НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2022 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

на тему: Нейронна мережа для підбору акордів пісень

Виконав : студентка __4__ курсу, групи ІП-84
(шифр групи)

_____ Шахова Поліна Миколаївна_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____доцент, к.т.н. Волокита А. М._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) професор, д.т.н., Сімоненко В. П._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2022 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2022 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студентки

Шахової Поліни Миколаївни

1. Тема проєкту *Нейронна мережа для підбору акордів пісень*
керівник проєкту Волокита Артем Миколайович, доцент, к.т.н.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 10 червня 2022 року №1033-с

2. Термін здачі студентом закінченого проєкту *11 червня 2022 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Огляд предметної області та аналіз існуючих рішень.

Розділ 2. Огляд алгоритмів та технологій для розробки системи.

Розділ 3. Деталі розробки системи.

Розділ 4. Аналіз розробленої системи.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна діаграма системи, діаграма модулів, послідовність дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В. П.		

7. Дата видачі завдання «30» серпня 2021 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2021-15.12.2021*	
2.	*Вивчення та аналіз завдання*	*15.12.2021-15.04.2022*	
3.	*Розробка архітектури та загальної структури системи*	*15.04.2022-25.04.2022*	
4.	*Розробка структур окремих підсистем*	*25.04.2022-05.05.2022*	
5.	*Програмна реалізація системи*	*5.05.2022-15.05.2022*	
6.	*Оформлення пояснювальної записки*	*15.05.2022-11.06.2022*	
7.	*Захист програмного продукту*	*11.06.2022*	
8.	*Передзахист*	*11.06.2022*	
9.	*Захист*	*25.06.2022*	

Студент-дипломник _____ Поліна ШАХОВА
(підпис)

Керівник проєкту _____ Артем ВОЛОКИТА
(підпис)

АНОТАЦІЯ

У даній роботі були розглянуті сучасні методи автоматичного розпізнавання послідовностей музичних акордів у піснях. Було проаналізовано використання згорткової, рекурентної (довгої короткочасної пам’яті), а також змішаної архітектур на прикладі вже спроєктованих для вирішення даної задачі моделей нейронних мереж. На основі аналізу було визначено сильні та слабкі сторони розглянутих систем, визначено ефективні стратегії для побудови нейронної мережі та її навчання. В результаті роботи спроєктовано глибоку нейронну мережу, що здатна розпізнавати розпізнавати акорди з музичних сигналів, проведено дослідження ефективності її роботи в залежності від заданих гіперпараметрів. Програмний продукт був розроблений на мові Python з використанням бібліотеки TensorFlow.

Ключові слова: нейронна мережа, акорд, розпізнавання акордів, CNN, RNN, Python, TensorFlow.

ANNOTATION

In this project for a Bachelor`s Degree, modern methods of automatic chord sequences recognition were considered in detail. Convolutional, Reccurent (Long Short-Term Memory) and Combined architecture of already built networks was analyzed for solving this problem. Based on the analysis, their strengths and weaknesses were recognized and the effective strategies for model building and training were considered. As a result of the work, deep neural network that, is able to recognize chords from music signals, was developed, and research of its work efficiency depending on setted hyperparameters was carried out. The software product was relized in Python using TensorFlor library.

Key words: neural network, chord, chord recognition, CNN, RNN, Python, TensorFlow.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Нейронна мережа	3		
			для підбору акордів			
			пісень			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Нейронна мережа	60		
			для підбору акордів			
			пісень			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Нейронна мережа	1		
			для підбору акордів			
			пісень			
			Структурна діаграма			
			системи			
	А4	***ІАЛЦ.4672008.005 Д2***	Нейронна мережа	1		
			для підбору акордів			
			пісень			
			Діаграма модулів			
	А4	***ІАЛЦ.467200.006 Д3***	Нейронна мережа	1		
			для підбору акордів			
			пісень			
			Послідовність дій			
			програмного забезпечення			
	А4	***ІАЛЦ.467200.007 Д4***	Нейронна мережа	8		
			для підбору акордів			
			пісень			
			Текст програмного коду			

					ІАЛЦ.467200.001 ОА	
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*		
Розроб		Шахова П.М			*Нейронна мережа для підбору акордів пісень* Опис альбому	Літ. / Аркуш 1 / Аркушів 1
Перев		Волокита А.М.				***КПІ ім. Ігоря Сікорського ФІОТ ІП-84***

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Нейронна мережа для підбору акордів пісень*»

Київ – 2022

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку нейронної мережі для розпізнавання музичних акордів в аудіозаписах пісень, а також на подальшу підтримку та вдосконалення розробленої системи.

Областю застосування розробленого проєкту є аналіз звукових сигналів, розробка акустичного обладнання, проєктування систем для класифікації, порівняння чи генерації музики, вилучення музичних транскрипцій, дослідження модуляції та підбору тональності, тощо.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка нейронної мережі для розпізнавання акордів у піснях та музичних файлах, що дозволить ефективно вирішувати дану задачу.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має відповідати таким вимогам:

- Вчитися та тестуватися на наборі перевірених, нормалізованих, різноманітних даних.
- Вміти розпізнавати 25 типів акордів (12 мажорних і 12 мінорних тризвуків та один “не акорд”).
- Мати якомога меншу кількість параметів та простішу структуру.

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.
- Python версії 3.8 або вище.
- Бібліотеки TensorFlow, Librosa, Numpy

5.3. Вимоги до апаратної частини

- ЦП не менше ніж Intel® Core (TM) i3-2100T.
- ROM не менше ніж 64 ГБ.
- RAM не менше ніж 4 ГБ.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.12.2021-15.12.2021
Вивчення та аналіз завдання	15.12.2021-15.04.2022
Розробка архітектури та загальної структури системи	15.04.2022-25.04.2022
Розробка структур окремих частин системи	25.04.2022-05.05.2022
Програмна реалізація системи	5.05.2022-15.05.2022
Виправлення помилок	15.05.2022-11.06.2022
Оформлення пояснювальної записки	11.06.2022

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Нейронна мережа для підбору акордів пісень*»

Київ – 2022

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ACR	(Automatic Chord Recognition) Автоматичне розпізнавання акордів
MIR	(Music Information Retrieval) Вилучення інформації з музичного контенту
HMM	(Hidden Markov Models) Приховані марковські моделі
CNN	(Convolutional Neural Network) Згорткова нейронна мережа
RNN	(Reccurent Neural Network) Рекурентна нейронна мережа

ВСТУП

Розпізнавання акордів в музичних аудіозаписах – це складне та трудомістке завдання, яке вимагає тривалої музичної підготовки, якщо виконується людиною вручну. Системи, що здатні виконувати дану задачу автоматично можуть бути застовані в широкому діапазоні областей, наприклад як інструмент для транскрибування музики, як автоматизований спосіб заповнення баз даних музичними метаданими, як інструмент підбору тональностей пісень, для автоматичної генерації мелодії, або як частина інших завдань MIR (Music Information Retrieval), таких як ідентифікація музичних композицій, класифікація пісень за жанром або емоційним забарвленням, тощо.

Ця дипломна робота має на меті проілюструвати, як математика, обробка сигналів і теорія музики можуть бути застосовані разом для вирішення проблеми розпізнавання акордів у музичних файлах. З цією метою буде розглянуто теорію аналізу Фур'є та її використання для вилучення важливої частотної інформації з аудіосигналів, зв'язок частоти з висотою звуку в музиці, будову та види нейронних мереж, а також їх ефективність у вирішенні задачі виявлення та класифікації музичних акордів.

РОЗДІЛ 1
ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Обробка звуку

Кожну задачу машинного навчання можна розділити на три підзадачі: робота з даними (пошук та збір даних, їх очищення та обробка, виокремлення ознак (features), на основі яких буде проводитись подальше навчання машинних алгоритмів), тренування (побудова моделі системи використовуючи відповідні знайдені ознаки) та оцінка (розрахунок ефективності побудованої моделі). Саме виокремлення ознак – індивідуальних вимірювальних характеристик або властивостей спостережуванного явища, – з вхідних даних є надзвичайно важливим кроком, від якого залежить подальша успішність класифікації досліджуваних об’єктів або передбачення їх значень, і, відповідно, ефективність всієї системи.

Різниця між людським та машинним сприйняттям інформації дуже велика, тому метою будь-якої обробки даних є їх числове представлення – таке, що є зрозумілим для машин. Проте, різні типи даних вимагають різних способів обробки.

Звукові сигнали (аудіосигнали) – сигнали, що спричиняють вібрації (коливання) повітря в певному чутному діапазоні частот. Під чутним діапазоном мається на увазі частоти від 20 до 20 тис. Гц, які здатне розрізняти людське вухо. В свою чергу, частота – це кількість коливань в секунду певної точки середовища, в якому поширюються гармонічні звукові хвилі.

Звук може представлятися у цифровому та аналоговому форматах. Аналоговий сигнал – неперервний у часі, періодичний, виражений синусоїдальними коливаннями, натомість цифровий або дискретний сигнал має обмежені часові рамки. Людина сприймає навколишні звуки в аналоговому

вигляді, в той час як звукові файли прийнято зберігати в цифровому. На цьому етапі доречно ввести поняття частоти дискретизації (sample rate), що представляє собою кількість відліків (семплів) за секунду, при перетворенні безперервного сигналу у дискретний (цей процес називають оцифровуванням звуку). Типові значення частоти дискретизації становлять 16 000, 22 050 Гц або 44 100 Гц тощо. Очевидно, що чим більшою є частота дискретизації, тим точнішою є оригінальна звукова хвиля (рис.) та чистішим звук, хоча для значень більших за 44 100 ця різниця майже непомітна.

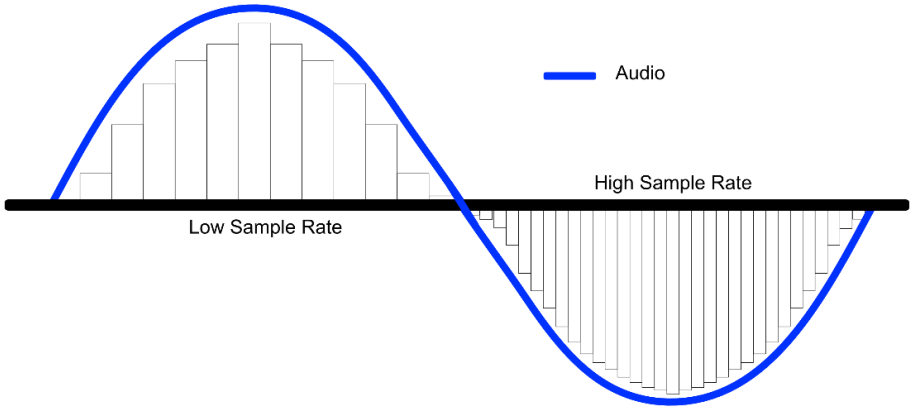


Рисунок 1.1 – Частота дискретизації.

Таким чином, звуковий сигнал може бути визначений як функція амплітуди коливань від часу. (рис.1.2.). Поглянувши на рисунок 1.2. стає очевидно, що лише форми звукового сигналу недостатньо для якісної обробки аудіофайлів. Необхідно вилучити інші, більш інформативні ознаки.

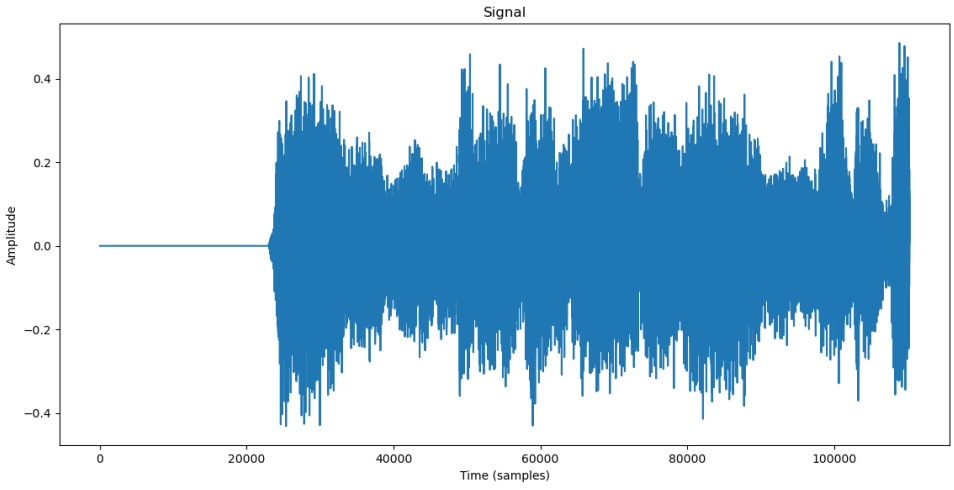


Рисунок 1.2 – Форма звукового сигналу пісні The Beatles “Help!”

Будь-який аудіосигнал складається з декількох одночастотних хвиль, тому алалізуючи його в початковому вигляді, ми розглядаємо лише їх результуючу амплітуду. Щоб отримати більше інформації з заданого звукового сигналу необхідно представити його як суму окремих синусоїд (тобто частот).

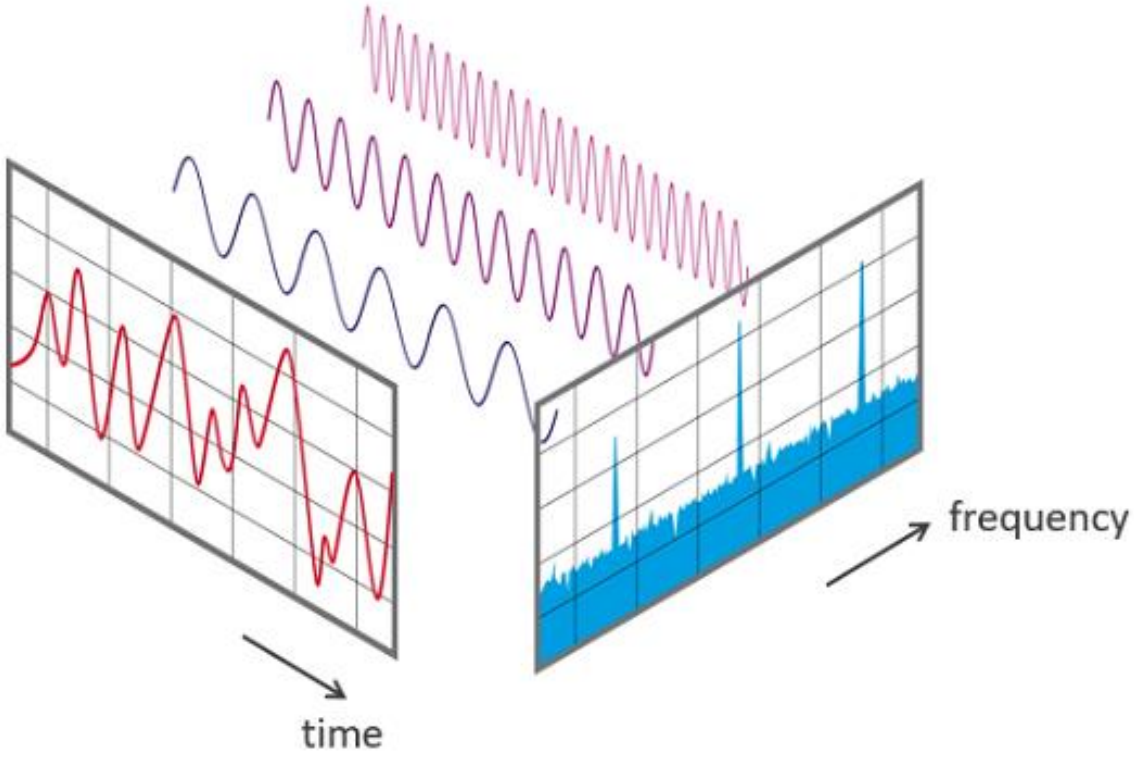


Рисунок 1.3 – Розкладання сигналу на окремі частоти та їх амплітуди (Перетворення Фур’є).

Цей метод був запропонований Джозефом Фур’є в 1822 році і відомий зараз як перетворення Фур’є (The Fourier Transform). Іншими словами вхідний сигнал конвертується з часової області в частотну. Результат такого перетворення називається спектром.

Швидке перетворення Фур’є (FFT – Fast Fourier Transform) — це алгоритм, який дозволяє ефективно обчислити перетворення Фур’є та знайти спектр.

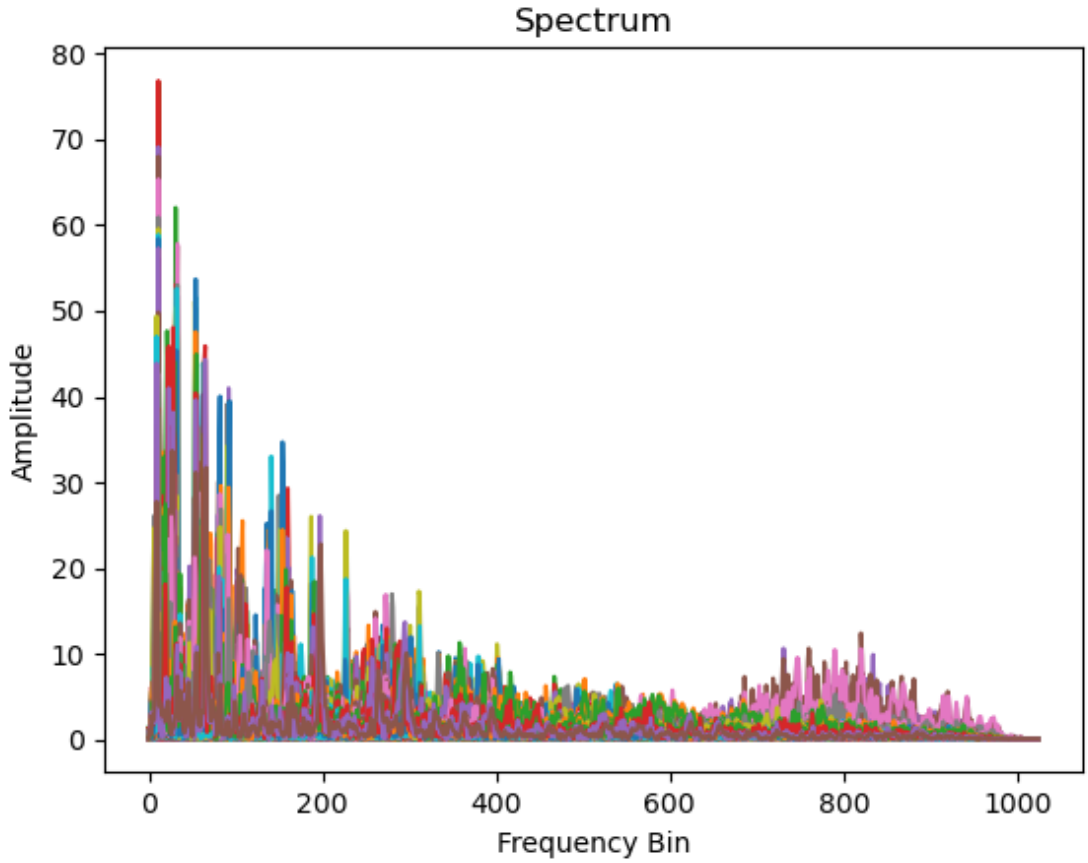


Рисунок 1.4 – Спектр звукового сигналу (перших 5с пісні Beatles “Help!”)

Перетворення Фур’є це дуже потужний засіб аналізу звукових сигналів, проте більша частина сигналів є неперіодичними (як музика чи запис голосу) – тобто такими, що їх частоти змінюються у часі. Класичний Фур’є не враховує цю зміну, а обчислює частоти та їх амплітуди лише на всьому заданому проміжку часу. Тому якщо часовий фактор є важливим для вирішення певної задачі (як зокрема розпізнавання акордів) замість класичного використовується алгоритм короткочасного перетворення Фур’є (Short-Time Fourier-Transform). Його відмінність полягає у попередньому розділенні вхідного сигналу на невеликі проміжки (зазвичай називаються вікнами), та обчисленні швидкого перетворення Фур’є на кожному з цих проміжків. Іноді вікна можуть накладатися одне на одне для забезпесення більш точного результату.

Короткочасного перетворення Фур’є обчислюється як:

$$STFT\{x(t)\}(\tau,\omega) \equiv X(\tau,\omega) = \int_{-\infty}^{\infty} x(t)w(t-\tau)e^{-i\omega t}dt,$$

де $w(\tau)$ – віконна функція (Ханна або Гауса), $x(t)$ – вхідний сигнал тривалістю t.

Загальну схему алгоритму можна побачити на рисунку 1.5.

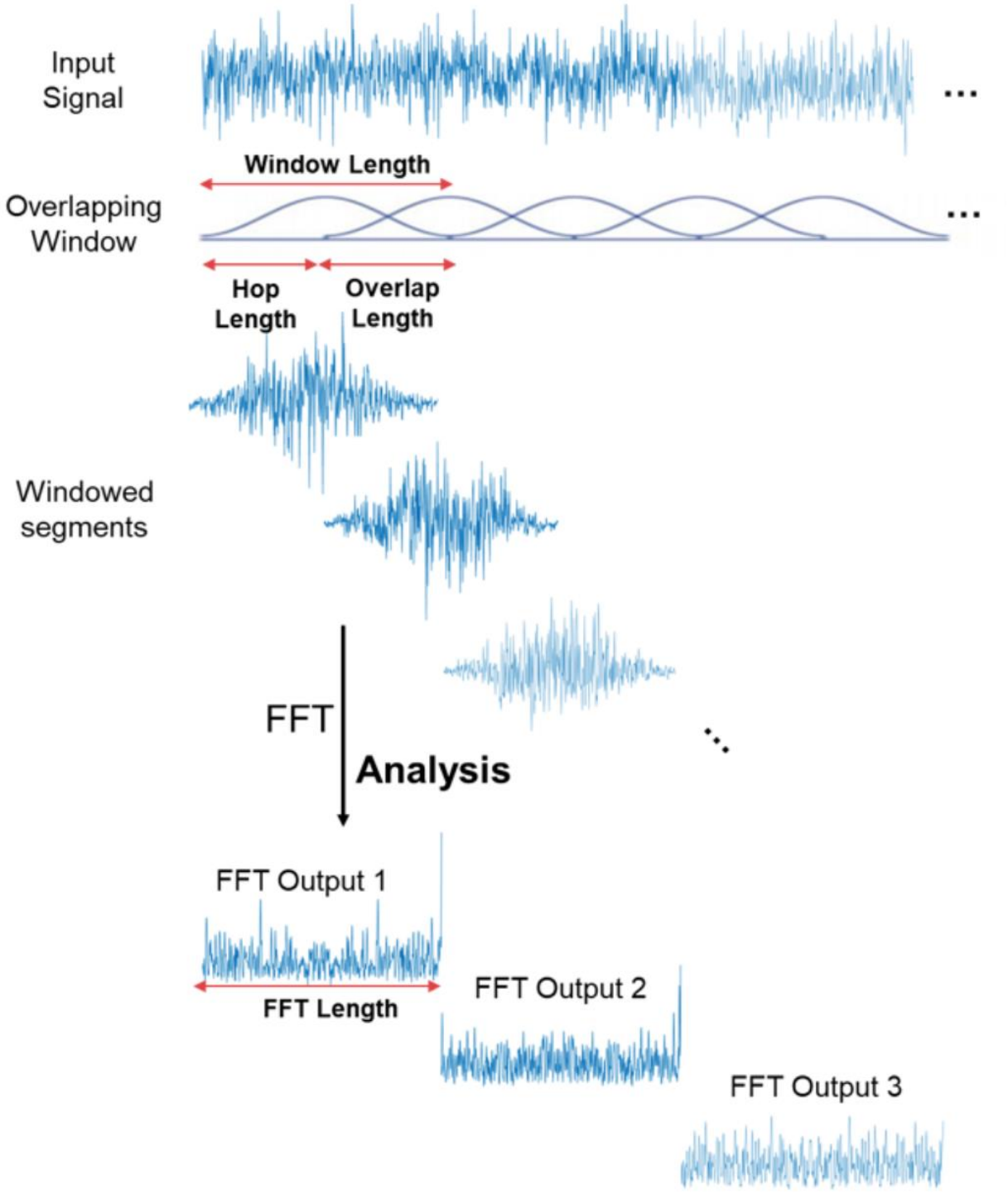


Рисунок 1.5 – Схема Short-Time Fourier-Transform

Як результат короткочасного перетворення Фур'є отримуємо частотно-часове представлення сигналу, або спектрограму.

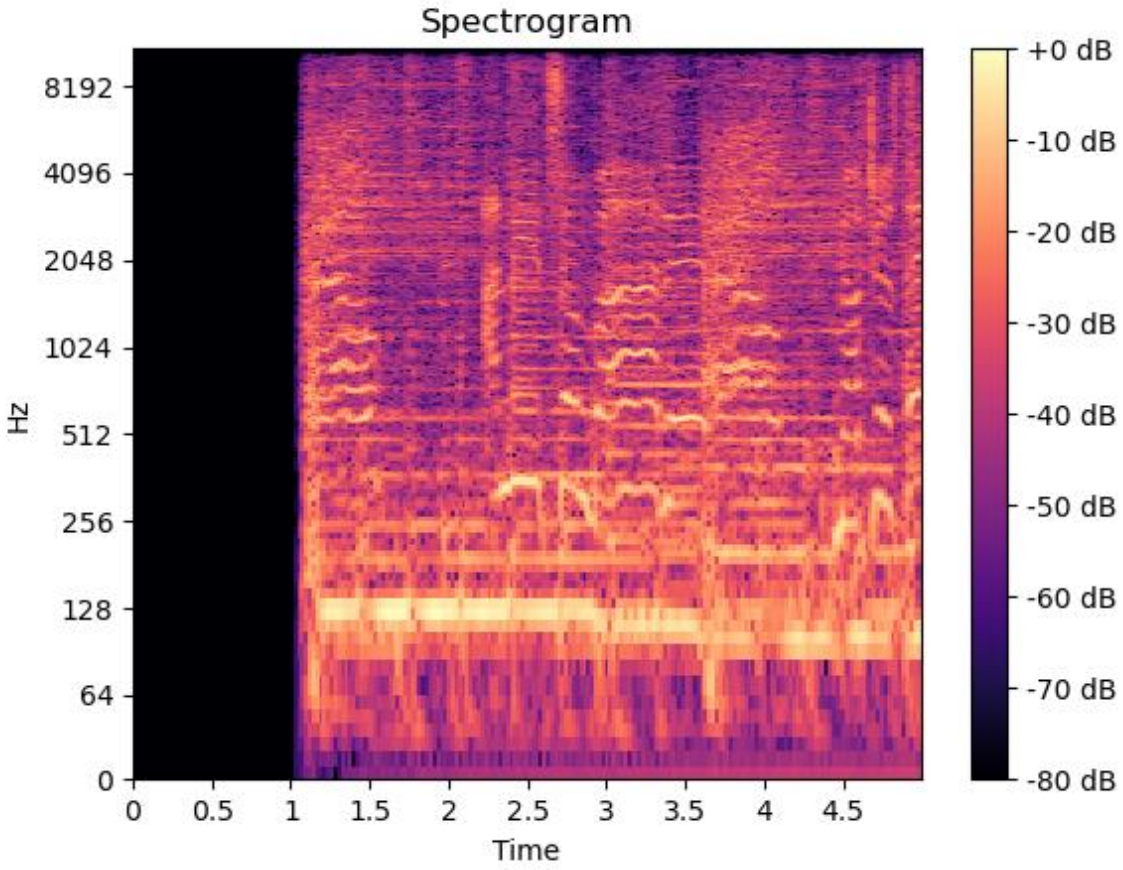


Рисунок 1.6 – Спектрограма для перших 5 секунд пісні The Beatles "Help!"

Спектрограма зберігає в собі майже всю інформацію, яку отримуємуємо ми прослухавши певний звуковий сигнал. Проте є ще одна важлива відмінність. Людське вухо сприймає частоти не у лінійній шкалі, тому нижчі частоти нам розрізняти набагато легше, аніж вищі. У початковому вигляді спектрограми не повністю відповідяють людському сприйняттю звуків. Тому у 1937 році Стівенсом, Фолькcманом і Ньюманом була запропонована нова одиниця виміру висоти звуку – мел, – що базується на психо-фізіологічному сприйнятті звуків людиною:

$$mel = 1127.01048 \ln(1 + \frac{frequency}{700})$$

Таким чином відстані між звуками, що здаються однаковими для слухача, є рівними за мел-шкалою, але мають логарифмічну залежність від реальної частоти.

Тому, однією з основних характеристик, які використовуються системах аналізу/обробки звуку (зокрема в нейронних мережах для навчання моделей) є мел-спектрограми.

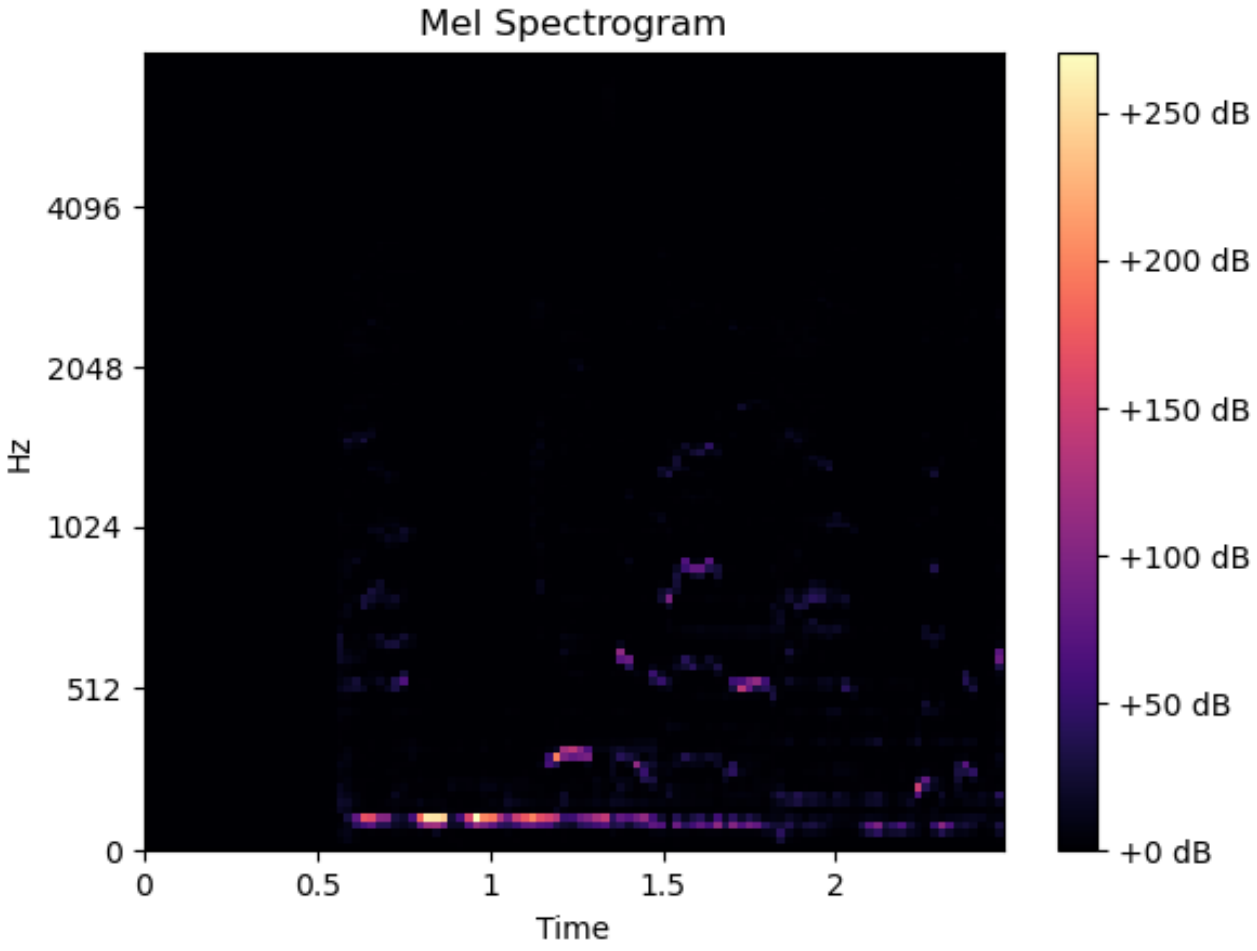


Рисунок 1.7 – Мел-спектрограма для перших 5 секунд пісні The Beatles “Help!”

Мел – спектрограму можна вважати таким собі вдосконаленням звичайної спектрограми. Її можна отримати як результат короткочасного перетворення Фур’є, з переведеними за мел-шкалою частотами.

1.2 Основи музичної теорії

Нота – найменша одиниця у музиці. В західній музичній теорії виділяють 7 основних нот, що позначаються сімома першими літерами алфавіту – A, B, C, D, E, F, G.

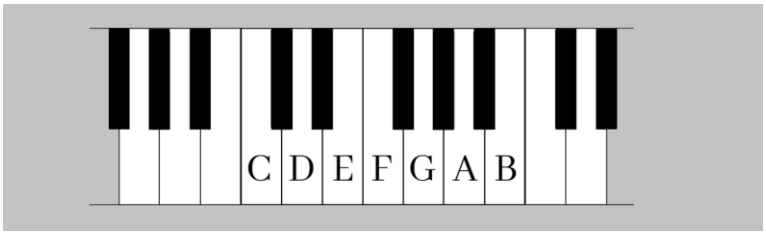


Рисунок 1.8 – Розташування основних нот на фортепіано.

Інтервал (відстань) між двома нотами складає один тон, за винятком двох випадків – між нотами B і C, а також F і G відстань становить півтон, тобто половину тону. Півтон – це найменший інтервал у музиці.

Між основними сімома нотами також є інші, що записуються за допомогою знаків алітерації «#» або «b» (дієз та бемоль), що відповідно підвищують або понижують основну ноту на півтон. Наприклад, A# та Bb фактично позначають один звук – ноту на півтон вище ніж A та на півтон нижче ніж B. В результаті отримуємо 12 нот з інтервалом у півтон: A, A#, B, C, C#, D, D#, E, F, F#, G, G#, – які складають одну октаву.

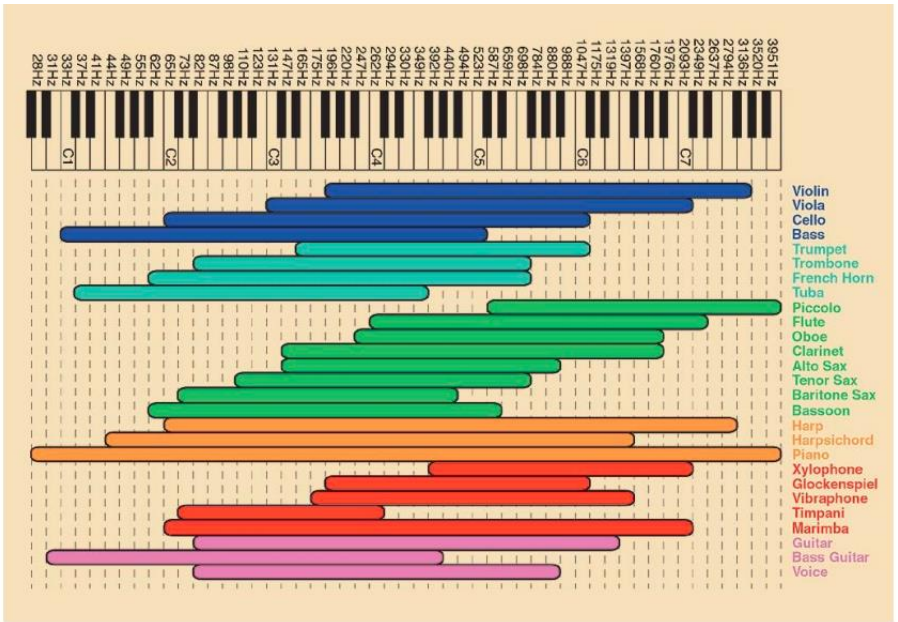


Рисунок 1.9 – Частотні діапазони різних музичних інструментів.

Фізичною характеристикою ноти є частота. Знаючи декілька основних правил досить легко визначити частоти звучання усіх нот:

1. Як «опорну» використовують ноту А з її частотою 440 Гц.
2. Подвоївши частоту будь-якої ноти отримаємо ту ж ноту на одну октаву вище, і навпаки зменшивши частоту у 2 рази – ту ж ноту на октаву нижче.
3. Усі інтервали між послідовними нотами рівні в логарифмічному масштабі.

Тобто, А, зіграна на октаву вище, матиме частоту 880 Гц; А# матиме частоту ноти А помножену на $2^{\frac{1}{12}}$, тобто $440 \times 2^{\frac{1}{12}} \approx 466.2$ Гц, і т.д.

Дані правила можна перевірити за таблицею 1.1, яка представляє частоти усіх 12 нот в різних октавах.

Таблиця 1.1 – Частоти нот в восьми октавах.

Ноти	**Октави**							
	1	2	3	4	5	6	7	8
A	55.0	110.0	220.0	440.0	880.0	1760.0	3520.0	7040.0
A#	58.3	116.5	233.1	466.2	932.3	1864.7	3729.3	7458.6
B	61.7	123.5	246.9	493.9	987.8	1975.5	3951.1	7902.1
C	65.4	130.8	261.6	523.3	1046.5	2093.0	4186.0	8372.0
C#	69.3	138.6	277.2	554.4	1108.7	2217.5	4434.9	8869.8
D	73.4	146.8	293.7	587.3	1174.7	2349.3	4698.6	9397.3
D#	77.8	155.6	311.1	622.3	1244.5	2489.0	4978.0	9956.1
E	82.4	164.8	329.6	659.3	1318.5	2637.0	5274.0	10548.1
F	87.3	174.6	349.2	698.5	1396.9	2793.8	5587.7	11175.3
F#	92.5	185.0	370.0	740.0	1480.0	2960.0	5919.9	11839.8
G	98.0	196.0	392.0	784.0	1568.0	3136.0	6271.9	12543.9
G#	103.8	207.7	415.3	830.6	1661.2	3322.4	6644.9	13289.8

Три або більше різні ноти, зіграні одночасно, формують музичний акорд. У даній роботі розглядатимуться лише мажорні та мінорні тризвуки, що є найпоширенішими. І мажорні, і мінорні акорди складаються з трьох нот. Мажорний акорд складається з основної ноти (root note), ноти на 2 тони вище від основної та ноти на 1.5 тони вище від другої. Для мінорного акорду спочатку застосовується інтервал у 1.5 тони від основної ноти, потім 2 тони від другої. Тому ноти C, E та G утворюють мажорний акорд C:major; а ноти C, Eb та G відповідно мінорний акорд C:minor. Мажорні акорди у музичних транскрипціях найчастіше записуються однією великою літерою основної ноти, а мінорні – з додатковою «m» в кінці, наприклад A та Am.

1.3 Автоматичне розпізнавання акордів

Далі будуть розглянуто поняття автоматичного розпізнавання музичних акордів, загальна схема та методи, які для цього використовуються.

1.3.1 Історична довідка

Завдання розпізнавання пісенних акордів визначається як вилучення синхронізованої у часі послідовності акордів; вхідними даними є необроблені музичні аудіозаписи. Тобто, при заданому звуковому сигналі $x(t)$, де $t \in [t_{start}, t_{end}]$ та множині можливих класів акордів Y, для кожного моменту часу t треба визначити акорд, який звучить у цей момент.

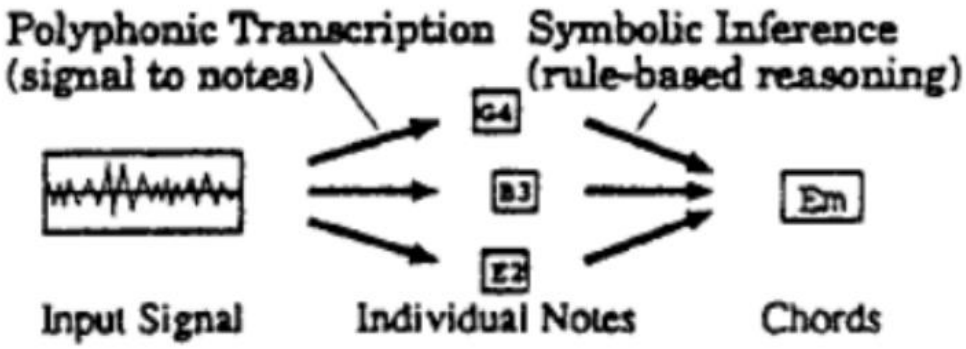


Рисунок 1.10 – Класичний підхід до розпізнавання музичних акордів

Проблему автоматичного розпізнавання акордів вперше було сформульовано приблизно у 1990-х роках. У перших роботах вона вирішувалася шляхом попереднього транскрибування, тобто розпізнавання окремих нот, та їх подальшого об'єднання в акорди.

Першим спосіб розпізнавання акордів без попереднього визначення нот запропонував Т. Фуджишима. Його метод ліг в основу всіх наступних досліджень даної області і є актуальним до сьогодні. В основі даного методу лежить отримання спектрограми, як частотно-часового представлення вхідного звукозапису.

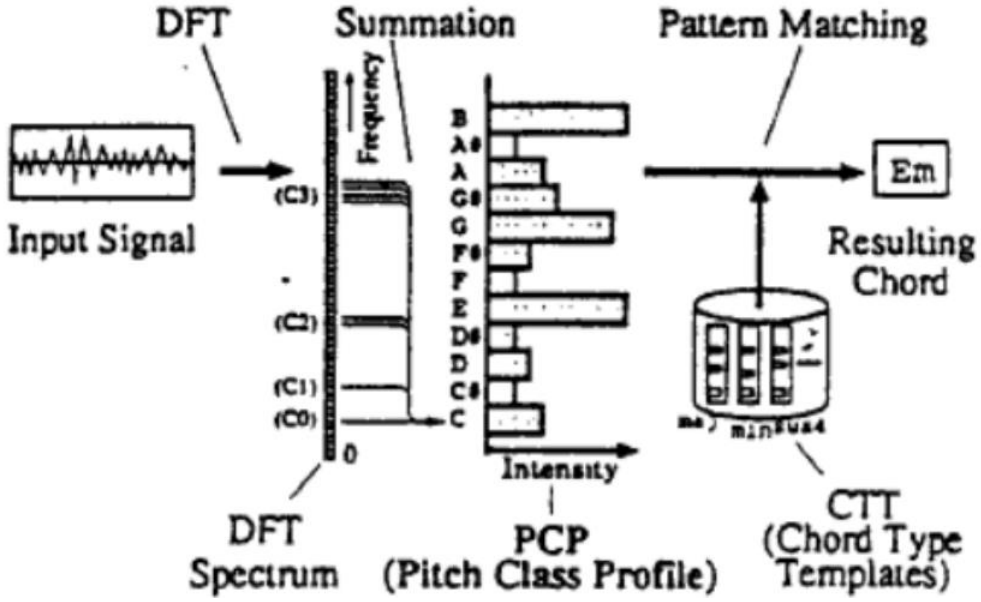


Рисунок 1.11 – Модель Т. Фуджишима

Як уже було згадано раніше, для побудови спектрограм вхідний аудіофайл необхідно попередньо розбити на невеликі частини. У вирішенні даної задачі існують різні підходи – використання фіксованих за розміром фрагментів (для музичних файлів зазвичай 0.1 – 0.3 с), динамічна зміна розмірів фрагменту залежно від ритму музики чи комбіновані підходи. Спектр також може бути отриманий різними методами, як дискретне перетворення Фур'є, Constant-Q перетворення. Після отримання спектру на кожному його фрагменті визначається акорд, що звучить у цей момент часу (та відповідає певному вектору з послідовності стовпців спектрограми).

Таким чином, у 2000-х роках задача вилучення послідовності акордів з музичних файлів та пошук нових ефективних методів її реалізації остаточно закріплюється як об’єкт досліджень. Результати цього напряму доповідаються на провідних міжнародних конференціях, як зокрема International Society for Music Information Retrieval (ISMIR), IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Sound and Music Computing Conference (SMC). З 2008 року проводяться змагання серед алгоритмів розпізнавання акордів у рамках щорічної кампанії з оцінки методів вилучення інформації з музичнго контенту MIREX. За цей час загальна ефективність систем розпізнавання акордів збільшилась з 65% до 80%, що є безперечно гарним результатом. Це ще раз підкреслює актуальність обраної теми.

1.3.2 Приховані моделі Маркова та нейронні мережі

Більшість систем автоматичного розпізнавання акордів складаються з трьох частин: виділення ознак, узгодження шаблонів і декодування послідовності музичних акордів.

Досить поширеним та ефективним підходом є використання прихованих моделей Маркова (HMM) для визначення послідовностей акордів. Вони дозволяють у явному вигляді моделювати ймовірності переходу між двома заданими акордами. Дану архітектуру використано, наприклад, у [5] та [6], та проаналізовано найефективніші характеристики (features), що можуть бути використані для навчання.

Незважаючи на досить якісні результати, такі методи мають свої недоліки, зокрема:

- Потребують велику кількість розмічених даних для навчання, які досить важко підготувати, і самі дані часто відрізняються для різних стилів музики, різних епох та різних композиторів.

- Існує небезпека перенавчання, адже моделі з великою кількістю параметрів найкраще підлаштовуються під доступний набір навчальних даних, але не гарантують правильність та якість розпізнавання у роботі з даними не з вибірки.
- Багатомірність даних призводить до експоненційного зростання кількості обробленої інформації та часу, необхідного на її обробку, а також до збільшення обсягу навчальної вибірки.
- Приховані марківські моделі передбачають залежність лише від попереднього елементу у послідовності акордів, нехтуючи таким чином часовим фактором.
- Оперуючи переважно штучними, математичними, а не музичними конструкціями, вони мають досить високу складність інтерпретації моделей.
- Дані ймовірносні методи добре пристосовані для моделювання зміни стану, тобто переходу від одного акорду до іншого, але гірше – для моделювання тривалості знаходження в одному стані.

Деякі з цих недоліків можна виправити за допомогою методів глибокого навчання, що мають в основі багатошарову нейронну мережу, кожен з шарів якої попередньо навчається без вчителя на нерозмічених даних, після чого потрібна лише невелика кількість розмічених прикладів для остаточного налаштування параметрів. Останнім часом багато досліджень в даній області було присвячено глибоким нейронним мережам, зокрема згортковим (CNN) та рекурентним (RNN), а також мережам довгої короткочасної пам’яті (LSTM).

Зокрема у [7] було використано рекурентну нейронну мережу для визначення послідовності акордів на основі спектрограм. Запропонована мережа здатна розпізнавати 146 типів акордів з ефективністю приблизно 80.6%.

[8] пропонує повнозв’язну нейронну мережу з одним прихованим шаром. Модель здатна розпізнавати 10 типів акордів (A, Am, Bm, C, D, Dm, E, Em, F,

G) на основі векторів висотного класу (Pitch Class Profiles). Для навчання та перевірки ефективності було використано власноруч зібраний датасет, що складався з “чистих” (зіграних окремо на різних інструментах) та зашумлених (вирізаних з музичних файлів) акордів. Таким чином, нейронну мережу було натреновано та протестовано за допомогою трьох різних комбінацій тренувальних даних:

- Навчання – на чистих акордах, валідація – на зашумлених;
- Навчання – на зашумлених, валідація – на чистих;
- Комбінований набір даних з чистих і зашумлених акордів для навчання і валідації.

Перший підхід виявився найменш ефективним, тоді як третій показав найкращий результат.

У [9] використовується CNN архітектура у комбінації з умовними випадковими полями (Conditional Random Fields). Згорткова нейронна мережа використовується для вилучення ознак з вхідних даних (мел-спектрограм), які потім обробляються умовними випадковими полями з використанням алгоритму Вітербі та декодуються в остаточну послідовність акордів.

Таким чином, на основі оглянутих реалізацій моделей глибокого навчання, у наступних розділах буде визначена і побудована власна нейронна нейронна мережа для розпізнавання музичних акордів у піснях.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі розглянуті способи обробки звуку та пояснені ключові поняття музичної теорії, такі як ноти і акорди, з математичної точки зору.

Аудіосигнал представляється функцією амплітуди коливань від часу і складається з однієї і більше одночастотних хвиль. Розкладання сигналу на окремі частоти та амплітуди називається перетворенням Фур'є, в результаті якого отримуємо спектр. Оскільки класичний Фур'є не враховує зміну частот у часі – на практиці застосовується метод короткочасного перетворення Фур'є: вхідний сигнал попередньо ділиться на фрейми, на кожному з яких застосовується розподіл на окремі частоти. В результаті отримуємо спектрограму – частотно-часову характеристику аудіо сигналу.

Людське вухо сприймає висоту звуку, тобто частоту, нелінійно. Тому в обробці звуку застосовується характеристика, близька за сприйняттям звуків до людського – мел-спектрограма. Мел-спектрограми отримують з спетрограм, адже вони пов'язані логарифмічною залежністю.

В основі утворення акордів лежать частотні інтервали, рівні в логарифмічному масштабі.

Проблема автоматичного розпізнавання акордів постала ще у 1990-х роках, а її вирішення залишається актуальним і досі. Деякі популярні підходи розглянуто в даному розділі. Окрім класичного методу зіставлення шаблонів, описаного Фудзісімою, було розглянуто ймовірнісні моделі, а саме – приховані марківські моделі, які є дуже популярним способом у вилученні акордних послідовностей з аудіофайлів. В основі даного підходу – ідея повторюваності акордів, які дозволяють відкидати найменш імовірні переходи між акордами та дають досить високу точність, хоча в той же час мають ряд недоліків. Деякі з них можна оминути за допомогою штучних нейронних мереж. Для задачі

підбору музичних акордів найчастіше використовують згорткові та рекурентні мережі.

Приклади існуючих систем розпізнавання акордів на основі нейронних мереж, розглянуті у даному розділі, безпосередньо демонструють популярність та актуальність обраної теми.

РОЗДІЛ 2
ОГЛЯД АЛГОРИТМІВ ТА ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ

2.1 Нейронні мережі

Далі буде визначено поняття нейронної мережі, її будови, а також способу роботи, розглянуто основні характеристики нейронних мереж, описано структуру згорткових та рекурентних нейронних мереж.

2.1.1 Основні поняття

Штучні нейронні мережі (ANN) є програмною імплементацією біологічних нейронних структур, що представляє собою мережу пов'язаних між собою штучних нейронів. Мережа обробляє вхідну інформацію та у процесі зміни свого стану у часі формує сукупність вихідних сигналів.

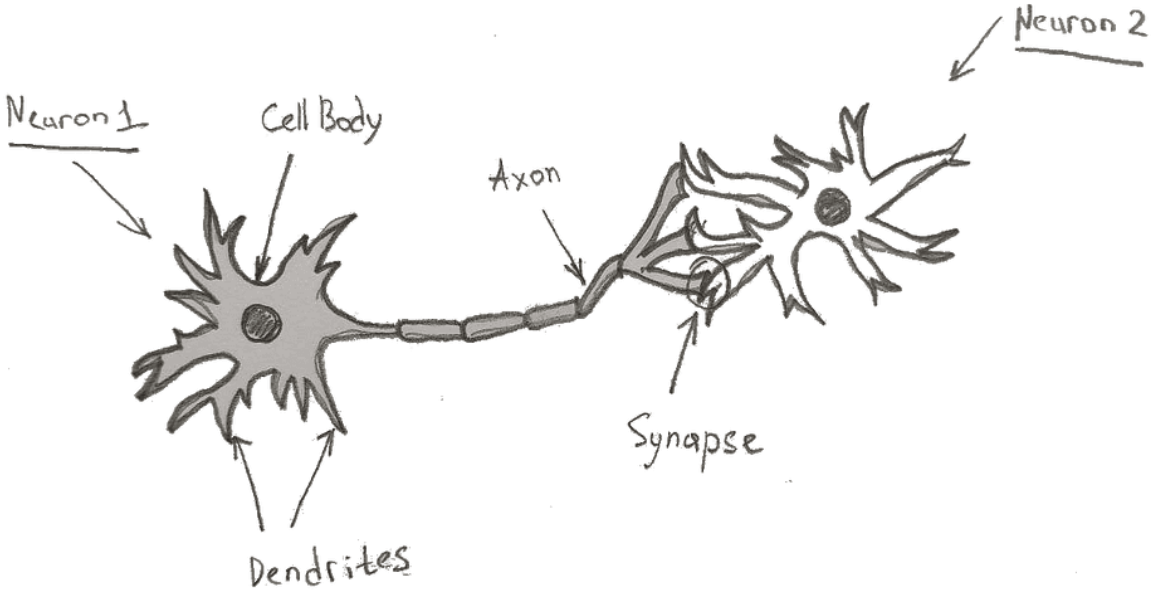


Рисунок 2.1 – Будова біологічного нейрона

Біологічний нейрон отримує вхідні сигнали від множини дендритів (входів), причому кожен дендрит отримує інформаційний сигнал від іншого

нейрона в нервовій системі. Коли сигнал через дендрити досягає тіла клітини, він викликає невелику зміну її електричного потенціалу. Одні дендрити викликають невелику позитивну зміну потенціалу, інші – негативну. Проте якщо сукупний ефект цих змін викликає збільшення потенціалу спокою до певного критичного значення, нейрон ініціює хвилю збудження вниз від тіла клітини по аксону (виходу), передаючи таким чином сигнал іншим нейронам у мережі.

Загалом можна описати три кроки поширення сигналу біологічними нейронами, на яких і грунтується принцип роботи штучних нейронних мереж:

1. Отримання інформації від багатьох інших нейронів.
2. Агрегування отриманої інформації шляхом зміни потенціалу тіла клітини.
3. Передача сигналу при перевищенні граничного рівня електричного потенціалу клітини. Сигнал може бути отриманий багатьма іншими нейронами у мережі.

Нейронні мережі виникли з досліджень у галузі штучного інтелекту, а саме зі спроб відтворити здатність біологічних нервових систем навчатися та виправляти помилки, моделюючи низькорівневу структуру мозку.

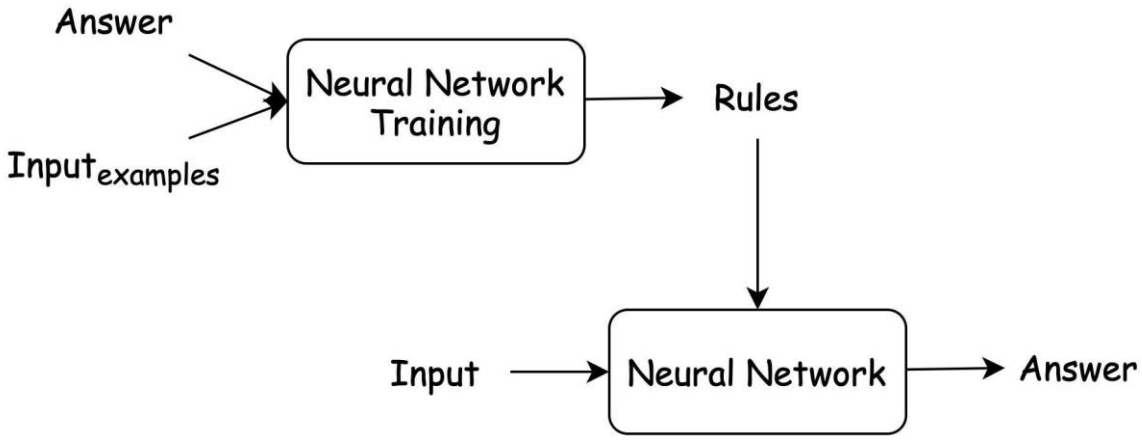


Рисунок 2.2 – Загальний принцип роботи нейронної мережі.

Вперше математична модель штучного нейрона була запропонована У. Маккалоком та У. Піттсом, та представляла собою ваговий суматор, вихідне значення якого задавала активаційна функція:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + w_0 x_0\right),$$

де x_i – вхідні сигнали, w_i – ваги входів.

Дана формула означає, що на вхід штучного нейрона надходить кілька сигналів, кожен із яких є виходом іншого нейрона. Кожен вхід множиться на відповідну вагу, аналогічну синоптичній силі, і всі добутки підсумовуються, визначаючи рівень активації нейрона.

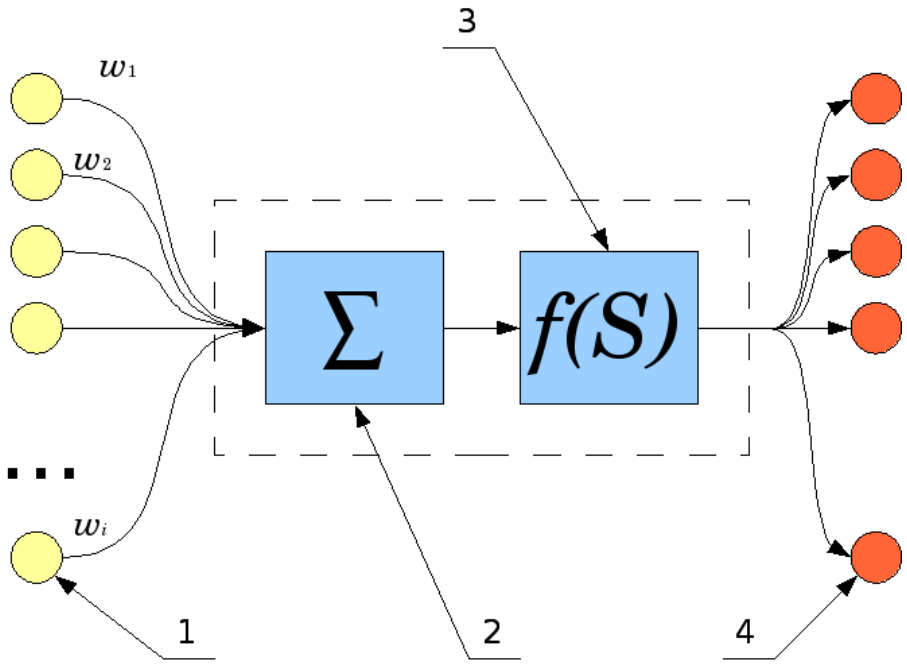


Рисунок 2.3 – Модель штучного нейрона Маккалока-Піттса. 1 – Нейрони, вихідні сигнали яких надходять на вхід даного нейрону; 2 – Суматор вхідних сигналів; 3 – Обчислювач передавальної функції; 4 – Нейрони, на входи яких подається сигнал даного нейрону; 5 – ваги вхідних сигналів.

Першою практичною реалізацією цієї мережі можна вважати алгоритм персептрона, створений і описаний у 1958 році Френком Розенблаттом. Бінарна природа входів та виходів персептрону значно обмежує його використання

зараз для побудови нейронних мереж. У багатьох випадках виникає необхідність робити прогнози на основі вхідних даних, отриманих з безперервних діапазонів значень, що робить перцептрони непридатними для вирішення таких завдань. Цей же аспект істотно ускладнює процес навчання, адже перехід від нульового до одиничного значення на виході перцептрону відбувається миттєво, що і ускладнює налаштування коефіцієнтів w і b відповідно до бажаного вихідного значення.

2.1.2 Функції активації

З розвитком нейронних мереж з'явилися також альтернативи нестійкій поведінці персептрону, а саме інші більш плавні функції активації:

1. Сигмоїда – плавна крива на проміжку від 0 до 1.

Сигмоїда визначається як:

$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

де $z = w \cdot x + b$, $e = 2.718 \ldots$

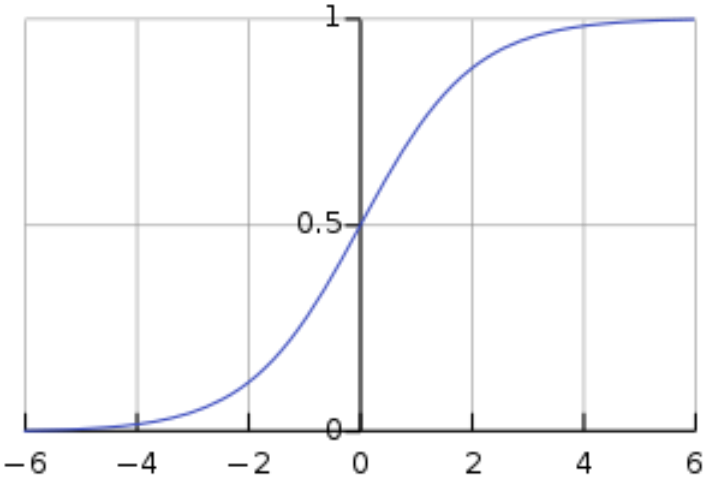


Рисунок 2.4 – Сигмоїда

Як видно з малюнку, для вхідних значень наближених до 0, функція повертає значення близько 0.5. Зі зростанням вхідних значень результат прямує до 1, і навпаки, зі зменшенням вхідних значень (у від'ємному напрямку) – прямує до 0.

Перевага сигмоїди над перцептроном очевидна: невеликі зміни параметрів мережі (w і b) викликають невеликі зміни z (z=w·x+b) і, відповідно, невеликі зміни активації нейрона (a). Великі (додатні або від'ємні) значення z є винятком: при екстремально великих за абсолютною величиною значеннях z сигмоїдні нейрони, подібно до перцептрону, виводитимуть 0 (для z < 0) або 1 (для z > 0). Аналогічно до перцептрону, це означатиме, що невеликі зміни ваг і зсувів під час навчання мало впливатимуть на результат, і, як наслідок, навчання зупиниться. Ця ситуація називається насиченням нейрона і може виникати з використанням більшості функцій активації.

2. Функція активації tanh – візуально схожа на сигмоїду, і визначається як:

$$\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

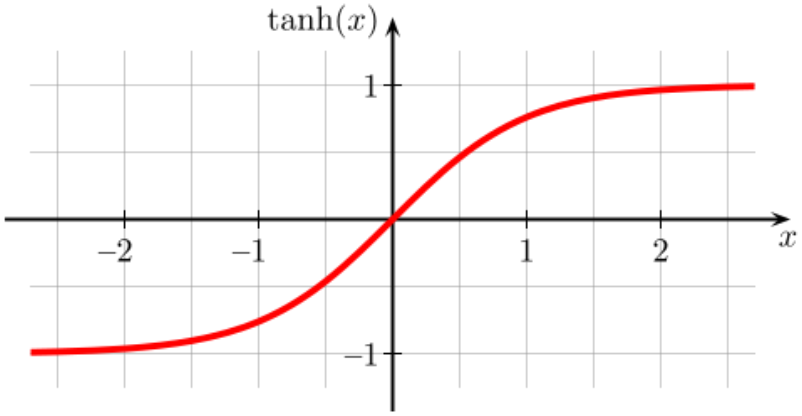


Рисунок 2.5 – Активаційна функція tanh

Функція tanh має ширшу за сигмоїду область значень – [-1:1]. Тобто від'ємним вхідним значенням відповідатимуть від'ємні активації (значення активаційної функції), а додатнім відповідно – додатні, серединою розподілу вихідних значень є число 0. Такі виходи з центром розподілу в точці 0 зазвичай є входами для інших штучних нейронів в мережі. В свою чергу у входах з центром розподілу в точці

0 ймовірність насичення нейронів менша, що дозволяє мережі навчатись ефективніше.

3. Rectified Linear Unit або ReLU – функція активації, форма якої цілком відрізняється від двох попередніх, і була створена на основі властивостей біологічних нейронів та популяризована Винодом Наїром та Джеффом Хінтоном для застосування у штучних нейронних мережах.

Форма функції ReLU визначається рівнянням:

$$a = \max(0, z),$$

що означає:

- якщо z > 0 – функція активації ReLU повертає z (тобто a = z);
- якщо z ≤ 0 – функція повертає своє мінімальне значення 0 (тобто a = 0).

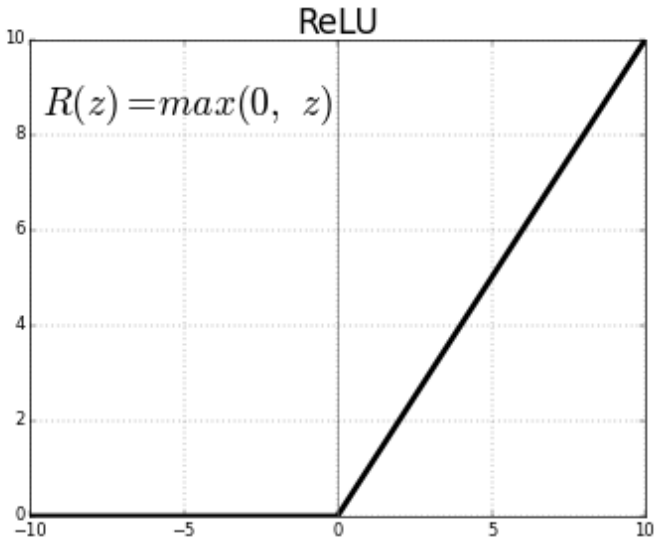


Рисунок 2.6 – Функція активації ReLU

Таким чином ReLU є нелінійною функцією, яка при цьому складається з двох лінійних. Нелінійність – дуже важлива властивість усіх активаційних функцій, адже саме вона дозволяє моделям глибинного навчання апроксимувати будь-яку неперервну функцію.

У мережах глибокого навчання через простоту форми активаційна функція ReLU є надзвичайно ефективною. Насамперед через те, що

процес визначення потрібних значень параметрів w і b (ваг і зсувів) включає визначення часткових похідних. Такі операції на лінійних частинах функції ReLU простіші у обчислювальному відношенні (а значить більш ефективні), ніж на кривих, таких як сигмоїда і tanh. Тому на сьогоднішній день дана функція активації найбільш широко використовується у прихованих шарах глибоких штучних нейронних мереж.

Типи нейронів у прихованих шарах штучної нейронної мережі визначаються їх функціями активації. Дана дипломна робота представляє модель нейронної мережі з нейронами типу ReLU, адже вони забезпечують більш високу ефективність обчислень в алгоритмах навчання.

В задачах класифікації з декількома класами, якою є і задача розпізнавання акордів в музичних файлах, у вихідних шарах краще за все використовувати окрему функцію активації – Softmax, яка для кожного з класів класифікації повертає ймовірність приналежності об’єкта, що аналізується, до даного класу.

2.1.3 Навчання мережі

Найпростішим і популярним способом навчання є метод навчання з учителем – метод зворотного поширення похибки (backpropagation). Backpropagation являє собою ітераційний процес, який починається з останнього шару і рухається у зворотному напрямі через всі шари, поки не досягне першого.

Припустимо, що для кожного шару відома похибка на виході з шару. Якщо відома помилка на виході, то не важко розрахувати зміни для ваги, тим самим зменшивши її. Але проблема полягає в тому, що похибку видно тільки на виході останнього шару. Тобто алгоритм зворотного поширення помилки дає можливість визначити помилку на виході попереднього шару, з огляду на помилку на виході в поточному шарі.

Детальний алгоритм представлено на рисунку 2.7.

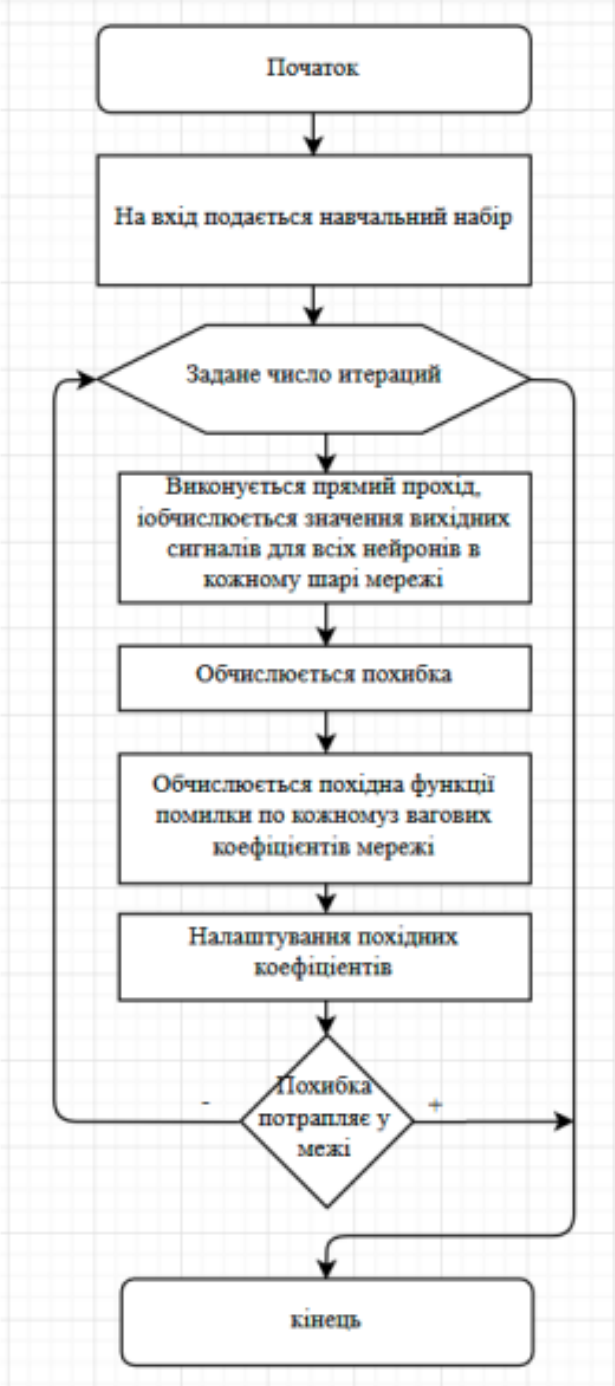


Рисунок 2.7 – Алгоритм зворотного поширення похибки

2.1.4. Функції вартості.

Для кількісного вираження оцінки якості навчання алгоритми машинного навчання часто використовують функції вартості (також відомі як функції

втрат). Зокрема, найбільш поширеними є квадратичну функцію вартості та перехресну ентропію, які будуть розглянуті у цьому пункті.

1. Квадратична функція вартості - одна з найпростіших з обчислювальної точки зору. Також її називають середньоквадратичною помилкою, що пояснюється наступною формулою:

$$C = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Для кожного і-го екземпляра обчислюється різниця (помилка) між дійсною міткою y_i та оцінкою мережі $\hat{y}_i$, яка потім підноситься до квадрату. Отримавши квадратичну помилку для кожного і-го екземпляра, ми можемо обчислити середню вартість C для всіх n екземплярів, додавши усі знайдені квадратичні похибки та поділивши на кількість екземплярів.

2. Перехресна ентропія – є кращим замінником функції квадратичної вартості, адже допомагає значно зменшити вплив насичених нейронів на швидкість навчання мережі (а саме її сповільнення). З цієї причини вона набагато частіше зустрічається в ролі функції вартості, її і буде використано в даній дипломній роботі при побудові нейронної мережі. Перехресна ентропія визначається за формулою:

$$C = -\frac{1}{n}\sum_{i=1}^{n}[y_i \ln \hat{y}_i + (1 - y_i)\ln(1 - \hat{y}_i)]$$

Так само як у квадратичній функції вартості, збільшення різниці між y_i та $\hat{y}_i$ супроводжується збільшення вартості. За аналогією з квадратом у квадратичній функції вартості натуральний логарифм ln у функції перехресної ентропії забезпечує експоненціальне збільшення вартості при лінійному збільшенні різниці між y_i та $\hat{y}_i$. Функція

перехресної ентропії структурована так, що чим більша різниця між y_i та $\hat{y}_i$, тим вища швидкість навчання нейрона.

2.2 Види нейронних мереж

2.2.1 Згорткові нейронні мережі

Згорткові нейронні мережі (Convolutional Neural Network, ConvNet, CNN) – це штучні нейронні мережі, що мають один або кілька згорткових шарів. Шари цього типу дозволяють моделям глибокого навчання ефективно обробляти просторові структури.

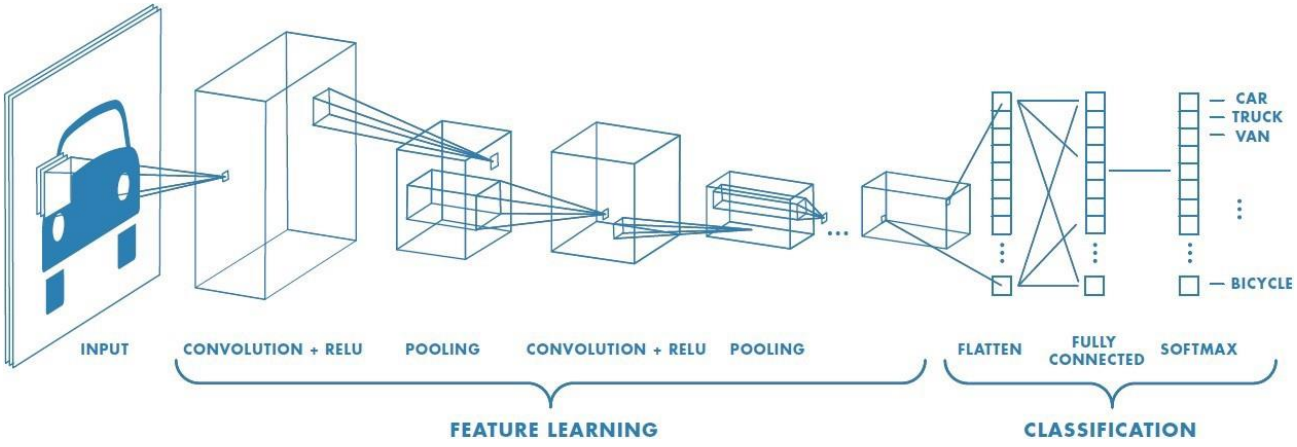


Рисунок 2.8 – Схема згорткової нейронної мережі

Архітектура ConvNet аналогічна моделі зв’язку нейронів у людському мозку і була натхненна принципом роботи зорової кори. Окремі нейрони реагують на подразники лише в обмеженій області поля зору – рецептивному полі. Набір таких полів перекривається, щоб охопити всю візуальну область.

Згорткові нейронні мережі вирізняються від інших типів нейронних мереж високою продуктивністю при обробці зображеннь, мови або аудіосигналів. Вони мають три основних типи шарів, а саме:

- Згортковий шар (Convolutional layer)
- Об’єднуючий шар (Pooling layer)
- Повнозв’язний шар (Fully-connected layer)

Згортковий шар є першим шаром у мережі. За ним можуть слідувати додаткові згорткові шари або шари об’єднання. Повнозв’язаний шар є останнім. З кожним шаром CNN збільшується у своїй складності, ідентифікуючи більші частини вхідних даних. В той час як початкові шари згорткової нейронної мережі зосереджені на ідентифікації більш простих ознак, останні визначають складні комбінації цих простих ознак.

Згортковий шар є основним будівельним блоком CNN, де відбувається більшість обчислень. Складається з вхідних даних, фільтру і карти ознак. Якщо припустити, що вхідними даними буде кольорове зображення, яке складається з матриці пікселів у 3D, то дані будуть мати три виміри (або канали) — висоту, ширину та глибину — які відповідають RGB-параметрам у зображенні. Детектор ознак, також відомий як ядро або фільтр, який буде переміщатися по сприйнятливих полях зображення, перевіряючи, наявність ознак. Цей процес відомий як згортка.

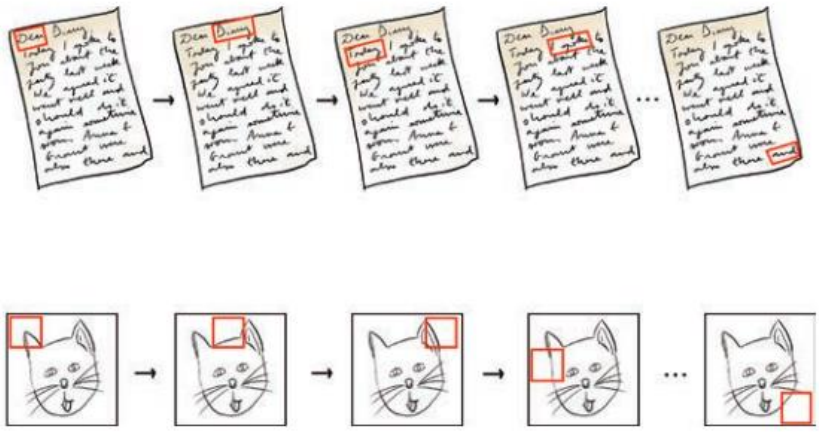


Рисунок 2.9 – Схематичне зображення процесу згортки

Іншими словами, детектор ознак — це двовимірний масив ваг, який представляє частину зображення. Фільтри можуть відрізнятися за розміром, але в більшості випадків визначаються матрицею розміром 3x3. Фільтр застосовується до області зображення, шляхом обчислення скалярного добутоку між ним та вхідними пікселями. Процес повторюється, кожного разу зміщуюси піксель на один крок, поки ядро не охопить усе зображення.

Остаточний вихід із ряду скалярних добутків називається картою ознак, картою активації або згорнутою ознакою.

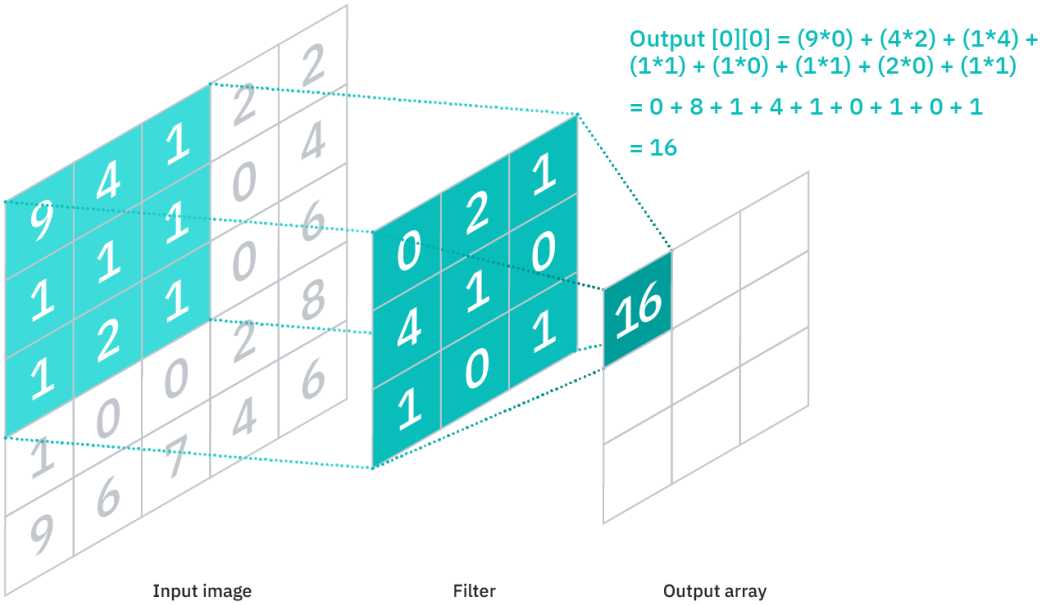


Рисунок 2.10 – Приклад обчислення карти ознак

Об’єднуючий шар, також відомий як шар суб дискретизації, зменшує розмірність, зменшуючи кількість параметрів у вхідних даних. Подібно до згорткового шару, операція об’єднання сканує вхідні дані за допомогою фільтру, проте в цьому випадку фільтр не має жодних ваг. Замість цього ядро застосовує функцію агрегації до значень у рецептивному полі, заповнюючи вихідний масив. Існує два основних типи об’єднання:

- Максимальне об’єднання (Max pooling): під час переміщення фільтра по вхідним даним на кожному кроці обирається піксель з максимальним значенням та записується до вихідного масиву. Цей підхід, як правило, використовується частіше в порівнянні з наступним.
- Середнє об'єднання (Average pooling): під час переміщення фільтра по вхідним даним, обчислюється середнє значення в рецептивному полі, яке і записується у вихідний масив.

Хоча багато інформації втрачається в об’єднуючому шарі, але це має ряд переваг для CNN, наприклад, зменшення складності, підвищення ефективність мережі та зменшення ризику перенавчання.

У повнозв’язному шарі кожен вихідний вузол зв’язаний з вузлом попереднього шару. Повнозв’язний шар виконує завдання класифікації на основі ознак, вилучених через попередні шари та їх фільтри. У той час як Convolutional layer та Pooling layer зазвичай використовують ReLu як функцію активації, у Fully-connected layer застосовується softmax для належної класифікації вхідних даних, повертаючи масив ймовірністей (значень між 0 і 1) приналежності об’єкта до кожного з заданих класів.

2.2.2 Рекурентні нейронні мережі

Рекурентні нейронні мережі (RNN) — це типи штучних нейронних мереж, які використовуються для роботи з послідовними даними чи даними часових рядів. Ці алгоритми глибокого навчання зазвичай використовуються для впорядкованих за часом завдань, таких як переклад текстів, обробка природної мови, розпізнавання мовлення; вони включені в популярні програми, такі як Siri, голосовий пошук і Google Translate. Рекурентні нейронні мережі відрізняються своєю «пам’яттю», оскільки беруть інформацію з попередніх входів, щоб впливати на поточний вхід і вихід. Тому, в той час як традиційні глибокі нейронні мережі розглядають входи та виходи вузлів незалежними один від одних, вихідні дані рекуррентних нейронних мереж залежать від попередніх елементів послідовності.

На рисунку 2.11 зелені блоки називаються прихованими станами. Сині кола, визначені вектором *a* всередині кожного блоку, називаються прихованими вузлами (кількість вузлів визначається гіперпараметром *d*). Кожен прихований стан можна сприймати як функцію активації, що діє на кожен прихований вузол. Вектор *h* — це вихід прихованого стану після застосування функції активації до прихованих вузлів.

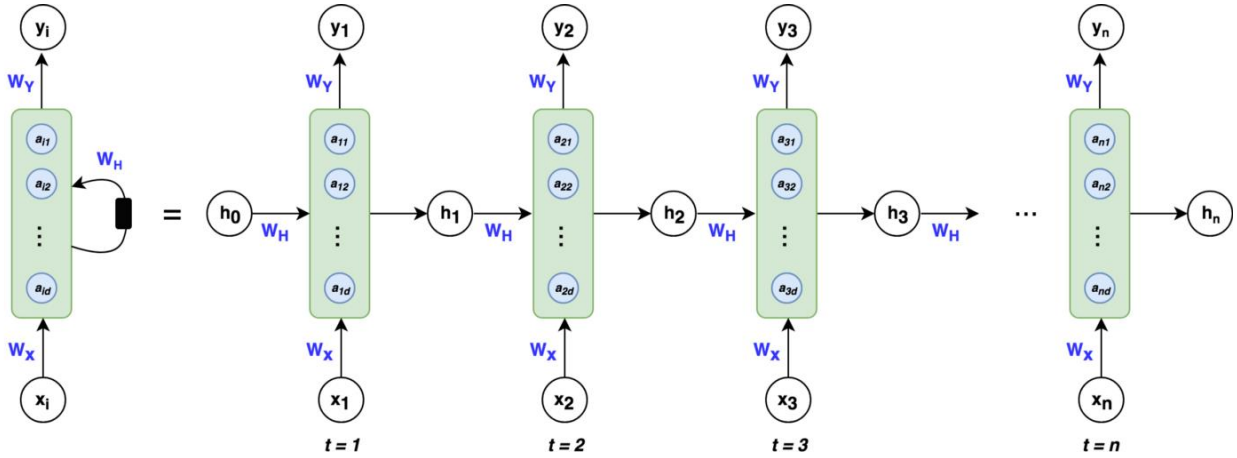


Рисунок 2.11 – Узагальнена схема нейронної мережі

В момент *t*, архітектура враховує те, що сталося в момент *t-1*, включаючи *h* з попереднього прихованого стану, а також вхід *x* в момент *t*. Це дозволяє враховувати інформацію з попередніх вводів, які послідовно відстають від поточного входу. Важливо зазначити, що вектор h_0 завжди буде починатися як вектор нулів, оскільки алгоритм не має інформації, що передує першому елементу в послідовності.

Матриці W_x, W_y, W_h — це коефіцієнти ваги RNN, які використовуються для всієї мережі. Тобто матриця ваг W_x при *t=1* не змінюється і при *t=2* і на кожному наступному кроці часу.

Вектор x_i— це вхідні дані для кожного прихованого стану, де *i=1, 2,..., n* для кожного елемента у вхідній послідовності.

На рисунку 2.12 зображені рівняння RNN. Приховані вузли є конкатенацією вихідних даних попереднього стану, зважених за ваговою матрицею W_h, і вхідних даних *x*, зважених за ваговою матрицею W_x. Результатом прихованого стану є функція активації *tanh*, застосована до прихованих вузлів. Передбачення – це вихід з поточного прихованого стану, зважений його ваговою матрицею W_y з активацією *softmax*. RNN можуть виконувати передбачення на будь-якому окремому етапі часу, проте це не завжди необхідно.

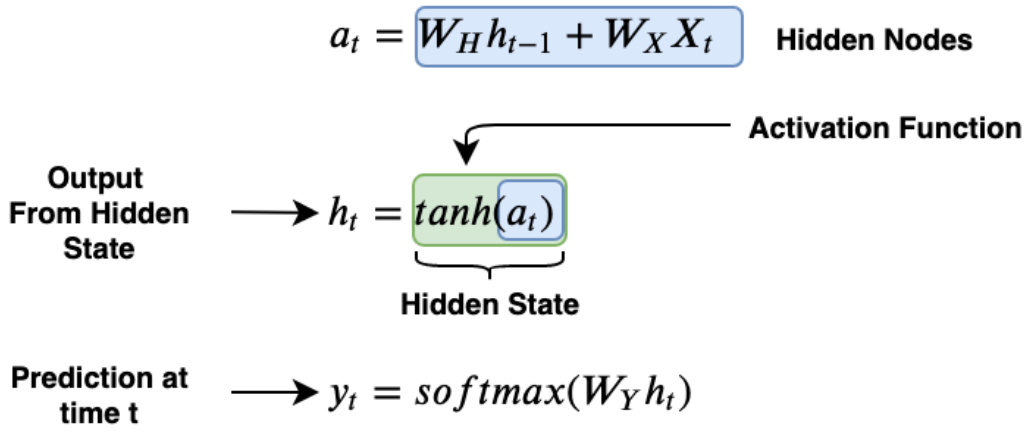


Рисунок 2.12 – Рівняння RNN

2.3 Технологічний стек

2.3.1 Мова програмування

Проєкти штучного інтелекту відрізняються від традиційних програмних проєктів. Відмінності полягають у наборі технологій, навичках, і необхідності глибокого дослідження. Саме тому при виборі мови програмування варто звертати увагу на її стабільність, гнучкість та доступний інструментарій. У цьому пункті мова Python буде проаналізована за даними критеріями.

1. Простота та послідовність

   Python пропонує стислий і читабельний код. Машинне навчання та системи штучного інтелекту часто використовують складні, комплексні алгоритми, в той же час Python дозволяє писати надійні системи за допомогою простих засобів. Це дозволяє досередитись на вирішенні проблеми машинного навчання замість технічних нюансів мови.

   Python є більш інтуїтивним, ніж інші мови програмування та вважається мовою загального призначення. Тому дає можливість

виконувати складні завдання з проєктування алгоримів та швидко створювати прототипи, які дозволять протестувати продукт для цілей машинного навчання.

2. Великий вибір бібліотек і фреймворків

   Добре структуроване та перевірене середовище є досить важливим для вирішення складних математичних задач та пошуку найкращих рішень.

   Python має надзвичайно багатий стек технологій та великий набір бібліотек, що дозволяє значно скоротити час розробки, зокрема:

   - Keras, TensorFlow і Scikit-learn для машинного навчання
   - NumPy для високопродуктивних наукових обчислень та аналізу даних
   - SciPy для просунутих обчислень
   - Pandas для аналізу даних загального призначення
   - Seaborn для візуалізації даних
   - Scikit-learn має різноманітні алгоритми класифікації, регресії та кластеризації, включаючи машини опорних векторів, випадкові ліси, підвищення градієнта, і призначений для роботи з числовими та науковими бібліотеками Python NumPy та SciPy.

3. Незалежність від платформи

   Одним із ключових факторів популярності Python є те, що це незалежна від платформи мова. Python підтримується Linux, Windows і macOS. Код Python можна використовувати для створення окремих виконуваних програм для більшості поширених операційних систем, що означає, що програмне забезпечення Python можна легко поширювати та використовувати в цих операційних системах без інтерпретатора Python.

Окрім того, для розробки моделей машинного навчання зазвичай використовуються сервіси Google або Amazon, однак часто можна знайти компанії та науковців, які надають перевагу власним машинам з потужними графічними процесорами (GPU). Той факт, що Python не залежить від платформи, робить таке навчання набагато дешевшим і легшим.

4. Гнучка інтеграція

   Проекти Python можна інтегрувати з іншими системами, закодованими різними мовами програмування. Крім того, оскільки Python є розширюваним і переносимим, може використовуватись для виконання міжмовних завдань. Адаптивність Python значно полегшує науковцям і розробникам навчання моделей машинного навчання.

5. Швидкість тестування

   Python надає багато інструментів для перегляду та тестування коду, таким чином можна швидко перевіряти правильність і якість коду.

6. Наявність гарних інструментів візуалізації

   Python поставляється з широким спектром бібліотек та пропонує хороші інструменти візуалізації. Деякі бібліотеки, такі як Matplotlib, дозволяють науковцям з даних створювати діаграми, гістограми та графіки. Крім того, різні API, які підтримує Python, покращують процес візуалізації.

7. Велика спільнота і популярність

   У опитуванні Developer Survey 2020, проведеному Stack Overflow, Python увійшов до п’ятірки найпопулярніших мов програмування. Є безліч форумів Python з активним обміном досвідом, пов’язаний із рішеннями машинного навчання. Для будь-якого завдання існує досить висока ймовірність того, що хтось інший уже мав справу з такою ж проблемою.

Доцільно буде порівняти Python з іншими мовами програмування, що можуть використовуватися для глибинного навчання:

Таблиця 2.1 – Порівняння інших мов для машинного навчання

Мова	Особливості	Недоліки
R	Застосовується для аналізу та маніпуляцій з даними для статистичних цілей	Досить повільний у роботі з великою кількістю даних.
	Пакети Gmodels, Class, Tm і RODBC дозволяють швидко впроваджувати алгоритми машинного	
	З ggplot2, ggvis, googleVis, Shiny, rCharts та ін. дає можливість будувати високоякісні графіки	
Scala	Дуже гарний інструмент для роботи з великими даними. Зокрема, пропонує ряд інструментів: Saddle, Scalalab і Breeze, та підтримку паралельності.	Набагато менша кількість інструментів для машинного навчання
Julia	Розроблена для обробки числових обчислювальних завдань	Не підтримується багатьма бібліотеками та поки не має сильної спільноти
	Синтаксис, подібний до Python	
	Надає підтримку глибокого навчання через обгортку TensorFlow.jl і фреймворк Mocha	
Java	Підтримується численними бібліотеками, такими як WEKA та Rapidminer	Не підтримує статистичне моделювання та візуалізацію
	Широко використовується для обробки природної мови, алгоритмів пошуку та нейронних мереж	
	дозволяє швидко створювати великомасштабні системи з відмінною продуктивністю	

2.3.2 Порівняння TensorFlow і PyTorch

TensorFlow — це наскрізна платформа глибокого навчання з відкритим кодом, розроблена Google і випущена в 2015 році. Вона відома підтримкою документації та навчання, масштабованими параметрами виробництва та розгортання, кількома рівнями абстракції та підтримкою різних платформ, таких як Android. TensorFlow також представляє бібліотеку, яка використовується для нейронних мереж і найкраще підходить для програмування потоків даних у ряді завдань. Він пропонує кілька рівнів абстракції для побудови та навчання моделей. TensorFlow перспективний і швидко зростаючий фреймворк у світі глибинного навчання, що пропонує гнучку, екосистему ресурсів спільноти, бібліотек та інструментів, які полегшують створення та розгортання програм машинного навчання.

PyTorch — це відносно нова платформа глибокого навчання, заснована на Torch. Розроблений дослідницькою групою AI Facebook і відкритий на GitHub у 2017 році, фреймворк використовується для додатків для обробки природної мови. PyTorch має репутацію простоти, зручності використання, гнучкості, ефективного використання пам’яті та динамічних обчислювальних графіків. Він також написаний на Python, що робить кодування більш керованим і збільшує швидкість обробки. Сьогодні PyTorch використовується для багатьох проектів глибокого навчання, і його популярність зростає серед дослідників штучного інтелекту, хоча з трьох основних фреймворків він найменш популярний.

Безперечною перевагою PyTorch є гнучкість, можливість налагодження та коротка тривалость навчання. Він працює на Linux, macOS та Windows.

В той же час завдяки добре задокументованій структурі та великій кількості навчених моделей і навчальних посібників TensorFlow має нижчий поріг входження та трохи більшу спільноту.

TensorFlow пропонує кращу візуалізацію, що дозволяє ефективніше налагоджувати та відстежувати процес навчання, тоді як PyTorch забезпечує обмежену візуалізацію.

TensorFlow також перевершує PyTorch у розгортанні навчених моделей у виробництві завдяки фреймворку TensorFlow Serving. PyTorch не пропонує таких інструментів, тому розробникам потрібно використовувати Django або Flask як бек-енд сервер.

У сфері паралелізму даних PyTorch отримує оптимальну продуктивність, покладаючись на вбудовану підтримку асинхронного виконання через Python. Із TensorFlow доведеться вручну кодувати та оптимізувати кожну операцію, щоб забезпечити розподілене навчання.

Таблиця 2.2 – Узагальнене порівняння PyTorch і TensorFlow.

Характеристика	PyTorch	TensorFlow
Рівень API	Низький	Високий і низький
Архітектура	Складна, менш читабельна	Не проста у використанні
Набори даних	Великі набори даних, висока продуктивність	Великі набори даних, висока продуктивність
Налагодження	Хороші можливості налагодження	Важко провести налагодження
Наявність навчених моделей	Так	Так
Популярність	Третій за популярністю	Другий за популярністю

Кінець таблиці 2.2.

Характеристика	PyTorch	TensorFlow
Швидкість	Швидкий, високопродуктивний	Швидкий, високопродуктивний
Написаний на	Lua	C++, CUDA, Python

Отже, зважаючи на вищенаведені характеристики, у даній дипломній роботі для проєктування та навчання нейронної мережі буде використано фреймворк TensorFlow.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі було розглянуто поняття нейронних мереж, визначено методи та технології, що будуть використані у даній дипломній роботі.

В основі нейронних мереж лежить ідея біологічних нейронів та зв’язків між ними. Загальний принцип обробки інформації штучними нейронами можна описати 3 кроками:

1. Отримання інформації від інших нейронів (попереднього шару мержі).
2. Обробка вхідних даних (через матрицю ваг і зсуви)
3. Обчислення функції активації та відповідна активація і передача значень при перевищенні граничного значення.

Першою практичною реалізацією можна вважати модель перцептрону, яка нині майже не використовується на практиці через обмеженість можливостей. Натомість дослідження та вдосконалення методів побудови нейронних мереж триває. Зокрема, використовуються різні функції активації, різні методи навчання та перевірки якості, детальний огляд яких було проведено в даному розділі.

Окрім того з’являються нові складніші та ефективніші архітектури нейронних мереж, що дозволяють вирішувати комплексні нелінійні задачі з високою продуктивністю. Згорткові нейронні мережі дозволяють простіше та ефективніше працювати з дво- та тривимірними масивами у якості вхідних даних і активно використовуються у аналізі зображень, звуку та відео. Рекурентні нейронні мережі дозволяють працювати з послідовностями даних, та є надзвичайно ефективними у обробці живої мови, текстів, тощо. З проаналізованих раніше прикладів можна зробити висновок, що обидві архітектури показують гарні результати у вирішенні задачі цієї дипломної роботи.

Окрім того було проаналізовано ефективність мови програмування Python, яка через свою простоту, гнучкість, великий вибір бібліотек та засобів, для візуалізації даних та машинного навчання зокрема, а також велику популярність є безперечно найкращим варіантом для поставленої у даній дипломній роботі задачі.

При порівнянні двох фреймворків для побудови нейронних мереж для даного технічного завдання було обрано TensorFlow. Він пропонує краще можливості візуалізації, простішу архітектуру, є швидким та високопродуктивним.

РОЗДІЛ 3
ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

Далі будуть детально розглянуто процес створення, навчання та тестування згорткової нейронної межежі для підбору акордів пісень.

3.1 Пошук датасету та обробка даних

Для навчання та тренування використовуватимуться колекції пісень The Beatles та Queen та їх акордові транскрипції з відомого та широко використовуваного у сфері аналізу музики датасету Isophonics. Окрім того даний набір даних було доповнено колекцією популярних рок та поп пісень, що була представлений на одному з етапів конкурсу MIREX. Остаточний датасет представляє собою набір еталонних даних, що складається з 379 пісень, проаналізованих командою професійних музикантів, транскрибованих і збережених у форматі:

Початок_звучання–Кінець_звучання–Акорд

	Start	End	Chord
0	0.000000	1.053119	N
1	1.053119	3.593854	B:min
2	3.593854	6.090000	G
3	6.090000	8.655804	E
4	8.655804	11.140340	A
5	11.140340	13.659705	A
6	13.659705	16.109410	C#:min
7	16.109410	18.686825	F#:min

Рисунок 3.1 – Транскрипція акордів перших 20-ти секунд пісні “Help!” The Beatles

Всього в датасеті представлено більше 900 класів акордів. Як можна побачити з рисунку, більшість з них є комплексними, а значить складними для розпізнавання нейронною мережею. Окрім того більшість із них зустрічається в транскрипціях дуже рідко, що додатково ускладнює процес навчання.

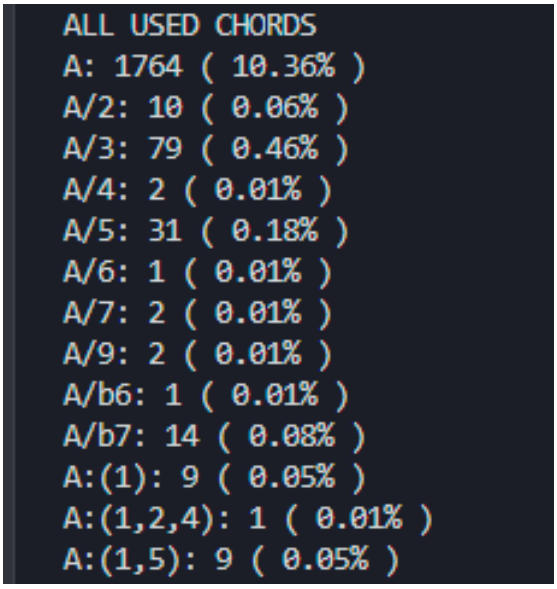


Рисунок 3.2 – Частина списку усіх використаних акордів у датасеті та їх кількість.

Саме тому усі складні акорди з датасету було спрощено до простих за загальними правилами спрощення музичних акордів (рисунок)

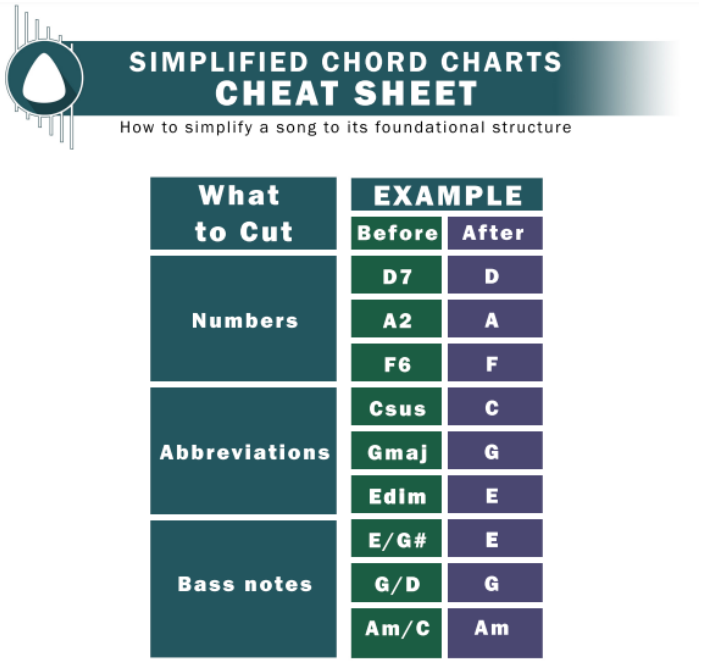


Рисунок 3.3 – Принципи спрощення комплексних акордів

Таким чином оригінальні анотації, були скорочені до міток з 24 акордів та одного N-класу, що позначає відсутність акорду відповідно. Акорди, що позначають один звук (дієзи і бемолі) також були винесені в один клас.

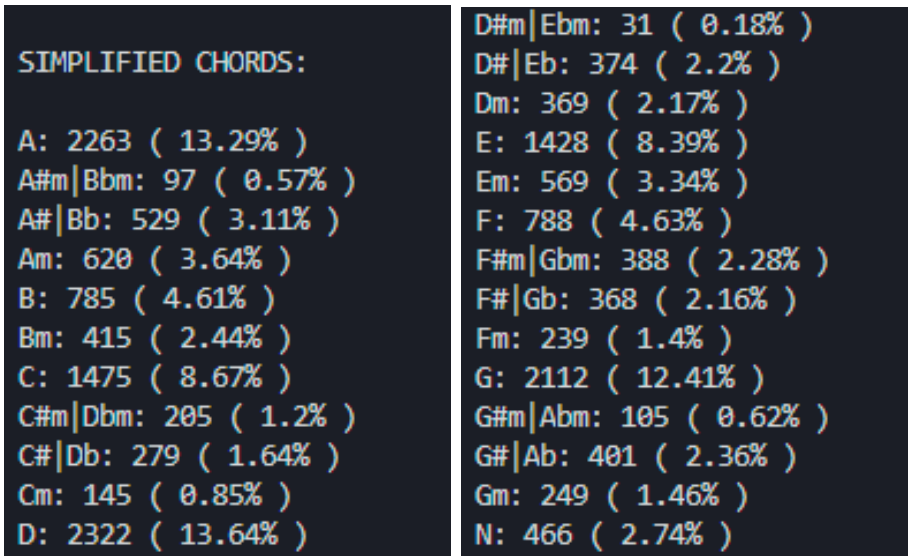


Рисунок3.4 – Спрощені класи акордів

3.2 Підготовка навчальних та тренувальних даних

Перш за все необхідно визначити ознаки на основі яких буде навчатися мережа. У даній дипломній роботі розпізнавання акордів відбуватиметься за допомогою мел-частотні кепстральних коефіцієнтів (Mel-Frequency Cepstral Coficients – MFCCs). Дана метрика спочатку використовувалась в різних техніках обробки мовлення, однак, оскільки область вилучення інформації з музичних композицій (MIR) активно розвивалась на додаток до машинного навчання, було виявлено, що MFCC можуть досить добре представляти тембр, зокрема, та інші характеристики. Звісно, основна перевага MFCCs перед іншими можливими ознаками – це досить невеликий об’єм даних, що в той же час надає досить багато інформації

Дані в датасеті все ще залишаються невирівняними по часу звучання. Вхідні дані нейронної мережі мають мати однакові розміри, що в свою чергу визначатиме кількіть вхідних нейронів першого шару. Тому усі аудіофійли з

піснями будуть поділені на фрейми (розміром 0.2с), відповідні файли транскрипцій також переформовані відповідно до розмірів фрейму.

Алгоритм дій (для кожного аудіофайлу та транскрипції акордів):

1. Поділити аудіосигнал на фрейми, заданої довжини – функція *cut_into_frames()* з файлу *create_dataset.py*

   Для кожного фрейму будуть обчислені мел-кепстральні коефіцієнти (MFCCs) – метрика, що часто використовується в аналізі звукових файлів, на рівні з хронограмою, висотними класами та мел-спектрограмами.
2. Поділити транскрипції акордів відповідно до розмірів фреймів – функція *framing()* з файлу *create_dataset.py*

   Для кожного акорду в транскрипції порахувати *n = ціла кількість фреймів, що вміщається в тривалість його звучання*. Залишок (у секундах) додати до тривалості наступного акорду. Кожен акорд записати в кінцевий масив *n* разів.

Записати отримані з пунктів 1 та 2 масиви (масив коефіцієнтів MFCCs та масив акордів) у структуру словник – функція create_dataset() з файлу create_dataset.py – у поля "MFCCs" та "chords" відповідно.

Зберегти як файл .json – функція save_json() з файлу create_dataset.py.

Отже, таким чином було створено масив даних (features і targets) для нейронної мережі.

3.3 Побудова моделі нейронної мережі

Тип архітектури нейронної мережі, що було обрано для вирішення заданої задачі – CNN або згорткова нейронна мережа. Вхідні дані – Мел-частотні кепстральні коефіцієнти – представляють собою матрицю з числами, іншими словами, можуть бути інтерпритовані як зображення. Саме тому очікується, що дана архітектура буде більш ефективною

Алгоритм дій:

1. Підготовка датасету - функція *prepare_dataset()* з файлу *cnn_chord_recognition.py*

   Розділення даних на тренувальні, тестові та валідаційні за допомогою функціїї *train_test_split()* з модуля *sklearn.model_selection*

2. Важливим етапом проєктування є нормацізація вхідних і вихідних даних – функція *main* з файлу *cnn_chord_recognition.py*

```
if __name__ == "__main__":

    # get train, validation, test splits
    X_train, X_validation, X_test, y_train, y_validation, y_test = prepare_datasets(0.3,

    mean = np.mean(X_train, axis=0)
    std = np.std(X_train, axis=0)
    X_train = (X_train - mean)/std

    mean = np.mean(X_validation, axis=0)
    std = np.std(X_validation, axis=0)
    X_validation = (X_validation - mean)/std

    mean = np.mean(X_test, axis=0)
    std = np.std(X_test, axis=0)
    X_test = (X_test - mean)/std


    n_classes = 25
    y_train = keras.utils.to_categorical(y_train, n_classes)
    print(y_train)
    y_validation = keras.utils.to_categorical(y_validation, n_classes)
    y_test = keras.utils.to_categorical(y_test, n_classes)
```

Рисунок 3.5 – Нормалізація вхідних і вихідних даних

В даному впадку функція to_categorical() перетворила кожен елемент масиву y (масиву target) в одиничні вектори, з різною позицією одиниці – в залежності від класу.

Для вхідних даних нормалізація буде обчислюватись за формулою:

$$x = \frac{x - \text{середнє значення}}{\text{середнє квадратичне відхилення}}$$

Нормалізація є дуже корисною, адже допомагає будувати набагато ефективніші нейронні мережі.

3. Побудова моделі – функція build_model()

```
model = keras.Sequential()

############### 1 ########################
model.add(keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape, padding='same'))
model.add(keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(keras.layers.MaxPooling2D((2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())

############### 2 ########################
model.add(keras.layers.Conv2D(64, (5, 5), activation='relu', padding='same'))
model.add(keras.layers.Conv2D(64, (5, 5), activation='relu', padding='same'))
model.add(keras.layers.MaxPooling2D((2, 2), padding='same'))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Dropout(0.2))

############### 3 ########################
model.add(keras.layers.Conv2D(128, (7, 7), activation='relu', padding='same'))
model.add(keras.layers.Conv2D(128, (7, 7), activation='relu', padding='same'))
model.add(keras.layers.MaxPooling2D((2, 2), padding='same'))
model.add(keras.layers.Dropout(0.3))

############### 4 ########################
model.add(keras.layers.Flatten())

############### 5 ########################
model.add(keras.layers.Dense(128, activation='relu'))
model.add(keras.layers.Dropout(0.2))

############### 6 ########################
model.add(keras.layers.Dense(25, activation='softmax'))
```

Рисунок 3.6 – Загальна структура моделі

Нейронна мережакладається з чотирьох згорткових шарів, за якими слідують шари субдискретизації, batch-нормалізації та Dropout, 2 останні шари – повнозв’язні.

4. Компіляція моделі та тренування моделі

   Як оптимізатор використано Adam (adaptive moment estimation). Метод Adam має два гіперпараметри, по одному для обчислення кожного з ковзних середніх. Для них використовуються такі значення за промовчанням: $\beta1 = 0.9$ та $\beta2 = 0.999$. Для швидкості навчання в методі Adam за замовчуванням використовується значення $\eta = 0.001$

Датасет розділено на тренувальні, валідаційні та тестові дані як 0.5, 0.2, 0.3 відповідно.

ВИСНОВОК ДО РОЗДІЛУ 3

В результаті проведеної роботи щодо розробки нейронної мережі для підбору музичних акордів пісень було виконано наступне:

- Знайдено та завантажено датасет, на основі якого буде навчатися нейронна мережа.
- Оброблено дані з датасету та замінено складні акорди простими.
- Спроєктовано згорткову нейронну мережу для розпізнавання. акордів.
- Нормалізовано вхідні та вихідні дані.

У наступному розділі нейронну мережу та її компоненти буде протестовано і виведено загальні резальтати.

РОЗДІЛ 4
АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

Базуючись на розробленій моделі нейронної мережі для підбору акордів пісень метою даного розділу є оцінка результатів її роботи.

4.1 Схема нейронної мережі для розпізнавання акордів

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 18, 40, 32)        320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 18, 40, 32)        9248
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 18, 40, 32)        9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 9, 20, 32)         0
_________________________________________________________________
batch_normalization (BatchNo (None, 9, 20, 32)         128
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 9, 20, 64)         51264
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 9, 20, 64)         102464
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 10, 64)         0
_________________________________________________________________
batch_normalization_1 (Batch (None, 5, 10, 64)         256
_________________________________________________________________
dropout (Dropout)            (None, 5, 10, 64)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 5, 10, 128)        401536
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 5, 10, 128)        802944
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 5, 128)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 3, 5, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 1920)              0
_________________________________________________________________
dense (Dense)                (None, 128)               245888
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 25)                3225
=================================================================
Total params: 1,626,521
Trainable params: 1,626,329
Non-trainable params: 192
```

Рисунок 4.1 – Схема розробленої мережі

На рисунку 4.1 зображено побудовану схему нейронної мережі для розпізнавання акордів у піснях. Мережа складається з чотирьох шарів згортки, чотирьох шарів субдискретизації, та двох повнозв’язних шарів.

Після кожного шару згортки + субдискретизації застосовується прийом пакетної нормалізації (Batch normalization). Пакетна нормалізація дозволяє шарам вчитися більш незалежно один від одного, бо, наприклад, великі значення в одному шарі не надмірно впливають на обчислення в наступному; дозволяє встановити більш високу швидкість навчання, тому що в нормованих активаціях немає екстремальних значень; вихідні дані шару нормалізуються за середнім значенням і стандартним відхиленням пакета, що додає елемент шуму (особливо при невеликих розмірах пакетів), що, у свою чергу, сприяє регуляризації, що в свою чергу допомагає мережі узагальнювати дані, яких вона ще не бачила.

Після нормалізації застосовується проріджування (Dropout), що дозволяє уникати перенавчання, шляхом виключення певного відсотка вибраних випадковим чином нейронів.

4.2 Функція точності та функція втрат

Ефективність моделі переважно визначається функціями точності (accuracy) та втрат (loss).

Функція втрат показує середню вартість на навчальних даних для епохи, а точності – кількість правильно класифікованих даних. Очевидно, що метою навчального процесу є мінімізація функції втрат та максимізація точності. На рисунку 4.2 можна побачити результати для побудованої моделі

```
5997/5997 [==============================] - 6s 944us/sample - loss: 0.6773 - accuracy: 0.7604 - val_loss: 0.8092 - val_acc
2499/2499 [==============================] - 1s 259us/sample - loss: 0.8683 - accuracy: 0.6907
Accuracy on test set is: 0.6906762719154358
Expected index: 9, Predicted index: [9]
```

Рисунок 4.2 – Результат навчання

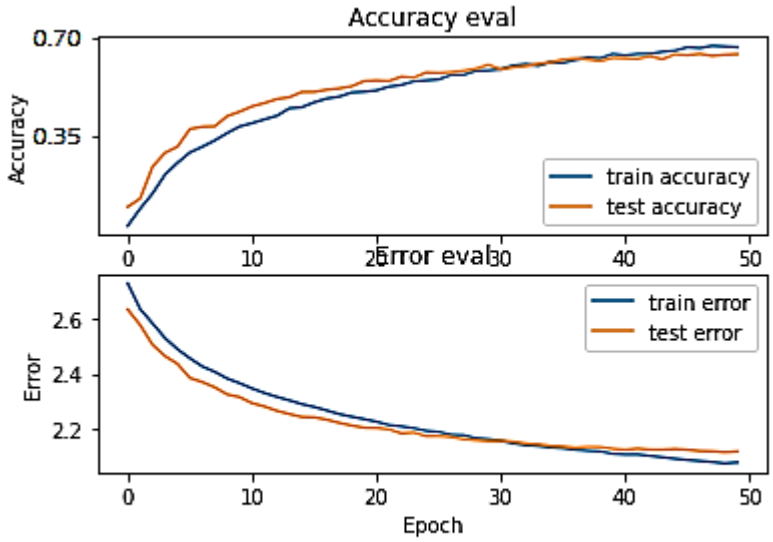


Рисунок 4.3 – Зміна функції втрат та точності.

Детальні зміни згаданих метрик з кожною епохою можна дослідити на рисунку 4.3. Як видно з рисунку під час всього процесу навчання величина втрат постійно зменшується, тоді як точність визначення даних зростає. Обидва графіки для тренувальних та тестових даних подібні за формою та змінюються синхронно. Це означає, що перенавчання не відбувається, і результати, що демонструє мережа приблизно однакові і при тренуванні і при перевірці.

4.3. Рекомендації щодо вдосконалення мережі

З метою покращення роботи спроєктованої системи, можна запропонувати деякі покращення.

По-перше, поекспериментувати з кількістю MFCC- коефіцієнтів, адже це може надати більше інформації і відповідно збільшити точність класифікації.

По-друге, бажано розширити набір даних. Очевидно, що чим більша різноманітність даних, тим краще система покаже себе у вирішенні реальних завдань, що, безумовно, і є головною ціллю будь-якого проєктування.

Складність, на жаль, полягає у тому, що подібних датасетів з транскрибованими піснями доволі обмежена кількість.

По-третє, можна збільшити кількість акордів, що здатна розпізнати мережа. Це завдання перш за все ускладнюється нерівномірністю використання різних акордів. Іншими словами складні акорди зустрічаються і будуть зустрічатися набагато рідше, ніж прості. Тому доцільно буде вдатися до більш складної архітектури нейронної мережі, зокрема згортково рекурентних мереж, або комбінації з прихованими марківськими моделями.

ВИСНОВОК ДО РОЗДІЛУ 4

У даному розділі було розглянуто побудовану модель нейронної мережі для підбору акордів пісень. Модель представляє собою згорткову нейронну мережу з 4 шарами згортки.

Для аналізу ефективності системи використовувались значення кінчевої точності мережі та втрат. Втрати рахувались методом перехресної ентропії.

Дана нейронна мережа змогла правильно визначити приблизно 70% акордів зі створеного датасету, що можна вважати досить гарним результатом. Функція втрат при цьому зменшилась до 0.886.

ВИСНОВКИ

Отже, у даній дипломній роботі було розроблено та реалізовано згорткову нейронну мережу для розпізнавання музичних акордів у піснях. Наразі дана система здатна розпізнати 25 типів акордів: 12 мажорних і 12 мінорних тризвуків та один “не акорд”.

У першому розділі розглянуто принципи роботи зі звуком: дискретизація вхідного сигналу, короткочасне перетворення Фур’є, перехід до мел-шкали частот. Також оглянуто основи теорії музики, зв’язок нот з частотами, правила побудови простих акодів. Надано визначення завдання автоматичного розпізнавання музичних акордів та освітлено основні методи вирішення, зокрема Приховані моделі Маркова та нейронні мережі. Також у розділі проаналізовано існуючі рішення визначеної задачі за допомогою згорткових та рекурентних нейронних мереж.

У другому розділі розглянуто поняття нейронних мереж, їх винайдення, будови та принципу роботи. Зроблено порівняння популярних активаційних функцій, та описано процес навчання мереж, оцінка їх ефективності. Також досить детально описано архітектури та принципи роботи згорткових та рекурентних нейронних мереж, згадано області їх застосування. Окрім того, у даному розділі обгрунтовано вибір використовуваних технологій, зокрема мови програмування Python. Розглянуто преваги використання Python в розробці проєктів глибинного навчання та порівняно його з іншими альтернативами у даній області. Проведено порівняльний аналіз фреймворків PyTorch та TensorFlow.

У третьому розділі детально розглянуто процес побудови системи розпізнавання акордів на основі штучної нейронної мережі. Коротко оглянуто вибраний для навчання датасет. Описано процес передобробки навчальних даних: спрощення складних акордів, зверення звуків з різними назвами, але

однаковим звучанням до одного класу. Описано процес створення набору навчальних даних з музичних аудіофайлів та їх акордних транскрипцій: поділ кожної композиції на фрейми, розміром 0.2с, обчислення коефіцієнтів MFCC на кожному аудіофреймі, та створення масивів features і targets (ознак та цілей) та експорт їх у JSON-файл. Наступним кроком розглянуто модель згорткової нейронної мережі, побудованої для вирішення задачі розпізнавання акордів.

У четвертому розділі проведено оцінку ефективності моделі на основі функцій втрат та точності, за якою розроблена система показала добрий результат.

Таким чином, розроблену систему можна вважати відповідною всім зазначеним на початку вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Eric J. Humphrey. Rethinking automatic chord recognition with convolutional neural networks : 11th International Conference on Machine Learning and Applications / Eric J. Humphrey, Juan P. Bello // IEEE. – 2012.
2. M. McVicar. Automatic Chord Estimation from Audio: A Review of the State of the Art. / M. McVicar, R. Santos-Rodriguez, Y. Ni, and T. D. Bie. // IEEE/ACM Transactions on Audio, Speech, and Language Processing 22. – 2014. – С. 556 – 575.
3. J Osmalskyj, J-J Embrechts, S Piérard, M van Droogenbroeck. Neural networks for musical chords recognition//Journées d'Informatique Musicale, 2012, Mons, France//ffhal03041758f
4. Heng-Tze Cheng, Yi-Hsuan Yang, Yu-Ching Lin, I-Bin Liao , and Homer H. Chen. Automatic chord recognition for music classification and retrieval. [Електронний ресурс] – Режим доступу до ресурсу: https://users.ece.cmu.edu/~hengtzec/papers/icme08_chord.pdf
5. Alexander Sheh and Daniel P.W. Ellis LabROSA, Dept. of Electrical Engineering. Chord Segmentation and Recognition using EM-Trained Hidden Markov Models. Columbia University, New York NY 10027 USA. [Електронний ресурс] – Режим доступу до ресурсу: https://www.ee.columbia.edu/~dpwe/pubs/ismir03-chords.pdf
6. N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Audio chord recognition with recurrent neural networks," in Proc. of the 14th ISMIR, Curitiba, Brazil, 2013. [Електронний ресурс] – Режим доступу до ресурсу: http://www-ens.iro.umontreal.ca/~boulanni/ISMIR2013.pdf
7. J. Osmalskyj, J-J. Embrechts, S. Piérard, M. Van Droogenbroeck INTELSIG Laboratory, University of Liège, Departement. EECS NEURAL NETWORKS FOR MUSICAL CHORDS RECOGNITION. Actes des

Journées d'Informatique Musicale (JIM 2012), Mons, Belgique, 9-11 mai 2012. [Електронний ресурс] – Режим доступу до ресурсу: https://hal.archives-ouvertes.fr/hal-03041758/document

8. Filip Korzeniowski and Gerhard Widmer. A FULLY CONVOLUTIONAL DEEP AUDITORY MODEL FOR MUSICAL CHORD RECOGNITION. 2016 Ieee International Workshop On Machine Learning For Signal Processing, Sept. 13–16, 2016, Salerno, Italy.[Електронний ресурс] – Режим доступу до ресурсу: http://www.cp.jku.at/research/papers/Korzeniowski_MLSP_2016.pdf
9. Chord recognition using duration-explicit hidden markov models: 10th International Society for Music Information Retrieval / Ruofeng Chen, Weibin Shen, Ajay Srinivasamurthy, Parag Chordia. // ISMIR. – 2012. – С. 445 – 460.
10. Eric J. Humphrey. Rethinking automatic chord recognition with convolutional neural networks: 11th International Conference on Machine Learning and Applications / Eric J. Humphrey, Juan P. Bello // IEEE. – 2012.
11. L. Oudre. Chord recognition by fitting rescaled chroma vectors. / Laurent Oudre, Yves Grenier, and Cédric Févotte. // IEEE transaction on Audio, Speech and Image processing. – 2011. – №19 – С. 2222 – 2223.
12. O. Khadkevich. Time-frequency reassigned features for automatic chord recognition: за матеріалами міжнар. наук.-практ. конф IEEE International Conference on Acoustics, Speech and Signal Processing / Omologo Khadkevich. // ICASSP. – 2011.
13. Nikolay Glaziryn. Audio chord estimation using chroma reduced spectrogram and self-similarity: за матеріалами міжнар. наук.-практ. конф. Music Information Retrieval Evaluation Exchange / Nikolay Glaziryn. // MIREX. – 2012.

14. T. Cho. On the Relative Importance of Individual Components of Chord Recognition Systems. / T. Cho and J. P. Bello // IEEE/ACM Transactions on Audio, Speech, and Language Processing 22. – 2014. – С. 477 – 492.
15. Takuya Fujishima. Realtime chord recognition of musical sound: a system using common lisp music / Takuya Fujishima. // In proceedings of international computer music association (ICMC) – 1999. – С. 464 – 467.

ДОДАТОК 1

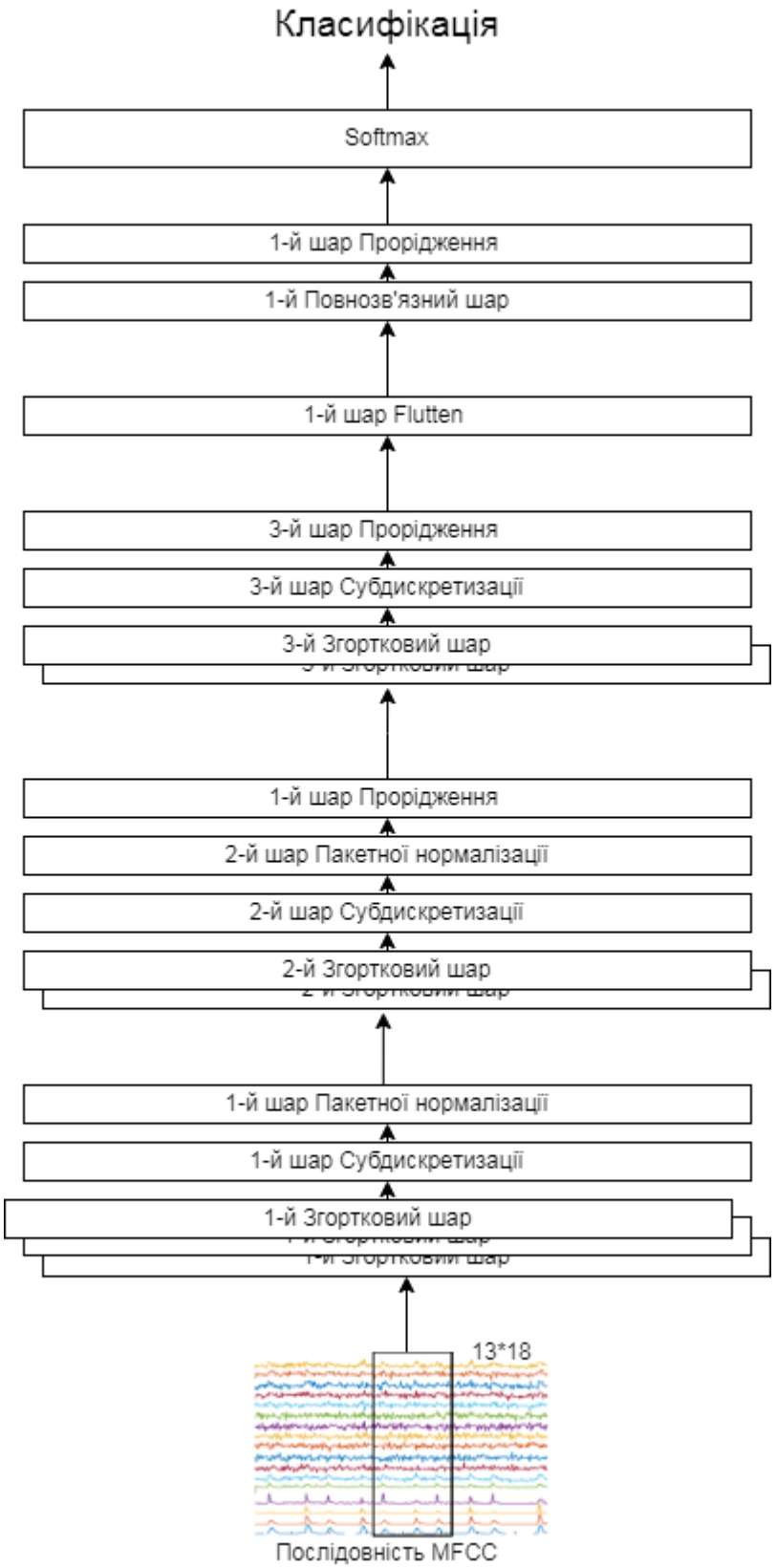
Нейронна мережа для підбору акордів пісень

Структурна схема системи

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.004 Д1	
		№ докум.	Підпис	Дата		
Розробила		Шахова П.М.			**Нейронна мережа для підбору акордів пісень** Структурна діаграма системи	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Волокита А. М.				
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-84
Н. Контр.		Сімоненко В. П.				
Затвердив						

ДОДАТОК 2

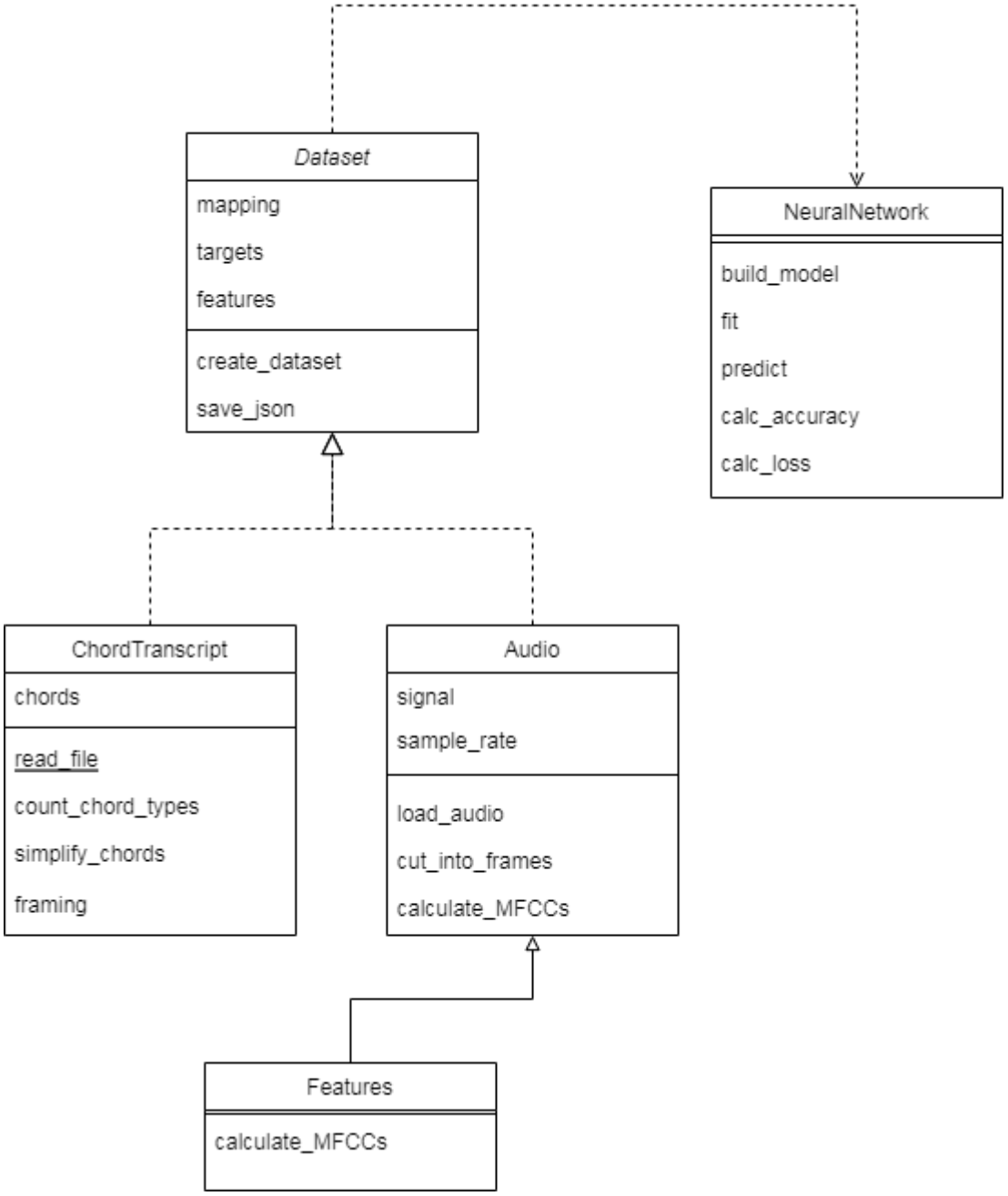
Нейронна мережа для підбору акордів пісень

Функціональна схема (діаграма класів)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.005 Д2	
		№ докум.	Підпис	Дата		
Розробила		Шахова П.М.			**Нейронна мережа для підбору акордів пісень** Діаграма модулів	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Волокита А. М.				
Н. Контр.		Сімоненко В. П.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-84
Затвердив						

ДОДАТОК 3

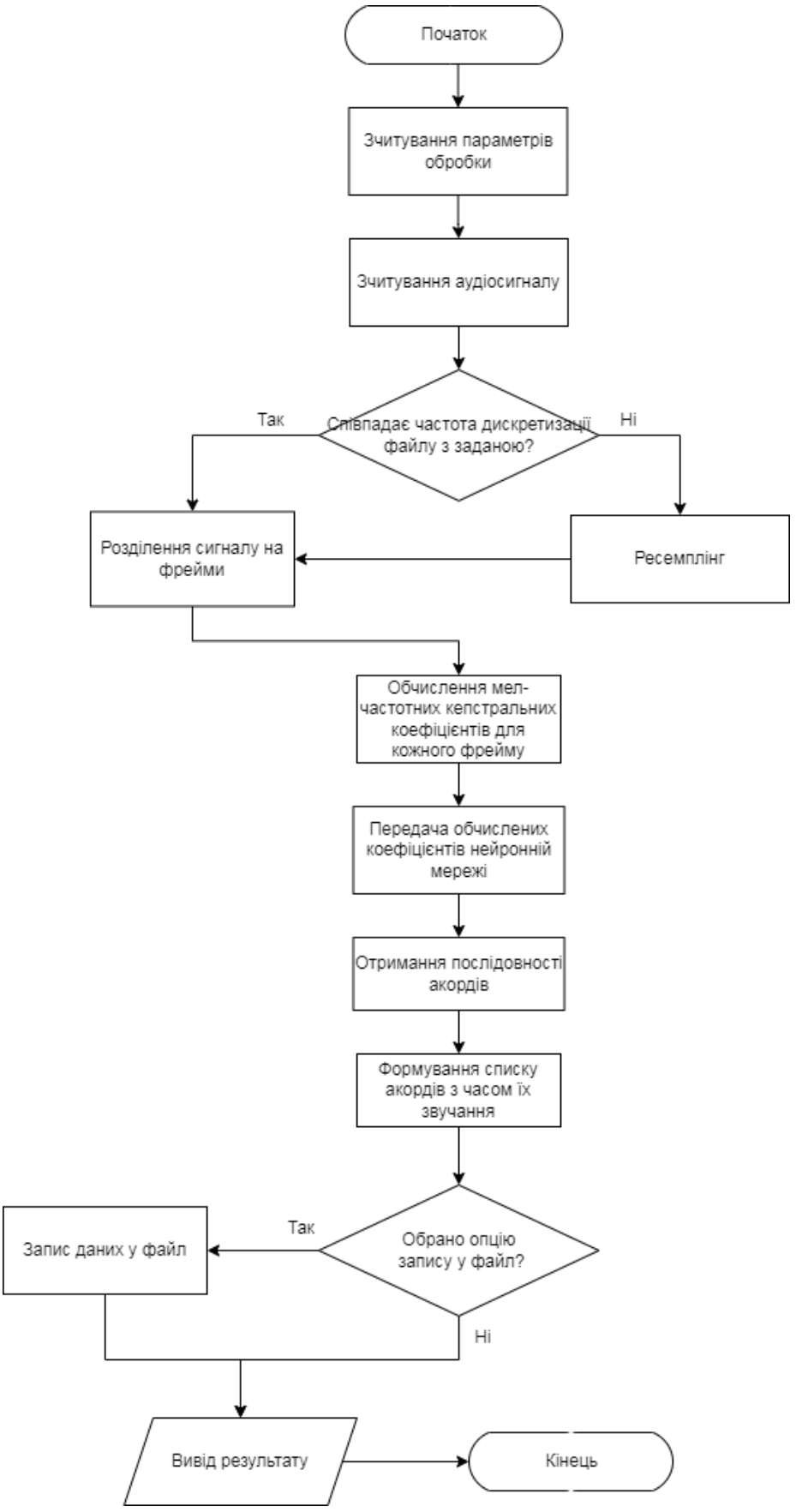
Нейронна мережа для підбору акордів пісень

Алгоритм дій програмного забезпечення

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.006 Д3	
		№ докум.	Підпис	Дата		
Розробив		Шахова П.М.			**Нейронна мережа для підбору акордів пісень** Послідовність дій програм-ного забезпечення	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Волокита А. М.				
Н. Контр.		Сімоненко В. П.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-84
Затвердив						

ДОДАТОК 4

Нейронна мережа для підбору акордів пісень

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 8

Київ 2022 р

```
chord_transcript.py
import csv
import numpy as np
import os
import re

CHORDS_DIR = 'data/chord_transcription'
FILE_NAME = "test_chord_list.lab"


def _read_file(song_path):
    try:
        with open(song_path, "r") as csv_file:
            try:
                data = csv.reader(csv_file, delimiter=" ")
                chord_transcription = [[float(row[0]), float(row[1]), str(row[2])] for row
in data]
                return chord_transcription
            except:
                print(f"Problem in processing {os.path.basename(song_path)}.\n"
                      "Data format is incorrect!")
    except OSError:
        print(f"Can't open {os.path.basename(song_path)}")

# t = read_song(FILE_NAME)
# t = np.array(t)[:, 2]
# print(t.tolist())

def count_unique_values(data):
    data = np.array(data).reshape(-1)
    # res = {}
    values, counts = np.unique(data, return_counts=True)
    return dict(zip(values, counts))

def _get_column(matrix, index):
    return [row[index] for row in matrix]

def get_chord_transcripts(dirname):
    transcriptions = {}
    for filename in os.scandir(dirname):
        if filename.is_file():
            t_chords = _read_file(filename)
            transcriptions[os.path.basename(filename)] = t_chords
    return transcriptions

def count_chord_types(song_transcripts_arr):
    all_used_chords = []
    for st in song_transcripts_arr:
        all_used_chords += _get_column(st, 2)
    chord_counts = count_unique_values(all_used_chords)
    return chord_counts

def _simplify_chord(chord):
    base = re.search("^[A-GN][#b]?", chord).group()   # find the base note in chord label)
    n = ["A", "B", "C", "D", "E", "F", "G"]
    if re.search(".*min.*|^[A-G][b#]?m.*", chord):    # if the chord is minor
        base += "m"                                   # add "m" to the base note
```

```
    if "#" in base:
        n.reverse()
        index = n.index(base[0]) - 1
        base += re.sub("[A-G]#", "|" + n[index] + "b", base)
    elif "b" in base:
        index = n.index(base[0]) - 1
        base = re.sub("[A-G]b", n[index] + "#", base) + "|" + base

    return base

def simplify_song(chord_transcript):
    for row in chord_transcript:
        row[2] = _simplify_chord(row[2])



# Getting a chord transcriptions for every song in directory
chord_transcriptions = get_chord_transcripts(CHORDS_DIR)


# Count and print all chords and how many times they had been used
print('ALL USED CHORDS')
chord_types = count_chord_types(chord_transcriptions.values())
[print(f'{key}: {value} ( {round(value*100/sum(chord_types.values()), 2)}% )')
                                            for key, value in chord_types.items()]

# Simplify complicated chords: replace them with one of the list of 24 base chords
[simplify_song(transcript) for transcript in chord_transcriptions.values()]

print('\nSIMPLIFIED CHORDS:\n')
simplified_chord_types = count_chord_types(chord_transcriptions.values())
[print(f'{key}: {value} ( {round(value*100/sum(simplified_chord_types.values()), 2)}% )')
                                            for key, value in
simplified_chord_types.items()]

chord_classes = list(simplified_chord_types.keys())
```

create_dataset.py

```
import json
from chord_transcript import chord_transcriptions, chord_classes
import librosa
import numpy as np
import os


AUDIO_DIR = 'data/audio_examples_wav'
AUDIO_TEST = 'data/audio_test'
JSON_PATH = 'data/dataset1.json'
filename = "GnR_test.wav"

SAMPLE_RATE = 44100
N_FFT = 2048
HOP_LENGTH = 512
WINDOW_SIZE = 0.2
```

```
########## TRANSCRIPTIONS PROCESSING FUNCTIONS ##########
def calc_duration(transcription):
    return transcription[-1][1]

def find_transcript(filename, transcriptions):
    key = filename[:-4] + ".lab"
    return transcriptions[key]

def use_window(transcript, window_size):
    windowed_chords = []
    delta = 0
    for row in transcript:
        chord_duration = row[1] - row[0]
        chord = row[2]

        chord_duration += delta
        windows_pro_chord = int((chord_duration ) // window_size)
        windowed_chords += [chord]*windows_pro_chord
        delta = chord_duration % window_size
    return windowed_chords

def get_classification_vector(chord, mapping):
    vector = np.zeros(len(mapping))
    # vector = [0 for _ in range(len(mapping))]
    index = mapping.index(chord)
    vector[index] = 1
    return vector

def get_chord_class_number(chord, mapping):
    return mapping.index(chord)


############### AUDIO PROCESSING FUNCTIONS ###############
def load_audio(filename, sample_rate, transcript):
    tr_duration = calc_duration(transcript)
    print('Transcript duration: ', tr_duration)
    signal, sr = librosa.load(filename, sr=sample_rate, duration=tr_duration)
    audio_duration = librosa.get_duration(signal, sr)
    print('Audio duration: ', audio_duration)
    return signal

def cut_into_fragments(signal, fragment_duration, sr):
    fragment_length = fragment_duration * sr
    number_of_fragments = int(len(signal) // fragment_length)
    cut_signal_length = int(number_of_fragments * fragment_length)
    cut_signal = np.array(signal[:cut_signal_length]).reshape(number_of_fragments, -1)
    return cut_signal

def calculate_MFCCs(signal, sample_rate, n_fft, hop_length):
    mfccs = [librosa.feature.mfcc(s,
                                  sr=sample_rate,
                                  n_fft=n_fft,
                                  hop_length=hop_length,
                                  n_mfcc=13).T.tolist() for s in signal]
    return mfccs

signal, sr = librosa.load(filename, sr=44100)
cs = cut_into_fragments(signal, 0.3, 44100)
```

```
mfcc = calculate_MFCCs(cs, 44100, 2048, 512)
print(type(mfcc[0][0]), len(mfcc[0][0]))
print(chord_classes)
print(list(chord_transcriptions.values())[0])

#################### DATASET CREATION ####################
def save_json(json_path, data):
    with open(json_path, "w") as f:
        json.dump(data, f, indent=4)
    print('\nJSON saved!\n')

def create_dataset(transcripts, chord_classes, audiodir, sample_rate, window_size, n_fft,
hop_length):

    dataset = {
        "mapping": [],
        "MFCCs": [],
        "chords": [],
    }

    dataset["mapping"] = chord_classes

    count = 1
    for _file in os.scandir(audiodir):
        filename = os.path.basename(_file)
        print(f'\n {count} Processing {filename}')
        count += 1
        if _file.is_file():
            transcript = find_transcript(filename, transcripts)
            signal = load_audio(_file, sample_rate, transcript)

            for row in transcript:
                # row[2] = get_classification_vector(row[2], chord_classes)
                row[2] = get_chord_class_number(row[2], chord_classes)

            w_transcript = use_window(transcript, window_size)
            w_signal = cut_into_fragments(signal, window_size, sample_rate)
            while len(w_transcript) != len(w_signal):
                print(len(w_transcript), '!=', len(w_signal),
                    'Number of transcript windows is not equal to the number of audio
windows.')
                l = min(len(w_transcript), len(w_signal))
                w_transcript = w_transcript[:l-1]
                w_signal = w_signal[:l-1]

            mfccs = calculate_MFCCs(w_signal,sample_rate, n_fft, hop_length)

            dataset["chords"] += w_transcript
            dataset["MFCCs"] += mfccs
    return dataset

chords_dataset = create_dataset(
    transcripts=chord_transcriptions,
    chord_classes=chord_classes,
    audiodir=AUDIO_DIR,
    sample_rate=SAMPLE_RATE,
    window_size=WINDOW_SIZE,
    n_fft=N_FFT,
```

```
    hop_length=HOP_LENGTH
)

# test_trans = dict({
#     "01_-_A_Hard_Day's_Night.lab": chord_transcriptions["01_-_A_Hard_Day's_Night.lab"],
#     "01_-_Come_Together.lab": chord_transcriptions["01_-_Come_Together.lab"],
# })

# test_dataset = create_dataset(
#     transcripts=test_trans,
#     chord_classes=chord_classes,
#     audiodir=AUDIO_TEST,
#     sample_rate=SAMPLE_RATE,
#     window_size=WINDOW_SIZE,
#     n_fft=N_FFT,
#     hop_length=HOP_LENGTH
# )

save_json(json_path=JSON_PATH, data=chords_dataset)
```

cnn_chord_recognition.py

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

DATASET_PATH = 'data/data-mfcc40.json'

labels = [
            "A",
            "A#m|Bbm",
            "A#|Bb",
            "Am",
            "B",
            "Bm",
            "C",
            "C#m|Dbm",
            "C#|Db",
            "Cm",
            "D",
            "D#m|Ebm",
            "D#|Eb",
            "Dm",
            "E",
            "Em",
            "F",
            "F#m|Gbm",
            "F#|Gb",
            "Fm",
            "G",
            "G#m|Abm",
            "G#|Ab",
            "Gm",
            "N",
                    ]
```

```
def vectorize(number):
    vector = [0 for _ in range(25)]
    vector[number] = 1
    return vector

def load_data(data_path):
    """Loads training dataset from json file.
        :param data_path (str): Path to json file containing data
        :return X (ndarray): Inputs
        :return y (ndarray): Targets
    """

    with open(data_path, "r") as fp:
        data = json.load(fp)

    X = np.array(data["MFCCs"])
    y = np.array(data["chords"])
    return X, y


def plot_history(history):
    """Plots accuracy/loss for train and validation set as a function of the epochs
        :param history: Training history of model
        :return:
    """

    fig, axs = plt.subplots(2)

    # create accuracy sublpot
    axs[0].plot(history.history["accuracy"], label="train accuracy")
    axs[0].plot(history.history["val_accuracy"], label="test accuracy")
    axs[0].set_ylabel("Accuracy")
    axs[0].set_title("Accuracy eval")

    # create error sublpot
    axs[1].plot(history.history["loss"], label="train error")
    axs[1].plot(history.history["val_loss"], label="test error")
    axs[1].set_ylabel("Error")
    axs[1].set_xlabel("Epoch")
    axs[1].set_title("Error eval")

    plt.show()


def prepare_datasets(test_size, validation_size):
    """Loads data and splits it into train, validation and test sets.
    :param test_size (float): Value in [0, 1] indicating percentage of data set to allocate
to test split
    :param validation_size (float): Value in [0, 1] indicating percentage of train set to
allocate to validation split
    :return X_train (ndarray): Input training set
    :return X_validation (ndarray): Input validation set
    :return X_test (ndarray): Input test set
    :return y_train (ndarray): Target training set
    :return y_validation (ndarray): Target validation set
    :return y_test (ndarray): Target test set
    """
```

```
    # load data
    X, y = load_data(DATASET_PATH)

    # create train, validation and test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)
    X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train,
test_size=validation_size)

    # add an axis to input sets
    X_train = X_train[..., np.newaxis]
    X_validation = X_validation[..., np.newaxis]
    X_test = X_test[..., np.newaxis]

    return X_train, X_validation, X_test, y_train, y_validation, y_test


def build_model(input_shape):
    """Generates CNN model
    :param input_shape (tuple): Shape of input set
    :return model: CNN model
    """

    # build network topology
    model = keras.Sequential()

    ############### 1 ##########################
    model.add(keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape,
padding='same'))
    model.add(keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(keras.layers.MaxPooling2D((2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    ############### 2 ##########################
    model.add(keras.layers.Conv2D(64, (5, 5), activation='relu', padding='same'))
    model.add(keras.layers.Conv2D(64, (5, 5), activation='relu', padding='same'))
    model.add(keras.layers.MaxPooling2D((2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.Dropout(0.2))

    ############### 3 ##########################
    model.add(keras.layers.Conv2D(128, (7, 7), activation='relu', padding='same'))
    model.add(keras.layers.Conv2D(128, (7, 7), activation='relu', padding='same'))
    model.add(keras.layers.MaxPooling2D((2, 2), padding='same'))
    model.add(keras.layers.Dropout(0.3))

    ############### 4 ##########################
    model.add(keras.layers.Flatten())

    ############### 5 ##########################
    model.add(keras.layers.Dense(128, activation='relu'))
    model.add(keras.layers.Dropout(0.2))

    ############### 6 ##########################
    model.add(keras.layers.Dense(25, activation='softmax'))

    return model
```

```
def predict(model, X, y):
    """Predict a single sample using the trained model
    :param model: Trained classifier
    :param X: Input data
    :param y (int): Target
    """

    # add a dimension to input data for sample - model.predict() expects a 4d array in this
case
    X = X[np.newaxis, ...]

    # perform prediction
    prediction = model.predict(X)

    # get index with max value
    predicted_index = np.argmax(prediction, axis=1)
    print("Probabilities:\n", prediction)
    index_t = int(np.where(y==1)[0])
    index_p = int(predicted_index[0])
    print(f"\nTarget: {labels[index_t]}, Predicted label: {labels[index_p]}")


    pred = np.array(prediction)
    sorted_p = pred.sort()



if __name__ == "__main__":

    # get train, validation, test splits
    X_train, X_validation, X_test, y_train, y_validation, y_test = prepare_datasets(0.3,
0.2)

    mean = np.mean(X_train, axis=0)
    std = np.std(X_train, axis=0)
    X_train = (X_train - mean)/std

    mean = np.mean(X_validation, axis=0)
    std = np.std(X_validation, axis=0)
    X_validation = (X_validation - mean)/std

    mean = np.mean(X_test, axis=0)
    std = np.std(X_test, axis=0)
    X_test = (X_test - mean)/std


    n_classes = 25
    y_train = keras.utils.to_categorical(y_train, n_classes)
    print(y_train)
    y_validation = keras.utils.to_categorical(y_validation, n_classes)
    y_test = keras.utils.to_categorical(y_test, n_classes)

    # create network
    input_shape = (X_train.shape[1], X_train.shape[2], 1)
    print(input_shape)
```

```
model = build_model(input_shape)

# compile model
model.compile(optimizer="adam",
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()

# train model
history = model.fit(X_train,
                    y_train,
                    validation_data=(X_validation, y_validation),
                    batch_size=32,
                    epochs=25)

# plot accuracy/error for training and validation
plot_history(history)

# evaluate model on test set
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)

# pick a sample to predict from the test set
X_to_predict = X_test[100]
y_to_predict = y_test[100]

# predict sample
predict(model, X_to_predict, y_to_predict)
```