МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ

Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

На тему: **«СЕПАРАЦІЯ АУДІО ДАНИХ НА ОСНОВІ МАШИННОГО НАВЧАННЯ»**

Виконав: студент 2 курсу, групи 8.1210-з
спеціальності 121 інженерія програмного забезпечення
(шифр і спеціальність)

О.В. Таранов
(ініціали та прізвище)

Керівник: Доцент кафедри програмної інженерії, к.ф.-м.ф., доцент Горбенко В.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент: Зав. кафедри фундаментальної та прикладної математики д.т.н., доцент Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“____” _____ 2021 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Таранову Олексію Вікторовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Сепарація аудіо даних на основі машинного навчання

керівник роботи Горбенко Віталій Іванович, доцент каф. програмної інженерії, к.ф.-м.ф.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від «09» 06 2021 року № 850-с

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи
1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Сепарація аудіо даних на основі машинного навчання

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Горбенко В.І., к.ф.-м.ф., доцент	01.09.2021	25.09.2021
2	Горбенко В.І., к.ф.-м.ф., доцент	25.09.2021	17.10.2021
3	Горбенко В.І., к.ф.-м.ф., доцент	17.10.2021	07.11.2021
Додатки	Горбенко В.І., к.ф.-м.ф., доцент	17.10.2021	07.11.2021

7. Дата видачі завдання 09.06.2021

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	01.09.2021	
2.	Збір вихідних даних.	04.09.2021	
3.	Обробка методичних та теоретичних джерел.	25.09.2021	
4.	Розробка першого та другого розділу.	17.10.2021	
5.	Розробка третього розділу.	07.11.2021	
6.	Оформлення та нормоконтроль	18.11.2021	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	09.12.2021	

Студент _____ О.В. Таранов
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.І. Горбенко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ С.П. Швидка
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Сепарація аудіо даних на основі машинного навчання»: 78 с., 26 рис., 6 табл., 16 джерел, 3 додатки.

ДЕКОМПОЗИЦІЯ ЗВУКУ, МАШИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, ОБРОБКА СТАТИСТИЧНИХ СИГНАЛІВ, СЕПАРАЦІЯ АУДІО СИГНАЛІВ.

Об’єкт дослідження – засоби поділу змішаних аудіо сигналів на основі алгоритмів машинного навчання.

Мета роботи: дослідження ефективності та використання сучасних програмних засобів поділу змішаних аудіо сигналів на складові частини на основі алгоритмів машинного навчання.

Метод дослідження – аналітичний.

У кваліфікаційній роботі розглядаються сучасні засоби поділу змішаних аудіо сигналів на індивідуальні, залежно від їхньої природи походження, на основі алгоритмів машинного навчання. Розглянуто концепцію побудови алгоритмів машинного навчання та нейронних мереж для вирішення задач поділу змішаних аудіо даних. Виконано порівняння характеристик та ефективності роботи існуючих відкритих бібліотек призначених для поділу змішаних аудіо даних. На основі цього матеріалу побудовано веб-додаток для практичного застосування алгоритмів машинного навчання при виконанні поділу змішаних аудіо сигналів. Результати роботи можуть бути використані для побудови нових алгоритмів машинного навчання для сепарації аудіо сигналів, а також для їх практичного застосування.

SUMMARY

Master’s qualifying paper «Audio data separation based on machine learning»: 78 pages, 26 figures, 6 tables, 16 references, 3 supplements.

AUDIO DECOMPOSITION, AUDIO SOURCE SEPARATION, MACHINE LEARNING, NEURAL NETWORKS, STATISTICAL SIGNAL PROCESSING.

The object of the study is - means of separation of mixed audio signals into component parts on the basis of machine learning.

The aim of the study is: research of modern software tools for dividing mixed audio signals into component parts based on machine learning, and the possibilities of their application.

The method of research is - analytical.

The qualification work considers modern means of dividing mixed audio signals into individual, depending on their nature of origin, based on machine learning algorithms. The concept of creation of machine learning algorithm and neural network for solving problems of mixed audio data separation is considered. The characteristics and performance of existing open libraries designed for the separation of mixed audio data are compared. Based on this material, a web application was built for the practical application of machine learning algorithms when performing the separation of mixed audio signals. The results of the work can be used to build new machine learning algorithms for the separation of audio signals, as well as their practical application.

ЗМІСТ

ВСТУП

Необхідність розділення аудіо джерел є добре відомою проблемою відокремлення певного джерела звуку, що викликає інтерес, у середовищі, наповненому слуховими подразниками та шумами. В своїй роботі «Деякі експерименти з розпізнавання мови з одним і двома вухами» Колін Чері вперше описав «ефект коктейльної вечірки»: здатність мозку людини відокремлювати одну розмову з людиною від навколишнього шуму повної кімнати людей, що спілкуються, а також від стороннього шуму [1]. Пізніше Альберт Брегман спробував зрозуміти, як мозок людини міг аналізувати складний аудіо сигнал і сегментувати його на окремі потоки [2]. Його робота з аналізу звукових сцен стала основою для створення обчислювального алгоритму, який намагається відтворити або моделювати досягнення мозку за допомогою алгоритмічних засобів [3], зокрема щодо можливостей розділення аудіо джерел.

Проблема привернула широку увагу дослідників за останні 25 років і було запропоновано декілька рішень, що здатні вирішити її в ряді особливих випадків. Тим не менш, в більшості випадках запропоновані методи зазнають невдачі, роблячи цю проблему все ще нерозв'язною.

З розвитком та широким розповсюдженням онлайн трансляцій, подкастів, ефірів та короткометражних відео, високо зріс попит до складних та ефективних засобів обробки звукових сигналів. Звук, що записано у цих видах контенту, зазвичай містить аудіо сигнали різних типів джерел, таких як мова, музика, різні акустичні події, фонові шуми та інше. Ці джерела можуть перекриватись і приховувати кожен інший. Це робить сепарацію цільового аудіо сигналу з початкової суміші сигналів надзвичайно складним завданням.

Потреба у засобах відокремлення різних типів аудіо джерел є надзвичайно високою. Здатність відокремлення цільового аудіо сигналу надає широкий спектр

можливостей з обробки аудіо джерел, таких як усунення фонових шумів та аудіо перешкод, відокремлення мов різних людей від одне одного, ідентифікація акустичних подій на промисловості, в діагностиці машин або обладнання, розділення музики на складові канали та багато інших можливостей.

Найбільш перспективним напрямом досліджень в області обробки та розділення аудіо сигналу на сьогоднішній день є використання алгоритмів глибокого навчання та штучних нейронних мереж.

Штучні нейронні мережі завоювали широку увагу до цього часу у трьох хвилях, викликаних:

1) алгоритмом перцептрона у 1957 році
2) алгоритмом зворотного розповсюдження у 1986 році
3) успіхом глибокого навчання у розпізнаванні мови та класифікації зображень у 2012 році

Настання третьої хвилі призвело до ренесансу глибокого навчання. Зокрема свого розповсюдження набули глибокі нейронні мережі із прямим зв’язком, згорткові нейронні мережі, мережі з довготривалою короткочасною пам’яттю. У цій парадигмі значну роль відіграють архітектури з великою кількістю параметрів, створені щоб навчатись з величезного обсягу даних, та які використовують останні досягнення паралельних обчислень (включаючи хмарні обчислення, графічні та тензорні процесори). Останній сплеск інтересу до глибокого навчання дозволив розширити частку практичного застосування нейронних мереж у багатьох областях обробки даних, часто за ефективністю перевершуючи традиційні засоби обробки даних у великих масштабах. На цій останній хвилі глибоке навчання спочатку набуло популярності в обробці зображень, але потім було значно поширене в обробці мови, а також у численних додаткових галузях, таких як геноміка, квантова хімія, пошук ліків, обробка мови, реклама та системи рекомендацій. Обробка аудіо та, зокрема, сепарація аудіо сигналів все ще не мала широкого напрацювання за допомогою нейронних мереж і глибокого навчання. В результаті, до недавнього

часу, раніше використовувані методи обробки аудіо сигналу, такі як моделі суміші Гауса, приховані моделі Маркова та факторизація негативної матриці, часто все ще перевершували моделі глибокого навчання у додатках для обробки аудіо сигналу.

Хоча при обробці зображень було прийнято багато методів глибокого навчання, існують важливі відмінності між областями, які вимагають особливого погляду на аудіо. Сирі аудіо вибірки утворюють одновимірний сигнал часового ряду, який принципово відрізняється від двовимірного зображення. Аудіо сигнали зазвичай перетворюються у двовимірні подання частоти і часу для обробки, але дві осі, час і частота не є однорідними як горизонтальна та вертикальна осі на зображенні. Зображення – це миттєві знімки цілі і часто аналізуються в цілому або в ділянках з невеликими обмеженнями порядку. Проте аудіо сигнали слід вивчати послідовно в хронологічному порядку. Ці властивості породили специфічні рішення для обробки аудіо.

Останні роки було досягнуто значного прогресу в побудові глибоких нейронних мереж для обробки джерела звуку. Майже всі попередні роботи переважно зосередженні на одному завданні відокремлення джерела звуку, наприклад, покращення якості звучання мови або розділення мови. Мета вдосконалення мови – відновити високоякісні мовні сигнали зі змішаного сигналу шуму та мови, а завдання розділення мовлення має на меті розділити мовлення різних ораторів. Внаслідок різниці цих двох завдань та різних характеристик розділеного цільового сигналу, також будуть відрізнятися область аналізу сигналу та структура нейронної мережі.

Останні дослідження [4] показали, що короткочасні перетворення Фур’є (STFT) більше підходять для посилення мови від фонових шумів, а аналіз та синтез сигналів часової області краще працюють для розділення мовлення. Різноманітність структур нейронних мереж успішно застосовувались для цих двох задач сепарації аудіо сигналу, включаючи рекурентні мережі з довготривалою короткочасною пам’яттю (LSTM), згорткові рекурентні мережі (CRN), тимчасові згорткові мережі

(TCN) та мережі на основі трансформації. Це також показує що захоплення довгострокової залежності від аудіо сигналу має вирішальне значення для кращої продуктивності розділення.

Ще одним завданням розділення аудіо джерел є відокремлення музичних джерел. Цей напрям також став одним з найважливіших при отриманні аудіо даних. Його мета – відокремити різні компоненти джерела звуку від музичного запису, такі як вокал, гітара, барабан та інші музичні інструменти. Класична система з відкритим кодом, що називається Open-Unmix, застосовує двонаправлену мережу LTSM для розділення різних музичних джерел по одному в частотній області. Така система оперує даними на основі отриманої спектрограми та отримує найсучасніші показники. Модель що базується на UNet-LSTM, Demucs, безпосередньо оперує формою хвилі у часової області та може отримувати якісні характеристики у більшості випадків. В такій моделі розглядаються обидві часова область та частотна область і визначається сукупна втрата для обмеження вивчення моделі для запобігання витоку джерела.

Дуже схоже на розділення джерел музики, завдання відділення голосового співу, де необхідно відокремити співочий голос та музичний супровід. Цієї задачі сьогодні також приділяється значна увага.

У цій роботі буде розглянуто задача розділення загального джерела звуку на мовлення, музику та фоновий шум засобами глибокого навчання. На відміну від універсальної задачі розділення джерел, яка розділяє всі довільні джерела звуку у частині змішаного аудіо, у цій роботі буде вивчено можливість поділу джерела звуку на фіксовану кількість цільових аудіо доріжок.

Вихідна мовна аудіо доріжка – це запис звука із голосом, який був відділений від загального джерела, а сигнал музичної доріжки – це широка категорія, яка може складатись із різних акомпанементів. За винятком музики та мови, будь які інші можливі фонові звуки, такі як закриття дверей, дзвінки, звуки тварин та деякі дратівливі шуми класифікуються як сигнали доріжки шуму.

В роботі також буде виконано порівняння існуючих засобів сепарації аудіо сигналу з відкритим вихідним кодом, а також оцінка та аналіз ефективності їх роботи. Вивчення можливостей існуючих моделей нейронних мереж та відмінностей в архітектурі.

У третьому розділі роботи описується розробка веб-додатку на мові Python з інтерфейсом для завантаження користувачем музичного файлу або файлу із сумішшю звукових сигналів і отримання окремих аудіо файлів із відділеними цільовими аудіо сигналами. Для розробки веб-додатку будуть використані моделі машинного навчання з відкритим вихідним кодом.

Завершальною частиною роботи є висновки, що були зроблені в результаті проведених досліджень, налаштуванні моделі нейронної мережі та її практичного застосування. Було зазначено головні аспекти використання алгоритмів машинного навчання та перспективні напрями для подальших досліджень у галузі сепарації аудіо сигналів на основі машинного навчання.

1 ОГЛЯД ВИДІВ ТА МОЖЛИВОСТЕЙ НЕЙРОННИХ МЕРЕЖ

1.1 Загальний опис нейронних мереж

Слідом за успіхами техніки машинного навчання в різних галузях, зокрема таких, як обробка зображень, багатьма дослідниками було прийнято передові алгоритми машинного навчання до парадигми розділення аудіо джерел. Найбільш широко обговорювані з них включають глибокі нейронні мережі, які використовують нейронні мережі з більш ніж одним шаром для вивчення інформації про джерела звуку.

Щоб зрозуміти концепцію розділення даних, їх треба представити в іншій формі, наприклад, класифікувати як належні до класу A або B.

Деякі дані, як наприклад, показано на рис. 1.1 можна розділити лінійно, тобто вони можуть бути представлені повністю з лінійним рівнянням.

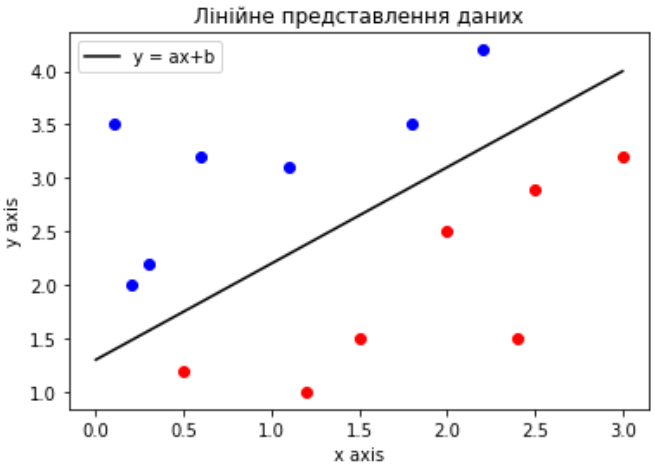


Рисунок 1.1 – Графік лінійного розподілення даних

У наведеному прикладі сині точки представляють дані, що належать до класу A, тоді як червоні крапки представляють дані, що належать до класу B.

Ці дані, таким чином, можна представити у вигляді лінійного рівняння:

$$y = ax + b, \tag{1.1}$$

де значення даних класу А можна визначити за рівнянням:

$$y > ax + b, \tag{1.2}$$

а значення даних класу В можна визначити за рівнянням:

$$y \leq ax + b. \tag{1.3}$$

Проте у більшості випадків, дані не можливо розділити за допомогою лінійної функції. На рисунку 1.2 наведено приклад представлення таких даних.

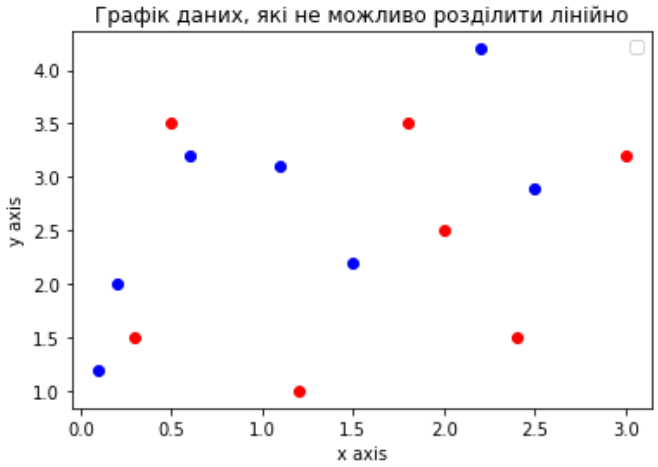


Рисунок 1.2 – Графік даних, які не можливо розділити лінійно

Нейроні мережі надають елегантне рішення знайти спосіб представлення даних, для подальшої обробки і розділення. Для того щоб детальніше вивчити можливості нейронних мереж для розділення даних, необхідно визначити основні поняття.

Нейронна мережа – це система обробки інформації, натхненна нервовою системою людини. Подібно до нервової системи, штучні нейронні мережі складаються зі з'єднань вузлів, які називаються нейронами. Кожен нейрон отримує вхідні дані, обробляє інформацію на вході та надає вихідні дані. Ці нейрони містять параметри, які необхідно оптимізувати за допомогою навчального набору даних. Набір даних містить в собі позначені коректні дані за цільовим напрямом роботи мережі. Після навчання мережа може бути використана для введення даних для отримання результатів тестування та подальшого використання у майбутньому. Хоча існує багато можливих принципів розташування нейронів, найчастіше використовується три вертикально розташованих шарів (приклад наведено на рисунку 1.3):

1) вхідний шар;
2) прихований шар;
3) вихідний шар;

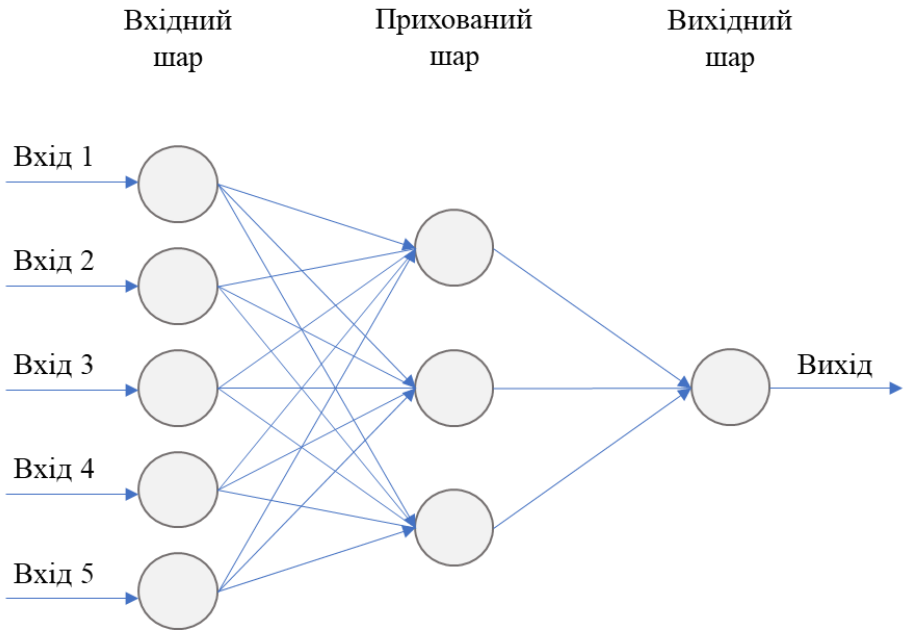


Рисунок 1.3 – Приклад базової повно зв'язної нейронної мережі

Кожен шар складається з нейронів, які можна математично описати в умові його параметрів. Ці параметри оптимізуються на етапі навчання мережі та описані нижче:

1) входи: X, що представляє вхідний вектор: $x_1 \ldots x_n$;
2) одиниця зміщення: x_0, що є постійним значенням, доданим до вхідного вектора;
3) ваги до кожного з входів та зміщення: W, що представляє вектор $w_0 \ldots w_n$;
4) нелінійна функція активації σ, яка може бути (приклади графіків функцій наведені у додатку А):

а) гіперболічний тангенс

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}; \tag{1.4}$$

б) сігмоїда

$$S(x) = \frac{1}{1 + e^{-x}}; \tag{1.5}$$

в) зрізаний лінійний вузол (ReLU)

$$f(x) = max\,(0, x); \tag{1.6}$$

5) вихід a, який може бути представлено як:

$$a = \sigma\left(\sum_{i=0}^{i=N} w_i\, x_i\right), \tag{1.7}$$

який також може бути представлено як:

$$a = \sigma\left(w_0 x_0 + \sum_{i=1}^{i=N} w_i\, x_i\right) = \sigma\left(b + \sum_{i=1}^{i=N} w_i\, x_i\right), \qquad (1.8)$$

де b – зміщення що додається до шару,

x – вектор вхідних активацій,

w – вектор ваг.

Значення b та w будуть параметрами, що оптимізуються під час тренування на сеті даних. Графічно вузол нейрону нейронної мережі можна представити як показано на рисунку 1.4:

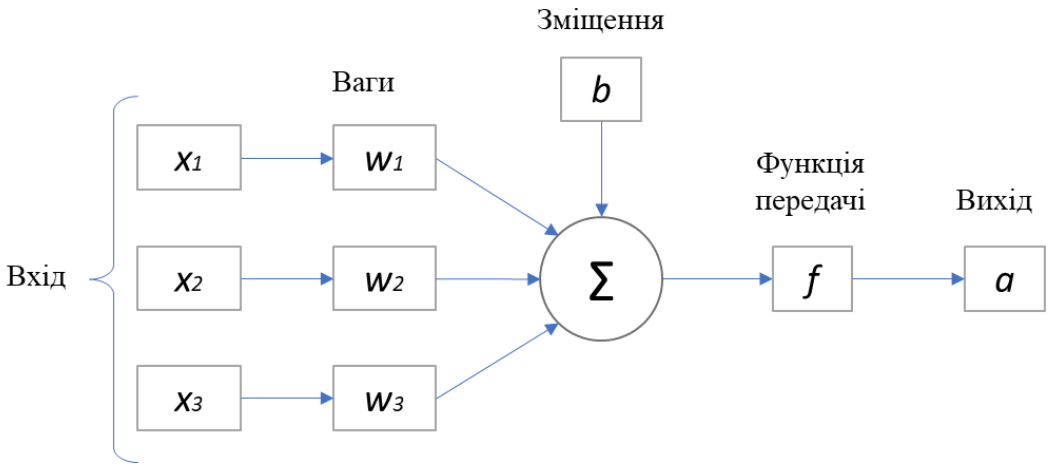


Рисунок 1.4 – Представлення вузла нейрону

Оскільки кожен вузол містить нелінійну функцію, мережа здатна вивчати різні нелінійні уявлення вхідних даних за допомогою різних комбінацій вхідних вузлів та функцій активації.

Глибокі нейронні мережі – це нейронні мережі з більш ніж одним прихованим шаром, як показано на рисунку 1.5.

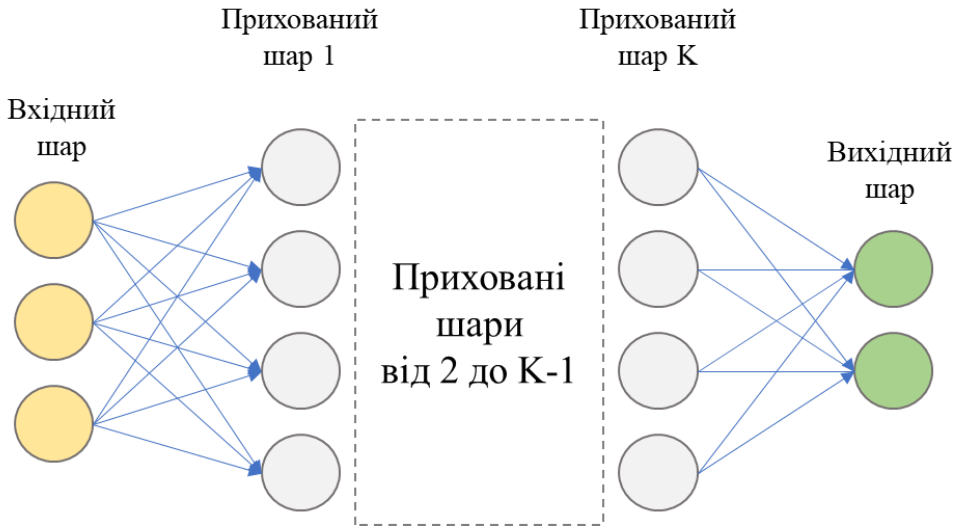


Рисунок 1.5 – Приклад глибокої нейронної мережі із прихованими шарами у кількості K

Оскільки дані проходять через кілька шарів, можна виявити більш абстрактні уявлення про дані, що може допомогти у кращій класифікації даних. Кожен рівень глибокої мережі має входи та виходи, і кількість входів кожного рівня залежить від кількості виходів попереднього. Якщо вхід k – го шару визначено як x_k, то його вхід можна записати як:

$$a_k = \sigma_k\left(b_k + \sum_{i=1}^{i=N} w_{k,i}\, x_{k,i}\right). \tag{1.8}$$

Слід відмітити, що у кожному з прихованих шарів може бути використана своя функція активації. Це пояснює наявність у рівнянні функції активації σ із індексом k.

Особливим типом нейронних мереж з прямим зв’язком є аутоенкодери, де вхідні дані співпадають с вихідними. Аутоенкодер складається з трьох компонентів: кодера, коду та декодера. Кодер стискає вхід і виробляє код, декодер потім

відновлює вхід лише за допомогою цього коду. Схема побудови енкодера зображена на рисунку 1.6:

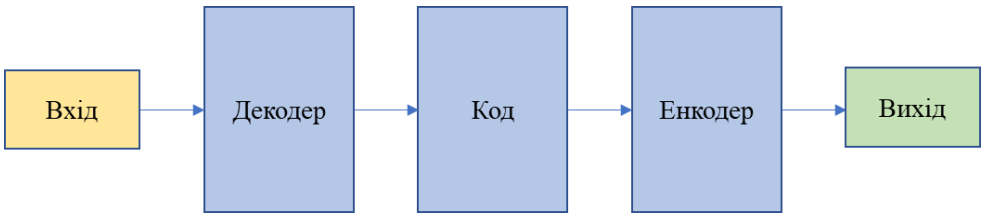


Рисунок 1.6 – Схема побудови аутоенкодера

Аутоенкодер – це, в основному, алгоритм зменшення розмірності (або стиснення) з кількома важливими властивостями:

- Специфічність даних: аутоенкодери здатні лише суттєво стискати дані, подібні до того, на чому вони були навчені. Оскільки вони вивчають особливості, характерні для навчальних даних, вони відрізняються від стандартного алгоритму стиснення даних, такого як zip.
- Втрати: вихід аутоенкодера не буде точно таким же, як вхід. Вихідні дані будуть наближені до вхідних, але будуть мати змінене представлення.
- Навчання без нагляду: для навчання аутоенкодера достатньо подавати на його вхід необроблені дані без необхідності додавання явних міток для навчання, оскільки вони перебувають під самоконтролем та створюють власні мітки з даних навчання.

Архітектурно як кодер, так і декодер є повнозв'язними нейронними мережами з прямим зв'язком. Код – це єдиний шар штучної нейронної мережі із завданою розмірністю. Кількість вузлів у кодовому шарі (розмір коду) є гіперпараметром, який встановлюється перед навчанням автодекодера.

1.2 Основні види нейронних мереж

Розглянемо основні види нейронних мереж та їх властивості. Існує два основних виду нейронних мереж – згорткові нейронні мережі та рекурентні нейронні мережі. Для розуміння та застосування можливостей, які надають ці два види нейронних мереж у контексті обробки та розділення аудіо сигналу, необхідно детальніше розглянути їх побудову та відмінності.

Згорткові нейронні мережі. Згорткові нейроні мережі – це варіант нейронних мереж, натхненних зоровою корою людини та який широко використовується в обробці зображень. Цей тип нейронних мереж використовує місцеву просторову кореляцію між вхідними нейронами зображення, використовуючи локальні сприйнятливі поля. З метою ілюстрації, вхідний шар замість того, щоб бути одновимірним шаром нейронів, можна розглядати як двовимірну матрицю, як у випадку з файлом зображення. Для зображень значення цієї двовимірної матриці представляють інтенсивність пікселів в індексі координат матриці. У звичайній нейронній мережі кожен із цих вхідних нейронів буде з’єднаний з кожним з нейронів першого прихованого шару, утворюючи таким чином повністю з’єднаний шар. Однак у згорткової нейронної мережі кожен нейрон першого прихованого шару з’єднаний лише з невеликою локальною областю вхідного шару, відомою як локальне сприйнятливе поле, як показано на рисунку 1.7.

Це місцеве сприйнятливе поле переміщується по вхідному масиву для формування прихованого шару. Рухи можуть мати різну довжину кроку. На рисунку 1.7 зображено крок (1, 1). Кількість нейронів у прихованому шарі залежить від кількості одиниць у вхідному шарі, кроку та кількості одиниць у локальному сприйнятливому полі. Якщо вхідний сигнал є матрицею розмірів M × N, крок (1, 1), а місцеве сприйнятливе поле – це масив A × B, тоді прихований шар матиме розміри (M – A + 1) (N – B + 1), оскільки локальне сприйнятливе поле може переміщати

одиниці M – A поперек і одиниці N – B вниз, якщо використовується довжина кроку 1.

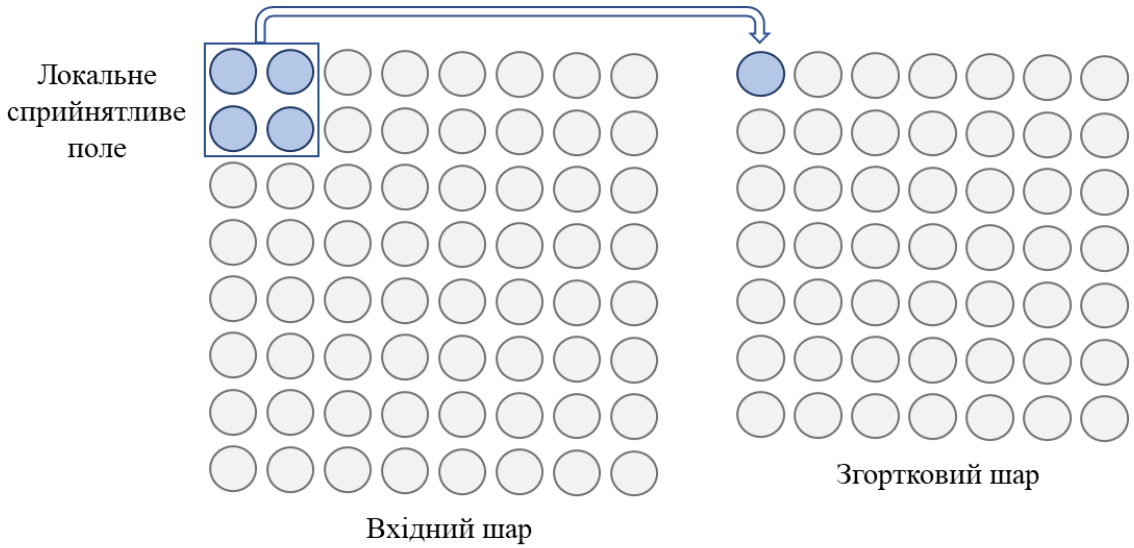


Рисунок 1.7 – Представлення локального сприйнятливого поля у згортковий нейронній мережі

Ключовою особливістю згорткових нейронних мереж є те, що нейрони прихованого згорткового шару поділяють вагу та зміщення так, що весь процес подібний до згортки матриці розміру A × B над матрицею розмірів M × N. Це називається розподілом ваги і є важливою концепцією в згорткових нейронних мережах. Математично вихід j, k – го нейронів згорткового шару можна виразити як:

$$a_{j,k} = \sigma\left(b + \sum_{l=0}^{l=A-1}\sum_{m=0}^{m=B-1} w_{l,m}\, x_{j+l,k+m}\right). \qquad (1.9)$$

Іншими словами згортковий шар обчислює активацію функції розмірів A × B у різних областях вхідного шару. Відображення від вхідного шару до згорткового шару часто називають картою ознак, а спільні ваги та одиниця зміщення

називаються ядром. Оскільки кожне з цих ядер виявляє одну особливість над вхідним шаром, згортковий рівень зазвичай включає більше одного ядра або карти об’єктів, як показано на рисунку 1.8.

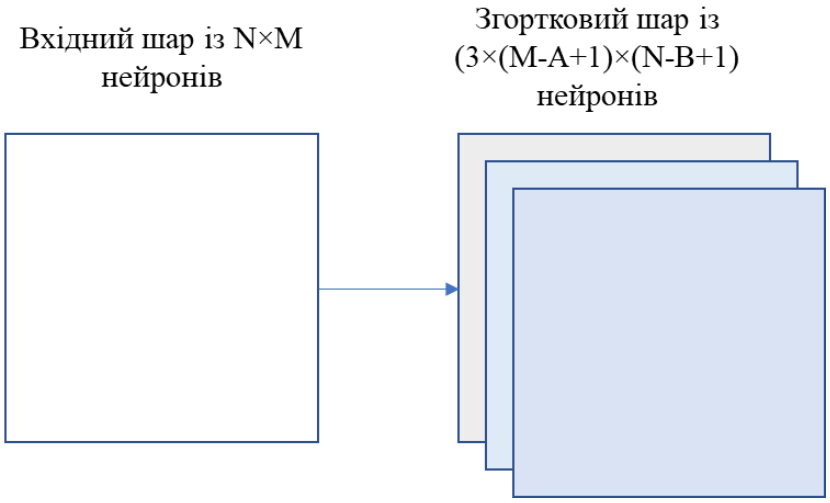


Рисунок 1.8 – Згорткова нейронна мережа: Три фільтра розміром A×B згортаються на вході M×N, в результаті чого утворюється шар з трьома ядрами форми (M – A + 1) × (N – B + 1)

Згорткові нейронні мережі широко використовуються для класифікації зображень, включаючи набір рукописних цифр MNIST. Однією з найбільших переваг використання згорткових нейронних мереж є те, що необхідна пам’ять та ресурси менші ніж у звичайній повністю підключеній нейронній мережі, оскільки ваги та зміщення у прихованих шарах є спільними.

Щоб реалізувати більш ніж одного згорткового рівня і знайти більш абстрактне представлення функцій, часто до згорткової нейронної мережі включаються об’єднуючі шари. Об’єднуючий шар стискає інформацію, що зберігається у згортковому шарі, приймаючи статистичну характеристику локальної області нейронів у згортковому шарі. Найбільш популярним шаром об’єднання є шар Max-Pooling, який просто бере максимум локальної області нейронів у згортковому шарі.

Крім Max-Pooling, досить часто використовується також об'єднання L2. У цьому типі об'єднання береться квадратний корінь із суми квадратів області нейронів, тим самим стискаючи інформацію, що зберігається в цих нейронах.

Рекурентні нейронні мережі. Оскільки аудіо сигнали мають часовий контекст, нейронній мережі необхідно дати деяку пам’ять, щоб додати контекстну інформацію з минулого. Одним із способів вирішення цього є використання рекурентної нейронної мережі. У цьому випадку прихований шар з’єднаний із самим собою, а вага додається до виходу часу t – 1, доданого до функції у момент часу t. Таким чином, вхід шару в момент часу t буде описуватись рівнянням:

$$J(j,t) = \sum X(i,t)W_1(i,j) + b_1W_1(i+1,j) + W_2H\big(J(j,t-1)\big), \qquad (1.10)$$

де $H(J(j,t-1)$ представляє гіпотезу мережі в момент часу t – 1.

Ця функція має нескінченну пам’ять оскільки кожному виводу на кожному часовому кроці надається рівна вага. У цьому випадку градієнт часто досягає надвисокого значення або досягає нульового значення. Тому, незважаючи на теоретичну корисність рекурентних нейронних мереж, практична реалізація може бути досить складною. Ця проблема особливо стосується нижніх шарів глибокої нейронної мережі. В той час як вищі шари вивчають інформацію за допомогою послідовних ітерацій, було помічено, що градієнт для нижніх шарів часто опускається до нуля, і тому подальша інформація про ці шари часто не вивчається.

Одним із простих рішень цієї проблеми є обмеження пам’яті вузла кількома входами, тим самим зменшуючи часовий контекст, який вузол використовує для навчання. Тому рекурентні нейронні мережі є корисними для моделювання короткострокових часових залежностей, але не в змозі моделювати довгострокові залежності. Було також виявлено, що правильна ініціалізація параметрів допомагає усунути ці проблеми.

Нейронні мережі з довгою короткочасною пам’яттю пропонують елегантне рішення обмеження пам’яті, використовуючи вентилі (у деякій літературі ворота), щоб вирішити, коли інформація отримана зі входу, корисна для збереження в контексті навчання, а коли ні. Параметри вентилів також є параметрами що змінюються на етапі навчання.

Приклад блоку з довгою короткочасною пам’яттю наведено на рисунку 1.9.

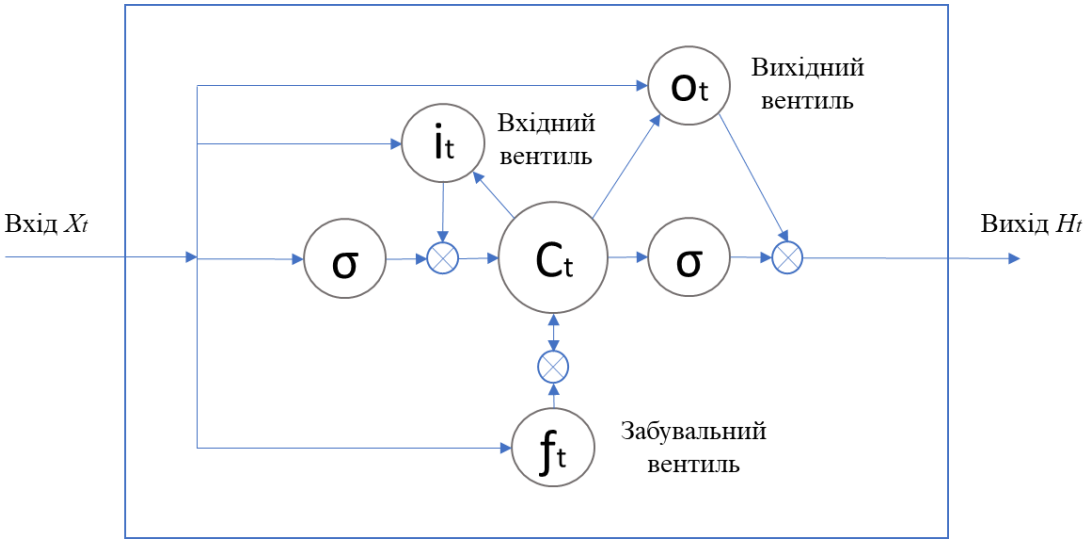


Рисунок 1.9 – Блок з довгою короткочасною пам’яттю

Такий блок має три вентилі: входовий, виходовий та забувальний. У такому блоку попередній часовий фрейм дорівнює *h(t – 1)*, а вхідний вектор *x(t)*. У будь-який момент часу блок з довгою короткочасною пам’яттю має стан нейрону C_{t-1}, який вказує чи потрібно запам’ятовувати вхідні дані.

Для мінімізації загальної похибки довга короткочасна пам’ять на тренувальних послідовностях може застосовувати ітеративний градієнтний спуск. Таким градієнтним спуском є зворотне поширення в часі, що використовується для зміни кожного вагового коефіцієнту пропорційно його похідної по відношенню до похибки. Основною проблемою з градієнтним спуском для стандартних рекурентних нейронних мереж є те, що градієнти похибок зникають експоненційно швидко з розміром часової затримки між важливими подіями [5]. Проте у блоках

довгої короткочасної пам’яті, коли значення похибки зворотно поширюються з виходу, похибка виявляється в пастці в частині пам'яті блоку. Такий ефект постійно подає похибку назад до кожного з вентилів, поки вони не стають натренованими відсікати це значення. Таким чином, регулярне зворотне поширення є дієвим при тренуванні блоку ДКЧП запам'ятовувати значення для дуже довгих тривалостей.

Блоки довгої короткочасної пам’яті можуть також тренуватися поєднанням штучної еволюції для вагових коефіцієнтів прихованих вузлів, і псевдообернення або методу опорних векторів для вагових коефіцієнтів виходових вузлів [6]. У застосуваннях навчання з підкріпленням ДКЧП може тренуватися методами градієнту стратегії, еволюційними стратегіями або генетичними алгоритмами.

1.3 Властивості та додаткові методи роботи нейронних мереж

Пакетна нормалізація. Пакетна нормалізація використовується для підвищення стабільності нейронної мережі. Процес полягає у нормалізації виходу попереднього активаційного шару шляхом віднімання середнього значення пакету та поділу на його стандартне відхилення. Це гарантує відсутність активації, яка б була дуже високою або дуже низькою та зменшує надмірне налагодження за рахунок налаштування параметризації моделі, щоб зробити поверхню втрат більш гладкою.

Короткочасна трансформація Фур’є . Короткочасна трансформація Фур’є – це перетворення, що використовується для визначення синусоїдальної частоти та фазового вмісту локальних ділянок сигналу, коли вони змінюються з часом [7]. На практиці процедура обчислення короткочасної трансформації Фур’є полягає в тому, щоб розділити сигнал більш тривалого часу на коротші сегменти однакової довжини, а потім обчислити перетворення Фур’є окремо для кожного меншого сегмента. Це розкриває спектр Фур’є на кожному коротшому сегменті. Потім

зазвичай відображають змінні спектри як функцію часу у вигляді спектрограми. Осі спектрограми можуть показувати час та частоту сигналу. Колір спектрограми представляє амплітуду частот у цьому діапазоні.

Для моделі, що розглядається у роботі, буде використовуватись функція torch.stft бібліотеки PyTorch. Torch.stft обчислює перетворення Фур'є коротких вікон вхідних даних, що перекриваються. Це дає частотні компоненти сигналу, коли вони змінюються з часом. Інтерфейс цієї функції змодельований на основі функції librosa stft та має вигляд [8]:

torch.stft(input, n_fft, hop_length=None, win_length=None, window=None, center=True, pad_mode='reflect', normalized=False, onesided=None, return_complex=None)

Функція повертає або комплексний тензор розміру (∗×N×T) якщо return_complex == true, або дійсний тензор розміру (∗×N×T×2). Де * це опціональний пакет розміру input, а N це кількість частот де застосовується короткочасне перетворення Фур'є. T це загальна кількість кадрів, що використовується.

Навчальний аналіз ітерацій/втрат. Щоб проаналізувати наскільки є доброю продуктивність моделі, використовується співвідношення між функцією втрат і кількістю ітерацій (епох). Менші втрати означають краще моделювання даних. Щоб зрозуміти, наскільки модель є ефективною для обробки нових даних, розраховуються втрати під час навчання та валідації (тесту), а її інтерпретація – це те, наскільки добре модель працює для цих двох наборів.

Для діагностики нейронної мережі використовуються два терміни:

Зміщення – показник, який вимірює, чи далекі середні прогнозовані значення від фактичних значень. Він вимірює, чи враховує модель складність даних чи ні. Він відноситься до навчальної вибірки.

Дисперсія – показник, який вимірює різницю між продуктивністю моделі в наборах даних для тренування та тестів. Проблема високої дисперсії говорить нам, що модель погано узагальнює нові дані.

Під час навчання модель може досягти стану відсутності прогресу в налаштуванні параметрів, оскільки модель запам’ятовує приклади навчання і стає неефективною для тестового набору. Цей ефект називається надмірною відповідністю. Надмірна відповідність також виникає у випадках, коли набір даних недостатньо великий. З іншого боку недостатня відповідність відбувається коли модель не здатна точно зафіксувати взаємозв’язки між функціями набору даних та цільовою змінною. Для зменшення обох проблем можуть бути застосовані деякі заходи. Щоб зменшити надмірну відповідність, необхідно надати більше прикладів для навчання, зменшити кількість функцій або збільшити регуляризацію. Щоб вирішити недостатню відповідність, необхідно отримати додаткові функції або також збільшити регуляризацію.

2 АЛГОРИТМИ РОЗДІЛЕННЯ АУДІОДЖЕРЕЛ

2.1 Огляд алгоритмів розділення аудіо джерел

Алгоритми розділення аудіо джерел, залежно від вхідних даних можна поділити на два типи: алгоритми на основі форми хвилі та на основі попередньо обробленої форми хвилі (наприклад спектрограми). Одним із прикладів системи на основі обробки форми хвилі є алгоритм нейронної мережі Wave-U-Net, який зображено на рисунку 2.1.

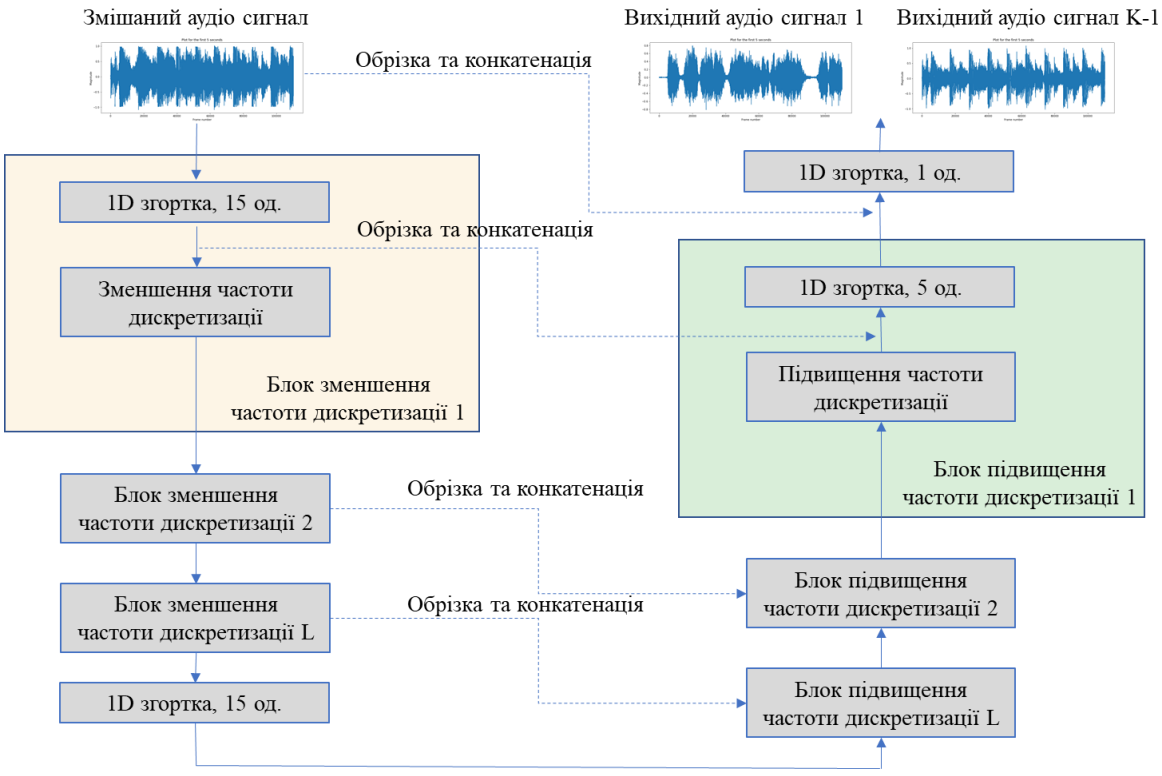


Рисунок 2.1 – Приклад нейронної мережі Wave-U-Net із *K* джерелами *L* та шарами

Алгоритми на основі форми хвилі розробляються як наскрізні системи. Робота в цій області дозволяє моделювати фазову інформацію, уникаючи

фіксованих спектральних перетворювань та розрахунків з низькою затримкою. Більшість з цих алгоритмів використовують декілька одновимірних згорткових шарів та структуру кодування-декодування, використовуючи зменшення та підвищення частоти дискретизації. Така структура називається структурою U-Net. Вона була вперше впроваджена у 2015 року в біомедичних зображеннях для покращення точності та локалізації мікроскопічних зображень [9]. Прикладом використання такої структури для розділення аудіо джерел може бути нейронна мережа Wave-U-Net, яка була описана в роботі «Wave-U-Net: багатомасштабна нейронна мережа для наскрізного розділення джерел звуку» [10]. Початкова звукова суміш використовується на останньому етапі через з'єднання в обхід нейронної мережі. Це надає інформацію про величину та фазу оригінального аудіо, яке використовується для отримання цільових аудіо джерел шляхом застосування ваг масок, отриманих із мережі. Найвідоміші алгоритми, які використовують таку схему це Demucs, Wave-U-Net та Conv-Tasnet. Незважаючи на те, що вони не є найбільш ефективними алгоритмами, останні результати показують потенціал наскрізних систем в розділенні аудіо джерел і ставлять ці моделі в авангарді списку результатів дослідження SISEC18 [11].

Другим та найбільш поширеним типом алгоритмів розділення аудіо джерел є алгоритм попередньо обробленої форми хвилі. Такі алгоритми працюють на спектрограмах, створених короткочасним перетворюванням Фур'є. Вони створюють маску для спектрів величин для кожного кадру та кожного джерела. Вихідний аудіо сигнал генерується шляхом обернення швидкого перетворювання Фур'є на спектрограмах із маскою. Однак вихід швидкого перетворювання Фур'є залежить від багатьох параметрів, таких як розмір та накладання аудіо кадрів і це може вплинути на час та частотну роздільну здатність.

Як і в алгоритмах на основі форми хвилі, ці алгоритми, зазвичай, структурно представляють собою мережу U-Net. Але в цьому випадку з використанням 2D згорткових шарів (наприклад, з використанням тривимірних тензорів з часовими та

частотними інтервалами). Одним з прикладів може бути модель нейронної мережі Spleeter [12], яка використовує глибокий згортковий шар U-Net з базового рівня. Приклад алгоритму на основі попередньо обробленої форми хвилі зображено на рисунку 2.2.

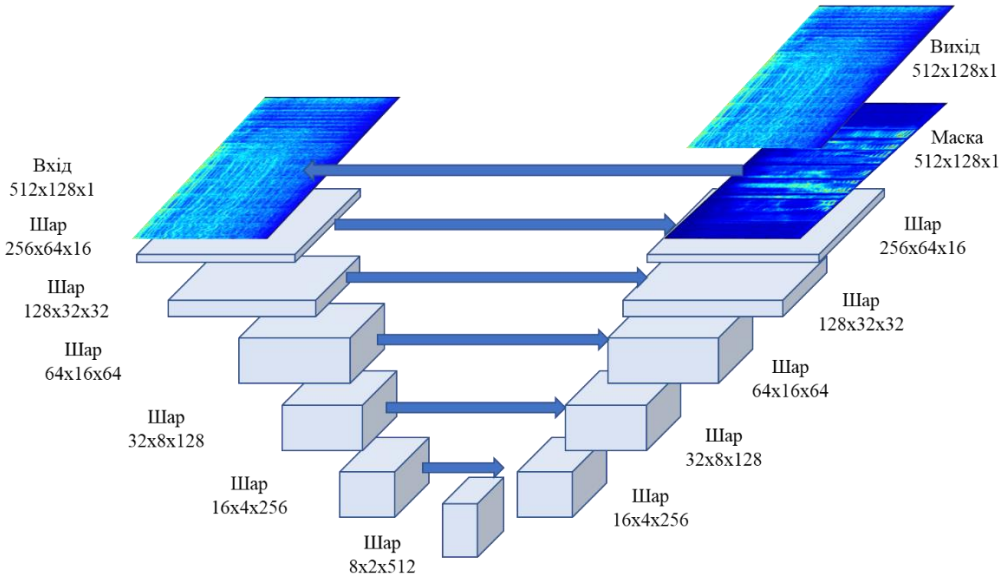


Рисунок 2.2 – Приклад алгоритму розділення аудіо джерел на основі попередньо обробленої хвилі

Найбільш популярними побудованими алгоритмами такого типу на сьогодні є Spleeter та Open-Unmix.

2.2 Оцінка результатів

Оцінка результатів моделі поділу аудіо джерел – це комплексне та складне завдання. Існує декілька розроблених метрик для об’єктивної оцінки ефективності роботи алгоритмів розділення аудіо джерел. З цією метою оцінка джерела $\hat{s}_j$ *(з* $j = 1 \ldots J$*)* розкладаються так:

$$\hat{s}_j = s_{target} + e_{inter\,f} + e_{noise} + e_{artif}, \quad (2.1)$$

де s_{target} – версія оригінального джерела,

$e_{inter\,f}$ – перешкоди, що надходять з небажаних джерел,

e_{noise} – шум сенсора,

e_{artif} – шум, що генерується самим алгоритмом розділення.

Ця декомпозиція встановлює об'єктивні показники та значення, які необхідно максимізувати:

1) відношення джерела до спотворення (Source to Distortion Ratio, SDR), яке визначається як:

$$SDR := 10 * log_{10} \frac{||s_{target}||^2}{||e_{inter\,f} + e_{noise} + e_{artif}||^2}; \quad (2.2)$$

2) відношення джерела до перешкод (Source to Interference Ratio, SIR), яке визначається як:

$$SIR := 10 * log_{10} \frac{||s_{target}||^2}{||e_{inter\,f}||^2}; \quad (2.3)$$

3) відношення джерела до шуму алгоритма (Source to Artefact Ratio, SAR), яке визначається як:

$$SAR := 10 * log_{10} \frac{||s_{target} + e_{inter\,f} + e_{noise}||^2}{||e_{artif}||^2}; \quad (2.4)$$

2.3 Огляд відкритих ресурсів

Для ефективного виконання завдання розділення аудіо джерела нейронною мережею необхідно вирішити багато питань із побудовою нейронної мережі та забезпеченням її роботи. На сьогодні існують рішення, у тому числі з відкритим вихідним кодом, для ефективного опрацювання багатьох аспектів нейронних мереж. Далі у роботі будуть розглянуті відкриті ресурси, які застосовуються для побудови, тренування та оцінки результатів роботи нейронних мереж, а також бібліотеки з відкритим вихідним кодом з побудованими моделями для розділення аудіо сигналу.

Розробка нейронної мережі для розділення аудіо сигналу виконується на мові програмування Python із використанням фреймворку PyTorch.

PyTorch – це фреймворк з відкритим вихідним кодом призначений для розробки проектів машинного навчання, який прискорює шлях від створення прототипів до розгортання програмного забезпечення. Пакет PyTorch містить структури даних для багатовимірних тензорів (контейнер даних) і визначаються математичні операції над ними. Цей функціонал забезпечується модулями torch.nn та torch.optim. Ці модулі дозволяють використовувати високо рівневі API для побудови глибоких нейронних мереж.

Набір даних MUSDB18. Розробка проекту сфокусована на оптимізації роботи бібліотеки Open-Unmix. Ця система виконує тренування нейронної мережі використовуючи датасет MUSDB18. Для роботи будь-якої нейронної мережі необхідний набір даних для виконання тренування параметрів мережі. Завдання розділення аудіо сигналу та спосіб його вирішення може значно відрізнятись в залежності від роду вхідного сигналу (його тематики, будь то музика, суміш речей спікерів, звуки природи чи будь-які інші). Також залежно від роду вхідного сигналу буде залежати набір даних для тренування, а отже і ефективність роботи всієї моделі. Для вирішення цієї проблеми тематика вхідного аудіо сигналу була звужена

до такого типу як музика. У цьому випадку сумішшю аудіо джерел на вході, яке необхідно обробити алгоритмами нейронної мережі, є вхідний музичний звукоряд у вигляді одного аудіофайла. Результатом роботи нейронної мережі та її цільовими аудіо джерелами є окремі аудіофайли із вокалом, виділеним з вхідного джерела, та музичний супровід (акомпанемент). Схема розділення набору даних MUSDB18 зображено на рисунку 2.3.

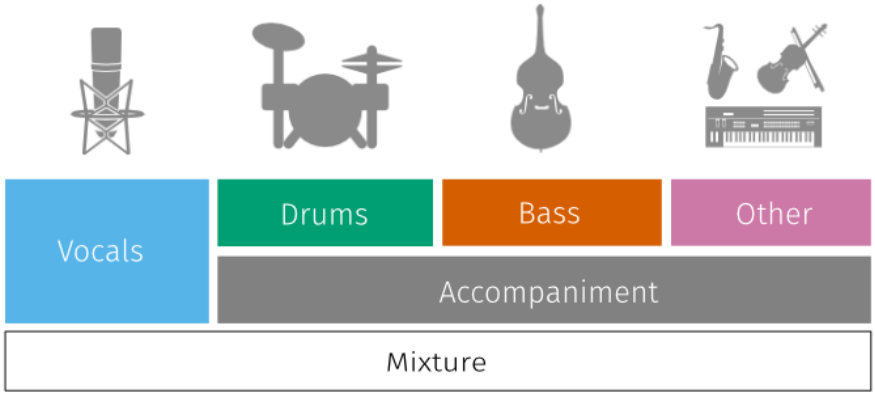


Рисунок 2.3 – Схема розділення аудіо даних тренувального набору MUSDB18 [13]

Така задача потребує відповідний набір даних для тренування нейронної мережі. Набір даних SigSep MUSDB18 складається із 150 пісень повного треку різних стилів і включає як стерео-суміші, так і вихідні джерела, поділені на навчальну підмножину та підмножину тестів. Усі доріжки стереофонічні та кодуються на частоті 44,1 кГц.

Мета цього набору – бути базою даних для проектування та оцінки алгоритмів розділення джерел. База даних представлена у стиснутому та нестиснутому (високоякісному) форматі. Стиснуті файли мають пропускну здатність, обмежену 16 кГц. На сьогодні MUSDB18 є найповнішою базою даних для задачі розділення музичних даних. Крім того, усі сучасні системи використовують цей набір даних для порівняння та обміну результатами.

Блокнот Google Colab. Потужним інструментом для роботи з кодом є інтерактивне хмарне середовище Google Colab. У основі цього хмарного сервісу є блокнот Jupiter для роботи на Python, який виконується на базі Google диска. Сервіс надає можливість працювати із центральним та графічним процесором (CPU та GPU). Значною перевагою роботи з сервісом Google Colab є також можливість застосовувати додаткові бібліотеки для роботи з даними, наприклад Matplotlib. Цей пакет надає можливість візуалізувати набір даних у вигляді графіків із гнучким налаштуванням параметрів.

Open-Unmix – реалізація глибокої нейронної мережі для розділення джерел музики, застосована для дослідників, звуко інженерів та митців [14]. Open-Unmix пропонує готові до використання моделі, які дозволяють користувачам розділити аудіо джерело із музичним файлом на складові джерела: вокал, барабани, бас та інші інструменти, що залишилися. Моделі були попередньо навчені на вільно доступному наборі даних MUSDB18. Для виконання поділу на декілька джерел Open-Unmix містить кілька моделей, які навчаються для кожної конкретної цілі. Хоча це робить навчання менш комфортним, воно дає можливість гнучко налаштовувати дані навчання для кожного цільового джерела. Схема моделі Open-Unmix зображена на рисунку 2.4.

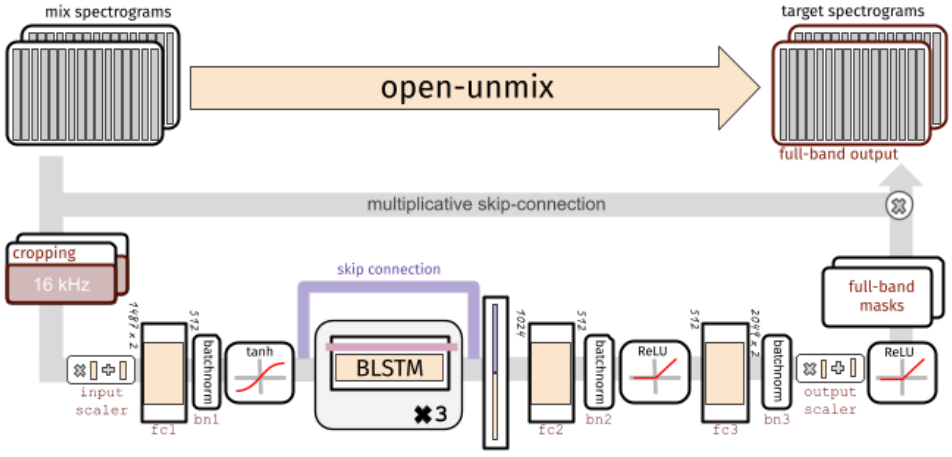


Рисунок 2.4 – Схема моделі Open-Unmix [12]

Кожна вихідна модель Open-Unmix базується на тришаровій двонаправленій глибокій моделі з довгою короткочасною пам’яттю. Модель навчається прогнозувати амплітудну спектрограму цільового джерела за величинною спектрограмою вхідної суміші. Внутрішньо, передбачення отримується шляхом застосування маски на вході. Модель оптимізована в області величин з використанням середньоквадратичної похибки. Вхідна спектрограма стандартизована з використанням загального середнього значення та стандартного відхилення для кожного частотного діапазону у всіх кадрах. Також у моделі застосовується пакетна нормалізація на кількох етапах, щоб зробити навчання більш надійним проти зміни коефіцієнта посилення. Довга короткочасна пам’ять не працює з роздільною здатністю вхідної спектрограми. Натомість на першому кроці після нормалізації мережа вчиться стискати частоту та осі каналу моделі, щоб зменшити надмірність та прискорити збіг моделі.

Ядром Open-Unmix є тришарова двонаправлена мережа з довгою короткочасною пам’яттю. Завдяки своєму періодичному характеру модель може бути навчена та оцінена на довільній довжині звукових сигналів. Оскільки модель одночасно бере інформацію з минулого та майбутнього аудіо джерела, її не можна використовувати в режимі онлайн (реального часу). За необхідності також можна навчити односпрямовану модель.

Після застосування довгої короткочасної пам’яті, сигнал декодується назад до вихідної розмірності. На останніх кроках вихідні дані помножуються на вхідну спектрограму величини, так що моделям пропонується вивчити маску.

Порівняння ефективності фреймворків. Окрім Open-Unmix існує багато інших фреймворків, які здатні із достатньо високою ефективністю виконувати розділення вхідних аудіо даних і зокрема музичних файлів.

У роботі SiSEC (Signal Separation Evaluation Campaign) було проведено порівняння ефективності систем розділення аудіо джерел та оцінка проблем поділу сигналів на основі різних підходів [15]. Методика порівняння була розроблена для

оцінки систем, які виділяють музичні інструменти та вокал з популярних музичних композицій. Дослідження проводилось для шести алгоритмів глибокого навчання для розділення аудіо джерела. Назва алгоритму та його позначення показано у таблиці 2.1.

Таблиця 2.1 – Позначення алгоритмів розділення аудіо джерел

Позначення	Алгоритм
TAK	Multi-scale multi-band DenseLSTM
TAU	Blending of MMDenseNets and LSTM
UHL	Bi-directional LSTM with 3 BLSTM layers
JY	Denoising auto-encoder with skip connections
STL	Wave-U-Net for end-to-end audio source separation

Порівняння проводилось за трьома оцінками співвідношення сигналу до шуму, що були розглянуті у розділі 2.2. Результати оцінок алгоритмів згідно дослідження наведені у таблиці 2.2.

Таблиця 2.2 – Результати оцінок ефективності роботи алгоритмів [16]

Алгоритм	SDR, (дБ)
TAK	5,2
UMX	5,18
TAU	5,17
UHL	5,12
JY	5,1
STL	3,1

Згідно результатів оцінок алгоритмів розділення аудіо джерел [16], найбільш високу оцінку отримав алгоритм Multi-scale multi-band DenseLSTM. Однак цей алгоритм не є алгоритмом з відкритим вихідним кодом, як алгоритм Open-Unmix. Таким чином алгоритм фреймворку Open-Unmix можна вважати алгоритмом, який отримав найвищу оцінку SiSEC серед алгоритмів з відкритим вихідним кодом.

2.4 Розділення аудіо джерела за допомогою фреймворку Open-Unmix

Фреймворк Open-Unmix може бути встановлений за допомогою команди pip install openunmix. Для виконання завантаження та зберігання аудіо файлів Open-Unmix використовує бібліотеку TorchAudio. Додатково може бути встановлена бібліотека tempeg для збільшення спектру підтримуваних форматів вхідних та вихідних файлів.

Open-Unmix працює в частото-часовій області для виконання свого прогнозування. Вхідними даними моделі є:

- models.Separator - вхідний сигнал у часовій області. Він приймає параметри у вигляді: (nb_samples, nb_channels, nb_timesteps),
- models.OpenUnmix – вхідний сигнал у вигляді спектрограми, яка подається ядру безпосередньо (наприклад, при попередньому обчисленні або завантаженні з диска). У цьому випадку вхідні параметри мають вигляд (nb_frames, nb_samples, nb_channels, nb_bins);

Значення вхідних параметрів визначається як:

- nb_samples (кількість зразків в пакеті),
- nb_channels (кількість каналів аудіо, наприклад 1 – моно, 2 – стерео),
- nb_timesteps (кількість аудіозразків у записі),
- nb_frames (часовий вимір короткочасного перетворення Фур'є),
- nb_bins (частотний вимір короткочасного перетворення Фур’є);

Тренування моделі не є частиною роботи пакету Open-Unmix, оскільки він використовує три основні попередньо навчені моделі розділення аудіо:

- umx – навчена на наборі стиснутих даних у форматі stems. Ця модель використовується за замовчуванням,
- umxhq – навчена на відкритому наборі даних MUSDB18-HQ, що містить набір даних у високій якості аудіо.
- umx – навчена на відкритому наборі даних MUSDB18, що містить стиснуті аудіо дані, пропускна здатність яких обмежена 16 кГц.

Усі моделі також доступні у вигляді моделей спектрограми (ядра), які беруть на вхід дані спектрограми та надають на вихід цільові розділені спектрограми. Такі моделі можуть бути завантажені використовуючи команди umxl_spec, umxhq_spec та umx_spec.

Для виконання власного тренування моделі може бути задіяно скрипт train.py, який знаходиться у каталозі scripts. Приклад команди для виконання тренування моделі для вокалу, що використовує набір даних MUSDB18:

python train.py --root path/to/musdb18 --target vocals

Значення аргументів команди будуть (див. таблиця 2.3).

Таблиця 2.3 – Значення аргументів команди для тренування моделі

Аргумент	Опис	Значення за замовчуванням
--root <str>	Шлях до набору даних на диску	Відсутнє
--target <str>	Цільова модель	Відсутнє

Список можливих параметрів команди навчання моделі є набагато ширшим. Параметри тренування моделі, та обґрунтування вибору попередньо тренованої моделі будуть додатково описані в розділі 2.5.

Фреймворк надає можливість виконати розділення аудіо джерела у надзвичайно простий спосіб. Для виконання розділення з параметрами за умовчуванням та на попередньо тренованій моделі необхідно виконати команду umx input_file.wav. Результатом роботи фреймворку будуть чотири окремих аудіо файли, які були відокремлені моделлю з вхідного аудіо джерела input_file.wav. Сепарація вхідного аудіо сигналу також може контролюватись додатковими параметрами команди, які будуть впливати на виконання розділення.

Фреймворк Open-Unmix також надає можливість виконати розділення аудіо сигналу безпосередньо у Python проекті. В основі процессу розділення аудіо знаходиться модуль Separator, який приймає на вході масив аудіо або torch.Tensor та розділяє їх на цільові доріжки. Для кожної цілі може бути завантажена власна модель. Наприклад, для моделей umx та umxhq підтримуються цільові доріжки [‘vocals’, ‘drums’, ‘bass’, ‘other’]. Моделі повинні бути передані в Separator як параметри target_models. Нижче надається приклад конструктора об’єкту Separator, який приймає аргументи із значеннями по замовчуванню:

```
seperator = openunmix.Separator(
    target_models: dict,
    niter: int = 0,
    softmask: bool = False,
    residual: bool = False,
    sample_rate: float = 44100.0,
    n_fft: int = 4096,
    n_hop: int = 1024,
    nb_channels: int = 2,
    wiener_win_len: Optional[int] = 300,
    filterbank: str = 'torch'
):
```

Для використання попередньо тренованої моделі у проекті Python використовується код:

separator = openunmix.umxl(...)

Виконання цього конструктора буде вимагати встановленого пакету openunmix. Фреймворком також надається можливість використовувати модулі, сумісні з torch.hub без необхідності встановлення openunmix. Прикладом використання такого коду є:

separator = torch.hub.load('sigsep/open-unmix-pytorch', 'umxl, device=device)

За допомогою створеного об’єкту Separator можна виконати розділення аудіо:

estimates = separator(audio, ...)

При цьому необхідно щоб аудіо сигнал на вході був правильної форми та частоти дискретизації. Щоб досягти правильних параметрів вхідного аудіо може бути використана попередня обробка в openunmix.utils.preprocess(…), яка приймає масив аудіо та перетворює його у вигляд, прийнятний до використання Open-Unmix.

Для виконання завантаження моделі, попередньої обробки аудіо та сепарації використовується код:

from openunmix import separate

estimates = separate.predict(audio, ...)

У фреймворку є можливість використовувати модель попередньо навчену користувачем. Якщо для команди --model вказано шлях замість ім’я, попередньо навчений об’єкт Separator буде завантажено з диску. Наприклад, якщо виконати команду --model mymodel --targets vocals, то будуть завантажені наступні файли:

- mymodel/separator.json,
- mymodel/vocals.pth
- mymodel/vocals.json

Слід зазначити, що Separator зазвичай об’єднує кілька моделей для кожної цілі і виконує поділ з використанням всіх моделей. Наприклад, якщо Separator

містить моделі vocals та drums, то генеруються два вихідних файла, якщо не вибрана опція --residual, та в цьому випадку буде створено додаткове джерело, що містить в собі всі канали окрім цільового.

Загалом фреймворк Open-Unmix базується на прикладі коду PyTorch MNIST. Структура фреймворку складається з окремих файлів, які легко можуть бути розширені:

- data.py – включає декілька наборів даних, які використовуються для тренування Open-Unmix,
- train.py – включає весь код, що необхідне для старту тренування,
- model.py – включає модулі open-unmix torch,
- test.py – включає код для передбачення/розділення аудіо файлів,
- eval.py – включає весь код, що виконує об’єктивну оцінку результатів, використовуючи museval на наборі даних MUSDB18,
- utils.py – включає додаткові інструменти, наприклад для завантаження аудіо та метаданих;

Одним із ключових аспектів Open-Unmix є гнучкий механізм розширення фреймворку і отже є хорошою відправною точкою для нових досліджень розділення аудіо джерел.

2.5 Обґрунтування обраних параметрів роботи нейронної мережі

У цьому розділі будуть проведені дослідження ефективності роботи фреймворку Open-Unmix для обґрунтування вибору моделі та пошуку оптимальних параметрів роботи нейронної мережі фреймворку для розділення аудіо сигналів.

Дослідження проводились у середовищі Google Colab на мові програмування Python. Повний код дослідження буде приведений у додатку Б.

Фреймворк Open-Unmix має попередньо треновані моделі нейронної мережі для виконання розділення аудіо джерел. Проте, функціонал фреймворку також надає можливість користувачу виконати власне тренування із завданням багатьох параметрів. Тренування моделі виконується за допомогою скрипта train.py або команди інтерфейсу командного рядка:

python train.py --root path/to/musdb18 --target vocals

Найбільш важливі параметри та їх значення за замовчуванням показано у таблиці 2.4.

Таблиця 2.4 – Найбільш важливі параметри тренування моделі

Аргумент	Описання	Значення
--batch-size <int>	Розмір пакета впливає на використання пам'яті та продуктивність рівня LSTM	16
--seq-dur <int>	Тривалість послідовності в секундах фрагментів, взятих з набору даних. Значення <=0,0 означає повне значення	6,0
--hidden-size <int>	Параметр прихованого розміру щільних шарів вузького місця	512
--lr <float>	Швидкість навчання	0,001
--weight-decay <float>	Зниження ваги для упорядкування	0,00001
--bandwidth <int>	Максимальна пропускна здатність в герцах, оброблена LSTM. Вхід і вихід завжди повна пропускна здатність	16000
--nfft <int>	Розмір fft	4096

Згідно документації тренування моделі за замовчуванням виконане на базі відеокарти Nvidia RTX2080, високошвидкісним SSD та параметром –nb-workers 4, що забезпечував обробку на чотирьох паралельних потоках. Тренування

проводилось на чотирьох різних насіннях для кожної цілі та вибрана модель з найнижчою втратою валідації. Тому найкращим рішенням для подальшої роботи використовувати одну із попередньо тренованих моделей.

Для вибору оптимальної моделі необхідно порівняти час виконання процесу розділення аудіо сигналу між різними моделями. Для проведення аналізу було підготовлено п’ять аудіо файлів формату wav, які містять записи музичних композицій, в составі яких присутні звуки різних музичних інструментів та вокал. Цільовим результатом роботи нейронної мережі є сепарація з вхідного сигналу окремо звуку вокалу та музичних інструментів у вигляді окремих аудіо файлів. Більш детальна інформація про параметри музичних файлів (далі – зразки) підготовлених для аналізу приведена в таблиці 2.5.

Таблиця 2.5 – Параметри аудіо файлів для аналізу

Назва файлу	Бітрейт (kbps)	Тривалість (с)	Розмір (MB)
a.wav	1411	8	1,37
b.wav	1411	7	1.18
c.wav	1411	8	1.34
d.wav	1411	8	1.34
e.wav	1411	8	1.43

Замір часу роботи процесора (CPU times) та загального часу роботи програми (Wall time) виконувався на всіх зразках без урахування часу завантаження моделей. Завдання на виконання розділення аудіо сигналу надавалося командою umx <шлях_до_зразка> --model <модель>.

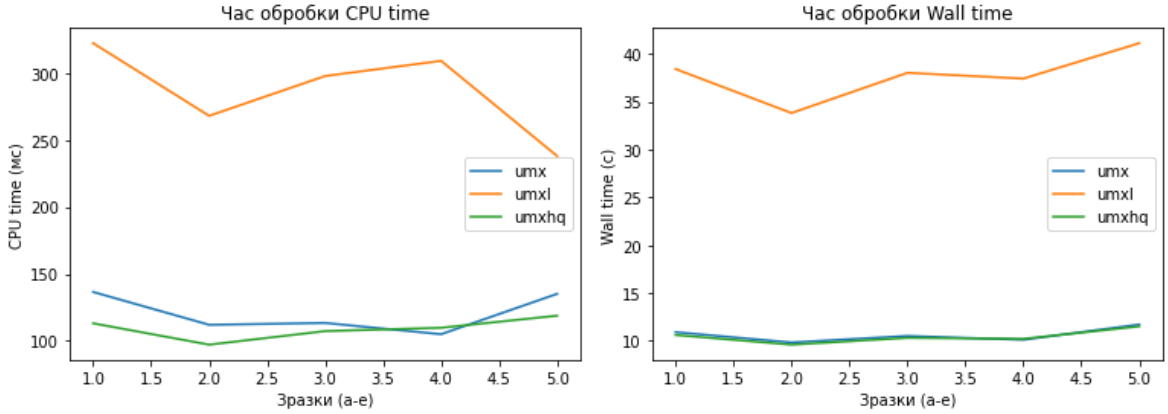
Приклад коду для заміру часу обробки першого зразка на моделі umx:

```
%%time
!umx a.wav --model umx
```

Результати заміру приведені в таблиці 2.6, та репрезентовано у вигляді діаграм на рисунку 2.5. Як видно на діаграмах, час обробки зразків моделлю umxl є значно вищій за час обробки моделями umx та umxhq.

Таблиця 2.6 – Результаті замірів часу обробки зразків на різних моделях

Зразок	CPU times (мс)						Wall time (с)		
	user			sys					
	umx	umxl	umxhq	umx	umxl	umxhq	umx	umxl	umxhq
a.wav	75,5	279	95,3	61	43,1	17,8	10,9	38,4	10,6
b.wav	76,9	226	71	35,1	41,9	26,2	9,82	33,8	9,59
c.wav	87,7	261	77,1	25,7	36,6	30,2	10,5	38	10,3
d.wav	78,9	270	84,9	26,1	38,9	24,9	10,1	37,4	10,2
e.wav	95,3	281	91,9	39,8	46,9	26,9	11,7	41,1	11,5



а) CPU time б) Wall time

Рисунок 2.5 – Графік часу обробки зразків

Наступним етапом вибору оптимальної моделі є аналіз якісних характеристик отриманих цільових аудіо файлів. Порівняння буде проводитись за чотирма показниками SDR, SIR, ISR, SAR, які були описані в розділі 2.2. Для отримання

коректних результатів заміру показників, необхідно порівняти рівень шумів різного походження цільового аудіо файлу, отриманого в результаті роботи моделі із оригінальним окремим каналом цільового аудіо. Для цього необхідно мати на вході моделі аудіо файл із сумішшю звукових сигналів, який буде розділятися нейронною мережею на окремі канали, та набір окремих аудіо каналів, які в суміші повністю відповідають аудіо на вході моделі. З цією метою буде задіяно набір даних MUSDB18, який містить набір аудіо файлів у форматі STEMS.MP4. Такий файл може бути перетворено на вході в модель як цілісні аудіо дані, а також може бути розкладений на окремі канали оригінальних аудіо даних за допомогою бібліотеки museval для виконання порівняння та розрахунку показників якості результативних аудіо файлів моделі.

Розрахунок показників бібліотекою museval виконується за допомогою функції estimate_and_evaluate(track). Порівняння буде проводитись при розділенні аудіо сигналу на чотири складові – vocals, drums, bass, other. Для проведення заміру, аргументом функції буде передано п’ятдесят зразків із набору даних MUSDB18. Результати замірів зберігаються у форматі JSON і можуть далі бути репрезентовані у графічному вигляді.

Команда для виконання заміру показників бібліотеки museval для формування масиву даних по різним моделям:

- !python -m openunmix.evaluate --model umx --outdir /content/evalout/umx/ --evaldir /content/evalresults/umx/
- !python -m openunmix.evaluate --model umxl --outdir /content/evalout/umxl/ --evaldir /content/evalresults/umxl/
- !python -m openunmix.evaluate --model umxhq --outdir /content/evalout/umxhq/ --evaldir /content/evalresults/umxhq/

Файли з даними оцінок показників роботи моделей сформуються у відповідних директоріях. Для репрезентації отриманих JSON файлів по трьом моделям, необхідно змінити формат даних на такий, що буде можливо відобразити

у вигляді графіків. Для цього у середовищі Google Colab виконується код для формування одного масиву даних для наступної обробки:

```
models = ['umx', 'umxl', 'umxhq']
allsongs = []
for i in models:
  counter = 0
  for filename in os.listdir('/content/' + i + '/'):
    onefile = pd.read_json('/content/' + i + '/' + filename)
    onefile = pd.DataFrame(onefile['targets']).to_json()
    onefile = onefile[:1] + '"model":"' + i + '","song_number":"' + str(counter
+ 1) + '","song_name":"' + filename + '",' + onefile[1:]
    counter += 1
    allsongs.append(onefile)
```

В результаті отримуємо один масив з усіма замірами по трьом моделям. Подальша обробка та підготовка даних у вигляді коду надана у додатку Б. Результуючим масивом даних для побудову порівняльних графіків буде масив об’ємом 3600 елементів, кожен з яких є словником що містить результати заміру трьох показників (SAR, SIR, SDR) одного фрейму від одного цільового сигналу по одній моделі кожного аудіо файлу. Таким чином загальна кількість елементів буде дорівнюватися:

$$N = \sum_{j-1}^{n} s_j \times \sum_{j-1}^{n} m_j \times \sum_{j-1}^{n} t_j \times \sum_{j-1}^{n} f_j, \tag{2.5}$$

де s – кількість аудіо файлів,

m – кількість моделей,

t – кількість цільових каналів,

f – кількість часових фреймів.

Графіки будуються за допомогою бібліотеки altair та її функцією alt.Chart(df), що приймає у якості параметра датафрейм df. Датафрейм являє собою перетворений масив словників у таблицю DataFrame. Перетворення у таблицю відбувається за допомогою бібліотеки pandas. Код перетворення у датафрейм та побудови одного з графиків:

```
df = pd.DataFrame(FullDictlist, columns=['num', 'song_num', 'song_name', 'model', 'sound', 'frame', 'SDR', 'SIR', 'SAR', 'ISR'])
        points = alt.Chart(df).mark_boxplot().encode(
          x='song_num',
          y='SDR',
          color='model').properties(width=800)
        lines = alt.Chart(df).mark_line().encode(
          x='song_num',
          y='mean(SDR)',
          color='model'
        ).properties(width=800).interactive(bind_y=False)
    points + lines
```

Порівняти результати заміру показників якості розділення аудіо джерел на різних моделях можна по результуючим графікам.

Для показника SDR (відношення джерела до спотворення) (див. рис. 2.6).

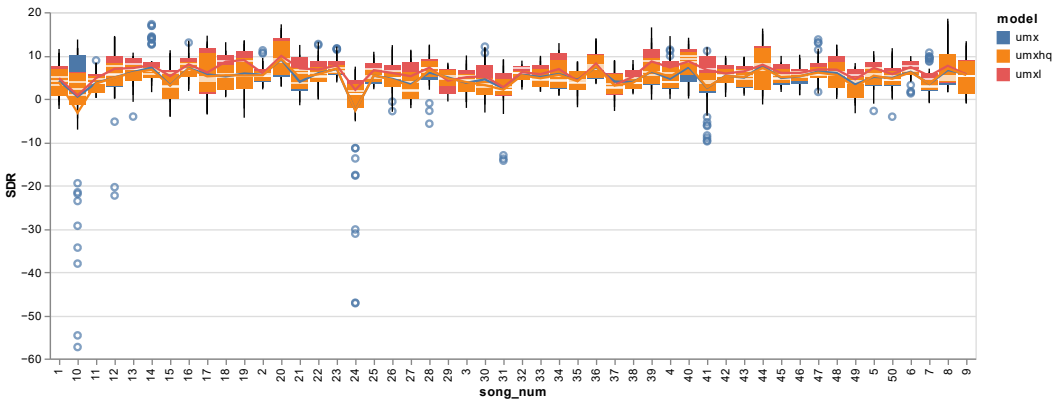


Рисунок 2.6 – Відношення джерела до спотворення

Також загальна оцінка SDR по моделях репрезентована у вигляді графіка на рисунку 2.7. Для побудови цього графіка значення оцінок SDR були підсумовані по всім зразкам, що розглядались, та по трьом моделям окремо. В результаті отримана загальна оцінка по всім зразкам і кожній окремій моделі для більш наглядного порівняння.

Python код для побудови графіка також буде надано у додатку Б.

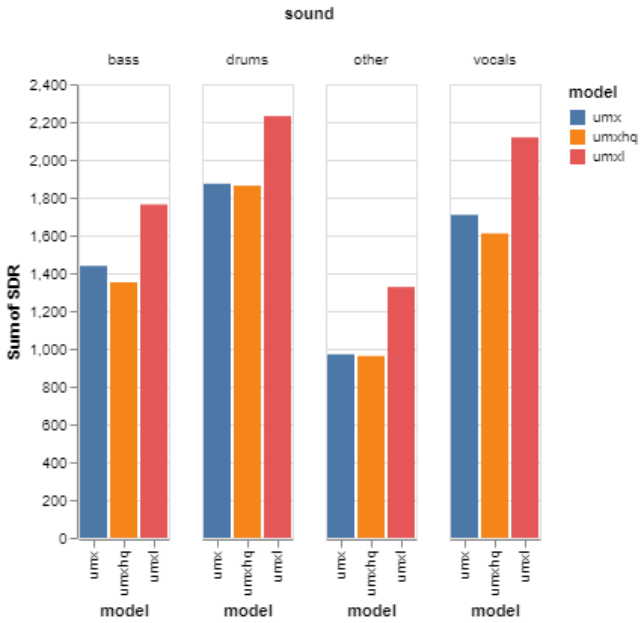


Рисунок 2.7 – Загальна оцінка SDR по моделях

Для показника SIR (відношення джерела до перешкод) (див.рис.2.8).

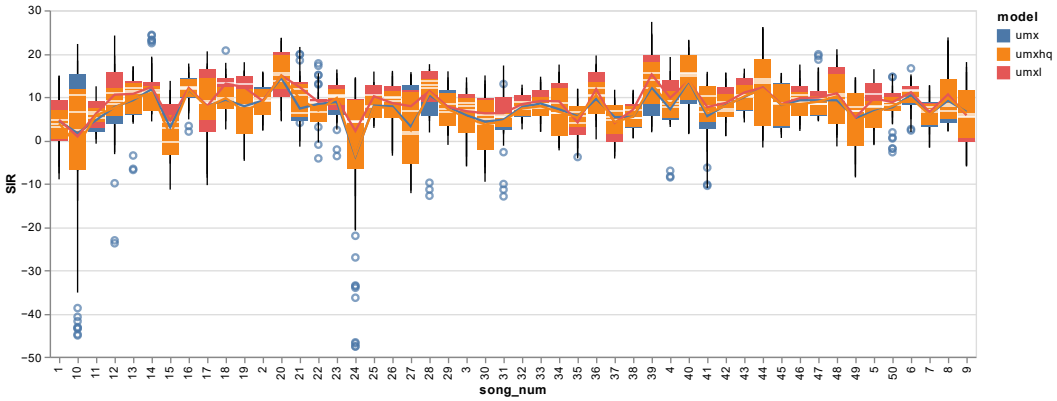


Рисунок 2.8 – Відношення джерела до перешкод

Також загальна оцінка SIR по моделях репрезентована у вигляді графіка (див. рис. 2.9).

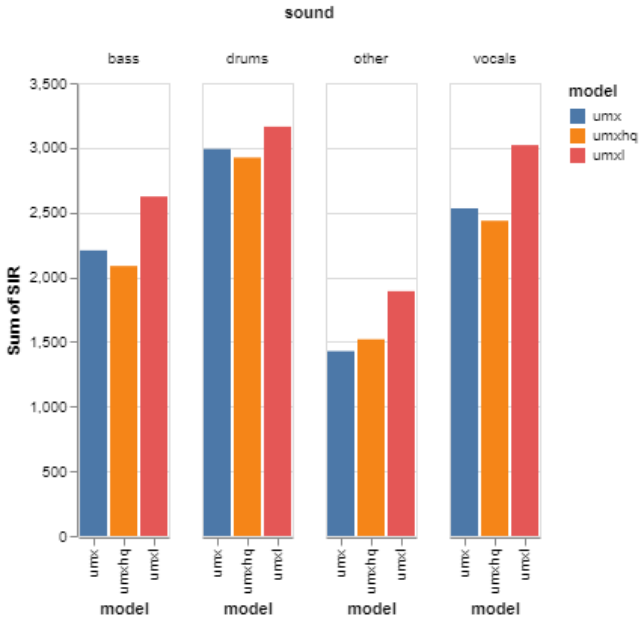


Рисунок 2.9 – Загальна оцінка SIR по моделях

Для показника SAR (відношення джерела до шуму алгоритма) (див. рис. 2.10).

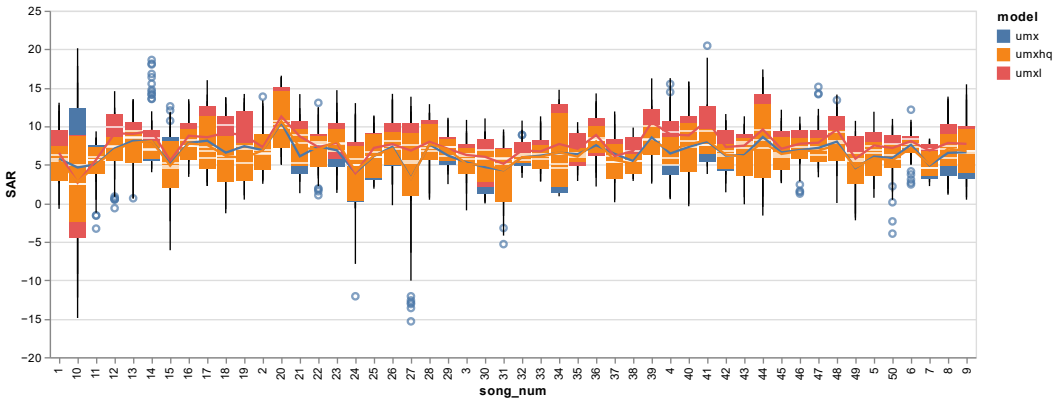


Рисунок 2.10 – Відношення джерела до шуму алгоритму

Також загальна оцінка SAR по моделях репрезентована у вигляді графіка (див. рис. 2.11).

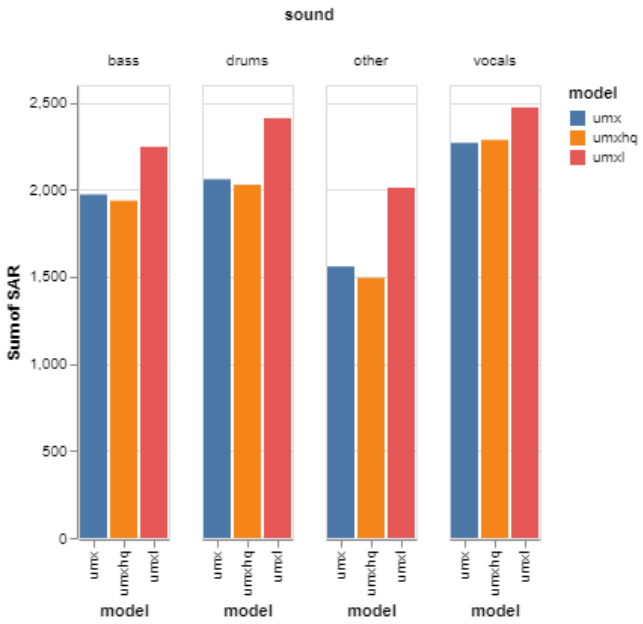


Рисунок 2.11 – Загальна оцінка SAR по моделях

Порівняння отриманих показників показує, що найбільшу оцінку має модель umxl, що є значно вище за оцінки інших двох моделей. Треба враховувати, що модель umxl також потребує найдовший час обробки аудіо файлу. Якщо порівнювати оцінки моделей umx та umxhq, кращий результат майже за всіма показниками показує модель umx.

Для практичного використання алгоритму нейронної мережі у веб застосунку, яке буде описано у розділі 3, необхідно прийняти рішення щодо використання оптимальної моделі. Оскільки часова затримка під час роботи майбутнього веб додатку є критичною для користувача, а при збільшені тривалості вхідного аудіо файлу часова затримка буде зростати ще більше, то модель umxl є менш прийнятною для використання. Різниця у якості вихідного аудіо сигналу буде занадто мала щоб бути критичною для слухового сприйняття людиною. Тож між двома моделями найбільш прийнятною є модель umx, як модель, що має найшвидший час обробки та є другою за якістю вихідних файлів після umxl.

3 ПРАКТИЧНЕ ЗАСТОСУВАННЯ МОДЕЛІ РОЗДІЛЕННЯ АУДІО ДЖЕРЕЛ

3.1 Побудова веб додатку для розділення аудіо файлу

Для практичного застосування моделі нейронної мережі фреймворку Open-Unmix та винайдених параметрів було побудовано веб застосунок (веб додаток), який надає можливість користувачу завантажити свій власний музичний файл та отримати роздільні файли із вокалом та музичним супроводом. У попередньому розділі було виявлено що найбільш оптимальним вибором моделі за критеріями часу обробки та якості вихідних аудіо сигналів є модель umx, тому ядром, що містить модель нейронної мережі яке виконує сепарацію аудіо сигналів із вхідного аудіо файлу, є модель Open-Unmix umx.

Веб додаток побудовано на основі серверу, виконаного на мові Python. Структура каталогів містить окремі каталоги для зберігання скриптів фреймворку нейронної мережі, завантажених та опрацьованих аудіо файлів, інших допоміжних файлів та має вигляд:

- openunmix – містить основні скрипти моделі нейронної мережі,
- cgi-bin – містить Python скрипти для управління фреймворком та завантаження аудіо файлів,
- uploads – каталог для зберігання завантажених аудіо файлів,
- downloads – каталог що генерується динамічно та зберігає файли розділених аудіо сигналів;

За роботу http серверу відповідає скрипт кореневого каталогу server.py, що запускає локальний сервер за допомогою команд:

```
server_address = ("", 8000)
httpd = HTTPServer(server_address, CGIHTTPRequestHandler)
```

```
httpd.serve_forever()
```

Перед початком роботи виконується перевірка та видалення каталогу із старими завантаженими та згенерованими аудіо файлами. За цей функціонал відповідає скрипт uploadFile.py.

```
files = os.listdir(directory)
if(files):
    os.remove(directory + "/" + files[0])
    sourcefn = files[0].split('.')[0]
    targetFolder = './' + sourcefn + '_umxl'
    if(os.path.exists(targetFolder)):
        shutil.rmtree(targetFolder)
```

Після завантаження нового аудіо фала для сепарації через html форму, відкривається сторінка із відображенням назви завантаженого файлу та меню користувача для вибору дій: виконати сепарацію або повернутись назад. Відкриття сторінки виконується за допомогою коду скрипту upload.py:

```
f = open(os.getcwd() + '/uploaded.html', 'r', encoding='UTF-8')
s = f.read()
f.close()
print('Content-type: text/html\n')
```

Та перевірка успішності завантаження аудіо файлу виконується за допомогою коду:

```
fileitem = form["filename"]
if fileitem.filename:
    fn = os.path.basename(fileitem.filename)
    open('uploads/' + fn, 'wb').write(fileitem.file.read())
else:
    message = 'Sound file was not uploaded'
    print(message)
```

Якщо користувач натиснув кнопку «Виконати розділення», скриптом bashFunc.py виконується управляюча команда терміналу з обраними параметрами для виконання розділення аудіо файлу за допомогою моделі нейронної мережі бібліотеки umx. Архітектурне рішення щодо виконання команди терміналу на сервері замість виклика відповідних функцій ядра фреймворку прийнято з метою зробити проект універсальним для застосування додаткових фреймворків. Проект може бути легко масштабований для використання інших фреймворків, таких як Demucs, Conv-Tas-Net та інші. Для цього достатньо буде додати в проект репозиторій фреймворку та сформувати відповідну команду термінала у скрипті bashFunc.py.

Код для виконання команди термінала для umx:

```
def bashFunc():
    bashCommand = "umx ./uploads/" + files[0] + " --targets vocals --residual true"
    os.system(bashCommand)
    try:
        bashFunc()
    except bashFuncError:
        print(bashFuncError)
```

В команді обраними параметрами є вибір моделі нейронної мережі umx, ім’я каталогу і завантажених файлів, цільові канали для сепарації.

Після розділення файлу створюється каталог із назвою, що відповідає назві моделі та назві вхідного файлу, в який розміщуються розділені аудіо файли. Після цього на сторінці відображаються кнопки для завантаження користувачем розділених файлів.

3.2 Опис клієнтської частини та приклад роботи

Клієнтська частина проекту виконана за допомогою мови розмітки html, каскадних таблиць стилів css та скриптів javascript.

Каркасом інтерфейсу веб додатка є три базові сторінки html:

– index.html,

– upload.html,

– download.html;

Стартова сторінка index.html що відкривається автоматично після старту сервера та переходу за його адресом (Для локального сервера http://localhost:8000/), містить форму для завантаження користувачем. Допустимий формат файлів для завантаження визначено .wav для зменшення можливостей виникнення помилок у роботі додатка. Методом передачі даних на сервер є метод post, що є стандартом для безпечного метода передачі даних з форм. Розмітка форми користувача має вигляд:

```
<div class="mainBlock">
    <form action="/cgi-bin/uploadFile.py" method="post"
    enctype="multipart/form-data" class="uploadForm">
        <div class="choose-area">
        <p class="mainText">Choose .wav file for upload</p>
        <input type="file" name="filename" accept=".wav" id="chooseFile">
    </div>
    <div class="send-area">
        <input type="submit" value="Send file" id="sendFile"
        class="mainButton">
    </div>
    </form>
</div>
```

Після завантаження файлу користувача на сервер, відображається форма із кнопкою для виконання розділення аудіо файлу, або повернення назад та видалення завантаженого файлу. Форми прописані у файлі uploaded.html та мають вигляд:

```
<form action="/cgi-bin/bashFunc.py">
        <button class="mainButton">Separate</button>
</form>
```

для виконання розділення файлу, та для повернення назад:

```
<form action="/index.html">
        <button>Back</button>
 </form>
```

Після виконання скрипту bashFunc.py, відображається сторінка download.html із кнопками для завантаження опрацьованих аудіо файлів:

```
<div class="mainButton i" onclick="location.href='/2_umxl/vocals.wav'" download>Download vocal</div>
<div class="mainButton i" onclick="location.href='/2_umxl/residual.wav'" download>Download music</div>
```

Опис стилів знаходиться у файлі index.css, який також забезпечує коректне позиціювання елементів та адаптивність сторінок під різні розміри екранів.

Файл index.js забезпечує підключення та відключення стилів для кнопок інтерфейсу в залежності від їх призначення. Наприклад, якщо аудіо файл не завантажено користувачем – кнопка для розділення файлу буде не активною за рахунок відключення відповідних стилів.

Приклади роботи інтерфейсу та повний код серверної та клієнтської частини веб-додатка наведено у додатку В.

3.3 Опис коду моделі нейронної мережі фреймворку Open-Unmix

Фреймворк Open-Unmix для побудови нейронної мережі використовує бібліотеку PyTorch та TorchAudio (що є також частиною проекту PyTorch).

PyTorch – це бібліотека для машинного навчання для мови Python з відкритим вихідним кодом. Розробка бібліотеки ведеться компанією Facebook.

Структура фреймворку Open-Unmix складається із файлів у кореневому каталогу та має наступний вигляд:

- __init__.py
- cli.py
- data.py
- evaluate.py
- filtering.py
- model.py
- predict.py
- transforms.py
- utils.py

Файл який містить код ініціалізації __init__.py містить набір функцій для виконання сепарації аудіо за допомогою набору моделей нейронної мережі. Функція що використовує модель umx, яка використовується в побудованому веб додатку, приймає наступні аргументи із значеннями за замовчуванням:

- targets=None
- residual=False
- niter=1
- device="cpu"
- pretrained=True
- filterbank="torch"

При цьому в проекті буде змінено параметр за замовчуванням residual на значення True оскільки веб додаток передбачає розділення вхідного аудіо файлу на два окремих каналу (голосовий спів та музичний акомпанемент). Якщо параметр залишити без змін, розділ буде відбуватись на чотири канали. В подальшому для розвитку проекту цей параметр можна змінювати з інтерфейсу веб додатку для розширення його можливостей.

Функція повертає Separator із завданими параметрами, що описано в файлі model.py. Клас Separator є наслідником класу Module бібліотеки torch.nn. Цей клас інкапсулює всю стереофільтрацію у якості модуля torch, щоб забезпечити сквозне навчання. Метод forward класу Separator виконує розділення прийнятого на вхід аудіо у вигляді тензора та повертає сумований тензор розділених форм хвиль.

Окрім цього в файлі model.py міститься клас OpenUnmix, що унаслідується від класу Module бібліотеки torch.nn. Цей клас є ядром модуля розділення аудіо на основі спектрограми.

Функція separate є функціональним інтерфейсом Open-Unmix. Вона виконує розділення torch.Tensor або контенту аудіо файлу. Функція використовує екземпляр класу Separator, який використовується для виводу. На вхід функція приймає такі параметри:

- audio: аудіо до обробки
- rate=None: використовується тільки у випадку якщо аудіо є тензором
- model_str_or_path="umxl": назва моделі або шлях до моделі, яка була тренована користувачем
- targets=None: цілі на які буде розділено вхідне аудіо
- niter=1: кількість ітерацій постобробки
- residual=False: параметр що дозволяє розділення аудіо на два канали
- wiener_win_len=300: кількість кадрів для використання при пакетної обробки
- aggregate_dict=None: додаткові строкові параметри

- separator=None: якщо True буде задіяно об’єкт model.Separator для виконання розділення
- device=None: пристрій для виконання розділення (CPU, GPU)
- filterbank="torch" : засіб реалізації банка фільтрів

Результатом роботи функції є estimates – набір даних у вигляді словника де значенням є тензори розділеного аудіо сигналу.

Для виконання короткочасного перетворювання Фур’є фреймворк має ряд класів у файлі transforms.py. Перетворювання виконується за допомогою методів, описаних у класах, що є також наслідниками nn.Module. У проекті буде задіяно клас TorchSTFT що використовується за замовчуванням та є достатнім для виконання розділення аудіо.

Використання фреймворку відбувається за допомогою команд командного рядка. Інтерфейс для взаємодії через командний рядок описано у файлі cli.py. Файл містить парсери для зчитання аргументів користувача для коректної роботи відповідної моделі із вказаними параметрами.

Використання у веб додатку можливостей інтерфейсу командного рядка надає можливість розширити його функціонал завдяки взаємодії з фреймворком та підключити інші фреймворки, що мають інтерфейс командного рядка, без суттєвих змін у архітектурі веб додатка. Це надає велику гнучкість у подальшому розвитку проекту.

ВИСНОВКИ

У проведеній роботі було розглянуто теоретичну базу сучасних принципів побудови нейронних мереж алгоритмів машинного навчання. Значна увага була звернута на різновиди нейронних мереж та їх відмінності для виконання завдань розділу аудіо сигналів. Розглянуто додаткові методи обробки аудіо сигналів та їх підготовку до обробки нейронною мережею, як наприклад швидка трансформація Фур'є, що використовується для створення спектрограми аудіо сигналу.

Існуючий на сьогодні спектр розроблених алгоритмів для розділення аудіо сигналу є надзвичайно широкий та продовжує стрімко зростати. Більш того, ефективність алгоритмів нейронних мереж в області розділення аудіо сигналів може значно відрізнятись залежно від таких аспектів як: призначення та походження аудіо, особливості нейронної мережі, модель нейронної мережі, дані та вимоги до них, обчислювальна складність, інтерпретація та адаптивність. Для кожного аспекту існує ще багато викликів для подальшої роботи над їх розв'язанням.

З огляду на аспект призначення алгоритму (спеціалізації в області розділення аудіо) та природи походження аудіо, такі рішення можуть використовуватись як для покращення мови, усунення шумів, ідентифікації аномальних звуків у роботі обладнання та промисловості, відокремлення цільових звуків від суміші різного походження або розділення музичного аудіо на окремі канали. В український літературі майже не розкрита тема розділення музичного аудіо на окремі канали, тож в цієї роботі була вивчена саме ця спеціалізація.

Аспект особливостей нейронних мереж для розділення аудіо, здебільшого представляє собою необхідність обирати один із двох принципів обробки аудіо – на основі форми хвилі або на основі спектрограми. Як було розглянуто у роботі найперспективнішим напрямом є обробка сигналу на основі спектрограми.

Подальший вибір фреймворку для побудови веб додатка ґрунтувався на цьому припущені. Однак вивчення алгоритмів на основі форми хвилі та удосконалення їх ефективності продовжується і може бути також перспективним напрямом досліджень.

Вибір моделі нейронної мережі буде сильно залежати від попередніх двох аспектів – призначення (спеціалізації) та особливостей роботи. Аудіо сигнал це послідовність значень в залежності від часу, а отже нейронна мережа повинна враховувати часовий контекст. З цією задачею справляються такі моделі як: згорткова нейронна мережа (CNN), рекурентна нейронна мережа (RNN) та згорткова рекурентна нейронна мережа (CRNN). Всі три моделі можуть вирішувати задачі з моделювання, класифікації та маркування послідовностей. Згорткові нейронні мережі мають фіксоване прийнятне поле, яке обмежує часовий контекст, який приймається для прогнозу і також дозволяє легко розширити або звузити використовуваний контекст. Рекурентні нейронні мережі можуть засновувати свої прогнози на необмеженому часовому контексті, але для цього може потребуватись адаптація моделі (наприклад додавання блоків з довгою короткочасною пам’яттю LTSM) та запобігти прямої залежності від розміру контексту. Крім цього вони потребують послідовну обробку вхідних даних, що сповільнює їх навчання в порівнянні зі згортковими мережами. Багато досліджень ще проводиться і питання яка модель є більш ефективною у яких випадках ще залишається відкритим. Проте на даний час використання моделі згорткової рекурентної мережі є компромісом, що об’єднує переваги згорткових та рекурентних моделей. Саме цьому у роботі вибір був зроблений у бік згорткової рекурентної моделі із довгою короткочасною пам’яттю, на основі якої побудовано фреймворк Open-Unmix.

Аспект наявності набору даних, що використовуються для навчання та тестування моделі нейронної мережі є надзвичайно важливим, та на мою думку, потребує подальшого напрацювання. Існуючий датасет, що використовується в навчанні майже всіх моделей відкритих фреймворків для розділення музичних

файлів складається із 150 композицій (у двох форматах якості). Для подальшого розвитку алгоритмів машинного навчання необхідно збільшувати об’єм датасету та розширювати спектр жанрів музикальних композицій.

Обчислювальна складність при обробці послідовностей нейронними мережами потребує значних обчислювальних потужностей, що в контексті сучасних цін на спеціалізовані пристрої або хмарні сервіси може потребувати значних фінансових витрат. Відкритий фреймворк Open-Unmix вже містить навчені моделі, які були використані у розробці веб додатку.

Побудований у практичній частині веб додаток на основі моделі нейронної мережі фреймворку Open-Unmix є прикладом практичної реалізації можливостей алгоритмів машинного навчання для виконання прикладних задач. Побудований веб додаток має просту архітектуру та може бути легко масштабовано для розширення функціоналу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Cherry E. C. Some experiments on the recognition of speech, with one and with two ears // *The Journal of the Acoustic Society of America.* 1953. №25. P. 974–979.
2. Bregman A. S. Auditory Scene Analysis. Cambridge: MIT Press, 1990. 790 p.
3. Wang D., Brown G. J. Computational Auditory Scene Analysis. Piscataway: IEEE Press, 2006. 395 p.
4. Tan K., Wang D. Learning complex spectral mapping with gated convolutional recurrent networks for monaural speech enhancement // *IEEE/ACM Transactions on Audio, Speech, and Language Processing*. 2020. №28. P. 380–390.
5. Hochreiter S., Bengio Y., Frasconi D., Schmidhuber M. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies: in Kremer S. C. and Kolen J. F. editors. Munchen: IEEE Press. 2001. 464 p.
6. Schmidhuber J., Wierstra D., Gagliolo M., Gomez F. Training Recurrent Networks by Evolino // *Neural Computation*. 2007. №19. P. 757–779.
7. Sejdić E., Djurović I., Jiang J. Time-frequency feature representation using energy concentration: An overview of recent advances // *Digital Signal Processing*. 2009. №19. P. 153–183.
8. PyTorch documentation [Електронний ресурс]. URL: https://pytorch.org/docs/stable/generated/torch.stft.html (дата звернення 20.09.2021).
9. Ronneberger O., Fischer P., Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. Freiburg: Computer Science Department and BIOSS Centre for Biological Signalling Studies. 2015. 8 p.
10. Stoller D., Ewert S., Dixon S. Wave-u-net: A multi-scale neural network for end-to-end audio source separation. *International society for music information retrieval conference*. Paris, 2018. P. 334–340.

11. Stoter F., Liutkus A., Ito N. The 2018 Signal Separation Evaluation Campaign. *14th International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA)*. Guildford. 2018. P 1-10.

12. Hennequin R., Khlif A., Voituret F., Moussallam M. Spleeter: a fast and state-of-the-art music source separation tool with pre-trained models // *Journal of Open Source Software*. 2019. №5. P. 1-4.

13. SigSep Datasets [Електронний ресурс] URL: https://sigsep.github.io/datasets/musdb.html#musdb18-compressed-stems (дата звернення 20.09.2021).

14. Open-Unmix for PyTorch [Електронний ресурс] URL: https://github.com/sigsep/open-unmix-pytorch (дата звернення 20.09.2021).

15. Ward D., Mason R., Kim C. SiSEC 2018: State of the art in musical audio source separation - subjective selection of the best algorithm. *WIMP: Workshop on Intelligent Music Production*. Huddersfield. 2018. P 1-4.

16. Stöter F., Uhlich S., Liutkus A., Mitsufuji Y. A Reference Implementation for Music Source Separation // *Journal of Open Source Software*. 2019. №4. P. 1-6.

ДОДАТОК А

Приклади графіків функцій активації

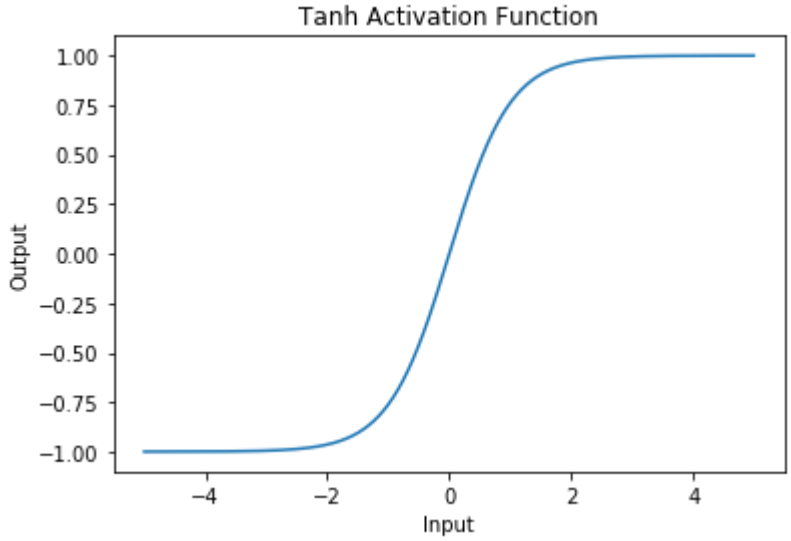


Рисунок А.1 – Приклад функції гіперболічного тангенсу

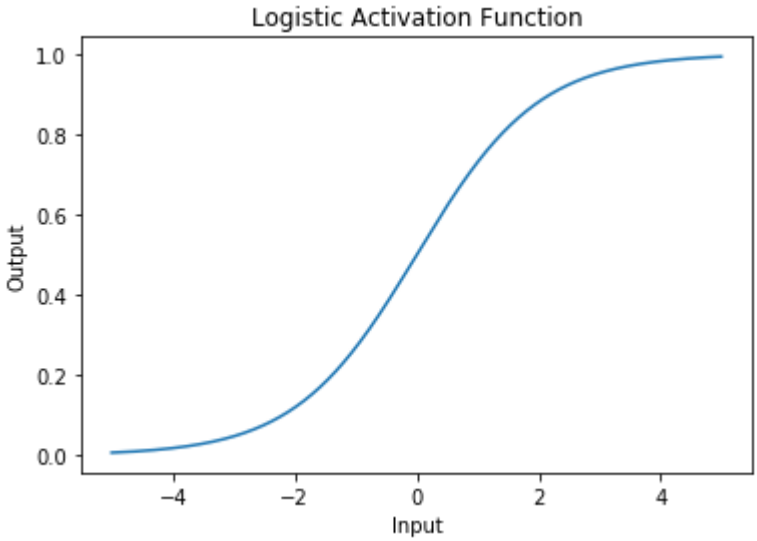


Рисунок А.2 – Приклад функції сігмоїди

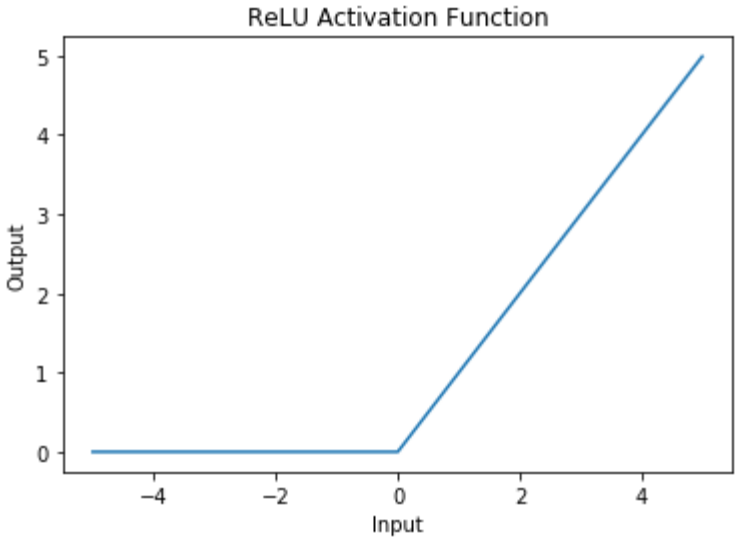


Рисунок А.3 – Приклад функції ReLU

ДОДАТОК Б

Код дослідження ефективності моделей

```
import zipfile
import io
import json
from google.colab import files
import numpy as np
import pandas as pd
import os as os
import matplotlib.pyplot as plt

#Завантаження даних у вигляді ZIP
uploaded = files.upload()
zf = zipfile.ZipFile(io.BytesIO(uploaded['eval-results.zip']), "r")
zf.extractall()

#запис всіх json в один масив строк (150 шт)
models = ['umx', 'umxl', 'umxhq']
allsongs = []
print(type(allsongs))
for i in models:
  counter = 0
  for filename in os.listdir('/content/' + i + '/'):
    onefile = pd.read_json('/content/' + i + '/' + filename)
    onefile = pd.DataFrame(onefile['targets'])
```

```
            onefile = onefile.to_json()
            onefile = onefile[:1] + '"model":"' + i + '","song_number":"' + str(counter + 1) +
'","song_name":"' + filename + '",' + onefile[1:]
            #print(onefile)
            counter = counter + 1
            #python_obj = json.loads(onefile)
            #print(python_obj['targets']['0']['frames'][0]['metrics']['SDR'])
            allsongs.append(onefile)

        #Пустий масив на 3600 елементів
        import copy
        FullDictRows = 3600        # 50songs * 3models * 4sounds * 6frames = 3600
        dict = {'num': 0, 'song_num': 0, 'song_name': 0, 'model': 0, 'sound': 0, 'frame': 0,
'SDR': 0, 'SIR': 0,'SAR': 0,'ISR': 0 }
        FullDictlist = [copy.deepcopy(dict) for x in range(FullDictRows)]
        print(FullDictlist[0])
        print(len(FullDictlist))
        #Перетворення allsongs в вигляд 50 елементів
        # allsongs[150] --> modAllsongs[50] все модели в одной строке
        modAllsongs = []
        modAllsongs = [0 for i in range(50)]
        for i in range(50):
          modAllsongs[i] = '{"umx":' + allsongs[i] + ',"umxl":' + allsongs[i + 50] +
',"umxhq":' + allsongs[i + 100] + '}'

        #преобразование массива 50 в массив 3600
        print(FullDictlist[0])
        values = ['SDR', 'SIR', 'SAR', 'ISR']
```

```
        times = [0.0, 1.0, 2.0, 3.0, 4.0, 5.0]
        targets = [0, 1, 2, 3]
        models = ['umx', 'umxl', 'umxhq']
        counterFull = 0
        counterSongs = 0
        for i in modAllsongs:
          for j in models:
            for k in targets:
              for l in times:
                for m in values:
                  FullDictlist[counterFull]['num'] = counterFull + 1
                  FullDictlist[counterFull]['song_num']=
json.loads(modAllsongs[counterSongs])['umx']['song_number']
                  FullDictlist[counterFull]['song_name']=
json.loads(modAllsongs[counterSongs])['umx']['song_name']
                  FullDictlist[counterFull]['model']=
json.loads(modAllsongs[counterSongs])[j]['model']
                  FullDictlist[counterFull]['sound']=
json.loads(modAllsongs[counterSongs])[j]['targets'][str(k)]['name']
                  FullDictlist[counterFull]['frame']=
json.loads(modAllsongs[counterSongs])[j]['targets'][str(k)]['frames'][int(l)]['time']
                  FullDictlist[counterFull]['SDR']=
json.loads(modAllsongs[counterSongs])[j]['targets'][str(k)]['frames'][int(l)]['metrics']['S
DR']
                  FullDictlist[counterFull]['SIR']=
json.loads(modAllsongs[counterSongs])[j]['targets'][str(k)]['frames'][int(l)]['metrics']['SI
R']
```

```
                FullDictlist[counterFull]['SAR']=
json.loads(modAllsongs[counterSongs])[j]['targets'][str(k)]['frames'][int(l)]['metrics']['SAR']
                FullDictlist[counterFull]['ISR']=
json.loads(modAllsongs[counterSongs])[j]['targets'][str(k)]['frames'][int(l)]['metrics']['ISR']
            counterFull = counterFull + 1
        counterSongs = counterSongs + 1
    print(FullDictlist[0])

    # Печать графіка
    import altair as alt
    points = alt.Chart(df).mark_boxplot().encode(
      x='song_num',
      y='SDR',
      color='model'
    ).properties(
      width=800
    )
    lines = alt.Chart(df).mark_line().encode(
      x='song_num',
      y='mean(SDR)',
      color='model'
    ).properties(
      width=800
    ).interactive(bind_y=False)
    points + lines
```

```
alt.Chart(df).mark_bar().encode(
    x='model',
    y='sum(SDR)',
    color='model',
    column='sound'
)
```

ДОДАТОК В

Код веб додатку

Файл index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Sound separator</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="mainBlock">
        <form action="/cgi-bin/uploadFile.py" method="post" enctype="multipart/form-data" class="uploadForm">
            <div class="choose-area">
                <p class="mainText">Choose .wav file for upload</p>
                <input type="file" name="filename" accept=".wav" id="chooseFile">
            </div>
            <div class="send-area">
                <input type="submit" value="Send file" id="sendFile" class="mainButton">
            </div>
        </form>
    </div>
    <script src="index.js"></script>
```

```
</body>
</html>
```

Файл download.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Sound separator</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="../style.css">
</head>
<body>
    <div class="mainBlock">
        <p class="mainText">Source file is successfully separated</p>
        <p class="secondText">Download separated files</p>
        <div class="downloadButtons">
            <span id="a_001"></span>
            <div class="mainButton i" onclick="location.href='/2_umxl/vocals.wav'" download>Download vocal</div>
            <div class="mainButton i" onclick="location.href='/2_umxl/residual.wav'" download>Download music</div>
            <span id="a_002"></span>
        </div>
        <p class="secondText">Return to home page</p>
        <form action="/index.html">
            <button>Home page</button>
        </form>
    </div>
```

```
</body>
</html>
```

Файл uploaded.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Sound separator</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="../style.css">
</head>
<body>
    <div class="mainBlock">
        <p class="mainText">Sound file is uploaded successfully</p>
        <p class="secondText">Click SEPARATE for run</p>
        <form action="/cgi-bin/bashFunc.py">
            <button class="mainButton">Separate</button>
        </form>
        <p class="secondText">Or click BACK for return</p>
        <form action="/index.html">
            <button>Back</button>
        </form>
    </div>
</body>
</html>
```

Файл server.py:

```
import os
```

```
#import index

from http.server import HTTPServer, CGIHTTPRequestHandler
server_address = ("", 8000)
httpd = HTTPServer(server_address, CGIHTTPRequestHandler)
httpd.serve_forever()
```

Файл uploadFile.py:

```
#!/usr/bin/env python3
import cgi
import cgitb
	cgitb.enable()
	import os
	import shutil
	#Очистка старых файлов
	directory = './uploads'
	print('directory' + directory)
	files = os.listdir(directory)
	if(files):
		print('files' + files[0])
		os.remove(directory + "/" + files[0])
		sourcefn = files[0].split('.')[0]
		targetFolder = './' + sourcefn + '_umxl'
		print('targetFolder' + targetFolder)
		if(os.path.exists(targetFolder)):
			shutil.rmtree(targetFolder)
			print('targetFolder DELETE')
```

```
f = open(os.getcwd() + '/uploaded.html', 'r', encoding='UTF-8')
s = f.read()
f.close()
print('Content-type: text/html\n')
form = cgi.FieldStorage()
fn = 'fileIsNotExist'

fileitem = form["filename"]
if fileitem.filename:
        fn = os.path.basename(fileitem.filename)
        open('uploads/' + fn, 'wb').write(fileitem.file.read())
        #message = 'Sound file is uploaded successfully'
        #print(message)
else:
        message = 'Sound file was not uploaded'
        print(message)
print(s)
```

Файл bashFunc.py:

```
#!/usr/bin/env python3
import os

f = open(os.getcwd() + '/download.html', 'r', encoding='UTF-8')
s = f.read()
f.close()
print('Content-type: text/html\n')
print(s.partition('<span id="a_001"></span>')[0])
directory = './uploads'
```

```
files = os.listdir(directory)
def bashFunc():
    bashCommand = "umx ./uploads/" + files[0] + " --targets vocals --residual true"
    os.system(bashCommand)
try:
    bashFunc()
except bashFuncError:
    print(bashFuncError)

sourcefn = files[0].split('.')[0]
targetFolder = '/' + sourcefn + '_umxl/'
donloadLinks = [
    targetFolder + 'vocals.wav',
    targetFolder + 'residual.wav'
]
print('<a href="' + donloadLinks[0] + '" download class="a_mainButton"><div class="mainButton">Download vocal</div></a>')
print('<a href="' + donloadLinks[1] + '" download class="a_mainButton"><div class="mainButton">Download music</div></a>')
print(s.partition('<span id="a_002"></span>')[2])
```

ДОДАТОК Г

Вигляд інтерфейсу

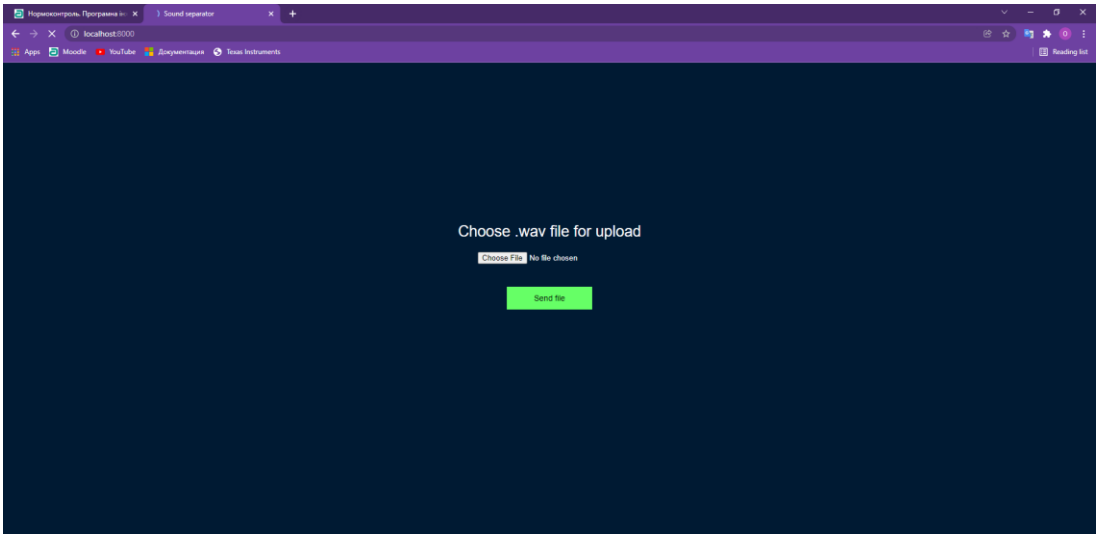


Рисунок Г.1 – Вигляд інтерфейсу до завантаження аудіо фалу

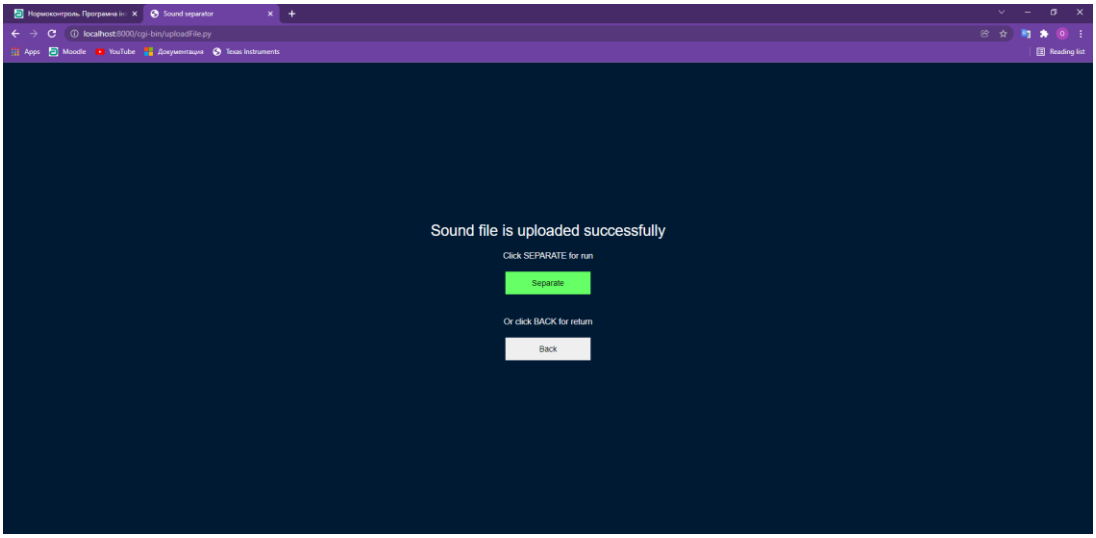


Рисунок Г.2 – Вигляд інтерфейсу після завантаження аудіо фалу

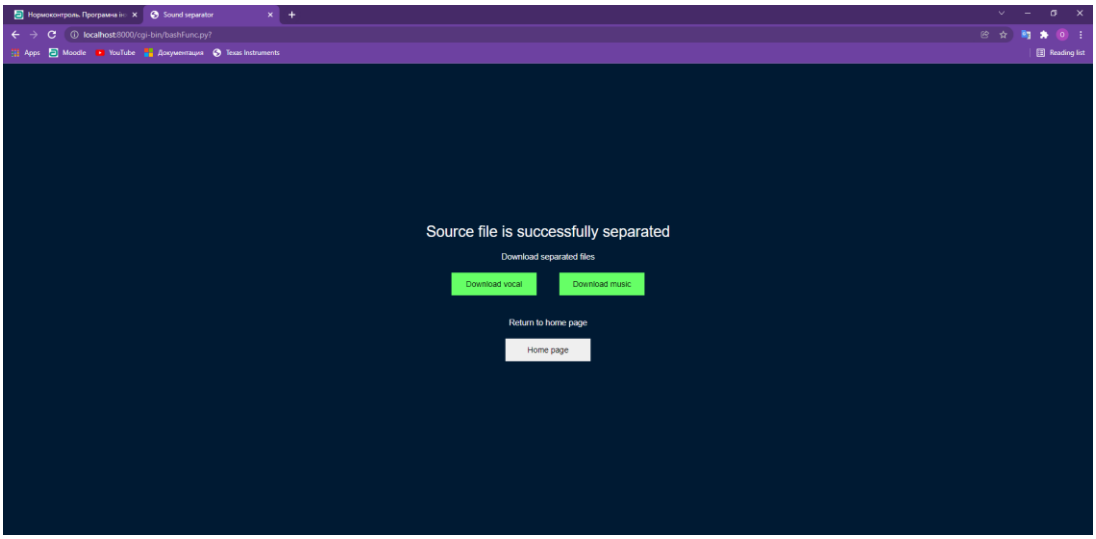


Рисунок Г.3 – Вигляд інтерфейсу після розділення аудіо фалу