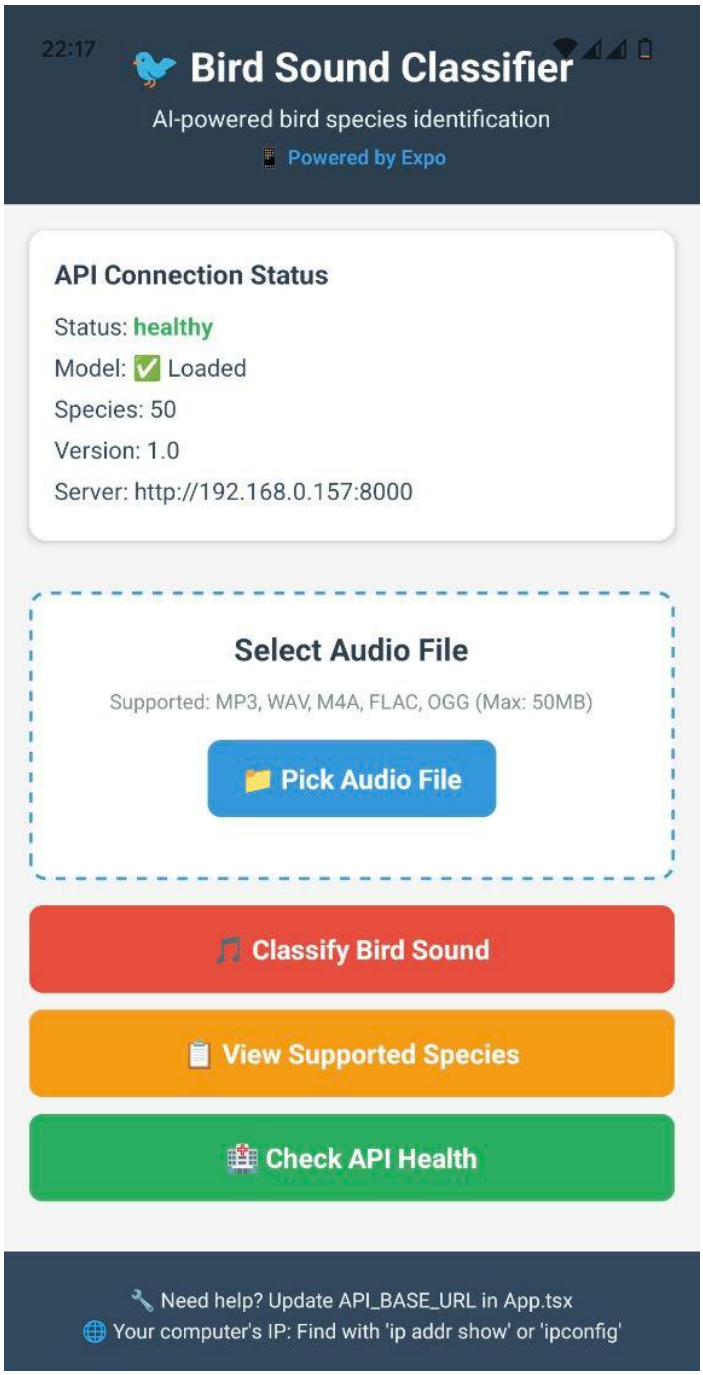

Рисунок 3.5 – Інтерфейс застосунку для телефонів

Технічні аспекти. Всі запити до API реалізовано через асинхронні функції з обробкою помилок. У разі втрати з’єднання або помилкової відповіді користувач отримує зрозуміле повідомлення. Сама класифікація відбувається повністю на сервері – застосунок не виконує жодної локальної

обробки аудіо. Завантаження файлів оптимізоване під великі розміри – запити відправляються у форматі multipart/form-data.

Висновки. Незважаючи на простоту інтерфейсу, мобільний застосунок реалізує повний цикл, зображений на рисунку 3.7.

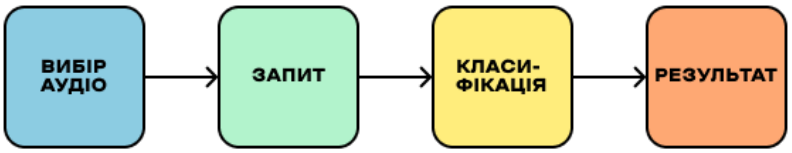

Рисунок 3.7 – цикл роботи застосунку

Це дає змогу користувачу без спеціальних знань перевірити, якого саме птаха він щойно почув. У поєднанні з веб- і серверною частинами, мобільна компонента формує повноцінну й доступну екосистему інтелектуального біоакустичного моніторингу.

3.8 Вебінтерфейс для класифікації аудіо

Паралельно з мобільним застосунком у системі реалізовано вебінтерфейс, що повторює основний функціонал – класифікацію пташиних голосів за аудіофайлом. Його призначення – надати користувачам (а також розробникам і дослідникам) альтернативний, десктопний доступ до API. Завдяки простоті реалізації та повторному використанню компонентів серверу, вебінтерфейс, зображений на рисунках 3.9 та 3.8, став ефективним засобом тестування й демонстрації системи.

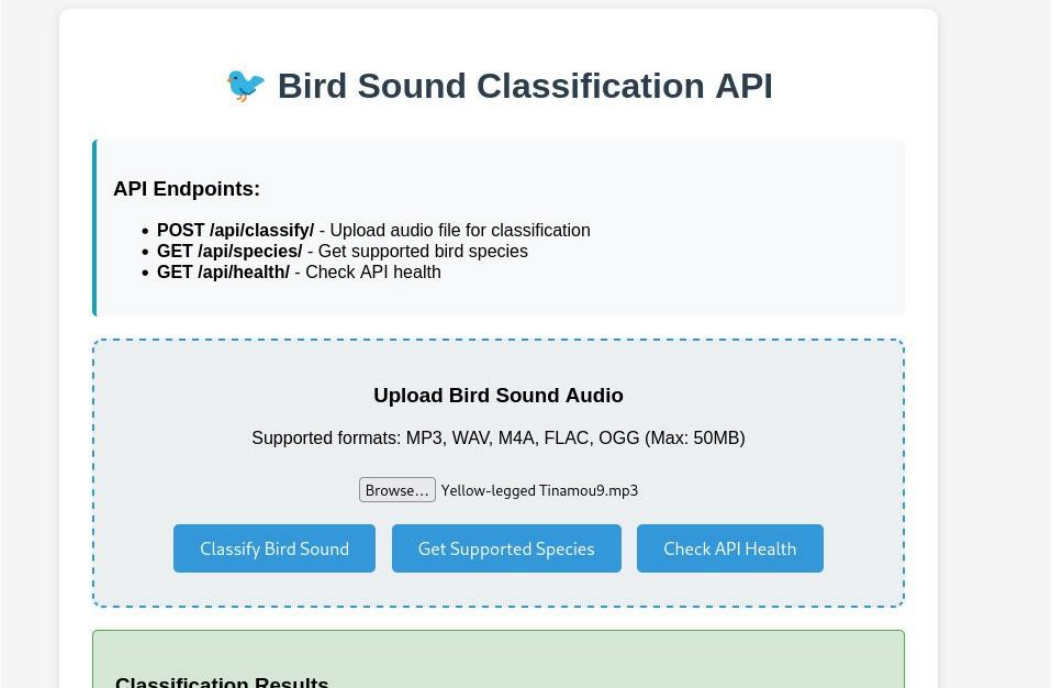

Рисунок 3.8 – Інтерфейс вебсторінки

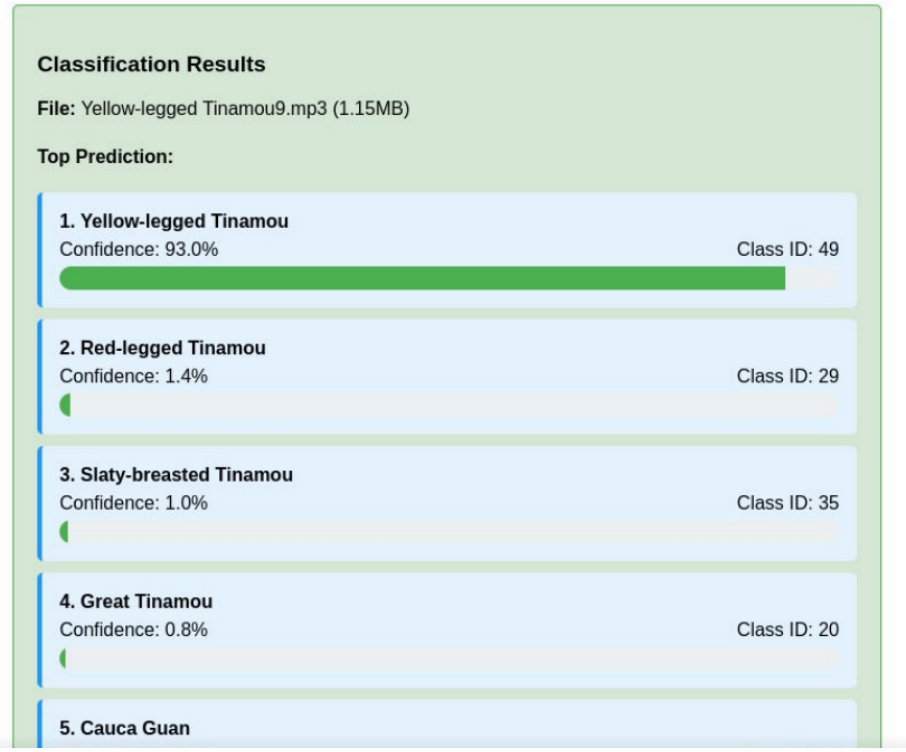

Рисунок 3.9 – Приклади результатів

Основні можливості. Інтерфейс дає змогу:

- обрати локальний аудіофайл (MP3, WAV, M4A, FLAC, OGG, до 50 МБ);
- надіслати файл на сервер і отримати топ-результати класифікації;
- переглянути список підтримуваних пташиних видів;
- перевірити працездатність API (/health).

З погляду користувача, вебінтерфейс повторює мобільний застосунок як у структурі, так і у стилістиці (кольори, кнопки, підказки). Форма взаємодії організована максимально лаконічно: жодних реєстрацій, ніяких складних форм – лише необхідне.

Технічна реалізація. Інтерфейс побудовано на основі Django (можливо, з вбудованими HTML-шаблонами або легким JS). Основна логіка – обробка події завантаження файлу та відправка POST-запиту через fetch() або axios. Серверна частина сприймає ці запити ідентично до мобільного клієнта, що забезпечує єдність API і зменшує технічний борг.

У відповідь користувач отримує форматований JSON, який перетворюється у таблицю або список із ймовірностями та назвами птахів. У разі помилки система повертає зрозуміле повідомлення (наприклад, про невірний формат або перевищений розмір файлу).

Практичне призначення. Хоч вебінтерфейс і не є основним способом взаємодії, він виконує важливу роль:

- як інструмент демонстрації – для освітніх і наукових цілей;
- як тестовий клієнт для перевірки роботи серверу без емуляції мобільного середовища;
- як основа для розширення – у майбутньому інтерфейс може бути перетворений на повноцінний публічний портал.

У результаті вебінтерфейс логічно доповнює систему, надаючи доступ до функціоналу там, де мобільний клієнт недоступний або недоречний. З технічного боку, він використовує той самий API, що й мобільний застосунок, тож не створює додаткового навантаження на інфраструктуру.

3.9 Тестування компонентів і взаємодії

Завершальна частина розробки інтелектуальної системи класифікації пташиних голосів полягала у ретельному тестуванні її складових – як окремих, так і в контексті всієї взаємодії між собою. Основна мета цього етапу – переконатися, що всі компоненти системи не лише функціонують відповідно до свого призначення, а й узгоджено працюють у межах єдиного технічного процесу: від моменту вибору аудіофайлу до повернення класифікаційного результату на екран користувача.

Початкове тестування торкалось безпосередньо моделі. Навіть після завершення тренування і позитивних валідаційних результатів, модель було додатково перевірено на низці аудіофрагментів, які не входили до навчального набору. Особливий інтерес становили ситуації з тишею, фонованими шумами або неповними зразками співу птахів. Виявилось, що модель досить впевнено ідентифікує "порожні" фрагменти – тобто фрагменти без пташиних звуків – як такі, що не відповідають жодному з класів, повертаючи низькі ймовірності для всіх варіантів. Це вказує на відсутність "фальшивих позитивів", які особливо критичні у біоакустичних застосуваннях. Водночас, при введенні навмисно схожих голосів – наприклад, представників одного роду або родини – модель іноді демонструє змішані результати, розподіляючи ймовірність між кількома близькими видами. Це цілком очікувана поведінка: деякі види птахів дійсно мають схожі вокальні патерни, і їх розпізнавання вимагає навіть від фахівців слухового порівняння.

Серверна частина також була піддана випробуванням. Зокрема, перевірялось, як вона поводиться у разі некоректного запиту, відсутності файлу, неправильного формату або перевищення обсягу. У всіх випадках система реагувала передбачувано – повертала HTTP-помилки з поясненням, не завершувалась аварійно і не потребувала перезапуску. Важливою

частиною тестування було переконатися, що модель не перевантажується з кожним запитом, а зберігається в пам’яті між викликами, що й було реалізовано через єдиний екземпляр сервісу класифікації, ініціалізований під час запуску додатку. Це дозволило зменшити час відповіді до декількох секунд, навіть при використанні лише CPU.

Мобільний застосунок, як і вебінтерфейс, проходив функціональне тестування у різних сценаріях використання. Було перевірено поведінку при виборі коректного та некоректного файлу, відсутності мережевого з’єднання, повторному надсиланні одного й того ж запису. Виявлено, що клієнтська частина грамотно обробляє відповіді серверу та відображає необхідну інформацію у доступній формі. Окремо перевірялись і крайові ситуації – наприклад, обробка файлу максимально дозволеного розміру (50 МБ), або навмисна відмова сервера – і в цих випадках інтерфейс поводився стабільно: користувачу виводиться опис проблеми, при цьому застосунок залишається повністю функціональним.

Особливе значення мало інтеграційне тестування – тобто перевірка того, як працює весь ланцюг: від вибору або запису аудіо до його класифікації та відображення результату. Такий наскрізний підхід дозволив виявити невеликі розсинхронізації між клієнтами й сервером на рівні обробки помилок, які було виправлено вже у процесі тестування. У результаті користувач отримує цілісний досвід: усі дії – вибір файлу, надсилання, очікування відповіді, вивід – відбуваються плавно, без затримок або неочікуваних переривань.

Таким чином, тестування показало, що створена система готова до використання в реальних умовах. Вона здатна працювати з різними джерелами звуку, витримує помірне навантаження, стійка до помилок, і головне – правильно реалізує свою цільову функцію: класифікацію пташиних голосів у зручному для користувача форматі.

Висновки до розділу 3

У цьому розділі було детально розглянуто процес побудови повноцінної інтелектуальної системи для класифікації пташиних голосів, що об’єднує сучасну модель глибокого навчання, серверну обробку, а також два незалежних клієнтських інтерфейси – мобільний та вебзастосунок. Починаючи від постановки задачі й аналізу вимог, через вибір інструментів, обробку даних, навчання та оцінку моделі, і аж до практичного впровадження інтерфейсів – було пройдено повний цикл проєктування і розробки функціонального програмного рішення.

Особливістю реалізованої системи є поєднання декількох рівнів складності: з одного боку, ядром виступає потужна модель Wav2Vec2, здатна оперувати з сирими аудіоданими; з іншого – інтерфейсні рішення зведено до максимально простих, що дозволяє використовувати систему навіть технічно непідготовленим користувачам. Вдалий вибір технологій – таких як PyTorch, HuggingFace Transformers, Django, React Native – забезпечив гнучкість, ефективність і швидкість реалізації.

Аналіз результатів тренування показав, що модель досягає високої точності на більшості класів, зберігаючи стійкість до шуму та змішаних сигналів. Інтеграція моделі в серверну архітектуру виконана з урахуванням продуктивності та безпеки, а обидва клієнтські додатки продемонстрували стабільну роботу під час тестування.

Загалом, виконана роботи не лише виконує поставлену задачу з технічної точки зору, а й закладає фундамент для подальшого розвитку: розширення кількості підтримуваних видів, адаптації під інші типи біоакустичних сигналів, або навіть створення освітньої платформи для ознайомлення громадськості з пташиним розмаїттям через звук.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі розглядається техніко-економічна доцільність створення програмного продукту, що реалізує інтелектуальне розпізнавання пташиних голосів за допомогою глибоких нейронних мереж. Система включає серверну модель класифікації на основі архітектури Wav2Vec2, API-сервер, вебінтерфейс та мобільний клієнт. Вибір конкретних компонентів і технологій базувався не лише на функціональних можливостях, а й на оцінці вартості їх реалізації, продуктивності та зручності використання.

Для обґрунтування вибору технічних рішень у розробці системи застосовується метод функціонально-вартісного аналізу (ФВА). Цей метод дозволяє співвіднести витрати на створення окремих модулів із їх значущістю для кінцевого користувача, виявити надлишкові витрати, а також оцінити якість реалізації ключових функцій за сукупністю технічних параметрів. Зокрема, у межах аналізу буде розглянуто два альтернативні варіанти реалізації інтерфейсної частини продукту – мобільний застосунок на React Native та вебінтерфейс – із подальшою оцінкою їхньої ефективності та вартості.

Застосування ФВА дозволяє зробити вибір на користь тієї конфігурації системи, яка забезпечує оптимальне поєднання технічної якості, функціональної повноти та економічної ефективності розробки

4.1 Постановка задачі

У межах цієї дипломної роботи використовується метод функціонально-вартісного аналізу (ФВА) для техніко-економічного

обґрунтування створення інтелектуальної системи класифікації пташиних голосів. Розроблений програмний продукт має на меті забезпечити автоматизовану обробку аудіосигналів, визначення ймовірного виду птаха на основі голосу, а також надати доступ до цієї функціональності через мобільний додаток і вебінтерфейс. Усі компоненти системи мають взаємодіяти через централізований API-сервер.

Функціонально-вартісний аналіз дозволяє зіставити витрати на реалізацію ключових функцій продукту з їх значущістю для кінцевого користувача, а також оцінити доцільність обраної архітектури у порівнянні з альтернативними варіантами. В якості об'єкта аналізу розглядається повноцінна програмна система, що включає модель глибокого навчання (Wav2Vec2), інтерфейс користувача, серверну частину для обробки запитів та хмарну інфраструктуру для зберігання й масштабування.

До ключових технічних вимог, які висуваються до системи, належать такі:

- стабільна робота на більшості сучасних мобільних пристроїв без потреби у спеціалізованому апаратному забезпеченні;
- здатність обробляти аудіосигнали у стандартних форматах з подальшою інференцією моделі;
- використання відкритого API для забезпечення доступу до класифікації сторонніми розробниками;
- зручний користувацький інтерфейс у мобільному додатку й вебверсії;
- швидка відповідь системи, придатна для використання в реальному часі або з мінімальною затримкою;
- економічна ефективність – доцільність витрат на розробку, розгортання і подальше обслуговування.

Аналіз функціональності та вартості реалізації зазначених модулів дозволить обґрунтувати технічне рішення, яке забезпечить найкраще співвідношення якості, швидкодії та витрат.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0– створення інтелектуальної системи розпізнавання пташиних видів за аудіосигналами з реалізацією у вигляді мобільного застосунку та вебінтерфейсу. Система повинна забезпечувати повний цикл обробки звуку: від завантаження файлу користувачем – до класифікації звуку нейронною мережею на основі Wav2Vec2.

На основі головної функції F_0 виділено три ключові підфункції:

- F_1 – підготовка та обробка аудіосигналу;
- F_2 – класифікація аудіо за допомогою моделі глибокого навчання;
- F_3 – передавання результатів класифікації користувачу через мобільний застосунок або вебінтерфейс.

Функція F_1:

- а) ресемплінг, нормалізація та приведення аудіо до фіксованої довжини (використання torchaudio);
- б) автоматичне визначення формату та перетворення сигналу у формат, сумісний із моделлю;
- в) ручна перевірка якості файлу перед обробкою.

Функція F_2:

- а) використання готової моделі Wav2Vec2, донавченої на датасеті пташиних голосів;
- б) використання базової версії Wav2Vec2 без донавчання;
- в) самостійна побудова меншої моделі для inference на мобільному пристрої (наприклад, CNN1D або CRNN).

Функція F_3:

- а) вебінтерфейс на Django з API на основі DRF;
- б) мобільний застосунок на React Native з Expo;

- в) відображення топ-N результатів з ймовірностями, можливість повторного прослуховування завантаженого аудіо.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

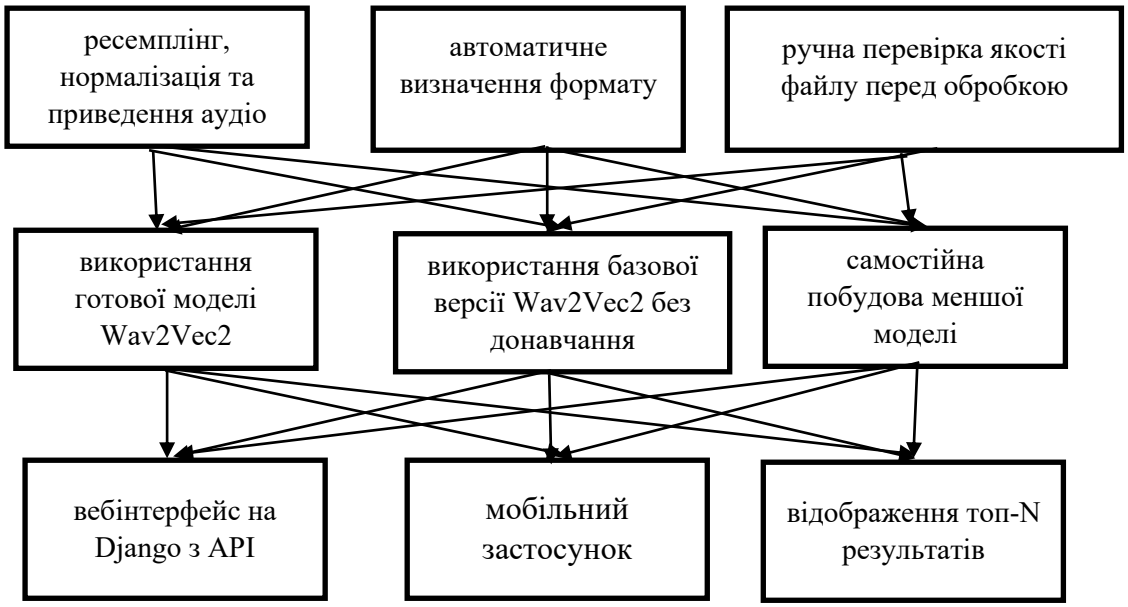

Рисунок 4.1 – Морфологічна карта

На основі аналізу реалізованих та потенційних варіантів робимо наступні висновки.

Функція F_1: найбільш доцільним виявився варіант а – автоматизована обробка аудіо на сервері з використанням бібліотеки torchaudio, оскільки він забезпечує повну інтеграцію з моделлю та зменшує залежність від формату вхідного сигналу. Інші варіанти вимагали б ручного втручання або нестандартних обробок з боку користувача.

Функція F_2: варіант а – використання донавченої моделі Wav2Vec2 – показав найкраще співвідношення між точністю та витратами на навчання. Варіант б не забезпечував достатньої точності, а варіант в потребував би значних ресурсів на повторну розробку.

Таблиця 4.1 – Позитивно-негативна матриця

Функція	*Варіант реалізації*	*Переваги*	*Недоліки*
F_1	а – Автоматична обробка з torchaudio	Надійна обробка форматів, ресемплінг, обрізання/доповнення, гнучкість	Серверне навантаження, необхідність перевірки якості аудіо
F_1	б – Ручна підготовка користувачем	Мінімальні серверні ресурси	Нестабільна якість, залежність від користувача, складність використання
F_2	а – Дообучена модель Wav2Vec2	Висока точність, масштабованість, готова до API	Великий об’єм, потреба у ресурсах на початкове навчання
F_2	б – Базовий Wav2Vec2 без навчання	Швидке використання без додаткової підготовки	Низька точність, неадаптована до пташиних голосів
F_2	в – Альтернативна проста модель (наприклад, CNN1D)	Менше споживання ресурсів, потенційно швидша робота	Менша точність, потребує багато часу на розробку та валідацію
F_3	а – Вебінтерфейс (Django + DRF)	Зручність адміністрування, доступ через браузер	Не зовсім зручний для мобільного користувача
F_3	б – Мобільний застосунок (React Native + Expo)	Зручний для кінцевого користувача, адаптований під мобільні платформи	Необхідність супроводу під різні ОС, потреба оновлень
F_3	в – Вивід лише у вигляді JSON без UI	Найпростіший варіант інтеграції	Не призначений для прямого використання користувачем

Функція F_3: доцільно використано комбінацію а + б – вебінтерфейс для адміністративного використання і API-запитів, мобільний застосунок як основний канал взаємодії з користувачем. Відображення результатів реалізовано через компонент, що подає список найімовірніших видів птахів з оцінками довіри.

Для подальшого техніко-економічного аналізу будуть розглянуті два найбільш доцільні варіанти реалізації:

- $F_{1a} - F_{2a} - F_{3b}$;
- $F_{1a} - F_{2a} - F_{3a} + b$ (комбіновано).

Наступні підрозділи будуть присвячені порівнянню цих варіантів за технічними параметрами та витратами на розробку.

4.3 Обґрунтування системи параметрів програмного продукту

З метою обґрунтування вибору оптимального варіанту реалізації програмного продукту – системи для розпізнавання пташиних голосів на основі глибокого навчання – було сформовано систему технічних параметрів. Вона враховує вимоги до ефективності моделі, її продуктивності на мобільних пристроях і серверах, а також ресурсоємності при розгортанні в реальних умовах.

Обрана система складається з чотирьох основних параметрів, які оцінюють ключові властивості розробки з позиції технічної доцільності:

- X_1 – орієнтовний обсяг коду (у рядках), необхідний для реалізації вибраної комбінації функцій (включаючи модель, сервер, API та інтерфейс);
- X_2 – обсяг оперативної пам’яті (RAM), що споживається під час інференсу аудіофайлу;
- X_3 – середній час виконання інференсу (включаючи обробку аудіо, векторизацію та класифікацію);
- X_4 – рівень автоматизації та зручності взаємодії з кінцевим користувачем (інтерфейс, зворотний зв’язок, формат результатів).

Таблиця 4.2 – Основні параметри програмного продукту

Назва параметра	*Позначення*	*Одиниця виміру*	*Гірші*	*Середні*	*Кращі*
Орієнтовний обсяг коду	X21	кількість рядків	4000	3000	2000
Обсяг використаної оперативної пам’яті	X2	МБ	1600	1024	512
Середній час інференсу	X3	секунди	8.0	4.0	1.5
Рівень інтерактивності та автоматизації UI	X4	балів (експертно)	1	2	3

Ці параметри дозволяють комплексно охарактеризувати технічний рівень різних варіантів реалізації системи та служать основою для подальшого експертного та економічного аналізу.

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

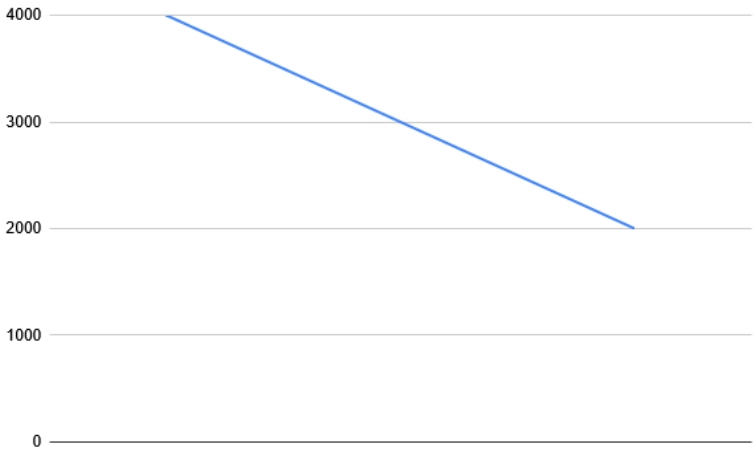

Рисунок 4.2 – *X*1, Орієнтовний обсяг коду

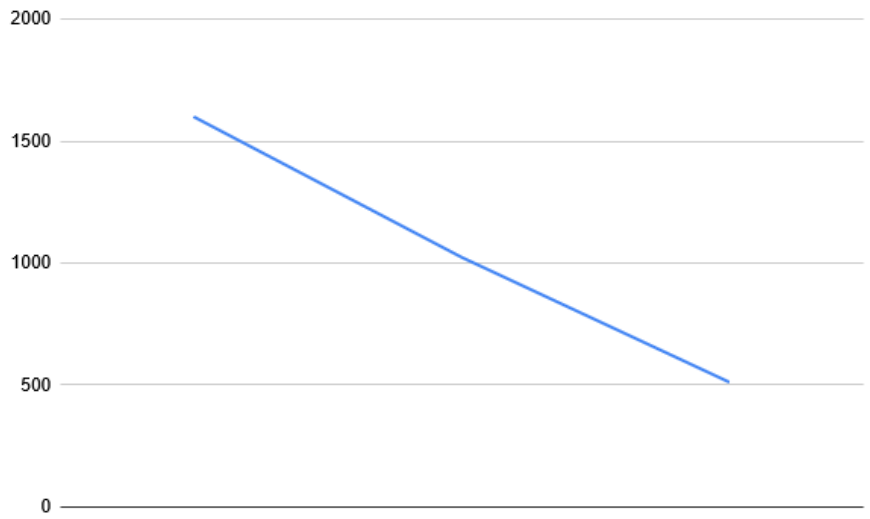

Рисунок 4.3 – $X2$, Обсяг використаної оперативної пам’яті

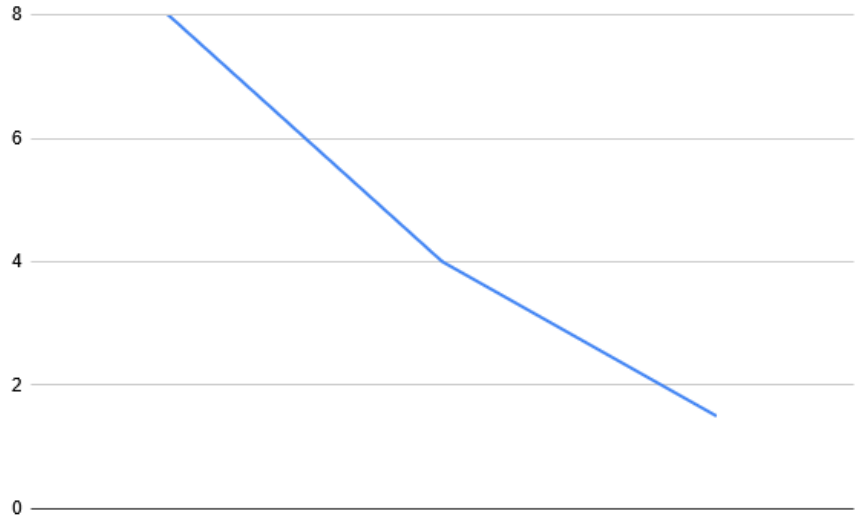

Рисунок 4.4 – $X3$ Середній час інференсу

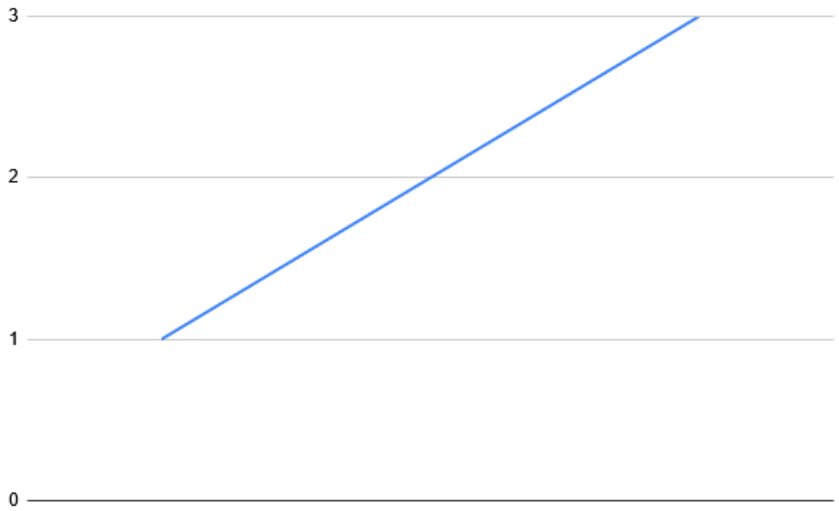

Рисунок 4.5 – *X*4 Рівень інтерактивності та автоматизації UI

4.4 Аналіз експертного оцінювання параметрів

Після визначення системи технічних параметрів для оцінювання варіантів реалізації програмного продукту – системи класифікації голосів птахів за допомогою глибокого навчання – було проведено експертне оцінювання їхньої значущості. Оцінювання дозволяє визначити вагу кожного параметра в загальній оцінці якості рішення та здійснити обґрунтований вибір оптимального варіанту реалізації.

До експертної групи було залучено 7 фахівців у галузі штучного інтелекту, розробки мобільних застосунків і вебсервісів. Експерти оцінювали чотири ключові параметри, визначені в попередньому пункті.

Метод оцінювання – метод попарного порівняння з подальшим розрахунком коефіцієнта узгодженості для перевірки достовірності результаті.

Таблиця 4.3 – Результати ранжування параметрів

Позн.	*Назва*	*Од. виміру*	*E1*	*E2*	*E3*	*E4*	*E5*	*E6*	*E7*	R_i	Δ_i	Δ_i^2
X_1	Обсяг коду	Рядків	4	4	3	2	3	2	2	20	2.5	6.25
X_2	Обсяг оперативної пам'яті	МБ	2	1	1	1	2	2	1	10	-7.5	56.25
X_3	Час інференсу	секунди	3	2	4	4	4	3	4	24	6.5	42.25
X_4	Зручність і автоматизація взаємодії	бали	1	3	2	3	1	3	3	16	-1.5	2.25

З метою перевірки достовірності отриманих експертних оцінок проведено аналіз ступеня узгодженості думок фахівців. Для цього були обчислені параметри.

1. Сума рангів для кожного параметра і загальна сума, формули (4.1, 4.2).

$$R_i = \sum_{j=1}^{N} r_{ij}, \tag{4.1}$$

$$R_{\text{загальна}} = \frac{N \cdot n(n+1)}{2} = \frac{7 \cdot 4(4+1)}{2} = 70, \tag{4.2}$$

де N=7 – кількість експертів;

n=4 – кількість параметрів.

2. Середнє значення суми рангів, формула (4.3).

$$T = \frac{R_{\text{загальна}}}{n} = \frac{70}{4} = 17.5. \tag{4.3}$$

3. Відхилення кожного параметра від середньої суми рангів, формула (4.4).

$$\Delta_i = R_i - T, \tag{4.4}$$

сума відхилень дорівнює нулю, що підтверджує правильність розрахунків.

4. Загальна сума квадратів відхилень, формула (4.5).

$$S = \sum_{i=1}^{n} \Delta_i^2 = 107. \quad (4.5)$$

5. Коефіцієнт узгодженості Веджа, формула (4.6).

$$W = \frac{12 \cdot S}{N^2(n^3 - n)} = \frac{12 \cdot 107}{7^2(64-4)} = \frac{1284}{2408} \approx 0{,}533. \quad (4.6)$$

Оскільки значення коефіцієнта узгодженості W=0.533 наближається до нормативного рівня $W_k = 0.56$, результати експертного ранжування можна вважати достатньо узгодженими та придатними до подальшого використання.

На основі ранжування сформована таблиця 4.4, що містить попарне порівняння параметрів.

Таблиця 4.4 - Результати ранжування параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X_1 і X_2	>	>	>	>	>	=	>	<	0,5
X_1 і X_3	>	>	<	<	<	<	<	<	0,5
X_1 і X_4	>	>	>	>	>	<	<	>	1,5
X_2 і X_3	<	<	<	<	<	<	<	<	0,5
X_2 і X_4	>	<	<	<	>	<	>	<	0,5
X_3 і X_4	>	<	>	>	>	=	>	>	1,5

Для визначення ступеня переваги одного параметра над іншим використовується числовий показник a_{ij}, який визначається як представлено у формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j, \\ 1.0 \text{ при } X_i = X_j, \\ 0.5 \text{ при } X_i < X_j. \end{cases}$$

На основі експертних оцінок, сформованих у таблиці попарного порівняння, будуємо матрицю переваг A=‖a_{ij}‖, де кожен елемент відображає числову перевагу одного параметра над іншим.

Далі розраховуються початкові вагомості параметрів $K_{вi}$ за формулою (4.8).

$$K_{\text{B}i} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \tag{4.8}$$

де b_i – це сума значень у відповідному рядку матриці, формула (4.9).

$$b_i = \sum_{j=1}^{n} a_{ij}. \tag{4.9}$$

Відносні оцінки уточнюються кілька разів до досягнення стабільного результату, коли зміни між послідовними ітераціями менші за 2%. На кожному наступному етапі обчислення вагомостей відбувається за формулою (4.10).

$$K_{\text{B}i} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{4.10}$$

де b_i'– нові значення, що обчислюються за формулою (4.11):

$$b_i' = \sum_{j=1}^{n} a_{ij} K_{\text{B}j}. \tag{4.11}$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.	
	X1	X2	X3	X4	b_i	K_{bi}	b_i^2	K_{bi}^2
X1	1,0	0,5	0,5	1,5	3,5	0,219	13,25	0,217
X2	1,5	1,0	0,5	0,5	3,5	0,219	13,25	0,217
X3	1,5	1,5	1,0	1,5	5,5	0,344	21,25	0,348
X4	0,5	1,5	0,5	1,0	3,5	0,219	13,25	0,217
Всього:					16	1	61	1

4.5 Аналіз рівня якості варіантів реалізації функцій

У цьому підпункті визначається рівень якості кожного варіанта реалізації основних функцій програмного продукту, розробленого для класифікації пташиних голосів на основі глибокого навчання. Метою є оцінка ефективності вибраної архітектури з урахуванням кількох техніко-економічних показників.

Абсолютні значення параметрів X2 (обсяг оперативної пам’яті), X3 (час виконання інференсу) та X4 (зручність взаємодії з користувачем) повністю відповідають технічним вимогам до роботи мобільного та вебзастосунку. Значення X1 (обсяг коду) знаходиться в межах прийнятної складності підтримки і не є обмежувальним чинником.

Для обчислення коефіцієнта технічного рівня кожного з варіантів використовується формула:

$$\mathrm{K_K}(j) = \sum_{n-1}^{n} K_{bi,j} B_{i,j},$$

де n – кількість параметрів оцінювання;

K_{bi}– коефіцієнт вагомості і-го параметра;

$B_{i,j}$– оцінка і-го параметра для j-го варіанту реалізації в балах.

На основі цього розрахунку буде визначено якісну перевагу кожного варіанту реалізації та обрано оптимальний з них для впровадження.

За даними з таблиці 4.6 визначаємо рівень якості кожного з варіантів, формула (4.13).

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \cdots + K_{\text{ТУ}}[F_{zk}].$$

$$K_{K1} = 0{,}65 + 0{,}54 + 1{,}04 = 2{,}23.$$

$$K_{K2} = 0{,}65 + 0{,}54 + 0{,}54 = 1{,}73.$$

Як видно з розрахунків, кращим є варіант 1, для якого коефіцієнт технічного рівня має найбільше значення.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	*Варіант реалізації функції*	*Параметри*	*Абсолютне значення параметра*	*Бальна оцінка параметра*	*Коефіцієнт вагомості параметра*	*Коефіцієнт рівня якості*
F1	A	X1	965	3	0,217	0,65
F2	A	X2	800 Мб	2,5	0,217	0,54
F2	A	X3	0,7	3	0,348	1,04
F3	A	X4	2,5	2,5	0,217	0,54

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки програмного продукту було виконано розрахунок трудомісткості, а також пов’язаних із цим витрат. Розглядається варіант реалізації інформаційної системи для класифікації пташиних голосів із застосуванням глибокого навчання.

У межах проєкту передбачено два основних завдання.

1. Розробка вбудованого програмного забезпечення (серверна логіка, REST API, модель).
2. Розробка допоміжного програмного забезпечення – веб та мобільного інтерфейсу користувача.

Перше завдання за ступенем новизни належить до групи А, а друге – до групи Б. За складністю алгоритмів – перше до групи 3, друге – до групи 1.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 працює з інформацією у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість визначається за формулою:

$$T_О = T_Р \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ,М},$$

де $T_Р$ – нормативна трудомісткість розробки ПП (люд.-днів);

$K_П$ – поправочний коефіцієнт, що враховує тип інформації;

$K_{СК}$ – коефіцієнт складності вхідної інформації;

$K_М$ – коефіцієнт, що враховує рівень мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних програмних модулів;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для завдання 1:

- $T_р = 45$ люд.-днів;
- $K_р = 1.4$ (довідкова інформація);
- $K_{sk} = 1$ (середня складність вхідної інформації);
- $K_m = 1$ (використано Python, JavaScript);
- $K_{st} = 0.8$ (використано готові модулі PyTorch, Django);
- $K_{st}.м = 1$ (стандартна математика, без складних моделей).

Розрахунок:

$$T1=45 \cdot 1.4 \cdot 1 \cdot 1 \cdot 0.8 \cdot 1=50.4 \text{ людино-днів}$$

Для завдання 2:

- T_p = 28 людино-днів;
- K_p = 1.1;
- K_{sk} = 1;
- K_m = 1;
- K_{st} = 0.9;
- K_{st}.м = 1.

Розрахунок:

- T2= 28 · 1.1 · 1 · 1 · 0.9 · 1 = 27.72 людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

T_o = (50.4 + 27.72) · 8 = 625 людино-годин.

Найбільш високу трудомісткість має варіант із додатковими аналітичними функціями, що потребують розширення візуалізації.

У розробці бере участь один програміст із місячним окладом 38000 грн. Визначимо середню погодинну заробітну плату за формулою:

$$C_ч = \frac{M}{T_m \cdot t} \text{ грн},$$

де M – сума місячного окладу;

T_m – кількість робочих днів у місяць;

t – кількість робочих годин на день.

$C_ч$ = 38000 / (21 · 8) = 226,19 грн.

Далі розраховуємо заробітну плату за формулою:

$$C_{ЗП} = C_ч \cdot T_i \cdot К_Д ,$$

де $C_ч$ – погодинна ставка;

T_i – трудомісткість у людино-годинах;

$К_Д$ – коефіцієнт додаткової оплати (приймаємо $К_Д = 1.2$).

$С_{ЗП} = 226,19 \cdot 625 \cdot 1.2 = 169642,5$грн.

Єдиний соціальний внесок становить 22% від $С_{ЗП}$:

$С_{ВІД} = 66967.5 \cdot 0.22 = 37321,35$ грн.

Тепер визначимо витрати на оплату однієї машино-години (СМ). Так як одна ЕОМ обслуговує одного програміста з окладом 38000 грн., з коефіцієнтом зайнятості 0.25, то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 38000 \cdot 0.25 = 114000$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = СГ \cdot (1 + К_З) = 36000 \cdot (1 + 0.2) = 136800$ грн.

Відрахування на соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.22 = 43200 \cdot 0.22 = 30096$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 41500 грн:

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.23 \cdot 0.25 \cdot 41500 = 12759.38 \text{ грн},$$

де $К_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж; $К_А$ – річна норма амортизації; $Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.23 \cdot 41500 \cdot 0.03 = 1531.35 \text{ грн},$$

де $К_Р$ – відсоток витрат на поточні ремонти

Ефективний годинний фонд часу ПК за рік:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t \cdot К_В =$$
$$= (365 – 105 – 21 – 16) \cdot 8 \cdot 0.87 = 1552.08 \text{ годин},$$

де $Д_К$ – кількість днів у році; $Д_В$ – кількість вихідних днів; $Д_С$ – кількість святкових днів; $Д_Р$ – кількість днів планових ремонтів; t – тривалість робочого дня; $К_В$ – коефіцієнт використання машини протягом зміни.

Витрати на оплату електроенергії:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1552.08 \cdot 0.3 \cdot 0.5 \cdot 9.43 = 2196.29 \text{ грн},$$

де $N_С$ – середньо-споживча потужність приладу; $К_З$ – коефіцієнт зайнятості приладу; $Ц_{ЕН}$ – тариф на електроенергію для 2 класу напруги з ПДВ – 9.43 грн/кВт·год.

Накладні витрати:

$С_Н = Ц_{ПР} \cdot 0.67 = 41500 \cdot 0.67 = 27805.00$ грн.

Тоді, річні експлуатаційні витрати:

$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н = 136800 + 30096 + 12759.38 + 1531.35 + 2196.29 + 27805 = 211188.02$ грн.

Собівартість однієї машино-години ЕОМ дорівнює:

$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 96895.02 / 1552.08 = 136.06$ грн/год.

Оскільки в цьому випадку всі роботи, які пов‘язані з розробкою програмного продукту, ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складають:

$С_М = С_{М\text{-}Г} \cdot Т = 62.43 \cdot 625 = 85037.5$грн.

Накладні витрати становлять 67% від заробітної плати:

$С_Н = С_{ЗП} \cdot 0.67 = 66968.75 \cdot 0.67 = 113659$ грн.

Отже, вартість розробки ПП за варіантами становить:

$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н = 66968.75 + 14733.13 + 39018.75 + 44868.06 = 405656.77$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР j} = {К_{Кj}}/{С_{Фj}},$$

$К_{ТЕР} = 20.4 / 405656.77 = 5.02888 \cdot 10\text{-}5.$

На основі проведеного аналізу вартості розробки програмного продукту визначено трудомісткість для двох ключових завдань: розробки вбудованого програмного забезпечення та допоміжної програми для візуалізації. Для першого завдання трудомісткість склала 50.4 людино-днів, з урахуванням використання довідкової інформації та стандартних модулів, що відображено поправочним коефіцієнтом 1.4 та коефіцієнтом використання модулів 0.8. Друге завдання, з трудомісткістю в 27.72 людино-днів, реалізується з алгоритмами першої групи складності та відповідним коефіцієнтом 1.1. Сумарна трудомісткість обох завдань у людино-годинах дорівнює 625, що дає змогу оцінити необхідні ресурси та час для реалізації проекту. Загальна вартість розробки програмного продукту становить 405656.77 гривень, що включає заробітну плату, витрати на оплату машинного часу, соціальні внески та накладні витрати. Значення коефіцієнта техніко-економічного рівня свідчить про доцільність подальшої оптимізації витрат та ефективного розподілу ресурсів для підвищення комерційної цінності проєкту.

Висновки до розділу 4

У цьому розділі було здійснено функціонально-вартісний аналіз інформаційної системи для класифікації пташиних голосів на основі глибокого навчання. Проведено оцінювання ключових параметрів, що впливають на ефективність програмного продукту, включаючи обсяг коду, споживання оперативної пам’яті, швидкодію моделі та рівень зручності для кінцевого користувача.

На основі експертного ранжування визначено вагомості технічних параметрів та обчислено коефіцієнти якості для двох варіантів реалізації, що дозволило встановити перевагу першого з них. Варіант із заморожуванням шарів моделі, балансуванням класів та веб-інтерфейсом із REST API забезпечив найвищий показник ефективності при збереженні високого рівня інтерактивності.

Окрім того, проведено повний економічний аналіз: розраховано трудомісткість виконання завдань, витрати на заробітну плату, експлуатацію обладнання, електроенергію та накладні витрати. Загальна вартість розробки системи склала 405 656,77 грн.

Обчислений коефіцієнт техніко-економічного рівня свідчить про ефективне співвідношення витрат до технічної якості розробленого програмного продукту, що підтверджує доцільність обраної архітектури та підходів до реалізації. У перспективі можлива подальша оптимізація окремих компонентів системи для підвищення продуктивності та зменшення витрат при масштабуванні.

ВИСНОВКИ

У межах переддипломної практики було сформульовано прикладну проблему – автоматичне розпізнавання пташиних видів за аудіосигналами з подальшим впровадженням у мобільний застосунок. Це завдання об’єднує напрямки мобільної розробки, штучного інтелекту та цифрової обробки сигналів, і має як наукову, так і просвітницьку цінність. Було визначено технічні й концептуальні основи проєкту, обґрунтовано доцільність використання AI-рішень у мобільному середовищі.

У ході практики було проведено ґрунтовне теоретичне дослідження предметної області та математичних основ проєкту, результатом чого стало написання перших двох розділів дипломної роботи. Аналіз включав сучасні підходи до обробки біоакустичних сигналів, архітектури глибинних моделей для класифікації аудіо, а також технічні аспекти реалізації подібних систем у вигляді мобільних застосунків із використанням серверної підтримки та публічного API.

Паралельно з аналітичною частиною було розпочато реалізацію самого проєкту: створено початкову версію кросплатформного мобільного застосунку на базі React Native, а також закладено основу серверної частини (API-сервер) для обробки запитів та інференсу моделі.

ПЕРЕЛІК ПОСИЛАНЬ

1. J. Salamon, J. P. Bello, “Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification,” IEEE Signal Processing Letters, vol. 24, no. 3, pp. 279–283, 2017.
2. BirdCLEF Challenge, “BirdCLEF 2022 Task overview,” LifeCLEF Lab, 2022.
3. K. Chaurasiya, A. Singh, “Client-Server vs Edge Computing in Mobile AI Applications,” International Journal of Computer Applications, vol. 180, no. 6, 2022.
4. DCASE Challenge, “Detection and Classification of Acoustic Scenes and Events,” DCASE Community.
5. S. Hershey et al., “CNN Architectures for Large-Scale Audio Classification,” ICASSP 2017, 2017.
6. A. Krizhevsky, I. Sutskever, G. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” NIPS 2012.
7. React Native, “React Native · Learn once, write anywhere,” Meta Platforms
8. TensorFlow, “TensorFlow Lite | ML for Mobile and Edge Devices,” TensorFlow.org (дата звернення 15.05.2025)
9. ONNX, “Open Neural Network Exchange (ONNX),” onnx.ai (дата звернення 15.05.2025).

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
IMPORT PANDAS AS PD
IMPORT GC
IMPORT RE
IMPORT NUMPY AS NP
IMPORT OS
IMPORT LOGGING
IMPORT PSUTIL
FROM DATETIME IMPORT DATETIME

LOGGING.BASICCONFIG(
  LEVEL=LOGGING.INFO,
  FORMAT='%(ASCTIME)S - %(LEVELNAME)S - %(MESSAGE)S',
  HANDLERS=[
    LOGGING.FILEHANDLER('BIRD_CLASSIFICATION.LOG'),
    LOGGING.STREAMHANDLER()
  ]
)
LOGGER = LOGGING.GETLOGGER(__NAME__)

IMPORT WARNINGS
WARNINGS.FILTERWARNINGS("IGNORE")

FROM TQDM IMPORT TQDM
TQDM.PANDAS()

FROM PATHLIB IMPORT PATH
FROM IMBLEARN.OVER_SAMPLING IMPORT RANDOMOVERSAMPLER
FROM IMBLEARN.UNDER_SAMPLING IMPORT RANDOMUNDERSAMPLER
FROM SKLEARN.UTILS.CLASS_WEIGHT IMPORT COMPUTE_CLASS_WEIGHT
IMPORT MATPLOTLIB.PYPLOT AS PLT
FROM SKLEARN.METRICS IMPORT (
  ACCURACY_SCORE,
  CONFUSION_MATRIX,
  CLASSIFICATION_REPORT,
  F1_SCORE
)

IMPORT TORCH
IMPORT TRANSFORMERS
PRINT(TRANSFORMERS.__VERSION__)

IMPORT TORCHAUDIO
PRINT(TORCHAUDIO.__VERSION__)

IMPORT EVALUATE
FROM IPYTHON.DISPLAY IMPORT AUDIO
FROM TRANSFORMERS IMPORT AUTOFEATUREEXTRACTOR,
AUTOMODELFORAUDIOCLASSIFICATION, PIPELINE, TRAININGARGUMENTS, TRAINER
```

```
FROM DATASETS IMPORT DATASET, IMAGE, CLASSLABEL

RATE_HZ = 16000
MAX_SECONDS = 10.0
MAX_LENGTH = INT(RATE_HZ * MAX_SECONDS)
MIN_RECORDS_PER_LABEL = 25
TEST_SIZE = 0.15
BATCH_SIZE = 10
MAX_SEGMENTS_PER_CLASS = NONE
MAX_BIRD_SPECIES = 50

DEF LOG_MEMORY_USAGE(STEP_NAME):
  MEMORY = PSUTIL.VIRTUAL_MEMORY()
  LOGGER.INFO(F"{STEP_NAME} - MEMORY USAGE: {MEMORY.PERCENT:.1F}% ({MEMORY.USED/1024**3:.2F}GB USED / {MEMORY.TOTAL/1024**3:.2F}GB TOTAL)")

DEF LOAD_DATA():
  LOGGER.INFO("STARTING DATA LOADING PROCESS...")
  LOG_MEMORY_USAGE("BEFORE DATA LOADING")

  FILE_LIST = []
  LABEL_LIST = []

  LOGGER.INFO("SCANNING ARCHIVE DIRECTORY FOR .MP3 FILES...")
  FOR FILE IN PATH('ARCHIVE/VOICE OF BIRDS/VOICE OF BIRDS/').GLOB('*/*.MP3'):
    LABEL = STR(FILE).SPLIT('/')[-2].SPLIT('_')[0]
    FILE_LIST.APPEND(FILE)
    LABEL_LIST.APPEND(LABEL)

  LOGGER.INFO(F"FOUND {LEN(FILE_LIST)} AUDIO FILES FROM {LEN(SET(LABEL_LIST))} UNIQUE SPECIES")

  DD = PD.DATAFRAME()
  DD['FILE'] = FILE_LIST
  DD['LABEL'] = LABEL_LIST

  LOGGER.INFO("DATA LOADING COMPLETED SUCCESSFULLY")
  LOG_MEMORY_USAGE("AFTER DATA LOADING")
  RETURN DD

DEF SPLIT_AUDIO(FILE):
  TRY:
    LOGGER.DEBUG(F"PROCESSING AUDIO FILE: {FILE}")
    AUDIO, RATE = TORCHAUDIO.LOAD(STR(FILE))
    NUM_SEGMENTS = MAX(1, LEN(AUDIO[0]) // MAX_LENGTH)
    NUM_SEGMENTS = MIN(NUM_SEGMENTS, 6)
    LOGGER.DEBUG(F"AUDIO DURATION: {LEN(AUDIO[0])/RATE:.2F}S, SEGMENTS: {NUM_SEGMENTS}")

    SEGMENTED_AUDIO = []
    FOR I IN RANGE(NUM_SEGMENTS):
      START = I * MAX_LENGTH
```

```
        END = MIN((I + 1) * MAX_LENGTH, LEN(AUDIO[0]))
        SEGMENT = AUDIO[0][START:END]

        IF LEN(SEGMENT) < MAX_LENGTH:
            SEGMENT = TORCH.CAT([SEGMENT, TORCH.ZEROS(MAX_LENGTH - LEN(SEGMENT))])

        IF RATE != RATE_HZ:
            TRANSFORM = TORCHAUDIO.TRANSFORMS.RESAMPLE(RATE, RATE_HZ)
            SEGMENT = TRANSFORM(SEGMENT)

        SEGMENT = SEGMENT.SQUEEZE(0).NUMPY().ASTYPE(NP.FLOAT32)
        SEGMENTED_AUDIO.APPEND(SEGMENT)

    DF_SEGMENTS = PD.DATAFRAME({'AUDIO': SEGMENTED_AUDIO})
    LOGGER.DEBUG(F"SUCCESSFULLY PROCESSED {LEN(SEGMENTED_AUDIO)} SEGMENTS FROM
{FILE}")
    RETURN DF_SEGMENTS

  EXCEPT EXCEPTION AS E:
    LOGGER.ERROR(F"ERROR PROCESSING FILE {FILE}: {E}")
    RETURN NONE

DEF PROCESS_DATA_IN_BATCHES():
  LOGGER.INFO("="*60)
  LOGGER.INFO("STARTING BATCH DATA PROCESSING")
  LOGGER.INFO("="*60)

  LOG_MEMORY_USAGE("BEFORE PROCESSING")

  PRINT("LOADING INITIAL DATA...")
  DF = LOAD_DATA()

  LOGGER.INFO("DISPLAYING SAMPLE DATA:")
  SAMPLE_DATA = DF.SAMPLE(5)
  PRINT("SAMPLE DATA:")
  PRINT(SAMPLE_DATA)
  LOGGER.INFO(F"SAMPLE DATA:\N{SAMPLE_DATA}")

  LABEL_COUNTS = DF['LABEL'].VALUE_COUNTS()
  LOGGER.INFO(F"TOTAL LABELS FOUND: {LEN(LABEL_COUNTS)}")

  UNDERSAMPLED_LABELS = LABEL_COUNTS[LABEL_COUNTS <
MIN_RECORDS_PER_LABEL].INDEX
  LOGGER.INFO(F"REMOVING {LEN(UNDERSAMPLED_LABELS)} UNDERSAMPLED LABELS (<
{MIN_RECORDS_PER_LABEL} SAMPLES)")
  LOGGER.INFO(F"UNDERSAMPLED LABELS: {LIST(UNDERSAMPLED_LABELS)}")

  DF = DF[~DF['LABEL'].ISIN(UNDERSAMPLED_LABELS)]

  LOGGER.INFO(F"FILTERED DATA SHAPE: {DF.SHAPE}")
  PRINT(F"FILTERED DATA SHAPE: {DF.SHAPE}")
```

```
  AVAILABLE_LABELS = DF['LABEL'].UNIQUE()
  IF LEN(AVAILABLE_LABELS) > MAX_BIRD_SPECIES:
    LOGGER.INFO(F"RANDOMLY SELECTING {MAX_BIRD_SPECIES} BIRD SPECIES OUT OF {LEN(AVAILABLE_LABELS)} AVAILABLE")
    SELECTED_LABELS = NP.RANDOM.CHOICE(AVAILABLE_LABELS, SIZE=MAX_BIRD_SPECIES, REPLACE=FALSE)
    DF = DF[DF['LABEL'].ISIN(SELECTED_LABELS)]
    LOGGER.INFO(F"SELECTED SPECIES: {SORTED(SELECTED_LABELS)}")
    PRINT(F"REDUCED TO {MAX_BIRD_SPECIES} SPECIES FOR MEMORY EFFICIENCY")

  UNIQUE_LABELS = DF['LABEL'].UNIQUE()
  LOGGER.INFO(F"FINAL UNIQUE LABELS: {LEN(UNIQUE_LABELS)}")
  PRINT(F"UNIQUE LABELS: {LEN(UNIQUE_LABELS)}")

  ALL_SEGMENTS = []
  TOTAL_FILES = LEN(DF)
  TOTAL_BATCHES = (TOTAL_FILES + BATCH_SIZE - 1) // BATCH_SIZE

  LOGGER.INFO(F"PROCESSING {TOTAL_FILES} FILES IN {TOTAL_BATCHES} BATCHES OF {BATCH_SIZE}...")
  PRINT(F"PROCESSING {TOTAL_FILES} FILES IN BATCHES OF {BATCH_SIZE}...")

  PBAR = TQDM(RANGE(0, TOTAL_FILES, BATCH_SIZE), DESC="PROCESSING BATCHES")

  FOR I IN PBAR:
    BATCH_NUM = I // BATCH_SIZE + 1
    PBAR.SET_DESCRIPTION(F"PROCESSING BATCH {BATCH_NUM}/{TOTAL_BATCHES}")

    BATCH_DF = DF.ILOC[I:I+BATCH_SIZE]
    BATCH_SEGMENTS = []

    FOR IDX, ROW IN BATCH_DF.ITERROWS():
      INPUT_FILE = ROW['FILE']
      INPUT_LABEL = ROW['LABEL']

      RESULTING_DF = SPLIT_AUDIO(INPUT_FILE)
      IF RESULTING_DF IS NOT NONE:
        RESULTING_DF['LABEL'] = INPUT_LABEL
        BATCH_SEGMENTS.APPEND(RESULTING_DF)

    IF BATCH_SEGMENTS:
      BATCH_COMBINED = PD.CONCAT(BATCH_SEGMENTS, AXIS=0)
      ALL_SEGMENTS.APPEND(BATCH_COMBINED)

      DEL BATCH_SEGMENTS, BATCH_COMBINED
      GC.COLLECT()

      IF BATCH_NUM % 20 == 0:
        PBAR.WRITE(F"MEMORY CHECK AT BATCH {BATCH_NUM}: {PSUTIL.VIRTUAL_MEMORY().PERCENT:.1F}% USED")

  PBAR.CLOSE()
```

```
    logger.info("Combining all processed segments...")
    print("Combining all processed segments...")
    final_df = pd.concat(all_segments, axis=0)

    del all_segments, df
    gc.collect()

    logger.info(f"Final data shape before limiting: {final_df.shape}")

    if MAX_SEGMENTS_PER_CLASS is not None:
        logger.info(f"Limiting to maximum {MAX_SEGMENTS_PER_CLASS} segments per class...")
        limited_segments = []
        for label in final_df['label'].unique():
            label_data = final_df[final_df['label'] == label]
            if len(label_data) > MAX_SEGMENTS_PER_CLASS:
                label_data = label_data.sample(n=MAX_SEGMENTS_PER_CLASS, random_state=42)
            limited_segments.append(label_data)

        final_df = pd.concat(limited_segments, axis=0)
        del limited_segments
        gc.collect()

        logger.info(f"Final data shape after limiting: {final_df.shape}")
    else:
        logger.info("Using ALL available segments per class (no limit)")
        logger.info(f"Final data shape: {final_df.shape}")

    print(f"Final data shape: {final_df.shape}")

    initial_size = len(final_df)
    final_df = final_df[~final_df['audio'].isnull()]
    logger.info(f"Removed {initial_size - len(final_df)} null audio entries")

    logger.info("Data processing info:")
    print("Data info:")
    final_df.info()

    classes = np.unique(final_df['label'])
    logger.info(f"Final classes ({len(classes)}): {classes}")
    print(f"Classes: {classes}")

    logger.info("Calculating class weights...")
    weights = compute_class_weight(class_weight='balanced', classes=classes, y=final_df['label'])
    class_weights = dict(zip(classes, weights))

    logger.info("Class weights calculated:")
    print("Class weights:")
    for label, weight in class_weights.items():
```

```
        print(f"  {label}: {weight:.4f}")
        logger.info(f"  {label}: {weight:.4f}")

    log_memory_usage("After data processing complete")
    logger.info("BATCH DATA PROCESSING COMPLETED")

    return final_df, classes, class_weights

def create_dataset_in_chunks(df, chunk_size=200):
    logger.info(f"Creating dataset in chunks of {chunk_size}...")

    chunks = []
    for i in range(0, len(df), chunk_size):
        chunk = df.iloc[i:i+chunk_size].copy()
        chunks.append(chunk)
        logger.info(f"Created chunk {len(chunks)} with {len(chunk)} samples")

    datasets = []
    for i, chunk in enumerate(chunks):
        logger.info(f"Converting chunk {i+1}/{len(chunks)} to dataset...")
        chunk_dataset = Dataset.from_pandas(chunk)
        datasets.append(chunk_dataset)
        del chunk
        gc.collect()

        if (i + 1) % 5 == 0:
            torch.cuda.empty_cache() if torch.cuda.is_available() else None
            log_memory_usage(f"After chunk {i+1}")

    logger.info("Concatenating chunk datasets...")
    from datasets import concatenate_datasets
    final_dataset = concatenate_datasets(datasets)

    del datasets, chunks
    gc.collect()
    torch.cuda.empty_cache() if torch.cuda.is_available() else None

    return final_dataset

def train_model(df, classes, class_weights):
    logger.info("="*60)
    logger.info("STARTING MODEL TRAINING")
    logger.info("="*60)

    log_memory_usage("Before model training")

    labels_list = sorted(list(df['label'].unique()))
    logger.info(f"Training with {len(labels_list)} classes: {labels_list}")

    label2id, id2label = dict(), dict()
    for i, label in enumerate(labels_list):
        label2id[label] = i
```

```
    id2label[i] = label

logger.info(f"Label mappings created:")
logger.info(f"ID to Label: {id2label}")
logger.info(f"Label to ID: {label2id}")
print("Mapping of IDs to Labels:", id2label, '\n')
print("Mapping of Labels to IDs:", label2id)

logger.info("Creating ClassLabels object...")
ClassLabels = ClassLabel(num_classes=len(labels_list), names=labels_list)

logger.info("Converting DataFrame to HuggingFace Dataset using chunked approach...")
dataset = create_dataset_in_chunks(df, chunk_size=200)
log_memory_usage("After dataset creation")

def map_label2id(example):
    example['label'] = ClassLabels.str2int(example['label'])
    return example

logger.info("Mapping labels to IDs...")
dataset = dataset.map(map_label2id, batched=True, batch_size=100)

logger.info("Casting label column to ClassLabel...")
dataset = dataset.cast_column('label', ClassLabels)

logger.info(f"Splitting dataset with test_size={TEST_SIZE}")
dataset = dataset.train_test_split(test_size=TEST_SIZE, shuffle=True, stratify_by_column="label")

logger.info(f"Train dataset size: {len(dataset['train'])}")
logger.info(f"Test dataset size: {len(dataset['test'])}")

del df
gc.collect()
log_memory_usage("After dataset split and cleanup")

model_str = "wav2vec"
logger.info(f"Loading pre-trained model: {model_str}")

logger.info("Loading feature extractor...")
feature_extractor = AutoFeatureExtractor.from_pretrained(model_str)

logger.info("Loading audio classification model...")
model = AutoModelForAudioClassification.from_pretrained(model_str, num_labels=len(labels_list), ignore_mismatched_sizes=True)

model.config.id2label = id2label

trainable_params = model.num_parameters(only_trainable=True) / 1e6
logger.info(f"Model has {trainable_params:.2f}M trainable parameters")
print(trainable_params)
```

```
  LOG_MEMORY_USAGE("AFTER MODEL LOADING")

  DEF PREPROCESS_FUNCTION(BATCH):
    INPUTS = FEATURE_EXTRACTOR(BATCH['AUDIO'], SAMPLING_RATE=RATE_HZ,
MAX_LENGTH=MAX_LENGTH, TRUNCATION=TRUE)
    INPUTS['INPUT_VALUES'] = INPUTS['INPUT_VALUES'][0]
    RETURN INPUTS

  LOGGER.INFO("PREPROCESSING TRAINING DATASET...")
  DATASET['TRAIN'] = DATASET['TRAIN'].MAP(PREPROCESS_FUNCTION,
REMOVE_COLUMNS="AUDIO", BATCHED=FALSE)

  LOGGER.INFO("PREPROCESSING TEST DATASET...")
  DATASET['TEST'] = DATASET['TEST'].MAP(PREPROCESS_FUNCTION,
REMOVE_COLUMNS="AUDIO", BATCHED=FALSE)

  GC.COLLECT()
  LOG_MEMORY_USAGE("AFTER DATASET PREPROCESSING")

  LOGGER.INFO("LOADING ACCURACY METRIC...")
  ACCURACY = EVALUATE.LOAD("ACCURACY")

  DEF COMPUTE_METRICS(EVAL_PRED):
    PREDICTIONS = EVAL_PRED.PREDICTIONS
    PREDICTIONS = NP.EXP(PREDICTIONS) / NP.EXP(PREDICTIONS).SUM(AXIS=1,
KEEPDIMS=TRUE)
    LABEL_IDS = EVAL_PRED.LABEL_IDS
    ACC_SCORE = ACCURACY.COMPUTE(PREDICTIONS=PREDICTIONS.ARGMAX(AXIS=1),
REFERENCES=LABEL_IDS)['ACCURACY']
    RETURN {"ACCURACY": ACC_SCORE}

  BATCH_SIZE = 2
  WARMUP_STEPS = 20
  WEIGHT_DECAY = 0.01
  NUM_TRAIN_EPOCHS = 15
  MODEL_NAME = "BIRD_SOUNDS_CLASSIFICATION"

  LOGGER.INFO(F"TRAINING CONFIGURATION:")
  LOGGER.INFO(F" BATCH SIZE: {BATCH_SIZE}")
  LOGGER.INFO(F" EPOCHS: {NUM_TRAIN_EPOCHS}")
  LOGGER.INFO(F" WARMUP STEPS: {WARMUP_STEPS}")
  LOGGER.INFO(F" WEIGHT DECAY: {WEIGHT_DECAY}")

  TRAINING_ARGS = TRAININGARGUMENTS(
    OUTPUT_DIR=MODEL_NAME,
    LOGGING_DIR='./LOGS',
    NUM_TRAIN_EPOCHS=NUM_TRAIN_EPOCHS,
    PER_DEVICE_TRAIN_BATCH_SIZE=BATCH_SIZE,
    PER_DEVICE_EVAL_BATCH_SIZE=BATCH_SIZE,
    LEARNING_RATE=5E-6,
    LOGGING_STRATEGY='STEPS',
```

```
    LOGGING_FIRST_STEP=TRUE,
    LOAD_BEST_MODEL_AT_END=TRUE,
    LOGGING_STEPS=25,
    EVAL_STRATEGY='EPOCH',
    WARMUP_STEPS=WARMUP_STEPS,
    WEIGHT_DECAY=WEIGHT_DECAY,
    EVAL_STEPS=1,
    GRADIENT_ACCUMULATION_STEPS=4,
    GRADIENT_CHECKPOINTING=TRUE,
    SAVE_STRATEGY='EPOCH',
    SAVE_TOTAL_LIMIT=2,
    REPORT_TO=NONE,
    DATALOADER_PIN_MEMORY=FALSE,
    FP16=TRUE,
    DATALOADER_NUM_WORKERS=0,
    REMOVE_UNUSED_COLUMNS=TRUE,
    METRIC_FOR_BEST_MODEL='ACCURACY',
    GREATER_IS_BETTER=TRUE,
)

LOGGER.INFO("CREATING TRAINER OBJECT...")
TRAINER = TRAINER(
    MODEL=MODEL,
    ARGS=TRAINING_ARGS,
    TRAIN_DATASET=DATASET['TRAIN'],
    EVAL_DATASET=DATASET['TEST'],
    TOKENIZER=FEATURE_EXTRACTOR,
    COMPUTE_METRICS=COMPUTE_METRICS,
)

LOG_MEMORY_USAGE("BEFORE TRAINING")

LOGGER.INFO("STARTING INITIAL EVALUATION...")
TRAINER.EVALUATE()

LOGGER.INFO("STARTING MODEL TRAINING...")
TRAINER.TRAIN()

LOGGER.INFO("STARTING FINAL EVALUATION...")
TRAINER.EVALUATE()

LOGGER.INFO("MAKING PREDICTIONS ON TEST SET...")
OUTPUTS = TRAINER.PREDICT(DATASET['TEST'])

LOGGER.INFO(F"TEST METRICS: {OUTPUTS.METRICS}")
PRINT(OUTPUTS.METRICS)

Y_TRUE = OUTPUTS.LABEL_IDS
Y_PRED = OUTPUTS.PREDICTIONS.ARGMAX(1)

ACCURACY_FINAL = ACCURACY_SCORE(Y_TRUE, Y_PRED)
F1 = F1_SCORE(Y_TRUE, Y_PRED, AVERAGE='MACRO')
```

```
  LOGGER.INFO(F"FINAL RESULTS:")
  LOGGER.INFO(F"  ACCURACY: {ACCURACY_FINAL:.4F}")
  LOGGER.INFO(F"  F1 SCORE: {F1:.4F}")

  PRINT(F"ACCURACY: {ACCURACY_FINAL:.4F}")
  PRINT(F"F1 SCORE: {F1:.4F}")

  IF LEN(LABELS_LIST) <= 120:
    LOGGER.INFO("GENERATING CONFUSION MATRIX...")
    CM = CONFUSION_MATRIX(Y_TRUE, Y_PRED)

    DEF SAVE_CONFUSION_MATRIX(CM, CLASSES, TITLE='CONFUSION MATRIX', FIGSIZE=(18,
16)):
      PLT.FIGURE(FIGSIZE=FIGSIZE)
      PLT.IMSHOW(CM, INTERPOLATION='NEAREST', CMAP=PLT.CM.BLUES)
      PLT.TITLE(TITLE)
      PLT.COLORBAR()
      TICK_MARKS = NP.ARANGE(LEN(CLASSES))
      PLT.XTICKS(TICK_MARKS, CLASSES, ROTATION=90)
      PLT.YTICKS(TICK_MARKS, CLASSES)
      PLT.YLABEL('TRUE LABEL')
      PLT.XLABEL('PREDICTED LABEL')
      PLT.TIGHT_LAYOUT()
      PLT.SAVEFIG('CONFUSION_MATRIX.PNG', DPI=150, BBOX_INCHES='TIGHT')
      PLT.CLOSE()
      LOGGER.INFO("CONFUSION MATRIX SAVED AS CONFUSION_MATRIX.PNG")

    SAVE_CONFUSION_MATRIX(CM, LABELS_LIST)

    LOGGER.INFO("GENERATING PER-CLASS ACCURACY BAR CHART...")
    PER_CLASS_ACCURACY = CM.DIAGONAL() / CM.SUM(AXIS=1)

    DEF SAVE_ACCURACY_BAR_CHART(ACCURACIES, CLASSES, TITLE='PER-CLASS ACCURACY',
FIGSIZE=(20, 12)):
      PLT.FIGURE(FIGSIZE=FIGSIZE)
      BARS = PLT.BAR(RANGE(LEN(CLASSES)), ACCURACIES, COLOR='SKYBLUE',
EDGECOLOR='NAVY', ALPHA=0.7)
      PLT.TITLE(TITLE, FONTSIZE=16, FONTWEIGHT='BOLD')
      PLT.XLABEL('BIRD SPECIES', FONTSIZE=14)
      PLT.YLABEL('ACCURACY', FONTSIZE=14)
      PLT.XTICKS(RANGE(LEN(CLASSES)), CLASSES, ROTATION=45, HA='RIGHT')
      PLT.YLIM(0, 1)
      PLT.GRID(AXIS='Y', ALPHA=0.3)

      FOR I, (BAR, ACC) IN ENUMERATE(ZIP(BARS, ACCURACIES)):
        PLT.TEXT(BAR.GET_X() + BAR.GET_WIDTH()/2, BAR.GET_HEIGHT() + 0.01,
            F'{ACC:.3F}', HA='CENTER', VA='BOTTOM', FONTSIZE=8)

      PLT.TIGHT_LAYOUT()
      PLT.SAVEFIG('PER_CLASS_ACCURACY.PNG', DPI=150, BBOX_INCHES='TIGHT')
      PLT.CLOSE()
```

```
        LOGGER.INFO("PER-CLASS ACCURACY BAR CHART SAVED AS
PER_CLASS_ACCURACY.PNG")

        AVG_ACCURACY = NP.MEAN(ACCURACIES)
        LOGGER.INFO(F"AVERAGE PER-CLASS ACCURACY: {AVG_ACCURACY:.4F}")
        PRINT(F"AVERAGE PER-CLASS ACCURACY: {AVG_ACCURACY:.4F}")

        BEST_SPECIES = CLASSES[NP.ARGMAX(ACCURACIES)]
        WORST_SPECIES = CLASSES[NP.ARGMIN(ACCURACIES)]
        LOGGER.INFO(F"BEST PERFORMING SPECIES: {BEST_SPECIES}
({NP.MAX(ACCURACIES):.4F})")
        LOGGER.INFO(F"WORST PERFORMING SPECIES: {WORST_SPECIES}
({NP.MIN(ACCURACIES):.4F})")
        PRINT(F"BEST PERFORMING SPECIES: {BEST_SPECIES} ({NP.MAX(ACCURACIES):.4F})")
        PRINT(F"WORST PERFORMING SPECIES: {WORST_SPECIES} ({NP.MIN(ACCURACIES):.4F})")

      SAVE_ACCURACY_BAR_CHART(PER_CLASS_ACCURACY, LABELS_LIST)

  LOGGER.INFO("GENERATING CLASSIFICATION REPORT...")
  REPORT = CLASSIFICATION_REPORT(Y_TRUE, Y_PRED, TARGET_NAMES=LABELS_LIST,
DIGITS=4)
  LOGGER.INFO(F"CLASSIFICATION REPORT:\N{REPORT}")
  PRINT()
  PRINT("CLASSIFICATION REPORT:")
  PRINT()
  PRINT(REPORT)

  LOGGER.INFO("SAVING TRAINED MODEL...")
  TRAINER.SAVE_MODEL()

  LOG_MEMORY_USAGE("AFTER TRAINING COMPLETE")
  LOGGER.INFO("MODEL TRAINING COMPLETED")

IF __NAME__ == "__MAIN__":
  START_TIME = DATETIME.NOW()
  LOGGER.INFO("="*80)
  LOGGER.INFO("BIRD SOUND CLASSIFICATION - TRAINING PIPELINE STARTED")
  LOGGER.INFO(F"START TIME: {START_TIME}")
  LOGGER.INFO("="*80)

  TRY:
    PICKLE_FILE = 'PROCESSED_AUDIO_DATA.PKL'

    IF OS.PATH.EXISTS(PICKLE_FILE):
      LOGGER.INFO("FOUND EXISTING PROCESSED DATA FILE. LOADING...")
      PRINT("LOADING EXISTING PROCESSED DATA...")
      LOG_MEMORY_USAGE("BEFORE LOADING PROCESSED DATA")

      PROCESSED_DF = PD.READ_PICKLE(PICKLE_FILE)

      LOGGER.INFO(F"LOADED PROCESSED DATA WITH SHAPE: {PROCESSED_DF.SHAPE}")
      PRINT(F"LOADED PROCESSED DATA WITH SHAPE: {PROCESSED_DF.SHAPE}")
```

```
        LOG_MEMORY_USAGE("AFTER LOADING PROCESSED DATA")

        CLASSES = NP.UNIQUE(PROCESSED_DF['LABEL'])
        LOGGER.INFO(F"CLASSES ({LEN(CLASSES)}): {CLASSES}")

        LOGGER.INFO("CALCULATING CLASS WEIGHTS...")
        WEIGHTS = COMPUTE_CLASS_WEIGHT(CLASS_WEIGHT='BALANCED', CLASSES=CLASSES,
Y=PROCESSED_DF['LABEL'])
        CLASS_WEIGHTS = DICT(ZIP(CLASSES, WEIGHTS))

        LOGGER.INFO("CLASS WEIGHTS CALCULATED:")
        PRINT("CLASS WEIGHTS:")
        FOR LABEL, WEIGHT IN CLASS_WEIGHTS.ITEMS():
          PRINT(F"  {LABEL}: {WEIGHT:.4F}")
          LOGGER.INFO(F"  {LABEL}: {WEIGHT:.4F}")
      ELSE:
        LOGGER.INFO("NO EXISTING PROCESSED DATA FOUND. PROCESSING DATA FROM
SCRATCH...")
        PRINT("NO EXISTING PROCESSED DATA FOUND. PROCESSING DATA FROM SCRATCH...")

        PROCESSED_DF, CLASSES, CLASS_WEIGHTS = PROCESS_DATA_IN_BATCHES()

        TRY:
          LOGGER.INFO("SAVING PROCESSED DATA FOR FUTURE USE...")
          PRINT("SAVING PROCESSED DATA...")
          PROCESSED_DF.TO_PICKLE(PICKLE_FILE)
          LOGGER.INFO("PROCESSED DATA SAVED SUCCESSFULLY!")
          PRINT("PROCESSED DATA SAVED SUCCESSFULLY!")
        EXCEPT OSERROR AS E:
          LOGGER.WARNING(F"COULD NOT SAVE PROCESSED DATA: {E}")
          PRINT(F"COULD NOT SAVE PROCESSED DATA: {E}")
          PRINT("CONTINUING WITHOUT SAVING...")

      LOGGER.INFO("STARTING MODEL TRAINING PHASE...")
      PRINT("STARTING MODEL TRAINING...")
      TRAIN_MODEL(PROCESSED_DF, CLASSES, CLASS_WEIGHTS)

      END_TIME = DATETIME.NOW()
      DURATION = END_TIME - START_TIME
      LOGGER.INFO("="*80)
      LOGGER.INFO("TRAINING PIPELINE COMPLETED SUCCESSFULLY")
      LOGGER.INFO(F"END TIME: {END_TIME}")
      LOGGER.INFO(F"TOTAL DURATION: {DURATION}")
      LOGGER.INFO("="*80)
      PRINT("PROCESSING COMPLETE!")

    EXCEPT EXCEPTION AS E:
      LOGGER.ERROR(F"TRAINING PIPELINE FAILED WITH ERROR: {E}")
      LOGGER.EXCEPTION("FULL TRACEBACK:")
      RAISE
```