КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра радіотехніки та радіоелектронних систем

До захисту допущено: «На правах рукопису»
Завідувач кафедри _____ Ігор АНІСІМОВ
18 травня 2023 р.

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Технологія машинного навчання для зменшення рівня шумів у телекомунікаційних мережах»

Виконав:
студент 2-го курсу магістратури
денної форми навчання
спеціальності 172 Телекомунікації та радіотехніка
ОНП «Інформаційна безпека телекомунікаційних систем і мереж»
Пирожок Владислав Андрійович _____

Науковий керівник:
канд. тех. н., доц. Жиров Геннадій Борисович _____

Рецензент:
д. т. н., проф. Вишнівський Віктор Вікторович _____

Засвідчую, що у цій магістерській роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____

Робота допущена до захисту в ЕК рішенням кафедри радіотехніки та радіоелектронних систем від 18 травня 2023 р., протокол № 18.

Завідувач кафедри радіотехніки та радіоелектронних систем,
доктор фіз.-мат. наук, професор
Анісімов Ігор Олексійович _____

Київ – 2023

ЗМІСТ

ВСТУП

Україна повинна здійснювати заходи для підвищення свого економічного та промислового потенціалу, а також для покращення рівня життя громадян і збереження національної незалежності та суверенітету. Одним із способів досягнення цих цілей є розвиток сучасної, швидкої та надійної інфраструктури, включаючи системи телекомунікацій.

Телекомунікаційні мережі є необхідною складовою нашого повсякденного життя, і їх якість напряму впливає на ефективність і якість комунікації між людьми та машинами.

У телекомунікаційних мережах проблема шумів є однією з найбільших, яка може призвести до втрати даних, погіршення якості зв'язку та відмов у роботі системи. Оскільки такі мережі постійно змінюються та розвиваються, боротьба зі шумами може бути викликом для дослідників та інженерів. Застосування технології машинного навчання може стати важливим інструментом у цій боротьбі. Зокрема, алгоритми машинного навчання можуть використовуватися для аналізу даних з телекомунікаційних мереж, виявлення, класифікації та видалення шумів з сигналу.

Застосування технології машинного навчання може відіграти важливу роль у поліпшенні якості зв'язку в телекомунікаційних мережах, зменшенні втрат даних та збільшенні ефективності їх роботи. Це має вагому значимість для бізнесу та громадян, які відчувають потребу у безперебійному функціонуванні телекомунікаційних мереж для своєї роботи, навчання, спілкування, розваг та інших цілей.

Застосування технології машинного навчання також може допомогти знизити витрати на підтримку телекомунікаційних мереж, скоротити час реакції на проблеми в мережі та забезпечити ефективне використання ресурсів.

Крім того, дослідження в галузі технології машинного навчання для зменшення рівнів шуму у телекомунікаційних мережах може сприяти розвитку індустрії телекомунікацій та створенню нових робочих місць. Отже, тема

дослідження "Технологія машинного навчання для зменшення рівнів шумів у телекомунікаційних мережах" є актуальною, оскільки може відіграти ключову роль у покращенні якості зв'язку, забезпеченні безперебійної роботи телекомунікаційних мереж, зниженні витрат на підтримку мережі та сприяти розвитку індустрії телекомунікацій.

Мета дипломної роботи - дослідити можливості та параметри нейронних мереж для фільтрації мовних сигналів на фоні шумових завад в телекомунікаційних мережах. Для досягнення цієї мети сформульовано і вирішено наступні конкретні науково-технічні завдання:

1. Порівняти переваги та недоліки методів зменшення шумів у аудіо сигналах за допомогою штучних нейронних мереж з іншими методами адаптивної обробки сигналів.
2. Дослідити параметри нейронних мереж та визначити оптимальні параметри для покращення результатів зменшення шумів у нейронних мережах.
3. Розробити алгоритм навчання нейронної мережі та програмний продукт на мові програмування Python для реалізації даного алгоритму.
4. Провести моделювання у програмному комплексі та проаналізувати отримані результати.

Об'єктом дослідження є методи фільтрації мовних сигналів у телекомунікаційних мережах, а предметом - методи зменшення шумової складової при цифровій обробці мовних сигналів з використанням нейронних мереж.

Отримані результати практичних значень полягають у можливості використання програмного комплексу у повсякденному житті, як окремого додатку для покращення показника сигнал/шум, і покращення якості звуку.

РОЗДІЛ 1.
ТЕОРЕТИЧНИЙ АНАЛІЗ ПРОБЛЕМИ ШУМІВ У ТЕЛЕКОМУНІКАЦІЙНИХ МЕРЕЖАХ

1.1. Основні причини виникнення шумів

Телекомунікаційна мережа передає повідомлення між приймачами та каналами зв'язку, передавачами. Цифрові системи зв'язку складаються з маршрутизаторів, які спільно працюють, щоб передати користувачеві необхідну інформацію[1]. Аналогова система зв'язку складається з одного або декількох комутаторів, які встановлюють зв'язок між двома або більше користувачами. Для обох типів мереж можуть знадобитися репітери для посилення або відтворення сигналу, коли він передається на великі відстані. Репітери використовуються для боротьби з загасаннями сигналу, коли сигнал неможливо відрізнити від шуму. Додатковою перевагою цифрових систем над аналоговими є простіше зберігання на пристрої. Тобто два значення напруги (високе і низьке) легше зберігати, ніж безперервний діапазон значень у аналогових системах [2].

Корисні сигнали рідко присутні в електричних ланцюгах в чистому вигляді. Практично завжди на них накладаються шуми і перешкоди. При цьому корисний сигнал спотворюється при передачі, і повідомлення відтворюється з деякою помилкою. Причиною помилок є як спотворення, що вносяться самим каналом, так і різного виду перешкоди, що впливають на сигнал при передачі. У власне пристроях каналу передачі інформації є два основних джерела шумів: дискретна структура струму в підсилювальних елементах (транзисторах, мікросхемах і т.д.) і тепловий рух вільних електронів в провідниках електричного кола. При цьому тимчасові і частотні характеристики каналу визначають лінійні спотворення. Крім того, радіоканал може вносити і нелінійні

спотворення, обумовлені нелінійністю тих чи інших його ланок, ланцюгів або пристроїв.

Процес зниження шуму (Noise reduction) полягає у видаленні некорисних звуків з сигналу для підвищення його суб'єктивної якості. Системи шумозниження (СШП) є засобом обробки сигналу, які допомагають збільшити співвідношення сигнал/шум. Вони широко використовуються у телекомунікаціях для обробки аудіо-, відео- та фото-сигналів. У електронних системах звукозапису, основним джерелом шуму є теплові процеси, які впливають на рух електронів. Ці випадкові теплові зміни підсилюються разом з корисним сигналом і при відтворенні сприймаються як шум. [3].

Для покращення якості звучання в системах запису та передачі звуку застосовується передкорекція звукового сигналу з використанням компандерів. Компандерні системи шумозниження використовують попереднє компресію сигналу при передачі, тобто стиснення динамічного діапазону, а при прийманні (відтворенні) отриманий сигнал експандується, тобто розширюється динамічний діапазон. Це дозволяє зменшити рівень завад і шумів, що проникають в канал передачі (запису). Системи шумозниження з компандерами широко використовуються в телекомунікаціях для обробки звукового сигналу.

Системи обробки сигналів мають дві сторони: передавальну і приймальну. Тому, систему шумозниження, яка працює в обох напрямках, називають "двосторонньою". Існує другий тип алгоритмів, який передбачає покращення якості наявного матеріалу, де обробка сигналу відбувається лише при його відтворенні. Такі системи шумозниження називаються "односторонніми" за прийнятою термінологією (від англ. single-ended).

Простий метод боротьби з шумом - гейт (або пороговий обмежувач шуму) - заблоковує проходження сигналів в паузах звукозапису. Він працює як перемикач, який повністю пропускає вхідний сигнал на вихід або повністю його пригнічує. У сучасних моделях можна задати поріг спрацювання, який визначає мінімальний рівень сигналу, щоб він пройшов[4]. Проте цей метод не

ефективний для зменшення шуму в тихих частинах запису, де рівень шуму залишається високим і помітним на слух, або такі частини можуть бути заглушені повністю.

1.2. Вплив шумів на якість передачі сигналу

Шум може впливати на якість передачі сигналу, оскільки він може призвести до додавання небажаних змін до сигналу, коли передається через канал зв’язку. Наприклад, коли сигнал передається по радіо частоті або по електричному проводу, шум може змінювати амплітуду сигналу або зміщувати фазу сигналу, що призводить до спотворення сигналу[4].

Це може призвести до неправильного декодування сигналу на приймачі та спотворення відправлених даних, або зниження якості аудіо та відео передачі. Шум також може вплинути на сигнал-шумове співвідношення, яке вимірює відношення потужності сигналу до потужності шуму на приймачі, і зниження його значення може погіршити якість передачі сигналу.

Для зменшення впливу шуму на якість передачі сигналу використовуються різні методи і техніки. Деякі з них виключають в себе:

- Фільтрацію шуму за допомогою аналогових або цифрових фільтрів, що дозволяє відфільтрувати небажані складові сигналу, що можуть бути викликані шумом.
- Використання технологій кодування, таких як корекція помилок та компресія даних, зменшуючи вплив шуму.
- Застосування алгоритмів, таких як адаптивна еквалізація, яка дозволяє компенсувати ефект шуму на передаваний сигнал, покращуючи якість передачі.
- Використання каналів зв’язку з більшою потужністю сигналу, що дозволяє зменшити вплив шуму на якість передачі.

Методи зменшення шуму на самому джерелі передачі, наприклад: шумознижуючі мікрофони та фільтри шуму в електронних пристроях.

Врахування впливу шуму на якість передачі сигналу є важливим в багатьох сферах, включаючи телекомунікації, радіозв'язок, аудіо та відео передачі, а також в областях, де використовується передача інформації з високою якістю та надійністю.

Співвідношення сигналу до шуму (ССШ або ВСШ, англ. SNR або S/N, Signal-to-noise ratio) - це метрика, яка використовується в науці та інженерії для вимірювання того, наскільки сильно сигнал спотворений шумом. Визначається як відношення потужності корисного сигналу до потужності шуму [5]. Значення ССШ більше 1:1 вказує на те, що сигнал більший за шум. Хоча SNR зазвичай застосовується до електричних сигналів, його можна використовувати для будь-яких видів сигналів.

Співвідношення сигнал/шум використовується для визначення того, наскільки великий сигнал порівняно з фоновим шумом. Ця міра визначається як відношення потужності сигналу, що містить значущу інформацію, до потужності фонового шуму, який не є бажаним сигналом.

$$SNR = \frac{P_{signal}}{P_{noise}},$$

Середню потужність позначають як P. Сигнал та шум повинні бути виміряні в одній або еквівалентній точці в системі та в межах однієї й тієї ж смуги пропускання системи.

1.3. Сучасні методи боротьби з шумами в телекомунікаційних мережах

Шум може бути будь-якими несприятливими впливами, що спотворюють передачу або отримання інформації. Зазвичай він виникає з технічних або електричних джерел. Для зменшення впливу шуму використовуються різноманітні фільтри, шумогасники та кодування (наприклад, кодування Клода

Шеннона). Кодування, які базуються на надмірній передачі інформації або розбитті її на частини з контрольними точками, є ефективними способами боротьби з шумом. Такі коди можуть нагадувати систему розмови військових по рації, де застосовуються контрольні точки, такі як "прийом" та "розумію".

Пропускна здатність каналу передачі даних повинна бути не тільки достатньою для забезпечення надійності, але й достатньою для забезпечення швидкості передачі. Це особливо важливо як для великих інформаційних систем з багатьма пристроями, так і для менших, наприклад, у випадку Wi-Fi точки доступу в кафе з максимальною швидкістю передачі даних 54 Мбіт/с. Якщо в кафе знаходиться 30 осіб з ноутбуками, то кожна з них матиме обмежену швидкість передачі даних.

Основна функція витої пари полягає у забезпеченні рівномірного розподілення зовнішніх шумів [3]. Паралельні провідники, які не є скрученими витками, мають різницю напруги 2 В. Але, якщо один з них опиниться в зоні електромагнітного поля, що генерує додаткову напругу 9 В, а інший - в зоні поля, що генерує додаткову напругу 5 В, то різниця напруг буде становити не 2 В, а 6 В, як показано на рисунку 1.1.

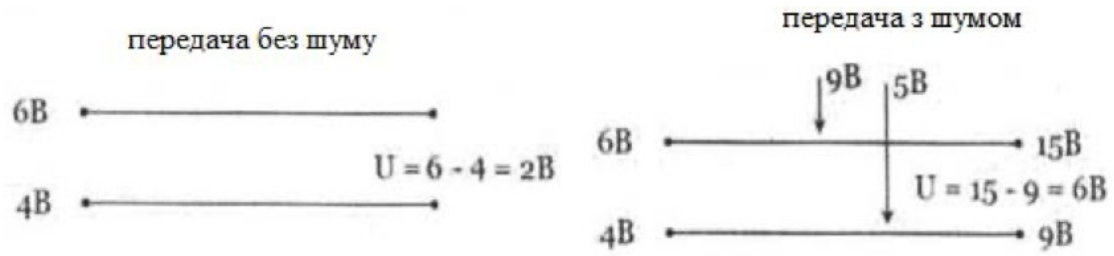

Рис.1.1. Шуми у не витій парі

На рис. 1.1 зображено розраховану передачу сигналу та безпосередній вплив шумів, наявних навколо нас. Оскільки попередити їх наявність, без попередніх дослідів, не є можливим, то почали використовувати виту пару, на яку шуми діють з однаковою інтенсивністю, тим самим нівелюються.

Також шуми можуть бути зменшені за допомогою таких методів:

1. Використання фільтрів: Фільтри можуть використовуватися для зменшення шумів, зокрема високочастотних шумів. Фільтри можуть бути реалізовані як окремий пристрій або як частина більш складної системи. Наприклад, у мобільних телефонах фільтри використовуються для зменшення шумів, що виникають внаслідок інтерференції з сигналами інших мобільних телефонів та інших джерел.

2. Використання кодування сигналу: Кодування сигналу може допомогти зменшити вплив шуму на якість передачі. Наприклад, у цифрових телевізійних системах використовуються кодеки, які забезпечують ефективне кодування сигналу та зменшення впливу шуму на якість передачі.

3. Використання алгоритмів шумозниження: Ці алгоритми допомагають відокремити корисний сигнал від шуму, що дозволяє підвищити якість передачі. Ці алгоритми можуть використовуватися у різних системах, таких як телефонія, аудіо- та відеозапис, медичне обладнання тощо.

4. Використання антен: Якщо шуми виникають на шляху передачі сигналу, використання антен може допомогти зменшити їх вплив. Наприклад, у мобільних телефонах використовуються різні типи антен, які дозволяють забезпечити кращу якість зв'язку в умовах шумів та інтерференції.

5. Використання шумозаглушення: Шумозаглушення - це процес, за допомогою якого шуми намагаються знищити або зменшити, створивши протилежний сигнал. Це може бути використано в системах звукового і відеообладнання, наприклад, в навушниках зі шумозаглушенням, щоб знизити рівень шуму навколишнього середовища, і у системах відеоконференцій для зменшення шумів на місці зйомки.

6. Використання технологій управління шумами: В деяких випадках шуми можуть бути управляні для зменшення їх впливу на передачу сигналу. Наприклад, в телевізійних мережах можуть використовуватися технології

управління шумами, які дозволяють змінювати рівень шумів в різних частинах мережі, щоб підвищити якість передачі.

Сучасний метод боротьби з шумами в телекомунікаційних мережах є використання нейронних мереж. Нейронні мережі - це математичні моделі, що імітують структуру та функцію нейронів в мозку[5]. Вони використовуються в телекомунікаційних мережах для покращення якості сигналу та боротьби з шумами. Нейронні мережі використовуються в різних аспектах телекомунікаційних мереж, наприклад, для фільтрації шумів в аудіо- та відеосигналах, для зменшення інтерференції в радіозв'язку, для покращення якості передачі даних та інших застосувань. Одним з методів використання нейронних мереж для боротьби з шумами є навчання мережі на основі набору даних з шумом та без шуму. Після навчання мережі можна використовувати її для фільтрації шумів в реальному часі.

РОЗДІЛ 2.
ОГЛЯД ТЕХНОЛОГІЙ МАШИННОГО НАВЧАННЯ ДЛЯ ЗМЕНШЕННЯ РІВНЯ ШУМІВ

2.1. Опис алгоритмів машинного навчання для розв’язання задачі відновлення сигналу

Для розв'язання задачі відновлення сигналу, яка полягає в відновленні чистого сигналу зі спотвореним сигналом, можуть використовуватися різні алгоритми машинного навчання, такі як: нейронні мережі, метод глибокого навчання, метод генеративно-протилежних мереж, автоенкодер[6], метод опорних векторів.

Нейронні мережі зі зворотним поширенням. Це один з найбільш поширених підходів до розв'язання задач відновлення сигналу. У цьому підході використовуються нейронні мережі, які навчаються на основі пар чистого та спотвореного сигналів. Після навчання мережа може використовуватися для відновлення чистого сигналу зі спотвореним.

Позитивні сторони:

- Нейронні мережі можуть працювати з різними типами сигналів, включаючи аналогові, цифрові, зображення та звук.
- Нейронні мережі можуть розв'язувати складні задачі, такі як відновлення великих масивів даних, що складаються з багатьох складових частин.
- Нейронні мережі можуть працювати з низькоякісними сигналами, що містять шуми та інші артефакти.

Негативні сторони:

- Нейронні мережі можуть бути вкрай чутливими до параметрів, таких як розмір та форма вхідного сигналу, що може впливати на їх ефективність.

- Нейронні мережі вимагають великої кількості даних для навчання, щоб досягти оптимальної ефективності.

Метод глибокого навчання. Цей метод полягає в навчанні нейронних мереж з багатьма шарами. У цьому випадку, кожен шар мережі навчається для вирішення конкретного аспекту задачі відновлення сигналу. Після навчання всіх шарів, мережа може використовуватися для відновлення чистого сигналу.

Позитивні сторони:

- Метод глибокого навчання може розв'язувати складні задачі, які не вдається вирішити звичайними методами машинного навчання.
- Метод глибокого навчання може автоматично відбирати риси та функції, що потрібні для відновлення сигналу.
- Метод глибокого навчання може працювати з різними типами сигналів.

Негативні сторони:

- Метод глибокого навчання вимагає великої кількості даних для навчання, щоб досягти оптимальної ефективності.
- Метод глибокого навчання може бути чутливим до параметрів, таких як розмір та форма вхідного сигналу.
- Метод глибокого навчання може перенавчитись на тренувальних даних, що призведе до зниження точності відновлення на нових даних.

Метод генеративно-протилежних мереж (GAN). У цьому методі використовуються дві мережі: генератор та дискримінатор[7]. Генератор навчається на основі пар чистого та спотвореного сигналів для генерування нових, чистих сигналів. Дискримінатор навчається на основі пар чистого та згенерованого генератором сигналів для розрізнення між чистим та згенерованим сигналом. Після навчання, генератор може використовуватися для відновлення чистого сигналу зі спотвореним[8].

Позитивні сторони:

- Можливість генерації нових сигналів на основі навчальних даних.

- Можливість ефективного вирішення задач відновлення сигналу в режимі реального часу.

Негативні сторони:

- Можливість перенавчання на тренувальних даних, що призводить до зниження точності відновлення на нових даних.
- Можливість генерації некоректних сигналів на основі збурених даних.

Автоенкодери (autoencoders) - це метод машинного навчання, який використовується для відновлення сигналу з зашумленого вхідного сигналу. Основна ідея полягає в тому, що автоенкодер навчається відтворювати вхідний сигнал вихідним сигналом, при цьому використовується зменшення кількості параметрів в процесі кодування сигналу в зв'язку з чим у процесі навчання модель намагається знайти найбільш важливі ознаки вхідного сигналу[9].

Позитивні сторони використання автоенкодерів для розв'язання задачі відновлення сигналу:

- Висока точність відновлення сигналу при невеликому рівні шуму.
- Можливість роботи з різноманітними типами сигналів, такими як звук, зображення, текст.
- Можливість використання великої кількості даних для навчання моделі.
- Зменшення розмірності даних, що полегшує їх обробку.
- Покращення ефективності обробки даних шляхом використання більш високорівневих представлень даних.
- Автоматичне вивчення корисних ознак даних, що полегшує їх аналіз і інтерпретацію.

Негативні сторони використання автоенкодерів для розв'язання задачі відновлення сигналу:

- Недостатня точність відновлення сигналу при великому рівні шуму.
- Потреба у великій кількості даних для навчання моделі.

- Складність інтерпретації результатів автокодувальника. Інколи може бути складно зрозуміти, які саме ознаки були вивчені моделлю.
- Потреба у великій кількості часу для навчання моделі[10].

Метод опорних векторів (Support Vector Machines, SVM) - це метод машинного навчання, який використовується для розв'язання задач класифікації та регресії, а також для відновлення сигналу. Метод SVM використовується для пошуку гіперплощини, яка максимально розділяє два класи[11].

Позитивні сторони:

- Висока точність: Метод опорних векторів зазвичай дозволяє досягти високої точності відновлення сигналу. Використання опорних векторів дозволяє зменшити кількість шуму в відновленому сигналі та підвищити якість сигналу.
- Широкий діапазон застосування: Метод опорних векторів може бути застосований для відновлення різних типів сигналів, таких як зображення, звук, відео та інші. Це робить його універсальним методом для відновлення сигналу.
- Різноманітність ядер: Метод опорних векторів дозволяє використовувати різноманітність ядер, що дозволяє пристосовувати метод до різних типів сигналів та отримувати кращі результати відновлення.
- Ефективність при обробці великих об'ємів даних: Метод опорних векторів може бути ефективним при обробці великих об'ємів даних. Він дозволяє швидко обчислювати велику кількість параметрів та зменшує час обробки даних.
- Можливість здійснювати прогнозування: Метод опорних векторів може бути використаний для прогнозування майбутніх значень сигналу на основі його попередніх значень. Це дозволяє використовувати метод для прогнозування різних явищ на основі даних з сигналів.

Негативні сторони:

- Високі вимоги до обчислювальних ресурсів: метод опорних векторів вимагає значних обчислювальних ресурсів, особливо для обробки великих обсягів даних.
- Чутливість до викидів: метод опорних векторів чутливий до викидів, тобто випадкових значень, які можуть змінити форму розподілу даних і знизити якість моделі.
- Вибір параметрів: метод опорних векторів вимагає вибору оптимальних параметрів, таких як тип ядра, параметри ядра та параметри регуляризації. Неправильний вибір параметрів може призвести до перенавчання або недонавчання моделі.
- Нездатність до роботи з багатовимірними даними: метод опорних векторів може бути нездатним до роботи з багатовимірними даними, такими як зображення або відео, через обчислювальну складність та чутливість до викидів.
- Ємність до переобчислення: метод опорних векторів може бути схильним до переобчислення, особливо при використанні неправильного типу ядра або параметрів регуляризації.

Основні характеристики наведено у таблиці 2.1.

Таблиця 2.1.

Метод / Алгоритм	Опис	Переваги	Недоліки
Нейронні мережі	Модель, що складається зі штучних нейронів, яка може навчатися на великих даних та розв'язувати завдання відновлення сигналу	Висока точність, можливість роботи з великими обсягами даних, здатність до автоматичної адаптації до нових даних	Вимагає великої кількості даних для навчання, складний процес настройки параметрів
Метод глибокого навчання	Вид нейронної мережі з глибокою архітектурою, що дозволяє розв'язувати	Висока точність, здатність до автоматичної адаптації до нових даних,	Вимагає великої кількості даних для навчання, складний

	завдання відновлення сигналу за допомогою низки шарів	можливість використовувати великі обсяги даних	процес настройки параметрів
Метод генеративно-протилежних мереж	Архітектура нейронної мережі, яка складається зі двох моделей: генератора та дискримінатора, що дозволяє генерувати нові сигнали та розв'язувати завдання відновлення сигналу	Можливість генерації нових сигналів, висока точність відновлення сигналу	Складність настройки параметрів, вимога до великої кількості даних для навчання
Автоенкодер	Модель нейронної мережі, що складається зі штучних нейронів, яка може навчатися на великих даних та розв'язувати завдання відновлення сигналу	Висока точність, можливість роботи з великими обсягами даних, здатність до автоматичної адаптації до нових даних	Вимагає великої кількості даних для навчання, складний процес настройки параметрів, низька стійкість до шумів
Метод опорних векторів	Алгоритм машинного навчання, що дозволяє розв'язувати завдання класифікації та регресії для векторних даних	Швидкодія, здатність до роботи з великими обсягами даних	Не підходить для розв'язування складних завдань відновлення сигналу, вимагає попередньої обробки даних

У таблиці 2.1 було розглянуто основні методи та алгоритми машинного навчання для розв’язання задачі відновлення сигналу, такі як нейронні мережі, метод глибокого навчання, метод генеративно-протилежних мереж, автоекодер, метод опорних векторів. Незалежно від підходу використання методів є дві основні проблеми з подавлення шуму:

1. Обробка аудіо послідовностей – послідовно із змінною довжиною.
2. Повільний час обробки, що приводить до затримок[12].

Через переваги методу автокодування або ж автоенкодеру у програмі ми використовуватимемо саме цей метод для роботи із сигналами у телекомунікаційних мережах .

2.2. Архітектура нейронних мереж

Напрямок розвитку штучного інтелекту, який називається нейронними мережами, спрямований на моделювання аналітичних механізмів, що відбуваються в людському мозку. Типова нейронна мережа може вирішувати завдання класифікації, передбачення та розпізнавання[13]. Ці мережі можуть навчатися і розвиватися самостійно, використовуючи власний досвід та здобутки на основі помилок.

Основою штучної нейронної мережі є штучні нейрони, які утворюють нейронну мережу завдяки зв'язкам між ними. Штучна нейронна мережа дозволяє ефективно обробляти інформацію та пристосовуватись до змін у вхідних даних або очікуваному результаті. Це досягається завдяки здатності нейронної мережі навчатись і адаптуватись до різних умов, що виникають у процесі її роботи.

Під час роботи штучної нейронної мережі відбувається обробка вхідних даних, що може включати як перетворення самого вектора вхідних даних, так і генерацію відповідного вихідного сигналу. Цей процес обчислень повністю залежить від вхідних даних та очікуваного результату від мережі, що в свою чергу визначає архітектуру мережі, її структуру і методи, використовувані для тренування та валідації. Ці параметри в свою чергу визначають характеристики нейронів мережі, які грають важливу роль у її роботі. Отже, розглянемо основні характеристики нейронної мережі:

- Штучні нейронні мережі можуть бути створені на основі типу зв'язків між нейронами, зокрема на основі прямого поширення, коли немає циклів і сигнал поширюється тільки вперед, або на основі рекурентного типу, який має зворотній зв'язок для корегування ваг нейронів.

- Кількість прихованих шарів у штучній нейронній мережі може бути різною і визначається архітектурою мережі. В залежності від кількості прихованих шарів, штучні нейронні мережі можуть бути класифіковані як одношарові з одним прихованим шаром або багатошарові з декількома прихованими шарами.

- Мережі з фіксованими вагами та мережі зі змінними вагами є двома різними типами штучних нейронних мереж.

- Статичні та динамічні нейронні мережі відрізняються за наявністю елементу пам'яті, тобто наявністю зворотнього зв'язку для корегування ваг нейронів.

Також, нейронні мережі різняться за їх парадигмою, яка визначає спосіб використання та навчання мережі. На даний момент відомі три основні парадигми навчання нейронних мереж: навчання з вчителем, навчання без вчителя та навчання з підкріпленням[14].

Нейронні мережі поділяються за їх структурою, тобто за тим, як нейрони пов'язані та взаємодіють між собою. Архітектура нейронної мережі включає в себе різні характеристики мережі, такі як типи нейронів та їх взаємодія. Важливо зауважити, що на одній і тій же архітектурі нейронної мережі можуть бути використані різні парадигми і навпаки.

Нейронні мережі можна класифікувати за різними критеріями, одним з яких є їх архітектура[15]. До основних архітектурних рішень нейронних мереж можна віднести персептрон і нейронні мережі прямого поширення.

Персептрон - це проста форма нейронної мережі, яка складається з одного або декількох шарів нейронів. Шари в персептроні повністю пов’язані між собою, тобто кожен нейрон з одного шару пов’язаний з кожним нейроном наступного шару.

Нейронні мережі прямого поширення - це найбільш поширені типи нейронних мереж, у яких відсутні цикли та всі нейрони повністю пов’язані між собою. Такі мережі мають вхідний шар, прихований шар або декілька таких шарів та вихідний шар. Кількість шарів може бути різною, але зазвичай використовується один або декілька прихованих шарів.

Важливо пам’ятати, що на одній і тій самій архітектурі нейронної мережі можуть бути використані різні підходи та парадигми. Наприклад, нейронні мережі прямого поширення можуть використовуватися з різними функціями активації та різними методами оптимізації.

Мережі прямого поширення з багатьма прихованими шарами називають глибокими мережами. Раніше вважалося, що навчання глибоких мереж неможливе через експоненційний ріст часу навчання зі збільшенням кількості шарів. Однак, зараз завдяки сучасним методам навчання можна використовувати мережі з більшою кількістю прихованих шарів та отримувати результати кращі, ніж у мереж з одним прихованим шаром. Проте для такого навчання необхідно використовувати більше ресурсів.

Рекурентні нейронні мережі використовують повторювані клітини, які дозволяють враховувати контекст інформації з попередніх ітерацій або вибірок при прийнятті рішення в поточному стані. Основна рекурентна мережа - мережа Йордана, в якій кожна прихована клітина має фіксовану затримку на отримання входу з попередньої ітерації. Такі мережі базуються на звичайних FNN, але мають додаткові зв'язки, які дозволяють зберігати попередні стани та використовувати їх у поточному рішенні[16].

Крім описаних вище архітектур, існує багато інших типів нейронних мереж, які використовуються в залежності від конкретної задачі. Наприклад, LSTM мережі використовуються для обробки послідовностей даних з тривалими залежностями, мережі Хопфілда використовуються для збереження та відновлення паттернів даних, а CNN використовуються для обробки зображень

та відео. Кожна з цих архітектур має свої переваги та обмеження, тому вибір конкретної залежить від завдання, яке потрібно вирішити.

Схема основних мереж зображено на рис. 2.1.

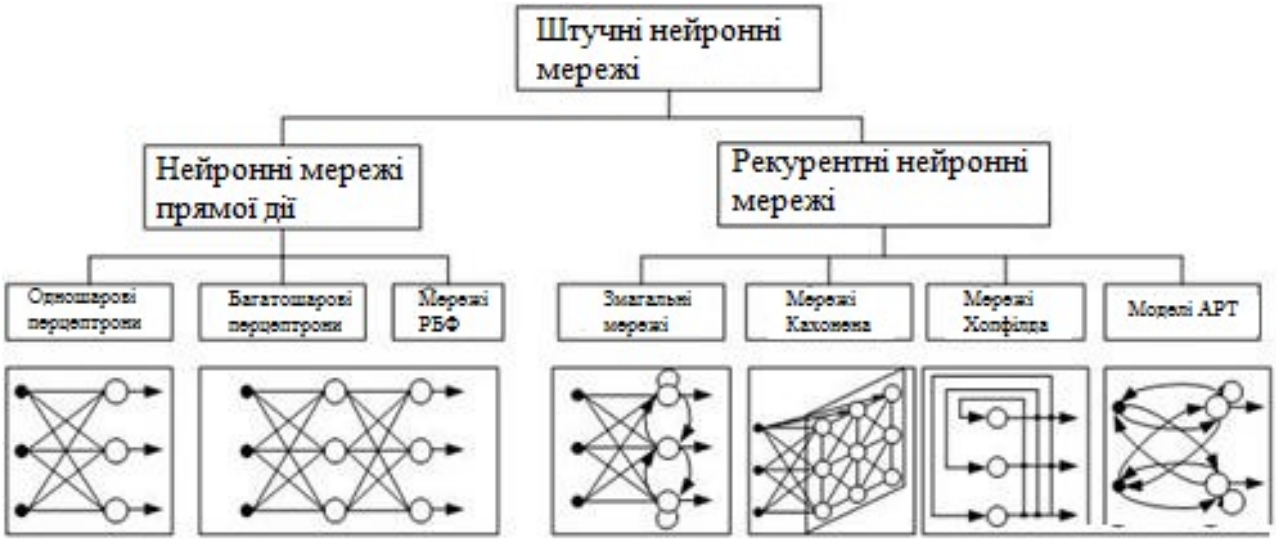

Рис. 2.1. Типи нейронних мереж

На рис. 2.1 представлено типи нейронних мереж. Згідно з попереднім визначенням, перший шар нейронної мережі є вхідним і отримує вектори тренувальних та тестових даних. Останній шар називається вихідним, оскільки повертає результати обчислень, проведених нейронною мережею. Шар, що розташований посередині, є прихованим, і ми також вже його описували раніше. Він отримав свою назву через те, що значення нейронів у цьому шарі не спостерігаються під час навчання мережі. Отже, в даній мережі ми маємо шар входу, прихований шар з нейронами та вихідний шар з нейронами.

2.3. Призначення нейромережі

Нейронні мережі зазвичай використовуються для вирішення складних завдань, які потребують аналітичних обчислень, подібних до тих, які виконує людський мозок [17]. Найпоширеніші застосування нейронних мереж включають:

-Класифікація - це процес поділу даних на різні групи за певними параметрами. Наприклад, можна використовувати нейронну мережу, щоб розподілити клієнтів за їхніми покупками та іншими характеристиками для того, щоб визначити, які товари вони будуть купувати в майбутньому.

-Прогнозування - це здатність нейронних мереж передбачати майбутній стан даних. Наприклад, можна використовувати нейронну мережу, щоб передбачити тенденції на фондовому ринку на основі аналізу попередніх змін цін акцій.

-Розпізнавання - це здатність нейронної мережі розпізнавати та ідентифікувати об’єкти у вхідних даних, таких як зображення або голосові команди. Наприклад, можна використовувати нейронну мережу для розпізнавання облич людей на фотографіях, щоб допомогти в поліцейських розслідуваннях або на контролі на пунктах пропуску.

Область застосування штучних нейронних мереж з кожним роком дедалі більше розширюється, на сьогоднішній день вони використовуються у таких сферах як:

- У робототехніці, нейронні мережі використовуються для створення складних алгоритмів для керування роботами з металевими "мозками".
- Машинне навчання є однією з гілок штучного інтелекту, яка базується на навчанні з прикладу за допомогою мільйонів схожих чи однотипних завдань. На сьогоднішній день машинне навчання широко використовується в пошукових системах, таких як Google, Bing, Baidu та Yahoo. Алгоритми цих систем вчаться за допомогою мільйонів пошукових запитів, які щодня вводяться користувачами, та намагаються демонструвати найбільш релевантні результати пошуку, щоб користувачі могли знайти необхідну інформацію, що відповідає їх критеріям.

- Нейронні мережі використовуються архітекторами комп'ютерних систем для вирішення проблем паралельних обчислень.
- Математики використовують нейронні мережі для вирішення складних математичних задач. • Комп'ютерне зорове розпізнавання використовується для розпізнавання образів, відеоаналітики та управління автономними автомобілями.
- Мовні технології використовуються для розпізнавання мови, перекладу текстів та генерації мовлення з використанням комп’ютерних програм і алгоритмів.
- У фінансах та інвестиціях застосовуються аналітичні технології для прогнозування змін на ринках, керування портфелем та виявлення шахрайства.
- В медицині використовуються технології обробки зображень, машинного навчання та інші інструменти для діагностики та прогнозування ризиків захворювань.
- У науці та матеріалах застосовуються різноманітні технології, включаючи моделювання, машинне навчання та інші методи, для дизайну нових матеріалів та прогнозування їх властивостей.
- Голосові помічники, такі як Siri та Alexa, засновані на технологіях розпізнавання мовлення та обробки природної мови і допомагають користувачам у повсякденних завданнях.
- У ігровій індустрії використовуються технології штучного інтелекту та машинного навчання для створення реалістичних персонажів та прогнозування їх рухів.

Мовні технології, фінанси та інвестиції, медицина, наука та матеріали, голосові помічники та ігрова індустрія - це лише кілька з безлічі сфер, які стали значно ефективнішими та продуктивнішими завдяки використанню штучного інтелекту. Нейронні мережі, машинне навчання та інші технології стали надзвичайно корисними для багатьох галузей, дозволяючи автоматизувати рутинні завдання та допомагаючи людям зосереджуватись на більш складних і

важливих завданнях. Крім того, штучний інтелект дозволяє збільшувати точність прогнозування та діагностики, а також знижувати ризики помилок. Ці технології стали невід'ємною частиною сучасного світу і, ймовірно, займуть все більш важливе місце в нашому майбутньому.

РОЗДІЛ 3.
РОЗРОБКА СИСТЕМИ ЗМЕНШЕННЯ ШУМІВ У МОВНИХ СИГНАЛАХ

3.1. Вибір моделі системи та класифікація

За час розвитку нейронних мереж було розроблено безліч різних типів, які використовуються в різних завданнях. На сьогодні складно класифікувати будь-яку мережу за однією ознакою. Це можливо зробити, виходячи з таких характеристик, як тип застосування, тип вхідної інформації, характер навчання, характер зв'язків та сферу застосування. Більш детальну інформацію можна знайти в таблиці 3.1.

Таблиця 3.1.

Нейронна мережа	Принцип застосування	Навчання	Сфера застосування
Персептрон	Класифікація	З вчителем	Розпізнавання образів
Зворотній розповсюдження помилки	Регресія	З вчителем	Прогнозування
Верхній шар-підписний класифікатор	Об’єднання навчання і без навчання	Змішане	Розпізнавання образів
Самоорганізуюча карта Кохонена	Кластеризація	Без вчителя	Компресія даних
Радіально-базисна функція	Регресія	З вчителем	Прогнозування
Рекурентна мережа	Обробка послідовностей	З вчителем	Машинний переклад

Світлова мережа	Оптичний пересувний обчислювач	Змішане	Оптичний інтерфейс

Два основних типи мереж, які є початковими моделями для багатьох нейромереж, - це згорткові та рекурентні мережі. Згорткові мережі є одним з найпопулярніших типів мереж і використовуються для розпізнавання інформації на фотографіях та відео, обробки мови та систем рекомендацій. Вони мають відмінну масштабованість, використовують об'ємні тривимірні нейрони та механізм просторової локалізації. Ідея створення такої складної системи нейромережі виникла під час вивчення зорової кори, відповідальної за обробку візуальної складової великих півкуль мозку. Згорткові мережі є невід'ємною частиною технологій глибокого навчання, відрізняються від перцептрону використанням обмеженої матриці ваг, що зсувається по шару, що обробляється, замість повнозв'язкової нейронної мережі.

- Рекурентні нейромережі - це тип мережі, де зв'язки між елементами можуть зберігати попередні стани мережі і використовувати їх для обробки поточної інформації. Це дає можливість рекурентним мережам працювати з послідовними даними, такими як мовлення, музика, текст або часові ряди. Рекурентні мережі є ефективним інструментом для аналізу послідовних даних, оскільки вони можуть враховувати контекст, що передує поточному елементу в послідовності.

Такі мережі використовуються в багатьох галузях, включаючи розпізнавання мови, машинний переклад, генерацію тексту, аналіз даних сенсорів, таких як акселерометри і гіроскопи, та управління роботами. Наприклад, у задачі розпізнавання мови, рекурентна мережа може використовувати попередні слова для кращого розуміння поточного слова. У

задачі розпізнавання рукописного тексту, рекурентна мережа може використовувати попередні символи для кращого розуміння поточного символу.

Рекурентні мережі мають безліч різних архітектур, включаючи мережі з одним шаром, як Хопфілда, та мережі з багатьма шарами, як Елмана та Джордана.

3.2. Згорткова нейронна мережа

Згорткова нейронна мережа (СНС, CNN) є спеціалізованою архітектурою, яка є основним інструментом для багатьох завдань штучного нейронного навчання, включаючи розпізнавання та класифікацію об'єктів, і зосереджена на ефективному розпізнаванні образів. Завдяки двомірній топології зображень, згорткова мережа виявляється більш точною у розпізнаванні об'єктів на зображеннях, ніж багатошаровий персептрон. Крім того, згорткові мережі є стійкими до невеликих зсувів, змін масштабу та поворотів об'єктів на вхідних зображеннях. Тому архітектури, що базуються на згорткових мережах, відомі своєю високою ефективністю та надійністю, і займають провідні позиції у змаганнях з розпізнавання образів[18].

Згорткова нейронна мережа (CNN) є найважливішим інструментом для розпізнавання та класифікації об’єктів на фотографіях, розпізнавання мови та вирішення завдань комп'ютерного зору. Існує безліч варіантів застосування CNN, такі як Deep Convolutional Neural Network (DCNN), Region-CNN (R-CNN), Fully Convolutional Neural Networks (FCNN) та Mask R-CNN[18]. Сьогодні CNN є лідером у галузі нейронних мереж, і можуть використовуватися для роботи з аудіо та будь-якими даними, які можна представити у вигляді матриць.

Згорткові мережі відрізняються від багатошарових персептронів тим, що вони спеціалізовані на роботі з зображеннями, тому вони можуть виявляти характеристики, що властиві саме цьому типу даних. У багатошарових персептронах кожна точка у векторі рівнозначна, незалежно від того, чи вона

розташована поруч або далеко від інших точок. Зображення ж мають локальну зв'язність, тобто точки, що мають значення, пов'язані між собою[22]. Наприклад, на зображенні особи очікується, що точки, які відповідають різним частинам обличчя, будуть знаходитися поруч. Таким чином, згорткові мережі є ефективнішими для обробки зображень.

У порівнянні з мережами прямого поширення, які працюють з даними у вигляді векторів, згорткові мережі оброблюють зображення у вигляді тензорів - це 3D масиви чисел, які складаються з матриць чисел. Згорткові нейронні мережі мають базові блоки, які можна збирати як конструктор, додаючи шар за шаром та отримуючи все більш потужні архітектури. Основними блоками згорткових мереж є згорткові шари, шари підвибірки (пулінгу), шари активації та пов'язані шари.

Згортковий шар нейронної мережі застосовує операцію згортки до виходів попереднього шару з використанням навчальних параметрів, які включають ваги ядра згортки та константний зсув (bias)[20]. У згортковому шарі може бути кілька згорток, і для кожної з них на виході формується відповідне зображення. Ядра згортки можуть бути тривимірними, і в такому випадку згортка тривимірного входу з тривимірним ядром відбувається аналогічно. Для того, щоб зберегти розмір зображення, використовується доповнення зображення (padding), яке додає пікселі до входів попереднього шару. Існують різні способи доповнення, такі як zero shift, border extension, mirror shift та cyclic shift. Зсув (stride) є ще одним параметром згорткового шару і визначає зрушення, з яким застосовується згортка до виходів попереднього шару.

- Зсув дозволяє зменшити розмір вихідного зображення, що може бути корисно при обробці великих зображень або при зменшенні кількості параметрів нейронної мережі. Однак, занадто великий зсув може призвести до втрати важливої інформації з вхідного зображення.

- Ще одним важливим параметром згорткового шару є кількість фільтрів, яка визначається перед навчанням моделі і не змінюється під час процесу навчання. Кожен фільтр є набором ваг ядра згортки та ваги зсуву, які навчаються за допомогою алгоритмів оптимізації.

- У згорткових шарах часто використовуються також функції активації, які надають нелінійність нейронній мережі. Популярними функціями активації є ReLU, Leaky ReLU, ELU та інші.

- Згорткові шари можуть бути поєднані з іншими типами шарів, такими як шари підсумовування (англ. pooling layers), що дозволяють зменшити розмір зображення, або звичайними повнозв’язними шарами (англ. fully-connected layers), що використовуються для класифікації зображень або для генерації нових зображень.

Загалом, згорткові нейронні мережі є потужним інструментом для обробки зображень і використовуються в багатьох сферах, таких як комп’ютерне зору, медицина, реклама та багато інших.

РОЗДІЛ 4.
МОДЕЛЮВАННЯ СИСТЕМИ ЗНИЖЕННЯ ШУМУ У МОВНИХ СИГНАЛАХ

4.1. Підготовка даних для навчання нейронної мережі

Важливим етапом навчання є підбір набору даних для навчання (датасет). У відкритому доступі є декілька наборів голосових даних. Для роботи обрано відкриту базу даних комбінації чистих та шумних аудіосемплів для навчання алгоритмів покращення мовлення за моделлю TTS единбурзького університету.

Даний датасет складений з чистої та шумної паралельних баз даних, що працюють на частоті 48кГц. У складеній базі присутні 109 носіїв англійської мови з різноманітними акцентами та зачитують по 400 речень кожний[14].

Мова датасету — англійська, розмір — 16 ГБ, 130 годин аудіо файлів перевірено та 130 годин повний об’єм датасету, ліцензія для використання — Creative Commons Attribution 4.0 International Public License формат аудіо файлів — WAV, а також знизу наведено статистику розподілу віку людей використаних голосів. Вибір англійської мови зумовлений тим, що датасет українською мовою занадто малий для досягнення бажаних результатів навчання нейронної мережі.

4.2. Розробка архітектури програмної реалізації системи

Для навчання нейронної мережі розроблена архітектура програми. На рис. 4.1 показано блок-схему алгоритму програми навчання нейронної мережі.

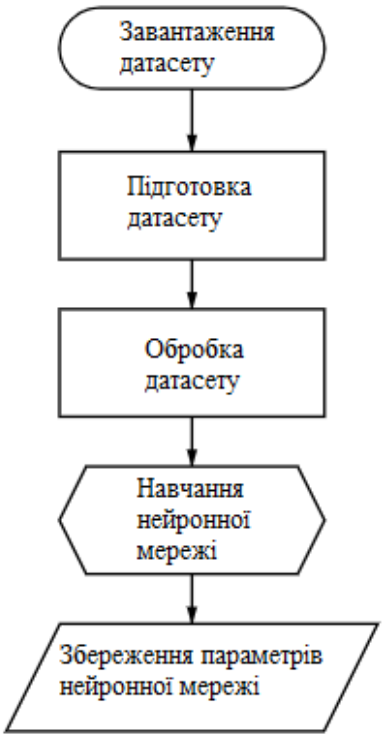

Рис. 4.1. Алгоритм програми навчання нейронної мережі

На рис. 4.1 показано: завантаження датасету (набір даних для навчання нейронної мережі) — завантаження попередньо підібраного датасету у програму. Підготовка датасету — вибір необхідних елементів датасету, поділ на датасет тренування та датасет валідації. Обробка датасету — компресія даних для збільшення швидкості навчання нейронної мережі. Навчання нейронної мережі — процес навчання нейронної мережі за допомогою наданого датасету та обраних параметрів навчання. Збереження даних — запис у файл параметрів нейронної мережі.

4.3. Опис розробленого коду

У програмі ми будемо використовувати тензорний потік, за допомогою бібліотеки PyTorch. Підключення бібліотек зображено на рисунку 4.2.

```
import tensorflow as tf
from tensorflow.keras.layers import Conv1D,Conv1DTranspose,Concatenate,Input
import numpy as np
import IPython.display
import glob
from tqdm.notebook import tqdm
import librosa.display
import matplotlib.pyplot as plt
import os
import math
```

Рис. 4.2. Підключення бібліотек

Дана частина є підключенням бібліотек до програми. Для взаємодії користувача та програмного засобу. Завантаження даних відбувається у частині, зображеній на рис. 4.3.

```
!mkdir "CleanData"
!mkdir "NoisyData"

!unzip "/content/drive/My Drive/clean_trainset_28spk_wav.zip" -d "CleanData"
!unzip "/content/drive/My Drive/noisy_trainset_28spk_wav.zip" -d "NoisyData"
```

Рис. 4.3. Завантаження тренувальних даних

Цей код створює два нові каталоги у поточній робочій директорії - "CleanData" та "NoisyData". Потім, за допомогою команди unzip, розпаковується вміст архівів, розташованих у відповідних шляхах "/content/drive/My Drive/clean_testset_wav.zip" та "/content/drive/My Drive/noisy_testset_wav.zip", у відповідні каталоги "CleanData" та "NoisyData".

На рис. 4.4 зображено демонстрацію завантаження даних до створених директорій.

```
Archive:  /content/drive/My Drive/clean_testset_wav.zip
   creating: CleanData/clean_testset_wav/
  inflating: CleanData/clean_testset_wav/p232_001.wav
  inflating: CleanData/clean_testset_wav/p232_002.wav
  inflating: CleanData/clean_testset_wav/p232_003.wav
  inflating: CleanData/clean_testset_wav/p232_005.wav
  inflating: CleanData/clean_testset_wav/p232_006.wav
  inflating: CleanData/clean_testset_wav/p232_007.wav
  inflating: CleanData/clean_testset_wav/p232_009.wav
  inflating: CleanData/clean_testset_wav/p232_010.wav
  inflating: CleanData/clean_testset_wav/p232_011.wav
  inflating: CleanData/clean_testset_wav/p232_012.wav
  inflating: CleanData/clean_testset_wav/p232_013.wav
  inflating: CleanData/clean_testset_wav/p232_014.wav
  inflating: CleanData/clean_testset_wav/p232_015.wav
  inflating: CleanData/clean_testset_wav/p232_016.wav
  inflating: CleanData/clean_testset_wav/p232_017.wav
  inflating: CleanData/clean_testset_wav/p232_019.wav
  inflating: CleanData/clean_testset_wav/p232_020.wav
  inflating: CleanData/clean_testset_wav/p232_021.wav
  inflating: CleanData/clean_testset_wav/p232_022.wav
  inflating: CleanData/clean_testset_wav/p232_023.wav
  inflating: CleanData/clean_testset_wav/p232_024.wav
  inflating: CleanData/clean_testset_wav/p232_025.wav
  inflating: CleanData/clean_testset_wav/p232_027.wav
  inflating: CleanData/clean_testset_wav/p232_028.wav
  inflating: CleanData/clean_testset_wav/p232_029.wav
  inflating: CleanData/clean_testset_wav/p232_030.wav
  inflating: CleanData/clean_testset_wav/p232_031.wav
  inflating: CleanData/clean_testset_wav/p232_032.wav
  inflating: CleanData/clean_testset_wav/p232_033.wav
  inflating: CleanData/clean_testset_wav/p232_034.wav
  inflating: CleanData/clean_testset_wav/p232_035.wav
  inflating: CleanData/clean_testset_wav/p232_036.wav
  inflating: CleanData/clean_testset_wav/p232_037.wav
  inflating: CleanData/clean_testset_wav/p232_038.wav
```

Рис. 4.4. Розпаковка архіву до директорій

Для завантаження даних ми будемо використовувати спеціальний модуль *tf.tensor.flow*, для збільшення швидкості завантаження даних на 50% використаємо *tf.audio()* разом з *tf.io.read_file()* замість звичною бібліотекою *librosa* і спеціальною функцією *librosa.load()*.

```
clean_sounds = glob.glob('/content/CleanData/clean_trainset_28spk_wav/*')
noisy_sounds = glob.glob('/content/NoisyData/noisy_trainset_28spk_wav/*')

clean_sounds_list,_ = tf.audio.decode_wav(tf.io.read_file(clean_sounds[0]),desired_channels=1)
for i in tqdm(clean_sounds[1:]):
  so,_ = tf.audio.decode_wav(tf.io.read_file(i),desired_channels=1)
  clean_sounds_list = tf.concat((clean_sounds_list,so),0)

noisy_sounds_list,_ = tf.audio.decode_wav(tf.io.read_file(noisy_sounds[0]),desired_channels=1)
for i in tqdm(noisy_sounds[1:]):
  so,_ = tf.audio.decode_wav(tf.io.read_file(i),desired_channels=1)
  noisy_sounds_list = tf.concat((noisy_sounds_list,so),0)

clean_sounds_list.shape,noisy_sounds_list.shape
```

Рис 4.5. Створення масивів даних

Використовуючи спеціальну функцію для завантаження окремих файлів аудіо *tf.audio.decode_wav()* та об'єднаємо їх, щоб отримати два тензори, масиви з іменами *clean_sounds_list* і noisy*_sounds_list.* Код зображено на рис. 4.5. та рис.4.6.

```
batching_size = 12000

clean_train,noisy_train = [],[]

for i in tqdm(range(0,clean_sounds_list.shape[0]-batching_size,batching_size)):
  clean_train.append(clean_sounds_list[i:i+batching_size])
  noisy_train.append(noisy_sounds_list[i:i+batching_size])

clean_train = tf.stack(clean_train)
noisy_train = tf.stack(noisy_train)

clean_train.shape,noisy_train.shape
```

Рис. 4.6. Завантаження даних у масив

Після завантаження ми виконуємо рівномірний розподіл однієї великої звукової хвилі. На рис. 4.7 зображено хвилю з чистим звуком.

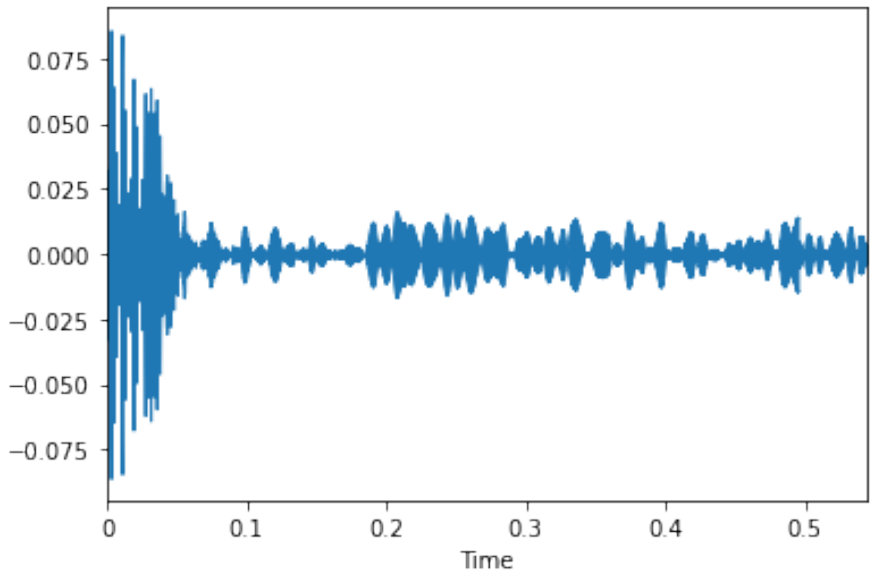

Рис. 4.7. Чистий звук

На рис. 4.7 зображено чистий звук, що отримали в результаті трансформації файлу розмірності WAV. На рис. 4.8 зображено ту ж саму доріжку, але з накладеним шумом, для навчання нейронної мережі.

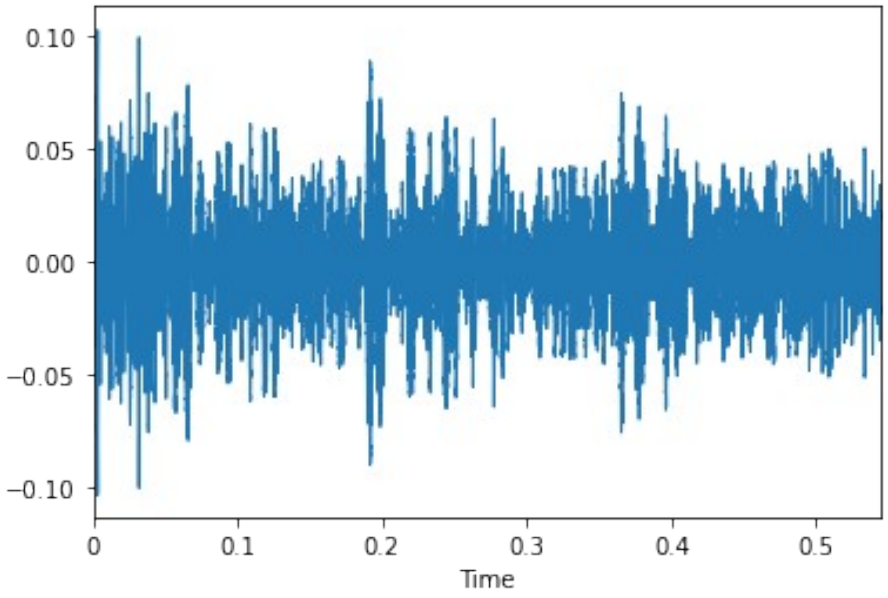

Рис. 4.8. Аудіо з накладеним шумом

На рис. 4.8 зображено аудіо доріжку з накладеним шумом. Візуалізацію виконували за допомогою модуля відображення *librosa*, що використовує

бібліотеку *matplotlib* для побудови даних. Із рис. 4.7 та рис. 4.8 видно шум, що може бути будь-якого походження від людей, автомобілів, тварин, заводів, чайнику і саме від них ми будемо позбавлятись за допомогою навчання нейронної мережі.

Створимо додаткову функцію для створення бази даних всередині програми Рис. 4.9.

```
def get_dataset(x_train,y_train):
  dataset = tf.data.Dataset.from_tensor_slices((x_train,y_train))
  dataset = dataset.shuffle(100).batch(64,drop_remainder=True)
  return dataset
```

```
train_dataset = get_dataset(noisy_train[:40000],clean_train[:40000])
test_dataset = get_dataset(noisy_train[4000:],clean_train[4000:])
```

Рис. 4.9. Функція бази даних

На рис. 4.9 частина коду, під назвою *get_dataset(),* описує створення бази даних для тренування з назвою tf.data.Dataset. В ній ми обираємо 40000 зразків для навчання, а останні залишаються для тестування.

4.4. Створення моделі нейронної мережі

Код створення моделі нейронної мережі буде зображено на рис. 4.10.

```
inp = Input(shape=(batching_size,1))
c1 = Conv1D(2,32,2,'same',activation='relu')(inp)
c2 = Conv1D(4,32,2,'same',activation='relu')(c1)
c3 = Conv1D(8,32,2,'same',activation='relu')(c2)
c4 = Conv1D(16,32,2,'same',activation='relu')(c3)
c5 = Conv1D(32,32,2,'same',activation='relu')(c4)

dc1 = Conv1DTranspose(32,32,1,padding='same')(c5)
conc = Concatenate()([c5,dc1])
dc2 = Conv1DTranspose(16,32,2,padding='same')(conc)
conc = Concatenate()([c4,dc2])
dc3 = Conv1DTranspose(8,32,2,padding='same')(conc)
conc = Concatenate()([c3,dc3])
dc4 = Conv1DTranspose(4,32,2,padding='same')(conc)
conc = Concatenate()([c2,dc4])
dc5 = Conv1DTranspose(2,32,2,padding='same')(conc)
conc = Concatenate()([c1,dc5])
dc6 = Conv1DTranspose(1,32,2,padding='same')(conc)
conc = Concatenate()([inp,dc6])
dc7 = Conv1DTranspose(1,32,1,padding='same',activation='linear')(conc)
model = tf.keras.models.Model(inp,dc7)
model.summary()
```

Рис. 4.10. Створення маоделі нейронної мережі.

Детальне пояснення коду на рис. 4.10 : дана частина коду створює модель автоенкодера, яка складається з енкодера та декодера. Автоенкодер - це нейронна мережа, яка намагається відновити вхідні дані на виході шляхом стиснення (енкодер) та розширення (декодер) даних. В даному випадку, модель отримує на вхід звукові сигнали, стискає їх в більш компактний вектор (енкодер), а потім розширює знову у звуковий сигнал (декодер).

Ця модель складається з 10 блоків, кожен з яких складається з 2 згорткових шарів (Conv1D) та 1 транспонованого шару згортки (Conv1DTranspose). Перші 5 блоків - енкодер, а наступні 5 блоків - декодер.

На вході моделі є 3D тензор форми (batching_size, time_steps, 1), де batching_size - розмір пакета (кількість зразків, які обробляються за один раз), а time_steps - кількість часових кроків у звуковому сигналі.

Кожен шар згортки має різні параметри, такі як кількість фільтрів, ядро згортки, крок зміщення та функцію активації. В цій моделі використовується функція активації "relu" для всіх шарів згортки, крім останнього транспонованого шару згортки, для якого використовується функція активації "linear".

Кожен транспонований згортковий шар повертає дані до початкових розмірів, або навіть більших, за рахунок збільшення розміру (upsampling) вхідних даних.

Після створення моделі, виконується її короткий опис (summary) за допомогою методу "summary()" моделі. У виводі опису показано кількість параметрів та форму кожного шару.

Далі в коді використовується функція Conv1DTranspose з бібліотеки Keras, яка реалізує 1D транспоновані згорткові шари (transpose convolutional layers) для роботи зі звуковими даними[19].

Ці шари допомагають відновлювати вихідні звукові сигнали з зашумлених зразків, і вони виконуються у зворотньому напрямку до звичайних згорткових шарів[17]. Таким чином, модель може вчитися усувати шум і відновлювати якість сигналу.

У даному коді реалізована мережа з 6 транспонованими згортковими шарами (Conv1DTranspose), які змінюють форму відповідно до форми згорткових шарів. Шари пропускають вхідний сигнал через кожен шар, збільшуючи розмір фільтра на кожному новому шарі, після чого відновлюють вихідний звуковий сигнал.

Крім того, у коді використовується функція Concatenate з бібліотеки Keras, яка допомагає об'єднувати вихідні сигнали з попередніх шарів з транспонованими сигналами. Це дозволяє відновлювати деталі звуку, які можуть бути втрачені під час обробки[19].

Загальна архітектура мережі є типовою для задачі відновлення сигналу[2], де застосовуються послідовні згорткові та транспоновані згорткові шари для

зменшення розміру сигналу та відновлення його до початкового розміру. Загальний вигляд моделі наведено на рис. 4.11.

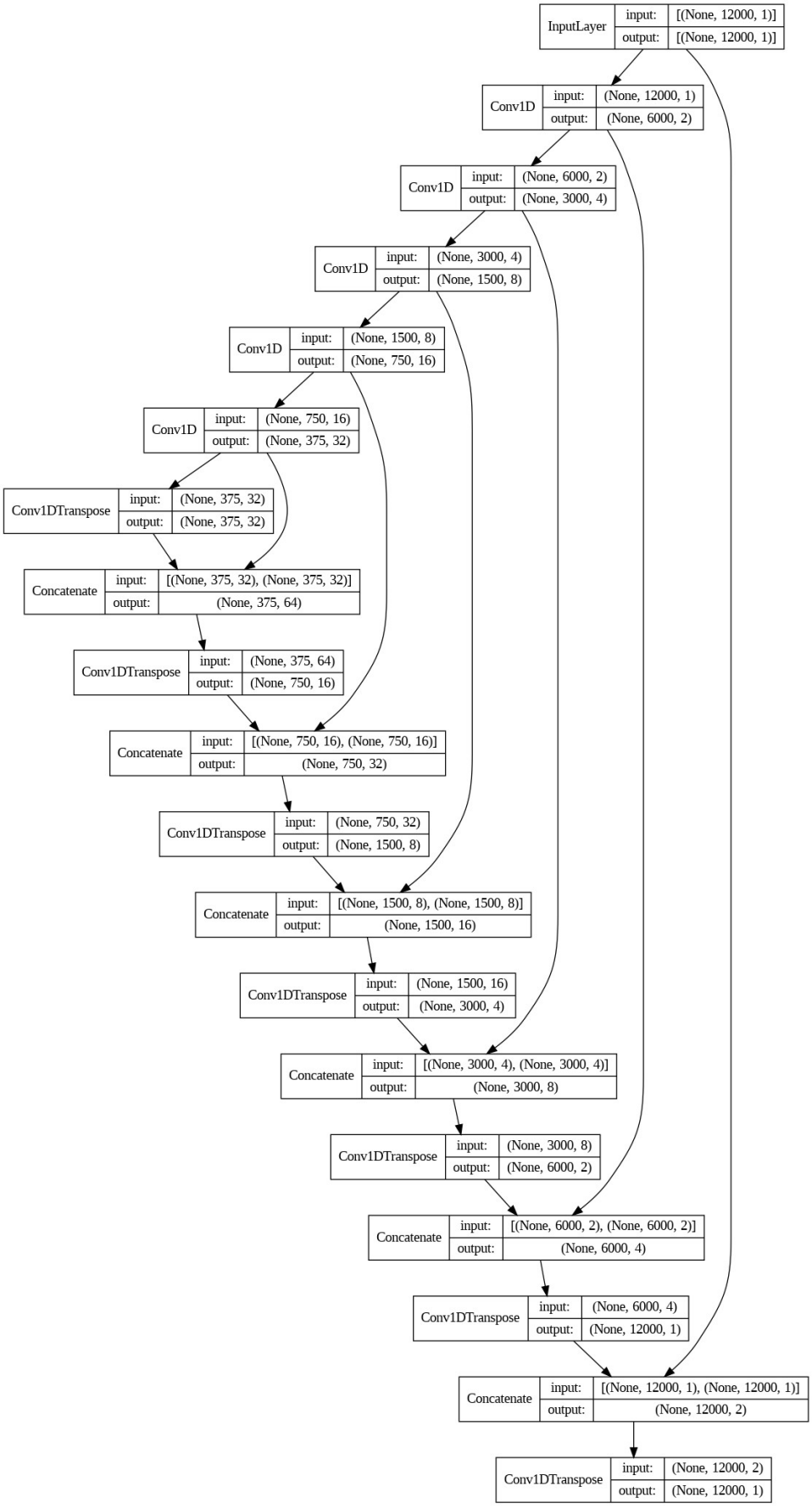

Рис. 4.11. Загальний вигляд моделі автоенкодера

4.5. Тренування нейронної мережі

Завантаження даних у тренувальну модель зображено на рис. 4.12.

```
model.compile(optimizer=tf.keras.optimizers.Adam(0.002),loss=tf.keras.losses.MeanAbsoluteError())
history = model.fit(train_dataset,epochs=20)
```

Рис. 4.12. Тренування нейронної мережі

Код, зображений на рис. 4.12, компілює навчальну модель, використовуючи оптимізатор Adam з швидкістю навчання 0.002 та середньою абсолютною похибкою як функцією втрат. Потім він навчає модель на тренувальному датасеті, що передається в якості вхідних даних через параметр train_dataset, протягом 20 епох. Під час навчання історія втрат зберігається в змінній history. На рис. 4.13 зображено основну інформацію.

```
Epoch 1/20
129/129 [==============================] - 20s 32ms/step - loss: 0.0156
Epoch 2/20
129/129 [==============================] - 4s 30ms/step - loss: 0.0106
Epoch 3/20
129/129 [==============================] - 4s 30ms/step - loss: 0.0095
Epoch 4/20
129/129 [==============================] - 4s 29ms/step - loss: 0.0090
Epoch 5/20
129/129 [==============================] - 4s 30ms/step - loss: 0.0086
Epoch 6/20
129/129 [==============================] - 4s 33ms/step - loss: 0.0084
Epoch 7/20
129/129 [==============================] - 4s 31ms/step - loss: 0.0083
Epoch 8/20
129/129 [==============================] - 4s 30ms/step - loss: 0.0082
Epoch 9/20
129/129 [==============================] - 4s 31ms/step - loss: 0.0080
Epoch 10/20
129/129 [==============================] - 4s 31ms/step - loss: 0.0079
Epoch 11/20
129/129 [==============================] - 4s 30ms/step - loss: 0.0079
Epoch 12/20
129/129 [==============================] - 4s 30ms/step - loss: 0.0078
Epoch 13/20
129/129 [==============================] - 4s 31ms/step - loss: 0.0077
Epoch 14/20
129/129 [==============================] - 4s 30ms/step - loss: 0.0076
Epoch 15/20
129/129 [==============================] - 4s 32ms/step - loss: 0.0076
Epoch 16/20
129/129 [==============================] - 4s 32ms/step - loss: 0.0076
Epoch 17/20
129/129 [==============================] - 4s 30ms/step - loss: 0.0076
Epoch 18/20
129/129 [==============================] - 4s 30ms/step - loss: 0.0076
Epoch 19/20
129/129 [==============================] - 4s 32ms/step - loss: 0.0075
Epoch 20/20
129/129 [==============================] - 4s 30ms/step - loss: 0.0075
```

Рис. 4.13. Історія тренування та втрати

Код на рус. 4.14 виводить аудіо-сигнал, який зберігається у змінній noisy_train[22] в якості звукового файлу за допомогою вбудованої функції Audio з бібліотеки IPython.display. Для відтворення звуку необхідно мати динаміки або навушники підключені до пристрою, на якому виконується код.

```
from IPython.display import Audio
Audio(np.squeeze(noisy_train[22].numpy()),rate=16000)
```


```
Audio(tf.squeeze(model.predict(tf.expand_dims(tf.expand_dims(noisy_train[22],-1),0))),rate=16000)
```

```
1/1 [==============================] - 0s 408ms/step
```


```
model.evaluate(test_dataset)
```

```
67/67 [==============================] - 2s 16ms/step - loss: 0.0075
0.007474950980395079
```

Рис. 4.14. Вивід аудіо-сигналу

model.predict() отримує на вхід звуковий сигнал, що зберігається у noisy_train[22] і виконує передбачення, що видає очищений від шуму сигнал. tf.expand_dims(tf.expand_dims(noisy_train[22],-1),0) перетворює звуковий файл на тензор розмірності (1, довжина_сигналу, 1), який передається у model.predict(). Результат передбачення є згенерованим звуковим файлом, який містить очищений сигнал від шуму.

Функція tf.squeeze() використовується для видалення розмірності 1 з тензору, отриманого з результату model.predict(), щоб зменшити кількість зайнятого місця в пам'яті. Результат передбачення подається на вхід функції Audio, яка відтворює його.

Збереження результату навчання моделі відбувається за допомогою функції model.save у змінній під назвою “NoiseSuppressionModel.h5”.

Обробку аудіовходів різного розміру можна вирішити шляхом накладання кадрів передбачення та видалення частини перетину з кінцевої форми сигналу на рис. 4.15.

```
def get_audio(path):
  audio,_ = tf.audio.decode_wav(tf.io.read_file(path),1)
  return audio
```

```
def inference_preprocess(path):
  audio = get_audio(path)
  audio_len = audio.shape[0]
  batches = []
  for i in range(0,audio_len-batching_size,batching_size):
    batches.append(audio[i:i+batching_size])

  batches.append(audio[-batching_size:])
  diff = audio_len - (i + batching_size)
  return tf.stack(batches), diff
```

```
def predict(path):
  test_data,diff = inference_preprocess(path)
  predictions = model.predict(test_data)
  final_op = tf.reshape(predictions[:-1],((predictions.shape[0]-1)*predictions.shape[1],1))
  final_op = tf.concat((final_op,predictions[-1][-diff:]),axis=0)
  return final_op
```

Рис. 4.15. Обробка аудіо-входів

Цей код містить три функції, які виконують наступні дії:

1. Функція get_audio(path) отримує шлях до аудіофайлу, декодує його та повертає об'єкт audio.

2. Функція inference_preprocess(path) отримує шлях до аудіофайлу, викликає get_audio() для отримання об'єкта audio, ділить його на невеликі частини (пакети) розміром batching_size та повертає набір пакетів test_data та різницю diff між довжиною аудіофайлу та кількістю оброблених даних.

3. Функція predict(path) отримує шлях до аудіофайлу, викликає inference_preprocess() для підготовки вхідних даних, передає їх у модель для отримання передбачень та повертає об'єднані та перетворені передбачення з формату (batch_size, 1) на (N * batch_size, 1), де N - кількість отриманих пакетів,

що є повними. Далі останній, неповний пакет об'єднується з отриманим результатом.

Модель навчена працювати із вхідними даними різного розміру, що залежить від нашого розміру batching_size. Якщо нам надходить сигнал більшого розміру від змінної, то відбувається розподіл на декілька елементів, що кратна змінній batching_size, для перекриття остачі використаємо зворотній розподіл із кінця.

Опис створення екземпляру перекладача з розподілом тензорів для визову та отримання результатів зображено на рис. 4.16.

```
interpreter = tf.lite.Interpreter(model_path='TFLiteModel.tflite')
interpreter.allocate_tensors()
```

```
def predict_tflite(path):
  test_audio,diff = inference_preprocess(path)
  input_index = interpreter.get_input_details()[0]["index"]
  output_index = interpreter.get_output_details()[0]["index"]

  preds = []
  for i in test_audio:
    interpreter.set_tensor(input_index, tf.expand_dims(i,0))
    interpreter.invoke()
    predictions = interpreter.get_tensor(output_index)
    preds.append(predictions)

  predictions = tf.squeeze(tf.stack(preds,axis=1))
  final_op = tf.reshape(predictions[:-1],((predictions.shape[0]-1)*predictions.shape[1],1))
  final_op = tf.concat((tf.squeeze(final_op),predictions[-1][-diff:]),axis=0)
  return final_op
```

Рис. 4.16. Функція обробки сигналу

Даний код виконує прогнозування на моделі, яка була збережена в форматі TFLite (TensorFlow Lite). Спочатку ініціалізується об'єкт interpreter, що представляє TFLite інтерпретатор, а також виділяється пам'ять для тензорів. Після цього функція predict_tflite() приймає шлях до аудіофайлу та попередньо обробляє його за допомогою функції inference_preprocess(), яка розбиває аудіодоріжку на частини.

Функція використовує метод get_input_details() та get_output_details(), щоб отримати вхідні та вихідні деталі моделі. За допомогою методу set_tensor() та методу invoke() передбачаються дані в моделі, і результати виводяться за допомогою методу get_tensor().

На цьому етапі функція відповідає за прогнозування на частинках аудіофайлу, які були оброблені раніше, і виводить список з прогнозованими значеннями.

Нарешті, функція використовує функцію tf.squeeze() та tf.stack() для створення фінального виводу, який включає всі прогнози з оброблених частинок аудіофайлу, та використовує метод tf.concat() для з'єднання неповних останніх фреймів зі всім рештою виводу.

Даний код є функцією, яка приймає шлях до аудіофайлу, обробляє його та використовує модель TFLite, щоб зробити прогноз для кожного елемента аудіо та повертає остаточний результат у вигляді одномірного масиву.

Перевіримо швидкодію програмного засобу за допомогою коду на рис. 4.17.

```
%%timeit
predict_tflite(noisy_sounds[3])
```

Рис. 4.17. Перевірка швидкодії.

Дана частина коду виконує бенчмарк (тест продуктивності) функції predict_tflite() на вхідному звуковому файлі noisy_sounds[2] з допомогою магічного слова %%timeit. Вона повторює виконання цієї функції кілька разів, щоб визначити середній час виконання. Результат виконання виводиться у вигляді часу виконання виконання функції. Результат зображено на рис. 4.18.

```
124 ms ± 22.6 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Рис. 4.18. Результат тесту продуктивності.

```
librosa.display.waveshow(np.squeeze(get_audio(noisy_sounds[4]).numpy(),-1))
librosa.display.waveshow(predict_tflite(noisy_sounds[4]).numpy())
```

Рис. 4.19. Код для зображення графіків

Зобразити результат можна за допомогою кода, зображеного на рис. 4.19. У даній частині відбувається відображення звукового сигналу з файлу із шумом, по верх нього накладається звуковий сигнал, що був згенерований у попередніх частинах коду. Що допоможе візуально порівняти результат роботи моделі та сигнал, що був зашумлений. Результат зображено на рис. 4.20.

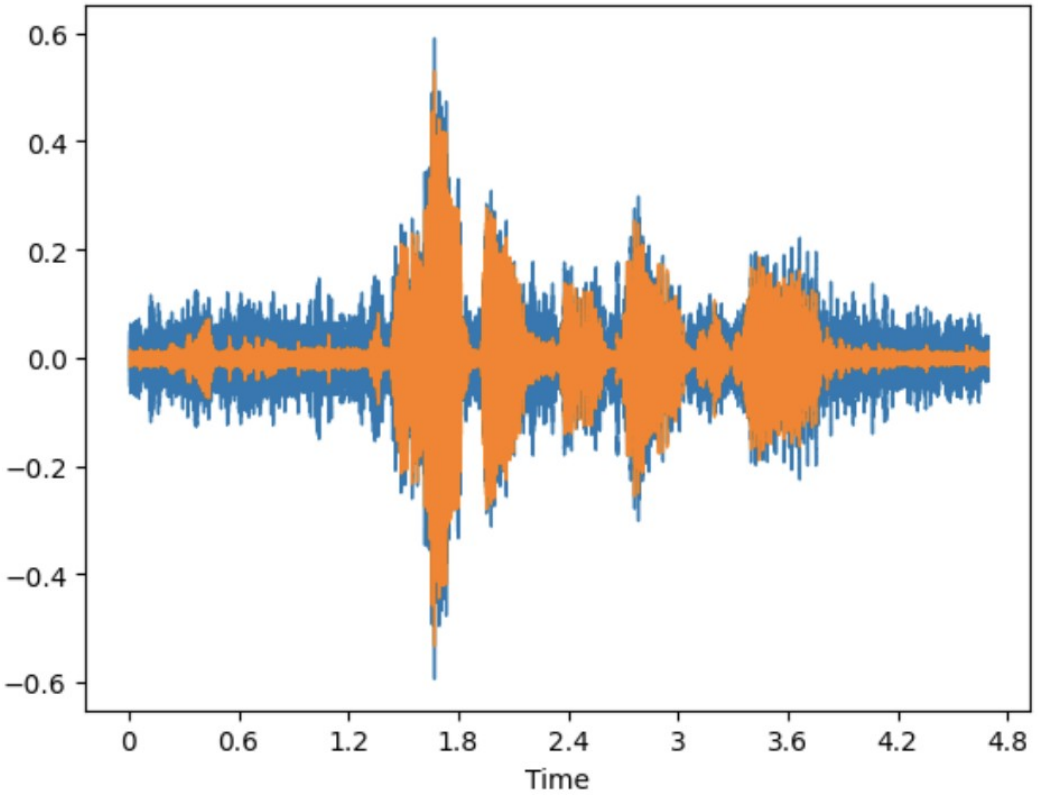

Рис. 4.20. Накладення відфільтрованого сигналу на зашумлений

На рис. 4.20 зображено накладання сигналу, що отримали в результаті обробки, на сигнал із шумами. При вилученні зашумленої частини дістаємо розбірливу частину запису голосу, що значить вдало виконане навчання нейронної мережі за допомогою датасетів.

ВИСНОВКИ

У даній роботі була вирішена актуальна науково-технічна задача створення програми фільтрації мовних сигналів на фоні шуму за допомогою нейронних мереж. Проведений аналіз дав змогу оцінити переваги та недоліки наявних моделей штучних нейронних мереж. В результаті було вибрано оптимальну функцію активації нейрона та структуру нейронної мережі для системи зменшення шумів у мовних сигналах. Також була розроблена програмна реалізація адаптивної системи зменшення рівня шумів у мовних сигналах, яка використовує обрані параметри нейронної мережі для поліпшення результатів навчання та має розроблені архітектурні рішення, що відрізняють її від інших програмних реалізацій системи зменшення шумів.

У ході проектування було створено алгоритм для навчання нейронних мереж, а також програму фільтрації мовних сигналів та програмний інструмент на мові програмування Python, який можна використовувати для фільтрації мовних сигналів від зайвих шумів, як завершений інструмент.

Переваги цієї програми полягають у якості обробки звуку, швидкості обробки сигналів, можливості збільшувати кількість навчальних даних, за необхідності. Для навчання мережі було використане онлайн середовище colab, що надає такі ресурси: system RAM 12.7 GB, GPU RAM 15 GB, disk 78,2 GBб а також мультиплатформенний запуск програмного забезпечення. За необхідності маємо можливість завантажити дані у будь-який момент.

Використання цих переваг дозволяє застосувати цю програму у різних сферах, як від цивільних напрямках, так і військових, для зменшення рівня шуму навколишнього середовища і кращого розпізнавання людських голосів. Програму можна запропонувати для використання у цифрових системах голосового зв’язку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сусліков Л.М., Дьордяй В.С. «Телекомунікації та радіотехніка» Ужгород – 2022.

2. Телекомунікаційні та інформаційні мережі: Підручник [для вищих навчальних закладів] / П.П. Воробієнко, Л.А. Нікітюк, П.І. Резніченко. – К.: САММІТ-Книга, 2010. – 708 с.: іл.

3. Зменшення шумів у телекомунікаційних мережах. URL: https://cpo.stu.cn.ua/Oksana/posibnik/840.html (Дата звернення: 19.02.2023).

4. Динамічний обмежувач шуму. URL: http://ni.biz.ua/7/7_13/7_138783_dinamicheskiy-ogranichitel-shuma.html (Дата звернення: 23.03.2023).

5. Співвідношення сигнал / шум URL: https://uk.wikipedia.org/wiki/Співвідношення_сигнал/шум (Дата звернення 15.03.2023).

6. Автоенкодери в Keras, частина перша, введення URL: https://habr.com/ru/post/331382/ (Дата звернення: 02.01.2023)

7. Що таке GAN – генеративно-змагальні нейронні мережі та як їх застосовувати для генерації зображень URL: https://evergreens.com.ua/ru/articles/gan.html (Дата звернення: 23.01.2023)

8. Теорія електричного зв'язку: навч. посіб. / О.Ю. Гусєв, Г.Ф. Конахович, В.І Корнієнко, Г.В. Кузнєцов, О.Ю. Пузиренка. - Львів: Магнолія, 2006, 2010. - 364 с.

9. Автоенкодери в Keras: Manifold learning та приховані (latent) змінні URL: https://habr.com/ru/articles/331500/ (Дата звернення: 04.02.2023)

10. Автоенкодери в Keras, Частина 4: Conditional VAE URL: https://habr.com/ru/articles/331664/ (Дата звернення: 23.02.2023)

11. Метод опорних векторів (SVM) О. І. Шеремет, О. В. Садовой 2013.

12. Автоенкодери в Keras, Частина 5: GAN(Generative Adversarial Networks) та tensorflow. URL: https://habr.com/ru/articles/332000/ (Дата звернення: 23.01.2023)

13. Hertz, J.; Palmer, Richard G.; Krogh, Anders S. (1991). Introduction to the theory of neural computation. Addison-Wesley. ISBN 978-0-201-51560-2. OCLC 21522159

14. Antoni Buades, Bartomeu Coll, Jean-Michel Morel. A non-local algorithm for image denoising. – 2003.

15. Kruse, Rudolf; Borgelt, Christian; Klawonn, F.; Moewes, Christian; Steinbrecher, Matthias; Held, Pascal (2013). Computational intelligence: a methodological introduction. Springer. ISBN 978-1-4471-5012-1. OCLC 837524179

16. Bishop, Christopher M. (1995). Neural networks for pattern recognition. Clarendon Press. ISBN 978-0-19-853849-3

17. Dewdney, A. K. (1997). Yes, we have no neutrons: an eye-opening tour through the twists and turns of bad science. New York: Wiley. ISBN 978-0-471-10806-1. OCLC 35558945

18. Автоенкодери в Keras, частина 6: VAE + GAN. URL: https://habr.com/ru/articles/332074/ (Дата звернення: 21.02.2023)

19. URL: https://arxiv.org/search/cs?searchtype=author&query=Pascual%2C+S (Дата звернення: 01.01.2023)

20. SUPERSEDED - TSTR VTTC Tsorpus: English Multi-speaker Tsorpus for TSTR Voice Tsloning Toolkit URL:https://datashare.ed.ac.uk/handle/10283/2651 (Дата звернення: 15.10.2022)

21. Wasserman, Philip D. (1993). Advanced methods in neural computing. Van Nostrand Reinhold. ISBN 978-0-442-00461-3. OCLC 27429729

22. Нейронні мережі. Види та класифікація штучних нейронних мереж. Біологічна основа нейрозв'язків. Особливості роботи згорткових нейромереж. URL: https://livingfo.com/shcho-take-nejronni-merezhi-ta-iak-vony-pratsiuiut/ (Дата звернення: 03.01.2023)

ДОДАТОК

```
      # Importing Libraries
      #
_____________________________________________________________________________

      import tensorflow as tf
      from                    tensorflow.keras.layers                    import
Conv1D,Conv1DTranspose,Concatenate,
      Input
      import numpy as np
      import IPython.display
      import glob
      from tqdm.notebook import tqdm
      import librosa.display
      import matplotlib.pyplot as plt
      import os
      import math

      # Extracting Data
      #
_____________________________________________________________________________

      !mkdir "CleanData"
      !mkdir "NoisyData"

      !unzip "/content/drive/My Drive/clean_testset_wav.zip" -d "CleanData"
      !unzip "/content/drive/My Drive/noisy_testset_wav.zip" -d "NoisyData"

      #
_____________________________________________________________________________

      from google.colab import drive
      drive.mount('/content/drive')

      # Load the Data
      #
_____________________________________________________________________________

      clean_sounds = glob.glob('/content/CleanData/clean_testset_wav/*')
      noisy_sounds = glob.glob('/content/NoisyData/noisy_testset_wav/*')

      clean_sounds_list,_ = tf.audio.decode_wav(tf.io.read_file(
      clean_sounds[0]),desired_channels=1)
      for i in tqdm(clean_sounds[1:]):
        so,_ = tf.audio.decode_wav(tf.io.read_file(i),desired_channels=1)
        clean_sounds_list = tf.concat((clean_sounds_list,so),0)

      noisy_sounds_list,_ = tf.audio.decode_wav(tf.io.read_file(
      noisy_sounds[0]),desired_channels=1)
      for i in tqdm(noisy_sounds[1:]):
        so,_ = tf.audio.decode_wav(tf.io.read_file(i),desired_channels=1)
        noisy_sounds_list = tf.concat((noisy_sounds_list,so),0)

      clean_sounds_list.shape,noisy_sounds_list.shape
```

```
batching_size = 12000

clean_train,noisy_train = [],[]

for i in tqdm(range(0,clean_sounds_list.shape[0]-batching_size,
batching_size)):
  clean_train.append(clean_sounds_list[i:i+batching_size])
  noisy_train.append(noisy_sounds_list[i:i+batching_size])

clean_train = tf.stack(clean_train)
noisy_train = tf.stack(noisy_train)

clean_train.shape,noisy_train.shape

# Create a tf.data.Dataset
#
```

```
def get_dataset(x_train,y_train):
  dataset = tf.data.Dataset.from_tensor_slices((x_train,y_train))
  dataset = dataset.shuffle(100).batch(64,drop_remainder=True)
  return dataset

train_dataset = get_dataset(noisy_train[:40000],clean_train[:40000])
test_dataset = get_dataset(noisy_train[4000:],clean_train[4000:])

# Reviewing Sample Waveform
#
```

```
librosa.display.waveshow(np.squeeze(clean_train[5].numpy(),axis=-1))
plt.show()
librosa.display.waveshow(np.squeeze(noisy_train[5].numpy(),axis=-1))
plt.show()

# Creating the Model
#
```

```
inp = Input(shape=(batching_size,1))
c1 = Conv1D(2,32,2,'same',activation='relu')(inp)
c2 = Conv1D(4,32,2,'same',activation='relu')(c1)
c3 = Conv1D(8,32,2,'same',activation='relu')(c2)
c4 = Conv1D(16,32,2,'same',activation='relu')(c3)
c5 = Conv1D(32,32,2,'same',activation='relu')(c4)

dc1 = Conv1DTranspose(32,32,1,padding='same')(c5)
conc = Concatenate()([c5,dc1])
dc2 = Conv1DTranspose(16,32,2,padding='same')(conc)
conc = Concatenate()([c4,dc2])
dc3 = Conv1DTranspose(8,32,2,padding='same')(conc)
conc = Concatenate()([c3,dc3])
dc4 = Conv1DTranspose(4,32,2,padding='same')(conc)
conc = Concatenate()([c2,dc4])
dc5 = Conv1DTranspose(2,32,2,padding='same')(conc)
conc = Concatenate()([c1,dc5])
```

```
      dc6 = Conv1DTranspose(1,32,2,padding='same')(conc)
      conc = Concatenate()([inp,dc6])
      dc7    =    Conv1DTranspose(1,32,1,padding='same',activation='linear')
(conc)
      model = tf.keras.models.Model(inp,dc7)
      model.summary()

      tf.keras.utils.plot_model(model,show_shapes=True,show_layer_names=Fal
se)

      # Training
      #
_____________________________________________________________________________

      model.compile(optimizer=tf.keras.optimizers.Adam(0.002),loss=tf.keras
.
      losses.MeanAbsoluteError())
      history = model.fit(train_dataset,epochs=20)

      # Testing Samples
      #
_____________________________________________________________________________

      from IPython.display import Audio
      Audio(np.squeeze(noisy_train[22].numpy()),rate=16000)

      Audio(tf.squeeze(model.predict(tf.expand_dims(tf.expand_dims(
      noisy_train[22],-1),0))),rate=16000)

      model.evaluate(test_dataset)

      model.save('NoiseSuppressionModel.h5')

      from IPython.display import Audio
      import tensorflow as tf
      model = NoiseSuppressionModel.h5
      noisy_train = p287_286.wav
      Audio(tf.squeeze(model.predict(tf.expand_dims(tf.expand_dims(
      noisy_train[22],-1),0))),rate=16000)

      # Inference
      #
_____________________________________________________________________________

      def get_audio(path):
        audio,_ = tf.audio.decode_wav(tf.io.read_file(path),1)
        return audio

      def inference_preprocess(path):
        audio = get_audio(path)
        audio_len = audio.shape[0]
        batches = []
        for i in range(0,audio_len-batching_size,batching_size):
          batches.append(audio[i:i+batching_size])

        batches.append(audio[-batching_size:])
        diff = audio_len - (i + batching_size)
```

```
  return tf.stack(batches), diff

def predict(path):
  test_data,diff = inference_preprocess(path)
  predictions = model.predict(test_data)
  final_op = tf.reshape(predictions[:-1],((
predictions.shape[0]-1)*predictions.shape[1],1))
  final_op = tf.concat((final_op,predictions[-1][-diff:]),axis=0)
  return final_op

Audio(np.squeeze(get_audio(noisy_sounds[4]).numpy(),-1),rate=16000)

Audio(tf.squeeze(predict(noisy_sounds[4])),rate=16000)

%%timeit
tf.squeeze(predict(noisy_sounds[3]))

librosa.display.waveshow(np.squeeze(get_audio(noisy_sounds[4]).
numpy(),-1))
librosa.display.waveshow(np.squeeze(predict(noisy_sounds[4])))
```