Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі»

Виконав: студент 2-го курсу, групи 2КН-21м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
Семчук В. О.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Колесницький О.К.
(прізвище та ініціали)
«15» 12 2022 р.

Опонент: к.т.н., професор каф. КСУ
Биков М.М.
(прізвище та ініціали)
«15» 12 2022 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
(прізвище та ініціали)
«16» 12 2022 р.

Вінниця ВНТУ - 2022 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Системи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
Д.т.н., проф. Яровий А.А.
14 09 **2022 року**

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Семчуку Вячеславу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі

керівник роботи к.т.н., доцент кафедри КН Колесницький О. К.
затверджені наказом вищого навчального закладу від "14" 09 2022 року №203
2. Строк подання студентом роботи 18 листопада 202_ року
3. Вихідні дані до роботи:
Вхідні дані – кількість нейронів в мережі – не менше 500, кількість шарів – не менше 2 , кількість вимовлених літер – не меенше 6, обсяг навчальної вибірки – не менше 120, обсяг тестової вибірки – не менше 60, використання об'єктно-орієнтованої мови програмування.

4. Зміст текстової частини:
Вступ, аналіз предметної області розпізнавання звукових сигналів, розробка інформаційної технології розпізнавання вимовлених літер, програмна реалізація інформаційної технології розпізнавання вимовлених літер, тестування та аналіз результатів роботи програмного забезпечення розпізнавання вимовлених літер, економічна частина, висновки, перелік використаних джерел, додатки

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень)
Алгоритм роботи програми розпізнавання вимовлених літер, структура нейронної мережі, структура інформаційної технології розпізнавання вимовлених літер, структура програми, UML діаграма класів програми, результати роботи програми розпізнавання вимовлених літер.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата завдання видав	Підпис, дата виконав прийняв
1-4	Колесницький О.К., к.т.н., доц. каф. КН	14.09.2022	12.12.2022
5	Бурєннікова Н. В., д. е. н., проф. каф. ЕПВМ	19.09.2022	12.12.2022

7. Дата видачі завдання 14.09 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз сучасного рівня інформаційних технологій розпізнавання вимовлених літер. Постановка задач дослідження	14.09.2022 – 31.10.2022
2	Побудова моделей розпізнавання вимовлених літер на основі нейронної мережі та функціонування нейронної мережі	17.10.2022 – 07.11.2022
3	Практичне застосування та оцінка ефективності розроблених моделей	03.11.2022 – 18.11.2022
4	Підготовка економічної частини	19.11.2022 – 22.11.2022
5	Апробація та/або впровадження результатів дослідження	23.11.2022 – 01.12.2022
6	Оформлення пояснювальної записки, графічного матеріалу та презентації	02.12.2022 – 14.12.2022

Студент _____ (підпис) Семчук В. О.

Керівник роботи _____ (підпис) Колесницький О.К.

АНОТАЦІЯ

УДК 004.8

Семчук В. О. Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі. Магістерська кваліфікаційна робота зі спеціальності 122 – комп'ютерні науки, освітня програма - комп'ютерні науки. Вінниця: ВНТУ, 2022. 108 с.

На укр. мові. Бібліогр.: 21 назв; рис.: 11; табл. 17.

Дана магістерська кваліфікаційна робота присвячена розробці програмного забезпечення для розпізнавання вимовлених літер на основі спайкінгової нейронної мережі. Були розглянуті та проаналізовані існуючі методи розпізнавання вимовлених літер, як найбільш перспективний, було обрано нейромережевий метод. Було проаналізовано різні парадигми штучних нейронних мереж та обґрунтовано вибір для даної задачі спайкінгової нейронної мережі. Було розроблено архітектуру обраного типу мережі. Було спроектовано програму розпізнавання вимовлених літер, написану мовою програмування Python у середовищі PyCharm з використанням фреймворку PySNN. Результати тестування довели, що розроблена програма має вищу достовірність (61,7%), ніж аналогічна програма (53,3%), а значить достовірність розпізнавання вимовлених літер покращена на 8,4%.

Графічна частина складається з 5 плакатів.

У економічному розділі визначено рівень комерційного потенціалу розробки, який становить 41,7 бала, що свідчить про комерційну важливість проведення даних досліджень. Термін окупності становить 0,71 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Ключові слова: розпізнавання, вимовлені літери, спайкінгова нейронна мережа.

ABSTRACT

Semchuk V. O. Information technology of recognition of spoken letters on the basis of spiking neural network. Master's thesis in the specialty 122 - Computer Sciences, educational program - Computer Sciences. Vinnytsia: VNTU, 2022. 108 p.

In Ukrainian language. Bibliogr .: 21 titles; fig .: 11; table 17.

This master's thesis is devoted to the development of software for recognizing spoken letters on the basis of spiky neural network. The existing methods of recognizing spoken letters were considered and analyzed, and the neural network method was chosen as the most promising. Different paradigms of artificial neural networks were analyzed and the choice of a spiking neural network for this task was substantiated. The architecture of the selected network type was developed. A spoken letter recognition program written in the Python programming language in the PyCharm environment using the PySNN framework was designed. The test results proved that the developed module has a higher reliability (61.7%) than the similar program (53.3%), which means that the accuracy is 8.4%.

The graphic part consists of 5 posters.

In the economic section, the level of the commercial development potential is determined, which is 41.7 points, which indicates the commercial importance of conducting these studies. The payback period is 0.71 years, which is less than 3 years, which indicates the commercial attractiveness of the scientific and technical development and may encourage a potential investor to finance the implementation of this development and its introduction to the market.

Key words: recognition, spoken letters, spiking neural network.

ЗМІСТ

ВСТУП

Актуальність. З появою інтелектуальних персональних помічників, які, в першу чергу покладаються на голосове введення, таких як Alexa та Google Assistant від Amazon, збільшується попит на системи розпізнавання мови. Вміння швидко та ефективно розпізнавати вимовлені літери - це перший крок у розробці такої системи. Для цієї мети можна використовувати нейронні мережі, особливо для точної класифікації приголосних звуків, які дуже схожі.

Розпізнавання мови стає все більш популярним; зростають вимоги до достовірності розпізнавання таких систем, у результаті чого зростає попит та потреба у точних і швидких системах розпізнавання мови. Використання нейронних мереж для розпізнавання мови не є новим, і минулі дослідження використовували різні нейронні мережі для своїх моделей. У цій роботі використовується спайкінгова нейронна мережа (СНМ), яка, як було встановлено, використовує імпульсне кодування і, наприклад, англійські зупинні приголосні (b, d, g, k, t, p) створюють чіткі шаблони імпульсів. Ці чіткі шаблони дозволяють ідентифікувати та класифікувати вимовлені приголосні, що може бути корисним у багатьох системах розпізнавання мовлення, таких як Alexa від Amazon, Google Assistant або Cortana від Microsoft.

Зв'язок роботи з науковими програмами, планами, темами. Магістерська робота виконана відповідно до напрямку наукових досліджень кафедри комп’ютерних наук Вінницького національного технічного університету 22 К1 «Моделі, методи, технології та пристрої інтелектуальних інформаційних систем управління, економіки, навчання та комунікацій» та плану наукової та навчально-методичної роботи кафедри.

Мета і завдання досліджень. Метою магістерської кваліфікаційної роботи є підвищення достовірності розпізнавального процесу вимовлених літер шляхом використання спайкінгової нейронної мережі.

Для досягнення мети розробки необхідно виконати такі задачі:

- провести аналіз предметної області розпізнавання вимовлених літер;

− розглянути існуючі методи розпізнавання вимовлених літер та обрати й обґрунтувати вибір методу, який задовольняє мету даної магістерської кваліфікаційної роботи;

− розробити метод вилучення ознак звукового сигналу,

− сформулювати стадії інформаційної технології, розробити структуру та алгоритм роботи програмного засобу;

− виконати програмну реалізацію запропонованої інформаційної технології;

− провести тестування програмного продукту та виконати аналіз отриманих результатів.

Об’єкт дослідження – процес комп’ютеризованого розпізнавання вимовлених літер із застосуванням інтелектуальних технологій.

Предмет дослідження – інформаційна технологія та програмні засоби розпізнавання вимовлених літер комп’ютерними засобами з використанням нейронних мереж та достовірність їх роботи.

Методи дослідження. У роботі використані наступні методи наукових досліджень: системного аналізу, теорії штучних нейронних мереж для реалізації інформаційної технології, методи математичної статистики для розробки процесу розв'язання задачі нейромережевого розпізнавання вимовлених літер та обрахунків результатів експериментів із програмним засобом, об’єктно-орієнтованого програмування.

Наукова новизна одержаних результатів.

1. Набула подальшого розвитку інформаційна технологія нейромережевого розпізнавання вимовлених літер, яка відрізняється використанням спайкінгової нейронної мережі, що дозволило підвищити достовірність програмних засобів розпізнавання вимовлених літер.

Практичне значення одержаних результатів полягає в тому, що на основі проведених досліджень розроблено програмне забезпечення розпізнавання вимовлених літер.

Запропонована інформаційна технологія сприяє підвищенню достовірності програмних засобів розпізнавання вимовлених літер, зокрема:

- розроблено алгоритм роботи програмного забезпечення розпізнавання вимовлених літер;
- розроблено програмні засоби для розпізнавання вимовлених літер.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується коректністю постановки завдання, коректністю використання математичного апарату методів дослідження, експериментальними дослідженнями тестування програмної реалізації інформаційної технології розпізнавання вимовлених літер. Адекватність розроблених математичних моделей підтверджується результатами експериментальних досліджень.

Особистий внесок здобувача. Усі результати, наведені у магістерській кваліфікаційній роботі, отримані самостійно. У працях, написаних у співавторстві, здобувачу належать: аналіз процесу розпізнавання вимовлених літер та методів підвищення достовірності програмних засобів розпізнавання вимовлених літер [1].

Апробація результатів роботи. Результати роботи були апробовані на конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)», Вінниця, 15 листопада 2022 року - 12 травня 2023 року.

Публікації. За результатами досліджень опубліковано одні тези доповіді на науково-технічній конференції [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗПІЗНАВАННЯ ЗВУКОВИХ СИГНАЛІВ

1.1 Аналіз сучасних систем розпізнавання окремих мовних одиниць

Перші розробки систем з розпізнавання мови не набагато відрізняються від сучасних, адже часто залежать від диктора, навчальної бази даних, потужності комп’ютера, розміру словника та інших характеристик. Похибка розпізнавання в ідеальних умовах не значна, але суттєво збільшується при наявності шумів та завад у звуковому сигналі.

Досліджені алгоритми орієнтовані на конкретну задачу розпізнавання мовних одиниць і не існує універсального підходу, що дозволив би розширити можливості інтелектуальної системи.

Голосове керування — це спосіб взаємодії з пристроєм за допомогою голосу. На відміну від розпізнавання мови, голосове керування призначене для введення керівних команд — наприклад, «відкрити файл», «показати погоду на завтра», «вимкнути звук». І хоча за допомогою системи голосового управління можна вводити й контент (числа й текст), таке введення буде вкрай некомфортним, оскільки оператору доведеться робити чіткі паузи між окремими словами. У наш час голосове керування мають побутові комп'ютери, автомобілі, музичні центри, кондиціонери, ліфти тощо.

На сьогоднішній день розпізнавання мовних одиниць зводиться до трьох типів задач [2]:

1. Розпізнавання окремо вимовлених літер.
2. Розпізнавання окремо вимовлених слів (команд).
3. Розпізнавання злитої мови.

Розпізнавання окремих вимовлених літер дещо простіше, ніж розпізнавання злитого тексту і не вимагає значних обчислювальних потужностей. Використовується для перетворення мови у текст. В загальному вигляді механізм розпізнавання буде виглядати так, як показано на рисунку 1.1.

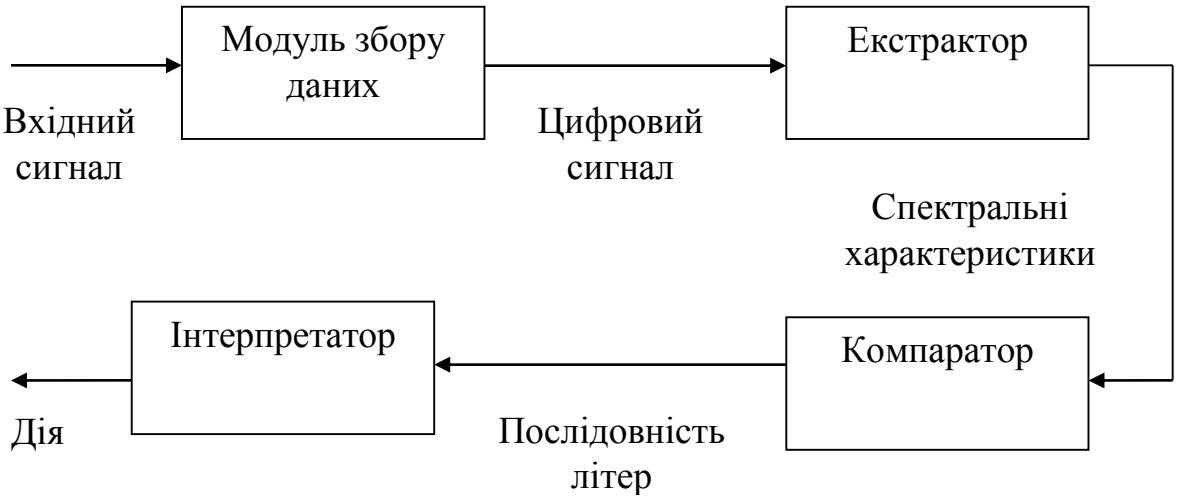


Рисунок 1.1 Механізм розпізнавання мовних одиниць

В загальному виділено 4 основних модуля при розпізнаванні вимовлених літер: модуль збору даних, екстрактор, компаратор, інтерпретатор.

Модуль збору даних включає в себе отримання вхідного сигналу і його попередню обробку, яка може включити автоматичний регулятор посилення, приглушення еха, виявлення присутності або відсутності мови і виявлення інтонаційного кінця фрази. Модуль також в себе включає виділення відрізка мови із вхідного сигналу.

Екстрактор виконує частотний аналіз сигналу. Акустично-фонетичний потік даних розбивається на короткі кадри, або вектори, тривалістю, як правило, біля 10 мс. Як правило, для кожного кадру визначається ряд інформаційних параметрів, використовуючи швидке перетворення Фур'є. Крім того можна ще використовувати й інші характеристики, наприклад, спектральні.

Компаратор здійснює акустичні порівняння: кожен кадр, або вектор, порівнюється з акустично-фонетичними зразками, які зберігаються в спеціальній базі даних. При цьому можуть порівнюватись як окремі фонеми, так і слова, і навіть фрази. При невеликій кількості слів, використовуваних диктором, більш високу надійність і швидкість можна очікувати від

розпізнавання цілих слів, але при збільшенні словника швидкість різко падає, і оптимальним стає розпізнавання окремих фонем.

Інтерпретатор – це основний інтелектуальний блок, що здійснює виконання певної дії інформаційною системою у відповідь на введену мовну команду, що була оброблена попередніми блоками механізму розпізнавання мови [2].

На кожному етапі процесу розпізнавання мовних одиниць існує багато моделей та алгоритмів реалізації, які були досліджені в рамках створення ефективного мовного інтерфейсу.

1.2 Аналіз методів автоматизованого розпізнавання мовних одиниць

Частіше за все використовується три основні методи для розпізнавання окремих мовних одиниць:

1. Алгоритм динамічної трансформації шкали часу. Використовує оптимізаційний принцип для мінімізації кількості помилок, що виникають при порівнянні розпізнаваного слова з еталонном [3].

2. Приховане Марківське моделювання. Використовує імовірнісні моделі слів. При використанні цієї технології для кожного можливого варіанта слова, яке розпізнається, обчислюється ймовірність, потім отримані ймовірності порівнюються і вибирається слово з найбільшою ймовірністю.

3. Нейронні мережі. Реалізація мовного інтерфесу за допомогою нейронних мереж набуває широкого поширення, але на даний час не розроблено алгоритмів, що могли б надавати найменші значення похибок розпізнавання та збільшити швидкість розпізнавання [4].

Розглянемо детальніше наведені алгоритми реалізації розпізнавання мовних одиниць.

Головний принцип дії як прихованих Марківських моделей, так і методу динамічної трансформації шкали часу полягає в генерації максимально правдоподібних еталонних сигналів на основі деякої автоматної граматики та

зіставлення отриманих еталонів з мовленнєвим сигналом, який розпізнається. Такий принцип обумовлює як переваги, так і недоліки цих методів. До важливої переваги генеративних методів слід віднести ефективне моделювання процесів, що нелінійно змінюються у часі, а серед недоліків можна відмітити не дуже високу дискримінанту спроможність, тобто один еталонний образ не має конкретного співставлення до введеної користувачем мовної одиниці.

До протилежного класу – дискримінативних методів – відносяться методи, що засновані на побудові меж між класами, що розпізнаються, у просторі ознак. Найбільш поширеним математичним апаратом для розробки дискримінативних методів розпізнавання є штучні нейронні мережі. Головними перевагами цього математичного апарату є те, що:

- багатошарові нейронні мережі мають високу дискримінантну спроможність;
- нейронна мережа під час навчання може знайти оптимальну комбінацію обмежень для класифікації образів, і при цьому немає необхідності у жорстких припущеннях про розподіл вхідних ознак (що необхідно, наприклад, у прихованих Марківських моделях);
- нейронна мережа у випадку розпізнавання обмеженої кількості мовних одиниць дає кращі результати, ніж метод прихованого Марківського моделювання.
- нейромережевий метод характеризується гарними швидкісними характеристиками за рахунок високого ступеню паралелізму.

До недоліків нейронних мереж можна віднести те, що за допомогою цього математичного апарату важко моделювати високу часову варіативність сигналів, що розпізнаються [5].

Порівняльна характеристика основних методів реалізації розпізнавання мовних одиниць зведена до таблиці 1.1

Таблиця 1.1 – Порівняльна характеристика основних методів

Назва методу	Швидкодія	Достовірність розпізнавання	Об’єм словника
Метод динамічної трансформації шкали часу	висока швидкодія	87-94%	більше 1000
Приховане Марківське моделювання	висока швидкодія	90-96%	більше 1000
Нейронні мережі	зі збільшенням словника швидкодія зменшується	72-98.6%	менше 100

Аналізуючи результати порівняння розглянутих методів розпізнавання мовних одиниць можна зробити висновок, що нейронні мережі є найбільш перспективними для проведення подальшого дослідження, адже використовуючи більш ефективні методи обробки звукової інформації та оптимізовані алгоритми навчання, можна досягти кращих результатів розпізнавання мовних одиниць, а саме збільшити швидкодію та покращити достовірність розпізнавання.

1.3 Обгрунтування вибору аналогу до програми розпізнавання вимовлених літер

На даний момент на ринку представлені наступні основні системи, які використовується для автоматичного розпізнавання мовних одиниць:

- Alexa (від Amazon);
- Google Assistant;
- Cortana (від Microsoft);
- IBM ViaVoice Gold.

Наведені вище програми вважаться найкращими, але ні одна із них не є ідеальною. Основні недоліки наведених вище програм полягають в:

- рівень безпомилковості розпізнавання мови не перевищує 85%;
- нерівномірна якість розпізнавання;
- низька якість розпізнавання назв і скорочених слів, повільна робота в середовищі деяких програм;
- затрата великого часу для налаштування системи.

А основним недоліком всіх систем є низька достовірність розпізнавання схожих слів, що починаються чи закінчуються приголосним звуком (літерою) [6]. Прикладами таких слів є бочка, почка, дочка, точка, кочка, ночка, мочка. Тому ключ до підвищення загальної достовірності систем розпізнавання мови лежить у підвищенні якості розпізнавання приголосних звуків, особливо таких як б, п, д, т, г, к. Саме цій задачі і присвячена дана магістерська робота.

Проблема дослідження якості розпізнавання приголосних звуків була досліджена у роботі [7], опублікованій в журналі Neurocomputing фахівцями з Центру перспективних комп'ютерних досліджень Університету Луїзіани. У цьому дослідженні автори використовували для класифікації вимовлених звуків нейронну мережу затримки часу (TDNN – Time Delay Neural Network). У результаті цього дослідження було створено програмний засіб для розпізнавання приголосних звуків, який показав середню достовірність їх розпізнавання 53,3%. Це надзвичайно низький показник, але і задача є складною, оскільки відноситься до задач розпізнавання сильно корельованих образів. Звідси і випливає мета дослідження даної магістерської роботи – підвищення середньої достовірності розпізнавання вимовлених приголосних звуків (літер).

Таким чином, порівняння методів розпізнавання мовних одиниць та дослідження існуючих праць показали, що найкращі результати дають методи на основі нейронних мереж. Наведені вище проблеми вимагають розробки оптимізованих методів та алгоритмів, які б надали можливість підвищення якості розпізнавання приголосних звуків. Саме оптимізація існуючих

алгоритмів обробки звукової інформації дасть можливість отримати максимум переваг від використання програми розпізнавання вимовлених літер, що матиме здатність розпізнавати приголосні звуки (літери) з кращими показниками швидкодії та достовірності.

1.4 Висновок до розділу 1

У першому розділі було розглянуто стан питання розпізнавання вимовлених літер. Проведено огляд відомих методів для роботи з інтелектуальними обчисленнями у сфері розпізнавання вимовлених літер. В ході аналізу предметної області розглянуто основні методи розпізнавання мовних одиниць та як найбільш перспективний, було обрано нейромережевий метод. Також було здійснено аналіз відомих програмних засобів розпізнавання вимовлених літер та як аналог до розроблюваної програми було обрано програму на основі нейронної мережі затримки часу TDNN.

2 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ РОЗПІЗНАВАННЯ ВИМОВЛЕНИХ ЛІТЕР НА ОСНОВІ СПАЙКІНГОВОЇ НЕЙРОННОЇ МЕРЕЖІ

Попередні дослідження були проведені на нейронних мережах та їх ефективності при класифікації різних звуків. В одному дослідженні розглядалася класифікація англійських стоп-приголосних за допомогою нейронної мережі затримки часу (TDNN), що дозволяє досягти непоганих результатів. В роботі буде досягнуто покращених результатів, використовуючи той самий метод вилучення особливостей, інший набір даних та спайкінгові нейронні мережі SNN.

2.1 Обгрунтування вибору спайкінгової нейронної мережі для розпізнавання вимовлених літер

Для розпізнавання мовних та звукових сигналів зараз дуже часто використовуються штучні нейронні мережі. Основним парадигмами таких мереж є багатошаровий персептрон, нейронна мережа Хемінга, радіально-базисні мережі та ін.. Однак всі ці мережі призначені для обробки вхідних сигналів, які представлені векторами чисел, тобто статичними сигналами. А мовний сигнал представляє собою динамічній сигнал, який залежить від часу і подається відліками амплітуди, взятими з певним часовим кроком. Щоб обробляти динамічний мовний сигнал на традиційних нейронних мережах, його потрібно перетворити у статичний вектор, тобто «витягнути» з нього якісь статичні ознаки. Це робиться за допомогою різних перетворень, наприклад, перетворення Фур’є. Але при цих перетвореннях втрачається корисна інформація, тому бажано використовувати такі штучні нейронні мережі, які працюють безпосередньо з динамічними сигналами без їх перетворення у вектор статичних ознак. Такими нейронними мережами є спайкінгові нейронні мережі [8,9].

Спайкінгові нейронні мережі, завдяки своїй схожості з мережами біологічних нейронів мають перед традиційними нейромережами наступні переваги:

1) можливість розпізнавання динамічних образів (динамічні зображення, мова та ін.);

2) багатозадачність (інформація про вхідні потоки циркулює в рекурентній нейронній мережі і на вихід одночасно можуть подаватись результати різних завдань за допомогою різних груп зчитувальних нейронів, навчених виконанню певного завдання). Це надзвичайно потрібна властивість для систем розпізнавання мовних одиниць;

3) розпізнавання з передбаченням (будь-який динамічний процес може бути розпізнаний навіть за неповною інформацією про нього, тобто раніше, ніж він завершиться);

4) простота процедури навчання (навчаються не всі нейрони мережі, а лише вихідні зчитувальні нейрони);

5) підвищена продуктивність обробки інформації і завадостійкість завдяки частотно-імпульсному поданню інформації.

Саме завдяки цим перевагам і було обрано спайкінгові нейронні мережі як основу розроблюваної інформаційної технології.

2.2 Розробка методу вилучення ознак звукового сигналу

Для обробки звукового сигналу спайкінговою нейронною мережею, його потрібно спочатку перетворити у вид, що сприймається вхідними нейронами СНМ. І не просто перетворити у потрібну форму (закодувати), а при цьому ще зробити попереднє перетворення звукового сигналу з метою вилучення інформативних ознак цього сигналу. При цьому корисно звернутись до біологічного принципу функціонування слухових органів людини. Будь-який звук викликає коливання у вушних стінках людини. Ці коливання призводять до збудження чи гальмування слухових рецепторів. Це призводить до

виникнення імпульсів, що обробляються центральною нервовою системою (ЦНС). Саме ЦНС розпізнає і приймає рішення про отриманий звуковий сигнал [10].

Для вирішення цієї проблема в межах розроблюваної інформаційної технології пропонується використовувати спектральний аналіз. Цей метод забезпечує кодування інформації в імпульсну форму, що найчастіше виконується окремим спайкінговим нейроном. На першій стадії із вхідного звукового сигналу формується певна кількість фреймів, що мають різну частоту, амплітуду, але мають однаковий період, що зазвичай складає долі секунди. На кожному фреймі визначається час піку амплітуди. По завершенні обробки звукового сигналу повертаються набори коефіцієнтів, що повністю відповідають інформативним ознакам вказаної мовної одиниці. Попереднє видалення шуму та нормалізація сигналу дає можливість врахувати всі завади при створенні мовної одиниці. Кожен мовець має свій тембр голосу і спосіб створення звукових сигналів, що залежить від фізіологічних особливостей людини. Метод є дуже зручним, так як не потребує великих обчислюваних ресурсів і в більшій мірі унеможливлює отримання похибок в процесі розпізнавання [11].

Детальніше зупинимось на спектральному аналізі, оскільки цей метод є одним із основних способів обробки мови в частотній області

Спектральний аналіз може бути реалізований за допомогою дискретного перетворення Фур′є. Смуги пропускання фільтрів вибираються так, щоб перекрити весь частотний діапазон мови. Середні значення модулей вихідних сигналів фільтрів будуть представляти значення спектральних коефіцієнтів у смугах.

Іноді частотний діапазон розбивають на нерівномірні смуги з урахуванням особливостей слухового сприйняття людини. Експериментально встановлено, що у внутрішньому вусі людини висота тону (частота) звукового сигналу перетворюється в механічні коливання певних ділянок базилярної мембрани. При цьому лінійним прирощуванням координати вздовж тіла

мембрани відповідають логарифмічні прирощування частоти звуку. Отже, частота звуку, що сприймається людиною, нелінійно залежить від дійсної фізичної частоти. Складний звук постійної гучності, що складається з декількох тонів, які лежать в межах критичної полоси, сприймається людиною з таким же суб′єктивним відчуттям, як одно-тональний звук, що відповідає центральній частоті критичної смуги [11].

Одним з найважливіших етапів класифікації мовлення є вибір адекватного методу вилучення ознак. Популярні методи включають отримання частотного спектра сигналу та отримання мел-кепстральних частотних коэфіцієнтів (Mel Frequency Cepstral Coefficients - MFCC) сигналу [12]. Під час вилучення характеристик сигналу у нашому випадку важливо, щоб ми виділяли фіксовану кількість об’єктів незалежно від тривалості сигналу. Це досягається шляхом розділення нашого сигналу на фіксовану кількість кадрів із накладанням 50%. Аудіо в наборі даних триває від 300 мс до 1000 мс, тому для наших цілей сигнал ділиться на 40 кадрів, в результаті чого кожен кадр триватиме від 15 мс до 50 мс. Розраховуємо довжину кадру, використовуючи таке рівняння :

$$\text{frame length} = \frac{L}{N(1-y)+y} \tag{2.1}$$

де L - довжина звукового сигналу в мілісекундах, N - кількість кадрів, на які ділиться наш сигнал, а y - перекриття зі значенням від 0 до 1. Для кожного з цих кадрів виконується витягування ознак, яке повертає 13-елементний вектор, що містить MFCC цього кадру.

Оскільки багато значень вектору ознак, повернених методом MFCC, менше або дорівнюють 0, то потрібна функція, яка масштабуватиме ці значення таким чином, щоб можна було отримати відповідний вхідний струм, який призведе до генерації імпульсів нашими вхідними нейронами СНМ. Наступне рівняння \- це функція для відповідного масштабування наших значень:

$$I_{inj} = \begin{cases} 8.5 & x = 0 \\ .05309x + 8.5 & x < 0 \\ .11806x + 8.5 & x > 0 \end{cases} \quad (2.2)$$

Коли амплітуда ознаки дорівнює 0, вхідний нейрон спрацьовуватиме зі швидкістю 20 Гц. Коли амплітуда ознаки менше 0, швидкість спрацьовування вхідного нейрона буде нижче 20 Гц, але більше 2 Гц. Нарешті, коли амплітуда ознаки більше 0, швидкість генерації вхідного нейрона буде більше 20 Гц, але не перевищуватиме 38 Гц.

2.3 Розробка структури спайкінгової нейронної мережі для розпізнавання вимовлених літер

На рисунку 2.1 показана абстрактна модель спайкінгової нейронної мережі у вигляді автомата з «плаваючими» станами. Як витікає з назви, ця модель має деяку схожість з кінцевим автоматом, але більш універсальна, і її «плаваючий» високорозмірний аналоговий стан x(t) змінюється безперервно в часі.

Формально, такий автомат M складається з фільтра L^M (тобто функція, що відображає вхідні потоки $\boldsymbol{u}(\cdot)$ в потоки $\boldsymbol{x}(\cdot)$, причому $\boldsymbol{x}(t)$ може залежати не лише від $\boldsymbol{u}(t)$, але і абсолютно довільно нелінійно від попередніх входів $\boldsymbol{u}(s)$: $\boldsymbol{x}(t)=L^M(\boldsymbol{u}(t))$), і з функції зчитування без запам'ятовування f^M, що відображає у будь-який момент часу t вихід фільтра $\boldsymbol{x}(t)$ («плаваючий стан») в деякий цільовий вихід $\boldsymbol{y}(t)$. Взагалі, такий автомат реалізує фільтр, який відображає $\boldsymbol{u}(\cdot)$ в $\boldsymbol{y}(\cdot)$.

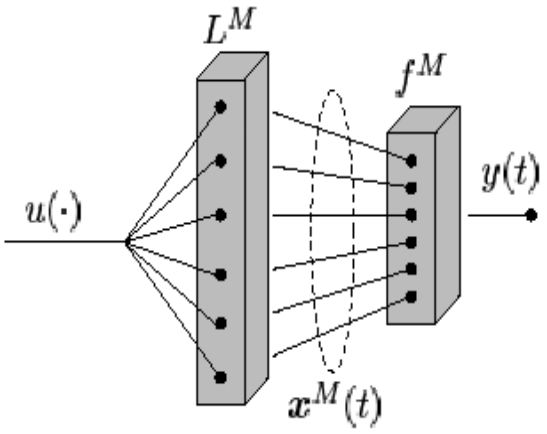


Рисунок 2.1 - Абстрактна модель спайкінгової нейронної мережі

Наведемо функцію, що реалізує вихідний фільтр:

$$y(t) = w \cdot x(t) + \omega_0 \tag{2.3}$$

де x – стан нейронної мережі; w – матриця ваг ; y – вихідне значення нейрона.

Структура нейронної мережі вимагає більш глибокого дослідження. Кількість входів та виходів мережі визначаються кількістю вхідних та вихідних параметрів досліджуваного об'єкта, явища, процесу, тощо. Число нейронів обирається емпіричним шляхом [13,14].

Більшість дослідників та інженерів, застосовуючи архітектуру до визначених проблем використовують загальні правила, зокрема:

1. Якщо складність у відношенні між отриманими та бажаними даними на виході збільшується, кількість нейронів прихованого прошарку повинна також збільшитись.

2. Якщо процес, що моделюється, може розділятись на багато етапів, потрібен додатковий прихований прошарок. Якщо процес не розділяється на етапи, тоді додаткові прошарки можуть допустити перезапам'ятовування і, відповідно, невірне загальне рішення.

Архітектура мережі.

У нашому випадку мережа складається з двох шарів спайкінгових нейронів Іжикевича [15], з'єднаних тренованими синапсами, як показано на рис. 2.2.

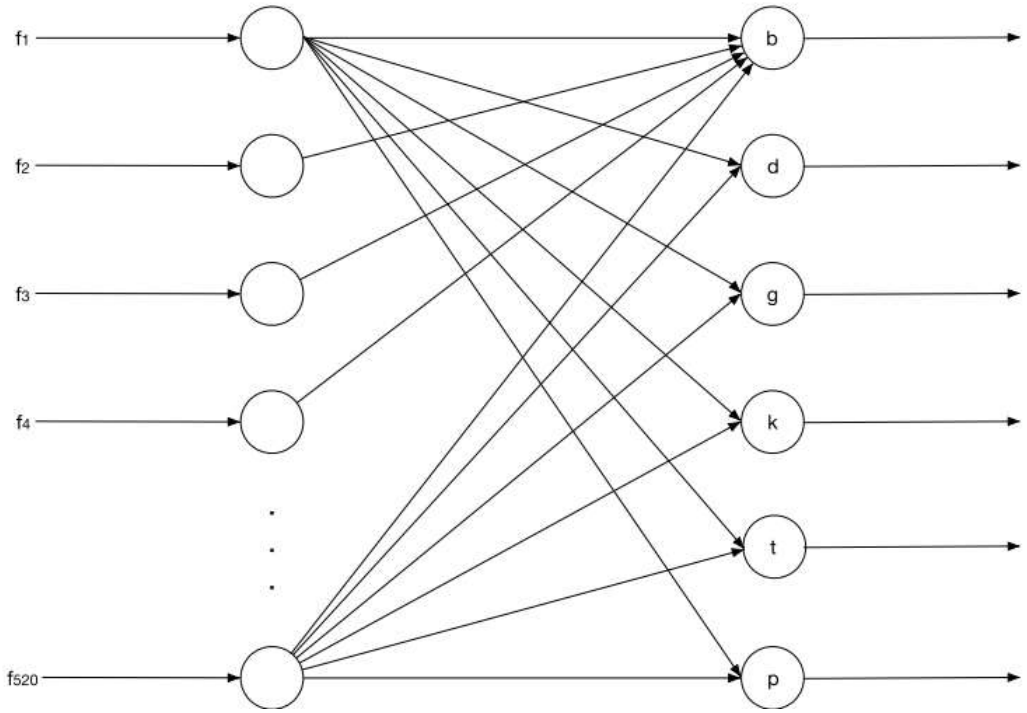


Рисунок 2.2 – Структура спайкінгової нейронної мережі для розпізнавання вимовлених літер

Перший шар мережі - це вхідний шар, що складається з 520 вхідних нейронів, кожен нейрон отримує значення амплітуди однієї ознаки як свій струм інжекції. Вихідний шар складається з 6 нейронів, що відповідають кожному з 6 приголосних звуків (літер). Вихідні нейрони містять ту саму конфігурацію параметрів, що і нейрони вхідного шару. Кожен вхідний нейрон підключений до кожного з вихідних нейронів через тренований синапс, в результаті чого кожен вихідний нейрон має 520 вхідних синапсів. Синапси були ініціалізовані з вагами, рівними 0,5. Навчання для цих синапсів описується рівнянням (2,6). Кожен з вихідних нейронів генерує унікальний спайк-шлейф на основі сигнальних характеристик, що надаються вхідному шару. Спайкові

шлейфи аналізуються і використовуються для класифікації приголосної, яка потрапляла в мережу, згідно з рівняннями (2.7) – (2.10).

У розроблюваній спайкінговій нейронній мережі використано модель спайкінгового нейрона Іжикевича [15] для генерації послідовності спайків.

$$\frac{dV}{dt} = 0.04V^2 + 5V + 140 - u + I_{inj}$$

$$\frac{du}{dt} = a(bV - u) \qquad (2/4)$$

де I_{inj} - це інжектований струм і він розраховується за допомогою рівняння (2.2).

Параметри, які використовувалися для моделі нейрону Іжикевича, наведені в таблиці 2.1.

Таблиця 2.1 – Параметри моделі нейрона Іжикевича

Параметер	Значення
a	0.02
b	0.2
c	-65
d	8
u	-13
V_{peak}	30

Рівняння скидання:

$$V > V_{peak}: \begin{cases} V = c \\ u = u + d \end{cases} \qquad (2/5)$$

Для навчання спайкінгової нейронної мережі використовується метод STDP (spike-timing dependent plasticity) – спайкінг-часово залежна пластичність

[16]. Нарешті, існує сигнал "вчителя", який визначає, які нейрони та їх синапси проходять Геббівську та анти-Геббівську спайкінг-часово залежну пластичність (STDP). Для певного звукового сигналу (літери) відповідний вихідний нейрон зазнає Геббівського STDP, тоді як інші проходять анти-Геббівський STDP.

2.4 Навчання спайкінгової нейронної мережі

Кожен з вхідних нейронів підключений до кожного з вихідних нейронів за допомогою синапсів, які пройдуть навчання за методом спайкінг-часово залежної пластичності (spike-timing dependent plasticity – STDP) [16]. Кожен синапс має функцію, яка моделює його провідність, залежну від часу, коли він отримує один вхідний імпульс в момент часу t:

$$I(t) = w * t * e^{-t/\tau} \tag{2.6}$$

де w - вага синапсу, яка є величиною, що коригується під час навчання. τ - константа часу, яка представляє час, коли синапс досягає максимальної провідності. Це моделює провідність лише одного синапсу протягом усього періоду експерименту. Щоб генерувати загальну синаптичну провідність 520 вхідних синапсів, підсумовуємо всі синапси та кожен їх вхідний імпульс (спайк) відповідно до рівняння нижче:

$$I_{total} = \sum_{k=1}^{520} \sum_{j=1}^{N_k} w * (t - t_{kj}) * e^{-\frac{t-t_{kj}}{\tau}} \tag{2.7}$$

де t_{kj} - час, коли синапс k отримує спайк j, а N_k позначає кількість спайків, отриманих синапсом k. Потім це значення поширюється на вихідний нейрон як вхідний струм. Існує два типи навчання STDP, які ми використовуємо; Геббівський STDP та анти-Геббівський STDP. Лише синапси, пов'язані з

бажаним вихідним нейроном, будуть проходити Геббівський STDP. Усі інші синапси будуть проходити анти-Геббівський STDP. Рівняння (2.8) охоплює Геббівський STDP:

$$\Delta w_{ij} = \begin{cases} 0.1e^{\frac{-(|t_j - t_i|)}{\tau}} & t_j - t_i \geq 0 \\ 0.1e^{\frac{-(|t_j - t_i|)}{\tau}} & t_j - t_i < 0 \end{cases} \quad (2.8)$$

Як для Геббівського STDP, так і для анти-Геббівського STDP є два випадки: довготривале потенціювання (long-term potentiation - LTP) та довготривала депрессія (LT depression - LTD). У Геббівському STDP, якщо постсинаптичний спайк генерується після пресинаптичного спайку, то можна визначити, що пресинаптичний спайк має прямий вплив на швидкість генерації спайку нейроном. Як результат, синаптична вага між цими двома нейронами збільшиться (LTP). З іншого боку, якщо вихідний нейрон генерує спайк до того, як він отримає пресинаптичний спайк, то синаптична вага між двома нейронами зменшується (LTD). Для анти-Геббівського STDP випадки міняються місцями, так що синапс зазнає LTD, коли різниця в часі між до- та постсинаптичними спайками є позитивною, а в протилежному випадку зазнає LTP.

Після того, як мережа пройде навчання, ми подаємо в мережу по одному із приголосних звуків та генеруємо базовий імпульсний паттерн спайків (спайк-шлейф), з яким будуть порівнюватись наші спайк-шлейфи тестових звуків. Наприклад, ми подамо в мережу звук «б» та отримаємо базовий спайк-шлейф, виданий з вихідного нейрона «б», і це буде нашим прототипом сигналу звуку «б». Це буде виконуватися для кожної з приголосних, поки не отримаємо базові спайк-шлейфи для кожного прототипу приголосного звуку (літери). Невелика частина набору даних, які не були використані для навчання чи генерації прототипів, будуть використані для тестування. Щоб провести класифікацію літер, порівняємо отримані спайк-шлейфи із відповідними спайк-шлейфами

прототипів, використовуючи синхронізацію спайків, розраховану за рівняннями (2.9) – (2.12) нижче. Формуємо показники збігу для кожної пари спайк-шлейфів (n, m), використовуючи рівняння (2.9):

$$C_i^{(n,m)} \begin{cases} 1 & if\ min_j \left(\left| t_i^{(n)} - t_j^{(m)} \right| \right) < \tau_{ij}^{(n,m)} \\ 0 & otherwise \end{cases} \tag{2.9}$$

де τ_{ij} - величина збігів спайків у 2 послідовностях, яка визначається як:

$$\tau_{ij} = \frac{\min\left\{ t_{i+1}^x - t_i^x, t_i^x - t_{i-1}^x, t_{j+1}^y - t_j^y, t_j^y - t_{j-1}^y \right\}}{2} \tag{2.10}$$

де x та y відповідають двом спайк-шлейфам, що порівнюються, а t_i та t_j відповідають часу генерації спайків spike i та spike j відповідно. Потім ми генеруємо нормалізований лічильник збігів для кожного спайка кожної послідовності:

$$C_i^{(n)} = \frac{1}{N-1} \sum_{m \neq n} C_i^{(n,m)} \tag{2.11}$$

де N - кількість показників збігів за участю спайк-шлейфу n.

Потім ми усереднюємо лічильники збігів загальної кількості спайків із загальної кількості спайк-шлейфів, використовуючи рівняння нижче:

$$S_C = \frac{1}{M} \sum_{k=1}^{M} C(t_k) \tag{2.12}$$

де M - загальна кількість спайків при об'єднанні всіх спайк-шлейфів. Значення, отримане з цього рівняння, буде мірою того, наскільки схожі два спайкових шлейфи з вищим значенням, що відповідає більшій подібності, і низьким значенням, що відповідає більшій несхожості.

2.5 Структура інформаційної технології розпізнавання вимовлених літер на основі спайкінгової нейронної мережі

Структура інформаційної технології розпізнавання вимовлених літер на основі спайкінгової нейронної мережі, детально описана у попередніх параграфах, зображена на рис. 2.3.

Рисунок 2.3 - Структура інформаційної технології розпізнавання вимовлених літер на основі спайкінгової нейронної мережі

Із рис. 2.3 видно, що в основі інформаційної технології лежить класифікація спайк-шлейфів приголосних звуків (вимовлених літер), яка виконується спайкінговою нейронною мережею. Для роботи інформаційної технології розпізнавання вимовлених літер на основі спайкінгової нейронної мережі вхідною є інформація про вимовлені літери у аудіофайлах формату .wav. Для обробки цих сигналів потрібно спочатку сформувати їх ознаки. Для вилучення ознак використовують мел-кепстральні коефіцієти. Для цього розбивають аудіофрагмент на 40 фреймів та для кожного фрейму знаходять 13-значний мел-кепстральний коефіцієт. Таким чином, отримують вектор із 520 чисел (13x40=520).

Далі відбувається процес ініціалізації (створення) спайкінгової нейронної мережі з 520 вхідними нейронами та 6 вихідними нейронами (рис. 2.2). Потім здійснюється процес навчання цієї спайкінгової нейронної мережі за методом STDP на навчальній вибірці, що формується на основі бази даних вимовлених різними дикторами літер. Після того, як спайкінгова нейронна мережа навчена, її можна використовувати для розпізнавання вимовлених літер. Для цього на вхідні нейрони мережі подаються сигнали у вигляді струмів інжекції, величина яких пропорційна мел-кепстральним коефіцієнтам. Вхідні нейрони перетворюють ці струми у імпульси пропорційної частоти [17]. Ці імпульси проходять через синаптичні ваги і вихідні нейрони, на виході яких формуються так звані спайк-шлейфи (образи) вимовленої літери. Спайк-шлейф являє собою кілька імпульсів на інтервалі часу тривалості звучання літери, певним чином розташовані у часі, тобто для кожної літери мають оригінальні моменти появи. Далі відбувається процес детектування ступеню співпадіння спайк-шлейфів на вихідах нейромережі з прототипами спайк-шлейфів 6 літер. Процес виведення результату розпізнавання вимовленої літери полягає у визначенні виходу нейромережі, який найбільше збігається з еталонним спайк-шлейфом одної із 6 літер.

Таким чином, розроблена структура інформаційної технології розпізнавання вимовлених літер на основі спайкінгової нейронної мережі може бути використана для подальшої розробки програмних засобів.

2.6 Розробка алгоритму роботи програмного забезпечення

На основі всього вищесказаного було розроблено схему алгоритму роботи програмного забезпечення розпізнавання вимовлених літер на основі спайкінгової нейронної мережі, яка представлена на рис. 2.4

Після запуску програми потрібно обрати один із двох можливих режимів роботи нейронної мережі – тренування (навчання) або тестування (функціонування). Якщо мережа перед цим не була навчена, то потрібно спочатку обрати режим навчання.

Рисунок 2.4 – Схема алгоритму роботи програми

В режимі навчання спочатку завантажується навчальна вибірка, яка являє собою 120 файлів формату .wav (по 20 звукових файлів на кожну з 6 приголосних літер б, п, д, т, г, к). Потім ініціалізується спайкінгова нейронна мережа, яка складається з 520 вхідних нейронів та 6 вихідних нейронів. Тобто матриця синапсів має розмір 520x6. Далі відбувається процес навчання мережі. Алгоритм навчання – спайкінг-часово залежна пластичність (spike-timing dependent plasticity – STDP). Навчання припиняється при зменшенні похибки до наперед заданого значення. Після цього виводиться час навчання, а ваги мережі зберігаються.

В режимі тестування на вхід мережі може бути поданий будь-який .wav файл із тестової вибірки. Тестова вибірка не повинна мати файлів, які увійшли до навчальної вибірки. Вхідний .wav файл розбивається на 40 фреймів, для кожного з яких формуються 13-значні мел-кепстральні коефіцієнти, які масштабуються і перетворюються у струми збудження вхідних нейронів мережі. Ці струми збудження викликають генерацію спайків пропорційної частоти на виходах вхідних нейронів. Ці спайки далі проходять через натреновані синапси на входи вихідних нейронів. На виходах вихідних нейронів формуються послідовності імпульсів, які ми називаємо спайк-шлейфами і які є оригінальними для кожної вимовленої літери. Далі ці спайк-шлейфи перевіряються на співпадіння із прототипами спайк-шлейфів літер і для кожного виходу мережі формується число в діапазоні 0…1, яке вказує ступінь відповідності тестового сигналу прототипу вимовленої літери. Вихідний нейрон, на якому зафіксовано максимальний ступінь співпадіння і визначає вимовлену літеру.

UML діаграма класів програми представлена на рис. 2.5.

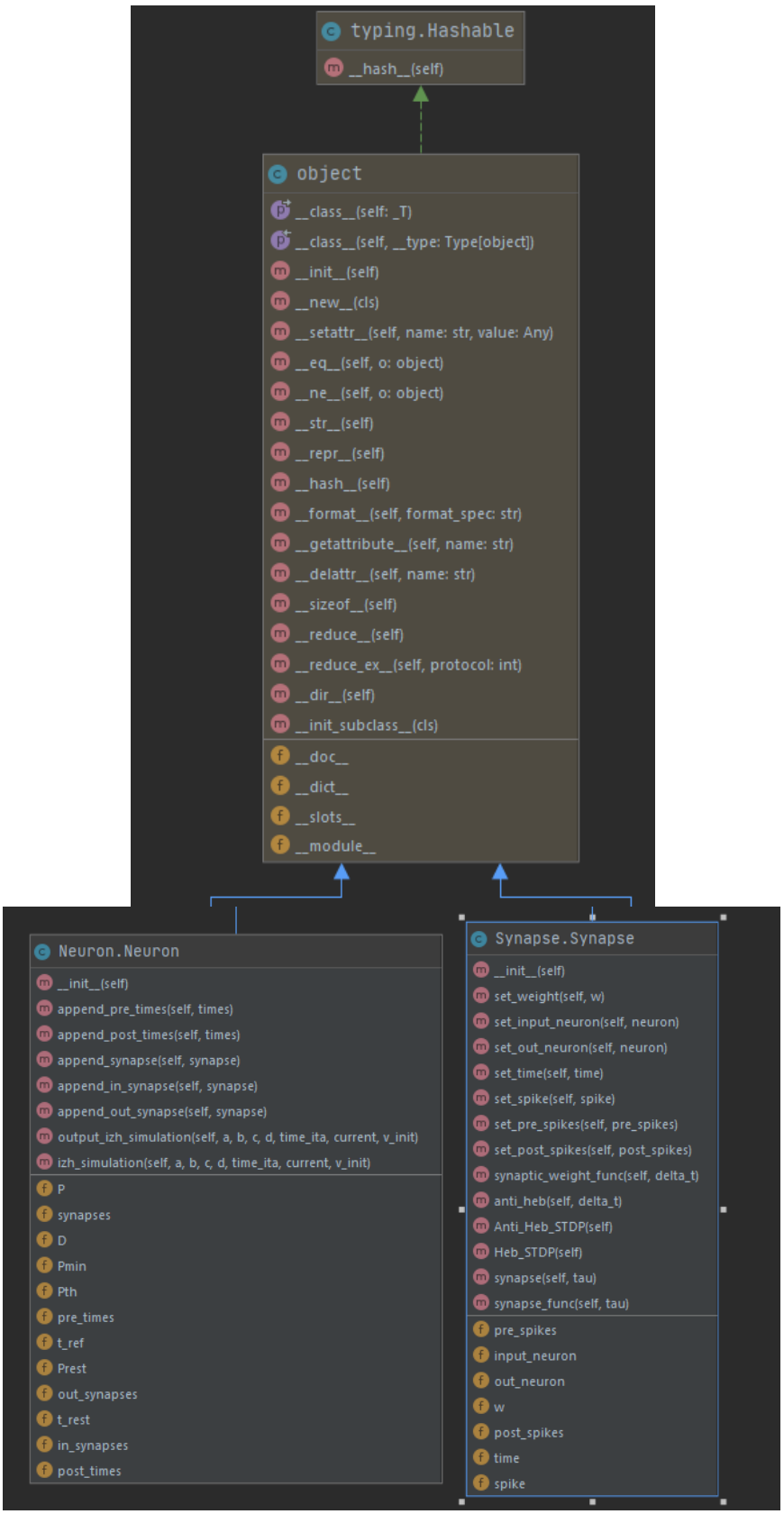


Рисунок 2.5 – UML діаграма класів програми розпізнавання вимовлених літер

Програма містить два основних класи Neuron і Synapse, який використовується першим.

2.7 Висновок до розділу 2

Отже, дослідивши відомі нейронні мережі, для інформаційної технології розпізнавання вимовлених літер, як основа, була обрана спайкінгова нейронна мережа. Розроблено основні принципи обробки звукових сигналів за допомогою спектрального аналізу та визначення коефіцієнтів спектрального потоку, що дає можливість визначити інформативні ознаки окремої вимовленої літери. Визначено структуру спайкінгової нейронної мережі, метод навчання нейронної мережі та принципи ідентифікації відповідної вимовленої літери. Було розроблено структуру інформаційної технології розпізнавання вимовлених літер з використанням спайкінгової нейронної мережі, на основі якої розроблено алгоритм роботи та UML діаграму програми розпізнавання вимовлених літер.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ РОЗПІЗНАВАННЯ ВИМОВЛЕНИХ ЛІТЕР НА ОСНОВІ СПАЙКІНГОВОЇ НЕЙРОННОЇ МЕРЕЖІ

3.1 Обгрунтування вибору мови та середовища програмування

Процес реалізації будь-якого програмного додатку можна значно полегшити, якщо обрати правильні інструментальні засоби розробки: мова програмування, середовище розробки, фреймворки та бібліотеки.

Вибір мови програмування залежить не тільки від її можливостей і сфери застосування, але і від великої кількості інших факторів, таких як: тип ліцензії, наявність зручних середовищ програмування, об’єм спільноти, кількість та якість відповідних бібліотек та фреймворків.

Для програмної реалізації модуля розпізнавання вимовлених літер на основі спайкінгової нейронної мережі було обрано мову програмування Python.

Python – це інтерпретована об'єктно-орієнтована мова програмування високого рівня з динамічною типізацією. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання існуючих компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована [18].

Основною перевагою Python є велика кількість бібліотек та фреймворків для наукових досліджень в області машинного навчання, зокрема – в області нейронних мереж. Також Python має ефективні структури даних високого рівня та простий, але ефективний підхід до обєктно-орієнтованого програмування,

що робить код лаконічним та простим для сприйняття, порівняно, наприклад, з C++. Елегантний синтаксис Python, динамічна обробка типів, а також те, що це інтерпретована мова, роблять її ідеальною для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ. Інтерпретатор мови Python може бути розширений функціями та типами даних, розробленими на C чи C++ (або на іншій мові, яку можна викликати із C).

В якості середовища програмування для Python було обрано IDE PyCharm, що має ряд зручностей, таких як: підсвічування синтаксису, автодоповнення, менеджер пакетів, інтеграція з системами контроля версій і т. д. Також це середовище програмування має студентський тип ліцензії, що робить його використання безкоштовним для студентів.

Для програмної реалізації процесів створення та навчання спайкінгової нейронної мережі було використано бібліотеки PySNN [19].

PySNN – це фреймворк спайкінгових нейронних мереж (СНМ), написаний поверх PyTorch для ефективного моделювання СНМ як на звичайному процесорі (CPU), так і на графічному процесорі (GPU). Фреймворк використовує методи навчання на основі кореляції. Бібліотека дотримується високомодульного та динамічного дизайну PyTorch і не вимагає від свого користувача вивчення нового фреймворку.

Потужність цього фреймворку полягає у простоті визначення та змішування нових об’єктів Neuron та Connection, які безперебійно працюють разом, навіть різних версій, в одній мережі.

PySNN призначений для забезпечення в основному об'єктів низького рівня для свого користувача, які можна комбінувати та змішувати, як і в PyTorch. Найбільша різниця полягає в тому, що мережа тепер складається з двох типів модулів, а не з єдиного nn.Module в звичайному PyTorch. Ці нові модулі - це pysnn.Neuron та pysnn.Connection.

3.2 Структура програми розпізнавання вимовлених літер на основі спайкінгової нейронної мережі

Структура програми розпізнавання вимовлених літер на основі спайкінгової нейронної мережі представлена на рис. 3.1. Основні блоки структури:

1) Формувач мел-кепстральних коефіцієнтів;
2) Спайкінгова нейромережа;
3) Детектор співпадінь.

Крім блоків, також потрібні інформаційні ресурси:

1) Навчальна вибірка;
2) Тестова вибірка;
3) Прототипи спайк-шлейфів.

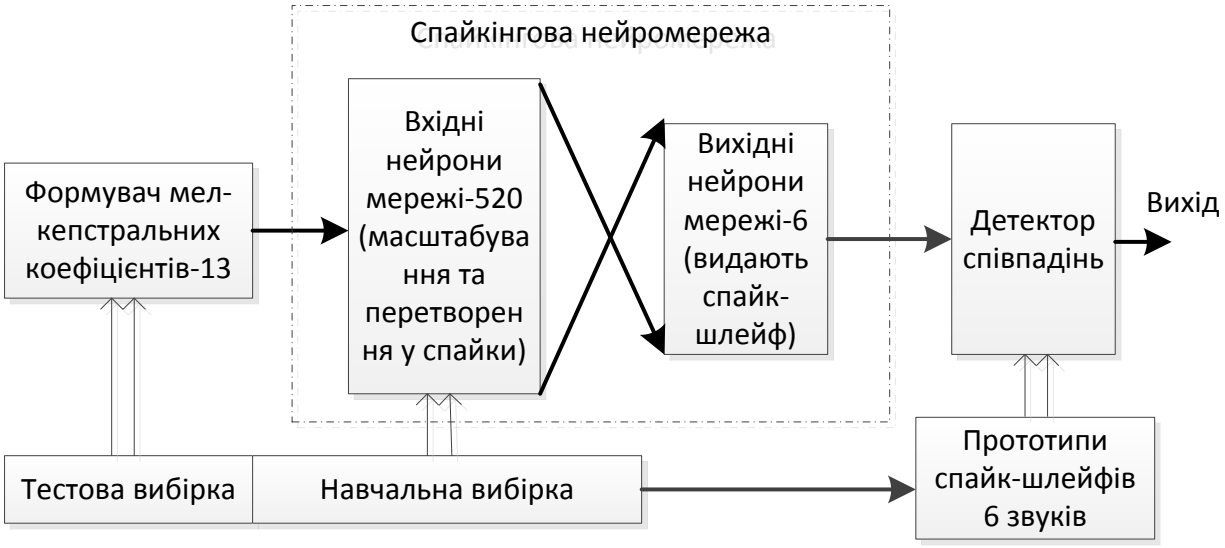


Рисунок 3.1 - Структура програми розпізнавання вимовлених літер на основі спайкінгової нейронної мережі

Формувач мел-кепстральних коефіцієнтів розбиває звуковий файл тривалістю 0,5 сек на 40 фреймів і для кожного фрейму формує 13 мел-кепстральних коефіцієнтів. Всього отримується 520 чисел, які перетворюються вхідними нейронами мережі у частоту від 2 до 38 Гц.

В режимі навчання спочатку завантажується навчальна вибірка, яка являє собою 120 файлів формату .wav (по 20 звукових файлів на кожну з 6 приголосних літер б, п, д, т, г, к). Потім ініціалізується спайкінгова нейронна мережа, яка складається з 520 вхідних нейронів та 6 вихідних нейронів. Тобто матриця синапсів має розмір 520x6. Далі відбувається процес навчання мережі. Алгоритм навчання – STDP, а ваги навченої мережі зберігаються.

В режимі тестування на вхід мережі може бути поданий будь-який .wav файл із тестової вибірки. Для цього файлу формуються мел-кепстральні коефіцієнти, які масштабуються і перетворюються у спайки (імпульси) вхідними нейронами. Ці спайки далі надходять на вихідні нейрони через натреновані синапси. На виходах вихідних нейронів формуються послідовності імпульсів, які ми називаємо спайк-шлейфами і які є оригінальними для кожної вимовленої літери. Далі ці спайк-шлейфи перевіряються детектором співпадінь на співпадіння із прототипами спайк-шлейфів літер і для кожного виходу мережі формується число в діапазоні 0…1, яке вказує ступінь відповідності тестового сигналу відповідному прототипу вимовленої літери.

Підготовка даних.

Дані, якими навчалася і тестувалася модель, були отримані від Консорціуму лінгвістичних даних Університету Пенсільванії. База даних, з якої отримані дані, - це база даних ISOLET Spoken Letter Database, яка містила 7800 вимовлених літер, відібраних із 150 ораторів, половина з яких – чоловіки, а інша половина - жінки, записаних у форматі .wav. Щоб перетворити вихідний аудіосигнал у придатний для використання формат для спайкінгової нейронної мережі, було виділено мель-кепстральні частотні коефіцієнти (MFCC), які використовувались як ознаки вимовлених літер. Спочатку мережа навчалася на 300 зразках, по 50 зразків на кожну приголосну (б, д, г, к, п, т). Це призвело до того, що деякі вихідні нейрони, зокрема нейрон, асоційований із звуком «г», не генерував жодних спайків незалежно від вхідного сигналу. Отже, було зменшено навчальний зразок до 120 зразків з 20 зразками для кожної приголосної літери.

Генерація спайків.

Для генерації вхідних спайків використано модель спайкінгового нейрона Іжикевича для генерації послідовності спайків, пов’язаної із заданим вхідним струмом.

Кожен з вихідних нейронів отримує вхідні дані від кожного з 520 вхідних нейронів через синапс. В результаті, інжектований струм для вихідних нейронів визначається інакше, ніж для вхідних нейронів. Рівняння (2.6) і (2.7) використовуються для визначення вхідного струму.

3.2 Програмна реалізація інформаційної технології розпізнавання вимовлених літер на основі спайкінгової нейронної мережі

Інсталяцію PySNN можна виконати за допомогою pip:

```
$ pip install pysnn
```

Якщо потрібно оновити бібліотеку без необхідності її перевстановлювати, замість цього скористайтеся такою командою встановлення:

```
$ git clone https://github.com/BasBuller/PySNN.git
$ cd PySNN/
$ pip install -e .
```

Деякі приклади потребують додаткових бібліотек. Щоб встановити їх, запустіть:

```
$ pip install pysnn[examples]
```

Для встановлення PySNN потрібна версія Python 3.6 або новіша, Python 2 не підтримується. PySNN також вимагає, щоб PyTorch був версії 1.2 або новішої.

Структура мережі. Намір полягає в тому, щоб відобразити більшу частину структури фреймворку PyTorch. Як приклад, наступний фрагмент коду показує, наскільки визначення Spiking Neural Network у PySNN схоже на визначення мережі в PyTorch. Графік мережі є циклічним завдяки зворотному зв’язку від вихідних нейронів до прихованих нейронів.

```
    class Network(SNNNetwork):
        def __init__(self):
            super(Network, self).__init__()

            # Input
            self.input = Input((batch_size, 1, n_in), *input_dynamics)

            # Layer 1
            self.mlp1_c = Linear(n_in, n_hidden, *connection_dynamics)
            self.neuron1 = LIFNeuron((batch_size, 1, n_hidden),
*neuron_dynamics)
            self.add_layer("fc1", self.mlp1_c, self.neuron1)

            # Layer 2
            self.mlp2_c = Linear(n_hidden, n_out, *connection_dynamics)
            self.neuron2 = LIFNeuron((batch_size, 1, n_out),
*neuron_dynamics)
            self.add_layer("fc2", self.mlp2_c, self.neuron2)

            # Feedback connection from neuron 2 to neuron 1
            self.mlp2_prev = Linear(n_out, n_hidden, *c_dynamics)
            self.add_layer("fc2_back", self.mlp2_prev, self.neuron1)

        def forward(self, input):
            spikes, trace = self.input(input)

            # Layer 1
            x_prev, _ = self.mlp2_prev(self.neuron2.spikes,
self.neuron2.trace)
```

```
        x_forw, _ = self.mlp1_c(x, t)
        x, t = self.neuron1([x_forw, x_rec, x_prev])

        # Layer out
        spikes, trace = self.mlp2_c(spikes, trace)
        spikes, trace = self.neuron2(spikes, trace)

        return x
```

Визначення мережі. Загальна структура визначення мережі така сама, як і в PyTorch, де це можливо. Усі нові визначені об’єкти успадковують клас nn.Module. Найбільші відмінності полягають у наступному:

– кожен рівень складається з об’єктів Connection і Neuron, оскільки вони обидва реалізують різну динаміку на основі часу,

– навчання не використовує градієнти,

– нейрони мають стан, який зберігається між послідовними часовими кроками,

– мережі успадковуються від спеціального класу pysnn.SNNNetwork.

Нейрони. Цей об’єкт є основною відмінністю від традиційних штучних нейронних мереж. Нейрони мають дуже нелінійну (а також недиференційовану) поведінку. Вони мають внутрішню напругу, коли вона перевищує порогове значення, нейрон генерує двійковий сплеск (недиференційована операція), який потім поширюється на наступний рівень нейронів через об’єкт Connection. Визначення нового класу нейронів досить просте, потрібно лише визначити нові функції динаміки нейронів для напруги та траси нейрона. Допоміжні функції (майже) всі визначені в базовому класі Neuron.

Зв'язки. Цей об’єкт містить ваги з’єднань і маршрутизує сигнали між різними рівнями. Він дійсно відрізняється від шарів PyTorch лише тим, що він має стан між ітераціями своєї минулої активності та можливістю затримки передачі сигналу між шарами.

Форми з'єднання

Щоб відстежувати траєкторії та затримки в тензорах, що передають інформацію, потрібен додатковий вимір порівняно з принципами PyTorch. Завдяки додаванню траєкторій спайків кожен тензор спайків містить додатковий вимір траєкторій як останній вимір. Отримане впорядкування вимірів виглядає наступним чином: для тензора зображення (траєкторія позначається як R, щоб не плутати з часом для відеоданих):

```
[batch size, channels, height, width, traces] (B,C,H,W,R)
```

Для повнозв’язаних шарів результуючий тензор має такий вигляд (вільний розмір можна використовувати так само, як і в PyTorch):

```
[batch size, free dimension, input elements, traces] (B,F,I,R)
```

Наразі явна тривимірна згортка неможлива, як зазвичай при обробці відео. На щастя, спайкінгові нейронні мережі мають вбудований часовий вимір і (на даний момент все ще теоретично) добре підходять для обробки відео подія за подією, тому не потребують тривимірної згортки.

Траєкторії з`єднань нейронів. Траєкторії зберігаються в об’єктах Neuron і Connection. Наразі об’єкти Connection беруть траєкторії від своїх пресинаптичних нейронів і поширюють траєкторію з часом, тобто не виконують жодної подальшої обробки траєкторій. Якщо потрібно, можна реалізувати окрему обробку траєкторій у спеціальному об’єкті Connection.

Траси зберігаються в тензорі в кожному Connection, а також затримка для кожної траєкторії, що поширюється через Connection. Через кожен синапс можна відстежити лише одну траєкторію (або сигнал). Якщо час затримки через синапс стає дуже довгим (довшим за рефрактерний період пресинаптичного нейрона), новий сигнал може потрапити в Connection до того, як через нього пройде попередній. У поточній реалізації старий сигнал буде

перезаписано, тобто інформація буде втрачена до того, як її можна буде використати.

Користувач повинен переконатися, що рефрактерні періоди будуть такими ж або довшими, ніж синаптична затримка в наступному з'єднанні.

Визначення модулів. При проектуванні потрібно переконатися, що кожен модуль має метод self.reset_state(). Він викликається з класу SNNNetwork і потрібен для правильного моделювання кількох входів.

Для представлення спайкінгової нейронної мережі на мові Python з використанням фреймворку PySNN застосовуються такі оператори, функції та класи.

Клас нейрон:

```
class Neuron:
def __init__(self):
self.Pth = Pth
self.t_ref = 4
self.t_rest = -1
self.P = Prest
self.D = D
self.Pmin = Pmin
self.Prest = Prest
self.post_times = []
self.pre_times = []
self.synapses = []
self.in_synapses = []
self.out_synapses = []
```

Функція моделювання спайкінгового нейрона Іжикевича:

```
def izh_simulation(self, a, b, c, d, time_ita, current, v_init):
# a,b,c,d parameters for Izhikevich model
# time_ita time iterations for euler method
```

```
# current list of current for each time step
# v_init initial voltage
spike_times = []
v = v_init
u = v * b
v_plt = np.zeros(time_ita)
u_plt = np.zeros(time_ita)
spike = np.zeros(time_ita)
num_spikes = 0
tstep = 0.1 # ms
ita = 0
while ita < time_ita:
v_plt[ita] = v
u_plt[ita] = u
v += tstep * (0.04 * (v ** 2) + 5 * v + 140 - u +
current[ita])
u += tstep * a * (b * v - u)
if v > 30.:
spike[ita] = 1
v = c
u += d
num_spikes += 1
#spike_times.append(ita)
ita += 1
time = np.arange(time_ita) * tstep
i = 0
for t in time:
if spike[i] == 1:
spike_times.append(t)
i += 1
return time, v_plt, spike, num_spikes, spike_times
```

Клас синапс.

```
class Synapse:
global spike, time, conductance_amplitude, w, spikes_received,
pre_spikes, post_spikes, input_neuron, out_neuron, voltage
def __init__(self):
self.post_spikes = []
self.pre_spikes = []
self.w = .5
def set_weight(self, w):
self.w = float(w)
def set_voltage(self, voltage):
self.voltage = voltage
def set_input_neuron(self, neuron):
self.input_neuron = neuron
def set_out_neuron(self, neuron):
self.out_neuron = neuron
```

Функція навчання Геббівського STDP:

```
# Our W function for Hebbian STDP
def synaptic_weight_func(self, delta_t):
tau_pre = 20
tau_post = 20
Apre = .10
Apost = -Apre
if delta_t >= 0:
return Apre*np.exp(-np.abs(delta_t)/tau_pre)
if delta_t < 0:
return Apost*np.exp(-np.abs(delta_t)/tau_post)
```

Функція навчання анти-Геббівського STDP:

```
# Our W function for anti-Hebbian STDP
def anti_heb(self, delta_t):
tau_pre = 20
```

```
tau_post = 20
Apre = .10
Apost = -Apre
if delta_t < 0:
return Apre*np.exp(-np.abs(delta_t)/tau_pre)
if delta_t >= 0:
return Apost*np.exp(-np.abs(delta_t)/tau_post)
```

Клас Network:

```
class Network:
def __init__(self, weights):
self.a = 0.02
self.b = 0.2
self.c = -65.
self.d = 8.
self.time_ita = 5000 # 100ms
# Build a layer of 120 input neurons (20 frames, 6 features for
each frame)
self.input_layer = []
for i in range(520):
n = Neuron.Neuron()
self.input_layer.append(n)
# Build output layer, one neuron for each letter of the alphabet
self.output_layer = []
for i in range(6):
o = Neuron.Neuron()
self.output_layer.append(o)
```

Таким чином, була виконана програмна реалізація модуля розпізнавання вимовлених літер на основі спайкінгової нейронної мережі. Лістінг програми наведено у додатку Б.

3.4 Висновок до розділу 2

У розділі було обгрунтовано вибір мови програмування Python, середовища розробки PyCharm та спеціалізованої бібліотеки для створення та навчання спайкінгових нейронних мереж PySNN. Розроблено структуру програми розпізнавання вимовлених літер на основі спайкінгової нейронної мережі та описано особливості її функціонування.. Здійснено програмну реалізація інформаційної технології розпізнавання вимовлених літер на основі спайкінгової нейронної мережі.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ РОЗПІЗНАВАННЯ ВИМОВЛЕНИХ ЛІТЕР НА ОСНОВІ СПАЙКІНГОВОЇ НЕЙРОННОЇ МЕРЕЖІ

4.1 Тестування програми розпізнавання вимовлених літер

Для використання програми розпізнавання вимовлених літер на основі спайкінгової нейронної мережі на комп'ютері має бути встановлений Python 3.8+ версії та пакети, які необхідні для запуску програми (якщо спробувати запустити програму, то пакети, які не встановлені, будуть вказані в терміналі – командою pip install name (і тоді їх потрібно встановити). Також додатково має бути встановлений компілятор C++.

Дані, які використовувались для навчання та тестування розробленої програми розпізнавання вимовлених літер, були отримані від Консорціуму лінгвістичних даних Університету Пенсільванії. Це база даних окремо вимовлених літер (ISOLET Spoken Letter Database), яка містить 7800 вимовлених літер, відібраних від 150 ораторів, половина з яких чоловіки і половина — жінки. Ці вимовлені літери записані у файли формату .wav. Щоб перетворити необроблений аудіосигнал із файлу формату .wav у придатний для спайкінгової нейронної мережі формат, ми витягли із аудіосигналу мел-частотні кепстральні коефіцієнти (MFCC) і використали їх значення як ознаки вимовленої літери. Спочатку наша нейромережа була навчена на 300 зразках, по 50 зразків для кожної із 6 літер. Було виявлено, що це призвело до того, що деякі вихідні нейрони, зокрема нейрон, пов'язаний зі звуком «г», не генерували жодних імпульсів незалежно від вхідного сигналу. Отже, навчальну вибірку було зменшено до 120 зразків (по 20 зразків для кожної із 6 літер).

Тренування:

Для тренування потрібні вхідні дані (тобто аудіофайли у форматі .wav, де кожен файл – це окрема вимовлена літера (приклад: -Б-.wav)), їх потрібно помістити за шляхом ../letter_audio/speech/isolet1 або ../letter_audio/speech/isolet2

папках (див. рис.4.1), які знаходяться в кореневому каталозі проекту (якщо папок немає, то потрібно їх створити).

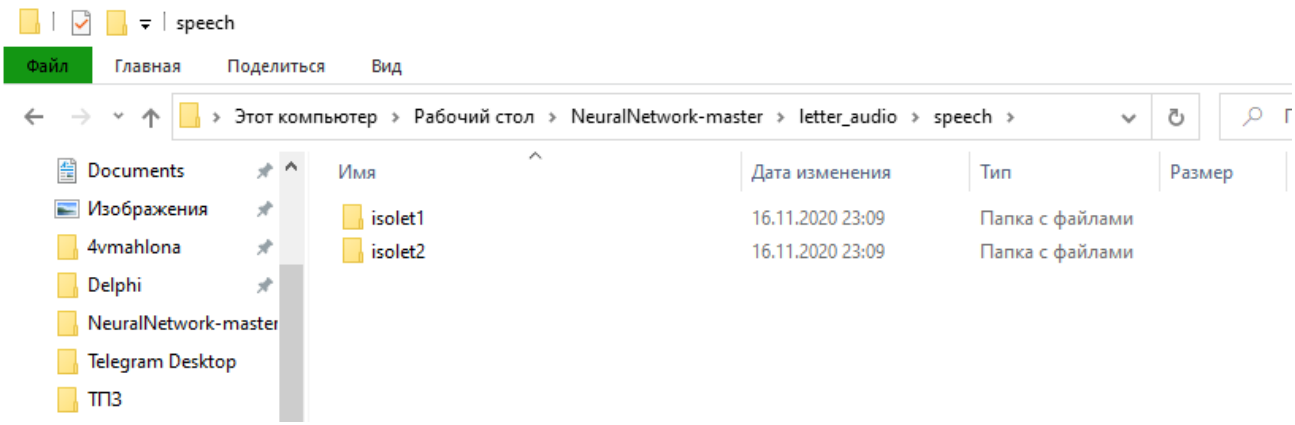


Рисунок 4.1 – Створення папок для навчальних даних

- Перейти до каталогу, що містить SNN.py
- Ввести та запустити команду python SNN.py train в терміналі

```
C:\Users\maglo\AppData\Local\Programs\Python\Python38\python.exe C:/Users/maglo/Desktop/NeuralNetwork-master/SNN.py train
```

В результаті отримаємо час запуску навчання та час закінчення. Також буде виведено результати у вигляді кількості спайків для кожного з нейронів кожної букви та ваги нашої створеної мережі.

```
C:\Users\maglo\AppData\Local\Programs\Python\Python38\python.exe C:/Users/maglo/Desktop/NeuralNetwork-master/SNN.py train
Training...
2020-11-17 18:07:19.017517
letter_audio/speech/isolet1\-A-.wav
    A: 5
2020-11-17 18:13:20.076336

Process finished with exit code 0
```

Тестування:

Для тестування потрібні вхідні дані (тобто аудіофайли у форматі **.wav**, де кожен файл – це окрема вимовлена буква (приклад: **-B-.wav**)), їх потрібно помістити за шляхом ../letter_audio/speech/isolet3 папку (див. рис.4.2), яка

знаходиться в кореневому каталозі проекту (якщо папки немає, то потрібно її створити).

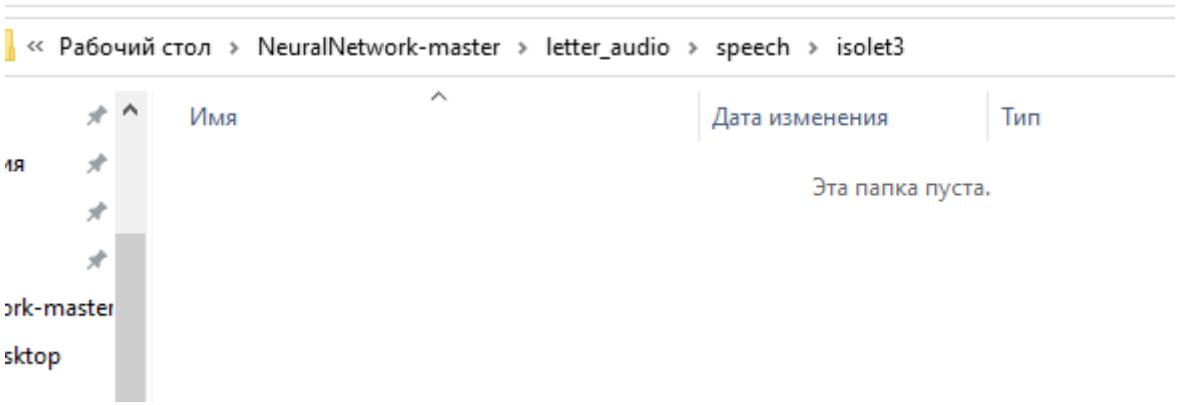


Рисунок 4.2 – Створення папки для тестових даних

- Перейти до каталогу, що містить SNN.py
- Ввести та запустити команду python SNN.py test в терміналі

```
C:\Users\maglo\AppData\Local\Programs\Python\Python38\python.exe C:/Users/maglo/Desktop/NeuralNetwork-master/SNN.py test
```

Отримуємо результати у вигляді кількості спайків для кожного з нейронів кожної літери та аналіз спайків (індекс та відстань).

```
C:\Users\maglo\AppData\Local\Programs\Python\Python38\python.exe C:/Users/maglo/Desktop/NeuralNetwork-master/SNN.py test
Testing
letter_audio/speech/isolet3\-A-.wav
    A: 58

Process finished with exit code 0
```

Тестування розробленої програми показало її надійну роботу у різних режимах та повну відповідність завданню.

4.2 Аналіз результатів роботи програми розпізнавання вимовлених літер

Після тренування в спайкінгову нейронну мережу було подано сигнал для кожної із приголосних літер і отримано відповідні послідовності спайків для

кожної літери. Ці спайкові послідовності були тими, з якими можна було б порівняти наші результати випробувань. Після кожного тесту кожен з вихідних нейронів генерував спайкові шлейфи на основі заданого вхідного сигналу. Кожен спайковий шлейф порівнювали зі своїм відповідним прототипом, використовуючи інтервали співпадіння спайків, як описано у рівняннях (2.9) – (2.12). Ми фіксуємо „співпадіння” коли тестова послідовність спайків має найбільшу схожість із відповідною еталонною (прототипною) послідовністю спайків.

Для тестування спайкінгової нейронної мережі та отримання результатів було використано частину набору даних окремо вимовлених літер, яка не використовувалася для навчання мережі чи генерації прототипів,. Кожна приголосна була протестована загалом 10 разів, і середнє співпадіння між тестовими спайк-шлейфами і прототипами спайк-шлейфів наведено у таблиці 4.1. І прототипування, і тестування проводились протягом 500 мс.

Таблиця 4.1 - Середнє співпадіння між тестовими спайк-шлейфами і прототипами спайк-шлейфів

Середнє співпадіння між тестовим сигналом і прототипом						
	Тестовий звук (літера)					
Вихідний нейрон	Б	Д	Г	К	П	Т
Б	0,93	0,64	0,84	0,68	0,74	0,73
Д	0,86	0,92	0,83	0,71	0,77	0,71
Г	0,76	0,7	0,94	0,63	0,72	0,63
К	0,73	0,76	0,69	0,91	0,83	0,77
П	0,67	0,69	0,80	0,66	0,92	0,87
Т	0,72	0,76	0,84	0,70	0,71	0,90

У табл. 4.1 наведено порівняння між прототипами послідовностей спайків і відповідними послідовностями спайків на вихідному нейроні відповідного звуку. Наприклад, коли вхідним сигналом є приголосна «Б», вихідний нейрон

«Д» має в середньому 86% схожості з прототипом спйк-шлейфу приголосної «Д» (виділена комірка у табл. 4.1).

Розробленій мережі вдалося створити чіткі спайк-шлейфи для кожної з вимовлених приголосних, як це видно на рис. 4.3 і 4.4.

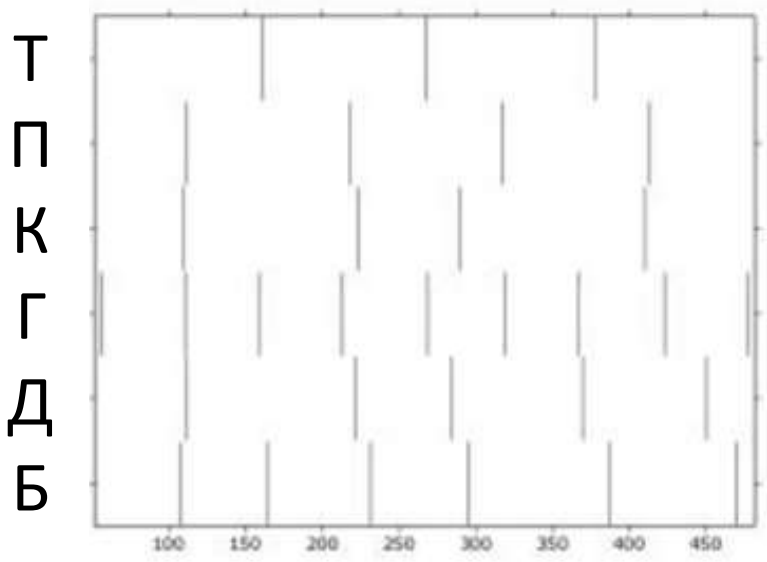


Рисунок 4.3 - Прототипи спайк-шлейфів для кожної з вимовлених приголосних у вікні 500 мс

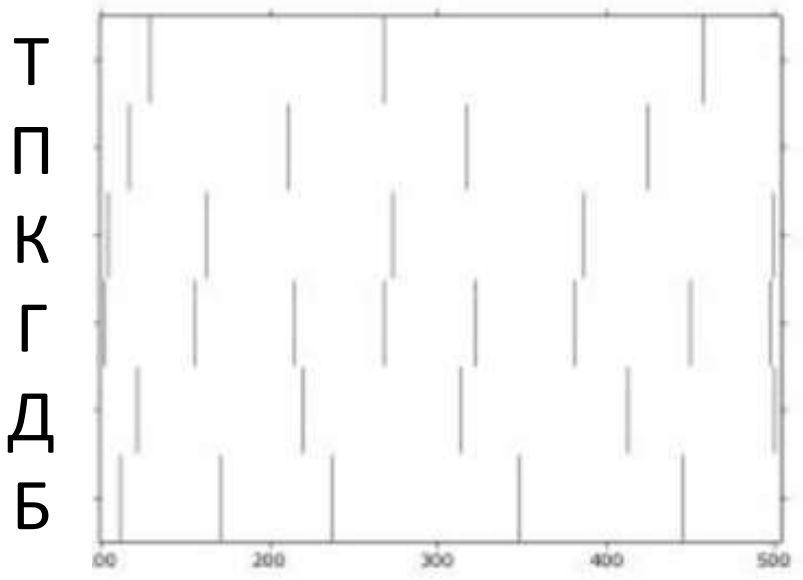


Рисунок 4.4 - Тестові спайк-шлейфи для кожної з вимовлених приголосних у вікні 500 мс

Однак результати, наведені в таблиці 4.1, дозволяють стверджувати, що спайк-шлейфи одних літер дуже подібні (від 0,63 до 0,87) до прототипів інших літер. І це зрозуміло, тому що звучання різних приголосних літер дійсно є схожим.

Наприклад, у табл. 4.2 показано результати одного тесту для кожної приголосної (співпадіння між тестовим спайк-шлейфом та прототипом спайк-шлейфу).

Таблиця 4.2 - Результати одного тесту для кожної приголосної (співпадіння між тестовим спайк-шлейфом та прототипом спайк-шлейфу)

Одиничний тест						
	Довільно вибраний тест (літера)					
Вихідний нейрон	Б	Д	Г	К	П	Т
Б	**0.99**	0.73	0.94	0.75	**0.99**	0.67
Д	0.91	**0.98**	0.94	0.86	0.67	**0.99**
Г	**0.99**	0.8	**0.98**	0.75	0.75	0.4
К	0.91	0.8	0.94	**0.98**	**0.99**	0.67
П	0.83	0.8	0.88	0.86	0.89	**0.99**
Т	0.91	0.89	0.94	0.86	**0.99**	0.8

Перевіримо достовірність роботи розробленої програми у порівнянні с аналогом [20]. Для тестування візьмемо вибірку звуків обсягом 60 (по 10 звуків на кожну із 6 літер).

Достовірність розпізнавання вимовлених літер розробленої програми та аналога наведена у табл. 4.3.

Із табл. 4.3 видно, що розроблена програма має вищу достовірність розпізнавання вимовлених літер (61,7%), ніж аналогічна програма (53,3%), а значить достовірність розпізнавання вимовлених літер покращена на 8,4% за рахунок використання спайкінгової нейронної мережі, тобто мета роботи досягнута.

Таблиця 4.3 - Достовірність розпізнавання вимовлених літер розробленою програмою у порівнянні з аналогом

	Достовірність розпізнавання, %						Середня достовірність
	Тестовий звук (літера)						
	Б	Д	Г	К	П	Т	
Аналог	50	60	40	50	70	50	53,3%
Розроблена програма	60	70	50	60	70	60	61,7%

Тестування розробленої програми розпізнавання вимовлених літер на основі спайкінгової нейронної мережі показало її надійну роботу. Програма повністю відповідає завданню. Інструкція користувача наведена у Додатку Г.

4.3 Висновок до розділу 4

У розділі було описано процес тестування розробленої програми розпізнавання вимовлених літер. Були наведені основні результати роботи розробленого програмного засобу, проведені експерименти над ним та проаналізовані результати даного тестування, що дозволило довести правильність роботи програми та визначили її перевагу перед аналогом у достовірності розпізнавання вимовлених літер. Розроблена програма має вищу достовірність розпізнавання вимовлених літер (61,7%), ніж аналогічна програма (53,3%), а значить достовірність розпізнавання вимовлених літер покращена на 8,4% за рахунок використання спайкінгової нейронної мережі, тобто мета роботи досягнута.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота з розробки та дослідження «Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

1) проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розраховано витрати на здійснення науково-технічної розробки;

3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Інформаційна технологія розпізнавання вимовлених літер на основі

спайкінгової нейронної мережі» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1 [20].

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 5.1

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуютьс я у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці 5.2.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	5	5	5
2. Ринкові переваги (наявність аналогів)	3	3	4
3. Ринкові переваги (ціна продукту)	4	4	4
4. Ринкові переваги (технічні властивості)	3	4	3
5. Ринкові переваги (експлуатаційні витрати)	0	0	0
6. Ринкові перспективи (розмір ринку)	3	2	3
7. Ринкові перспективи (конкуренція)	3	2	3
8. Практична здійсненність (наявність фахівців)	4	4	3
9. Практична здійсненність (наявність фінансів)	3	3	3
10. Практична здійсненність (необхідність нових матеріалів)	5	5	5
11. Практична здійсненність (термін реалізації)	5	4	4
12. Практична здійсненність (розробка документів)	5	4	5
Сума балів	43	40	42
Середньоарифметична сума балів $СБ_c$	41,7		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3 [20].

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі» становить 41,7 бала, що, відповідно до таблиці

5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

5.2 Оцінювання рівня новизни розробки

Виводячи на ринок новинку виробник вважає, що тієї новизни, якою наділена нова розробка є достатньо для того, щоб вона була сприйнята споживачем як нова. Але це не завжди так, в силу того, що споживач і виробник неоднозначно визначать її рівень новизни. Тому доцільним є визначення рівня новизни розробки отриманої в результаті досліджень за темою «Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі».

Саме визначення рівня і ступеня інтегральної новизни є найбільш актуальним, оскільки її рівень визначає ступінь однакового позитивного сприйняття новизни розробки як виробником, так і споживачем, а отже і ринком в цілому, а це, у свою чергу, є гарантією того, що новинка знайде своє місце на ринку, користуватиметься попитом у споживачів і забезпечить відшкодування витрат, зазнаних товаровиробником під час розроблення та виробництва технічної розробки [21].

Рівень новизни нової продукції розраховуємо експертним методом шляхом протиставлення нової продукції та її аналогів, що існують в даний час на ринку, за чинниками що визначають її значення, в системі «краще-гірше». Рівень новизни встановлюємо відносно рівня аналога (або продукту, що досить близький до аналога).

Для визначення *i*-го виду новизни, застосуємо чинники, які впливають на її рівень. Кожен чинник *i*-го виду новизни розраховуємо в балах. Більша кількість набраних балів свідчить про більший рівень новизни. Для оцінювання рівня новизни використаємо думки експертів, які встановлюють визначені бали відповідним чинникам. Бал відповідності проставляється в діапазоні від (-5 –

значно гірше аналога до +5 – значно краще аналога). Результати попереднього оцінювання зведемо до відповідного листа оцінювання (таблиця 5.4).

Таблиця 5.4 – Лист оцінювання рівня новизни експертами

Види та чинники		Бали та експерти		
		Експерт 1	Експерт 2	Експерт 3
1		*2*	*3*	*4*
Споживча новизна	Питома вага 0,24	Максимальний бал $Б_{i\ MAX}$		25
1. Зміна поведінкових звичок споживача		4	4	4
2. Ступінь задоволення потреб і запитів		5	5	5
3. Спосіб задоволення потреби		3	3	4
4. Формування нової потреби		4	4	4
5. Формування нового споживача		1	1	1
Середній бал експертів $Б_{i\ отр}$		17		
Товарна новизна	Питома вага 0,202	Максимальний бал $Б_{i\ MAX}$		30
1. Параметричні зміни показників продукції				
1.1. Якісні		4	4	4
1.2. Технічні		4	4	3
1.3. Економічні		3	3	3
1.4. Сервісні		4	4	4
2. Якість продукції по відношенню до конкурентів		3	3	3
3. Функціональні зміни		4	4	4
Середній бал експертів $Б_{i\ отр}$		22		
Виробнича новизна	Питома вага 0,042	Максимальний бал $Б_{i\ MAX}$		25
1. Рівень унікальності товару для підприємства		5	5	5
2. Рівень унікальності для галузі		3	4	3
3. Рівень унікальності товару для країни		1	1	1
4. Зміна виробничої системи		4	4	4
5. Відносно існуючого асортименту		2	2	2
Середній бал експертів $Б_{i\ отр}$		15		
Прогресивна новизна	Питома вага 0,2	Максимальний бал $Б_{i\ MAX}$		25
1. Зміна технології виготовлення		4	4	4
2. Рівень застосування нових компонентів і матеріалів		1	2	1
3. Зміна технологічного принципу дії виробу		1	2	1
4. Зміна конструктивного виконання		3	2	3
5. Рівень застосування інновацій		2	2	2
Середній бал експертів $Б_{i\ отр}$		11		

Продовження таблиці 5.4

Види та чинники		Бали та експерти		
		Експерт 1	Експерт 2	Експерт 3
1		*2*	*3*	*4*
Ринкова новизна	Питома вага 0,1	Максимальний бал $Б_{i\ MAX}$		20
1. Новий виріб на новому ринку		0	0	0
2. Новий виріб на відомому ринку		4	4	4
3. Модернізований виріб		2	2	2
4. Нова модель		3	2	2
Середній бал експертів $Б_{i\ omp}$		8		
Екологічна новизна	Питома вага 0,035	Максимальний бал $Б_{i\ MAX}$		20
1. Рівень екологічної чистоти технології виробництва		5	5	5
2. Рівень впровадження мало- та безвідходних технологій		5	5	5
3. Рівень екологічно небезпечних режимів експлуатації продукції		5	5	5
4. Рівень забруднення навколишнього середовища		5	5	5
Середній бал експертів $Б_{i\ omp}$		20		
Соціальна новизна	Питома вага 0,036	Максимальний бал $Б_{i\ MAX}$		20
1. Використання нового товару приводить до покращення стану здоров'я нації		0	0	0
2. Використання нового товару приводить до зростання доходів населення		0	0	0
3. Виробництво нового товару приводить до збільшення (зменшення) кількості робочих місць на підприємстві		4	5	4
4. Виробництво нового товару приводить до підвищення кваліфікації персоналу		3	3	3
Середній бал експертів $Б_{i\ omp}$		7		
Маркетингова новизна	Питома вага 0,145	Максимальний бал $Б_{i\ MAX}$		20
1. Нові методи маркетингових досліджень		0	0	0
2. Вживання нових стратегій сегментації ринку		3	3	3
3. Вибір нової маркетингової стратегії обхвату і розвитку цільового сегмента		2	3	2
4. Побудова нових каналів збуту		0	1	1
Середній бал експертів $Б_{i\ omp}$		6		

Значення *і*-го виду новизни розрахуємо за формулою [21]:

$$I_i = \frac{Б_{i\,omp}}{Б_{i\,MAX}}, \qquad (5.1)$$

де $Б_{i\ отр}$ – отримана кількість балів за шкалою оцінок чинників, що визначають *i*-й вид новизни;

$Б_{i\ max}$ – максимальна кількість балів, що може бути отримана за *i*-м видом новизни.

Загальний рівень інтегральної новизни розраховуємо шляхом перемноження отриманого значення *i*-го виду новизни на її вагомість, причому вагомість *i*-го виду новизни визначаємо експертним методом, за формулою [21]:

$$N_{інт} = \sum_{i}^{n} W_i \cdot I_i \ , \tag{5.2}$$

де $N_{інт}$ – рівень інтегральної (сукупної) новизни;

W_i – вагомість (питома вага) *i*-го виду новизни;

n – загальна кількість видів новизни.

$$N_{інт} = (0{,}24 \cdot 17/25)+(0{,}202 \cdot 22/30)+(0{,}042 \cdot 15/25)+(0{,}2 \cdot 11/25)+(0{,}1 \cdot 8/20)+$$
$$(0{,}035 \cdot 20/20)+(0{,}036 \cdot 7/20)+(0{,}145 \cdot 6/20) = 0{,}562.$$

Отримане значення інтегрального рівня новизни зіставляємо зі шкалою, що наведена в табл. 5.5 [20].

Таблиця 5.5 – Рівні новизни нового товару та їхня характеристика

Рівні новизни товару	Значення інтегральної новизни	Характеристика товару	Вид нового товару
Найвища	1,00	Абсолютно новий товар	Новий товар, що наділений ознаками інноваційності (інноваційний товар)
Висока	0,8…0,99	Товар, який не має аналогів	
Значуща	0,6…0,79	Принципова зміна споживчих властивостей товару	
Достатня	0,4…0,59	Принципова технологічна модифікація товару	
Незначна	0,2…0,39	Кардинальна зміна параметрів	Новий товар
Помилкова	0,00…0,19	Малоістотна модифікація	

Згідно таблиці 5.5 розробка відповідає рівню при значенні інтегральної новизни 0,562 - достатня новизна; за характеристикою: принципова технологічна модифікація товару; вид розробки - новий товар, що наділений ознаками інноваційності (інноваційний товар).

5.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов'язані з проведенням науково-дослідної роботи на тему «Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.3.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [20]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \qquad (5.3)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

T_p – середнє число робочих днів в місяці, T_p=21 дні.

$$З_о = 15000{,}00 \cdot 63 / 21 = 45000{,}00 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці 5.6.

Таблиця 5.6 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
1. Керівник роботи	15000,00	714,29	63	45000,00
2. Консультант (логопед)	11500,00	547,62	5	2738,10
3. Інженер-програміст 1-ї категорії	14600,00	695,24	54	37542,86
4. Лаборант	7000,00	333,33	40	13333,33
Всього				98614,29

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i , \qquad (5.4)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} , \qquad (5.5)$$

де $М_М$ – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо *Мм*=6700,00 грн;

$К_і$ – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [20];

$К_с$ – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$Т_р$ – середнє число робочих днів в місяці, приблизно $Т_р$ = 21 дн;

$t_{зм}$ – тривалість зміни, год.

$С_1 = 6700{,}00 \cdot 1{,}10 \cdot 1{,}7 / (21 \cdot 8) = 74{,}58$ *грн.*

$З_{р1} = 74{,}58 \cdot 6{,}00 = 447{,}46$ *грн.*

Таблиця 5.7 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Установка обчислювального обладнання для проведення досліджень	6,00	2	1,10	74,58	447,46
Підготовка робочого місця інженера-програміста	4,50	4	1,50	101,70	457,63
Встановлення аудіоблоків	5,00	5	1,70	115,26	576,28
Інсталяція програмного забезпечення для моделювання та розробки	5,00	4	1,50	101,70	508,48
Формування бази даних	25,00	3	1,35	91,53	2288,17
Всього					4278,03

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.6)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 10%.

$З_{дод}$ = (98614,29 + 4278,03) · 10 / 100% = 10289,23 грн.

5.3.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.7)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (98614,29 + 4278,03 + 10289,23) · 22 / 100% = 24899,94 грн.

5.3.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою «Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі».

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (5.8)$$

де H_j – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу j-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j = 1,1 \ldots 1,15$);

$В_j$ – маса відходів j-го найменування, кг;

$Ц_{вj}$ – вартість відходів j-го найменування, грн/кг.

$М_1 = 2,0 \cdot 279,00 \cdot 1,1 - 0,000 \cdot 0,00 = 613,80$ *грн.*

Проведені розрахунки зведемо до таблиці 5.8.

Таблиця 5.8 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір FAXE-500 A4	279,00	2,0	-	-	613,80
Папір для записів AXE 70 A5-250	125,00	5,0	-	-	687,50
Органайзер офісний OFFICE 30/50	220,00	2,0	-	-	484,00
Набір офісний DATA XOD	205,00	3,0	-	-	676,50
Картридж для принтера	2100,00	1,0	-	-	2310,00
Диск оптичний OPTIMA CD	22,50	3,0	-	-	74,25
Пам'ять GOODRAM 32ГБ C10A	420,00	1,0	-	-	462,00
Всього					5308,05

5.3.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі», розраховуємо, згідно з їхньою номенклатурою, за формулою:

$$К_в = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j \quad (5.9)$$

Де $Н_j$ – кількість комплектуючих j-го виду, шт.;

$Ц_j$ – покупна ціна комплектуючих j-го виду, грн;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 ... 1,15*).

$К_в = 1 \cdot 1850{,}00 \cdot 1{,}1 = 2035{,}00$ *грн.*

Проведені розрахунки зведемо до таблиці 5.9.

Таблиця 5.9 – Витрати на комплектуючі

Найменування комплектуючих	Кількість, шт.	Ціна за штуку, грн	Сума, грн
Зовнішня звукова карта Behringer U-PHORIA UMC404HD	1	1850,00	2035,00
Мікрофон Behringer C1	1	250,00	275,00
Всього			2310,00

5.3.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення. Витрати на придбання спецустаткування необхідного для проведення досліджень відсутні.

5.3.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (5.10)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань програмних засобів.

$B_{прг} = 8410,00 \cdot 1 \cdot 1,11 = 9335,10$ *грн.*

Отримані результати зведемо до таблиці 5.10:

Таблиця 5.10 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
ОС Windows	1	8410,00	9335,10
Прикладний пакет Microsoft Office	1	7770,00	8624,70
Прикладний пакет обробки та аналізу звукових сигналів	1	7560,00	8391,60
Прикладний пакет мови програмування Python у середовищі PyCharm	1	4800,00	5328,00
Фреймворк PySNN	1	1320,00	1465,20
Всього			33144,60

5.3.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12} , \qquad (5.11)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (27450{,}00 \cdot 3) / (2 \cdot 12) = 3431{,}25$ *грн.*

Проведені розрахунки зведемо до таблиці 5.11.

Таблиця 5.11 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Персональний комп'ютер проектувальника EVEREST E5000	27450,00	2	3	3431,25
Обчислювальна система проектування	22650,00	2	3	2831,25
Робоче місце інженера-програміста	8620,00	5	3	431,00
Пристрій виводу інформації	6750,00	5	3	337,50
Оргтехніка	9800,00	5	3	98,00
Приміщення лабораторії	324000,00	20	3	4050,00
Всього				11179,00

5.3.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію ($B_е$) розраховуємо за формулою:

$$B_е = \sum_{i=1}^{n} \frac{W_{уі} \cdot t_i \cdot Ц_е \cdot K_{вп i}}{\eta_i}, \qquad (5.12)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e$ = 6,20 грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0{,}32 \cdot 480{,}0 \cdot 6{,}20 \cdot 0{,}95 / 0{,}97 = 952{,}32$ *грн.*

Проведені розрахунки зведемо до таблиці 5.12.

Таблиця 5.12 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний комп'ютер проектувальника EVEREST E5000	0,32	480,0	952,32
Обчислювальна система проектування	0,12	400,0	297,60
Робоче місце інженера-програміста	0,08	400,0	198,40
Пристрій виводу інформації	0,65	9,0	36,27
Оргтехніка	0,75	11,0	51,15
Зовнішня звукова карта Behringer U-PHORIA UMC404HD	0,02	200,0	24,80
Всього			1560,54

5.3.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи на тему «Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також

витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (5.13)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 25%.

$B_{св}$ = (98614,29 + 4278,03) · 25 / 100% = 25723,08 грн.

5.3.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (5.14)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 42%.

$B_{сп}$ = (98614,29 + 4278,03) · 42 / 100% = 43214,77 грн.

5.3.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.15)$$

Де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 80%.

$І_в$ = (98614,29 + 4278,03) · 80 / 100% = 82313,85 грн.

5.3.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.16)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 140%.

$В_{нзв}$ = (98614,29 + 4278,03) · 140 / 100% = 144049,24 грн.

Витрати на проведення науково-дослідної роботи на тему «Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.17)$$

$В_{заг}$= 98614,29 +4278,03 +10289,23 +24899,94035 +5308,05 +2310,00 +0,00 + 33144,60 + 11179,00 +1560,54 +25723,08 +43214,77 +82313,85 +144049,24 = 486884,62 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (5.18)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,9.

ЗВ = 486884,62 / 0,9 = 540982,91 грн.

5.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою «Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі» передбачають комерціалізацію протягом 4-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

Показник	1-й рік	2-й рік	3-й рік	4-й рік
Збільшення кількості споживачів, осіб	1650	2000	2000	1850

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 7300 осіб;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 9800,00 грн;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 482,50 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 4-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [20]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.19)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2022 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо ρ =40%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2022 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1$ =(482,50·7300,00+10282,50·1650)·0,83·0,4·(1-0,18/100%)=5577755,21 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2$ =(482,50·7300,00+10282,50·3650)·0,83·0,4·(1-0,18/100%)=11176370,81 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3$ =(482,50·7300,00+10282,50·5650)·0,83·0,4·(1-0,18/100%)=16774986,41 грн.

Збільшення чистого прибутку 4-го року:

$\Delta П_4$ =(482,50·7300,00+10282,50·7500)·0,83·0,4·(1-0,18/100%)=21953705,84 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.20)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ=0,16;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП$ =5577755,21/(1+0,16)^1+11176370,81/(1+0,16)^2+16774986,41/(1+0,16)^3+ +21953705,84/(1+0,16)^4=4808409,66+8305864,16+10747023,77+12124836,30=359 86133,89 грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \quad (5.21)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 540982,91 грн.

$PV = k_{інв} \cdot ЗВ$ = 2 · 540982,91 = 1081965,83 грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{5.22}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 35986133,89 грн;

PV – теперішня вартість початкових інвестицій, 1081965,83 грн.

$E_{абс} = ПП - PV$ = 35986133,89 - 1081965,83 = 34904168,06 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.23}$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 34904168,06 грн;

PV – теперішня вартість початкових інвестицій, 1081965,83 грн;

$T_{ж}$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 4 роки.

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+34904168,06/1081965,83)^{1/4} = 1,40.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \tag{5.24}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d =0,11;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,4.

$\tau_{мін}$ = 0,11+0,4 = 0,51 < 1,40 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \tag{5.25}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 1,40 = 0,71 р.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.4 Висновок до розділу 5

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі» становить 41,7 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

Також термін окупності становить 0,71 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може

спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі».

ВИСНОВКИ

При виконанні магістерської кваліфікаційної роботи розв’язано задачу розробки інформаційної технології та програмного забезпечення розпізннання вимовлених приголосних літер на основі спайкінгової нейронної мережі.

У першому розділі магістерської роботи було розглянуто стан питання розпізнавання вимовлених літер. Проведено огляд відомих методів для роботи з інтелектуальними обчисленнями у сфері розпізнавання вимовлених літер. В ході аналізу предметної області розглянуто основні методи розпізнавання мовних одиниць та як найбільш перспективний, було обрано нейромережевий метод. Також було здійснено аналіз відомих програмних засобів розпізнавання вимовлених літер та як аналог до розроблюваної програми було обрано програму на основі нейронної мережі затримки часу TDNN.

У другому розділі, дослідивши відомі нейронні мережі, для інформаційної технології розпізнавання вимовлених літер, як основа, була обрана спайкінгова нейронна мережа. Розроблено основні принципи обробки звукових сигналів за допомогою спектрального аналізу та визначення коефіцієнтів спектрального потоку, що дає можливість визначити інформативні ознаки окремої вимовленої літери. Визначено структуру спайкінгової нейронної мережі, метод навчання нейронної мережі та принципи ідентифікації відповідної вимовленої літери. Було розроблено структуру інформаційної технології розпізнавання вимовлених літер з використанням спайкінгової нейронної мережі, на основі якої розроблено алгоритм роботи та UML діаграму програми розпізнавання вимовлених літер.

У третьому розділі було обгрунтовано вибір мови програмування Python, середовища розробки PyCharm та спеціалізованої бібліотеки для створення та навчання спайкінгових нейронних мереж PySNN. Розроблено структуру програми розпізнавання вимовлених літер на основі спайкінгової нейронної мережі та описано особливості її функціонування. Здійснено програмну

реалізація інформаційної технології розпізнавання вимовлених літер на основі спайкінгової нейронної мережі.

У четвертому розділі описано процес тестування розробленої програми розпізнавання вимовлених літер. Були наведені основні результати роботи розробленого програмного засобу, проведені експерименти над ним та проаналізовані результати даного тестування, що дозволило довести правильність роботи програми та визначили її перевагу перед аналогом у достовірності розпізнавання вимовлених літер. Розроблена програма має вищу достовірність розпізнавання вимовлених літер (61,7%), ніж аналогічна програма (53,3%), а значить достовірність розпізнавання вимовлених літер покращена на 8,4% за рахунок використання спайкінгової нейронної мережі, тобто мета роботи досягнута.

У п`яттому розділі визначено рівень комерційного потенціалу розробки, який становить 41,7 бала, що свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий). Термін окупності становить 0,71 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок. Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі».

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. В. О. Семчук, Д. А. Милосердов, О. К. Колесницький «Інформаційна технологія розпізнавання вимовлених літер на основі спайкінгової нейронної мережі», в Матеріали конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)», Вінниця, 2023, [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2023/paper/viewFile/16854/14055.

2. Webb A. Statistical Pattern Recognition. 1 ed. / A. Webb. — New York : Arnold Publishers, 1999. — 340 p.

3. Juang, B. H.; Rabiner, Lawrence R. "Automatic speech recognition–a brief history of the technology development" (PDF): Proc. Interactive Voice Technology for Telecommunications Applications (IVTTA), pp. 57-60, Oct. 2015.

4. Руденко О.Г. Искусственные нейронные сети / О.Г. Руденко, Е.В. Бодянский – Харьков, 2005. – 407с.

5. Круг П. Г. Нейронные сети и нейрокомпьютеры: учебное пособие / П. Г. Круг. — М. : Издательство МЭИ, 2002. — 176 с.

6. Esposito, A., Ezin, C. E., Ceccarelli, M., 1996, “Preprocessing and Neural Classification of English Stop Consonants”, ICSLP 96. Proceedings, vol. 2, 1249-1252

7. Tavanaei, A., Maida, A. S., 2017, “A Spiking Network that Learns to Extract Spike Signatures from Speech Signals”, Neurocomputing, 240, 191-199

8. Maass W. Pulsed Neural Networks / W. Maass, C. M. Bishop. — Cambridge : MIT Press, 1999. — 384 p.

9. Maass W. Real-time computing without stable states: A new framework for neural computation based on perturbations / W. Maass, T. Natschläger, H. Markram // Neural Computation. — 2002. — Vol. 14(11). — P. 2531—2560.

10.T. Natschläger. The «liquid computer»: A novel strategy for real-time computing on time series / T. Natschläger, W. Maass, and H. Markram // Special

Issue on Foundations of Information Processing of TELEMATIK. — 2002. — Vol. 8(1). — P. 39—43.

11.Дедус Ф. Ф., Махортых С. А., Устинин М. Н., Дедус А. Ф. Обобщённый спектрально-аналитический метод обработки информационных массивов. — М.: Машиностроение, 1999. — 356 с. — (Задачи анализа изображений и распознавания образов). — ISBN 5-217-02929-3

12.Fang Zheng, Guoliang Zhang and Zhanjiang Song (2001), "Comparison of Different Implementations of MFCC," J. Computer Science & Technology, 16(6): 582–589.

13.Бардаченко В. Ф. Перспективи застосування імпульсних нейронних мереж з таймерним представленням інформації для розпізнавання динамічних образів / В. Ф. Бардаченко, О. К. Колесницький, С. А. Василецький // УСіМ. — 2003. — № 6. — С. 73—82.

14.Neurocomputer architecture based on spiking neural network and its optoelectronic implementation / Oleh K. Kolesnytskyj; Vladislav V. Kutsman; Krzysztof Skorupski; Mukaddas Arshidinova, Proc. SPIE 11176, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2019, 1117609 (6 November 2019); doi: 10.1117/12.2536607.

15.Izhikevich E.M. (2003) Simple Model of Spiking Neurons. IEEE Transactions on Neural Networks, 14:1569- 1572

16.Song S, Miller KD, Abbott LF (September 2000). "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity". Nature Neuroscience. 3 (9): 919–26. doi:10.1038/78829.

17.Колесницький О. К. Комп'ютерне моделювання методу розпізнавання сигналів мультисенсорів газів на основі імпульсної нейронної мережі / О.К. Колесницький, С.І. Лукаш, Є.О. Гордишевська // Вимірювальна та обчислювальна техніка в технологічних процесах. – 2013. – №1. – С.121-126, ISSN 2219-9365 [Електронний ресурс]. Режим доступу - http://journals.khnu.km.ua/vottp/pdf/2013_1/41kol.pdf.

18.Python [Електронний ресурс]. – Режим доступу: https://www.python.org/.

19.Project description/ PySNN. [Електронний ресурс]. Режим доступу: https://pypi.org/project/pysnn/

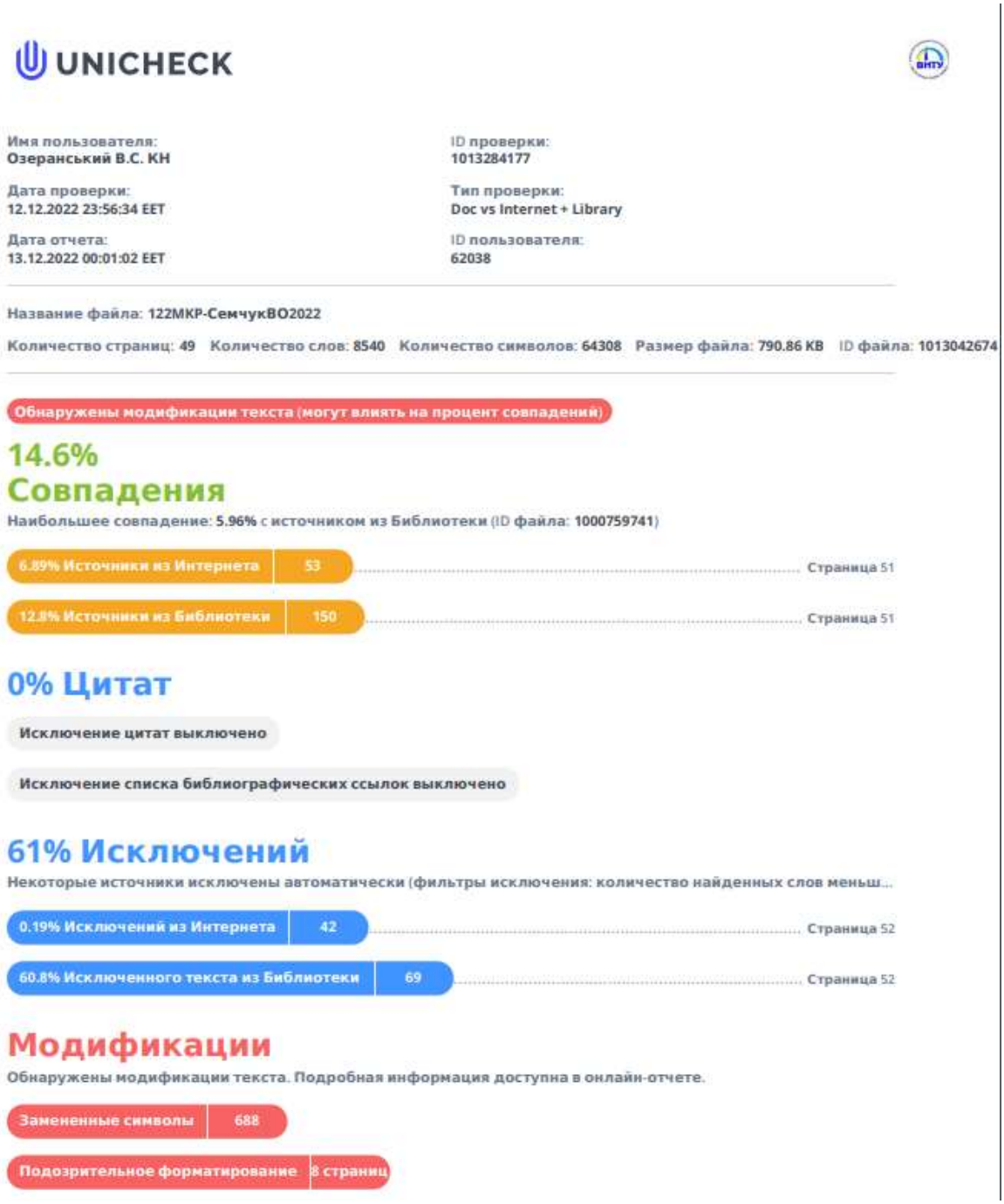
20.Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

21.Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа – Вінниця : ВНТУ, 2016. – 113 с.

ДОДАТКИ

Додаток А (обов'язковий)

Результат перевірки на плагіат в онлайн-системі UNICHECK



UNICHECK

Имя пользователя:
Озеранський В.С. КН

ID проверки:
1013284177

Дата проверки:
12.12.2022 23:56:34 EET

Тип проверки:
Doc vs Internet + Library

Дата отчета:
13.12.2022 00:01:02 EET

ID пользователя:
62038

Название файла: 122МКР-СемчукВО2022

Количество страниц: 49 Количество слов: 8540 Количество символов: 64308 Размер файла: 790.86 KB ID файла: 1013042674

Обнаружены модификации текста (могут влиять на процент совпадений)

14.6%
Совпадения

Наибольшее совпадение: 5.96% с источником из Библиотеки (ID файла: 1000759741)

6.89% Источники из Интернета 53 ..... Страница 51

12.8% Источники из Библиотеки 150 ..... Страница 51

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

61% Исключений

Некоторые источники исключены автоматически (фильтры исключения: количество найденных слов меньш...

0.19% Исключений из Интернета 42 ..... Страница 52

60.8% Исключенного текста из Библиотеки 69 ..... Страница 52

Модификации

Обнаружены модификации текста. Подробная информация доступна в онлайн-отчете.

Замененные символы 688

Подозрительное форматирование 8 страниц

Додаток Б (обов'язковий)

Лістинг програми

Network.py

```
import numpy as np
import Synapse
import Neuron
import Utils
from matplotlib import pyplot as plt
from neuronpy.graphics import spikeplot

global input_layer
global output_layer
global hidden_layer
global synapses
global a, b, c, d, time_ita

class Network:

    def __init__(self, weights):
        # self.a = 0.03
        # self.b = -2
        # self.c = -50.
        # self.d = 100.
        self.a = 0.02
        self.b = 0.2
        self.c = -65.
        self.d = 8.
        self.time_ita = 3000  # 100ms

        # Build a layer of 120 input neurons (20 frames, 6 features for each frame)
        self.input_layer = []
        for i in range(520):
            n = Neuron.Neuron()
            self.input_layer.append(n)

        # Build output layer, one neuron for each letter of the alphabet
        self.output_layer = []
        for i in range(3):
            o = Neuron.Neuron()
            self.output_layer.append(o)

        i = 0
        # For each input neuron, append one synapse to each output neuron
        for n in self.input_layer:
            for out in self.output_layer:
                synapse = Synapse.Synapse()
                n.append_synapse(synapse)
                out.append_synapse(synapse)

        if weights is not None:
            i = 0
            n = 0
            s = 0
            with open(weights) as f:
                for line in f:
                    # For every 520 synapses, go to the next neuron
                    if i % 520 == 0 and n < len(self.output_layer):
                        s = 0
                        out = self.output_layer[n]
                        n += 1
                        out.synapses[s].set_weight(line)
                        s += 1
                    elif s < 520:
                        out.synapses[s].set_weight(line)
                        s += 1
                    i += 1

    # Calculates the current from the given MFCC value
    def get_current(self, x):
        if x == 0:
            return 8.5
        elif x < 0:
            return (.0530914398 * x) + 8.5
```

```
        else:
            return (.11805555555 * x) + 8.5
        # if x == 0:
        #     return 8.8
        # elif x < 0:
        #     return (.0949 * x) + 8.8
        # else:
        #     return (.1181 * x) + 8.8
        # if x == 0:
        #     return 4.7
        # elif x < 0:
        #     return (.0507 * x) + 4.7
        # else:
        #     return (.05 * x) + 4.7

    # Equation 8
    def get_total(self, neuron):
        g_tot = 0
        for synapse in neuron.synapses:
            t = 0
            i = 0
            tau = 2
            #print("Synapse")
            for j in synapse.spike:
                if j == 1:
                    t_kj = synapse.time[i]
                    #g_tot += synapse.synapse(tau)
                    t = np.abs(t - t_kj)
                    #print("\t%s" % t)
                    g_tot += synapse.w * (t - t_kj) * np.exp(-(t - t_kj) / tau)
                    t = t_kj
                i += 1
        return g_tot


    # Get the total synaptic output for this neuron
    def total_synaptic_value(self, neuron):
        conductance = 0
        for syn_k in neuron.synapses:
            output = syn_k.synapse(2)
            conductance += output
        return conductance

    # if result == 0, then our target neuron is the first neuron in the output layer
    # result == 1 --> 2nd output neuron, result == 3 --> 3rd output neuron and so on
    def conduct_training(self, result):
        i = 0
        for out in self.output_layer:
            if i == result:
                # Undergo Hebbian STDP
                for syn in out.synapses:
                    syn.Heb_STDP()
            else:
                # Undergo anti-Hebbian STDP for non-target synapses
                for syn in out.synapses:
                    syn.Anti_Heb_STDP()
            i += 1


    # Perform analysis on the given filename using mel_Freq command
    def start(self, fname):

        features = Utils.get_mel(fname)
        features = features[:520]

        #Feed features into our network and get spike information (number of spikes, time of
largest spike)
        i = 0

        # Use for mel_freq. 520 input neurons
        for feature in features:
            n = self.input_layer[i]
            #current = np.ones(self.time_ita) * feature
            current = np.ones(self.time_ita) * self.get_current(feature)
            time, v_plt, spike, num_spikes, spike_times =
n.izh_simulation(self.a,self.b,self.c,self.d,self.time_ita, current, self.c)
            # plt.plot(time, v_plt, 'b-')
            # plt.show()
            # Set pre spikes for each synapse connected to this neuron
```

```
            for synapse in n.synapses:
                synapse.set_pre_spikes(spike_times)
                synapse.set_time(time)
                synapse.set_spike(spike)
            i += 1


        # Create a 3 neuron output vector
        outputs = [0] * 3
        spikes = []
        v_plts = []
        currents = []
        i = 0
        for out in self.output_layer:
            current = np.ones(self.time_ita) * self.total_synaptic_value(out)
            time, v_plt, spike, num_spikes, spike_times =
out.output_izh_simulation(self.a,self.b,self.c,self.d,self.time_ita, current, self.c)
            spikes.append(spike_times)
            v_plts.append(v_plt)
            currents.append(current)
            for syn in out.synapses:
                syn.set_post_spikes(spike_times)

            outputs[i] = num_spikes
            i += 1

        return outputs, currents, time, v_plts, spikes
```

Neuron.py

```
import numpy as np
import random
import Synapse

global Pref, Pmin, Pth, D, Prest, pre_times, post_times, synapses
global out_synapses
global in_synapses
Pref = 0
Prest = 0
Pmin = -1
Pth = 5.5
D = 0.5


class Neuron:
    def __init__(self):
        self.Pth = Pth
        self.t_ref = 4
        self.t_rest = -1
        self.P = Prest
        self.D = D
        self.Pmin = Pmin
        self.Prest = Prest
        self.post_times = []
        self.pre_times = []
        self.synapses = []
        self.in_synapses = []
        self.out_synapses = []

    def append_pre_times(self, times):
        self.pre_times = times

    def append_post_times(self, times):
        self.post_times = times

    def append_synapse(self, synapse):
        self.synapses.append(synapse)

    def append_in_synapse(self, synapse):
        self.in_synapses.append(synapse)

    def append_out_synapse(self, synapse):
        self.out_synapses.append(synapse)

    def output_izh_simulation(self, a, b, c, d, time_ita, current, v_init):
        # a,b,c,d parameters for Izhikevich model
        # time_ita time iterations for euler method
        # current list of current for each time step
        # v_init initial voltage
        spike_times = []
```

```
        v = v_init
        u = v * b
        v_plt = np.zeros(time_ita)
        u_plt = np.zeros(time_ita)
        spike = np.zeros(time_ita)
        num_spikes = 0
        tstep = 0.1  # ms
        ita = 0
        while ita < time_ita:
            if ita < 200:
                v_plt[ita] = v_init
            else:
                v_plt[ita] = v
            u_plt[ita] = u
            v += tstep * (0.04 * (v ** 2) + 5 * v + 140 - u + current[ita])
            u += tstep * a * (b * v - u)
            if v > 30.:
                if ita > 200:
                    spike[ita] = 1
                    num_spikes += 1
                v = c
                u += d

                # spike_times.append(ita)

            ita += 1
        time = np.arange(time_ita) * tstep
        i = 0
        for t in time:
            if spike[i] == 1:
                spike_times.append(t)
            i += 1
        return time, v_plt, spike, num_spikes, spike_times

    def izh_simulation(self, a, b, c, d, time_ita, current, v_init):
        # a,b,c,d parameters for Izhikevich model
        # time_ita time iterations for euler method
        # current list of current for each time step
        # v_init initial voltage
        spike_times = []
        v = v_init
        u = v * b
        v_plt = np.zeros(time_ita)
        u_plt = np.zeros(time_ita)
        spike = np.zeros(time_ita)
        num_spikes = 0
        tstep = 0.1  # ms
        ita = 0
        while ita < time_ita:
            v_plt[ita] = v
            u_plt[ita] = u
            v += tstep * (0.04 * (v ** 2) + 5 * v + 140 - u + current[ita])
            u += tstep * a * (b * v - u)
            if v > 30.:
                spike[ita] = 1
                v = c
                u += d
                num_spikes += 1
                #spike_times.append(ita)

            ita += 1
        time = np.arange(time_ita) * tstep
        i = 0
        for t in time:
            if spike[i] == 1:
                spike_times.append(t)
            i += 1
        return time, v_plt, spike, num_spikes, spike_times
```

SNN.py

```
from random import shuffle
import os
import sys
import Utils
import Network
import pyspike as spk
from pyspike import SpikeTrain
from datetime import datetime
from matplotlib import pyplot as plt
```

```
from neuronpy.graphics import spikeplot

prototype_trains = [None] * 3

def write_weights(network):
    i = 0
    with open("weights.txt", "a") as f:
        for out in network.output_layer:
            if i == 0:
                f.write("A\n")
            elif i == 1:
                f.write("B\n")
            elif i == 2:
                f.write("C\n")
            elif i == 3:
                f.write("D\n")
            elif i == 4:
                f.write("E\n")
            elif i == 5:
                f.write("F\n")
            elif i == 6:
                f.write("G\n")
            elif i == 7:
                f.write("H\n")
            elif i == 8:
                f.write("I\n")
            elif i == 9:
                f.write("J\n")
            elif i == 10:
                f.write("K\n")
            elif i == 11:
                f.write("L\n")
            elif i == 12:
                f.write("M\n")
            elif i == 13:
                f.write("N\n")
            elif i == 14:
                f.write("O\n")
            elif i == 15:
                f.write("P\n")
            elif i == 16:
                f.write("Q\n")
            elif i == 17:
                f.write("R\n")
            elif i == 18:
                f.write("S\n")
            elif i == 19:
                f.write("T\n")
            elif i == 20:
                f.write("U\n")
            elif i == 21:
                f.write("V\n")
            elif i == 22:
                f.write("W\n")
            elif i == 23:
                f.write("X\n")
            elif i == 24:
                f.write("Y\n")
            elif i == 25:
                f.write("Z\n")
            for syn in out.synapses:
                if i == 0:
                    f.write("%s\n" % syn.w)
                    i = 1
                else:
                    f.write("%s\n" % syn.w)
                    i = 0
            i += 1

def print_result(results):
    print('\tA: ' + str(results[0]))
    print('\tB: ' + str(results[1]))
    print('\tX: ' + str(results[2]))
    print('\tD: ' + str(results[3]))
    print('\tE: ' + str(results[4]))
    print('\tF: ' + str(results[5]))
    print('\tG: ' + str(results[6]))
    print('\tH: ' + str(results[7]))
    print('\tI: ' + str(results[8]))
```

```
    print('\tJ: ' + str(results[9]))
    print('\tK: ' + str(results[10]))
    print('\tL: ' + str(results[11]))
    print('\tM: ' + str(results[12]))
    print('\tN: ' + str(results[13]))
    print('\tO: ' + str(results[14]))
    print('\tP: ' + str(results[15]))
    print('\tQ: ' + str(results[16]))
    print('\tR: ' + str(results[17]))
    print('\tS: ' + str(results[18]))
    print('\tT: ' + str(results[19]))
    print('\tU: ' + str(results[20]))
    print('\tV: ' + str(results[21]))
    print('\tW: ' + str(results[22]))
    print('\tX: ' + str(results[23]))
    print('\tY: ' + str(results[24]))
    print('\tZ: ' + str(results[25]))

# Generate a spike train from the given spike
def generate_prototypes(spike, key):
    global prototype_trains
    spike_train = SpikeTrain(spike, [0.0, 300.0])
    if key == 'A':
        prototype_trains[0] = spike_train
    elif key == 'B':
        prototype_trains[1] = spike_train
    elif key == 'X':
        prototype_trains[2] = spike_train

def spike_analysis(spikes, value):
    distances = []
    i = 0
    for spike in spikes:
        spike_train = SpikeTrain(spike, [0.0, 300.0])
        isi_profile = spk.spike_sync(prototype_trains[i], spike_train)
        distances.append(isi_profile)
        i += 1

    val, idx = max((val, idx) for (idx, val) in enumerate(distances))
    print("Distance: %.8f" % val)
    print("Index: %s" % idx)


def show_plots(time, v_plts, currents, spikes):
    plt.figure('A')
    plt.plot(time, v_plts[0], 'g-')
    plt.plot(time, currents[0], 'r-')
    plt.figure('B')
    plt.plot(time, v_plts[1], 'b-')
    plt.plot(time, currents[1], 'y-')
    plt.figure('X')
    plt.plot(time, v_plts[2], 'k-')
    plt.plot(time, currents[2], 'm-')
    plt.figure('All')
    plt.plot(time, v_plts[0], 'g-')
    plt.plot(time, currents[0], 'r-')
    plt.plot(time, v_plts[1], 'b-')
    plt.plot(time, currents[1], 'y-')
    plt.plot(time, v_plts[2], 'k-')
    plt.plot(time, currents[2], 'm-')
    sp = spikeplot.SpikePlot()
    sp.plot_spikes(spikes)
    plt.show()

# Test our network
def test():
    global prototype_trains
    prototype_trains = [None] * 3
    mapping = dict()

    weights = "weights.txt"

    network = Network.Network(weights=weights)
    audio_path = "letter_audio/speech/isolet3"

    audio = [os.path.join(root, name)
                for root, dirs, files in os.walk(audio_path)
                for name in files
                if name.endswith((".wav"))]
```

```
    for fname in audio:
        mapping[fname] = Utils.get_label(fname)

    a_count = 1
    b_count = 1
    x_count = 1
    count = 3
    for key in mapping:
        if mapping[key] == 'A' and a_count != 0:
            print(key)
            results, currents, time, v_plts, spikes = network.start(key)
            print_result(results)
            a_count -= 1
            count -= 1
            if a_count == 0:
                # Generate a spike train for the 'A' sound
                generate_prototypes(spikes[0], 'A')
            elif a_count == 0 and b_count == 0 and x_count == 0:
                spike_analysis(spikes)
                #show_plots(time, v_plts, currents, spikes)
                #spike_analysis(spikes)
        elif mapping[key] == 'B' and b_count != 0:
            print(key)
            results, currents, time, v_plts, spikes = network.start(key)
            print_result(results)
            b_count -= 1
            count -= 1
            if b_count == 0:
                # Generate a spike train for the 'B' sound
                generate_prototypes(spikes[1], 'B')
            elif a_count == 0 and b_count == 0 and x_count == 0:
                spike_analysis(spikes)
                #show_plots(time, v_plts, currents, spikes)
                #spike_analysis(spikes)
        elif mapping[key] == 'X' and x_count != 0:
            print(key)
            results, currents, time, v_plts, spikes = network.start(key)
            print_result(results)
            x_count -= 1
            count -= 1
            if x_count == 0:
                # Generate a spike train for the 'X' sound
                generate_prototypes(spikes[2], 'X')
            elif a_count == 0 and b_count == 0 and x_count == 0:
                spike_analysis(spikes)
        elif count == 0:
            print(key)
            value = ''
            if mapping[key] == 'A':
                value = 'A'
            elif mapping[key] == 'B':
                value = 'B'
            elif mapping[key] == 'X':
                value = 'X'
            results, currents, time, v_plts, spikes = network.start(key)
            print_result(results)
            spike_analysis(spikes, value)


# Train the network
def train():
    network = Network.Network(weights=None)

    mapping = dict()

    audio_path = "letter_audio/speech/isolet1"

    # Gets list of all audio files in the directory
    audio = [os.path.join(root, name)
             for root, dirs, files in os.walk(audio_path)
             for name in files
             if name.endswith((".wav"))]

    audio_path = "letter_audio/speech/isolet2"
    audio2 = [os.path.join(root, name)
              for root, dirs, files in os.walk(audio_path)
              for name in files
              if name.endswith((".wav"))]
```

```
audio.extend(audio2)

shuffle(audio)

# Get a mapping of labels to audio
for fname in audio:
    mapping[fname] = Utils.get_label(fname)

print(datetime.now())

a_count = 20
b_count = 20
c_count = 20
d_count = 20
e_count = 20
f_count = 20
g_count = 20
h_count = 20
i_count = 20
j_count = 20
k_count = 20
l_count = 20
m_count = 20
n_count = 20
o_count = 20
p_count = 20
q_count = 20
r_count = 20
s_count = 20
t_count = 20
u_count = 20
v_count = 20
w_count = 20
x_count = 20
y_count = 20
z_count = 20

for key in mapping:
    if mapping[key] == 'A' and a_count > 0:
        print(key)
        results, currents, time, v_plts, spikes = network.start(key)
        print_result(results)
        network.conduct_training(0)
        a_count -= 1
    elif mapping[key] == 'B' and b_count > 0:
        print(key)
        results, currents, time, v_plts, spikes = network.start(key)
        print_result(results)
        network.conduct_training(1)
        b_count -= 1
    elif mapping[key] == 'C' and c_count > 0:
        print(key)
        results, currents, time, v_plts, spikes = network.start(key)
        print_result(results)
        network.conduct_training(2)
        c_count -= 1
    elif mapping[key] == 'D' and d_count > 0:
        print(key)
        results = network.start(key)
        print_result(results)
        network.conduct_training(3)
        d_count -= 1
    elif mapping[key] == 'E' and e_count > 0:
        print(key)
        results = network.start(key)
        print_result(results)
        network.conduct_training(4)
        e_count -= 1
    elif mapping[key] == 'F' and f_count > 0:
        print(key)
        results = network.start(key)
        print_result(results)
        network.conduct_training(5)
        f_count -= 1
    elif mapping[key] == 'G' and g_count > 0:
        print(key)
        results = network.start(key)
        print_result(results)
```

```
    network.conduct_training(6)
    g_count -= 1
elif mapping[key] == 'H' and h_count > 0:
    print(key)
    results = network.start(key)
    print_result(results)
    network.conduct_training(7)
    h_count -= 1
elif mapping[key] == 'I' and i_count > 0:
    print(key)
    results = network.start(key)
    print_result(results)
    network.conduct_training(8)
    i_count -= 1
elif mapping[key] == 'J' and j_count > 0:
    print(key)
    results = network.start(key)
    print_result(results)
    network.conduct_training(9)
    j_count -= 1
elif mapping[key] == 'K' and k_count > 0:
    print(key)
    results = network.start(key)
    print_result(results)
    network.conduct_training(10)
    k_count -= 1
elif mapping[key] == 'L' and l_count > 0:
    print(key)
    results = network.start(key)
    print_result(results)
    network.conduct_training(11)
    l_count -= 1
elif mapping[key] == 'M' and m_count > 0:
    print(key)
    results = network.start(key)
    print_result(results)
    network.conduct_training(12)
    m_count -= 1
elif mapping[key] == 'N' and n_count > 0:
    print(key)
    results = network.start(key)
    print_result(results)
    network.conduct_training(13)
    n_count -= 1
elif mapping[key] == 'O' and o_count > 0:
    print(key)
    results = network.start(key)
    print_result(results)
    network.conduct_training(14)
    o_count -= 1
elif mapping[key] == 'P' and p_count > 0:
    print(key)
    results = network.start(key)
    print_result(results)
    network.conduct_training(15)
    p_count -= 1
elif mapping[key] == 'Q' and q_count > 0:
    print(key)
    results = network.start(key)
    print_result(results)
    network.conduct_training(16)
    q_count -= 1
elif mapping[key] == 'R' and r_count > 0:
    print(key)
    results = network.start(key)
    print_result(results)
    network.conduct_training(17)
    r_count -= 1
elif mapping[key] == 'S' and s_count > 0:
    print(key)
    results = network.start(key)
    print_result(results)
    network.conduct_training(18)
    s_count -= 1
elif mapping[key] == 'T' and t_count > 0:
    print(key)
    results = network.start(key)
    print_result(results)
    network.conduct_training(19)
```

```
            t_count -= 1
        elif mapping[key] == 'U' and u_count > 0:
            print(key)
            results = network.start(key)
            print_result(results)
            network.conduct_training(20)
            u_count -= 1
        elif mapping[key] == 'V' and v_count > 0:
            print(key)
            results = network.start(key)
            print_result(results)
            network.conduct_training(21)
            v_count -= 1
        elif mapping[key] == 'W' and w_count > 0:
            print(key)
            results = network.start(key)
            print_result(results)
            network.conduct_training(22)
            w_count -= 1
        elif mapping[key] == 'X' and x_count > 0:
            print(key)
            results = network.start(key)
            print_result(results)
            network.conduct_training(23)
            x_count -= 1
        elif mapping[key] == 'Y' and y_count > 0:
            print(key)
            results = network.start(key)
            print_result(results)
            network.conduct_training(24)
            y_count -= 1
        elif mapping[key] == 'Z' and z_count > 0:
            print(key)
            results = network.start(key)
            print_result(results)
            network.conduct_training(25)
            z_count -= 1

    write_weights(network)
    print(datetime.now())


if __name__ == "__main__":
    if len(sys.argv) > 1:
        if sys.argv[1] == 'train':
            print('Training...')
            train()
        else:
            print('Testing')
            test()
```

Synapse.py

```
import numpy as np
import random as random


class Synapse:

    global spike, time, conductance_amplitude, w, spikes_received, pre_spikes, post_spikes,
input_neuron, out_neuron

    def __init__(self):
        self.post_spikes = []
        self.pre_spikes = []
        #self.w = random.uniform(0, 1)
        self.w = .5

    def set_weight(self, w):
        self.w = float(w)

    def set_input_neuron(self, neuron):
        self.input_neuron = neuron

    def set_out_neuron(self, neuron):
        self.out_neuron = neuron

    def set_time(self, time):
        self.time = time
```

```
    def set_spike(self, spike):
        self.spike = spike

    def set_pre_spikes(self, pre_spikes):
        self.pre_spikes = pre_spikes

    def set_post_spikes(self, post_spikes):
        self.post_spikes = post_spikes

    # Our W function for Hebbian STDP
    def synaptic_weight_func(self, delta_t):
        tau_pre = 20
        tau_post = 20
        Apre = .10
        Apost = -Apre
        if delta_t >= 0:
            return Apre*np.exp(-np.abs(delta_t)/tau_pre)
        if delta_t < 0:
            return Apost*np.exp(-np.abs(delta_t)/tau_post)

    # Our W function for anti-Hebbian STDP
    def anti_heb(self, delta_t):
        tau_pre = 20
        tau_post = 20
        Apre = .10
        Apost = -Apre
        if delta_t < 0:
            return Apre*np.exp(-np.abs(delta_t)/tau_pre)
        if delta_t >= 0:
            return Apost*np.exp(-np.abs(delta_t)/tau_post)

    # Same as Hebbian STDP except the cases are reversed
    def Anti_Heb_STDP(self):
        delta_w = 0
        for t_pre in self.pre_spikes:
            for t_post in self.post_spikes:
                delta_w += self.anti_heb(t_post - t_pre)
        # for t_pre in self.pre_spikes:
        #     for t_post in self.post_spikes:
        #         delta_w += self.anti_heb(t_post - t_pre)
        self.w += delta_w
        if self.w < 0:
            self. w = 0


    # Change in synaptic weight is the sum over all presynaptic spike times (t_pre) and
postsynaptic spike times (t_post)
    # of some function W of the difference in these spike times
    def Heb_STDP(self):
        delta_w = 0
        # for t_pre in self.pre_spikes:
        #     for t_post in self.post_spikes:
        #         delta_w += self.anti_heb(t_post - t_pre)
        for t_pre in self.pre_spikes:
            for t_post in self.post_spikes:
                delta_w += self.synaptic_weight_func(t_post - t_pre)
        self.w += delta_w
        if self.w < 0:
            self.w = 0


    # Calculates synaptic output
    def synapse(self, tau):
        synapse_output = np.zeros(len(self.time))
        for t in range(len(self.time)):
            tmp_time = self.time[t] - self.time[0:t]
            synapse_output[t] = np.sum(((tmp_time * self.spike[0:t]) / tau) * np.exp(-(tmp_time *
self.spike[0:t]) / tau))
        return self.w * synapse_output

    def synapse_func(self, tau):
        time = np.arange(10000) * 0.1
        func = time / tau * np.exp(-time / tau)
        return time, func
```

Utils.py

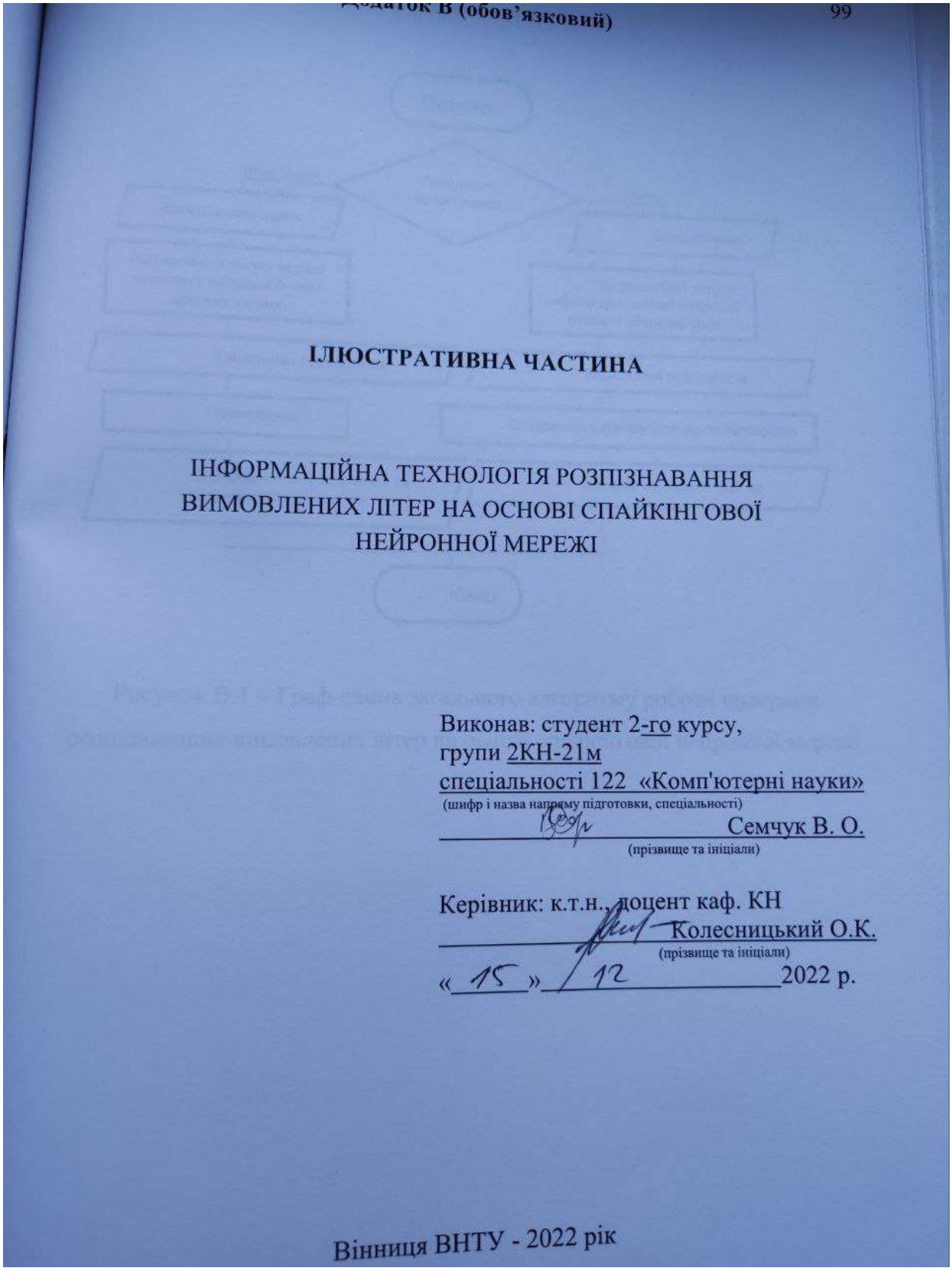
```
import numpy as np
import copy
import math
```

```
import ntpath
from scipy.fftpack import fft
from scipy.io import wavfile
from numpy.lib import stride_tricks
from python_speech_features import mfcc
from python_speech_features import delta
from python_speech_features import logfbank
from python_speech_features.sigproc import framesig
from python_speech_features.sigproc import powspec
from matplotlib.pyplot import specgram

import matplotlib.pyplot as plt
import wave
import contextlib


def mel_Freq(file_name):

    (rate, sig) = wavfile.read(file_name)

    with contextlib.closing(wave.open(file_name, 'r')) as f:
        frames = f.getnframes()
        rate = f.getframerate()
        duration = frames / float(rate)

    number_of_frames = 40
    samples_per_frame = len(sig) / number_of_frames

    win_length = duration / number_of_frames

    i = 0
    x = 1
    frames = []
    frame_sample = []
    for n in range(0, number_of_frames):
        frame_sample = sig[i:(samples_per_frame*x)]
        i = samples_per_frame*x
        x += 1
        frames.append(frame_sample)


    mfcc_frames = []
    for frame in frames:
        mel = mfcc(frame, rate, win_length)
        mfcc_frames.append(mel)

    return mfcc_frames


def get_features(file_name):
    (rate, sig) = wavfile.read(file_name)

    with contextlib.closing(wave.open(file_name, 'r')) as f:
        frames = f.getnframes()
        rate = f.getframerate()
        duration = frames / float(rate)

    # Frame our signal into 20 frames with 50% overlap
    number_of_frames = 40
    frame_len = len(sig) / (number_of_frames*(.5) + .5)
    frames = framesig(sig, frame_len, frame_len * .5)

    # A list of 20 frequency lists for each frame. 6 frequency bands with the average energy of
each
    features = []
    band0 = []
    band1 = []
    band2 = []
    band3 = []
    band4 = []
    band5 = []
    for frame in frames:
        spectrum, freqs, t, img = specgram(frame, Fs=rate)
        i = 0
        bands = []
        for freq in freqs:
            if freq <= 400:
                band0.extend(spectrum[i])
            elif freq > 400 and freq <= 800:
```

```
                band1.extend(spectrum[i])
            elif freq > 800 and freq <= 1600:
                band2.extend(spectrum[i])
            elif freq > 1600 and freq <= 2800:
                band3.extend(spectrum[i])
            elif freq > 2800 and freq <= 4400:
                band4.extend(spectrum[i])
            elif freq > 4400:
                band5.extend(spectrum[i])
            i += 1
        bands.append(sum(band0) / len(band0))
        bands.append(sum(band1) / len(band1))
        bands.append(sum(band2) / len(band2))
        bands.append(sum(band3) / len(band3))
        bands.append(sum(band4) / len(band4))
        bands.append(sum(band5) / len(band5))
        features.append(bands)

    values = []
    for feature in features:
        for f in feature:
            values.append(f)

    return values


def get_mel(file_name):
    (rate, sig) = wavfile.read(file_name)

    with contextlib.closing(wave.open(file_name, 'r')) as f:
        frames = f.getnframes()
        rate = f.getframerate()
        duration = frames / float(rate)

    number_of_frames = 40
    frame_len = len(sig) / (number_of_frames * (.5) + .5)
    step = frame_len * .5
    frames = framesig(sig, frame_len, step)
    win_length = (duration * 1000) / number_of_frames

    mel_values = []
    for frame in frames:
        mel_values.append(mfcc(frame, rate, win_length))

    values = []
    for v in mel_values:
        for coefs in v:
            for coef in coefs:
                values.append(coef)
    return values
    # (rate, sig) = wavfile.read(file_name)
    #
    # with contextlib.closing(wave.open(file_name, 'r')) as f:
    #     frames = f.getnframes()
    #     rate = f.getframerate()
    #     duration = frames / float(rate)
    #
    # number_of_frames = 20
    # frame_len = len(sig) / (number_of_frames*(.5) + .5)
    # frames = framesig(sig, frame_len, frame_len * .5)
    #
    # win_length = (duration * 1000) / number_of_frames
    #
    # mel_values = []
    # for frame in frames:
    #     mel_values.append(mfcc(frame, rate, win_length))
    #
    # values = []
    # for v in mel_values:
    #     for coefs in v:
    #         for coef in coefs:
    #             values.append(coef)
    # return values


# def get_features(file_name):
#     (rate, sig) = wavfile.read(file_name)
#
#     with contextlib.closing(wave.open(file_name, 'r')) as f:
```

```
#         frames = f.getnframes()
#         rate = f.getframerate()
#         duration = frames / float(rate)
#
#     # Frame our signal into 20 frames with 50% overlap
#     number_of_frames = 20
#     frame_len = len(sig) / (number_of_frames*(.5) + .5)
#     frames = framesig(sig, frame_len, frame_len * .5)
#
#     # A list of 20 frequency lists for each frame. 6 frequency bands with the average energy of
each
#     features = []
#     band0 = []
#     band1 = []
#     band2 = []
#     band3 = []
#     band4 = []
#     band5 = []
#     for frame in frames:
#         spectrum, freqs, t, img = specgram(frame, Fs=rate)
#         i = 0
#         bands = []
#         for freq in freqs:
#             if freq <= 400:
#                 band0.extend(spectrum[i])
#             elif freq > 400 and freq <= 800:
#                 band1.extend(spectrum[i])
#             elif freq > 800 and freq <= 1600:
#                 band2.extend(spectrum[i])
#             elif freq > 1600 and freq <= 2800:
#                 band3.extend(spectrum[i])
#             elif freq > 2800 and freq <= 4400:
#                 band4.extend(spectrum[i])
#             elif freq > 4400:
#                 band5.extend(spectrum[i])
#             i += 1
#         bands.append(np.log2(sum(band0) / len(band0)))
#         bands.append(np.log2(sum(band1) / len(band1)))
#         bands.append(np.log2(sum(band2) / len(band2)))
#         bands.append(np.log2(sum(band3) / len(band3)))
#         bands.append(np.log2(sum(band4) / len(band4)))
#         bands.append(np.log2(sum(band5) / len(band5)))
#         #print(bands)
#         features.append(bands)
#
#     return features

# def get_features(file_name):
#
#     (rate, sig) = wavfile.read(file_name)
#
#     with contextlib.closing(wave.open(file_name, 'r')) as f:
#         frames = f.getnframes()
#         rate = f.getframerate()
#         duration = frames / float(rate)
#
#     number_of_frames = 15
#     win_length = duration / number_of_frames
#     frame_len = len(sig) / number_of_frames
#     frames = framesig(sig, frame_len, frame_len*.5)
#
#     mfcc_frames = []
#     for frame in frames:
#         mfcc_frames.append(mfcc(frame, rate, win_length))
#
#     return mfcc_frames[:29]


# Get label associtaed with this file
def get_label(filename):
    head, tail = ntpath.split(filename)
    start = tail.index('-')
    tail = tail[(start+1):]
    end = tail.index('-')
    fname = tail[0:end]
    return fname[0]
```

Додаток В (обов'язковий)



Додаток В (обов'язковий) 99

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ РОЗПІЗНАВАННЯ ВИМОВЛЕНИХ ЛІТЕР НА ОСНОВІ СПАЙКІНГОВОЇ НЕЙРОННОЇ МЕРЕЖІ

Виконав: студент 2-го курсу,
групи 2КН-21м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
Семчук В. О.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Колесницький О.К.
(прізвище та ініціали)
«15» 12 2022 р.

Вінниця ВНТУ - 2022 рік

Рисунок В.1 – Граф-схема загального алгоритму роботи програми розпізнавання вимовлених літер на основі спайкінгової нейронної мережі

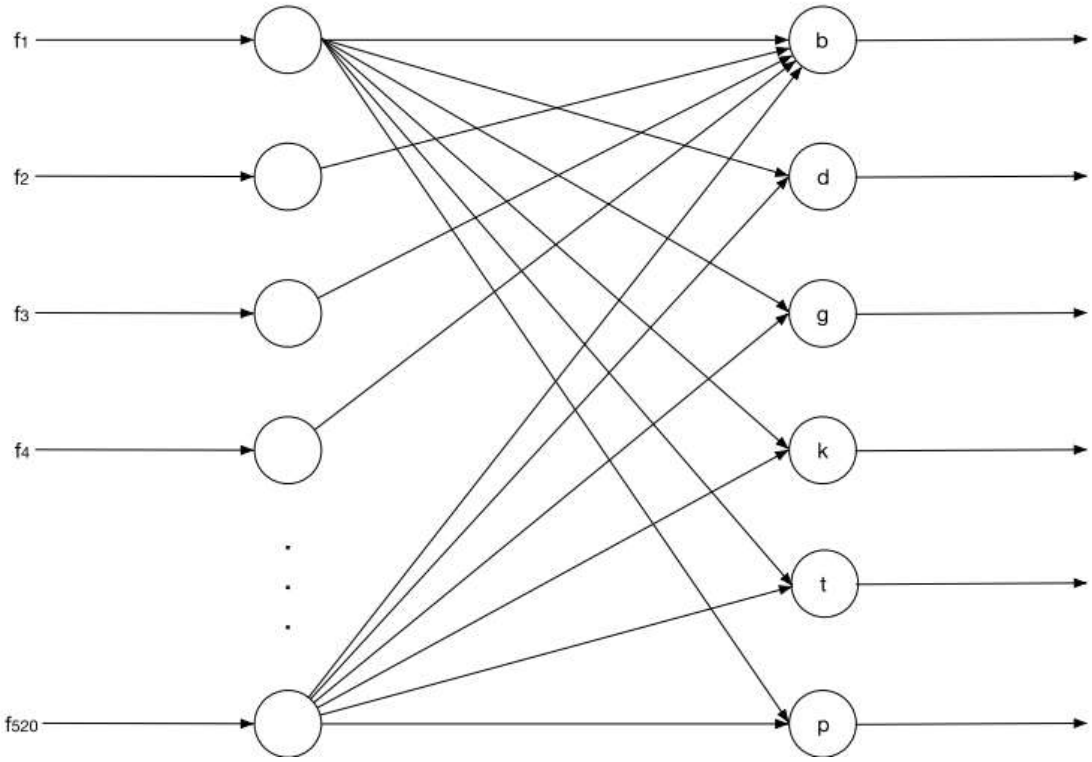


Рисунок В.2 – Структура спайкінгової нейронної мережі

Рисунок В.3 – Структура інформаційної технології розпізнавання вимовлених літер на основі спайкінгової нейронної мережі

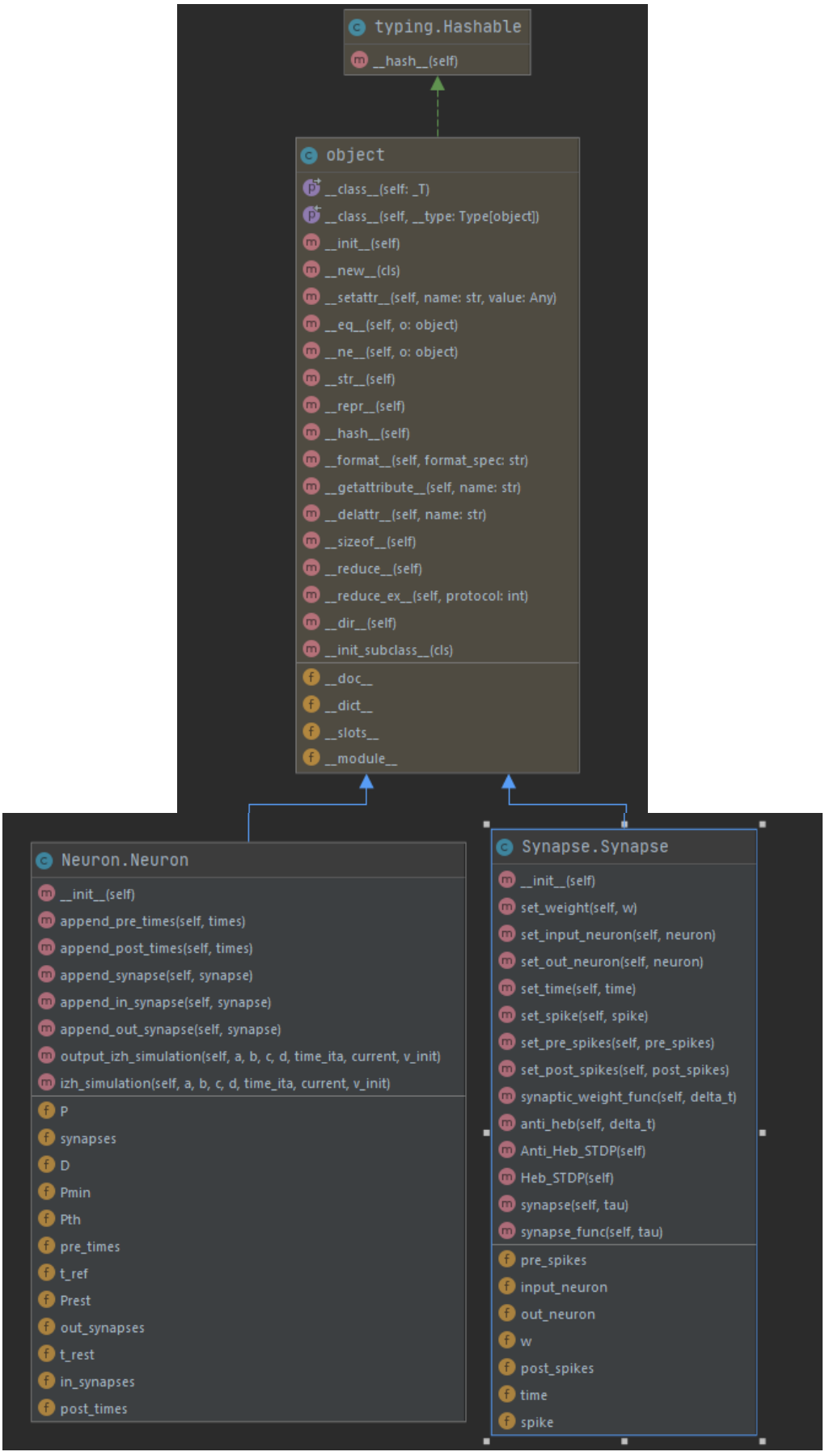


Рисунок В.4 – Діаграма класів програми розпізнавання вимовлених літер

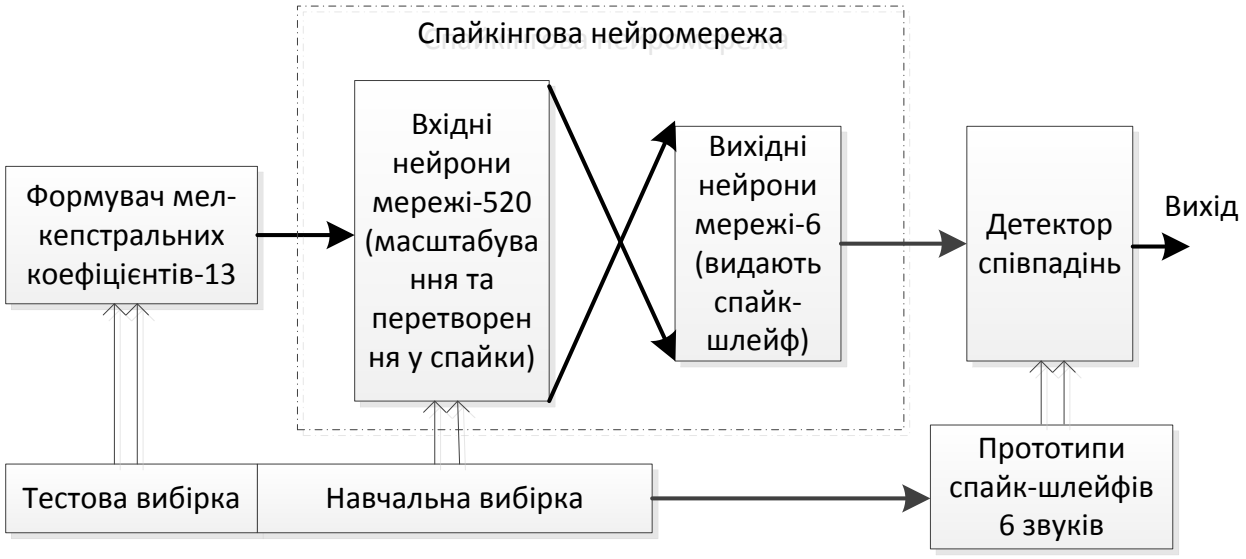
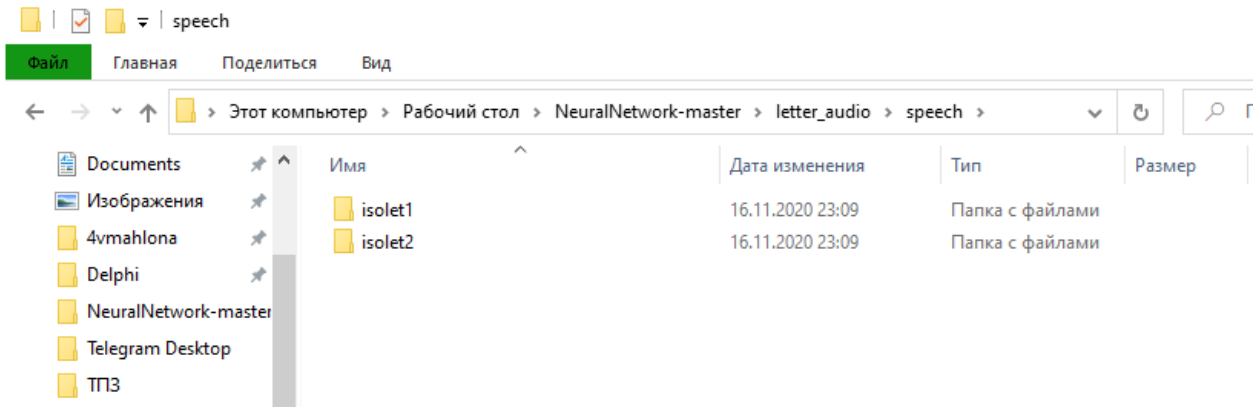
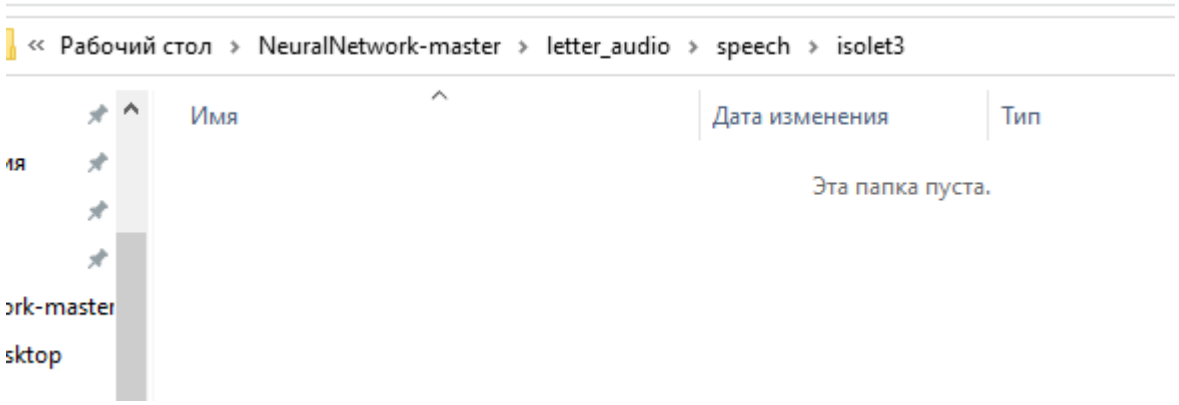


Рисунок В.5 – Структура програми розпізнавання вимовлених літер на основі спайкінгової нейронної мережі




```
C:\Users\maglo\AppData\Local\Programs\Python\Python38\python.exe C:/Users/maglo/Desktop/NeuralNetwork-master/SNN.py train
```

```
C:\Users\maglo\AppData\Local\Programs\Python\Python38\python.exe C:/Users/maglo/Desktop/NeuralNetwork-master/SNN.py train
Training...
2020-11-17 18:07:19.817517
letter_audio/speech/isolet1\-A-.wav
    A: 5
2020-11-17 18:13:20.876336

Process finished with exit code 0
```




```
C:\Users\maglo\AppData\Local\Programs\Python\Python38\python.exe C:/Users/maglo/Desktop/NeuralNetwork-master/SNN.py test
```

```
C:\Users\maglo\AppData\Local\Programs\Python\Python38\python.exe C:/Users/maglo/Desktop/NeuralNetwork-master/SNN.py test
Testing
letter_audio/speech/isolet3\-A-.wav
    A: 58

Process finished with exit code 0
```

Рисунок В.6 – Результати роботи програми розпізнавання вимовлених літер

Додаток Г (довідниковий)

Інструкція користувача

На комп’ютері має бути встановлений Python 3.8+ версії та пакети, які необхідні для запуску програми (якщо спробувати запустити програму, то пакети, які не встановлені, будуть вказані в терміналі – командою pip install name встановлюємо її). Також додатково має бути встановлений компілятор C++.

Тренування:

Для тренування потрібні вхідні дані (тобто аудіофайли у форматі .wav, де кожен файл – це окрема вимовлена буква (приклад: -Б-.wav)), їх потрібно помістити за шляхом ../letter_audio/speech/isolet1 або ../letter_audio/speech/isolet2 папках (див. рис.Г.1), які знаходяться в кореневому каталозі проекту (якщо папок немає, то потрібно їх створити).

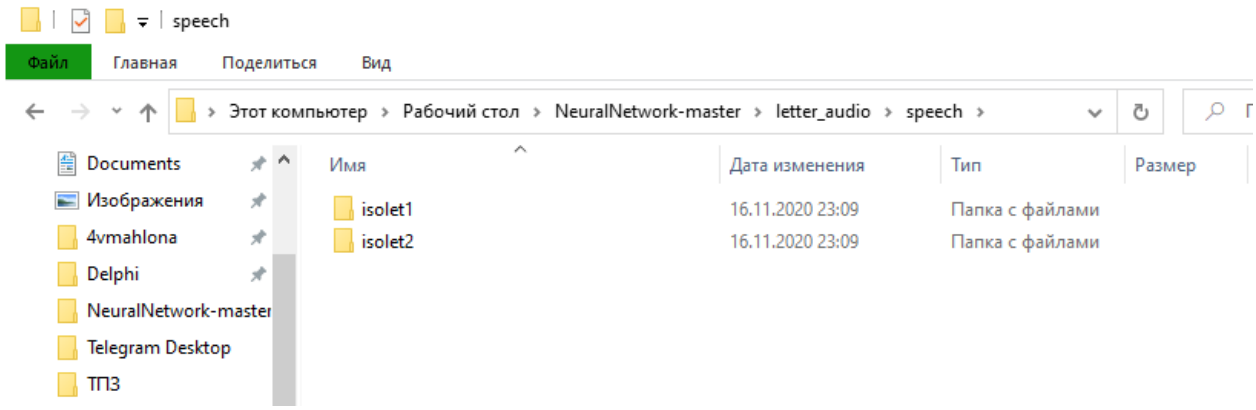


Рисунок Г.1 – Створення папок для навчальних даних

- Перейти до каталогу, що містить SNN.py
- Ввести за запустити команду python SNN.py train в терміналі

В результаті отримаємо час запуску навчання та час закінчення. Також буде виведено результати у вигляді кількості спайків для кожного з нейронів кожної букви та ваги нашої створеної мережі.

```
C:\Users\maglo\AppData\Local\Programs\Python\Python38\python.exe C:/Users/maglo/Desktop/NeuralNetwork-master/SNN.py train
Training...
2020-11-17 18:07:19.017517
letter_audio/speech/isolet1\-A-.wav
    A: 5
2020-11-17 18:13:20.076336

Process finished with exit code 0
```

Тестування:

Для тестування потрібні вхідні дані (тобто аудіофайли у форматі **.wav**, де кожен файл – це окрема вимовлена буква (приклад: **-A-.wav**)), їх потрібно помістити за шляхом ../letter_audio/speech/isolet3 папку (див. рис. Г.2), яка знаходиться в кореневому каталозі проекту (якщо папки немає, то потрібно її створити).

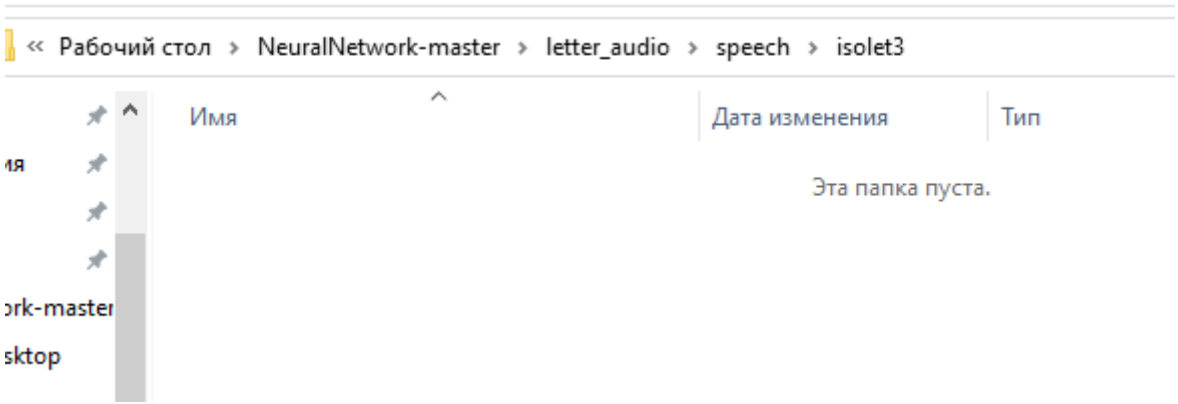


Рисунок Г.2 – Створення папок для тестових даних

- Перейти до каталогу, що містить SNN.py
- Ввести за запустити команду python SNN.py test в терміналі

```
C:\Users\maglo\AppData\Local\Programs\Python\Python38\python.exe C:/Users/maglo/Desktop/NeuralNetwork-master/SNN.py test
```

Отримуємо результати у вигляді кількості спайків для кожного з нейронів кожної букви та аналіз спайків (індекс та відстань).