КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра моделювання складних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА
за спеціальністю 113 Прикладна математика
на тему:

Перетворення Фур'є у цифровій обробці сигналів

Виконав студент 4-го курсу
Володимир КУЗЬ _____

Науковий керівник:
доктор фіз.-мат. наук, доцент
Андрій ШАТИРКО _____

Робота заслухана на засіданні кафедри моделювання складних систем та рекомендована до захисту в ЕК, протокол № ..15.... від ..07.06..... 2024 р.

Завідувач кафедри МСС _____ доктор технічних наук, доцент
Дмитро ЧЕРНІЙ

Київ – 2024

Анотація

Випускна кваліфікаційна робота бакалавра: 56 сторінок, 22 рисунки, 2 додатки, 13 інформаційних джерел.

Тема: перетворення Фур’є у цифровій обробці сигналів.

Об’єкт дослідження: класифікація музичних інструментів в аудіозаписі.

Мета роботи: дослідження методів обробки та аналізу сигналів.

Предмет дослідження: реалізація програмного забезпечення для аналізу звукового сигналу та класифікації музичних інструментів в мелодії.

В даній дипломній роботі ми детально розібрали теорію перетворення Фур’є, розглянули декілька базових задач, ознайомившись з основними програмними інструментами. Реалізували задачу класифікації інструментів в аудіозаписі

Ключові слова: перетворення Фур’є, обробка даних, аналіз сигналів, алгоритм перетворення, фільтрація сигналу, класифікація інструментів.

Abstract

Bachelor's Thesis: 56 pages, 22 figures, 2 appendices, 13 sources.

Topic: Fourier Transform in Digital Signal Processing.

Research Object: Classification of musical instruments in audio recordings.

Objective: Investigation of signal processing and analysis methods.

Research Subject: Implementation of software for sound signal analysis and classification of musical instruments in melody.

In this thesis, we have thoroughly examined the theory of Fourier Transform, discussed several basic tasks, familiarized ourselves with the main software tools. Implemented the task of classifying instruments in audio recordings.

Keywords: Fourier transform, data processing, signal analysis, transformation algorithm, signal filtering, instrument classification.

Зміст

Вступ

Перетворення Фур'є є актуальним математичним інструментом для аналізу сигналів та обробки даних. Це поняття було розроблене Жозефом Фур'є у 19 столітті і відтоді було застосоване в інженерії, фізиці, обробці зображень, комп'ютерній науці. В сучасних застосунках перетворення Фур’є використовується в таких сферах: комп’ютерна графіка, звукова обробка, медична техніка, телекомунікації, криптографія, квантова механіка, машинне навчання та інші. Використання даного перетворення у великій кількості областей демонструє наскільки цей інструмент корисний.

Перетворення Фур'є переводить сигнал або функцію з часової області в частотну область, що дозволяє аналізувати сигнали за їх частотним складом. Це можна використовувати для розкладання сигналів на окремі складові, для фільтрації шуму, для визначення частот, які домінують у сигналі, та для багатьох інших застосувань.

Мета роботи: метою даної курсової роботи є огляд та дослідження основних теоретичних відомостей і практичних завдань на тему Перетворення Фур’є, постановка та реалізація задачі класифікації музичних інструментів в аудіозаписі.

Перший розділ містить теорію по основним типам перетворень, їхнім властивостям, а також додаткові відомості по даній темі. Більшу частину теоретичного матеріалу взято з джерела [5]. В другому розділі розглядається практичне застосування перетворень Фур’є та постановка задач. Описано реалізації сформульованих задач та коментарі щодо отриманих результатів. В третьому розділі розглядається задача класифікації музичних інструментів в аудіозаписі, необхідні програмні засоби для цього та теоретичні відомості. У додатку знаходиться програмний код реалізацій.

В цій дипломній роботі не використано праць авторів без відповідних посилань.

РОЗДІЛ 1. Основні теоретичні відомості

1.1 Пряме перетворення Фур'є

Відомо, що пряме перетворення Фур'є зручний інструмент для опису аналогових та дискретних сигналів у частотній області. Його також називають спектром $X_a(iw)$ аналогового сигналу $x_a(t)$:

$$X_a(iw) = \int_0^\infty x_a(t)e^{-i\omega t}\,dt.$$

Дискретне перетворення Фур'є використовується, щоб перетворити сигнал із часової області в частотну і назад. Прямим дискретним перетворенням Фур'є називається перетворення послідовності $x(n)\quad n = 0, \dots, N-1$ у послідовність $X(k), k = 0, \dots, N-1$ за наступною формулою:

$$X(k) = \sum_{n=0}^{N-1} x(nT)e^{-2\pi ink/N} = \sum_{n=0}^{N-1} x(nT)W^{-nk}, \text{ де } k = 0, \dots, N-1.$$

• N – кількість компонент розкладу, число виміряних за період значень сигналу;

• n - номер відліку дискретизованого сигналу, $n = 0, \dots, N-1$;

• k - номер гармоніки компонента перетворення, а T - період часу, протягом якого бралися вхідні дані;

• $W = e^{-2\pi/N}$ - поворотний множник.

У цій формулі $X(kT) = X\left(e^{iwt}\right)$ є спектральною щільністю (спектром) дискретної послідовності.

Вираз для спектра дискретної послідовності можна знайти, замінивши в її $Z-$ формі змінну $z = e^{iwt}$.

Використовуючи формулу Ейлера

$$e^{iwt} = \cos(wT) + isin(wT),$$

можна визначити дійсну та уявну складові, а також модуль та аргумент спектральної щільності, які пов'язані з дійсною та уявною частинами спектру через формули теорії функції комплексної змінної.

Модуль: $|X(kT)| = \sqrt{Re(X)^2 + Im(X)^2}$.

Фаза (аргумент): $arg(X(kT)) = arc\tan(Im(X)/Re(X))$.

Таким чином, ДПФ для N вхідних відліків сигналу ставить у відповідність N спектральних відліків. Щоб обчислити один спектральний відлік потрібно виконати N операцій комплексного множення та додавання. Оскільки таких операцій N, то загальна обчислювальна складність ДПФ дорівнює N^2.

1.2 Зворотне (обернене) перетворення Фур'є

Зворотне перетворення (ОДПФ) є трансфер послідовності $X(k)$, $k = 0, \dots, N-1$ в послідовність $x(n)$, $n = 0, \dots, N-1$ за формулою:

$$X(nT) = \frac{1}{N}\sum_{n=0}^{N-1} X(k)e^{-2\pi ink/N} = \frac{1}{N}\sum_{n=0}^{N-1} X(k)W^{nk}, \text{ де}$$

$x(n)$ - виміряна послідовність у дискретних часових точках, значення якої є вхідними даними для прямого перетворення і вихідними для зворотного (оберненого) $X(k)$;

$N-$послідовність комплексних амплітуд синусоїдальних сигналів, що утворюють вхідний сигнал $x(n)$, значення послідовності є вихідними даними для прямого перетворення та вхідними для зворотного.

Оскільки амплітуди спектральних відліків - комплексні величини, то з них можна обчислити одночасно і амплітуду, і фазу (аргумент) сигналу.

Як випливає з теореми Найквіста-Котельникова, ДПФ точно відповідає неперервному перетворенню Фур'є, якщо функція, що перетворюється, має обмежений спектр, при цьому частота дискретизації $F_Д$ повинна бути не менше подвоєної максимальної частоти спектра F_B. Слід зазначити, що для ДПФ справедливі правила та властивості, що були розглянуті для Z-перетворення. Матрицею $k * n$ елементів можна визначити ДПФ.

1.3 Властивості дискретного перетворення Фур'є

Про всі властивості дискретного перетворення Фур’є дуже детально описано в джерелах [1], [5] та [6].

1. **Лінійність:** сума спектрів сигналів дорівнює спектру суми сигналів.

Нехай послідовність $x_3(n) = a_1x_1(n) + a_2x_2(n)$ є лінійною комбінацією послідовностей $x_1(n)$ і $x_2(n)$; величини a_1 та a_2 є константами. Тоді для спектрів цих послідовностей можна записати

$X_3(e^{i\omega T}) = a_1X_1(e^{i\omega T}) + a_2X_2(e^{i\omega T})$, що вказує на властивість лінійності перетворень Фур'є.

Ця властивість говорить про те, що спектр суми незалежних дискретних сигналів дорівнює сумі спектрів цих сигналів, а при множенні дискретного сигналу на константу його спектр також множиться на цю константу.

2. **Періодичність:** Спектр $X(e^{i\omega T})$ дискретної послідовності є періодичною комплексною функцією по частоті ω з періодом, рівним частоті дискретизації $\omega_\sigma = 2\pi/T$. Тобто виконується співвідношення

$$X\left(e^{i\omega T}\right) = X\left(e^{i(\omega+k\omega_\sigma)T}\right),$$

де k – ціле число.

Очевидно, що амплітудні складові спектра також є періодичними по частоті(модуль спектра $X(e^{i\omega T})$) і фазові складові(аргумент $arg\{X\left(e^{i\omega T}\right)\}$). Крім того, для дійсних послідовностей $x(\mathrm{n})$, випливають справедливі співвідношення

$$|X\left(e^{i\omega T}\right)| = |X\left(e^{-i\omega T}\right)|,$$

$$arg\{X\left(e^{i\omega T}\right)\} = -arg\{X\left(e^{i\omega T}\right)\}.$$

3. **Зсув за відліками (за часом):**

Циклічний зсув сигналу на m відрахунків призводить до повороту фазового спектру, а амплітудний спектр при цьому не змінюється.

$$X'(k) = X(k)e^{-\frac{2\pi i}{N}km}.$$

4. **ДПФ від парних та непарних функцій:**

ДПФ парної функції вироджується у косинусне перетворення Фур'є

$$X(k) = \sum_{n=0}^{N-1} x(n)\cos(2\pi nk/N).$$

ДПФ непарної функції вироджується у синусне перетворення Фур'є

$$X(k) = \sum_{n=0}^{N-1} x(n)\sin(2\pi nk/N),$$

де $k = 0, \dots, N-1$.

5. **ДПФ циклічної згортки сигналів:**

Для сигналу $x(n)$, який є результатом циклічної згортки двох сигналів $a(n)$ та $b(n)$:

$$X(n) = 1/N \sum_{m=0}^{N-1} a(n)\mathrm{b}(n-m).$$

N-точкове ДПФ послідовності дорівнює: $X(k) = A(k) \bullet B(k)$,

де $A(k)$, $B(k)$– спектри сигналів.

Таким чином, спектр циклічної згортки двох сигналів дорівнює добутку спектрів цих сигналів. Ця властивість дозволяє використовувати швидкі алгоритми ДПФ для обчислення згортки.

6. **ДПФ добутку сигналів:**

Для сигналу $x(n)$, який є результатом добутку двох сигналів a(n) та b(n) спектр дорівнює: $X(k) = 1/N \sum_{m=0}^{N-1} A(m)\mathrm{B}(k-m)$.

7. **Зсув за частотою:**

Аналогічно другій властивості (часовий зсув), якщо є спектр $x(k-m)$ зсунутий по частоті на m, то після ОДПФ послідовність $x(n)$ набуває наступного вигляду:

$$x'(n) = x(n)e^{-2\pi i/N}km.$$

Звідси випливає, що зсув спектра здійснюється множенням сигналу на комплексну експоненту. Ця властивість використовується для перенесення

частот по діапазону. Зауважимо, що після множення на експоненту сигнал буде комплексним, а його спектр перестане бути симетричним.

8. **Теорема Парсеваля:**

Середня потужність дискретизованої функції часу дорівнює сумі потужностей окремих спектральних складових і залежить від їх фаз.

Спектр ДПФ зберігає інформацію про енергію вихідного сигналу, нормована енергія сигналу $x^2(n)$ дорівнює:

$$\sum_{n=0}^{N-1} x^2(n) = \frac{\sum_{n=0}^{N-1}|X^2(n)|}{N}.$$

Як бачимо, властивості ДПФ є двоїстими, тобто, властивості ДПФ справедливі як для сигналу, так й для спектру.

1.4 Поворотні множники

Коефіцієнти матриці ДПФ (twiddle factor) або поворотні множники W_{nk} можна знайти за наступною формулою:

$$W_{nk} = e^{\frac{-2\pi ink}{N}}.$$

Таким чином, матриця ДПФ без урахування нормуючого множника влаштована так:

перші рядок і стовпець складаються з одиниць, у другому рядку стоять корені з одиниці порядку n в природному порядку, наступні рядки є послідовними степенями другого рядка.

1.5 Перехід від ДПФ до ШПФ (швидке перетворення Фур'є)

Перетворення Фур'є лежить в основі методів згортки і проектування цифрових кореляторів. Однак до появи комп'ютерів ДПФ використовувалося рідко, оскільки обчислення ДПФ навіть для 64 відліків вимагає 4096 операцій комплексного множення і практично стільки ж операцій додавання, що вручну вважається досить довгим і трудомістким. Для N =1024 потрібно близько

мільйона операцій комплексного множення та мільйон операцій комплексного додавання. Чим більше точок обчислення (що більше довжина ДПФ), то більше часу витрачається на обчислення у зв'язку зі збільшенням кількості операцій. Обчислення перетворення Фур'є за стандартною формулою передбачає виконання великої кількості операцій додавання та множення. Очевидно, що виникає необхідність розробити алгоритми, які зменшують кількість математичних процесів при розрахунку ДПФ.

Слід зазначити, що обчислювати ДПФ безпосередньо не обов'язково і можна обійтися значно меншим числом операцій. Розглянемо основну ідею ШПФ, яка полягає у розбитті вихідної N -мірної послідовності $x(n), n = 0, \dots, N-1$ на частини. При цьому для кожної частини можна обчислити ДПФ окремо, а потім просумувати лінійно з іншими, щоб отримати вихідне перетворення. У свою чергу, ці частини меншого розміру можна розбити на ще менші частини, і зробити ті самі операції.

Нехай довжина періодичної послідовності дорівнює N , тоді для обчислення одного спектрального відліку буде потрібно N операцій комплексного множення та додавання. Таким чином, загальна обчислювальна складність алгоритму ДПФ складе N^2 множень та додавань. Якщо розділити вихідну послідовність на дві рівні частини $\frac{N}{2}$ елементів, то для виконання обчислення перетворення за класичною формулою на кожному етапі потрібно в два рази менше операцій додавання та множення.

При цьому кожне з $N/2$-точкових ДПФ можна обчислити шляхом заміни $N/2$ -точкового ДПФ на два $N/4$ -точкових. І тут кількість операцій комплексного додавання і множення зменшується ще вдвічі.

Сутність даного алгоритму ДПФ полягає в тому, що розбиття вихідної послідовності можна продовжувати, поки можливе ціле ділення послідовності на двійку. Зрозуміло, якщо довжина вхідної послідовності $N = 2m$, де m - додатне ціле число, то вихідну послідовність можна розділити навпіл всього m

разів. Алгоритми ШПФ, з довжиною послідовності $N = 2m$, називаються алгоритми ШПФ за основою 2 (Radix-2 FFT).

Ефективність алгоритму ШПФ повністю залежить від способу розбиття та поєднання послідовності. Послідовності на дві частини можна ділити різними методами, від цього залежить, чи зможемо при об'єднанні отримати неспотворений спектр сигналу, і які обчислювальні ресурси доведеться витратити. Кількість операцій ШПФ лінійно залежить від довжини послідовності N.

1.6 Порівняння ефективності

Нижче наведено таблицю, що показує порівняння ефективності алгоритмів ШПФ у порівнянні з ДПФ.

Ефективність алгоритму ШПФ та кількість операцій, що виконуються, лінійно залежить від довжини послідовності N.

З таблиці видно, що використання ШПФ істотно економить кількість операцій, причому, чим більше довжина послідовності N, тим більше стає економія.

Наприклад, для $N = 8192$ відліків при обчисленні ДПФ потрібно 67 мільйонів операцій комплексного додавання та множення! Використовуючи алгоритми ШПФ, можна знизити ці числа у 1260 та 630 разів відповідно!

1.7 Алгоритми ШПФ

Існує два основних методи обчислення ШПФ за основою 2 (Radix-2): з проріджуванням (або децимацією) за частотою та за часом. Розглянемо обидва варіанти.

Розглянемо перший спосіб розбиття послідовності - проріджування за часом, який називають алгоритмом ШПФ з «децимацією за часом» FFT Decimation-in-time [DIT]. Ідея полягає в тому, що вхідна послідовність відліків

$x(n)$ із довжиною N розбивається на дві послідовності $x_0(n)$ та $x_1(n)$ рівної довжини $N/2$.

Причому x0(n) - послідовність парних відліків - $x_0(n) = x(2n), n = 0, \dots, \frac{N}{2} - 1$,

а $x_1(n)$ - послідовність непарних відліків - $x_0(n) = x(2n+1), n = 0, \dots, \frac{N}{2} - 1$.

Оминаючи математичні викладки (їх можна знайти в будь-якій літературі з цифрової обробки сигналів), запишемо основні правила обчислення ШПФ шляхом розбиття послідовності на парні та непарні.

Алгоритм ШПФ з децимацією за часом:

• здійснити двійково-інверсну перестановку відліків вхідного сигналу, забезпечивши розбиття початкової вхідної послідовності;

• зробити $N/2$ операцій «Метелик» для отримання першого об'єднання, використовуючи поворотні коефіцієнти;

• повторити операцію «Метелик» для переходу на наступні етапи, використовуючи також поворотні коефіцієнти.

Після всіх вищеописаних дій отримаємо на виході ДПФ початкової вхідної послідовності.

«Метелик» - спрямований граф, за допомогою якого обчислюється пара комплексних відліків за попередніми значеннями.

Для ШПФ з проріджуванням за часом метелик за основою 2 записується за формулою: $X = A + BW_N^{-k}$; $Y = A - BW_N^{-k}$.

В алгоритмі ШПФ з децимацією за часом проводився поділ вхідного сигналу відповідно до двійково-інверсної перестановки - на парні та непарні частини. Тим самим отримаємо першу та другу половину спектру. В алгоритмі з проріджуванням за частотою навпаки, вхідний сигнал ділиться навпіл, а на виході отримують дві послідовності спектральних відліків – парну та непарну (тому алгоритм і називається «проріджування за частотою»).

Послідовність відліків $x(n)$ із довжиною N розбивається на дві послідовності $x_0(n)$ і $x_1(n)$ рівної довжини $N/2$.

Причому x0(n) - послідовність першої половини даних – $x_0(n) = x(n)$, n=0, ..., $\frac{N}{2} - 1$, а $x_1(n)$ - послідовність другої половини даних – $x_1(n) = x(n)$, $n = \frac{N}{2}, \dots, N - 1$.

Принципова різниця алгоритмів у тому, що при проріджуванні за часом множення на поворотні коефіцієнти проводилося після ДПФ парної та непарної послідовності, а при використанні децимації частоти, множення виконується до ДПФ. При цьому обчислювальна ефективність та швидкість обох алгоритмів ідентична.

Алгоритм ШПФ з децимацією за частотою:

• зробити $N/2$ операцій «Метелик» для отримання першого об'єднання, використовуючи поворотні
коефіцієнти;

• повторити операцію «Метелик» для переходу на наступні етапи, використовуючи також поворотні
коефіцієнти.

• здійснити двійково-інверсну перестановку результуючого сигналу;

Метелик у цьому випадку виглядає дещо інакше:

$$X = A + B,$$
$$Y = (A - B)W_N^{-k}.$$

1.8 Згортка сигналів

На базі ШПФ можна обчислити згортку довгих послідовностей. Цей метод застосовується в умовах обмеженості обчислювальних ресурсів (наприклад, пристрої програмованих логічних інтегральних схем - ПЛІС). У наступних розділах буде детально розглянуто задача згортки послідовностей.

1.9 Ефект Гіббса

Пульсації в області різких перепадів сигналу пов'язані з ефектом Гіббса. Цей ефект пов'язаний із послідовним наближенням до початкової форми у вигляді часткових сум ряда Фур'є. Іншими словами, природа пульсацій у відновлюваному сигналі безпосередньо пов'язана з ефектом Гіббса. Ефект Гіббса яскраво виражений за різких порушень монотонності функції. На різких перепадах та стрибках цей ефект максимальний.

На рисунку нижче представлений графік відновлення лінійного сигналу за допомогою часткової суми ряду Фур'є за різних значень кількості сумарних відліків

РОЗДІЛ 2. Базові задачі на застосування перетворення Фур’є

Перетворення Фур'є використовується для вирішення різних задач в аналізі сигналів та обробці даних. Ось декілька задач, для яких може використовуватися перетворення Фур'є:

Аналіз спектру сигналу: Перетворення Фур'є дозволяє виділити окремі складові сигналу за їх частотою, що дозволяє зробити аналіз сигналу більш детальним та точним.

Фільтрація сигналу: Застосування фільтра до сигналу в частотній області може допомогти зменшити шум та інші не потрібні компоненти сигналу.

Компресія даних: Перетворення Фур'є може використовуватися для зменшення обсягу даних, зберігаючи при цьому інформацію про частотний склад сигналу.

Розв'язування диференціальних рівнянь: Перетворення Фур'є може бути використане для розв'язування диференціальних рівнянь в часткових похідних шляхом переведення диференціального рівняння в алгебраїчне.

Аналіз зображень: Перетворення Фур'є може бути використане для аналізу зображень та для знаходження частот, що домінують у зображенні.

Аналіз аудіосигналів: Перетворення Фур'є може допомогти знайти звукові елементи у записі аудіо та для видалення шуму зі звукового сигналу.

Шифрування даних: Перетворення Фур'є може бути використане для шифрування даних, що дозволяє зберігати конфіденційну інформацію.

Ці задачі є лише декількома з безлічі застосувань перетворення Фур'є.

Постановка задач.

2.1 Розглянемо стандартну задачу на використання перетворення Фур'є.

Нехай є сигнал, який представляє собою суму двох синусоїд з різними частотами. Розкладемо цей сигнал на складові частоти та визначимо амплітуду і фазу кожної з них.

План виконання:

1) Знайти амплітуду і частоту синусоїд.

2) Визначити спектр частот, застосувавши перетворення Фур'є до сигналу.

3) Проаналізувати спектр частот.

4) Визначити амплітуду і фазу складових.

5) Застосувати обернене перетворення Фур'є й отримати вихідний сигнал у вигляді суми складових частот.

2.2 Знайдемо перетворення Фур'є від сигналу:
$f(t) = \cos(2\pi f_0 t) + \cos(2\pi(f_0 + \Delta f)t) + \cos(2\pi(f_0 + 2\Delta f)t)$, де $f_0 = 100$ Гц і $\Delta f = 10$ Гц. Обчислимо спектральну щільність потужності цього сигналу та візуалізуємо її.

2.3 Використовуючи перетворення Фур'є створимо фільтр шумного сигналу.

Згенеруємо синусоїдальний сигнал: $f(t) = \sin(2\pi 50t) + \sin(2\pi 120t)$. Додамо до сигналу випадковий шум. Знайдемо перетворення Фур'є від шумного сигналу та визначимо вісь частот. Побудуємо графік амплітудного спектру сигналу. Налаштуємо фільтр, який пропускатиме лише частоти в певному діапазоні. Отримаємо відфільтрований сигнал і візуалізуємо графік сигналів.

Перетворення Фур’є можна використовувати великою кількістю засобів, різні середовища та мови програмування мають багато бібліотек та функцій для використання.

Для виконання поставлених задач, я буду використовувати мову програмування Python, середовище Jupyter Notebook та потрібні для реалізації задач бібліотеки цієї мови.

Мова Python є одною з найпопулярніших мов у світі, особливо серед дослідників та інженерів, які працюють з аналізом даних та обробкою сигналів, включаючи використання перетворення Фур’є. Ця мова дуже зручна у використанні, бо має легко зрозумілий синтаксис, широкі можливості візуалізації даних, велику кількість наукових бібліотек у відкритому доступі, такі як NumPy, SciPy, Matplotlib, Pandas та інші, які дозволяють просто і ефективно реалізувати поставлені завдання. Тому Python є дуже потужним інструментом для виконання операцій обробки сигналів.

Jupyter Notebook – інтерактивне середовище програмування, дозволяє виконувати код та зберігати його у зручному вигляді. Підтримує Python, R, Scala та інші. В даному середовищі можна запускати код частинами, що дає можливість кожного кроку перевіряти і налагоджувати роботу програми. Можна створювати графіки, діаграми та інші елементи візуалізації даних безпосередньо у середовищі, саме тому це дуже зручний засіб для роботи.

Далі продемонстровані результати реалізацій задач 2.1-2.3 у пунктах 3.1-3.3 відповідно, а програмний код знаходиться у додатку.

Результати реалізацій задач.

2.1

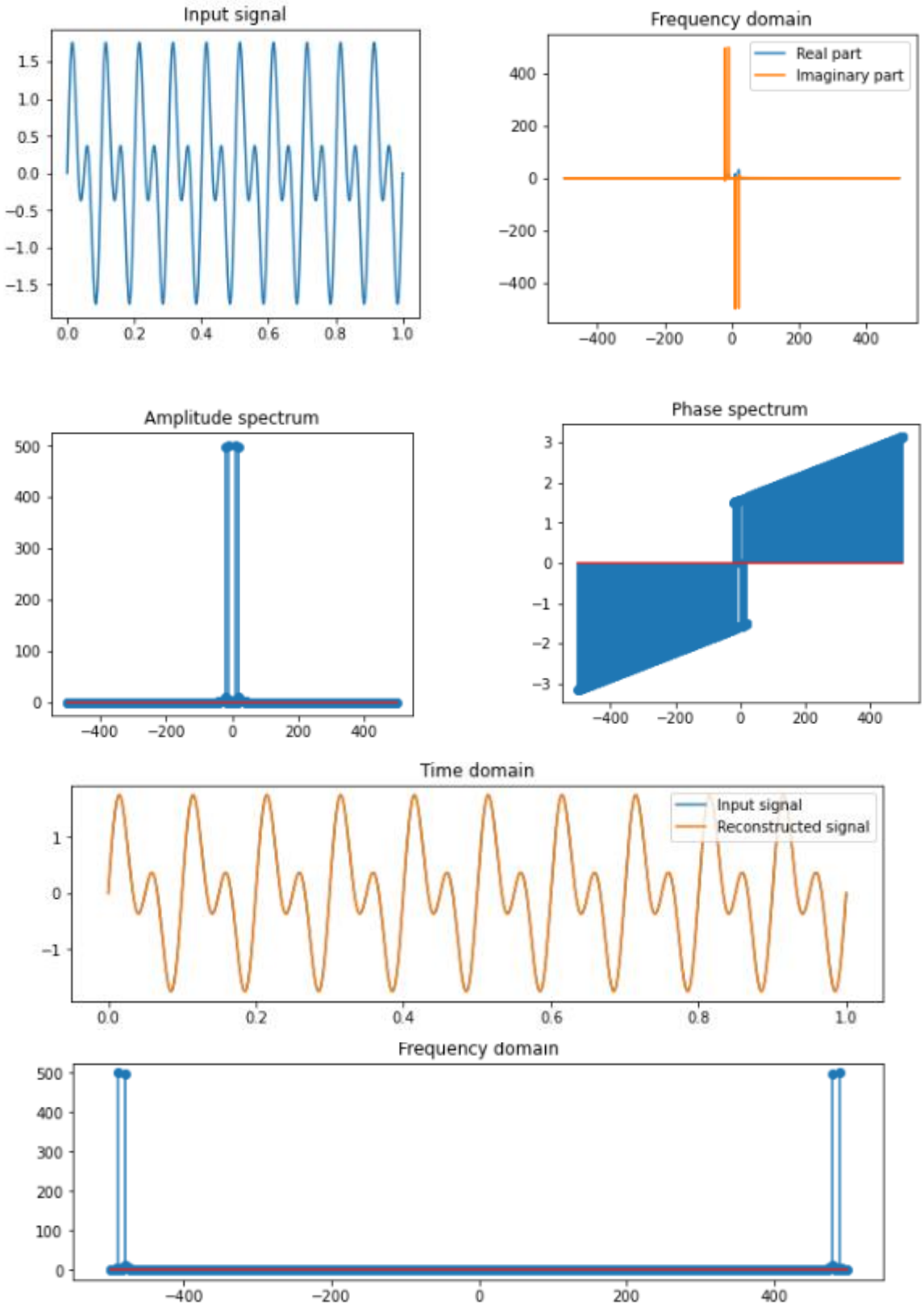

Рис. 2.1-2.6 Графіки вхідного сигналу, амплітуди, частоти і фази.

2.2

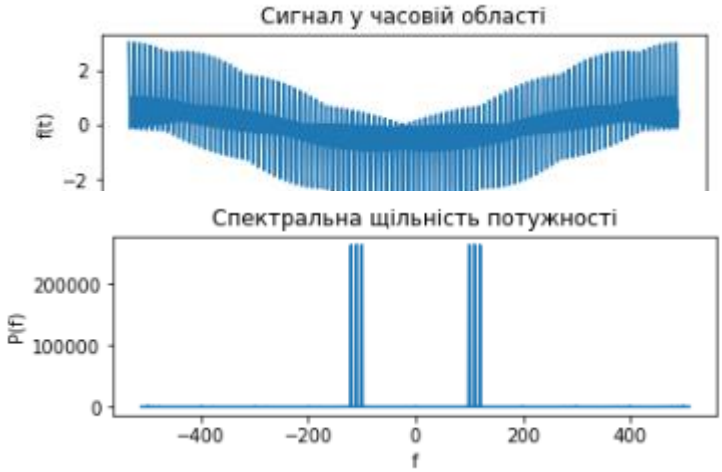

Рис. 2.7-2.8 Розв’язок другої задачі.

Задачі 2.1 і 2.2 є базовими задачами на перетворення Фур’є. В цих завданнях потрібно було знайти амплітуду, частоту, фазу, спектр частот, спектральну щільність потужності сигналу. Ці характеристики легко знайти і відобразити завдяки бібліотекам NumPy і Matplotlib.

2.3

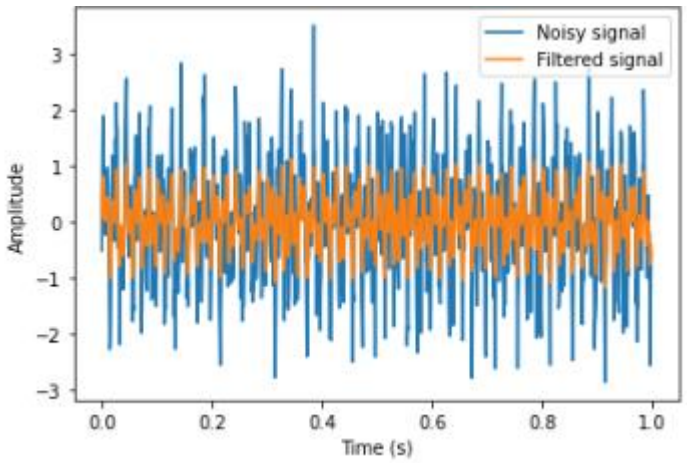

Рис. 2.9 Графік сигналу з шумом та після застосування фільтру.

Фільтр пропускає сигнали в діапазоні від 40 до 60 Гц та в діапазоні від 110 до 130 Гц, а решта частот мають рівень 0. Далі обчисливши зворотне перетворення Фур’є отримали відфільтрований сигнал. На графіку різними кольорами зображено шумний сигнал та відфільтрований.

Було розглянуто основні теоретичні відомості, поняття та методи, пов'язані з цим математичним інструментом. **Проаналізовано** як дискретні, так і неперервні версії перетворення Фур'є. Також були розглянуті їхні використання в аналізі сигналів та застосування для оптимізації обчислень. **Досліджено** їх основні властивості, такі як лінійність, зворотність та симетричність. **Виявлено**, що перетворення Фур'є являється дійсно потужним засобом для аналізу та обробки сигналів і даних, що має широкі застосування в різних галузях, таких як телекомунікації, обробка зображень, медична діагностика та багато інших.

В другому розділі розглянули приклади завдань, які можна виконати за допомогою перетворення Фур’є. **Побудовано** моделі двох стандартних задач, в яких потрібно було знайти базові речі в області обробки сигналів, а також зробити простий фільтр за допомогою перетворення Фур’є в третій задачі. **Написано програми,** які здійснюють виконання сформульованих задач. Далі в третьому розділі відображено роботу прогам. **Результати моделювання продемонстрували здатність** інструментів з бібліотек мови Python дуже легко розв’язати поставлені задачі і здатність Jupyter Notebook зручно візуалізовувати отримані дані, код програм знаходиться в додатку. Вигляд коду показує наскільки просто і зрозуміло можна виконувати подібні завдання за допомогою засобів цієї мови програмування.

Саме тому **встановлено**, що перетворення Фур’є досі залишається корисним і дієвим інструментом для обробки сигналів, а сучасні бібліотеки значно спрощують його використання.

РОЗДІЛ 3

Постановка завдання:

Ми маємо аудіозапис мелодії, і потрібно автоматично визначити, які інструменти грають у цьому записі та їхню кількість. Для цього ми плануємо використати методи цифрової обробки сигналів, зокрема перетворення Фур'є, та інші програмні інструменти.

Передумови аналізу звукових сигналів

Музика завжди була важливою складовою нашого життя. Вона сповнена емоцій, надихає та сприймається різноманітними способами. Наш мозок виконує величезну кількість складних процесів, щоб упорядкувати дані слухового сенсорного введення в інформативні структури. Наприклад, підсвідомо, відділяючи машину, що проїжджає зліва ззаду, від соло електрогітари та ритму барабанів у ваших навушниках. У повсякденному контексті слухання музики людський розум декодує вхідний аудіопотік на елементарні будівельні блоки, пов'язані з різними акустичними та музичними аспектами.

Не зважаючи на те, якою важливою дисципліною є обробка звуків музики, вона все ще є відносно молодою порівняно з обробкою мовлення. Тому виникла потреба в розвитку інноваційних методів обробки звуку для розуміння та аналізу музичних композицій. Хтось може стверджувати про простоту проблеми, оскільки кожна людина здатна розрізнити скрипку від фортепіано. Однак завдання набагато складніше, оскільки воно включає фізичні властивості музичних інструментів, правила, що накладаються музичною системою на композицію, а також перцептивну та когнітивну обробку отриманих звуків. Моделювання проблеми загальним і цілісним способом все ще створює багато труднощів для штучних систем. Крім того, навіть люди демонструють обмеження у своїх здібностях розрізняти музичні інструменти, потребують таланту й часу, щоб розвинути музичний слух.

По суті завданням являється автоматичне розпізнавання музичних інструментів із аудіосигналів. Метою є ідентифікація інструментів, що складають музичний твір, за умови невідомого твору. Це важливе завдання, оскільки воно відкриває нові можливості для аналізу музики, розуміння її структури та автоматизації процесів створення та обробки аудіозаписів. Наявність цієї інформації може полегшити операції індексації та пошуку для керування музичними колекціями, а також може бути корисною для багатьох інших завдань, навіть в інших сферах застосування.

Більшість досліджень з автоматичного розпізнавання музичних інструментів до цього часу проводилися на окремих нотах або на уривках соло-виконань. Останніми роками зростає інтерес до досліджень складнішої та реалістичнішої задачі з розділенням звуків поліфонічного аудіозапису з багатьма інструментами. У поліфонічних сумішах, що складаються з декількох інструментів, взаємне перекриття звуків, що лунають одночасно, ймовірно, обмежить ефективність розпізнавання. Проте взаємне перекриття можна зменшити, попередньо розділивши суміш на сигнали, що складаються з окремих джерел звуку. Багато алгоритмів розділення джерел звуку мають на меті відокремити найпомітніший гармонійний звук від суміші. Зазвичай вони спочатку відстежують висоту тональності цільового звуку, а потім використовують гармонійну структуру та синусоїдальне моделювання під час розділення.

У цій роботі ми зосереджуємося на застосуванні перетворення Фур'є для розпізнавання інструментів у музичних композиціях. Ця техніка знайшла широке застосування у різних галузях, зокрема у аудіоаналізі та обробці сигналів у музичній індустрії. Перетворення Фур'є дозволяє нам розглядати сигнали у частотному представленні, що дозволяє отримати важливу інформацію про їх складові. Застосування цього перетворення до аудіозаписів надає можливість виявлення характерних особливостей звуків різних музичних інструментів. Наприклад, відмінності у частотному спектрі можуть допомогти в ідентифікації звучання скрипки від звучання флейти або інших інструментів.

Можна використовувати методи класифікації, такі як SVM або нейронні мережі, для автоматичного розпізнавання інструментів на основі ознак звуку.

Первинна обробка аудіозапису

Перш за все, потрібно проаналізувати, який аудіофайл ми завантажуємо. Звук представлений у формі аудіосигналу має такі параметри, як частота, смуга пропускання, децибел тощо. Типовий аудіосигнал можна виразити як функцію амплітуди та часу.

Прикладами таких форматів аудіофайлів є:

wav (Waveform Audio File)

mp3 (MPEG-1 Audio Layer 3)

WMA (Windows Media Audio)

Аудіофрагменти у форматі .wav можна використовувати майже в будь-якому редакторі. Звукові хвилі відцифровуються методом вибірки з дискретних інтервалів, відомих як частота дискретизації (як правило, 44,1 кГц для аудіо з CD-якість, тобто 44 100 семплів в секунду). Це поширений формат аудіофайлів, який зберігає звук без стиснення, в його оригінальній формі, без втрати якості.

Кожен семпл представляє собою амплітуду хвилі в часовому інтервалі. Де динамічний діапазон сигналу показує, наскільки деталізованим буде семпл. Глибина в бітах визначає, скільки інформації може бути закодовано в кожному семплі. Більша глибина призводить до більш детального представлення звукової хвилі, але також до збільшення розміру файлу.

Семплінг - це перетворення безперервного сигналу в набір дискретних значень. Кількість семплів за певний фіксований проміжок часу називається частотою дискретизації. Збільшення кількості семплів призводить до меншої втрати інформації, але відповідно робить обчислення більш складним.

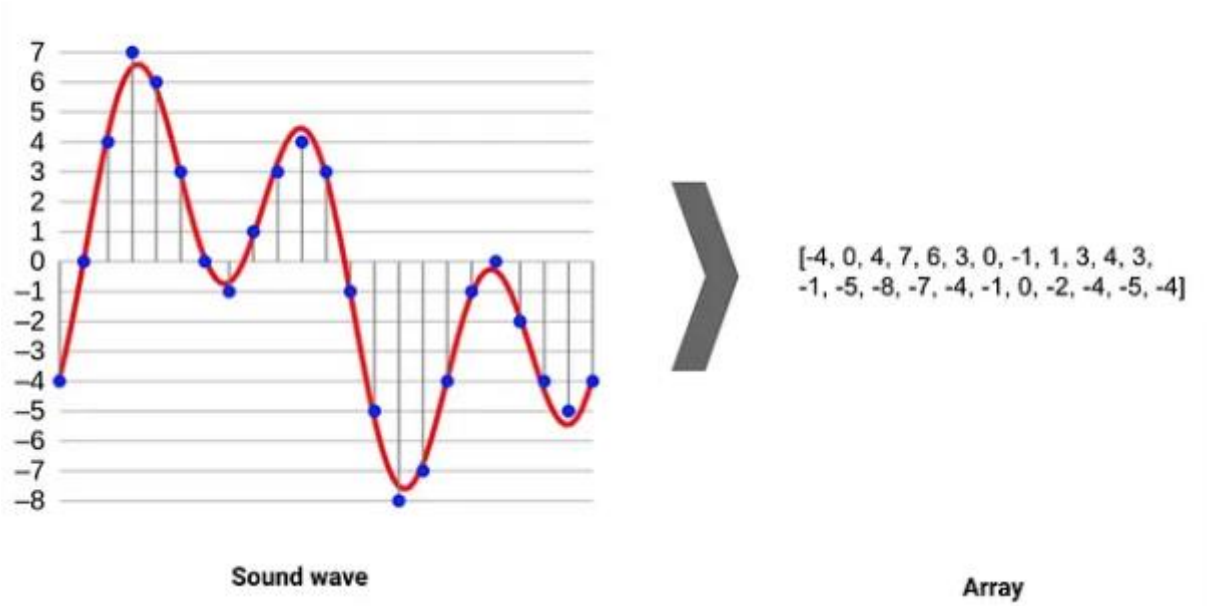

Рис. 3.1 Результат семплінгу

Тож потрібно враховувати, з яким саме форматом аудіофайлу ми будемо працювати, бо від цього можуть залежати наші подальші дії.

Процес обробки звуку включає вилучення акустичних характеристик, що стосуються поставленого завдання, за якими йдуть схеми прийняття рішень, які включають виявлення, класифікацію та об'єднання знань.

Обчислення перетворення Фур'є - допоможе перевести сигнал у частотний домен.

Створення спектрограми - покаже інтенсивність звуку в залежності від часу та частоти. Спектрограма є візуальним способом представлення рівня або "гучності" сигналу в часі на різних частотах, які присутні у формі хвилі. Зазвичай зображується у вигляді теплової карти.

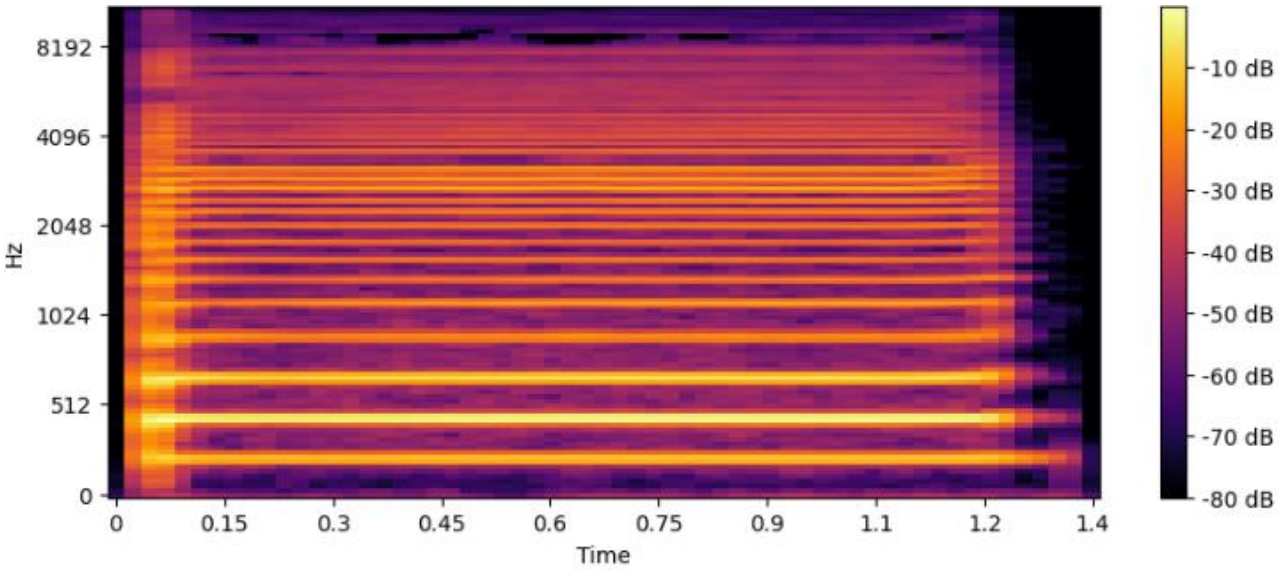

Рис. 3.2 Приклад спектрограми

На спектрограмі горизонтальна вісь представляє час, а вертикальна вісь - частоту. Колір пікселя на спектрограмі відповідає рівню сигналу на цій частоті в цей час. Чим темніший колір, тим вищий рівень сигналу.

Кожен аудіосигнал складається з великої кількості ознак. Нам потрібно буде визначити ті характеристики, які відносяться до поставленого нам завдання і використати алгоритми класифікації - задля навчання моделі на основі ознак звуку для розпізнавання інструментів(використання бібліотеки scikit-learn).

Можливі складнощі:

Низька якість аудіозапису може ускладнити визначення інструментів.

Перекриття звуків: якщо декілька інструментів грають одночасно, вони можуть перекривати один одного, що ускладнює розпізнавання. Тому варто оцінити точність класифікації та зробити висновки про результати.

Це лише загальний опис плану дій. Потрібно детально налаштувати та оптимізувати кожен етап для досягнення кращих результатів. Також слід враховувати, що для успішної класифікації може знадобитися великий обсяг даних для тренування моделі.

Визначення спектральних ознак

Щоб реалізувати автоматичне розпізнавання інструментів в аудіозаписі і виконати поставлене завдання, потрібно отримати всі необхідні характеристики звуку для роботи з нейронними мережами. Навчити модель на основі певних визначених характеристик, з використанням бібліотек мови програмування Python.

Для виконання поставленої задачі нам знадобляться такі бібліотеки:

1. Librosa

Це потужна бібліотека Python для аналізу музики та аудіо, яка допомагає працювати з різними форматами аудіофайлів. Бібліотека має відкритий вихідний код і знаходиться у вільному доступі за ліцензією ISC.

2. Pandas

Основна бібліотека в Python для роботи з даними. Робота з будь-якими даними потребує аналізу та підготовки: необхідно видалити або заповнити пропуски, відфільтрувати, відсортувати або якимось чином змінити дані. Pandas дозволяє швидко виконати всі ці дії, а в більшості випадків ще й автоматизувати їх. Вона допомагає підготувати та провести первинний аналіз даних, щоб потім використовувати їх у машинному або глибокому навчанні. Бібліотека підтримує основні статистичні методи, які необхідні для роботи з даними. Наприклад, розрахунок середніх значень, їх розподіл за квантилями та інші.

3. Numpy

Це бібліотека мови програмування Python, яку застосовують майже для будь-яких математичних обчислень. Ця бібліотека використовується на всіх етапах роботи з даними: витягування і перетворення, аналіз, моделювання і оцінка, репрезентація.

4. Scikit-learn

Це один з найбільш популярних пакетів Python для машинногонавчання. Він містить функції та алгоритми класифікації, прогнозування та кластеризації даних.

5. Keras

Це бібліотека призначена для глибокого навчання машин. Вона дозволяє швидше створювати та налаштовувати моделі. Проте Keras використовується як додаток до інших бібліотек, оскільки сама нездатна виконувати складних обчислень.

6. Matplotlib

Бібліотека для візуалізації та побудови графіків будь-якого виду: лінійні, кругові діаграми, спектрограми, гістограми й інші. Підтримує двовимірні та тривимірні графіки.

А також деякі менш важливі бібліотеки для зручного завантаження файлів, візуалізації даних та взаємодії з ними.

Для аналізу даних та машинного навчання зазвичай використовуються спеціальні Google Colab або Jupyter Notebook. Це спеціалізовані інтегровані середовища розробки, що дозволяють працювати з даними крок за кроком і ітеративно, без необхідності створювати повноцінний додаток. В них дуже зручно реалізовувати навчання моделі за ознаками, тож ці середовища будуть зручними у використанні для нашого завдання.

Далі розберемося з основними аспектами, що потрібні для реалізації поставленого завдання.

Звісно спершу потрібно завантажити аудіофайл:

```
import librosa
audio_data = '/../audio.wav'
x , sr = librosa.load(audio_data)
print(type(x), type(sr))
#<class 'numpy.ndarray'> <class 'int'>
print(x.shape, sr)
#(94316,) 22050
```

Використовуючи librosa.display.waveplot можна побудувати графік масиву аудіо:

```
%matplotlib inline
import matplotlib.pyplot as plt
import librosa.display
plt.figure(figsize=(14, 5))
librosa.display.waveplot(x, sr=sr)
```

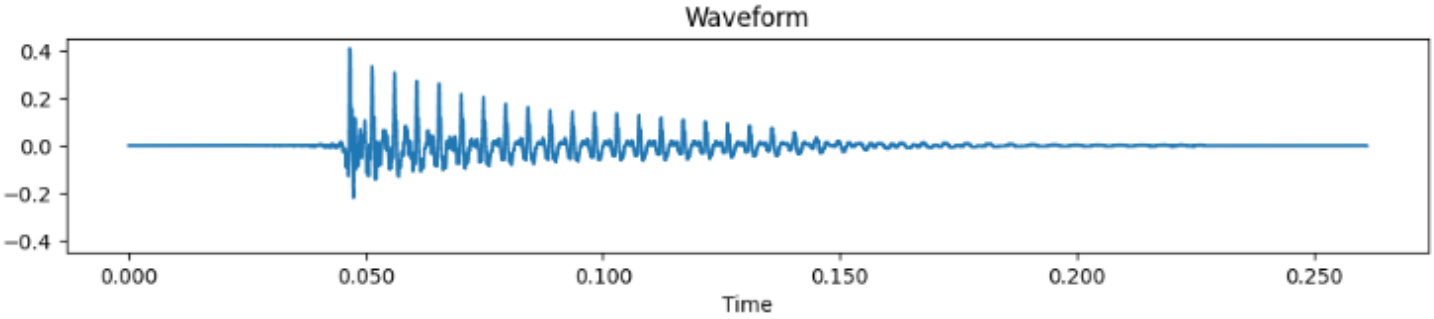

Рис. 3.3 Графік звукового сигналу

Далі відобразимо спектрограму звуку за допомогою librosa.display.specshow:

```
X = librosa.stft(x)
Xdb = librosa.amplitude_to_db(abs(X))
plt.figure(figsize=(14, 5))
librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')
plt.colorbar()
```

.stft() перетворює дані у короткочасне перетворення Фур'є. За допомогою STFT можна визначити амплітуду різних частот, що відтворюються у даний момент часу аудіосигналу. Для відображення спектрограми використаємо .specshow.

На вертикальній вісі відображені частоти (від 0 до 10 кГц), а на горизонтальній — час. Оскільки всі події відбуваються у нижній частині спектра, ми можемо перетворити вісь частот на логарифмічну.

```
librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='log')
plt.colorbar()
```

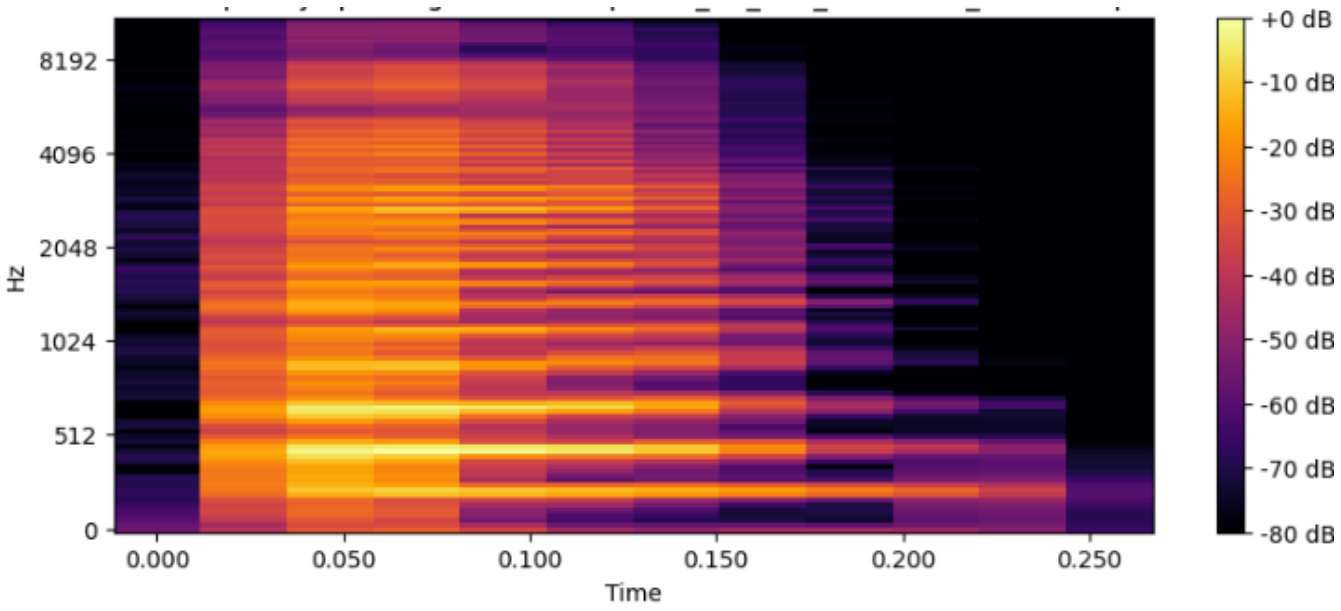

Рис. 3.4 Спектрограма з логарифмічною віссю частот

Створення аудіосигналу:

```
import numpy as np
sr = 22050 # частота дискретизації
T = 5.0    # секунди
t = np.linspace(0, T, int(T*sr), endpoint=False) # змінна часу
x = 0.5*np.sin(2*np.pi*220*t) # чиста синусоїдна хвиля при 220 Гц
# програвання аудіо
ipd.Audio(x, rate=sr) # завантаження масиву NumPy
# збереження аудіо
librosa.output.write_wav('tone_220.wav', x, sr)
```

Кожний аудіосигнал складається з множини ознак. Ми будемо витягувати лише ті характеристики, які стосуються вирішуваної нами проблеми. Це й називається як витягування ознак.

Спектральні (частотні) ознаки отримуються шляхом перетворення часового сигналу в частотну область за допомогою перетворення Фур'є. Це перетворення дозволяє розкласти сигнал на його складові частини за різними частотами.

До спектральних ознак належать: частота основного тону; сила сигналу на кожній частоті; середня частота сигналу; спектральний центроїд, потік, щільність та спад.

Спектральні ознаки використовуються в різних областях, включаючи:
Обробку аудіо: Для аналізу та класифікації звукових сигналів, наприклад, для розпізнавання мови або музики.
Обробку зображень: Для аналізу та класифікації зображень, наприклад, для виявлення об'єктів або тексту.
Медичну діагностику: Для аналізу біологічних сигналів, наприклад, для виявлення серцевих захворювань або раку.

1. **Спектральний центроїд**

Це величина, яка вказує, на якій частоті зосереджена енергія спектра, тобто фактично вказує, де розташований центр мас для звуку.

$$f_c = \frac{\sum_k f(k)S(k)}{\sum_k S(k)},$$

де $S(k)$ – спектральна величина елемента k, а $f(k)$ – частота елемента k.

За допомогою librosa.feature.spectral_centroid можна обчислити спектральний центроїд для кожного кадру(фрейму) у сигналі:

```
import sklearn

spectral_centroids = librosa.feature.spectral_centroid(x, sr=sr)[0]
spectral_centroids.shape(775,)
plt.figure(figsize=(12, 4))
frames = range(len(spectral_centroids))
t = librosa.frames_to_time(frames)

# Нормалізація спектрального центроїда для візуалізації

def normalize(x, axis=0):
    return sklearn.preprocessing.minmax_scale(x, axis=axis)

# Побудова спектрального центроїда разом з формою хвилі
librosa.display.waveplot(x, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_centroids), color='b')
```

А spectral_centroid повертає масив з кількістю стовпців, що дорівнює кількості кадрів, що представлені у семплі.

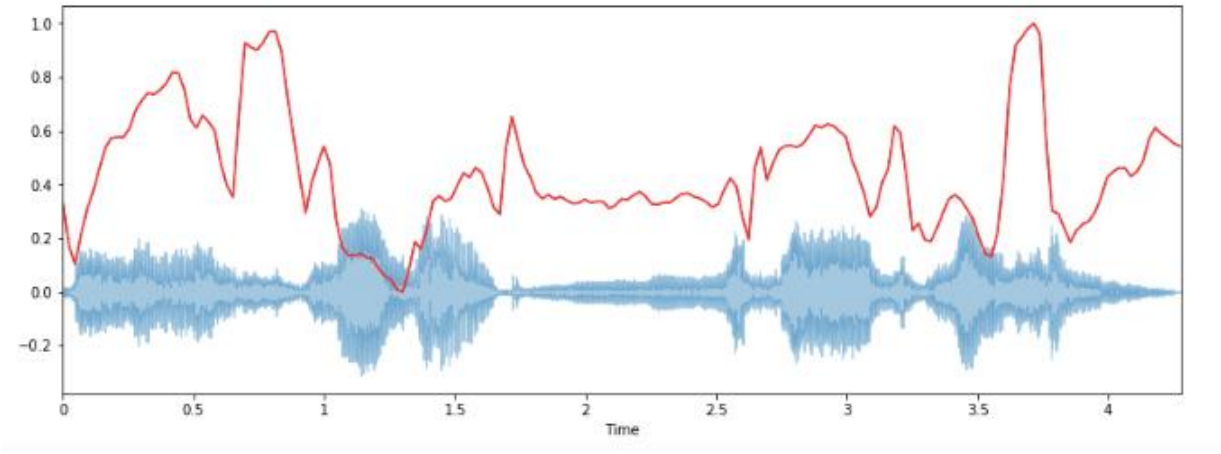

Рис. 3.5 Візуалізація спектрального центроїда і хвилі звуку

2. Спектральний спад(Spectral Rollof)

Це міра форми сигналу, що представляє собою частоту, в якій високі частоти знижуються до нуля.

$$\sum_{k}^{R_t} S(k) = 0.85 \sum_{k} S(k).$$

Щоб отримати його, потрібно розрахувати частку елементів у спектрі потужності, де 85% його потужності знаходиться на нижчих частотах.

За допомогою librosa.feature.spectral_rolloff можна обчислити частоту затухання для кожного кадру у сигналі:

```
spectral_rolloff = librosa.feature.spectral_rolloff(x+0.01, sr=sr)[0]
plt.figure(figsize=(12, 4))
librosa.display.waveplot(x, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_rolloff), color='r')
```

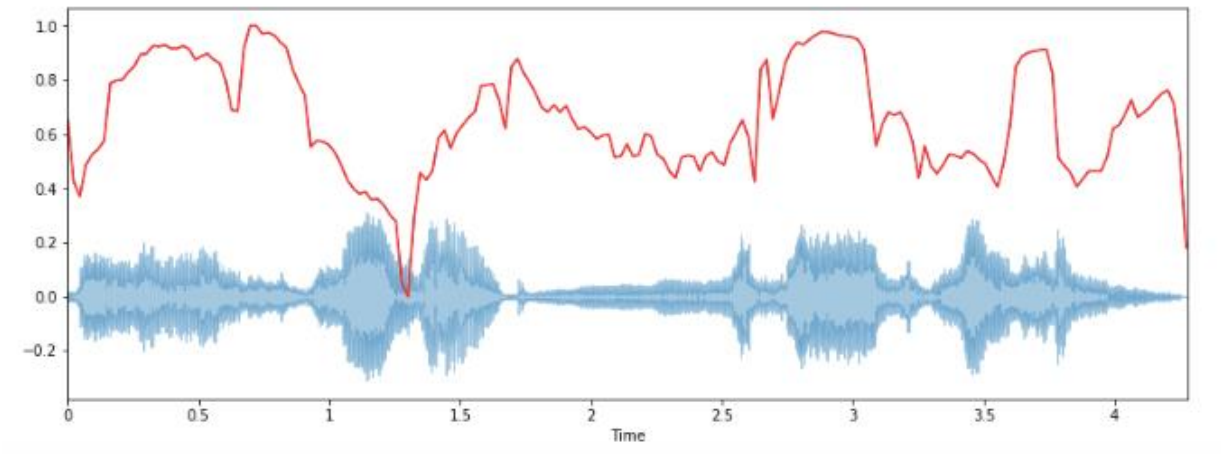

Рис. 3.6 Графік спаду частот

3. Спектральна ширина(Spectral Bandwidth)

Спектральна ширина визначається як ширина світлової смуги на половині максимальної точки(або повна ширина на половині максимума) і представлена двома вертикальними червоними лініями та λ_{SB} на вісі довжин хвиль.

$$SB_p = (\sum_k S(k)(f(k) - f_c)^p)^{\frac{1}{p}}.$$

Ширина смуги - це різниця між верхніми та нижніми частотами у континуальній смузі частот. Де $S(k)$ - спектральна амплітуда на частоті розмірністю k, $f(k)$ - частота k, а f_c - спектральний центроїд.

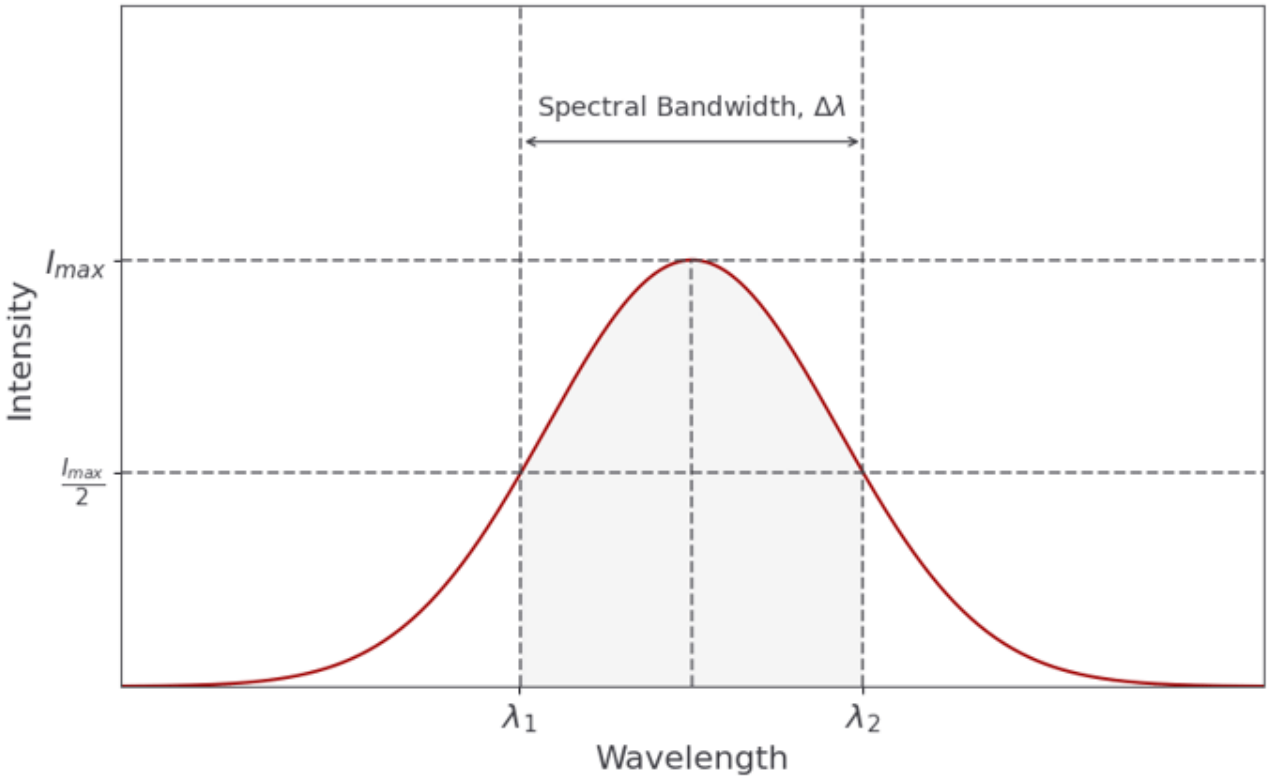

Рис. 3.7 Графік спектральної ширини

За допомогою librosa.feature.spectral_bandwidth можна обчислити спектральну ширину порядку p.

4. Кількість перетинів нуля

Кількість перетинів нуля(Zero Crossing Rate) вказує на кількість разів, коли сигнал перетинає горизонтальну вісь. ZCR низька для гармонійних звуків і висока для шумних звуків.

$$ZCR = \frac{1}{T-1}\sum_{t=1}^{T-1} II\{s_t s_{t-1} < 0\},$$

де s_t – це сигнал довжиною t; II – функція-індикатор, яка дорівнює 1 якщо X true, else = 0.

Простий спосіб вимірювання плавності сигналу полягає у визначенні кількості перетинів нуля у межах сегмента цього сигналу. Голосовий сигнал коливається повільно. Наприклад, сигнал частотою 100 Гц буде перетинати нуль 100 разів на секунду, тоді як "німий" фрикативний сигнал може мати 3000 перетинів нуля на секунду.

Вищі значення спостерігаються у таких високоударних звуках, як метал і рок. Тепер давайте візуалізуємо цей процес і розглянемо вимірювання швидкості перетину нуля.

```
x, sr = librosa.load('/../../gruesome.wav')
plt.figure(figsize=(14, 5))
librosa.display.waveplot(x, sr=sr)
n0 = 9000
n1 = 9100
plt.figure(figsize=(14, 5))
plt.plot(x[n0:n1])
plt.grid()
```

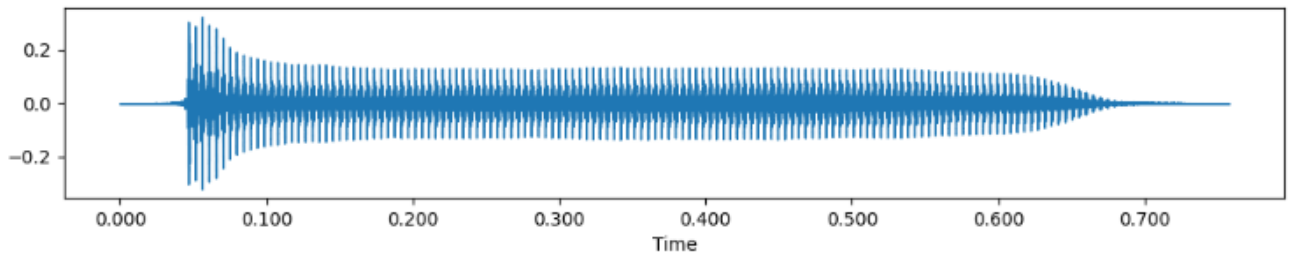

Рис. 3.8 Графік перетину нуля

5. Мел-частотні кепстральні коефіцієнти

Мел-частотні кепстральні коефіцієнти (MFCCs) сигналу - це набір ознак, який коротко описує загальну форму спектрального огортання. MFCC беруть

STFT сигналу(віконний уривок сигналу) та відображають потужності спектра на мел-шкалу, використовуючи трикутні перекриваючі вікна. Далі беруться логарифми потужностей на кожній з мел-частот. Потім беруть дискретне косинусне перетворення списку логарифмів потужностей на мел-частотах. MFCCs - це амплітуди отриманого спектра, які коротко описують загальну форму спектральної кривої.

```
mfccs = librosa.feature.mfcc(x, sr=fs)
print(mfccs.shape)
(20, 97)
# Відображення MFCC:
plt.figure(figsize=(15, 7))
librosa.display.specshow(mfccs, sr=sr, x_axis='time')
```

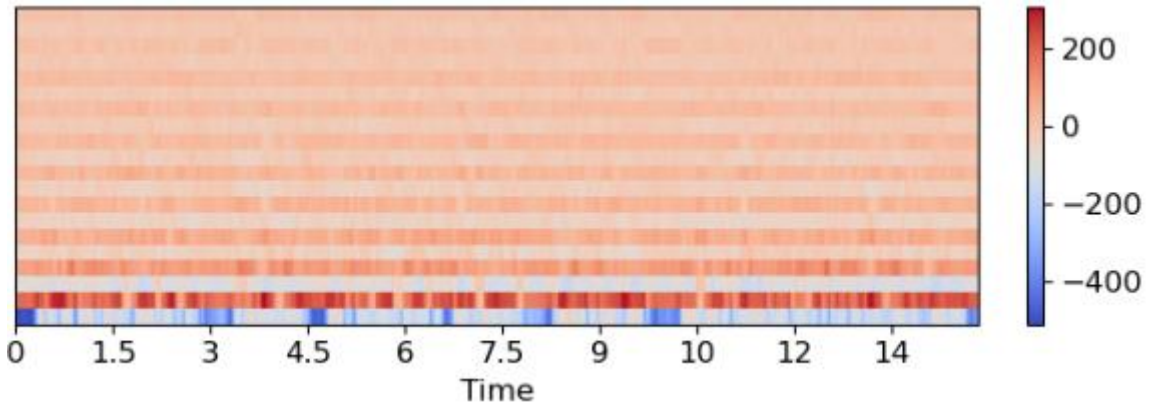

Рис. 3.9 MFCC

6. Вектор насиченості

Це вектор ознак, який зазвичай складається з 12 елементів, в якому вказано кількість енергії кожного висотного класу {C, C#, D, D#, E, E#...} в сигналі. Використовується для опису ступеня схожості між музичними творами. Для визначення ознак насиченості використовується librosa.feature.chroma.stft:

```
chromagram = librosa.feature.chroma_stft(x, sr=sr, hop_length=hop_length)
plt.figure(figsize=(15, 5))
librosa.display.specshow(chromagram, x_axis=’time’, y_axis=’chroma’,
hop_length=hop_length, cmap=’coolwarm’)
```

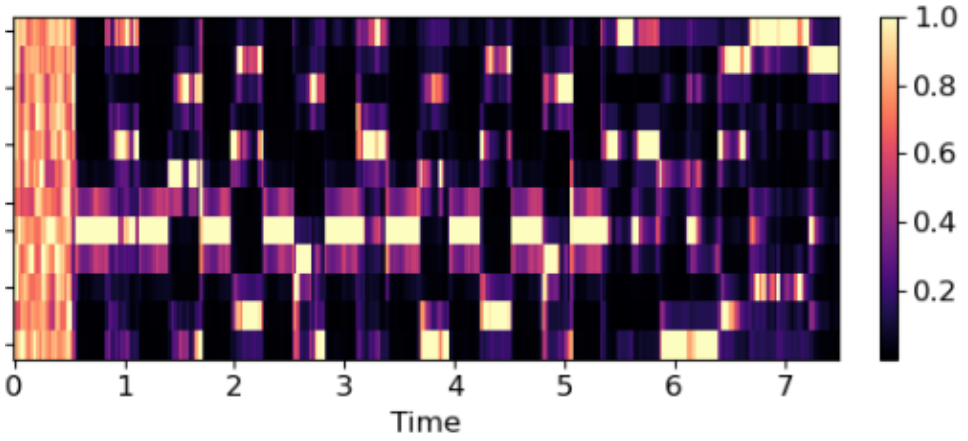

Рис. 3.10 Спектрограма насиченості

7. Спектральний контраст

Ознака спектрального контрасту(SC) враховує спектральний пік та частоту спаду кожного піддіапазону окремо, таким чином, відображаючи відносні спектральні характеристики нижніх діапазонів. Спектральні піки та спади оцінюються шляхом обчислення середнього значення невеликого оточення(вказаного через α) навколо максимуму та мінімуму піддіапазону.

k - це частотний діапазон, а $x_{k,i}$ - піддіапазон аудіосигналу, впорядкований за спаданням величини.

$$Peak_k = \log\left(\frac{1}{\alpha N}\sum_{i=1}^{\alpha N} x_{k,i}\right), \qquad Valley_k = \log\left(\frac{1}{\alpha N}\sum_{i=1}^{\alpha N} x_{k,N-i+1}\right),$$

$$SC_k = Peak_k - Valley_k.$$

8. Спектральний потік(Spectral Flux)

Це міра того, наскільки швидко потужність спектра сигналу обчислюється порівнянням потужності спектра одного кадру з потужністю спектра попереднього кадру. Результат спектрального потоку через $H(x)$ в основному використовується для визначення моменту початку гри музичного інструмента, оскільки він містить лише інформацію про збільшену енергію порівняно з попереднім кадром.

$$SF(n) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} H(|X(n,k)| - |X(n-1,k)|), \qquad H(x) = \frac{x+|x|}{2}.$$

Після отримання всіх необхідних для навчання моделі спектральних ознак. Перед нами постає фінальне, найбільш цікаве і складне завдання – використання алгоритму машинного навчання, задля тренування моделі на основі набору музичних творів. З них потрібно видобути потрібні нам характеристики для порівняння з даним аудіозаписом та класифікувати інструменти, які грають в нашій мелодії.

Проте існує багато алгоритмів машинного навчання, їх застосовують для великої кількості завдань. Вони мають різні підходи та особливості, які роблять їх придатними для різних завдань. Деякі з них зосереджені на класифікації даних, інші на прогнозуванні або кластеризації. Деякі можуть працювати лише з великими обсягами даних, тоді як інші підходять для роботи з обмеженою кількістю наборів даних. Тож потрібно визначити, яка модель є оптимальною для нашого завдання з точки зору отримання хорошої точності результату, за використання невеликої потужності системи.

Розглянемо моделі, які можна застосувати для нашого завдання:

1) Approximate Nearest Neighbor(ANN)

Алгоритм пошуку приблизного найближчого сусіда дозволяє повертати точки, відстань яких, не більше ніж в C разів перевищує відстань від запиту до його найближчих точок. Цей підхід дуже зручний тому, що в багатьох випадках приблизне значення найближчого сусіду дуже близьке до точного значення.

2) KNN

Це непараметрична методологія, яка передбачає нові дані з інформацією про k найближчих сусідів існуючих даних, коли надходять нові дані. Гіперпараметрами KNN є кількість сусідів для пошуку k та метод вимірювання відстані.

Різниця між KNN і ANN полягає в тому, що на фазі прогнозування всі навчальні точки беруть участь у пошуку k -найближчих сусідів в алгоритмі

KNN, але в ANN цей пошук починається лише з невеликої підмножини точок-кандидатів.

3) Support Vector Machine

SVM – це метод навчання “з вчителем”, який припускає будь-які границі прийняття рішень. Серед даних, що належать до кожного класу, найближчі до межі дані називаються опорними векторами. Сума відстані між опорним вектором та границею прийняття рішень називається зазором ($|w|$). SVM є методом для пошуку границі класифікації, яка максимізує зазори.

$$obj(f) = max\frac{2}{||w||_2} \rightarrow min\frac{1}{2}||w||_2^2, \qquad sub: y_i(w^T x_i + b) \geq 1.$$

4) Softmax regression

$$\begin{bmatrix} P(y = 1 | x) \\ \cdots \\ P(y = n | x) \end{bmatrix} = \begin{bmatrix} \frac{exp(\theta_1^T x)}{\sum_k exp(\theta_k^T x)} \\ \cdots \\ \frac{exp(\theta_n^T x)}{\sum_k xp(\theta_k^T x)} \end{bmatrix}.$$

Це спосіб вивчити найбільш ймовірні параметри y -мітки у просторі ймовірностей. Де $x(i)$ - вектор для всіх ознак, $x(i)$ - одноколоночний вектор форми $[D,1]$. - параметр softmax регресії, і кожна мітка має свій відповідний параметр.

5) Gaussian Naive Bayes

$$P(x_d | y = k) = \frac{1}{\sqrt{2\pi\sigma_{d,k}^2}} exp(-\frac{(x_d - \mu_{d,k})^2}{2\sigma_{d,k}^2}.$$

Якщо вектор ознак x має кілька вимірів, тобто якщо $x = (x_1, \ldots, x_d)$, то ймовірність всіх ймовірностей $x = (x_1, \ldots, x_D)p(x_1, \ldots, x_D | y = k)$. Однак, чим більше вимірів, тим складніше отримати цю багатовимірну ймовірність асоціації, тому припущення полягає в тому, що окремі незалежні змінні кожного виміру умовно незалежні одна від одної. Модель, яка застосовує це припущення до класифікаційної моделі Байєса, - це модель класифікації

наївного Байєса. Для кожної незалежної змінної x_d і для класу k припускається відмінний від нуля розподіл Гауса.

6) Decision Tree

$$IG[Y,X] = H[Y] - H[Y|X]H(x) = -\sum_x P(x)\log P(x).$$

Дерева рішень (DTs) - це непараметрична методологія. Метод визначення правила класифікації полягає у пошуку найкращої незалежної змінної та посилання, які зменшують ентропію між батьківським вузлом та дочірнім вузлом на найменше значення. Кількісне визначення цих критеріїв - це приріст інформації(IG). Основна ідея приросту інформації полягає в тому, що малоймовірні події є більш інформативними, ніж часто відбуваються події.

7) Random Forest

Випадковий ліс - це модель, яка поєднує метод, що використовує дерево рішень як окрему модель. В цьому методі вибирається та використовується лише частина простору ознак даних. Однак ми не вибираємо кращі незалежні змінні, порівнюючи всі незалежні змінні під час розділення вузла, а випадковим чином зменшуємо розмір незалежних змінних, а потім серед них вибираємо незалежну змінну.

Table 1: Accuracy of Hypotheses set.ML-Algorithm

hypotheses set	SVM	Softmax	GaussianNB	KNN	DT	RF
Timber	91.0	88.0	77.5	84.5	70.5	88.0
Pitch/harmony	63.5	36.5	42	55.0	44.5	55
Rhythm	15.5	14.5	14.0	8.5	6.5	6.5
Timber + Pitch/harmony	90.5	88.0	30	71.0	63.5	89.0
Timber + Rhythm	92.5	88.0	73.0	85.5	61.5	88.5
Pitch/harmony + Rhythm	64	40.5	30.0	56.0	20.0	45.5
Timber + Rhythm + Pitch/harmony	91.5	87.0	77.5	73.0	62.5	90.5

Рис. 3.11 Таблиця порівняння точності моделей машинного навчання.

Було проведено порівняння точності даних моделей тренованих на основі набору даних, який складався з 1200 тренувальних уривків тривалістю в 4 секунди кожен(джерело [10]). Можемо побачити, що найбільш точним методом є SVM, з точністю навіть більшою за 90%. Інші моделі продемонстрували теж непогану точність. Проте тривалість даних аудіозаписів доволі мала, при збільшенні тривалості – точність буде

зменшуватися. Тому в загальному випадку, для довших аудіозаписів, отримати таку точність буде досить важко.

Отримання результатів

Тож, розглянувши всі необхідні спектральні ознаки та розібравшись, як їх видобувати із спектрограм за допомогою бібліотек, ми можемо перейти до використання всіх перелічених функцій та побудувати модель, щоб класифікувати музичні інструменти в аудіозаписах.

Ідея класифікації:

Беремо навчальний набір даних – аудіозаписи невеликої тривалості, кожен з яких міститиме гру всього одного інструменту. Для кожного аудіозапису створимо спектрограму та видобудемо всі необхідні спектральні ознаки. За цими ознаками проведемо тренування моделі. Також виберемо набір аудіозаписів різної тривалості для тестування, в якому вже може бути мелодія декількох інструментів. Застосуємо для цього набору нашу модель та перевіримо її точність.

Код програм знаходиться в додатку.

Отже, спершу знаходимо і завантажуємо набори даних. Далі імпортуємо всі необхідні бібліотеки: librosa, pandas, numpy, matplotlib, PIL(для роботи з зображеннями), pathlib, csv(зручний формат для роботи з моделлю), sklearn і keras.

Наступним кроком конвертуємо наші дані в спектрограми у форматі PNG. Вилучаємо всі необхідні спектральні ознаки: спектральний центроїд, MFCC, кількість перетинів нуля, частоту насиченості і спад спектру. Та зберігаємо ознаки в файлі csv, розділяючи на навчальний набір та тестувальний.

Створюємо нашу модель ANN:

```
model = Sequential()
model.add(layers.Dense(256, activation='relu', input_shape=(X_train.shape[1],)))
model.add(layers.Dense(128, activation='relu'))
```

```
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

classifier = model.fit(X_train,
                       y_train,
                       epochs=100,
                       batch_size=128)
```

Навчання проводимо на 100 епохах, використовуючи не дуже потужну систему. Кожен навчальний аудіозапис містить гру лиш одного інструменту: саксофон, труба, скрипка, віолончель, флейта чи духовий інструмент гобой.

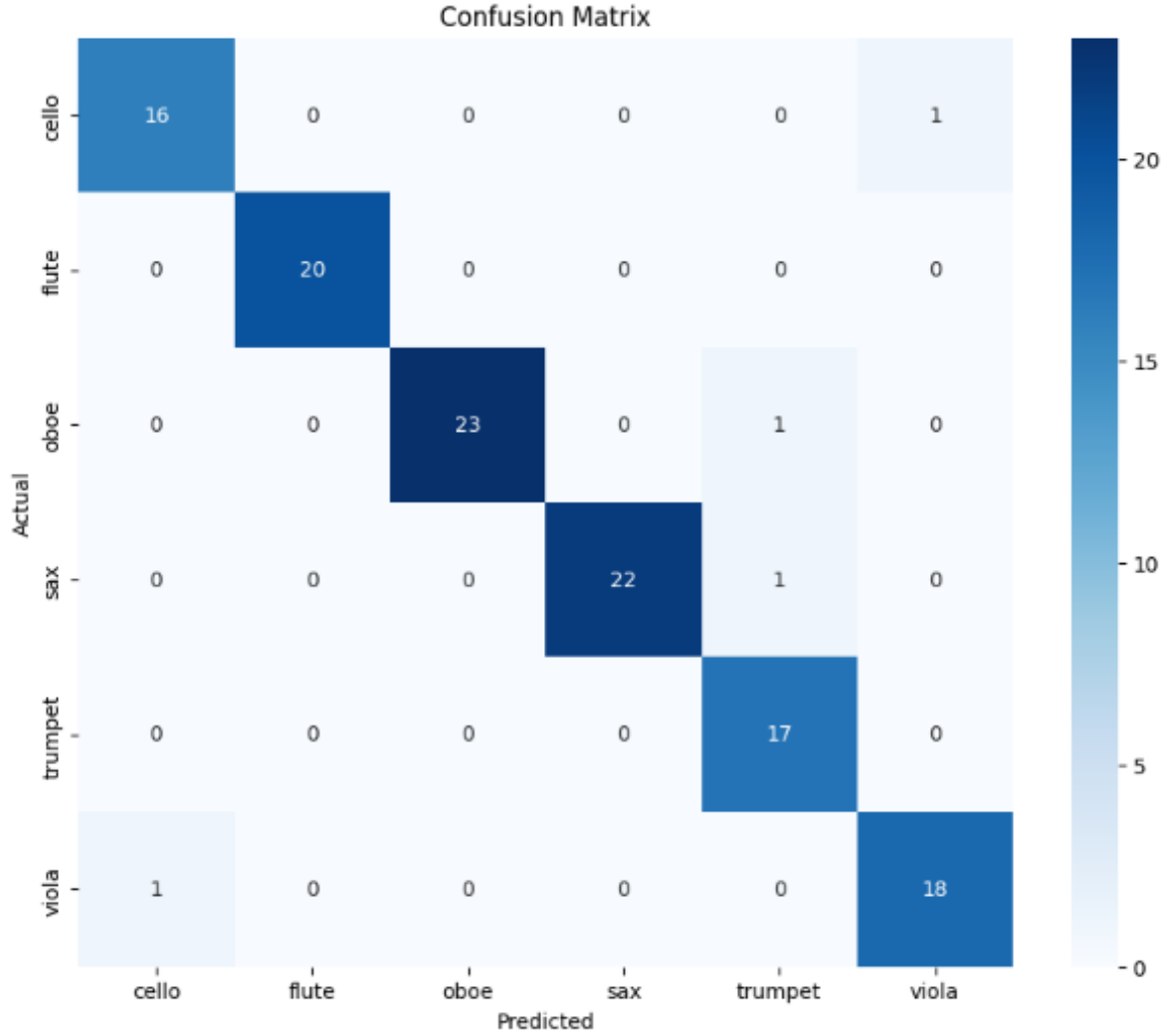

Рис. 3.12 Діаграма результату класифікації інструментів моделі ANN

Після цього можемо перевірити точність класифікації яку ми отримали. Переглянувши csv файл з результатами ми бачимо, що точність складає 96,67%.

Також реалізуємо модель SVM:

```
# SVM
#model_svm = LinearSVC(random_state=0, tol=1e-5, max_iter=5000)
svclassifier = SVC(kernel='rbf', C = 10.0, gamma=0.1)

svclassifier.fit(train_set, train_classes);

joblib.dump(svclassifier, 'trainedSVM.joblib')

predicted_labels = svclassifier.predict(test_set)
```

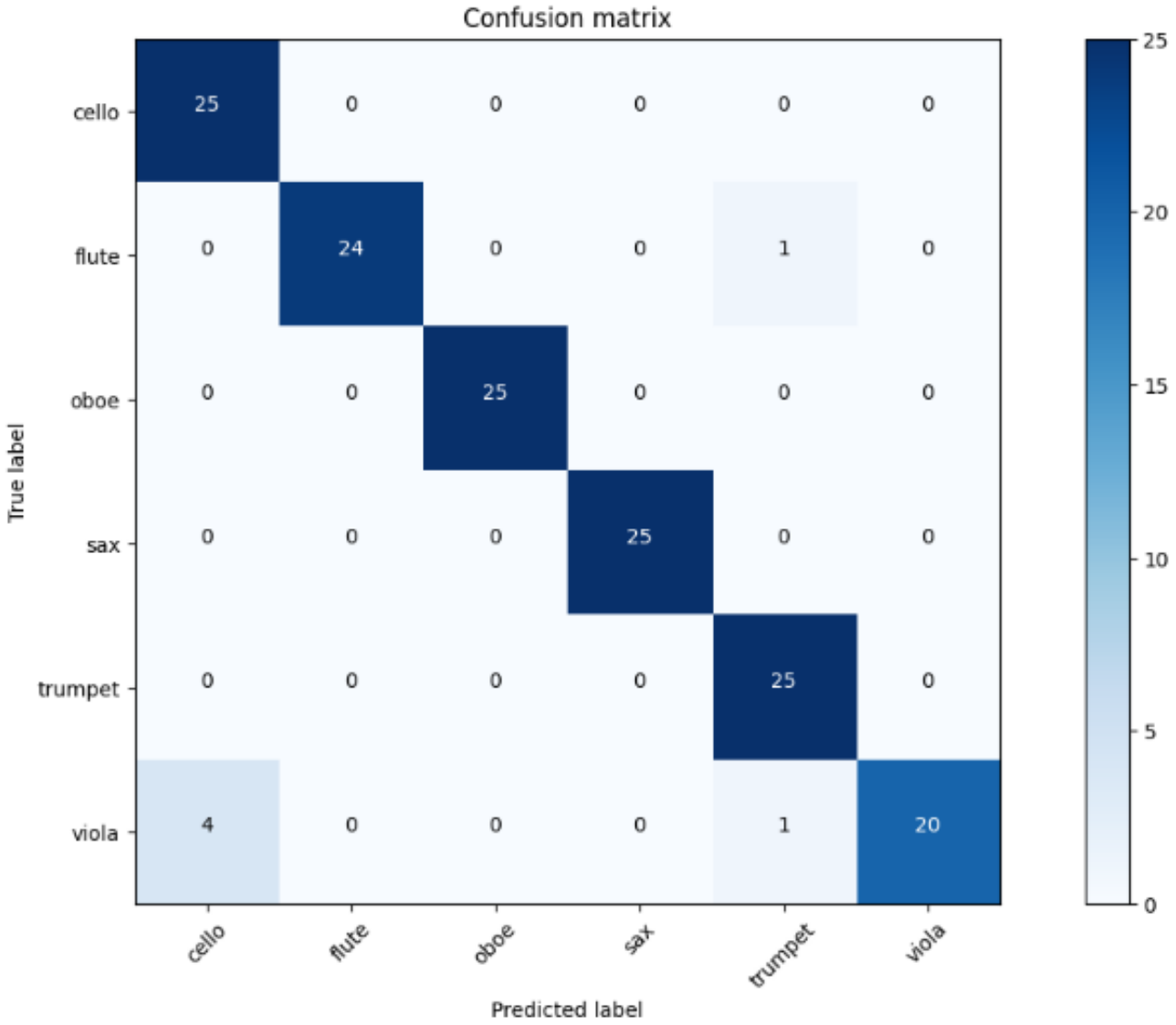

Рис. 3.13 Діаграма результату класифікації інструментів моделі SVM

Точність даного підходу складає 96%

Висновок

Ми вилучили необхідні спектральні ознаки: спектральний центроїд, MFCC, кількість перетинів нуля, частоту насиченості і спад спектру. Реалізували моделі ANN та SVM, провели навчання і отримали в результаті дуже хорошу точність класифікації.

Проаналізувавши реалізації методів, вважаю, що будь-який з них можна використати за умов, що аудіозаписи не будуть довгої тривалості та інструменти в них не будуть перекривати один одного. Мушу визнати, що для задачі класифікації інструментів в багатозвучному оркестрі, даний підхід не буде достатньо продуманим. Він працює тільки за конкретних умов, але дуже простий в розумінні та реалізації. Такий підхід також можна використовувати для класифікації жанрів музики, де порівняння ознак за спектрограмами доволі точно відображає співпадіння характеристик аудіозаписів.

Як ми вже побачили, навіть використовуючи поширені методики, потрібно враховувати всі нюанси завдання. Не можна просто взяти і перенести модель з однієї задачі на іншу, навіть якщо це в межах однієї сфери застосування. Необхідно більш продумано підходити до обробки і аналізу сигналу, підбору даних та оптимізації алгоритмів.

Список використаних джерел

1. Заболотній С. В. Цифрове оброблення сигналів: Посібник для студентів напряму підготовки 6.050901 "Радіотехніка" усіх форм навчання [Електронний ресурс] / Авт.-укл. С. В.Заболотній; за ред. проф. Ю. Г. Леги; М-во освіти і науки України, Черкас. держ. технол. ун-т. – Черкаси: ЧДТУ, 2010. – 119 с. ISBN 978-966-402-093-7.

2. Сміт С. - Цифрова обробка сигналів. Посібник-Додека (2012).
Digital Signal Processing A Practical Guide for Engineers and Scientists by Steven W. Smith. Library of Congress Cataloging-in-Publication Data ISBN 0-75067444-X.

3. (Undergraduate Lecture Notes in Physics) Tilman Butz (auth.) - Fourier Transformation for Pedestrians-Springer International Publishing (2015).
Faculty of Physics and Earth Science, University of Leipzig, Leipzig, Germany. ISSN 2192-4791 ISSN 2192-4805(electronic). Undergraduate Lecture Notes in Physics. ISBN 978-3-319-16984-2 ISBN 978-3-319-16985-9(eBook). DOI 10.1007/978-3-319-16985-9.

4. Нуссбаумер Г. – Швидке перетворення Фур’є і алгоритми обчислення згорток - Радіо і зв’язок. By Springer-Verlag Berlin Heidelberg 1981 and 1982.
ISBN-13:978-3-540-11825-1. DOI 10.1007/978-3-642-81897-4

5. Сергієнко А. Б. Цифрова обробка сигналів 2011. 768с.

6. Лайонс Р. Цифрова обробка сигналів “Біном-Прес” 2007. 656с.

7. José Unpingco Python for Signal Processing.
San Diego, CA, USA. ISBN 978-3-319-01341-1 ISBN 978-3-319-01342-8 (eBook). DOI 10.1007/978-3-319-01342-8. Springer Cham Heidelberg New York Dordrecht London.

8. Fundamentals of Music Processing, Meinard Müller, International Audio Laboratories, Erlangen, Germany,
ISBN 978-3-319-21944-8 ISBN 978-3-319-21945-5 (eBook),
DOI 10.1007/978-3-319-21945-5

9. Automatic musical instrument recognition from polyphonic music audio signals Ferdinand Fuhrmann TESI DOCTORAL UPF / 2012

10. G.Tzanetakis and P.Cook Musical genre classification of audio signals. In IEEE Transactions on Speech and Audio Processing, c. 293-302. IEEE, 2002.

11. Jiang, Dan-Ning and Lu, Lie and Zhang, Hong-Jiang and Tao, Jian-Hua and Cai, Lian-Hong Music type classification

by spectral contrast feature In Proceedings. IEEE International Conference on Multimedia and Expo, c. 113–116. IEEE, 2002.

12. McFee, Brian and Raffel, Colin and Liang, Dawen and Ellis, Daniel PW and McVicar, Matt and Battenberg, Eric and Nieto, Oriol librosa: Audio and music signal analysis in python Proceedings of the 14th python in science conference, 2015.

13. A brief introduction to audio data processing and genre classification using Neural Networks and python by Nagesh Singh Chauhan, KDnuggets on February 19, 2020 in Audio, Data Processing, Deep Learning, Python.

Додаток 1

Програмний код реалізації 3.1

```
import numpy as np
import matplotlib.pyplot as plt

# Згенеруємо сигнал, що складається з двох синусоїд з різними частотами
t = np.linspace(0, 1, 1000)
f1, f2 = 10, 20
x = np.sin(2*np.pi*f1*t) + np.sin(2*np.pi*f2*t)

# Застосовуємо перетворення Фур'є
X = np.fft.fft(x)

# Обчислюємо частоти
freqs = np.fft.fftfreq(len(x), t[1] - t[0])

# Розбиваємо сигнал на дійсну та уявну частини
re_X = np.real(X)
im_X = np.imag(X)

# Визначаємо амплітуду та фазу кожної складової
amp_X = np.abs(X)
phase_X = np.angle(X)

# Відображаємо сигнал та його складові частоти
fig, axs = plt.subplots(2, 2, figsize=(10, 8))
axs[0, 0].plot(t, x)
axs[0, 0].set_title('Input signal')
axs[0, 1].plot(freqs, re_X, label='Real part')
axs[0, 1].plot(freqs, im_X, label='Imaginary part')
axs[0, 1].legend()
axs[0, 1].set_title('Frequency domain')
axs[1, 0].stem(freqs, amp_X)
axs[1, 0].set_title('Amplitude spectrum')
axs[1, 1].stem(freqs, phase_X)
axs[1, 1].set_title('Phase spectrum')
plt.show()

# Застосовуємо обернене перетворення Фур'є
x_reconstructed = np.fft.ifft(X)
```

```
# Відображаємо вихідний сигнал та відновлений сигнал
fig, axs = plt.subplots(2, 1, figsize=(10, 6))
axs[0].plot(t, x, label='Input signal')
axs[0].plot(t, np.real(x_reconstructed), label='Reconstructed signal')
axs[0].legend()
axs[0].set_title('Time domain')
axs[1].stem(freqs, np.abs(np.fft.fftshift(X)))
axs[1].set_title('Frequency domain')
plt.show()
```

Програмний код реалізації 3.2

```
import numpy as np
import matplotlib.pyplot as plt

# Задані параметри сигналу
f0 = 100
delta_f = 10

# Задана функція f(t)
def func(t):
    return np.cos(2 * np.pi * f0 * t) + np.cos(2 * np.pi * (f0 + delta_f) * t) + np.cos(2
* np.pi * (f0 + 2 * delta_f) * t)

# Обчислення перетворення Фур'є
N = 1024
T = 1 / f0
t_values = np.linspace(0, N * T, N)
k_values = np.arange(-N/2, N/2)
f_values = np.zeros_like(k_values, dtype=np.complex128)

for i, k in enumerate(k_values):
    integrand = func(t_values) * np.exp(-1j * 2 * np.pi * k * t_values)
    f_values[i] = np.sum(integrand)

# Обчислення спектральної щільності потужності
P_values = np.abs(f_values) ** 2

# Візуалізація результату
plt.subplot(211)
plt.plot(t_values, func(t_values))
plt.xlabel('t')
plt.ylabel('f(t)')
plt.title('Сигнал у часовій області')
```

```
plt.subplot(212)
plt.plot(k_values, P_values)
plt.xlabel('f')
plt.ylabel('P(f)')
plt.title('Спектральна щільність потужності')

plt.show()
```

Програмний код реалізації 3.3

```
import numpy as np
import matplotlib.pyplot as plt

# Generate noisy sine wave
fs = 1000
t = np.arange(0, 1, 1/fs)
signal = np.sin(2 * np.pi * 50 * t) + np.sin(2 * np.pi * 120 * t)
noise = 0.5 * np.random.randn(len(t))
noisy_signal = signal + noise

# Compute Fourier Transform and frequency axis
fourier = np.fft.fft(noisy_signal)
freq = np.fft.fftfreq(len(noisy_signal), 1.0 / fs)

# Define band-pass filter
filter = np.zeros(len(freq))
filter[(freq >= 40) & (freq <= 60)] = 1
filter[(freq >= 110) & (freq <= 130)] = 1

# Filter signal in frequency domain
filtered_fourier = fourier * filter

# Compute inverse Fourier Transform to get filtered signal
filtered_signal = np.real(np.fft.ifft(filtered_fourier))

# Plot original and filtered signals
plt.plot(t, noisy_signal, label='Noisy signal')
plt.plot(t, filtered_signal, label='Filtered signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.legend()
plt.show()
```

Додаток 2

ANN

```
#Імпортуємо всі необхідні бібліотеки
import librosa
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import os
from PIL import Image
import pathlib
import csv
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
import keras
from keras import layers
from keras import layers
import keras
from keras.models import Sequential
import warnings
warnings.filterwarnings('ignore')


#Конвертуємо файли аудіозаписів в PNG – робимо спектрограму для кожного з них
cmap = plt.get_cmap('inferno')
plt.figure(figsize=(8, 8))

# Список інструментів
instruments = 'sax flute viola cello trumpet oboe'.split()

# Шлях до аудіофайлів
base_path = r'E:\Windows Kits\analis\audio\london_phill_dataset_multi'

# Створення спектрограм для кожного інструмента
for instrument in instruments:
    instrument_path = os.path.join(base_path, instrument)
    output_path = os.path.join('img_data', instrument)
    pathlib.Path(output_path).mkdir(parents=True, exist_ok=True)

    for filename in os.listdir(instrument_path):
        audioname = os.path.join(instrument_path, filename)
        try:
```

```
            y, sr = librosa.load(audioname, mono=True, duration=5)
            plt.specgram(y, NFFT=2048, Fs=2, Fc=0, noverlap=128, cmap=cmap,
sides='default', mode='default', scale='dB')
            plt.axis('off')
            plt.savefig(os.path.join(output_path, f'{filename[:-4]}.png'))
            plt.clf()
        except Exception as e:
            print(f"Error loading {audioname}. Error: {e}")


# Створення заголовка для CSV файлу
header = 'filename chroma_stft rmse spectral_centroid spectral_bandwidth rolloff
zero_crossing_rate'
for i in range(1, 21):
    header += f' mfcc{i}'
header += ' label'
header = header.split()

# Запис заголовка у CSV файл
with open(csv_file_path, 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(header)

instruments = 'sax flute viola cello trumpet oboe'.split()

# Отримання ознак та запис у CSV файл
for instrument in instruments:
    instrument_path = os.path.join(base_path, instrument)
    for filename in os.listdir(instrument_path):
        audioname = os.path.join(instrument_path, filename)
        try:
            y, sr = librosa.load(audioname, mono=True, duration=30)
            rmse = librosa.feature.rms(y=y)
            chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)
            spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)
            spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
            rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
            zcr = librosa.feature.zero_crossing_rate(y)
            mfcc = librosa.feature.mfcc(y=y, sr=sr)
            to_append = f'{filename} {np.mean(chroma_stft)} {np.mean(rmse)}
{np.mean(spec_cent)} {np.mean(spec_bw)} {np.mean(rolloff)} {np.mean(zcr)}'
            for e in mfcc:
                to_append += f' {np.mean(e)}'
            to_append += f' {instrument}'
```

```
            with open(csv_file_path, 'a', newline='') as file:
                writer = csv.writer(file)
                writer.writerow(to_append.split())
        except Exception as e:
            print(f"Error processing {audioname}. Error: {e}")

# Обробка даних
data = pd.read_csv(csv_file_path)
print(data.head())

# Видалення непотрібних стовпчиків
data = data.drop(['filename'], axis=1)

# Створення міток
instrument_list = data.iloc[:, -1]
encoder = LabelEncoder()
y = encoder.fit_transform(instrument_list)

# Масштабування стовпчиків ознак
scaler = StandardScaler()
X = scaler.fit_transform(np.array(data.iloc[:, :-1], dtype=float))

# Розділення даних на навчальний та тестовий набір
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Перевірка результатів
print(f'X_train shape: {X_train.shape}')
print(f'X_test shape: {X_test.shape}')
print(f'y_train shape: {y_train.shape}')
print(f'y_test shape: {y_test.shape}')
#Створюємо і підлаштовуємо модель ANN
model = Sequential()
model.add(layers.Dense(256, activation='relu', input_shape=(X_train.shape[1],)))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

classifier = model.fit(X_train,
                       y_train,
                       epochs=100,
```

```
                batch_size=128)
```

SVM

```
import numpy as np
import pickle
import itertools

import os, fnmatch

import seaborn #visualization library, must be imported before all other plotting libraries
import matplotlib.pyplot as plt
from IPython.display import HTML, display

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import recall_score, precision_score, accuracy_score
from sklearn.metrics import confusion_matrix, f1_score, classification_report
from sklearn.svm import LinearSVC, SVC
import joblib


from numpy.random import seed
seed(1)

import librosa.display, librosa

path='E:\\Windows Kits\\analis\\audio\\london_phill_dataset_multi'

files = []
for root, dirnames, filenames in os.walk(path):
    for filename in fnmatch.filter(filenames, '*.mp3'):
        files.append(os.path.join(root, filename))

print("found %d audio files in %s"%(len(files),path))


labels =[]
classes=['flute','sax','oboe', 'cello','trumpet','viola']
color_dict={'cello':'blue',        'flute':'red',        'oboe':'green',        'trumpet':'black',
'sax':'magenta', 'viola':'yellow'}
color_list=[]
```

```
for filename in files:
    for name in classes:
        if fnmatch.fnmatchcase(filename, '*'+name+'*'):
            labels.append(name)
            color_list.append(color_dict[name])
            break
    else:
        labels.append('other')


labelencoder = LabelEncoder()
labelencoder.fit(labels)
print(len(labelencoder.classes_), "classes:", ", ".join(list(labelencoder.classes_)))
classes_num = labelencoder.transform(labels)


fs = 44100         # Sampling Frequency
n_fft = 2048       # length of the FFT window
hop_length = 512   # Number of samples between successive frames
n_mels = 128       # Number of Mel bands
n_mfcc = 13        # Number of MFCCs

# Machine Learning Parameters
testset_size = 0.25 #Percentage of data for Testing
n_neighbors=1


def get_features(y, sr=fs):
    S = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=n_mels)
    mfcc = librosa.feature.mfcc(S=librosa.power_to_db(S), n_mfcc=n_mfcc)
    feature_vector = np.mean(mfcc, axis=1)
    #feature_vector                                =                                (feature_vector-
np.mean(feature_vector))/np.std(feature_vector)
    return feature_vector


feature_vectors = []
sound_paths = []
for i, f in enumerate(files):
    print("get %d of %d = %s" % (i+1, len(files), f))
    try:
        y, sr = librosa.load(f, sr=fs)
        y /= y.max()  # Normalize
        if len(y) < 2:
```

```
            print("Error loading %s" % f)
            continue
        feat = get_features(y, sr)
        feature_vectors.append(feat)
        sound_paths.append(f)
    except Exception as e:
        print("Error loading %s. Error: %s" % (f, e))

print("Calculated %d feature vectors" % len(feature_vectors))


scaler = StandardScaler()
scaled_feature_vectors = scaler.fit_transform(np.array(feature_vectors))
print("Feature vectors shape:",scaled_feature_vectors.shape)


filename="mfcc_feature_vectors.pl"
with open(filename, "wb") as f:
    pickle.dump( scaled_feature_vectors, f)


splitter = StratifiedShuffleSplit(n_splits=1, test_size=testset_size,
random_state=0)
splits = splitter.split(scaled_feature_vectors, classes_num)
for train_index, test_index in splits:
    train_set = scaled_feature_vectors[train_index]
    test_set = scaled_feature_vectors[test_index]
    train_classes = classes_num[train_index]
    test_classes = classes_num[test_index]


print("train_set shape:",train_set.shape)
print("test_set shape:",test_set.shape)
print("train_classes shape:",train_classes.shape)
print("test_classes shape:",test_classes.shape)

# SVM
#model_svm = LinearSVC(random_state=0, tol=1e-5, max_iter=5000)
svclassifier = SVC(kernel='rbf', C = 10.0, gamma=0.1)

# SVM
#model_svm.fit(train_set, train_classes);
svclassifier.fit(train_set, train_classes);
```

```
# Save
joblib.dump(svclassifier, 'trainedSVM.joblib')
#Load
#svclassifier = joblib.load('filename.joblib')

#predicted_labels = model_svm.predict(test_set)
predicted_labels = svclassifier.predict(test_set)

print("Recall: ", recall_score(test_classes, predicted_labels,average=None))

print("Precision: ", precision_score(test_classes, predicted_labels,average=None))

print("F1-Score: ", f1_score(test_classes, predicted_labels, average=None))

print("Accuracy: %.2f ," % accuracy_score(test_classes, predicted_labels,normalize=True), accuracy_score(test_classes, predicted_labels,normalize=False) )
print("Number of samples:",test_classes.shape[0])


cnf_matrix = confusion_matrix(test_classes, predicted_labels)
np.set_printoptions(precision=2)


def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
    """
    #print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
```

```
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


plt.figure(figsize=(12,7))
plot_confusion_matrix(cnf_matrix, classes=labelencoder.classes_,
                      title='Confusion matrix')


wrong_predictions = [i for i, (e1, e2) in
enumerate(zip(predicted_labels,test_classes)) if e1 != e2]


print(np.array(labels)[test_index[wrong_predictions]])
print(predicted_labels[wrong_predictions].T)
print(labelencoder.inverse_transform(predicted_labels[wrong_predictions]))
print(np.array(files)[test_index[wrong_predictions]])
```