Міністерство освіти і науки України

Національний технічний університет

«Дніпровська політехніка»

Інститут електроенергетики

(інститут)

факультет інформаційних технологій

(факультет)

Кафедра інформаційних систем та технологій та комп'ютерної інженерії

(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня магістра

(бакалавра, спеціаліста, магістра)

Студента Ястребцева Олексія Дмитровича

(ПІБ)

академічної групи 126м-20-1

(шифр)

спеціальності 126 « Інформаційні системи та технології»

(код і назва спеціальності)

за освітньо-професійною програмою

«Інформаційні системи та технології»

(офіційна назва)

на тему Дослідження методів нейронних мереж для використання в інформаційних системах пошуку жанрової музики

(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	доц. Гаркуша І.М			
розділів:				

Рецензент				

Нормоконтролер				

Дніпро

2022

ЗАТВЕРДЖЕНО:

завідувач кафедри

інформаційних технологій

та комп'ютерної інженерії

(повна назва)

_____ Гнатушенко В.В.

(підпис) (прізвище, ініціали)

«_____»_____2021_року

ЗАВДАННЯ
на кваліфікаційну роботу

ступеня магістр

(бакалавра, спеціаліста, магістра)

студенту Ястребцеву О.Д. **академічної групи** 126м-20-1

(прізвище та ініціали) (шифр)

спеціальності 126 « Інформаційні системи та технології »

за освітньою-професійною програмою «Інформаційні системи та технології»

на тему Розробка інформаційної системи пошуку та аналізу жанрової музики

затверджену наказом ректора НТУ «Дніпровська політехніка» від

_____№ 2241-Л

Розділ	Зміст	Термін виконання
Розділ 1	Аналіз стану області рішень	1.10.2021 – 31.10.2021
Розділ 2	Моделі на методи розв'язання задачі	1.11.2021 – 30.11.2021
Розділ 3	Розробка інформаційної системи пошуку та аналізу жанрової музики	1.12.2021 – 21.12.2021

Завдання видано _____ Гаркуша І.М.

(підпис керівника) (прізвище, ініціали)

Дата видачі 1.10.2021 р.

Дата подання до екзаменаційної комісії 23.12.2021 р.

Прийнято до виконання _____ Ястребцев О.Д.

(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: __ стор., __ рис., __ додатків, __ джерел.

Об'єкт дослідження: методи побудування інформаційної системи пошуку,аналізу та класифікації інформації.

Предмет дослідження: інформаційна система – нейрона мережа пошуку жанрової музики.

Мета магістерської роботи: створити інформаційну систему нейронної мережі для пошуку жанрової музики.

У вступі подано стан проблеми та виконана постановка задачі дослідження.

Перший розділ присвячено аналізу теми дослідження. Наведено огляд відкритих джерел та існуючі рішення аналізу аудіофайлів.

В другому розділі наведено проектну складову вирішення завдання.

Розглянуто можливості відомих архітектур нейроних мереж та їх комбіннацій. Описано переваги та недоліки різних підходів до проектування нейронних мереж.

Третій розділ присвячено розробці інформаційної системи на мові програмування Python на вирішенню проблеми класифікації музики на жанри.

Практична цінність результатів полягає у тому, що розроблена інформаційна система розширює границі процесу аналізу та класифікації жанрової музики.

МОВА PYTHON, KERAS, TENSORFLOW, НЕЙРОННІ МЕРЕЖІ, MACHING LEARNING, DEEP LEEARNING

ABSTRACT

Explanatory note: __ pages, __ figures, __ appendices, __ sources.

Object of research: methods of building an information system for searching, analyzing and classifying information.

Subject of research: information system - neural network of genre music search.

The purpose of the master's work: to create a neural network information system for searching genre music.

The introduction presents the state of the problem and the formulation of the research task.

The first section is devoted to the analysis of the research topic. An overview of open sources and existing solutions for analyzing audio files is given.

The second section presents the project component of the problem.

Possibilities of known architectures of neural networks and their combinations are considered. The advantages and disadvantages of different approaches to neural network design are described.

The third section is devoted to the development of an information system in the Python programming language to solve the problem of classifying music into genres.

The practical value of the results is that the developed information system expands the boundaries of the process of analysis and classification of genre music.

LANGUAGE PYTHON, KERAS, TENSORFLOW, NEURAL NETWORKS, MACHING LEARNING, DEEP LEEARNING

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ……6

ВСТУП……..7

РОЗДІЛ 1 АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАДАЧІ

1.1 Задачі які вирішують нейронні мережі……..9

1.2 Переваги та недоліки використання нейронних мереж……..12

1.3 Існуючі рішення для аналізу аудіофайлів……15

1.3.1 Robust Constellations...……16

1.3.2 Fast Combinatorial Hashing…….18

1.3.3 Searching and Scoring…..22

1.3.4 Шумостійкість……23

1.3.5 Швидкість……...24

1.3.6 Специфічність і хибні позитиви……...……...24

1.4 Існуючі вільні цифрові сервіси-постачальники музики (API)……25

РОЗДІЛ 2. МОДЕЛІ ТА МЕТОДИ РОЗВ’ЯЗАННЯ ЗАДАЧІ

2.1 Попередня обробка даних ……..……...….27

2.2 LSTM ……..30

2.3 Згортка нейронна мережа ……33

2.3.1 Конволюційний шар...……34

2.3.2 Об’єднаний шар ……37

2.3.3 Повністю підключений шар ……..…38

2.4 Згорткова рекурентна модель……...38

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗРОБЛЕНИХ ФРАГМЕНТІВ АБО КОМПОНЕНТІВ КОМП’ЮТЕРНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

3.1 Розробка класу для перетворення вхідних даних на тренувальну вибірку.42

3.2 Розробка класу нейронної мережі ……45

3.3 Функція – точка старту/початку……46

ВИСНОВКИ……48

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ……...49

ДОДАТОК А……...50

ДОДАТОК Б……...51

ДОДАТОК В……...56

ДОДАТОК Г……...57

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ШНМ – штучні нейроні мережі

API – Application Programming Interface набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

MLP - Multilayer Perceptron

CNN - Convolutional Neural Network

RNN - Recurrent Neural Network

ВСТУП

Нейронні мережі, штучний інтелект і Deep Learning - найгарячіші напрямки, які в міру вдосконалення алгоритмів стрімко захоплюють нові сфери та ринки [1]. За експертними прогнозами, у перспективі глибокі нейронні мережі здатні перевершити людину за різними функціями [2]. Одна з підстав такої популярності – їх чудові здібності до навчання за спостережуваними прикладів та формування прийнятних висновків на базі неповної та неточної вхідної інформації. Роботи з нейронних мереж спочатку було розпочато біологами. За допомогою нейромереж дослідники прагнули вивчити властивості та особливості роботи головного мозку. Нині все більший інтерес до штучних нейронних мереж виявляють різні галузі промисловості та непромислової сфери. Штучні нейронні мережі ефективно використовуються для розпізнавання відеозображень, письмового тексту та мовлення, розв'язання різноманітних задач прогнозування та у багатьох інших областях. Наразі відома велика кількість комерційних програмних систем моделювання, що дозволяють досліджувати та розробляти штучні нейронні мережі для різних додатків, а також розроблено значну кількість нейрокомп'ютерних систем.

Зокрема набувають великої популярності та актуальності нейронні мережі для розпізнавання та пошуку музичних композицій у зв'язку з збільшенням числа інтернет сервісів з музичною тематикою та зростання їх популярності.

Завдання точної ідентифікації жанру виявляється складною як для людини, так і для комп'ютера. Часто немає загальноприйнятого поняття класифікаційних ознак тієї чи іншої жанру та розподілу типів жанрів. Незважаючи на те, що поділ музики за жанрами є в значній мірі суб'єктивним, для опису того чи іншого жанру можна використовувати критерії, пов'язані з фактурою, інструментуванням та ритмічною структурою музики. Наявність надійних тестових даних великого обсягу також є ключовою вимогою для ефективного навчання музичних

класифікаторів музичних творів за жанрами, проте у відкритому доступі практично немає подібних баз даних.

Нейромережеві класифікатори які нині є широко використовуваними, зважаючи на їх здатність навчатися на обмеженій множині прикладів та наявністю ефективних алгоритмів навчання.

РОЗДІЛ 1

АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАДАЧІ

1.1 Задачі які вирішують нейронні мережі

Основне призначення нейронних мереж – це вирішення інтелектуальних завдань. Тобто завдань, вирішення яких відбувається не за заздалегідь визначеним, точним алгоритмом, а завдяки самонавчанню як властивості системи, що навчається. Відразу можна сказати, що наведені нижче типи задач не можна вважати жорстко розмежованими - вони можуть плавно перетікати один одного.

Розпізнавання образів (класифікація) -тип завдання, у якому нейронна мережу відносить той чи інший об'єкт одного з класів з урахуванням аналізу його ознак. Бінарна класифікація – класифікація з двома класами. Multilabeling – вид класифікації, у якому класифікований об'єкт може ставитися до одного класу одночасно.

Класифікація - типова і одне з найпоширеніших завдань, наприклад:

- розпізнавання осіб
- Розпізнавання емоцій
- класифікація типів огірків на конвеєрі при сортуванні тощо.

Завдання розпізнавання образів - це ситуація, коли ми маємо кілька класів (від 2 і більше) і нейронна мережа має віднести образ до тієї чи іншої категорії. Це завдання класифікації.

Регресія – завдання оцінки істинного (числового) значення деякої незалежної змінної (вихід нейронної мережі) від сукупності залежних змінних (вхід мережі).

Сутність цього завдання – отримати на виході з нейронної мережі не клас, а конкретне число, наприклад:

- Визначення віку за фото
- прогнозування курсу акцій
- Оцінка вартості нерухомості

Наприклад, у нас є дані про об'єкт нерухомості, такі як: метраж, кількість кімнат, тип ремонту, розвиненість інфраструктури на околицях цього об'єкта, віддаленість від метро і т.д.

І нашим завданням є передбачити ціну на цей об'єкт, виходячи із наявних даних. Завдання цього - це завдання регресії, т.к. на виході ми очікуємо отримати вартість – конкретне число.

Прогнозування часових рядів - це тип завдання, при якому володіючи впорядкованим за часом рядом значень, нам потрібно зрозуміти, які значення будуть йти в ньому далі.

Суть її полягає в тому, що у нас є динамічна часова низка значень, і нам потрібно зрозуміти, які значення будуть йти в ньому далі.

Наприклад, це завдання з передбачення:

- курсів акцій, нафти, золота, біткойна
- Зміни процесів у котлах (тиск, концентрація тих чи інших речовин тощо)
- Кількості трафіку на сайті
- обсягів споживання електроенергії тощо.

Навіть автопілот Tesla також можна віднести до завдання цього типу, адже відеоряд слід розглядати як тимчасовий ряд зображень. Тобто, з одного боку, ми передбачаємо тимчасовий ряд тиску та концентрації тих чи інших речовин, а з

іншого - розпізнаємо по цьому тимчасовому ряду патерни, щоб оцінити, чи критична ситуація, чи ні. І тут уже йде розпізнавання образів.

Саме тому ми одразу наголосили на тому, що не завжди є чітка межа між різними типами завдань.

Кластеризація / Навчання без вчителя - тип задач, у якому відбувається розбиття вибірки на групи подібних об'єктів, у своїй випробувана система спонтанно навчається виконувати поставлене завдання без втручання із боку експериментатора чи середовища.

У цій ситуації ми маємо багато даних і невідомо, які з них до якого класу відносяться, при цьому є припущення, що там є кілька класів.

Наприклад, це:

- Виявлення класів читачів при email-розсилках
- Виявлення класів зображень

Типове маркетингове завдання - ефективно вести email-розсилку. Припустимо, у нас є мільйон email-адрес, і ми ведемо якусь розсилку.

Зрозуміло, люди поводяться абсолютно по-різному: хтось відкриває майже всі листи, хтось не відкриває майже нічого.

Хтось постійно клікає за посиланнями у листах, а хтось просто їх читає та клікає вкрай рідко. Хтось відкриває листи вечорами у вихідні, а хтось - вранці у будні, і таке інше.

Завдання кластеризації у цьому разі - це аналіз всього обсягу даних, і виділення кількох класів передплатників, які мають подібними поведінковими патернами (у межах, кожного класу, очевидно).

Інший приклад цього завдання - це, наприклад, виявлення класів для зображень. Один із проектів у нашій Лабораторії – це робота з фентезі-картинками.

Якщо передати ці зображення нейронної мережі, то, після їх аналізу, вона, наприклад, "скаже" нам, що виявила 17 різних класів. Припустимо, один клас - це зображення із замками. Інший – це картинки із зображенням людини по центру. Третій клас – це коли на картинці дракон, четвертий – це багато персонажів на тлі природного ландшафту, відбувається якась битва, тощо. Це приклади типових завдань кластеризації.

Генерація - це тип завдання, при якому нейронна мережа створює контент різного типу, починаючи з текстів, зображень та аудіо, і закінчуючи "розфарбовуванням" чорно-білих фільмів та "зміною" сезону у відеороликах.

Генеративні мережі (GAN) - це найновіший тип мереж, який стрімко розвивається. Якщо говорити коротко, то їхнє завдання – машинна творчість. Під машинною творчістю мається на увазі генерація будь-якого контенту:

- текстів (вірші, тексти пісень, оповідання)
- зображень (у тому числі фотореалістичних)
- аудіо (генерація голосу, музичних творів) тощо.

Крім того, до цього списку можна додати завдання трансформації контенту:

- розфарбовування чорно-білих фільмів у кольорові

- Зміна сезону у відеоролику (наприклад, трансформація навколишнього середовища із зими в літо) та ін.

У випадку з "переробкою" сезону зміни торкаються всіх важливих аспектів. Наприклад, є відео з реєстратора, в якому машина взимку їде вулицями міста, довкола лежить сніг, дерева стоять голі, люди у шапках тощо.

Паралельно з цим роликом йде інший, перероблений у літо. Там уже замість снігу зелені газони, дерева з листям, люди легко одягнені тощо.

1.2 Переваги та недоліки використання нейронних мереж

Як було згадано раніше ШНМ мають великий потенціал у вирішенні задач без відомого закінченого алгоритму рішення. Наряду з цим використання нейронних мереж має наступні перевваги:

- Нейронні мережі мають здатність вчитися самостійно і генерувати вихідні дані, які не обмежуються вхідними даних, які вони надають.
- Вхідні дані зберігаються у власних мережах замість бази даних. Отже, втрата даних не впливає на спосіб їх роботи.
- Нейронна мережа буде вчитися на екземплярах і адаптувати їх, коли відбувається подібна подія, що дозволить їм функціонувати через подію в режимі реального часу.
- Навіть якщо нейрон не відповідає або інформація втрачена, мережа все одно здатна виявити несправність і створити вихід.
- Нейронні мережі виконують кілька завдань паралельно, не впливаючи на продуктивність системи.
- Зберігання інформації по всій мережі.
- Вміння працювати з неповними знаннями.
- Наявність відмовостійкості.
- Наявність розподіленої пам’яті.
- Можливість паралельної обробки.

Незважаючи на перерахованні переваги використання ШНМ, високий потенціал їх розвитку та складність завдань, які вони вирішують є й недоліки та обмеження їх використання такі як:

Апаратна залежність:

- Штучні нейронні мережі за своєю структурою вимагають процесорів з паралельною потужністю обробки.

- З цієї причини реалізація обладнання залежить.

Незрозуміле функціонування мережі:

Це найважливіша проблема ШНМ.

- Коли ANN дає пробне рішення, воно не дає уявлення про те, чому і як.
- Це знижує довіру до мережі.

Забезпечення належної структури мережі:

- Немає конкретного правила для визначення структури штучних нейронних мереж.
- Відповідна структура мережі досягається шляхом досвіду та методом проб і помилок.

Складність відображення проблеми в мережі:

- ШНМ можуть працювати з числовою інформацією.
- Проблеми мають бути переведені в числові значення, перш ніж вводити їх в ANN.
- Механізм відображення, який потрібно визначити, безпосередньо впливатиме на продуктивність мережі.
- Це залежить від можливостей користувача.

Тривалість мережі невідома:

- Мережа зведена до певного значення помилки на вибірці означає, що навчання завершено.
- Значення не дає нам оптимальних результатів.

1.3 Існуючі рішення для аналізу аудіофайлів

Shazam [3] – застосунок для розпізнавання мелодії, який за ліченні секунди аналізує вхідний аудіозапис та здійснює пошук інформації про музичний твір. Shazam Entertainment, Ltd. була заснована в 2000 році з ідеєю надання послуги, яка могла б зв'язати людей музики шляхом розпізнавання музики в навколишньому середовищі за допомогою їхні мобільні телефони, щоб безпосередньо розпізнавати музику. Алгоритм повинен був мати можливість розпізнавати короткий звуковий зразок музики, яка транслювалася, змішану з важким навколишнім шумом, піддану реверберації та іншій обробці, знятий невеликим мікрофоном мобільного телефону, підданий стиснення голосового кодека та відключення мережі.

Це була важка проблема, і на той час не було відомих алгоритмів, які могли б задовольнити всі ці обмеження. Згодом інженери з програмного забезпечення в Shazam розробили власні методи, які відповідали всім експлуатаційним обмеженням [4].

Алгоритм Shazam можна використовувати в багатьох програмах, окрім просто розпізнавання музики через мобільний телефон. Завдяки здатності глибоко копатися в шумі, ми можемо розпізнати музику, приховану за гучним голосом, наприклад, в рекламі на радіо. З іншого боку, алгоритм також дуже швидкий і може використовуватися для моніторингу авторських прав зі швидкістю пошуку понад 1000 разів у реальному часі, що дозволяє скромному серверу відстежувати значно багато медіа-потоків. Алгоритм також підходить для підказки та індексування на основі вмісту для використання в бібліотеці та архіві.

Основний принцип дії алгоритму:

Кожен аудіофайл має «відбиток пальців», процес, у якому витягуються відтворювані хеш-токени. І «база даних», і «зразок» аудіофайлів піддаються одному аналізу.

Відбитки пальців з невідомого зразка порівнюються з великим набором відбитків пальців, отриманих з музичної бази даних. Збіги кандидатів згодом оцінюються на правильність відповідності. Деякі керівні принципи використання атрибутів як відбитків полягають у тому, що вони повинні бути тимчасово локалізованими, інваріантними щодо трансляції, надійними та достатньо ентропічними. Інструкція щодо тимчасової місцевості передбачає, що кожен хеш відбитків пальців обчислюється за допомогою звукових зразків поблизу відповідного моменту часу, тому що віддалені події не впливають на хеш. Аспект інваріантності трансляції означає, що хеші відбитків пальців, отримані з відповідного відповідного вмісту, відтворюються незалежно від позиції в аудіофайлі, якщо тимчасова локалізація, що містить дані, з яких обчислюється хеш, міститься у файлі. Це має сенс, оскільки невідомий зразок може виходити з будь-якої частини оригінальної звукової доріжки. Надійність означає, що хеші, згенеровані з оригінальної чистої доріжки бази даних, повинні відтворюватися з деградованої копії аудіо. Крім того, маркери відбитків пальців повинні мати достатньо високу ентропію, щоб мінімізувати ймовірність помилкових збігів маркерів у невідповідних місцях між невідомим зразком і треками в базі даних.

Недостатня ентропія призводить до надмірних і помилкових збігів у невідповідних місцях, що вимагає більшої обробної потужності для вилучення результатів, а занадто велика ентропія зазвичай призводить до крихкості та невідтворюваності маркерів відбитків пальців за наявності шуму та спотворень. Є 3 основні компоненти:

1.3.1 **Robust Constellations**.

Щоб вирішити проблему надійної ідентифікації за наявності дуже значного шуму та спотворень, ми експериментували з різноманітними можливостями, які могли б пережити кодування GSM за наявності шуму. Ми зупинилися на спектрограмних піках через їхню стійкість при наявності шуму та наближену

лінійну супердоступність [4]. Частотно-часова точка є кандидатом на пік, якщо вона має вищий вміст енергії, ніж усі її сусіди в області з центром навколо точки. Кандидатурні піки вибираються відповідно до критерію щільності, щоб гарантувати, що частотно-часова смуга для аудіофайлу має достатньо рівномірне покриття. Піки в кожній частотно-часовій місцевості також вибираються відповідно до амплітуди, з обґрунтуванням того, що піки з найвищою амплітудою, швидше за все, витримають спотворення, перераховані вище.

Таким чином, складна спектрограма, як показано на рис. 1А, може бути зведена до розрідженого набору координат, як показано на рис. 1В.

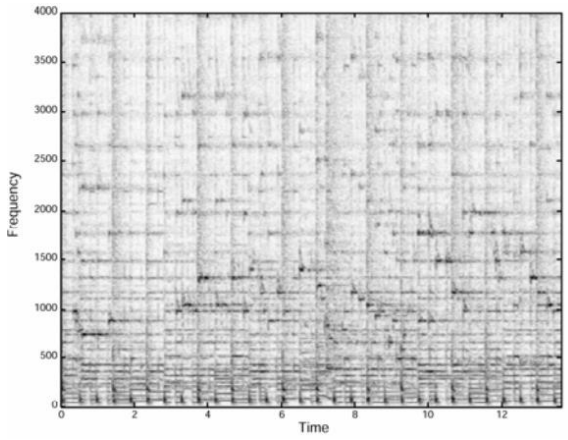

Рисунок 1А. Спектрограма

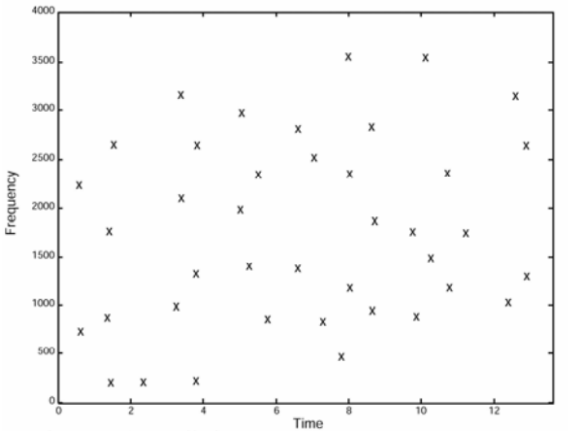

Рисунок 1Б. Карта сузір’їв

Зверніть увагу, що на цьому етапі амплітудна складова була обмежена. Це скорочення має перевагу в тому, що воно досить нечутливе до EQ, оскільки, як правило, пік у спектрі все ще є піком із тими ж координатами у відфільтрованому спектрі (за умови, що похідна функції передачі фільтра досить мала — піки поблизу різкий перехід в передатній функції є незначно зміщеними по частоті). Ми називаємо розріджені списки координат «картами сузір’їв», оскільки графіки розсіювання координат часто нагадують зоряне поле. Шаблон точок має бути однаковим для відповідності сегментів аудіо. Якщо ви помістите карту сузір’їв пісні з бази даних на стрічкову діаграму, а карту сузір’їв короткого відповідного звукового зразка тривалістю кілька секунд на прозорий шматок пластику, потім пересуньте останню поверх першої, у певний момент значний кількість точок збігатиметься, коли буде розташовано правильне зміщення часу та два карти сузір'їв вирівнюються в реєстрі. Кількість точок збігу буде значною за наявності помилкових піків, що вводяться через шум, оскільки позиції піків є відносно незалежними; крім того, кількість збігів також може бути значною, навіть якщо багато правильних очок було видалено. Таким чином, реєстрація карт сузір'їв є потужним способом зіставлення за відсутності шуму та/або видалення об'єктів. Ця процедура зводить проблему пошуку до свого роду «астронавігації», в якій невелика ділянка частотно-часових сузір’їв має бути швидко розташована в межах великого всесвіту точок у всесвіті смужкової діаграми з розмірами обмеженої частоти в порівнянні з майже мільярдом секунд у базі даних.

Нажаль, оффіційного відкритого API Shazam не має [5]. Тому більш детальний аналіз викликає складнощі.

1.3.2 Fast Combinatorial Hashing.

Знаходження правильного зміщення реєстрації безпосередньо з карт сузір’їв може бути досить повільним через необроблені точки сузір’я з низькою ентропією. Наприклад, вісь частоти з 1024 бітами дає лише не більше 10 біт

частотних даних на пік. Ми розробили швидкий спосіб індексації карт сузір'їв. Хеші відбитків пальців формуються з карти сузір'їв, на якій комбінаторно пов'язані пари час-частотних точок. Вибираються опорні точки, кожна з яких має цільову зону, пов’язану з нею. Кожна опорна точка послідовно об’єднується в пару з точками в межах цільової зони, кожна пара дає дві частотні компоненти плюс різницю в часі між точками (Рис. 1В і 1Г).

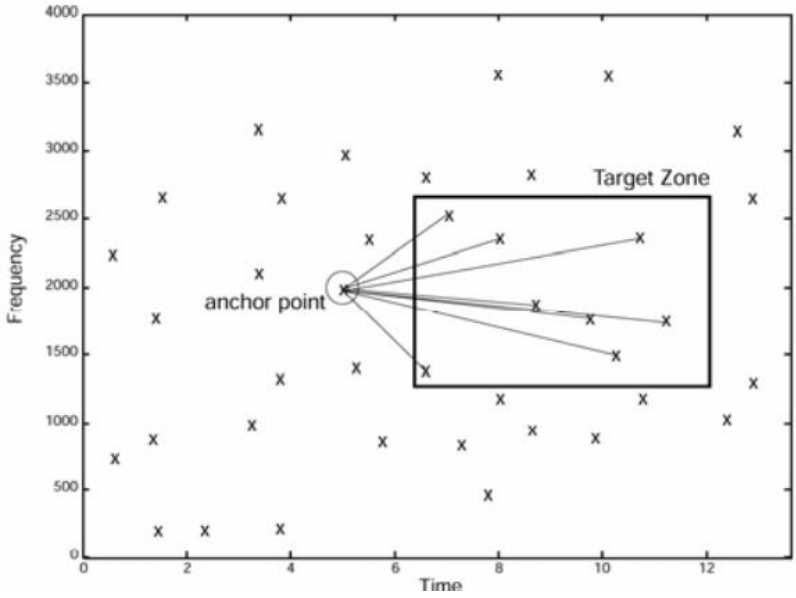

Рисунок 1В. Комбінаторне генерування хешу

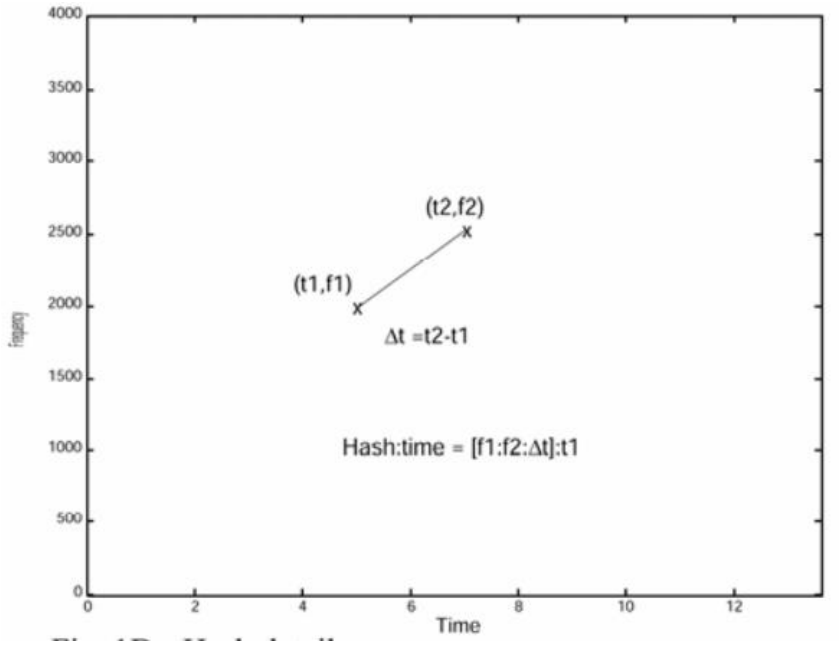

Рисунок 1Г. Деталі хешу

Ці хеші досить відтворювані, навіть при наявності шуму та стиснення голосового кодека. Крім того, кожен хеш можна упакувати в 32-розрядне ціле число без знака. Кожен хеш також пов’язаний із зміщенням часу від початку відповідного файлу до його точки прив’язки, хоча абсолютний час не є частиною самого хешу.

агрегована 64-розрядна структура, 32 біти для хеша і 32 біти для зміщення часу та ідентифікатора треку. Щоб полегшити швидку обробку, 64-розрядні структури сортуються відповідно до значення хеш-токена. Кількість хешів за секунду аудіозапису, що обробляється, приблизно дорівнює щільності точок сузір’я за секунду, помноженій на коефіцієнт розгортання в цільову зону. Наприклад, якщо кожну точку сузір'я прийняти як опорну точку, і якщо цільова зона має розгалуження розміром F=10, то кількість хешів приблизно дорівнює F=10-кратній кількості точок сузір'я, витягнутих з файл. Обмежуючи кількість точок, обраних у кожній цільовій зоні, ми прагнемо обмежити комбінаторний вибух пар. Коефіцієнт роздування безпосередньо веде до фактора витрат з точки зору простору для зберігання.

Утворюючи пари замість пошуку збігів за окремими точками сузір’я, отримуємо величезне прискорення процесу пошуку. Наприклад, якщо кожна частотна складова має 10 біт, а компонент Δt також 10 біт, то збіг пари точок дає 30 біт інформації, проти лише 10 для однієї точки. Тоді специфічність хешу була б приблизно в мільйон разів більшою через 20 додаткових бітів, і, таким чином, швидкість пошуку для одного хеш-токена так само прискорюється. З іншого боку, завдяки комбінаторному генерації хешів, припускаючи симетричну щільність і розмах як для генерації хешу бази даних, так і для зразка, існує у F разів більше комбінацій маркерів у невідомій вибірці для пошуку та у F разів більше маркерів у базі даних, таким чином, загальне прискорення становить приблизно 1000000/F2, або приблизно 10000, для пошуку маркерів на основі окремих точок сузір’я.

Зауважте, що комбінаторне хешування квадратує ймовірність виживання точки, тобто якщо p є ймовірністю виживання піку спектрограми від вихідного вихідного матеріалу до запису захопленого зразка, то ймовірність хешування від пари точок, що виживе, приблизно дорівнює p2. Це зниження живучості хешування є компромісом проти величезної кількості прискорення. Знижена ймовірність виживання окремих хешів пом’якшується комбінаторним генеруванням більшої кількості хешів, ніж вихідні точки сузір’я. Наприклад, якщо F=10, то ймовірність того, що принаймні один хеш виживе для даної точки прив’язки, буде спільним ймовірність того, що опорна точка і принаймні одна цільова точка в її цільовій зоні виживе. Якщо спрощено припустити ймовірність виживання IID p для всіх залучених точок, то ймовірність виживання принаймні одного хешування на опорну точку дорівнює p*[1-(1-p)F]. Для досить великих значень F, напр. F>10, а розумні значення p, напр. p>0,1, маємо приблизно

$$p \approx p*[1-(1-p)F] \quad (1)$$

тому нам насправді не набагато гірше, ніж раніше.

Використовуючи комбінаторне хешування, розробники продали приблизно в 10 разів більше місця для зберігання приблизно в 10 000 разів на підвищення швидкості та невелику втрату у ймовірності виявлення сигналу.

Для різних умов сигналу можна вибрати різні коефіцієнти розвіювання та щільності. Для відносно чистого звуку, напр. для додатків радіомоніторингу F може бути помірно малим, а щільність також може бути низькою, порівняно з дещо складнішою споживчою програмою для мобільних телефонів. Таким чином, різниця у вимогах до обробки може охоплювати багато порядків.

1.3.3 Searching and Scoring.

Щоб виконати пошук, вищезазначений крок відбитків пальців виконується на записаному звуковому файлі зразка, щоб створити набір записів хеш: зміщення часу. Кожен хеш із зразка використовується для пошуку відповідних хешів у базі даних. Для кожного відповідного хешу, знайденого в базі даних, відповідні часи зміщення від початку файлів вибірки та бази даних пов’язуються з парами часу. Пари часу розподіляються по ячейках відповідно до ідентифікатора треку, пов'язаного з відповідним хешем бази даних. Після того, як усі зразки хешів були використані для пошуку в базі даних, щоб утворити відповідні часові пари, бункери скануються на збіги. Усередині кожного ящика набір часових пар являє собою діаграму розсіювання асоціації між звуковими файлами зразка та бази даних. Якщо файли збігаються, функції збігу мають відбуватися з однаковими відносними зміщеннями від початку файлу, тобто послідовність хешів в одному файлі також має відображатися у відповідному файлі з такою ж відносною часовою послідовністю. Проблема визначення того, чи знайдено збіг, зводиться до виявлення значного скупчення точок, що утворюють діагональну лінію в межах діаграми розсіювання. Для виявлення можна використовувати різні методи, наприклад, перетворення Хафа або інший метод надійної регресії. Такі методи занадто загальні, дорогі з точки зору обчислень і схильні до викидів.

Через жорсткі обмеження задачі наступна методика вирішує проблему приблизно за N*log(N) часу, де N – кількість точок, що з'являються на діаграмі розсіювання. Для цілей цього обговорення ми можемо вважати, що нахил діагональної лінії дорівнює 1,0. Потім відповідні часи відповідності ознак між відповідні файли мають відношення:

$$tk' = tk + \text{зміщення}, \quad (1.1)$$

де tk' – часова координата об'єкта у відповідному (чистому) звуковому файлі бази даних, а tk – часова координата відповідного об'єкта у зразку звукового файлу, який потрібно ідентифікувати. Для кожної координати (tk',tk) на діаграмі розсіювання ми обчислюємо:

$$\delta tk = tk' - tk. \quad (1.2)$$

Потім ми обчислюємо гістограму цих значень δtk і скануємо пік. Це можна зробити шляхом сортування набору значень δtk та швидкого пошуку набору значень. Діаграми розсіювання зазвичай дуже розріджені через специфічність хешів через комбінаторний метод генерації, як обговорювалося вище. Оскільки кількість пар часу в кожному бункері невелика, процес сканування займає порядку мікросекунд на ячейку або менше. Оцінка матчу – це кількість відповідних точок на піку гістограми.

1.3.4 Шумостійкість

Алгоритм добре працює зі значними рівнями шуму і навіть нелінійними спотвореннями. Він може правильно розпізнати музику за наявності голосів, шуму від транспорту, відключення та навіть іншої музики. Щоб дати уявлення про потужність цієї техніки, із сильно пошкодженої 15-секундної вибірки можна знайти статистично значущий збіг визначається, коли лише приблизно 1-2% згенерованих хеш-токенів фактично виживають і вносять свій внесок у офсетний кластер. Властивість техніки гістограми діаграми розсіювання полягає в тому, що

розриви не мають відношення, забезпечуючи імунітет до випадань і маскування через перешкоди. Дещо дивним результатом є те, що навіть з великою базою даних ми можемо правильно ідентифікувати кожну з кількох доріжок, змішаних разом, включаючи кілька версій одного фрагмента, властивість, яку ми називаємо «прозорістю».

Зразок шуму був записаний у галасливому пабі для імітації «реальних» умов. З середини кожного тесту були взяті аудіоуривки тривалістю 15, 10 та 5 секунд треку, кожен з яких взятий з тестової бази даних.

1.3.5 Швидкість

Для бази даних близько 20 тисяч треків, реалізованих на ПК, час пошуку становить близько 5-500 мілісекунд, залежно від налаштувань параметрів і програми. Служба може знайти відповідну доріжку для сильно пошкодженого звукового зразка протягом кількох сотень мілісекунд основного часу пошуку. За допомогою звуку «радіоякості» ми можемо знайти відповідність менш ніж за 10 мілісекунд, при цьому ймовірна ціль оптимізації досягає 1 мілісекунди на запит.

1.3.6 Специфічність і хибні позитиви

Алгоритм був розроблений спеціально для цільового розпізнавання звукових файлів, які вже присутні в базі даних. Не очікується узагальнення на живі записи. Тим не менш, ми анекдотично виявили кількох артистів на концерті, які, мабуть, або мають надзвичайно точний і відтворений час (з точністю до мілісекунд), або більш правдоподібно синхронізують губи.

Алгоритм, навпаки, дуже чутливий до того, яка саме версія треку була відібрана. Враховуючи безліч різних виконання однієї і тієї ж пісні артиста, алгоритм може вибрати правильний, навіть якщо людське вухо практично не розрізняє їх.

Час від часу ми отримуємо повідомлення про помилкові результати. Часто ми бачимо, що алгоритм насправді не був неправильним, оскільки він підібрав

приклад «вибірки» або плагіату. Як згадувалося вище, існує компроміс між істинними попаданнями та помилковими спрацьовуваннями, і, таким чином, максимально допустимий відсоток помилкових спрацьовувань є параметром дизайну, який вибирається відповідно до програми.

1.4 Існуючі вільні цифрові сервіси-постачальники музики (API)

Існує кілька різних наборів даних з музичними даними — GTZan і набір даних Million Songs (MSD) є 2 найбільш часто використовуваними. Але обидва ці набори даних мають обмеження. GTZan має лише 100 пісень у кожному жанрі, а MSD має 1 мільйон пісень, але лише їхні метадані, без аудіофайлів. Було вирішено використати набір даних Free Music Archive Small. Його можна завантажити як невеликий набір даних (8 ГБ), який містить необроблені аудіофайли + метадані. У невеликому наборі даних FMA, який використовується, було рівномірно розподіллено 8 жанрів і 1000 пісень у кожному жанрі. Вісім жанрів: електронний, експериментальний, фолк, хіп-хоп, інструментальний, інтернаціональний, поп і рок.

Після того, як ми підготували наш набір даних, ми хочемо, щоб навчити нашу модель. Музика – це часовий ряд даних. Це означає, що музика є лінійною в часі. LSTM досить добре справляються із вилученням патернів у просторі вхідних ознак, коли вхідні дані охоплюють довгі послідовності. Враховуючи архітектуру LSTM, яка дозволяє маніпулювати станом пам'яті, вони ідеально підходять для вирішення таких завдань.

Також гідні згадки такі дата-сети як:

Ballroom. Набір даних (Cano et al., 2006) складається із чітких та постійних ритмічних патернів, що робить його придатним для завдання розпізнавання.

Розширений Ballroom. Набір даних (Marchand and Peeters, 2016) був запропонований у 2016 році Марчандом, який розширив оригінальний

набір даних Ballroom. Порівняно з оригіналом, розширена версія розширена версія містить у шість разів більше треків з найкращою якістю звуку.

РОЗДІЛ 2. МОДЕЛІ ТА МЕТОДИ РОЗВ'ЯЗАННЯ ЗАДАЧІ.

2.1 Попередня обробка даних

Набір даних, який використовувується у роботі, завантажений із сайту Kaggle та містить близько 100 пісень під кожним із 10 ярликів або жанрів. Кожна з пісень триває 30 секунд. Як згадувалося раніше, такий обсяг даних значно менший для навчання LSTM-моделі. Щоб вирішити цю проблему, розділимо кожен аудіофайл на 10 сегментів, кожен із яких триває 3 секунди. Таким чином, кількість пісень під кожною міткою тепер становить 1000, що є пристойним числом для навчання моделі та досягнення гарної точності.

Тепер, коли є готові дані, нам потрібно отримати ознаки, які будуть використовуватися в нашій мережі. Для отримання ознак ми будемо використовувати MFCC. Librosa використовується для отримання ознак з кожного аудіофрагменту. Створюємо словник з міткою або категорією жанру як ключ і всі витягнуті ознаки з усіх 1000 сегментів як масив ознак під цією міткою. Після того як цю процедуру повторять в циклі для всіх 10 категорій, отриманні результати скинемо у файл JSON. Цей JSON-файл і стане набором даних, на якому навчатиметься модель. Переходячи до кодування для попередньої обробки набору даних, спочатку визначаємо кількість сегментів і частоту дискретизації кожного сегмента. Частота дискретизації необхідна для того, щоб дізнатися про швидкість відтворення пісні. Тут ми зберігаємо її постійною кожному за сегмента.

Отже, у отриманому масиві записано цифрове уявлення звукового сигналу у часовій області. Тобто, потрібно відомості про те, як змінювалася амплітуда сигналу з часом. У 19 столітті Жан Батіст Джозеф Фур'є зробив визначне відкриття. Полягає воно в тому, що будь-який сигнал у часовій області еквівалентний сумі деякої кількості (можливо, нескінченного) простих синусоїдальних сигналів, за умови, що кожна синусоїда має певну частоту,

амплітуду та фазу. Набір синусоїд, які формують вихідний сигнал, називають поряд Фур'є.

Іншими словами, можна уявити практично будь-який сигнал, розгорнутий у часі, просто задавши набір частот, амплітуд та фаз, що відповідають кожній із синусоїд, які цей сигнал формують. Таке уявлення сигналів називають набором частотних інтервалів. У певному сенсі, відомості про частотні інтервали є чимось на кшталт «відбитків пальців» або сигнатур сигналів, розгорнутих у часі, даючи нам статичне уявлення динамічних даних(рис.2.1).

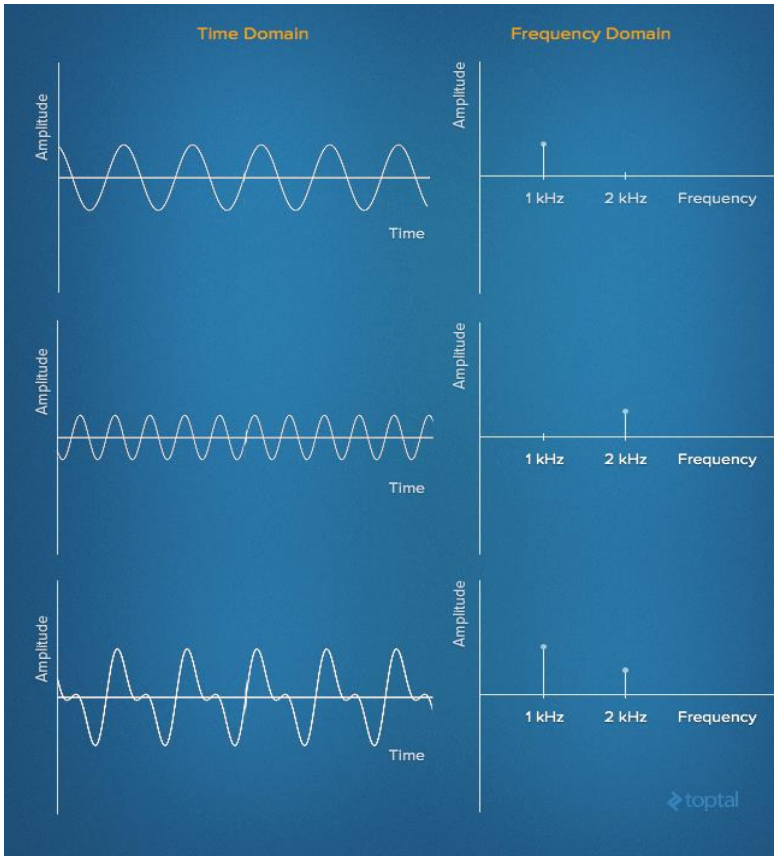

Рис.2.1 Сигнали, розгорнуті в часі, та їх частотні характеристики

Отже, необхідно визначити спосіб отримання частотних показників сигналів, розгорнутих у часі. У цьому допоможе дискретне перетворення Фур'є (ДПФ, DFT, Discrete Fourier Transform). ДПФ це математичний метод аналізу Фур'є для дискретних сигналів. З його допомогою можна перетворити кінцевий набір

зразків сигналу, взятих з рівними проміжками часу, в список коефіцієнтів кінцевої комбінації комплексних синусоїд, упорядкованих за частотою, враховуючи, що ці синусоїди були дисретизовані з однією частотою.

Один із найпопулярніших чисельних алгоритмів для обчислення ДПФ називається швидке перетворення Фур'є (БПФ, FFT, Fast Fourier Transformation). Насправді БПФ представлений цілим набором алгоритмів. Серед них найчастіше використовуються варіанти алгоритму Кулі-Тьюкі (Cooley-Tukey). В основі цього алгоритму лежить принцип «поділяй і владарюй». У результаті обчислень використовується рекурсивне розкладання вихідного ДПФ на дрібні частини. Пряме обчислення ДПФ для деякого набору даних n вимагає O(n2) операцій, а використання алгоритму Кулі-Тьюкі дозволяє вирішити ту ж задачу за O(n log n) операцій(рис.2.2).

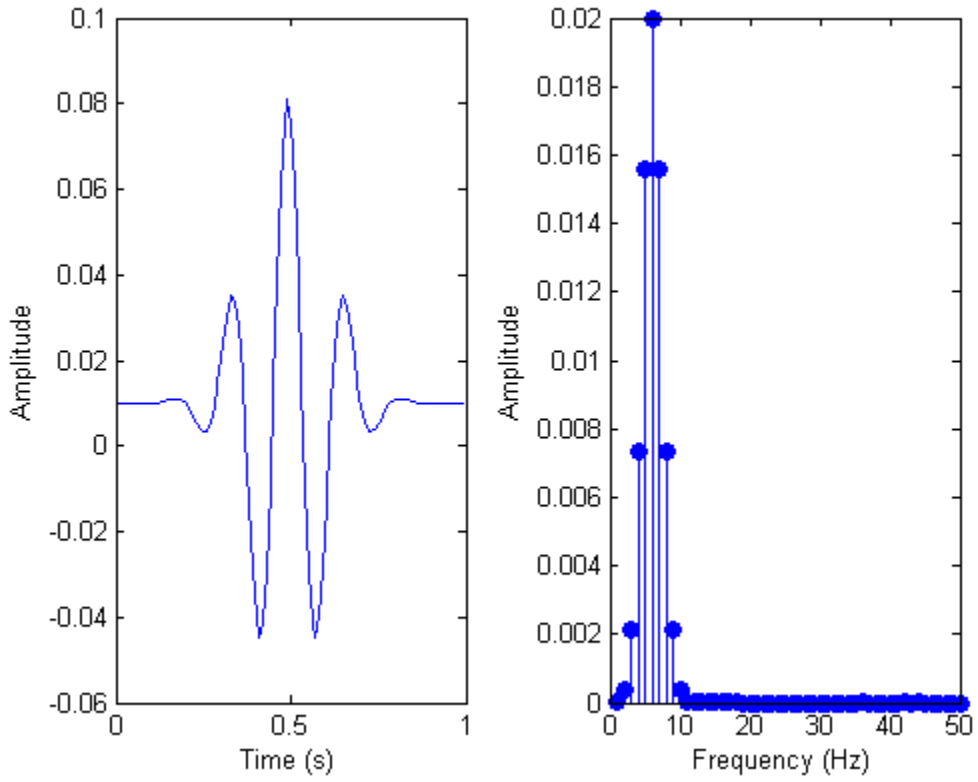

Рис.2.2 Сигнал до та після БПФ-аналізу

Так як задача розподілення музики на жанри є задачею класифікації то слід використовувати такі моделі нейронних мереж як рекурентні нейронні мережі (Recurrent neural network; RNN) та згорткові нейронні мережі(convolutional neural network, CNN).

У даному проекті було прийнято рішення про комбінацію обох цих підходів проектування нейронних мереж.

2.2 LSTM

LSTM - це архітектура рекуррентної нейронної мережі (RNS), яка запам'ятовує значення для випадкових проміжних мереж часу. LSTM добре підходить для класифікації, обробки та прогнозування часових рядів з урахуванням тимчасових задержек невідомої тривалості. Відносна нечутливість до довжини промежутка дає перевагу LSTM перед альтернативними РНС, скритими моделями Маркова та іншими методами навчання послідовності.

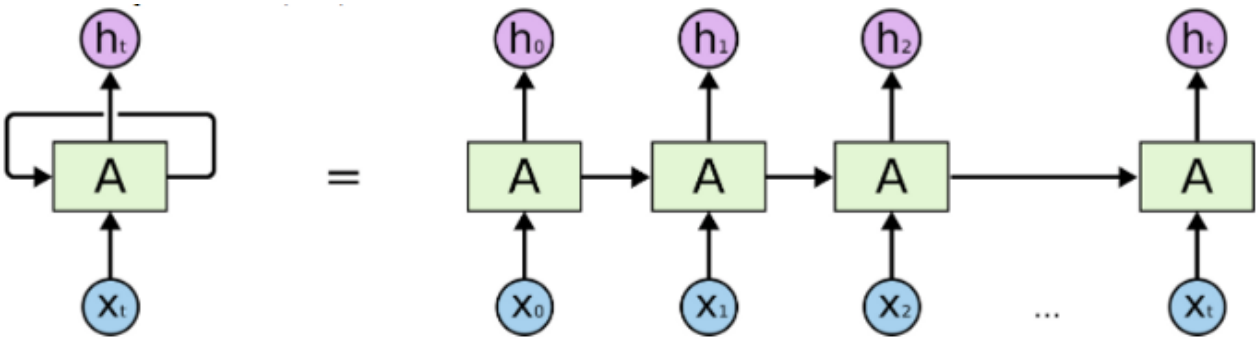

Рис. 2.3 Структура RNN

Структура RNN дуже подібна до прихованої марківської моделі. Однак основна відмінність полягає в тому, як розраховуються та будуються параметри. Однією з переваг LSTM є нечутливість до довжини проміжку. RNN та HMM покладаються на прихований стан перед викидом/послідовністю. Якщо ми хочемо передбачити послідовність через 1000 інтервалів замість 10, модель забуде початкову точку на той час.

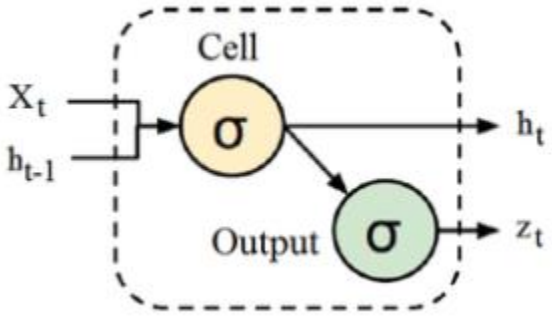

Рис. 2.4 RNN комірка

Комірка RNN(рис. 2.4) приймає два входи, вихід з останнього прихованого стану та спостереження в момент = t. Крім прихованого стану, немає інформації про минуле, яку можна пам’ятати.

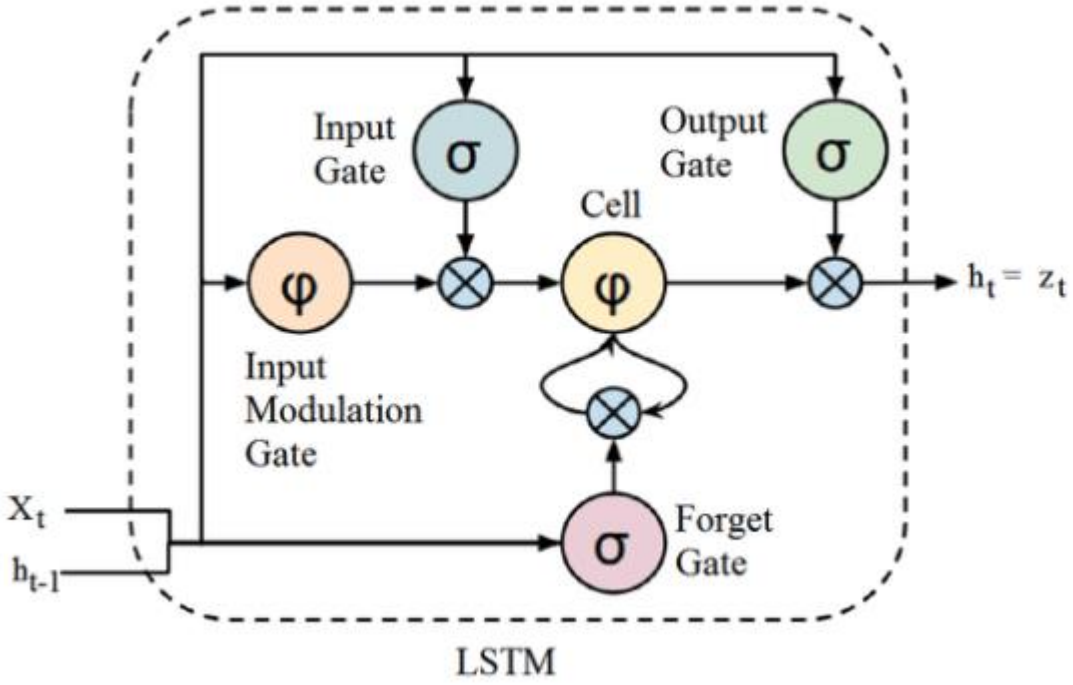

Рис. 2.5 LSTM комірка

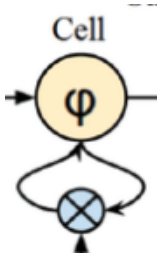

Рис. 2.6 Стан комірки

Довгострокова пам'ять зазвичай називається станом клітини (рис. 2.6). Циклічні стрілки вказують на рекурсивну природу осередку. Це дозволяє зберігати в осередку LSTM інформацію з попередніх інтервалів. Стан осередку змінюється затвором забування, розташованим нижче стану осередку, а також регулюється затвором модуляції входу. З рівняння випливає, що попередній стан осередку забувається шляхом перемноження зі стробом забування та додається нова інформація через вихід вхідних стробів.

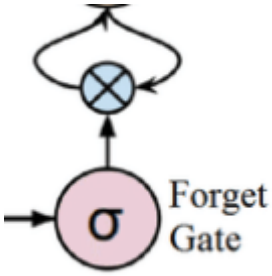

Рис. 2.7 Межа забування

Вектор запам'ятовування зазвичай називають межею забування (forget gate). Вихід стробу забування повідомляє стан клітини, яку інформацію слід забути, помножуючи 0 на позицію в матриці. Якщо вихід строба забування дорівнює 1 інформація зберігається в стані клітини. З рівняння(2.1) випливає, що сигмоїдна функція застосовується до зваженого входу/спостереження та попереднього прихованого стану.

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_t\right) \tag{2.1}$$

Вектор збереження зазвичай називають вхідною межею. Ц межа визначає, яка інформація повинна увійти в стан клітини/довготривалу пам'ять. Важливими частинами є функції активації для кожного воріт. Вхідний вентиль є сигмовидною функцією і має діапазон [0,1]. Оскільки рівняння стану клітинки є підсумовуванням між попереднім станом комірки, сама сигмовидна функція лише додає пам’ять і не зможе видалити/забути пам’ять. Якщо ви можете додати

лише число з плаваючою точкою між [0,1], це число ніколи не буде нульовим/вимкненим/забутим.

$$\mathbf{i}_t = \sigma\left(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i\right) \quad (2.2)$$

Вектор фокусування зазвичай називають вихідною межею. Робочу пам'ять зазвичай називають прихованим станом.

2.3 Згортка нейронна мережа [6]

CNN відрізняються від інших нейронних мереж чудовою продуктивністю при роботі із зображеннями, мовленням або аудіосигналами. Вони мають три основних типи шарів, а саме:

- Конволюційний шар
- Шар об'єднання
- Повністю пов'язаний шар (FC)

Конволюційний шар – це перший шар конволюційної мережі. За конволюційними шарами можуть йти додаткові конволюційні шари або шари об'єднання, але останнім шаром є повністю пов'язаний шар. З кожним шаром складність CNN зростає, ідентифікуючи дедалі більші частини зображення. Попередні шари фокусуються на простих характеристиках, таких як кольори та краї. У міру проходження даних зображення через шари CNN, вона починає розпізнавати більші елементи або форми об'єкта, поки, нарешті, не ідентифікує передбачуваний об'єкт.

2.3.1 Конволюційний шар

Конволюційний шар є основним структурним блоком CNN, і у ньому відбувається більшість обчислень. Для його роботи потрібно кілька компонентів: вхідні дані, фільтр та карта ознак. Припустимо, що вхідними даними буде кольорове зображення, яке складається із матриці пікселів у 3D. Це означає, що

вхідні дані будуть мати три виміри - висоту, ширину та глибину, які відповідають RGB у зображенні. У нас також є детектор ознак, також відомий як ядро або фільтр, який переміщається по рецептивним полям зображення, перевіряючи, чи є ознака. Цей процес відомий як згортка.

Детектор ознак є двовимірним (2-D) масив ваг, який являє собою частину зображення. Хоча їх розмір може змінюватись, розмір фільтра зазвичай являє собою матрицю 3x3; це також визначає розмір рецептивного поля. Потім фільтр застосовується до області зображення, і обчислюється точковий добуток між вхідними пікселями та фільтром. Потім цей точковий твір надходить у вихідний масив. Після цього фільтр зсувається на рядок, повторюючи процес доти, доки ядро не охопить все зображення. Кінцевий результат серії точкових добутків вхідного сигналу та фільтра відомий як карта ознак, карта активації або згорнута карта ознак.

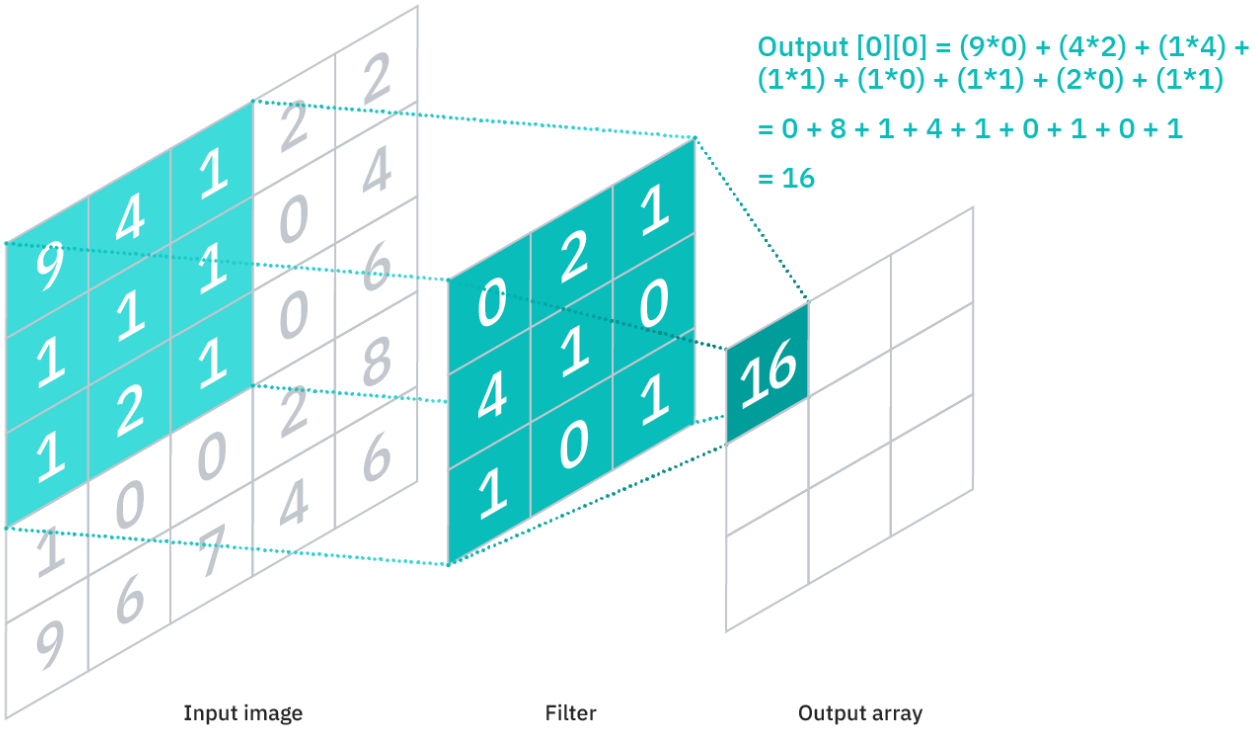

Рис. 2.7 Приклад роботи CNN

Як ви можете бачити на зображенні вище(рис 2.7), кожне вихідне значення на карті об’єктів не обов’язково має з’єднуватися з кожним значенням пікселя у вхідному зображенні. Його потрібно лише підключитися до сприйнятливого поля, де застосовується фільтр. Оскільки вихідний масив не потребує безпосереднього відображення кожного вхідного значення, згорткові (і об’єднані) шари зазвичай називають «частково зв’язаними» шарами. Однак цю характеристику можна також описати як локальну зв’язність.

Зауважте, що ваги в детекторі функцій залишаються фіксованими, коли він рухається по зображенню, що також відоме як спільне використання параметрів. Деякі параметри, як-от значення ваги, коригуються під час навчання за допомогою процесу зворотного поширення та градієнтного спуску. Проте є три гіперпараметри, які впливають на розмір виводу, який необхідно встановити перед початком навчання нейронної мережі. До них належать:

1. Кількість фільтрів впливає на глибину виводу. Наприклад, три різні фільтри дадуть три різні карти об’єктів, створюючи глибину трьох.

2. Stride – це відстань або кількість пікселів, на які ядро переміщається по вхідній матриці. Хоча значення кроку два або більше зустрічається рідко, більший крок дає менший результат.

3. Нульовий заповнення зазвичай використовується, коли фільтри не підходять до вхідного зображення. Це обнуляє всі елементи, які виходять за межі вхідної матриці, створюючи більший або однаковий за розміром вихід. Існує три типи підкладки:

- Дійсний відступ: це також відоме як відсутність заповнення. У цьому випадку остання згортка скидається, якщо розміри не вирівнюються.
- Таке заповнення: це заповнення гарантує, що вихідний шар має той самий розмір, що і вхідний шар

- Повне заповнення: цей тип заповнення збільшує розмір виводу, додаючи нулі до межі вхідних даних.
- Після кожної операції згортки CNN застосовує перетворення Rectified Linear Unit (ReLU) до карти ознак, вносячи нелінійність у модель.

Як було описано раніше, інший шар згортки може слідувати за початковим шаром згортки. Коли це станеться, структура CNN може стати ієрархічною, оскільки пізні рівні можуть бачити пікселі в сприйнятливих полях попередніх шарів. Як приклад, припустимо, що ми намагаємося визначити, чи містить зображення велосипед. Ви можете думати про велосипед як про суму частин. Він складається з рами, керма, коліс, педалей тощо. Кожна окрема частина велосипеда складає шаблон нижнього рівня в нейронній мережі, а комбінація його частин представляє шаблон вищого рівня, створюючи ієрархію функцій у CNN.

2.3.2 Об’єднаний шар

Об’єднання шарів, також відоме як зниження дискретизації, зменшує розмірність, зменшуючи кількість параметрів у вхідних даних. Подібно до згорткового шару, операція об’єднання об’єднує фільтр по всьому входу, але різниця в тому, що цей фільтр не має жодних ваг. Замість цього ядро застосовує функцію агрегації до значень у прийнятному полі, заповнюючи вихідний масив. Існує два основних типи об’єднання:

- Максимальне об’єднання: під час переміщення фільтра по входу він вибирає піксель із максимальним значенням для відправки на вихідний масив. Крім того, цей підхід, як правило, використовується частіше в порівнянні зі середнім об’єднанням.

- Середній пул: коли фільтр переміщується по вхідному сигналу, він обчислює середнє значення в поле сприйняття для надсилання до вихідного масиву.

Хоча багато інформації втрачається на рівні об’єднання, це також має ряд переваг для CNN. Вони допомагають зменшити складність, підвищити ефективність та обмежити ризик переобладнання.

2.3.3 Повністю підключений шар

Назва повнозв’язного шару влучно описує себе. Як згадувалося раніше, значення пікселів вхідного зображення не пов’язані безпосередньо з вихідним шаром у частково з’єднаних шарах. Однак у повністю підключеному шарі кожен вузол вихідного шару підключається безпосередньо до вузла попереднього шару.

Цей шар виконує завдання класифікації на основі ознак, витягнутих через попередні шари та їх різні фільтри. У той час як рівні згортки та об’єднання, як правило, використовують функції ReLu, рівні FC зазвичай використовують функцію активації softmax для належної класифікації вхідних даних, створюючи ймовірність від 0 до 1.

2.4. Згорткова рекурентна модель

Спектограма – це візуальне представлення звуку в частотному та часовому вимірі. CNN має сенс, оскільки спектрограми пісні схожі на зображення, кожна з яких має свої індивідуальні шаблони. RNN чудово розуміють послідовні дані, роблячи прихований стан в момент t залежним від прихованого стану в момент t-1. Спектограми мають часову складову, і RNN можуть набагато краще визначити короткострокові та довгострокові тимчасові особливості в пісні.

У цій моделі використовуються одномірні CNN, які виконують операцію згортки тільки по временному виміру. Кожен шар свертки 1D виявляє особливості з невеликого фрагмента спектрограми. Активація RELU застосовується після

операції свертки. Виповнюється пакетна нормалізація та, нарешті, 1D Max Pooling, яка зменшує просторовий розмір зображення та попередньо повертається надмірну підгонку.

Ця цепочка операцій - 1D Convolution - RELU - Пакетна нормалізація - 1D Max Pooling виконується 3 рази. Вихід зі слоя 1D-конволюції представлений у LSTM, який повинен знайти короткострокову та довгострокову структуру пісень. LSTM використовує 96 скритих одиниць. Вихід з LSTM передается в щільний шар з 64 одиниць. Останній вихідний шар моделі являє собою щільний шар із активацією Softmax і 8 скритими одиницями для присвоєння ймовірності 8 класів. Для зменшення споживчої підгонки моделі між усіма слоями використовувалися як відсев, так і L2-регулятор. На рисунку (рис 2.8) нижче показана загальна архітектура моделі.

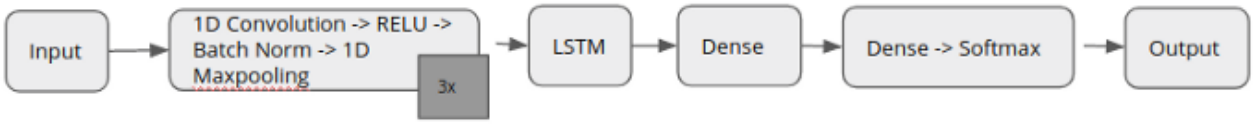

Рис 2.8. Загальна архітектура нейронної мережі

РОЗДШЛ 3

РОЗРОБКА НЕЙРОННОЇ МЕРЕЖІ ЖАНРОВОЇ КЛАСИФІКАЦІЇ МУЗИКИ

Слідуючи кращим практикам розробки ПЗ код програми було розроблено згідно парадигми ООП на мові Python. З допомогою таких бібліотек як:

- TensorFlow [7] – це наскрізна платформа з відкритим вихідним кодом для машинного навчання. Вона має велику, гнучку екосистему інструментів, бібліотек та ресурсів спільноти, яка дозволяє дослідникам просувати передові досягнення в області ML, а розробникам - легко створювати та впроваджувати програми на базі ML
- Librosa [8] – це пакет Python для аналізу музики та аудіо. Він надає будівельні блоки для створення структур, які допомагають отримувати інформацію про музику.
- NumPy [9] - open-source модуль для python, який надає загальні математичні та числові операції у вигляді пре-компільованих, швидких функцій. Вони об'єднуються у високорівневі пакети. Вони забезпечують функціонал, який можна порівняти із функціоналом MatLab. NumPy (Numeric Python) надає базові методи для маніпуляції з великими масивами та матрицями. SciPy (Scientific Python) розширює функціонал numpy величезною колекцією корисних алгоритмів, таких як мінімізація, перетворення Фур'є, регресія та інші прикладні математичні техніки.
- Matplotlib [10] – це комплексна бібліотека для створення статичних, анімованих та інтерактивних візуалізацій мовою Python. Matplotlib робить легкі речі простими, а важкі – можливими.

У якості інегрованого середовища розробки для мови Python було обрано PyCharm. Ажде він маж багато переваг:

PyCharm робить розробку максимально продуктивною завдяки функціям автодоповнення та аналізу коду, миттєвому підсвічуванню помилок та швидким виправленням. Автоматичні рефакторинги допомагають ефективно редагувати код, а зручна навігація дозволяє миттєво переміщатися за проектом.

Допомога під час редагування

Розумний редактор PyCharm призначений для максимально продуктивної розробки на Python, JavaScript, CoffeeScript, TypeScript, CSS та популярних мовах шаблонів. Функції автодоповнення, виявлення помилок та швидкі виправлення враховують особливості кожної з мов, що підтримуються.

Зручна навігація

Розумний пошук дозволяє швидко перейти до будь-якого класу, файлу або символу, а також потрібного вікна або дії IDE. Перехід до вищестоящого методу, тесту, оголошення, входження або реалізації здійснюється в одне натискання.

Швидкі та безпечні рафакторинги

PyCharm надає широкі можливості реорганізації коду за допомогою рефакторингів Rename та Delete, Extract Method, Introduce Variable, Inline Variable, Inline Method та багатьох інших. Рефакторинги враховують особливості конкретної мови або фреймворку, допомагаючи вносити зміни по всьому проекту.

Перш за все необхідно підключити усі необхідні модулі

```
import json
import os
from sklearn.model_selection import train_test_split
import tensorflow.keras as keras
import math
import librosa
import matplotlib.pyplot as plt
import numpy as np
```

Рис.3.1 Необхідні модулі

Далі потрібно явно оголосити глобальні константи.

```
DATASET_PATH = "./dataset"
JSON_PATH = "outputs.json"

RATE = 19999
TRACK_D = 30
S_PER_T = RATE * TRACK_D

PADDING = 'same'
ACTIVATION = 'relu'
LOSS = 'sparse_categorical_crossentropy'
METTRICS =['accurcy']
```

Рис.3.2 Глобальні константи

3.1 Розробка класу для перетворення вхідних даних на тренувальну вибірку

Отже, перед тим як розробляти кодову базу для нейронної мережі потрібно підготувати набір даних для найбільш ефективного навчання мережі.

Тому було вирішено розділити логіку на 2 класи.

Класс DatasetPreporator виконує функцію перетворення звичнайних данних у формат MFCC - похідні від типу кепстрального представлення аудіокліпу (нелінійний «спектр спектру»). Різниця між кепстром і кепстром mel-частоти полягає в тому, що в MFC смуги частот рівномірно розташовані за шкалою mel, що наближає реакцію слухової системи людини більш точно, ніж лінійно-рознесені смуги частот, які використовуються в нормальному спектрі. Таке перекос частоти може дозволити краще відображати звук, наприклад, під час стиснення звуку.

```
class DatasetPreparator:
    def __init__(self, dataset_path, json_path, num_of_traning_inputs=13, n_fft=2049, hop_length=522, amount_of_segments=6):
            self.num_of_traning_inputs = num_of_traning_inputs
            self.amount_of_segments =amount_of_segments;
            self.hop_length= hop_length;
            self.json_path = json_path;
            self.n_fft=n_fft
            self.dataset_path = dataset_path;
            self.data = {
                "genre_labels": [],
                "traning_inputs": [],
                "outputs": []
            }
```

Рис.3.3 Конструктор ініціалізує необхідні для подальшою роботи змінні

Для того щоб не перетворити кожен аудіофайл треба ініціалізувати кладений цикл, що обійде усі жанрові папки та усі аудіофайли в них. Для цьйого існух базовий/стандартний модуль для роботи з файловою системою Python - os.

Після перетворення у MFCC формат аудіофайли для зручності записуємо у окремий файл у популярному форматі JSON.

```
def load_and_transform_audio(self):
        samples_per_segment = int(S_PER_T / self.amount_of_segments),
        num_mfcc_vectors_per_segment = math.ceil(samples_per_segment / self.hop_length),
        for i, (directory_path, directory_names, filenames) in enumerate(os.walk(self.dataset_path)):
            if directory_path is not self.dataset_path:
                semantic_label = directory_path.split("/")[-1]
                self.data["genre_labels"].append(semantic_label)

                for filename in filenames:

                    file_path = os.path.join(directory_path, filename)
                    signal, sample_rate = librosa.load(file_path, sr=RATE)

                    for d in range(self.amount_of_segments):
                        start = samples_per_segment * d
                        finish = start + samples_per_segment

                        mfcc = librosa.feature.mfcc(signal[start:finish], sample_rate, n_mfcc=self.num_of_traning_inputs, n_fft=self.n_fft,
                                                    hop_length=self.hop_length)
                        mfcc = mfcc.T
                        if len(mfcc) == num_mfcc_vectors_per_segment:
                            self.data["traning_inputs"].append(mfcc.tolist())
                            self.data["outputs"].append(i - 1)

def save_the_outputs_in_json(self):
    with open(self.json_path, "w") as fp:
        json.dump(self.data, fp, indent=4)
```

Рис.3.4 Функції перетворення та збереження

3.2 Розробка класу нейронної мережі

Фуннкція prepape_dataset яка розділяє усі аудіофайли на тренувальну та перевірочну(20%) частини.

```
class NN:
    def __init__(self, input_data_path):
        self.input_data_path = input_data_path;
        self.X_axs = [];
        self.y_axs = [];

    def load_dataset_from_json(self):
        with open(self.input_data_path, "r") as fp:
            data = json.load(fp)
        self.X_axs = np.array(data["traning_inputs"])
        self.y_axs = np.array(data["outputs"])




    def prepare_datasets(self, test_size, validation_size):
        self.X_train = X_train
        self.X_validation = X_validation
        self.X_test = X_test
        self.y_train = y_train
        self.y_validation = y_validation
        self.y_test = y_test

        self.input_shape = (X_train.shape[1], X_train.shape[2], 1)
```

Рис.3.5 Перша частина класу NN

Скритий шар нейроннох мережі що розробляєтся складатимется з 3 колонок штучних нейронів, по 65 у кожному. Та вихідний шар з 64 штучних нейронів.

Після генерациі усіх шарва модель потрібно скомпілювати для отримання результатів з тренувальних даних.

```
def build_model(self):
    Sequential = keras.Sequential()

    for i in range(3):
        conv = keras.layers.Conv2D(65, (4, 4), ACTIVATION, self.input_shape)
        max_pooling = keras.layers.MaxPooling2D((4, 4),(2, 2), PADDING)
        batch = keras.layers.BatchNormalization()

        Sequential.add(conv)
        Sequential.add(max_pooling)
        Sequential.add(batch)

    flat_vector = keras.layers.Flatten()
    dense = keras.layers.Dense(64,ACTIVATION)
    dropout = keras.layers.Dropout(0.3)
    dense_2 = keras.layers.Dense(10, 'softmax')

    Sequential.add(flat_vector)
    Sequential.add(dense)
    Sequential.add(dropout)
    Sequential.add(dense_2)

    self.model = Sequential

def compileModal(self):
    optimiser = keras.optimizers.Adam(learning_rate=0.0001)
    self.model.compile(optimizer, LOSS, METTRICS)
    self.model.summary()
```

Рис.3.6 Друга частина класу NN

3.3 Функція – точка старту/початку

Після визначення необхідних класів створюємо послідовно їх екземпляри = спочатку DatasetPreparator, потім NN.

```
if __name__ == "__main__":
        datasetPreparator = DatasetPreparator(DATASET_PATH, JSON_PATH)
        NN = NN(JSON_PATH)

        datasetPreparator.load_and_transform_audio()
        datasetPreparator.save_the_outputs_in_json()

        NN.prepare_datasets(0.24, 0.3)
        NN.load_dataset_from_json()
        NN.build_model()
        NN.compileModal()
        NN.train_model()
```

Рис.3.7 Головна функція

ВИСНОВКИ

Нейронні мережі нині все набувають все більшої актуальності адже вони вирішують низку складних нелінійних задач.

Комерційні проекти аналізу музики такі як Spotify, Shazam, Youtube.Music, змагаються за слухачів, розробляючи все нові й нові засоби проектування та розробки нейроних мереж для аналізу, розпізнавання, класифікації музики.

Висвітлено проблему категорізації жанрової музики та її сучасні рішення у світчі.

Розроблено згортну нейронну мережу для аналізу та категорізації 8 Гігабайт музики з 10 жанрів.

Перевагою інформаційної системи є її простотота та відносна шумостійкість.

Розробленну нейронну мережу можливо застосовувати для категорізації музики на жанри. При цьому чим більше вибірка тим більше

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stanford University. State of AI in 10 Charts URL: https://hai.stanford.edu/news/state-ai-10-charts (дата звернення: 10.10.2021).
2. Sciencedirect. Reviewed ANN applications framework URL: https://www.sciencedirect.com/science/article/pii/S2405844018332067#fig7 (дата звернення: 10.10.2021).
3. Сторінка Shazam на Wikipedia URL: https://uk.wikipedia.org/wiki/Shazam (дата звернення: 11.10.2021)
4. Method for search in an audio database URL: https://patentscope.wipo.int/search/en/detail.jsf?docId=WO2002011123 (дата звернення: 12.10.2021)
5. Quora - “Is there an API for Shazam or SoundHound?” URL: https://www.quora.com/Is-there-an-API-for-Shazam-or-SoundHound (дата звернення: 12.10.2021)
6. IBM Convolutional Neural Networks URL: https://www.ibm.com/cloud/learn/convolutional-neural-networks#toc-how-do-con--z4UwR2M
7. TensorFlow URL: https://www.tensorflow.org/
8. Librosa URL: https://librosa.org/
9. NumPy URL: https://numpy.org/
10. Matplotlib URL: https://matplotlib.org/

ДОДАТОК А

		Позначення	Найменування	Кільк. аркушів	Примітка
1					
2			Документація		
3					
4		**ІТКІ.КР 19.03.ДА.ПЗ**	Пояснювальна записка		
5					
6			Презентація		
7					
8			Диск CD з презентацією	1	

Зм.	Ар-куш	№ докум	Підпис	Дата	**ІТКІ.КР 19.03.ДА.ПЗ**			
Розроб.		О.Д. Ястребцев			**Матеріали кваліфікаційної роботи**	Літ.	Аркуш	Аркушів
						Н	1	1
Керівник		І.М. Гаркуша				НТУ «ДП», 126м-20-1		
Рецензент								
Н.контр.		Г.М. Коротенко						
Зав.каф.		В.В.Гнатушенко						

Додаток Б

Програмний код (лістинг) компонентів застосунку

main.py

```
import json

import os

from sklearn.model_selection import train_test_split

import tensorflow.keras as keras

import math

import librosa

import matplotlib.pyplot as plt

import numpy as np


DATASET_PATH = "./dataset"

JSON_PATH = "outputs.json"


RATE = 19999

TRACK_D = 30

S_PER_T = RATE * TRACK_D


PADDING = 'same'

ACTIVATION = 'relu'

LOSS = 'sparse_categorical_crossentropy'

METTRICS =['accurcy']


class DatasetPreparator:

    def __init__(self, dataset_path, json_path, num_of_traning_inputs=13, n_fft=2049, hop_length=522, amount_of_segments=6):

        self.num_of_traning_inputs = num_of_traning_inputs

        self.amount_of_segments =amount_of_segments;
```

```
        self.hop_length= hop_length;
        self.json_path = json_path;
        self.n_fft=n_fft
        self.dataset_path = dataset_path;
        self.data = {
            "genre_labels": [],
            "traning_inputs": [],
            "outputs": []
        }
    def load_and_transform_audio(self):
        samples_per_segment = int(S_PER_T / self.amount_of_segments),
        num_mfcc_vectors_per_segment = math.ceil(samples_per_segment / self.hop_length),
        for i, (directory_path, directory_names, filenames) in enumerate(os.walk(self.dataset_path)):
            if directory_path is not self.dataset_path:
                semantic_label = directory_path.split("/")[-1]
                self.data["genre_labels"].append(semantic_label)

                for filename in filenames:

                    file_path = os.path.join(directory_path, filename)
                    signal, sample_rate = librosa.load(file_path, sr=RATE)

                    for d in range(self.amount_of_segments):
                        start = samples_per_segment * d
                        finish = start + samples_per_segment

                        mfcc = librosa.feature.mfcc(signal[start:finish], sample_rate, n_mfcc=self.num_of_traning_inputs, n_fft=self.n_fft,
                                                    hop_length=self.hop_length)
                        mfcc = mfcc.T
```

```
                    if len(mfcc) == num_mfcc_vectors_per_segment:
                        self.data["traning_inputs"].append(mfcc.tolist())
                        self.data["outputs"].append(i - 1)

    def save_the_outputs_in_json(self):
        with open(self.json_path, "w") as fp:
            json.dump(self.data, fp, indent=4)



class NN:
    def __init__(self, input_data_path):
        self.input_data_path = input_data_path;
        self.X_axs = [];
        self.y_axs = [];

    def load_dataset_from_json(self):
        with open(self.input_data_path, "r") as fp:
            data = json.load(fp)
        self.X_axs = np.array(data["traning_inputs"])
        self.y_axs = np.array(data["outputs"])




    def prepare_datasets(self, test_size, validation_size):
        self.X_train = X_train
        self.X_validation = X_validation
        self.X_test = X_test
        self.y_train = y_train
```

```
    self.y_validation = y_validation
    self.y_test = y_test

    self.input_shape = (X_train.shape[1], X_train.shape[2], 1)



def build_model(self):
    Sequential = keras.Sequential()

    for i in range(3):
        conv = keras.layers.Conv2D(65, (4, 4), ACTIVATION, self.input_shape)
        max_pooling = keras.layers.MaxPooling2D((4, 4),(2, 2), PADDING)
        batch = keras.layers.BatchNormalization()

        Sequential.add(conv)
        Sequential.add(max_pooling)
        Sequential.add(batch)

    flat_vector = keras.layers.Flatten()
    dense = keras.layers.Dense(64,ACTIVATION)
    dropout = keras.layers.Dropout(0.3)
    dense_2 = keras.layers.Dense(10, 'softmax')

    Sequential.add(flat_vector)
    Sequential.add(dense)
    Sequential.add(dropout)
    Sequential.add(dense_2)

    self.model = Sequential
```

```
    def compileModal(self):
        optimiser = keras.optimizers.Adam(learning_rate=0.0001)
        self.model.compile(optimizer, LOSS, METTRICS)
        self.model.summary()


if __name__ == "__main__":
        datasetPreparator = DatasetPreparator(DATASET_PATH, JSON_PATH)
        NN = NN(JSON_PATH)

        datasetPreparator.load_and_transform_audio()
        datasetPreparator.save_the_outputs_in_json()

        NN.prepare_datasets(0.24, 0.3)
        NN.load_dataset_from_json()
        NN.build_model()
        NN.compileModal()
        NN.train_model()
```

ДОДАТОК В

ВІДГУК

на кваліфікаційну роботу магістра

" Розробка інформаційної системи пошуку та аналізу жанрової музики "

студента групи 126м-20-1 Ястребцева Олексія Дмитровича

ДОДАТОК Г

РЕЦЕНЗІЯ

на кваліфікаційну роботу магістра

" Розробка інформаційної системи пошуку та аналізу жанрової музики "

студента групи 126м-20-1 Ястребцева Олексія Дмитровича