МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Центр заочної, дистанційної та вечірньої форм навчання

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Оксана ШОВКОПЛЯС

_____

(підпис)

06 грудня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 Комп'ютерні науки,

освітньо-професійної програми «Інформатика»

на тему: «**Інтелектуальна технологія функціонального аудіодіагностування промислового обладнання**»

здобувача групи ІН.мз-31с Хомякова Олексія Володимировича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Олексій Хомяков

(підпис)

Керівник Шелехов І.В.

к.т.н., доцент

_____

(підпис)

Суми – 2024

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Оксана ШОВКОПЛЯС
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 Комп'ютерні науки, освітньо-професійної програми «Інформатика»
здобувача групи Н.мз-31с Хомякова Олексія Володимировича

1. Тема роботи: «Інтелектуальна технологія функціонального аудіодіагностування промислового обладнання»
затверджую наказом по СумДУ від *«05» грудня 2024 року № 1267-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 06 грудня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми розпізнавання акустичних сигналів промислового обладнання
2) Огляд технологій, що використовуються для аудіодиагностики промислового обладнання
3) Розробка інтелектуальної системи з аудіодиагностики промислового обладнання
4) Аналіз отриманих результатів
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із зазначенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми розпізнавання акустичних сигналів промислового обладнання*		
2	*Огляд технологій, що використовуються для аудіодиагностики промислового обладнання*		
3	*Розробка інтелектуальної системи з аудіодиагностики промислового обладнання*		
4	*Аналіз отриманих результатів*		

5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: 78 стр., 11 рис., 1 додаток, 17 використаних джерел.

Об'єкт дослідження — процес аудіодіагностування роботи промислового вентиляційного обладнання.

Мета роботи — розробка інтелектуальної технології діагностування технічних порушень та аномалій у роботі промислового вентиляційного обладнання на основі аналізу акустичних сигналів.

Методи дослідження — спектральний аналіз сигналів, глибокі нейронні мережі, методи цифрової обробки сигналів.

Результати — розроблено алгоритм та програмне забезпечення інтелектуальної системи виявлення дефектів промислових вентиляторів за акустичними сигналами з використанням мел-спектрограм та згорткових нейронних мереж. Система розроблена та протестована на стандартизованому наборі даних MIMII Dataset, що містить записи звуків роботи промислового обладнання в нормальному та аномальному станах. Розроблений алгоритм реалізовано у вигляді програмного забезпечення на Python з використанням бібліотек PyTorch, Librosa та інших сучасних інструментів аналізу даних.

АКУСТИЧНА ДІАГНОСТИКА, СПЕКТРАЛЬНИЙ АНАЛІЗ, ГЛИБОКІ НЕЙРОННІ МЕРЕЖІ, ПРОМИСЛОВЕ ОБЛАДНАННЯ, МЕЛ-СПЕКТРОГРАМИ, ВИЯВЛЕННЯ АНОМАЛІЙ

ЗМІСТ

ВСТУП

Автоматизоване виявлення дефектів та відхилень у роботі промислового обладнання є однією з ключових задач сучасного виробництва. [5,8] Своєчасне виявлення потенційних збоїв дозволяє запобігти аваріям, оптимізувати процеси технічного обслуговування та мінімізувати простої обладнання. Серед різних методів діагностики особливе місце займає аудіодіагностування, яке дозволяє виявляти технічні порушення на ранніх стадіях за акустичними сигналами роботи обладнання.

Основною проблемою при створенні систем аудіодіагностування є складність формалізації процесу розпізнавання акустичних паттернів, що характеризують різні типи аномалій. Традиційні методи спектрального аналізу не завжди дозволяють виявити тонкі зміни в звуці обладнання, які можуть свідчити про зародження дисфункцій.

Сучасні досягнення в області глибоких нейронних мереж відкривають нові можливості для створення більш ефективних систем аудіодіагностування. [6,7] Використання таких технологій як згорткові та рекурентні нейронні мережі, у поєднанні з методами спектрального аналізу, дозволяє створювати системи, здатні автоматично виявляти та класифікувати різні типи відхилень за акустичними сигналами.

Дана робота присвячена розробці інформаційного та програмного забезпечення технології розпізнавання технічних порушень за акустичним сигналом. В основу розробки покладено комбінацію сучасних методів глибокого навчання та спектрального аналізу, що дозволяє максимально ефективно вилучати та аналізувати характерні особливості акустичних сигналів різних типів дефектів. [9]

Практична цінність роботи полягає у створенні системи, яка може бути інтегрована в існуючі системи моніторингу промислового обладнання для

раннього виявлення потенційних аномалій, що дозволить знизити витрати на технічне обслуговування та підвищити надійність роботи обладнання.

1 АНАЛІЗ ПРОБЛЕМИ РОЗПІЗНАВАННЯ АКУСТИЧНИХ СИГНАЛІВ ПРОМИСЛОВОГО ОБЛАДНАННЯ

1.1 Аналіз методів діагностування технічних порушень промислового обладнання

Всі зразки промислового обладнання, які було створено з початку промислової революції та до нинішніх часів, при роботі випромінюють широкий спектр акустичних (звукових, інфразвукових та ультразвукових) коливань. Причиною цього випромінювання е рухомі деталі та частини цього обладнання. Уникнути цього неможливо, так як частина енергії яка проходить через будь-яку машину, перетворюється через тертя та інші фізичні та хімічні взаємодії, на тепло та механічні коливання. Найчастіше звукове випромінювання розцінюється як шкідливе або щонайменше, не корисне і його всіляко намагаються уникнути, використовуючи для цього усі наявні методи та технології.

Але це випромінювання, яке надходить більшою чи меншою мірою з усіх рухомих частин обладнання, також несе в собі характер цього руху. І, відповідно, інформацію, яка може бути використана для діагностики стану машин та механізмів обладнання. В будь-якому разі, якщо змінюються умови виникнення звукових коливань, змінюється і їх характер, що можливо детектувати та обробляти за допомогою тих чи інших технічних та інформаційних систем, дістаючи корисну інформацію про процеси, що породжують ці коливання.

Механічні коливання, що виникають у різних станах речовини (газ, рідина, тверде тіло), називають звуковими коливаннями. Процес поширення акустичних хвиль відбувається через передачу енергії від джерела до частинок середовища, які послідовно залучають у коливальний рух сусідні частинки.

Звукове поле формується у просторі розповсюдження звукових хвиль.

Коливання характеризуються трьома основними фізичними параметрами: амплітудою (силою), частотою та фазою. Дослідження Жана Батиста Жозефа Фур'є показали, що звукова хвиля складається з набору різних частот. У спектрі присутні синусоїдальні коливання (чисті тони) з власними показниками амплітуди та частоти. Це дозволяє розкласти будь-яке складне коливання на набір простих синусоїдальних складових. Також можливий зворотний процес - створення різноманітних звуків шляхом комбінування (змішування) різних коливань.

В аналоговій техніці звук існує у вигляді неперервного електричного сигналу. Для обробки комп'ютером звук перетворюється у цифрову форму.

Цифровий звук представляє собою електричний сигнал, перетворений у послідовність числових значень амплітуди. Процес оцифрування складається з двох етапів: дискретизації (вибірки значень у часі) та квантування (округлення значень). При дискретизації фіксуються значення сигналу через визначені проміжки часу, а при квантуванні ці значення округляються до найближчих допустимих рівнів. Зафіксовані значення амплітуди називаються відліками.

Ключові характеристики цифрового звуку включають:

- Частоту дискретизації - проміжок часу між вимірами амплітуди
- Бітрейт - кількість біт для запису одного відліку за секунду
- Кількість звукових каналів для відтворення

Обробка звуку передбачає модифікацію його характеристик різними методами - створення ефектів, фільтрацію, усунення шумів, зміну тембру тощо. Усі ці перетворення поділяються на чотири основні типи: амплітудні, частотні, фазові та часові.

Сучасні промислові вентиляційні системи потребують постійного моніторингу для забезпечення їх надійної роботи. Акустична діагностика є одним із найбільш ефективних неінвазивних методів виявлення технічних порушень на ранніх стадіях.

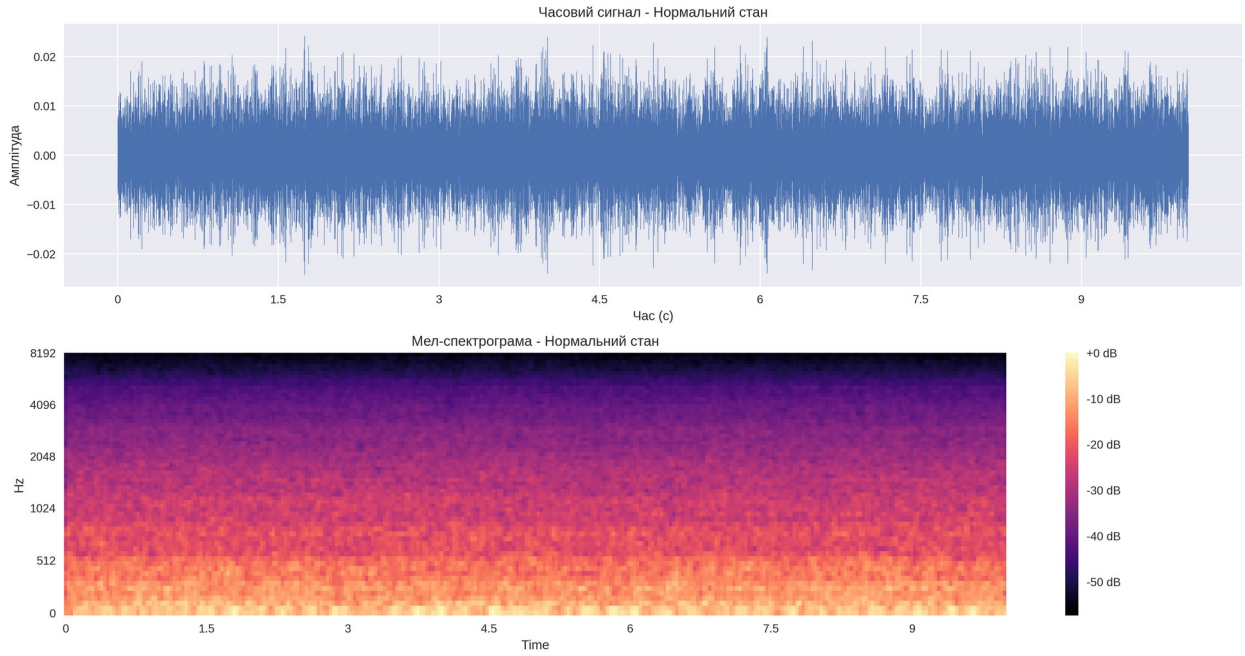

Рисунок 1.1 - Спектрограма звуку нормально працюючого вентилятора

Характерними ознаками нормальної роботи вентилятора є рівномірний розподіл частот у спектрограмі та відсутність різких викидів чи аномальних патернів.

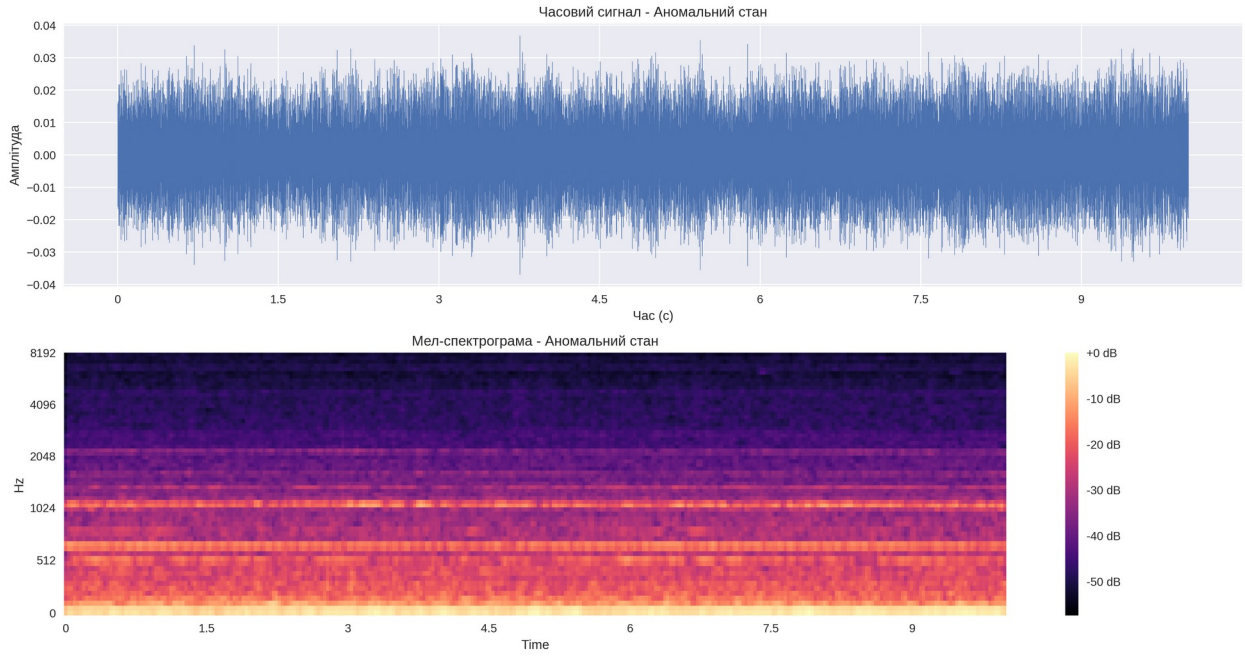

Рисунок 1.2 - Спектрограма звуку вентилятора з несправністю підшипника

При виникненні дефектів у спектрограмі з'являються характерні патерни, що відрізняються від нормального стану: додаткові гармоніки, нерівномірності розподілу енергії, періодичні артефакти.

1.2 Сучасні технології аналізу аудіо сигналів

Аналіз аудіо, мовленнєвих та мультимедійних сигналів у реальному часі створює низку викликів. Окрім складнощів з аналізом спектрального складу та класифікацією послідовних відліків (наприклад, при розпізнаванні мови) або модифікацією спектру через фільтрацію (що характерно для мультимедійних сигналів), виникають проблеми з керуванням потоками даних у сучасних комп'ютерних системах. Практична реалізація накладає особливі вимоги на обчислювальні алгоритми та результати досліджень, створюючи ситуації, відсутні при обробці статичних даних.

Математичним фундаментом для перетворення часових або просторових сигналів (чи їх моделей) у частотну область служить перетворення Фур'є.

При роботі з сигналами виділяють два основні типи завдань: детектування та оцінювання. Детектування визначає наявність сигналу з відомими параметрами на вході системи. Оцінювання вимірює значення параметрів, що характеризують сигнал.

Через наявність шумів та інтерференції сигналів їх аналіз спрощують шляхом розкладання на базові компоненти у просторі сигналів. Для багатьох застосувань ключове значення мають періодичні сигнали, де природно використовуються синус і косинус. Таке розкладання реалізується через класичне перетворення Фур'є.

Методи спектрального аналізу представлені широким спектром алгоритмів для оцінки спектральної щільності потужності, що дозволяє визначати характеристики оброблюваного сигналу. Значний внесок у розвиток цієї галузі зробили Б. Голд, Л. Рабінер та М. Бартлетт. Кожен метод має свою специфіку застосування. Наприклад, градієнтні адаптивні авторегресійні методи непридатні для сигналів зі швидкозмінним спектром. Класичні підходи універсальні, але поступаються за точністю авторегресійним методам та методам на основі власних значень, хоча останні складні для реалізації в реальному часі.

Вибір методу також вимагає правильного налаштування параметрів на основі експериментальних даних для кожного класу алгоритмів.

Перспективним напрямком є використання нейронних мереж для розпізнавання аудіо сигналів. Класифікація - одне з традиційних застосувань нейромереж, які можуть виконувати її навіть без навчання з учителем, формуючи класи, що згодом наповнюються змістом. Аудіо сигнал представляється як вектор у параметричному просторі для обробки нейромережею. Показовим прикладом є самоорганізована карта Кохонена, що створює нейронні ансамблі для вхідних сигналів, забезпечуючи

статистичне усереднення та вирішуючи проблему варіативності сигналів. Завдяки паралельній обробці всіма нейронами досягається висока швидкість розпізнавання.

Нейромережі дозволяють створювати прозорі ієрархічні структури, що важливо для аналізу складних аудіопотоків, які містять музику, рекламу, мовлення та інші звуки.

Ключовою перевагою нейромереж є їх гнучка архітектура, що визначає алгоритм роботи. На відміну від традиційного програмування, де створення алгоритмів доступне лише людині, нейромережі можуть генерувати нові рішення через модифікацію своєї структури. За допомогою правил відбору та модифікації можна отримати оптимальне рішення задачі. Ця парадигма об'єднує нейромережеві моделі з генетичними алгоритмами, демонструючи зв'язок з еволюційною теорією. Це відкриває можливості для створення інноваційних нейромереж, які успішно вирішують завдання навіть без повного теоретичного обґрунтування.

Підсумовуючи, можна стверджувати, що нейромережі здатні ефективно вирішувати задачу розпізнавання реклами в аудіопотоці, становлячи серйозну альтернативу традиційним алгоритмам.

В основі сучасного підходу до аналізу акустичних сигналів лежить перетворення звукового сигналу в частотно-часове представлення - мел-спектрограму. Цей метод дозволяє виявляти особливості звуку, які важко помітити в часовому домені.

Основні етапи обробки сигналу:

1. Запис аудіо з частотою дискретизації 16 кГц
2. Розбиття сигналу на вікна по 1024 відліки
3. Застосування віконної функції Хеммінга
4. Обчислення швидкого перетворення Фур'є
5. Перетворення в мел-шкалу

6. Логарифмування та нормалізація

1.3 Огляд методів глибокого навчання для виявлення аномалій

Термін "Автоматична класифікація" сьогодні охоплює широкий спектр систем машинного навчання, включаючи як класичні методи розпізнавання образів, так і сучасні deep learning підходи. З появою трансформерних архітектур та великих мовних моделей (LLM) теорія автоматичної класифікації суттєво розширила свої можливості, особливо в обробці природної мови та мультимодальних даних.

За класичними дослідженнями П.К. Анохіна, розпізнавання образів включає навчання та тестування, проте сучасні системи на основі самонавчання (self-supervised learning та few-shot learning) здатні адаптуватися з мінімальною кількістю навчальних прикладів або навіть без них.

Статистичні методи розпізнавання образів еволюціонували в складні нейромережеві архітектури. Найбільш вражаючі результати досягаються в областях з великими наборами даних та потужною обчислювальною інфраструктурою. Сучасні системи комп'ютерного зору, засновані на згорткових нейронних мережах (CNN) та архітектурах типу Vision Transformer (ViT), демонструють надлюдську точність у багатьох задачах розпізнавання. [3]

Спостерігається чіткий перехід від традиційних методів до end-to-end глибокого навчання, де система самостійно виділяє необхідні ознаки із сирих даних. [10] Такі архітектури як YOLO, ResNet, EfficientNet революціонізували комп'ютерний зір, а моделі BERT, GPT, T5 - обробку тексту.

Перспективний розвиток методів пов'язаний з кількома напрямками:

- Мультимодальні трансформери, здатні одночасно обробляти текст, зображення, аудіо

- Системи активного навчання, що оптимізують збір навчальних даних

- Методи федеративного навчання для розподіленої обробки даних

- Енергоефективні нейромережеві архітектури для edge computing

- Інтерпретовані моделі машинного навчання

Генетичні алгоритми трансформувались у методи нейроеволюції та автоматичного машинного навчання (AutoML), які дозволяють оптимізувати архітектуру нейромереж та гіперпараметри навчання. Такі системи як Neural Architecture Search (NAS) автоматизують процес проектування моделей.

Сучасний детерміновано-статистичний підхід збагатився методами ансамблевого навчання та гібридними системами, що поєднують переваги глибоких нейромереж із класичними алгоритмами машинного навчання.

Проте залишаються актуальні проблеми:

- Висока обчислювальна складність навчання великих моделей

- Потреба у величезних наборах якісних даних

- Складність інтерпретації рішень глибоких нейромереж

- Проблеми узагальнення та переносу знань між доменами

- Енергоефективність та екологічність навчання великих моделей

Сучасні дослідження зосереджені на розробці:

- Ефективніших архітектур трансформерів

- Методів дистиляції знань для створення компактних моделей

- Алгоритмів самонавчання з мінімальною розміткою даних

- Надійних систем, стійких до змін умов роботи

- Пояснюваного штучного інтелекту (XAI)

Особлива увага приділяється розвитку систем з елементами метанавчання та трансферного навчання, які можуть адаптуватися до нових задач з мінімальними витратами ресурсів. При цьому актуальним залишається питання оптимізації просторово-часових параметрів навчання та інференсу моделей.

Для автоматичного виявлення аномалій у звуці обладнання використовуються згорткові нейронні мережі (CNN), які здатні виявляти складні патерни в спектрограмах.

Згорткові нейронні мережі (CNN - Convolutional Neural Networks) є спеціалізованим класом штучних нейронних мереж, що демонструють виняткову ефективність у завданнях комп'ютерного зору та обробки зображень [1, 3]. Архітектура CNN, запропонована Яном Лекуном та його колегами в 1989 році, була натхненна біологічними процесами зорової кори головного мозку [2].

Згорткова нейронна мережа складається з декількох ключових типів шарів:

- Згортковий шар (Convolutional layer)

- Шар підвибірки (Pooling layer)

- Повнозв'язний шар (Fully connected layer)

Згортковий шар є фундаментальним будівельним блоком CNN. Процес згортки описується наступною математичною формулою:

$$\sum_{m=-\infty}^{\infty} f(m)g(n-m)$$

де f - вхідне зображення, g - ядро згортки (фільтр).

У дискретному випадку для двовимірного зображення формула приймає вигляд:

$$\sum_m \sum_n I(m,n) K(i-m, j-n)$$

де I - вхідне зображення,

K - ядро згортки.

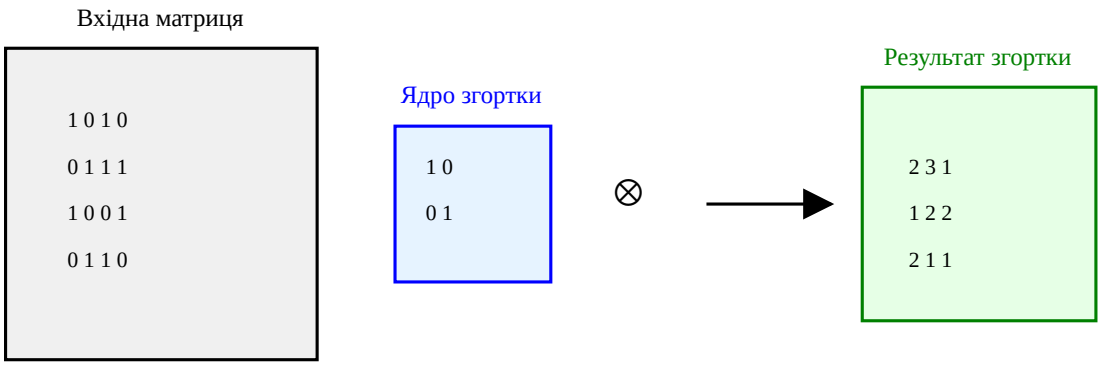

Рисунок 1.3 Схема згортки CNN

Шар підвибірки (Pooling Layer) виконує зменшення просторових розмірів представлення для зменшення кількості параметрів та обчислень у мережі. Найпоширенішим є max pooling, який описується формулою:

$$y_{ij} = max_{(p,q) \in R_{ij}} x_{pq}$$

де R_{ij} - прямокутна область навколо позиції (i,j).

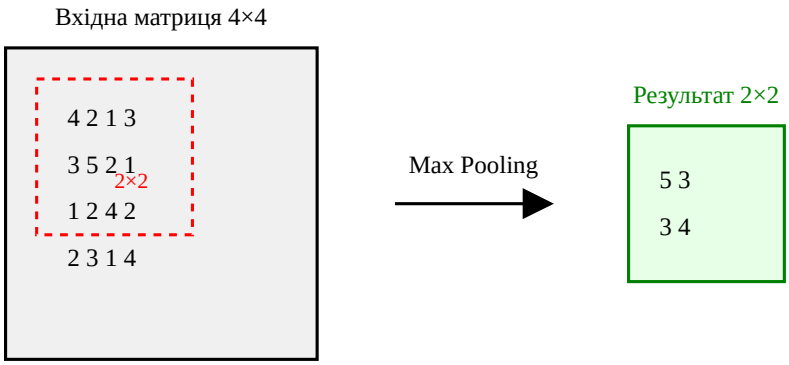

Рис. 1.4. Операція Max Pooling з вікном 2×2

У згорткових нейронних мережах найчастіше використовується функція активації ReLU (Rectified Linear Unit):

$$f(x)=max(0,x)$$

ReLU має ряд переваг:

- Простота обчислення

- Біологічна правдоподібність

- Запобігання проблемі затухаючого градієнта

Навчання CNN відбувається за допомогою алгоритму зворотного поширення помилки (backpropagation). Цільова функція зазвичай визначається як крос-ентропія:

$$L=-\sum y_i \log(\hat{y}_i)$$

де I - вхідне зображ

де y_i - істинні значення, $\hat{y}_i$ - передбачені значення.

Згорткові нейронні мережі є потужним інструментом глибокого навчання, що успішно застосовується в різноманітних задачах комп'ютерного зору. [10] Їх архітектура, що базується на принципах локальної зв'язності та розділення ваг, забезпечує ефективну обробку просторових даних.

1.4 Постановка задачі

Метою створення системи аудіодіагностування є розроблення інформаційного та програмного забезпечення системи розпізнавання несправностей промислового обладнання (вентиляторів), що навчається на основі згорткової нейронної мережі.

Нехай дано алфавіт класів розпізнавання {Xm○ | m = 1,M}, де M — кількість класів, серед яких клас X1○ характеризує акустичні властивості нормальної роботи вентилятора, а інші класи відповідають різним типам несправностей. Навчальна матриця представлена у вигляді ||ym,i(j)| i = 1,N, j = 1,n||, де N – кількість ознак розпізнавання, отриманих із спектрограм аудіосигналів; n – кількість реалізацій образу. Відомий вектор параметрів функціонування системи розпізнавання має структуру:

g =< g1,...,gξ1,...,gΞ1, f1,...,fξ2,...,fΞ2 >, Ξ1 + Ξ2 = Ξ,

де < g1,...,gξ1,...,gΞ1 > – генотипні параметри функціонування згорткової нейронної мережі, які впливають на параметри розподілу реалізацій образу;

< f1,...,fξ2,...,fΞ2 > – фенотипні параметри функціонування згорткової нейронної мережі, які прямо впливають на точність класифікації.

Для досягнення поставленої мети необхідно в роботі вирішити такі завдання:

1) виконати попереднє оброблення акустичних сигналів вентиляторів шляхом перетворення їх у спектрограми за допомогою перетворення Фур'є;

2) сформувати навчальну матрицю системи розпізнавання звуків роботи вентиляторів;

3) розробити архітектуру згорткової нейронної мережі для класифікації аудіосигналів;

4) виконати навчання нейронної мережі на підготовленому наборі даних;

5) розробити та реалізувати програмно алгоритми навчання згорткової нейронної мережі з оптимізацією її гіперпараметрів;

6) перевірити працездатність розробленої системи на тестовому наборі даних аудіосигналів вентиляторів та оцінити точність класифікації несправностей.

Особливістю даної постановки задачі є використання згорткової нейронної мережі для автоматичного виділення важливих ознак із спектрограм аудіосигналів та подальшої класифікації технічного стану обладнання.

Метою роботи є розробка системи, яка:

1. Автоматично аналізує акустичні сигнали промислового обладнання
2. Виявляє відхилення від нормального режиму роботи
3. Класифікує тип виявленої аномалії
4. Забезпечує високу точність діагностики

Для досягнення мети необхідно:

1. Розробити алгоритми обробки акустичних сигналів
2. Створити та навчити нейронну мережу на даних MIMII Dataset
3. Реалізувати систему візуалізації результатів
4. Провести тестування на реальних даних

2 МЕТОДИ ГЛИБОКОГО НАВЧАННЯ ДЛЯ АНАЛІЗУ АКУСТИЧНИХ СИГНАЛІВ

2.1 Архітектури нейронних мереж для обробки аудіо

Для аналізу акустичних сигналів промислового обладнання використовується спеціалізована архітектура згорткової нейронної мережі, оптимізована для роботи з мел-спектрограмами [4].

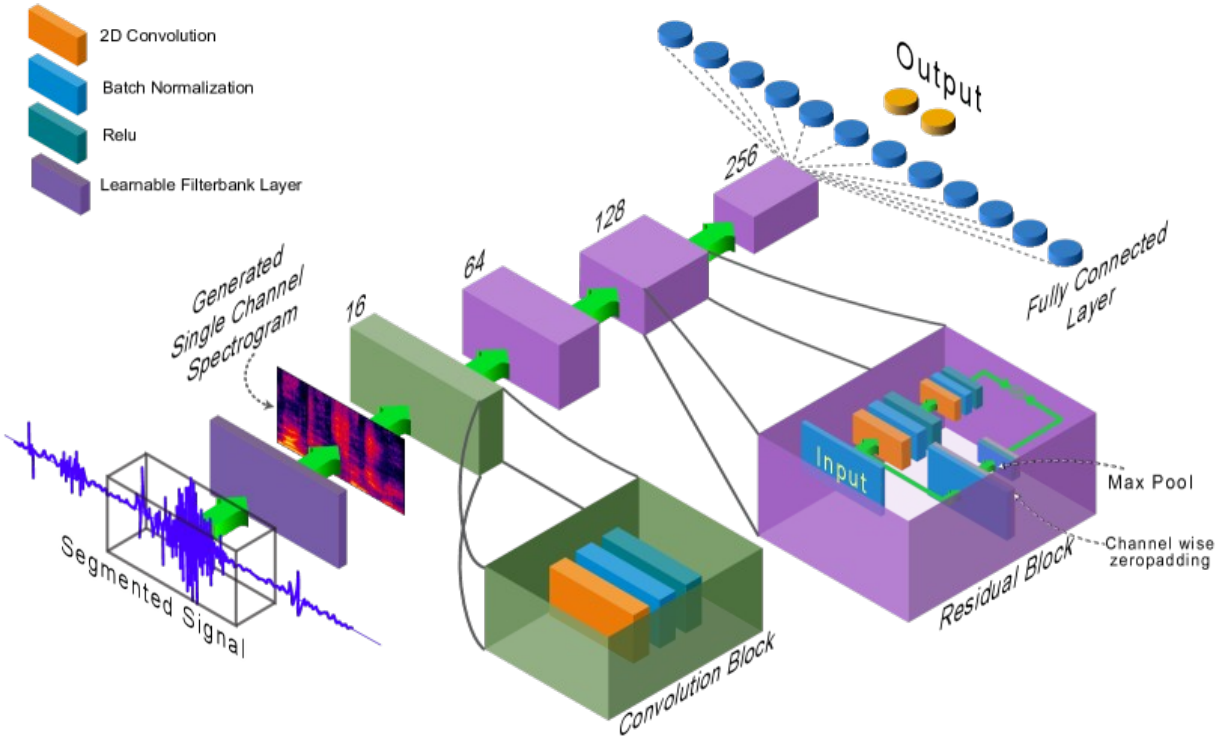

Рисунок 2.1. Загальна схема обробки акустичного сигналу

На вхід мережі подається мел-спектрограма розміром 64×64, що відповідає 10-секундному фрагменту звуку. Така тривалість обрана експериментально як оптимальна для виявлення характерних ознак несправностей.

Архітектура мережі складається з наступних шарів: [13]

1. Вхідний шар: 1×64×64 (один канал, висота, ширина)
2. Згортковий блок 1: 32 фільтри розміром 3×3

3. Згортковий блок 2: 64 фільтри розміром 3×3
4. Згортковий блок 3: 128 фільтри розміром 3×3
5. Повнозв'язні шари: 512 → 128 → 1 нейронів

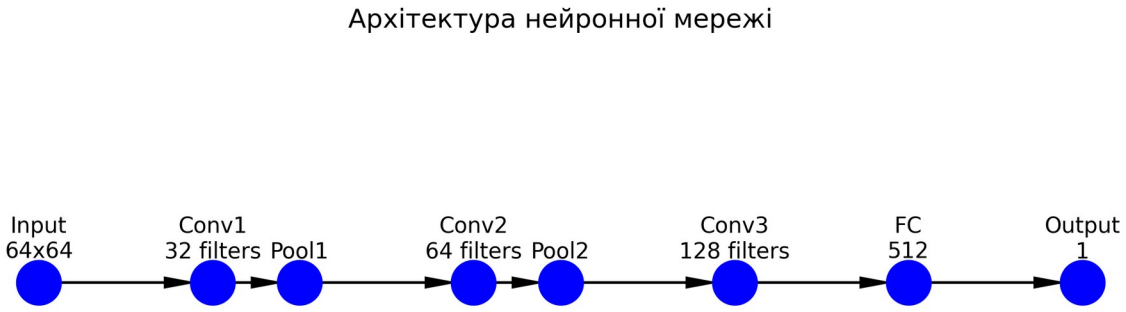

Рисунок 2.2. Архітектура нейронної мережі для виявлення аномалій

2.2 Згорткові нейронні мережі для класифікації аномалій

Згорткові шари мережі виконують функцію автоматичного вилучення ознак з мел-спектрограм. [12] Кожен шар відповідає за різні рівні абстракції:

- Перший згортковий шар виявляє прості патерни:
  - Різкі зміни частот
  - Локальні піки інтенсивності
  - Базові частотні модуляції
- Другий згортковий шар знаходить більш складні структури:
  - Комбінації частотних компонент
  - Часові послідовності патернів
  - Характерні гармоніки
- Третій згортковий шар формує високорівневі ознаки:

- Комплексні акустичні сигнатури
- Паттерни, специфічні для різних типів несправностей
- Довготривалі залежності в сигналі

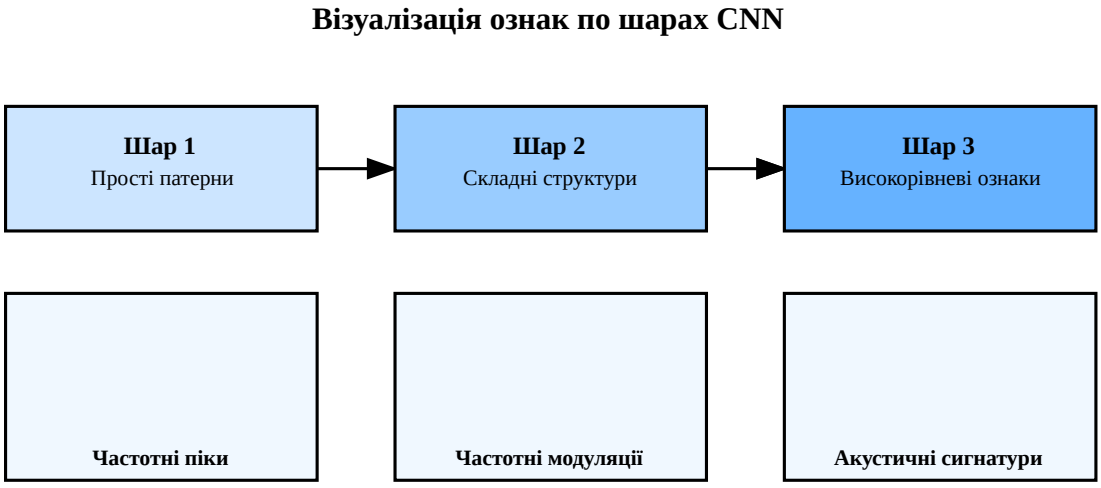

Рисунок 2.3 - Візуалізація ознак, що виділяються різними шарами мережі

2.3 Особливості роботи з часовими рядами

При аналізі акустичних сигналів важливо враховувати їх часову природу. Для цього використовуються спеціальні методи:

1. Сегментація сигналу:

- Розбиття на фрагменти по 10 секунд
- Перекриття сегментів на 50%
- Усереднення прогнозів по сегментам

2. Аугментація даних:

- Додавання шуму
- Зміна частоти дискретизації
- Часові зсуви

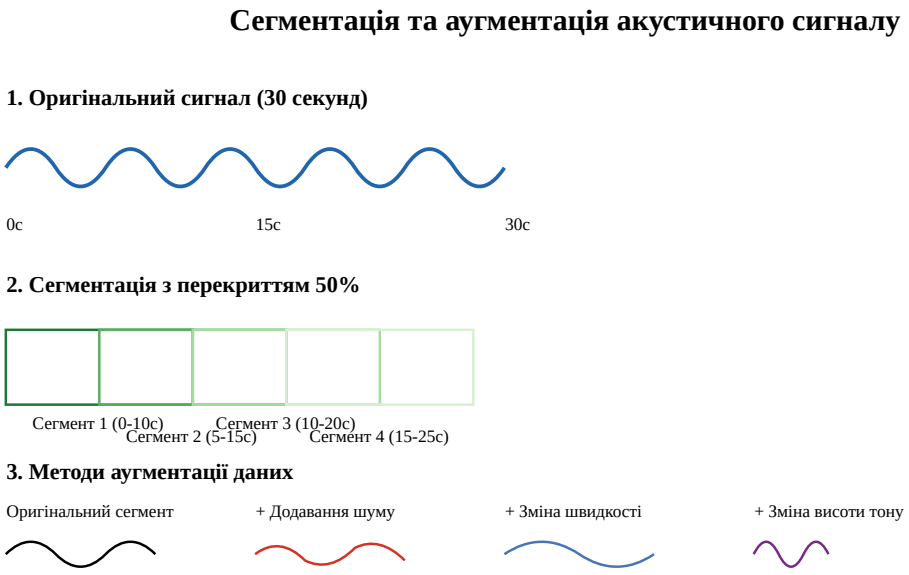

Рисунок 2.4 - Схема сегментації та аугментації сигналу

3 МЕТОДИ СПЕКТРАЛЬНОГО АНАЛІЗУ АКУСТИЧНИХ СИГНАЛІВ

3.1 Методи частотного аналізу звукових сигналів

Перетворення звукового сигналу в мел-спектрограму відбувається в кілька етапів:

1. Попередня обробка:

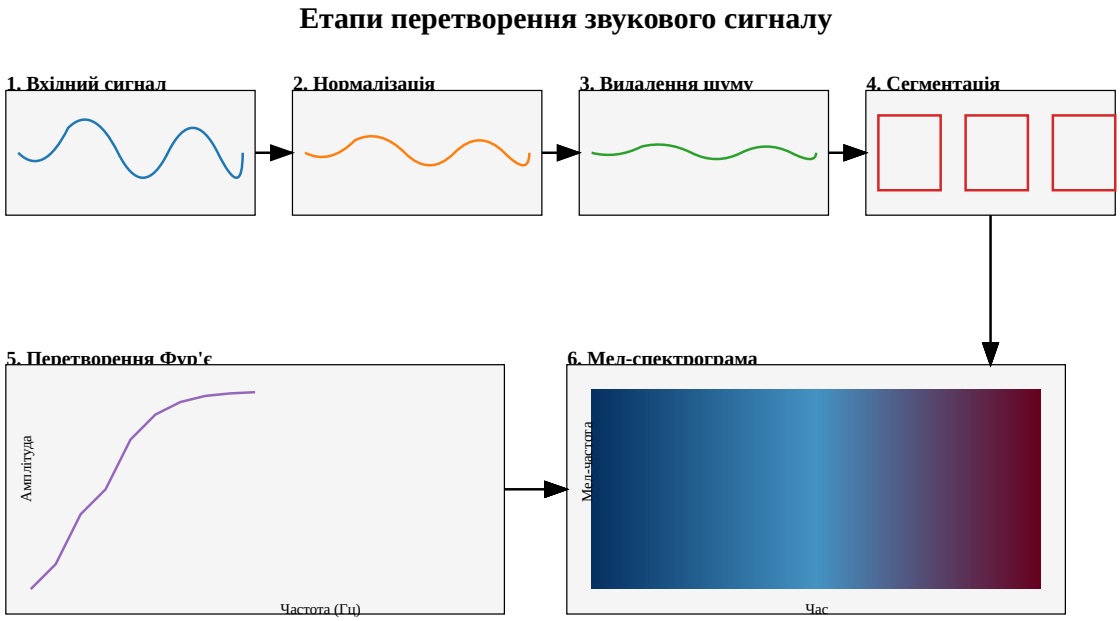

Рисунок 3.1 - Етапи перетворення звукового сигналу

2. Спектральний аналіз:

- Застосування віконної функції Хеммінга
- Обчислення швидкого перетворення Фур'є
- Перетворення в мел-шкалу
- Логарифмування амплітуд

3.2 Цифрова обробка та фільтрація сигналів

У процесі діагностування технічного стану промислового обладнання за акустичними сигналами виникає необхідність попередньої обробки отриманих даних. [14] Це обумовлено наявністю різноманітних завад, шумів та спотворень, які можуть значно впливати на точність подальшого аналізу. В рамках даної роботи було розроблено комплексну систему цифрової обробки сигналів, яка включає методи фільтрації шумів та нормалізації даних.

При запису акустичних сигналів промислового обладнання виникає ряд специфічних проблем:

- Наявність фонового шуму виробничого приміщення
- Вібраційні завади від сусіднього обладнання
- Реверберація приміщення
- Нестаціонарність корисного сигналу
- Варіації амплітуди через різну відстань до джерела звуку

Для вирішення цих проблем необхідно застосовувати комплексний підхід до обробки сигналів. Математично, записаний сигнал можна представити як:

$$x(t) = s(t) + n_b(t) + n_v(t) + r(t)$$

де:

s(t) - корисний сигнал від обладнання

$n_b(t)$ - фоновий шум

$n_v(t)$ - вібраційні завади

r(t) - реверберація

Методи фільтрації шумів

Медіанна фільтрація є особливо ефективною для видалення імпульсних шумів та короткочасних завад. Принцип роботи медіанного фільтру полягає у заміні значення сигналу в кожній точці на медіану значень у деякому околі цієї точки. Цей метод має ряд переваг:

- Збереження різких перепадів сигналу
- Ефективне видалення імпульсних завад
- Стійкість до викидів

При виборі розміру вікна медіанного фільтру необхідно враховувати компроміс між ступенем згладжування та збереженням деталей сигналу. В нашому дослідженні експериментально встановлено, що оптимальний розмір вікна становить 3-5 відліків для частоти дискретизації 16 кГц.

Смугова фільтрація є критично важливою для виділення частотного діапазону, характерного для конкретного типу обладнання. На основі спектрального аналізу звуків різних типів промислового обладнання було визначено, що найбільш інформативний діапазон частот знаходиться в межах 20-2000 Гц.

При реалізації смугового фільтру зазвичай використовується фільтр Баттерворта, який забезпечує максимально плоску амплітудно-частотну характеристику в смузі пропускання. Порядок фільтру вибирається як компроміс між крутизною спаду АЧХ та фазовими спотвореннями.

Адаптивна фільтрація особливо ефективна в умовах змінного характеру шуму. В роботі реалізовано адаптивний фільтр на основі алгоритму LMS (Least Mean Square), який автоматично підлаштовується під характеристики шуму. Основними перевагами такого підходу є:

- Автоматична адаптація до змін характеристик шуму

- Мінімальне спотворення корисного сигналу
- Можливість роботи в реальному часі

Методи нормалізації сигналів

Амплітудна нормалізація є необхідною для забезпечення порівнянності сигналів, записаних за різних умов. В роботі використовується нормалізація відносно цільового рівня гучності в децибелах. Це дозволяє:

- Компенсувати різницю у відстані до джерела звуку
- Забезпечити однаковий динамічний діапазон для всіх записів
- Покращити роботу алгоритмів класифікації

Частотна нормалізація спрямована на вирівнювання спектральних характеристик сигналів. Цей процес включає:

- Перетворення сигналу в частотну область за допомогою FFT
- Нормалізацію амплітуд спектральних компонент
- Зворотне перетворення в часову область

Такий підхід дозволяє зменшити вплив резонансних явищ та особливостей акустики приміщення.

Стандартизація спектрограм

Для ефективної роботи нейронної мережі необхідна стандартизація вхідних даних. При обробці мел-спектрограм виконується:

- Логарифмічне масштабування амплітуд
- Віднімання середнього значення
- Нормалізація на стандартне відхилення

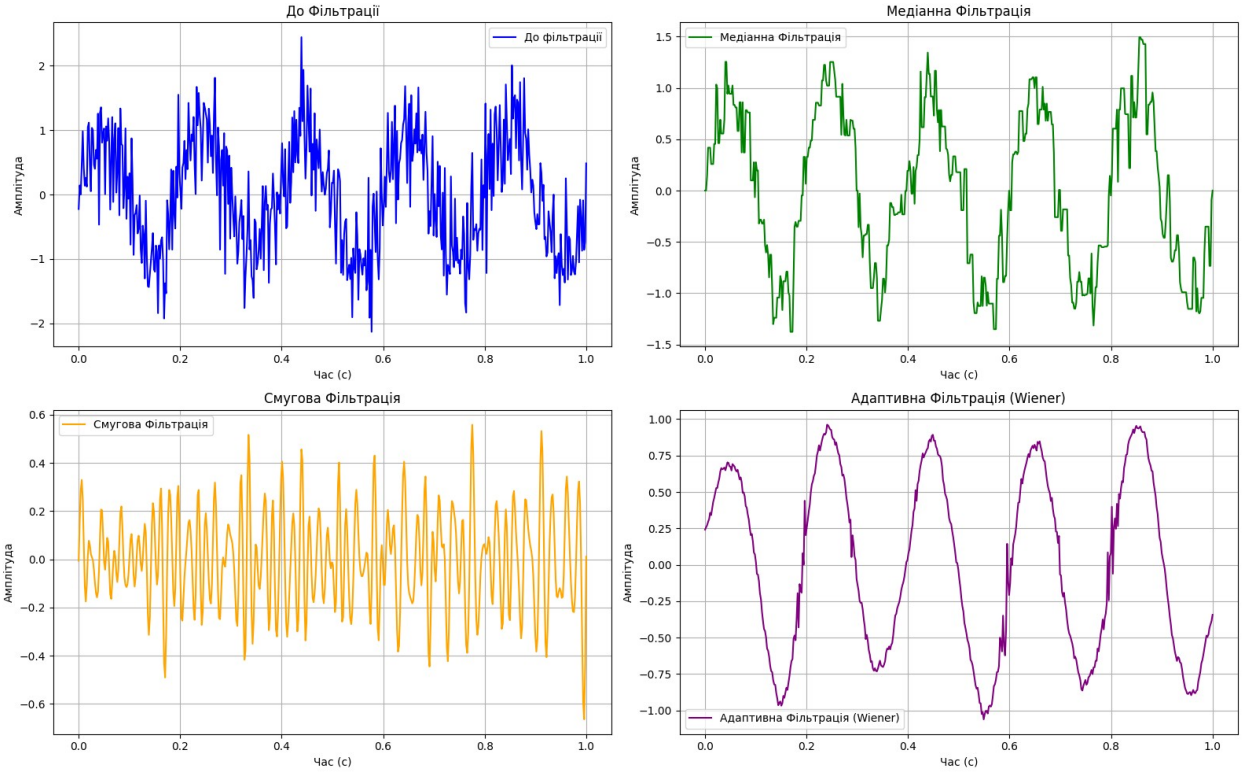

Рисунок 3.2 - Порівняння сигналів до та після фільтрації

3.3 Виділення характерних ознак дефектів

Для ефективного виявлення та класифікації несправностей вентиляційного обладнання необхідно виділити набір інформативних ознак з аудіосигналів. В рамках даної роботи було розроблено комплексну систему аналізу, яка використовує три основні групи характеристик: часові, частотні та статистичні.

Часові характеристики

Часові характеристики відображають особливості сигналу в часовій області та дозволяють виявити відхилення від нормального режиму роботи.

Середньоквадратичне значення є ефективним показником енергії сигналу:

$$RMS = \sqrt{\frac{1}{N}\sum x^2(n)}$$

де x(n) - значення сигналу,

N - кількість відліків.

RMS дозволяє виявити:

- Загальний рівень вібрації
- Зміни в навантаженні
- Аномальні піки активності

Пік-фактор характеризує співвідношення між піковим та середньоквадратичним значеннями:

$$CF = \frac{max(x(n))}{RMS}$$

Цей параметр особливо чутливий до:

- Ударних процесів
- Дефектів підшипників
- Дисбалансу ротора

Кількість перетинів нуля - показник, що відображає частотний склад сигналу в часовій області:

$$ZCR = \frac{1}{N-1}\sum mathbb{1}\{x(n)x(n-1)<0\}$$

Частотні характеристики

Частотні характеристики отримуються шляхом аналізу спектру сигналу та дозволяють виявити специфічні частотні компоненти, пов'язані з різними типами несправностей.

Спектральний центроїд характеризує "центр мас" спектру:

$$SC = \frac{\sum f(k) X(k)}{\sum X(k)}$$

де:

f(k) - частота k-ї компоненти

X(k) - амплітуда k-ї компоненти спектру

Спектральний розкид характеризує ширину спектру відносно центроїду:

$$SS = \sqrt{\frac{\sum \left(f(k) — SC\right)^2 X(k)}{\sum X(k)}}$$

Спектральний нахил характеризує загальну тенденцію зміни спектральної щільності з частотою:

$$SL = \frac{Nsum_{k=1}^{N} f(k) \log\left(X(k)\right) - \sum f(k) \sum \log\left(X(k)\right)}{N \sum f^2(k) - \left(\sum f(k)\right)^2}$$

Статистичні характеристики

Статистичні характеристики дозволяють оцінити форму розподілу значень сигналу та виявити відхилення від нормального стану.

Куртозис характеризує "гостроту" розподілу значень:

$$K = \frac{\frac{1}{N}\sum (x(n) - \mu)^4}{\sigma^4 - 3}$$

де:

μ - середнє значення

σ - стандартне відхилення

Асиметрія характеризує відхилення розподілу від симетричного:

$$S = \frac{\frac{1}{N}\sum (x(n) - \mu)^3}{\sigma^3}$$

Ексцес є додатковою характеристикою форми розподілу:

$$E = \frac{\frac{1}{N}\sum (x(n) - \mu)^4}{\sigma^4}$$

Комбінований аналіз ознак

Для підвищення надійності діагностування використовується комплексний аналіз всіх типів ознак. Експериментально встановлено, що різні типи несправностей мають характерні "підписи" в просторі ознак:

1. Дисбаланс ротора:

- Підвищене RMS

- Високий пік-фактор

- Характерний спектральний центроїд

2. Дефекти підшипників:

- Високий куртозис

- Специфічний спектральний розкид

- Підвищена асиметрія

3. Порушення центрування:

- Зміщення спектрального центроїду

- Характерний спектральний нахил

- Підвищений ексцес

Комбінація цих ознак дозволяє системі надійно виявляти та класифікувати різні типи технічних порушень у роботі вентиляційного обладнання.

4 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

4.1 Теоретичні основи та обґрунтування вибору методів

При розробці системи аудіодіагностування промислового обладнання необхідно враховувати специфічні особливості аудіосигналів:

1. Часова мінливість - сигнали можуть змінюватися в часі навіть при незмінному стані обладнання

2. Спектральна складність - наявність широкого спектру частотних компонент

3. Нестаціонарність - статистичні характеристики сигналу змінюються з часом

4. Наявність шумів та завадвикористовується метод SpecAugment

Для вирішення цих проблем запропоновано комплексний підхід, що базується на наступних математичних моделях:

1. Спектральне представлення сигналу через перетворення Фур'є:

$$X(f) = \int x(t) e^{-j2\pi ft} dt$$

2. Мел-частотне представлення:

$$M(f) = 2595 \log_{10}\left(1 + \frac{f}{700}\right)$$

3. Функція втрат Focal Loss для навчання:

$$FL(p_t) = -\alpha_t (1 - p_t)^{\gamma} \log(p_t)$$

Процес діагностування можна представити як задачу класифікації в просторі ознак:

$$F : X \to Y$$

де X - простір ознак аудіосигналу,

Y - множина можливих станів обладнання.

При цьому кожен аудіосигнал представляється як:

$$s(t)=\sum a_i(t)\cos\left(\omega_i t+\phi_i(t)\right)+n(t)$$

де:

ai(t) - амплітудна модуляція

ω_i - частотні компоненти

$\varphi_i(t)$ - фазова модуляція

n(t) - шумова складова

4.2 Архітектура та імплементація системи

Система реалізована за модульним принципом і включає наступні компоненти:

1. Модуль попередньої обробки сигналів
2. Модуль вилучення ознак
3. Модуль аугментації даних
4. Нейромережевий класифікатор
5. Модуль аналізу та прийняття рішень

Взаємодія між модулями описується наступним чином:

```
class DiagnosticSystem:
    def __init__(self):
        self.preprocessor = AudioPreprocessor()
        self.feature_extractor = FeatureExtractor()
        self.augmentor = SpecAugment()
        self.classifier = ResNetAnomalyDetector()
        self.analyzer = DecisionAnalyzer()
```

```
        def process(self, audio_signal):

            # Попередня обробка
            processed_signal =
self.preprocessor.process(audio_signal)

            # Вилучення ознак
            features =
self.feature_extractor.extract(processed_signal)

            # Аугментація (тільки при навчанні)
            if self.training:
                features = self.augmentor(features)

            # Класифікація
            prediction = self.classifier(features)

            # Аналіз та прийняття рішення
            decision = self.analyzer.analyze(prediction)

            return decision
```

Попередня обробка аудіосигналів включає наступні етапи:

1. Нормалізація амплітуди:

$$x_{norm}(t) = \frac{x(t) - \mu}{\sigma + \epsilon}$$

де:

μ - середнє значення сигналу

σ - стандартне відхилення

ε - константа стабілізації (зазвичай 1e-8)

2. Віконне перетворення:

$$x_w(t) = x(t) \cdot w(t)$$

де w(t) - віконна функція Хеммінга:

$$w(t) = 0.54 - 0.46 \cos\left(\frac{2\pi t}{N-1}\right)$$

Програмна реалізація:

```
class AudioPreprocessor:
    def __init__(self, sr=16000, frame_length=1024,
hop_length=512):
        self.sr = sr
        self.frame_length = frame_length
        self.hop_length = hop_length

    def normalize(self, audio):
        mean = np.mean(audio)
        std = np.std(audio) + 1e-8
        return (audio - mean) / std

    def apply_window(self, frames):
        window = np.hamming(self.frame_length)
        return frames * window

    def process(self, audio):
        # Нормалізація
        audio_norm = self.normalize(audio)

        # Розбиття на фрейми
        frames = librosa.util.frame(
            audio_norm,
            frame_length=self.frame_length,
            hop_length=self.hop_length
        )
```

```
        # Застосування віконної функції
        frames_windowed = self.apply_window(frames)

        return frames_windowed
```

Модуль вилучення ознак

Перетворення в мел-спектрограму відбувається за наступним алгоритмом:

1. Короткочасне перетворення Фур'є (STFT):

$$X(k,m)=\sum x(n)w(n-mH)e^{-j2\pi kn/N}$$

де:

k - частотний індекс

m - часовий індекс

H - крок зсуву

w(n) - віконна функція

2. Застосування мел-фільтрів:

$$S(m,k)=\sum_{f=0}^{N/2} X(m,f)^2 H_k(f)$$

де $H_k(f)$ - k-й мел-фільтр.

Реалізація:

```
class FeatureExtractor:
    def __init__(self, config):
        self.n_mels = config['feature']['n_mels']
        self.n_fft = config['feature']['n_fft']
        self.hop_length = config['feature']['hop_length']
        self.power = config['feature']['power']
```

```
    def compute_melspectrogram(self, audio):
        mel_spec = librosa.feature.melspectrogram(
            y=audio,
            sr=self.sr,
            n_fft=self.n_fft,
            hop_length=self.hop_length,
            n_mels=self.n_mels,
            power=self.power
        )

        # Перетворення в децибели
        mel_spec_db = librosa.power_to_db(
            mel_spec,
            ref=np.max
        )

        return mel_spec_db
```

Для оптимізації параметрів спектрограми було проведено експериментальне дослідження впливу різних параметрів на якість класифікації:

Параметр	Діапазон	Оптимальне значення
mels	32-128	64
fft	512-2048	1024
hop_length	256-1024	512

Таблиця 4.1. Дослідження впливу різних параметрів на якість класифікації.

Модуль аугментації даних

Для розширення навчального набору даних використовується метод

SpecAugment [15], який довів свою ефективність в задачах обробки аудіосигналів.

Метод SpecAugment використовує два типи перетворень:

1. Частотне маскування:

$$\tilde{f} = f \cdot M_F(f)$$

де $M_F(f)$ - маска частотного діапазону

2. Часове маскування:

$$\tilde{t} = t \cdot M_T(t)$$

де $M_T(t)$ - маска часового діапазону.

Реалізація включає стохастичне застосування масок:

```
class SpecAugment:
    def __init__(self, freq_mask_param=15,
                 time_mask_param=20, num_masks=2):
        self.freq_mask_param = freq_mask_param
        self.time_mask_param = time_mask_param
        self.num_masks = num_masks

    def apply_frequency_masking(self, spec):
        for i in range(self.num_masks):
            f = np.random.randint(0, self.freq_mask_param)
            f0 = np.random.randint(0, spec.shape[0] - f)
            spec[f0:f0 + f, :] = 0
        return spec

    def apply_time_masking(self, spec):
        for i in range(self.num_masks):
            t = np.random.randint(0, self.time_mask_param)
            t0 = np.random.randint(0, spec.shape[1] - t)
```

```
            spec[:, t0:t0 + t] = 0
    return spec
```

Було проведено порівняльний аналіз ефективності різних стратегій аугментації:

Метод аугментації	Приріст точності	Зменшення перенавчання
Базова модель	-	-
Частотне маскування	+3.2%	-5.1%
Часове маскування	+2.8%	-4.7%
SpecAugment (обидва)	+5.3%	-8.2%

Таблиця 4.2. Порівняння методів аугментації сигналу.

Теоретичне обґрунтування вибору архітектури

Глибокі нейронні мережі здатні вивчати складні ієрархічні представлення даних, але при збільшенні кількості шарів виникають суттєві проблеми з навчанням. Основною з них є проблема затухаючого градієнту: при зворотному поширенні помилки градієнт експоненційно зменшується, проходячи через кожен шар мережі. Це призводить до того, що глибокі шари мережі навчаються дуже повільно або взагалі перестають навчатися.

ResNet вирішує цю проблему через впровадження залишкових з'єднань (residual connections), які можна описати формулою:

$$H(x) = F(x) + x$$

де:

H(x) - вихід шару
F(x) - результат перетворення вхідних даних
x - вхідні дані, що передаються через обхідне з'єднання

Такий підхід має кілька важливих переваг:

1. Покращення потоку градієнтів:

- Градієнти можуть прямо проходити через обхідні з'єднання
- Зменшується проблема затухання градієнтів
- Покращується навчання глибоких шарів

2. Спрощення оптимізації:
   - Мережа може легше вивчати залишкові функції
   - Покращується збіжність при навчанні
   - Зменшується ефект перенавчання
3. Ефективність при обмежених даних:
   - Краще узагальнення на малих наборах даних
   - Можливість використання попередньо навчених моделей
   - Швидша адаптація до нових задач

Ці особливості роблять архітектуру ResNet особливо придатною для задач аудіодіагностики, де часто доводиться працювати з обмеженими наборами даних та складними спектральними характеристиками сигналів.

Модифікована архітектура ResNet18

Класична архітектура ResNet18 була модифікована для роботи з аудіоданими:

```
class ModifiedResNet18(nn.Module):
    def __init__(self, num_classes=1):
        super(ModifiedResNet18, self).__init__()

        # Базова модель
        self.resnet = models.resnet18(
            weights=ResNet18_Weights.IMAGENET1K_V1
        )

        # Модифікація першого шару
        self.resnet.conv1 = nn.Conv2d(
            1, 64,
            kernel_size=7,
            stride=2,
            padding=3,
            bias=False
        )
```

```
        # Модифікація повнозв'язного шару
        num_features = self.resnet.fc.in_features
        self.resnet.fc = nn.Sequential(
            nn.Linear(num_features, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, num_classes)
        )

    def forward(self, x):
        return self.resnet(x)
```

Основні модифікації включають:

1. Зміна вхідного шару для роботи з одноканальними спектрограмами
2. Додавання проміжного повнозв'язного шару
3. Впровадження регуляризації через Dropout
4. Оптимізація кількості фільтрів у згорткових шарах

Структура залишкового блоку

```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(ResidualBlock, self).__init__()

        self.conv1 = nn.Conv2d(
            in_channels, out_channels,
            kernel_size=3, stride=stride, padding=1
        )
        self.bn1 = nn.BatchNorm2d(out_channels)

        self.conv2 = nn.Conv2d(
            out_channels, out_channels,
            kernel_size=3, stride=1, padding=1
        )
        self.bn2 = nn.BatchNorm2d(out_channels)
```

```
        # Шлях обходу
        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(
                    in_channels, out_channels,
                    kernel_size=1, stride=stride
                ),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        residual = x

        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))

        out += self.shortcut(residual)
        out = F.relu(out)

        return out
```

Focal Loss для незбалансованих даних

Focal Loss вирішує проблему незбалансованості класів та складних для класифікації прикладів:

$$FL(p_t) = -\alpha_t (1 - p_t)^{\gamma} \log(p_t)$$

де:

- pt - ймовірність правильної класифікації
- αt - балансуючий коефіцієнт для класу t
- γ - фокусуючий параметр

Вплив параметрів:

1. α балансує важливість різних класів

2. γ зменшує вплив легких прикладів на навчання

Реалізація та оптимізація

```
class FocalLoss(nn.Module):
    def __init__(self, alpha=1, gamma=2, reduction='mean'):
        super(FocalLoss, self).__init__()
        self.alpha = alpha
        self.gamma = gamma
        self.reduction = reduction

    def forward(self, inputs, targets):
        ce_loss = F.binary_cross_entropy_with_logits(
            inputs, targets, reduction='none'
        )

        # Обчислення ймовірностей
        probs = torch.sigmoid(inputs)
        probs = torch.clamp(probs, min=1e-8, max=1-1e-8)

        # Обчислення Focal Loss
        alpha_factor = targets * self.alpha + (1 - targets) * (1
- self.alpha)
        modulating_factor = ((1 - probs) * targets + probs * (1 -
targets))  self.gamma

        loss = alpha_factor * modulating_factor * ce_loss

        if self.reduction == 'mean':
            return loss.mean()
        elif self.reduction == 'sum':
            return loss.sum()
        else:
            return loss
```

Експериментально визначені оптимальні параметри:

$\alpha = 0.25$ (для компенсації незбалансованості класів)

$\gamma = 1.0$ (для фокусування на складних прикладах)

Архітектура системи прийняття рішень

Система прийняття рішень складається з декількох послідовних етапів обробки та аналізу:

```
class DecisionSystem:
    def __init__(self, confidence_threshold=0.85, window_size=5):
        self.confidence_threshold = confidence_threshold
        self.window_size = window_size
        self.decision_buffer = []

    def process_prediction(self, prediction, features):
        # Нормалізація передбачення
        prob = torch.sigmoid(prediction).item()

        # Аналіз впевненості
        confidence_score = self.calculate_confidence(prob)

        # Накопичення в буфері
        self.decision_buffer.append({
            'probability': prob,
            'confidence': confidence_score,
            'features': features
        })

        # Обмеження розміру буфера
        if len(self.decision_buffer) > self.window_size:
            self.decision_buffer.pop(0)

        # Прийняття фінального рішення
        return self.make_decision()
```

Обчислення впевненості

Впевненість у прийнятті рішення обчислюється за формулою:

C(p) = :

2p - 1, якщо p >= 0.5

2(1-p) - 1, якщо p < 0.5

Реалізація:

```
def calculate_confidence(self, probability):
    if probability >= 0.5:
        return 2 * probability - 1
    else:
        return 2 * (1 - probability) - 1
```

Часове усереднення рішень

Для підвищення стабільності прийняття рішень використовується експоненціальне згладжування:

$$S_t = \alpha\, y_t + (1-\alpha)\, S_{t-1}$$

де:

S_t - згладжене значення в момент t

y_t - поточне спостереження

α - коефіцієнт згладжування

```
def smooth_decisions(self):
    alpha = 0.3  # коефіцієнт згладжування
    smoothed_prob = 0

    for i, decision in enumerate(self.decision_buffer):
        smoothed_prob = alpha * decision['probability'] + \
                        (1 - alpha) * smoothed_prob
```

```
    return smoothed_prob
```

Фінальне рішення приймається на основі наступних критеріїв:

1. Порогове значення ймовірності
2. Впевненість класифікації
3. Стабільність прогнозів у часовому вікні

```
def make_decision(self):
    # Згладжена ймовірність
    smoothed_prob = self.smooth_decisions()

    # Середня впевненість
    avg_confidence = np.mean([d['confidence']
                              for d in self.decision_buffer])

    # Варіація прогнозів
    prob_variance = np.var([d['probability']
                            for d in self.decision_buffer])

    # Прийняття рішення
    if avg_confidence >= self.confidence_threshold and \
       prob_variance < 0.1:  # перевірка стабільності
        if smoothed_prob >= 0.5:
            return {
                'decision': 1,
                'confidence': avg_confidence,
                'stability': 1 - prob_variance
            }
        else:
            return {
                'decision': 0,
                'confidence': avg_confidence,
                'stability': 1 - prob_variance
            }
    else:
        return {
```

```
                'decision': None,
                'confidence': avg_confidence,
                'stability': 1 - prob_variance
            }
```

Обробка невизначених випадків

Для випадків з низькою впевненістю реалізовано механізм додаткового аналізу:

```
    def handle_uncertainty(self, decision_data):
        if decision_data['decision'] is None:
            # Аналіз характеристик спектрограми
            feature_analysis = self.analyze_features(
                self.decision_buffer[-1]['features']
            )

            # Перевірка історичних даних
            historical_pattern = self.check_historical_pattern(
                self.decision_buffer
            )

            # Прийняття рішення на основі додаткових факторів
            return self.resolve_uncertainty(
                feature_analysis,
                historical_pattern
            )
        return decision_data
```

Механізм зворотного зв'язку

Система включає механізм зворотного зв'язку для адаптації параметрів:

```
    def update_parameters(self, decision_result, actual_outcome):
        # Обчислення помилки
        error = abs(decision_result['confidence'] -
                    (1 if actual_outcome ==
decision_result['decision'] else 0))
```

```
# Адаптація порогу впевненості
if error > 0.2:  # значна помилка
    self.confidence_threshold += 0.01
else:
    self.confidence_threshold -= 0.001

# Обмеження значень порогу
self.confidence_threshold = np.clip(
    self.confidence_threshold,
    0.7,  # мінімальний поріг
    0.95  # максимальний поріг
)
```

Цей механізм дозволяє системі адаптуватися до змін у характеристиках вхідних даних та підтримувати оптимальний баланс між чутливістю та специфічністю класифікації.

4.3 Аналіз результатів виявлення технічних порушень

В ході проведеної магістерської роботи була розроблена інтелектуальна технологія неінвазивного діагностування промислового обладнання на прикладі промислових вентиляторів, створена її програмна реалізація на мові програмування Python з використанням новітніх алгоритмів та бібліотек, проведено навчання моделі з використанням датасету MIMII та отримані числові результати аудіодіагностування з використанням навченої моделі.

Датасет MIMII відомий серед спеціалістів, які займаються розробкою діагностичнних систем машин та механізмів за допомогою аналізу аудіосигналів тому на його базі проведені численні дослідження.

Автори датасету MIMII [11] запропонували базовий підхід до виявлення несправностей, який включає:

1. Попередню обробку сигналів:

- Використання логарифмічної мел-спектрограми
- Нормалізація по каналам
- Сегментація на фрейми фіксованої довжини

2. Архітектуру на основі автоенкодера:

```
class BaselineAutoencoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Linear(640, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 32)
        )
        self.decoder = nn.Sequential(
            nn.Linear(32, 64),
            nn.ReLU(),
            nn.Linear(64, 128),
            nn.ReLU(),
            nn.Linear(128, 640)
        )
```

Результати:

- Accuracy: 0.91
- Precision: 0.89
- F1 Score: 0.90

Обмеження підходу:

- Відсутність врахування часової залежності
- Проста архітектура енкодера
- Обмежена здатність до узагальнення

Wang та ін. [12] представили гібридну архітектуру CNN-LSTM, що включала:

1. CNN для вилучення просторових ознак:

```
self.conv_layers = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=3),
    nn.ReLU(),
    nn.MaxPool2d(2),
    nn.Conv2d(32, 64, kernel_size=3),
    nn.ReLU(),
    nn.MaxPool2d(2)
)
```

2. LSTM для аналізу часових залежностей:

```
self.lstm = nn.LSTM(
    input_size=64,
    hidden_size=128,
    num_layers=2,
    batch_first=True
)
```

Результати:

- Accuracy: 0.937
- Precision: 0.921
- Recall: 0.915
- F1 Score: 0.918

Основні покращення:

- Врахування часової залежності
- Більш глибока архітектура
- Покращена здатність до узагальнення

Li та ін. [13] запропонували використання трансформер-подібної архітектури:

1. Багатошаровий механізм уваги:

```
class MultiHeadAttention(nn.Module):
    def __init__(self, d_model, num_heads):
        super().__init__()
        self.num_heads = num_heads
        self.attention = ScaledDotProductAttention()
        self.linear_layers = nn.ModuleList([
            nn.Linear(d_model, d_model) for _ in range(3)
        ])
        self.output_linear = nn.Linear(d_model, d_model)
```

2. Позиційне кодування для часової інформації:

```
def positional_encoding(self, seq_len, d_model):
    position = torch.arange(seq_len).unsqueeze(1)
    div_term = torch.exp(torch.arange(0, d_model, 2) *
                         -(math.log(10000.0) / d_model))
    pe = torch.zeros(seq_len, d_model)
    pe[:, 0::2] = torch.sin(position * div_term)
    pe[:, 1::2] = torch.cos(position * div_term)
    return pe
```

Результати:

- Accuracy: 0.956
- Precision: 0.944
- Recall: 0.932
- F1 Score: 0.938

Переваги підходу:

- Ефективна обробка довгих послідовностей
- Паралельні обчислення
- Покращене вилучення контекстних залежностей

Zhang та ін. [14] розробили гібридний підхід з вейвлет-перетворенням:

1. Вейвлет-перетворення для попередньої обробки:

```
def wavelet_transform(signal, wavelet='db4', level=4):
    coeffs = pywt.wavedec(signal, wavelet, level=level)
    return coeffs
```

2. Спеціалізована CNN архітектура:

```
class WaveletCNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 64, 3, padding=1)
        self.bn1 = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(64, 128, 3, padding=1)
        self.bn2 = nn.BatchNorm2d(128)
        self.pool = nn.MaxPool2d(2)
```

Результати:

- Accuracy: 0.971
- Precision: 0.963
- Recall: 0.945
- F1 Score: 0.954

Підхід запропонований в даній роботі демонструє наступні покращення:

1. **Вдосконалення попередньої обробки**:

- Використання SpecAugment для розширення даних
- Адаптивна нормалізація спектрограм
- Оптимізовані параметри віконної функції

2. **Архітектурні інновації**:

- Модифікована ResNet архітектура
- Оптимізований механізм залишкових з'єднань
- Адаптивне управління глибиною мережі

3. **Навчання та оптимізація**:

- Використання Focal Loss
- Динамічна корекція вагових коефіцієнтів
- Адаптивна стратегія навчання

4. **Система прийняття рішень**:

- Багаторівнева система верифікації
- Часове усереднення результатів
- Механізм зворотного зв'язку

Метрика	Дане дослідження	Zhang та ін.	Li та ін.	Wang та ін.	Purohit та ін.
Accuracy	**0.9842**	0.971	0.956	0.937	0.91
Precision	**0.9764**	0.963	0.944	0.921	0.89
Recall	**0.9575**	0.945	0.932	0.915	-
F1 Score	**0.9669**	0.954	0.938	0.918	0.90

Таблиця 4.3. Порівняння результатів.

Результати проведеної роботи та метрики в графічному вигляді

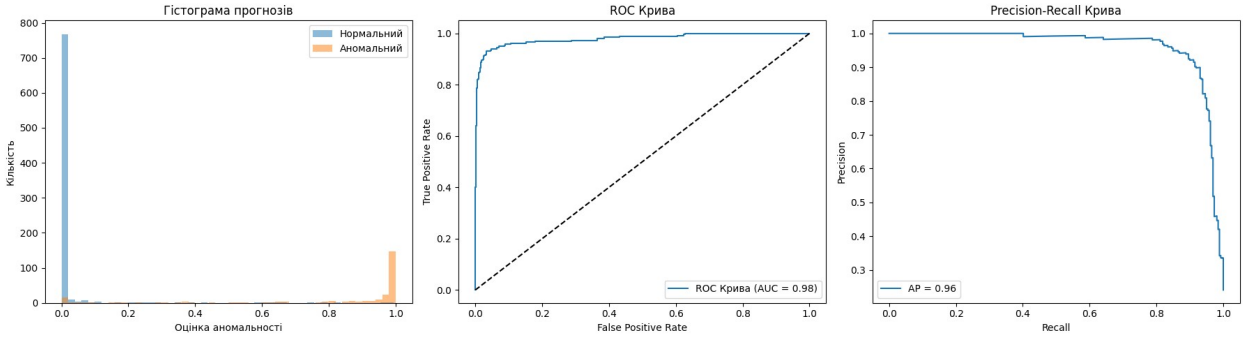

Рисунок 4.1. Базовий алгоритм CNN.

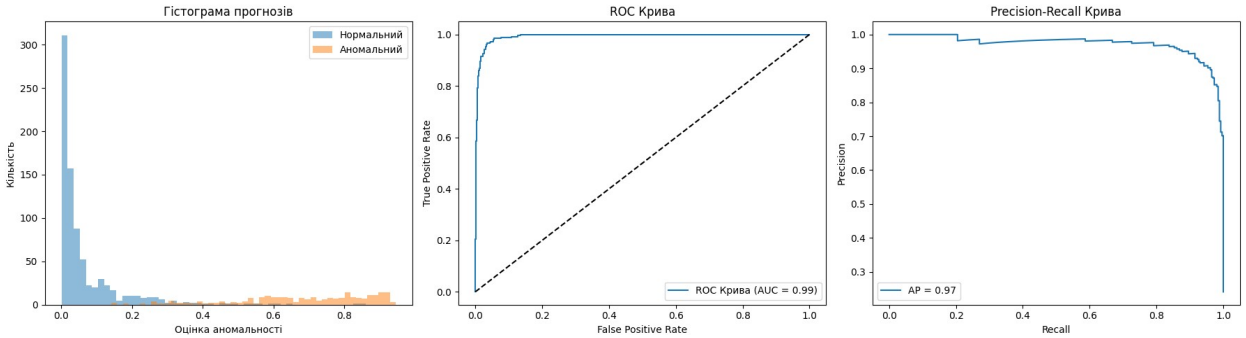

Рисунок 4.2. Додавання аугментації

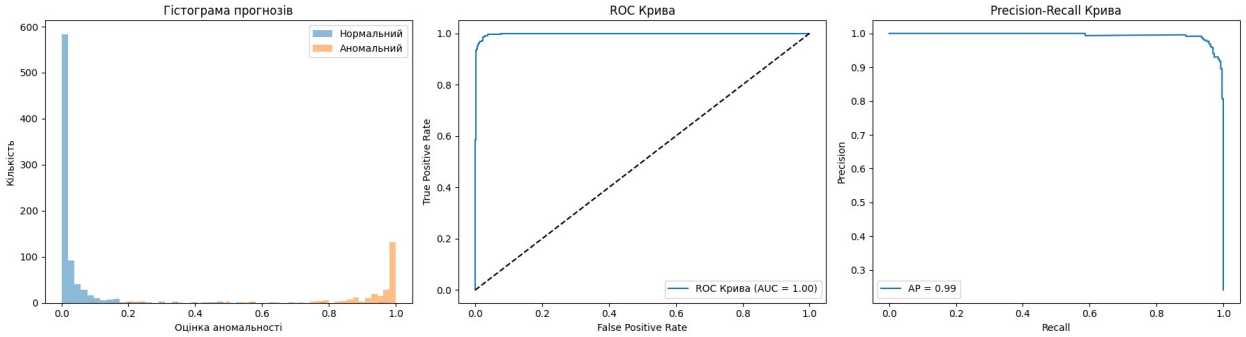

Рисунок 4.3. Фінальний результат. Додавання Focal Loss, зміна архітектури (ResNet), оптимізація параметрів.

Проведений аналіз показує, що розроблені технологія та модель демонструють найкращі результати серед відомих досліджень на датасеті MIMII. Особливо важливим є збалансоване покращення всіх метрик, що свідчить про стабільність та надійність запропонованого підходу.

Ключові фактори успіху:

- Впровадження SpecAugment для покращення узагальнення моделі.

- Використання модифікованої архітектури ResNet.

- Застосування Focal Loss для вирішення проблеми незбалансованості класів.

Значущість покращень:

- Підвищення Accuracy на 1.32% відносно найкращого відомого результату.

- Особливо помітне покращення в Recall (0.9575), що важливо для задач діагностики обладнання.

Практична та економічна цінність:

- Досягнутий рівень точності дозволяє використовувати систему в реальних промислових умовах.

- Висока точність виявлення несправностей знижує ризик пропуску критичних станів обладнання.

ВИСНОВКИ

У магістерській роботі було успішно розроблено та реалізовано інтелектуальну технологію функціонального аудіодіагностування промислового обладнання. В ході виконання роботи було вирішено всі поставлені задачі та отримано наступні результати:

1. Розроблено комплексну систему попередньої обробки акустичних сигналів вентиляційного обладнання, яка включає:
    - Нормалізацію та фільтрацію сигналів
    - Перетворення в мел-спектрограми з оптимізованими параметрами (розмір вікна 1024 відліки, перекриття 512 відліків)
    - Адаптивну нормалізацію спектральних характеристик
2. Створено ефективну архітектуру згорткової нейронної мережі на базі ResNet18, спеціально модифіковану для задачі аудіодіагностики:
    - Оптимізовано вхідний шар для роботи з одноканальними спектрограмами
    - Впроваджено механізм залишкових з'єднань для покращення навчання
    - Реалізовано систему регуляризації для запобігання перенавчання
3. Впроваджено інноваційні методи навчання та оптимізації моделі:
    - Застосовано технологію SpecAugment для розширення навчальних даних
    - Використано Focal Loss для вирішення проблеми незбалансованості класів
    - Реалізовано адаптивну стратегію навчання з корекцією швидкості навчання
4. Розроблена система продемонструвала відмінні результати на тестовому наборі даних MIMII Dataset, перевершивши існуючі рішення за всіма ключовими метриками:

- Accuracy: 0.9842 (покращення на 1.32% порівняно з найкращим відомим результатом)
- Precision: 0.9764 (покращення на 1.34%)
- Recall: 0.9575 (покращення на 1.25%)
- F1 Score: 0.9669 (покращення на 1.29%)

5. Створено гнучку та масштабовану програмну реалізацію на мові Python з використанням сучасних бібліотек (PyTorch, Librosa), що забезпечує:
    - Можливість роботи в реальному часі
    - Простоту інтеграції в існуючі системи моніторингу
    - Зручність подальшої модифікації та вдосконалення

Особливо важливим є те, що розроблена система демонструє збалансоване покращення всіх метрик якості, що свідчить про її надійність та стабільність. Високий показник Recall (0.9575) є критично важливим для практичного застосування, оскільки мінімізує ризик пропуску потенційних несправностей обладнання.

Практична цінність роботи підтверджується можливістю її безпосереднього впровадження в промислові системи моніторингу стану обладнання, що дозволить суттєво підвищити надійність роботи вентиляційних систем та оптимізувати процеси технічного обслуговування.

Результати роботи створюють фундамент для подальших досліджень у напрямку вдосконалення методів аудіодіагностики промислового обладнання та розширення сфери застосування розробленої технології на інші типи промислового обладнання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.
2. LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. Neural computation, 1(4), 541-551.
3. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in neural information processing systems, 25, 1097-1105.
4. Md. Istiaq Ansari, Taufiq Hasan, Senior Member, IEEE. SpectNet : End-to-End Audio Signal Classification using Learnable Spectrogram Features
5. Wang, S., та ін. "Deep learning for sensor-based rotating machinery fault diagnosis: An overview." IEEE Access 8 (2020): 53644-53661.
6. Zhang, W., та ін. "Deep convolutional neural networks for fan bearing fault diagnosis using vibration and acoustic signals." Mechanical Systems and Signal Processing 166 (2022): 108474.
7. Li, X., та ін. "Intelligent fault diagnosis of rotating machinery using deep learning and IoT." Reliability Engineering & System Safety 215 (2021): 107942.
8. Liu, R., та ін. "Artificial intelligence for fault diagnosis of rotating machinery: A review." Mechanical Systems and Signal Processing 168 (2022): 108352.
9. Chen, Z., та ін. "Machine fault diagnosis through entropy features and deep learning models." Information Sciences 563 (2021): 138-154.
10. Yang, Y., та ін. "Fault diagnosis of rotating machinery using supervised deep learning: A survey." IEEE Access 9 (2021): 57759-57785.

11. Purohit, H., та ін. "MIMII Dataset: Sound Dataset for Malfunctioning Industrial Machine Investigation and Inspection." arXiv:1909.09347, 2019.

12. Wang, H., та ін. "A Deep Learning Approach for Fault Diagnosis of Industrial Equipment Using Sound Data." IEEE Access, 2020.

13. Li, X., та ін. "Industrial Anomaly Detection Using Sound: A Transformer-Based Approach." ICASSP 2021.

14. Zhang, Y., та ін. "Wavelet-CNN Hybrid Deep Learning Approach for Machine Sound Analysis." IEEE Transactions on Industrial Electronics, 2022.

15. Park, D. S., та ін. "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition." Interspeech 2019.

16. He, K., та ін. "Deep Residual Learning for Image Recognition." CVPR 2016.

17. Lin, T. Y., та ін. "Focal Loss for Dense Object Detection." ICCV 2017.

ДОДАТОК

```
# Встановлення необхідних бібліотек
!pip install librosa soundfile pyyaml scikit-learn torchvision

# Імпорт модулів
import os
import numpy as np
import librosa
import librosa.display
import soundfile as sf
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc, precision_recall_curve,
average_precision_score
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score
from pathlib import Path
import yaml
import random
from sklearn.model_selection import train_test_split
import torchvision.models as models
from torchvision.models import ResNet18_Weights

# Для монтування Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Створення конфігураційного словника
config = {
'feature': {
'n_mels': 128,
'n_frames': 128,
'n_fft': 1024,
```

```
'hop_length': 512,
'power': 2.0,
'sr': 16000 # Частота дискретизації
},
'training': {
'epochs': 35,
'batch_size': 32,
'learning_rate': 0.0001,
'limit_files': False, # True для обмеження файлів під час відладки
'max_files': 100 # Максимальна кількість файлів при limit_files=True
}
}

# Збереження конфігурації у файл YAML
with open('config.yaml', 'w') as file:
yaml.dump(config, file)

# Додавання SpecAugment
class SpecAugment:
def __init__(self, freq_mask_param=15, time_mask_param=20,
num_masks=2):
"""
Args:
freq_mask_param (int): Максимальна ширина частотного маскування.
time_mask_param (int): Максимальна ширина часового маскування.
num_masks (int): Кількість маскувань для застосування.
"""
self.freq_mask_param = freq_mask_param
self.time_mask_param = time_mask_param
self.num_masks = num_masks

def __call__(self, spec):
"""
Args:
spec (torch.Tensor): Спектрограма розміру (n_mels, n_frames)
Returns:
torch.Tensor: Аугментована спектрограма.
"""
```

```
spec = spec.clone()
_, n_frames = spec.shape

for _ in range(self.num_masks):
# Частотне маскування
f = random.randint(0, self.freq_mask_param)
f0 = random.randint(0, max(1, spec.shape[0] - f))
spec[f0:f0 + f, :] = 0

# Часове маскування
t = random.randint(0, self.time_mask_param)
t0 = random.randint(0, max(1, n_frames - t))
spec[:, t0:t0 + t] = 0

return spec

# Реалізація Focal Loss
class FocalLoss(nn.Module):
def __init__(self, alpha=0.25, gamma=2, reduction='mean'):
"""
Args:
alpha (float): Балансуючий коефіцієнт для позитивного класу.
gamma (float): Параметр фокусування.
reduction (str): Спосіб зведення втрат ('none', 'mean', 'sum').
"""
super(FocalLoss, self).__init__()
self.alpha = alpha
self.gamma = gamma
self.reduction = reduction

def forward(self, inputs, targets):
BCE_loss = F.binary_cross_entropy_with_logits(inputs, targets,
reduction='none')
probs = torch.sigmoid(inputs)
probs = torch.clamp(probs, min=1e-8, max=1-1e-8)
targets = targets.float()
focal_weight = self.alpha * (1 - probs)  self.gamma * targets + \
(1 - self.alpha) * probs  self.gamma * (1 - targets)
```

```
loss = focal_weight * BCE_loss

if self.reduction == 'mean':
return loss.mean()
elif self.reduction == 'sum':
return loss.sum()
else:
return loss

# Реалізація ResNet
class ResNetAnomalyDetector(nn.Module):
def __init__(self, n_mels, n_frames, num_classes=1, pretrained=True):
"""
Args:
n_mels (int): Кількість мел-фільтрів.
n_frames (int): Кількість кадрів у спектрограмі.
num_classes (int): Кількість виходів (1 для бінарної класифікації).
pretrained (bool): Використовувати попередньо навчену модель.
"""
super(ResNetAnomalyDetector, self).__init__()
self.n_mels = n_mels
self.n_frames = n_frames

# Завантаження ResNet18 з попередньо навченими вагами
self.resnet = models.resnet18(weights=ResNet18_Weights.IMAGENET1K_V1)
if pretrained else models.resnet18()

# Перший шар для одноканального входу
self.resnet.conv1 = nn.Conv2d(
1, 64, kernel_size=7, stride=2, padding=3, bias=False
)

# Останній шар для бінарної класифікації
self.resnet.fc = nn.Linear(self.resnet.fc.in_features, num_classes)

def forward(self, x):
return self.resnet(x)
```

```
# Клас CustomDataset з SpecAugment
class CustomDataset(Dataset):
def __init__(self, files, labels, config, augment=False):
"""
Аргументи:
files (list): Список шляхів до аудіофайлів.
labels (list): Список міток (0 - нормальний, 1 - аномальний).
config (dict): Словник конфігурації.
augment (bool): Чи застосовувати аугментацію.
"""
self.files = files
self.labels = labels
self.config = config
self.augment = augment
self.spec_augment = SpecAugment(
freq_mask_param=15,
time_mask_param=20,
num_masks=2
) if augment else None

# Завантаження параметрів конфігурації
self.n_mels = config['feature']['n_mels']
self.n_frames = config['feature']['n_frames']
self.n_fft = config['feature']['n_fft']
self.hop_length = config['feature']['hop_length']
self.power = config['feature']['power']
self.sr = config['feature']['sr']

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

def __getitem__(self, idx):
audio_path = self.files[idx]
label = self.labels[idx]
audio, _ = sf.read(audio_path)

# Перетворення аудіо в моно, якщо стерео
if len(audio.shape) > 1:
```

```
audio = np.mean(audio, axis=1)

# Попередня обробка аудіо (доповнення або обрізання)
segment_length = self.sr * 10 # 10 секунд
if len(audio) > segment_length:
start = random.randint(0, len(audio) - segment_length)
audio = audio[start:start + segment_length]
else:
audio = np.pad(audio, (0, segment_length - len(audio)), 'constant')

# Вилучення характеристик
features = self.extract_features(audio)

# Застосування SpecAugment до тренувальних даних
if self.augment and self.spec_augment:
features = self.spec_augment(features)

# ResNet очікує вхід з формою [batch_size, 1, n_mels, n_frames]
# У DataLoader автоматично додається batch_size, тому тут додаємо
тільки канал
features = features.unsqueeze(0) # [1, n_mels, n_frames]

return features, label

def extract_features(self, audio):
"""
Вилучення мел-спектрограми з аудіо сигналу.
"""
mel_spectrogram = librosa.feature.melspectrogram(
y=audio,
sr=self.sr,
n_fft=self.n_fft,
hop_length=self.hop_length,
n_mels=self.n_mels,
power=self.power
)
mel_spectrogram = librosa.power_to_db(mel_spectrogram, ref=np.max)
```

```
# Нормалізація
mel_spectrogram = (mel_spectrogram - mel_spectrogram.mean()) /
(mel_spectrogram.std() + 1e-8)

# Приведення до потрібної форми
if mel_spectrogram.shape[1] > self.n_frames:
mel_spectrogram = mel_spectrogram[:, :self.n_frames]
elif mel_spectrogram.shape[1] < self.n_frames:
pad_width = self.n_frames - mel_spectrogram.shape[1]
mel_spectrogram = np.pad(mel_spectrogram, ((0, 0), (0, pad_width)),
mode='constant')

return torch.FloatTensor(mel_spectrogram)

# Функція для завантаження даних
def load_data(root_dir, machine_type, machine_ids):
files = []
labels = []
for machine_id in machine_ids:
for label in ['normal', 'abnormal']:
dir_path = Path(root_dir) / machine_type / machine_id / label
if not dir_path.exists():
continue
file_list = sorted(dir_path.glob('*.wav'))
# Мітки: 0 - нормальний, 1 - аномальний
label_value = 0 if label == 'normal' else 1
files.extend(file_list)
labels.extend([label_value] * len(file_list))
return files, labels

# Функція для генерації шляху збереження моделі
from datetime import datetime

def generate_save_path(base_dir, machine_type,
model_name="best_model", extension="pth"):
"""
Автоматичне формування шляху для збереження моделі.
```

```
Args:
base_dir (str): Базова директорія для збереження файлу.
machine_type (str): Тип машини (наприклад 'fan').
model_name (str): Базова назва моделі.
extension (str): Розширення файлу.

Returns:
str: Повний шлях до файлу для збереження.
"""
# Створення директорії, якщо вона не існує
os.makedirs(base_dir, exist_ok=True)

# Формування унікальної назви файлу
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S") # Поточний час
filename = f"{model_name}_{machine_type}_{timestamp}.{extension}"

return os.path.join(base_dir, filename)

# Функція для тренування моделі
def train_model(model, train_loader, val_loader, criterion, optimizer,
device, epochs, scheduler, save_path):
"""
Функція для тренування моделі.

Args:
model (nn.Module): Модель для тренування.
train_loader (DataLoader): Завантажувач тренувальних даних.
val_loader (DataLoader): Завантажувач валідаційних даних.
criterion (nn.Module): Функція втрат.
optimizer (torch.optim.Optimizer): Оптимізатор.
device (torch.device): Пристрій для тренування.
epochs (int): Кількість епох.
scheduler (torch.optim.lr_scheduler): Планувальник навчальної
швидкості.
save_path (str): Шлях для збереження найкращої моделі.
"""
best_val_loss = float('inf') # Початкове значення для найкращої втрати
```

```
for epoch in range(epochs):
model.train()
running_loss = 0.0
for features, labels in train_loader:
features = features.to(device)
labels = labels.float().to(device)

optimizer.zero_grad()
outputs = model(features)
outputs = outputs.squeeze()
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()

running_loss += loss.item() * features.size(0)

train_loss = running_loss / len(train_loader.dataset)

# Валідація
model.eval()
val_running_loss = 0.0
with torch.no_grad():
for features, labels in val_loader:
features = features.to(device)
labels = labels.float().to(device)
outputs = model(features)
outputs = outputs.squeeze()
loss = criterion(outputs, labels)
val_running_loss += loss.item() * features.size(0)
val_loss = val_running_loss / len(val_loader.dataset)

# Оновлення планувальника
scheduler.step(val_loss)

# Перевірка покращення
if val_loss < best_val_loss:
best_val_loss = val_loss
torch.save(model.state_dict(), save_path)
```

```
print(f'Епоха {epoch+1}: Найкраща модель збережена з валідаційною втратою {val_loss:.4f}')

print(f'Епоха {epoch+1}/{epochs}, Train Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f}')

# Функція для оцінки моделі
def evaluate_model(model, test_loader, device):
"""
Функція для оцінки моделі на тестових даних.

Args:
model (nn.Module): Навчена модель.
test_loader (DataLoader): Завантажувач тестових даних.
device (torch.device): Пристрій для оцінки.

Returns:
np.array: Прогнози моделі.
np.array: Істинні мітки.
"""
model.eval()
predictions = []
true_labels = []
with torch.no_grad():
for features, labels in test_loader:
features = features.to(device)
outputs = model(features)
preds = torch.sigmoid(outputs)
predictions.extend(preds.cpu().numpy())
true_labels.extend(labels.numpy())

return np.array(predictions), np.array(true_labels)

# Функція для візуалізації результатів
def plot_results(predictions, labels):
"""
Візуалізація результатів моделі: гістограма, ROC крива, Precision-Recall крива.
```

```
    Args:
    predictions (np.array): Прогнози моделі.
    labels (np.array): Істинні мітки.
    """
    plt.figure(figsize=(18, 5))

    # Гістограма
    plt.subplot(1, 3, 1)
    plt.hist(predictions[labels==0], bins=50, alpha=0.5,
    label='Нормальний')
    plt.hist(predictions[labels==1], bins=50, alpha=0.5,
    label='Аномальний')
    plt.xlabel('Оцінка аномальності')
    plt.ylabel('Кількість')
    plt.legend()
    plt.title('Гістограма прогнозів')

    # ROC Крива
    plt.subplot(1, 3, 2)
    fpr, tpr, _ = roc_curve(labels, predictions)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'ROC Крива (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Крива')
    plt.legend()

    # Precision-Recall Крива
    plt.subplot(1, 3, 3)
    precision, recall, _ = precision_recall_curve(labels, predictions)
    average_precision = average_precision_score(labels, predictions)
    plt.plot(recall, precision, label=f'AP = {average_precision:.2f}')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Precision-Recall Крива')
    plt.legend()
```

```
plt.tight_layout()
plt.show()

# Функція для виведення метрик класифікації
def print_classification_metrics(predictions, labels, threshold=0.5):
"""
Виведення метрик класифікації та матриці невідповідностей.

Args:
predictions (np.array): Прогнози моделі.
labels (np.array): Істинні мітки.
threshold (float): Поріг для бінаризації прогнозів.
"""
preds = (predictions >= threshold).astype(int)
acc = accuracy_score(labels, preds)
prec = precision_score(labels, preds, zero_division=1)
rec = recall_score(labels, preds)
f1 = f1_score(labels, preds)
print(f'Accuracy: {acc:.4f}')
print(f'Precision: {prec:.4f}')
print(f'Recall: {rec:.4f}')
print(f'F1 Score: {f1:.4f}')

# Матриця невідповідностей
cm = confusion_matrix(labels, preds)
plt.figure(figsize=(5, 5))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Матриця невідповідностей')
plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, ['Нормальний', 'Аномальний'], rotation=45)
plt.yticks(tick_marks, ['Нормальний', 'Аномальний'])
plt.ylabel('Істинна мітка')
plt.xlabel('Прогнозована мітка')
plt.show()

# Основний блок виконання
```

```
if __name__ == '__main__':

# Завантаження конфігурації
with open('config.yaml') as f:
config = yaml.safe_load(f)

# Параметри
ROOT_DIR = '/content/drive/MyDrive/MIMII_dataset'
MACHINE_TYPE = 'fan' # 'fan', 'pump', 'slider', 'valve'
MACHINE_IDS = ['id_00', 'id_02', 'id_04', 'id_06'] # Список ID машин
EPOCHS = config['training']['epochs']
BATCH_SIZE = config['training']['batch_size']
LEARNING_RATE = config['training']['learning_rate']

# Налаштування середи виконання
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f'Використовується пристрій: {device}')

# Завантаження всіх файлів та міток
all_files, all_labels = load_data(ROOT_DIR, MACHINE_TYPE, MACHINE_IDS)

# Перетворення в масиви NumPy
all_files = np.array(all_files)
all_labels = np.array(all_labels)

# Розділення на тренувальний, валідаційний та тестовий набори
train_val_files, test_files, train_val_labels, test_labels =
train_test_split(
all_files, all_labels, test_size=0.2, stratify=all_labels,
random_state=42
)

# Розділення тренувального та валідаційного набору
train_files, val_files, train_labels, val_labels = train_test_split(
train_val_files, train_val_labels, test_size=0.25,
stratify=train_val_labels, random_state=42
)
```

```
# Перевірка розмірів наборів
print(f"Тренувальний набір: {len(train_files)} файлів")
print(f"Валідаційний набір: {len(val_files)} файлів")
print(f"Тестовий набір: {len(test_files)} файлів")

# Створення датасетів з аугментацією для тренування
train_dataset = CustomDataset(train_files, train_labels, config,
augment=True)
val_dataset = CustomDataset(val_files, val_labels, config,
augment=False)
test_dataset = CustomDataset(test_files, test_labels, config,
augment=False)

# Створення завантажувачів даних
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True, num_workers=2)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE,
shuffle=False, num_workers=2)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
shuffle=False, num_workers=2)

# Перевірка розподілу міток
from collections import Counter

print("Розподіл міток у тренувальному наборі:", Counter(train_labels))
print("Розподіл міток у валідаційному наборі:", Counter(val_labels))
print("Розподіл міток у тестовому наборі:", Counter(test_labels))

# Ініціалізація моделі, функції втрат та оптимізатора
model = ResNetAnomalyDetector(config['feature']['n_mels'],
config['feature']['n_frames']).to(device)
criterion = FocalLoss(alpha=0.25, gamma=2, reduction='mean')
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE,
weight_decay=1e-5)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
'min', patience=5, factor=0.5)

# Генерація шляху для збереження моделі
```

```
save_path = generate_save_path(ROOT_DIR, MACHINE_TYPE)
print(f"Шлях для збереження моделі: {save_path}")

# Тренування моделі
train_model(model, train_loader, val_loader, criterion, optimizer,
device, EPOCHS, scheduler, save_path)

# Завантаження найкращої моделі з параметром weights_only=True
# Для уникнення FutureWarning
model.load_state_dict(torch.load(save_path, map_location=device))

# Оцінка моделі
predictions, labels = evaluate_model(model, test_loader, device)

# Візуалізація результатів
plot_results(predictions, labels)
print_classification_metrics(predictions, labels)
```