НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри

_____ Оксана ТИМОЩУК

«___»_____21__ р.

Дипломна робота

на здобуття ступеня бакалавра за освітньо-професійною програмою «Системи і методи штучного інтелекту» спеціальності 122 «Комп'ютерні науки» на тему: «Розпізнавання мови з використанням спектрограм та глибинного навчання»

Виконав:
студент курсу, групи КА-76
Собко Іван Олександрович _____

Керівник:
асистент, Кухарєв Сергій Олександрович _____

Консультант з нормоконтролю:
канд.техн.наук, доцент Коваленко Анатолій Єпіфанович _____

Консультант з економічного розділу:
канд.економ.наук, доцент Рощина Надія Василівна _____

Рецензент:
кандидат техн. наук, доцент Безносик Олександр Юрійович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2021 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп’ютерні науки»

Освітня програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Оксана ТИМОЩУК
«26» травня 2021 р.

ЗАВДАННЯ
на дипломну роботу студенту
Собко Івану Олександровичу

1. Тема роботи «Розпізнавання мови з використанням спектрограм та глибинного навчання», керівник роботи Кухарєв Сергій Олександрович, асистент кафедри «ММСА» ННК «ІПСА», затверджені наказом по університету від «26» травня 2021 р. № 1344-с

2. Термін подання студентом роботи 8.06.2021.

3. Вихідні дані до роботи

4. Зміст роботи

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Надія Василівна, канд.економ.наук, доцент		

7. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Затвердження теми БДР	15.04.2021 - 21.04.2021	Виконано
2.	Ознайомлення зі структурою БДР згідно з Положенням про державну атестацію студентів НТУУ «КПІ»	15.04.2021 - 28.04.2021	Виконано
3.	Ознайомлення з ДСТУ 3008-95 та стандарти ЄСПД	21.04.2021 - 28.04.2021	Виконано
4.	Проведення дослідження за темою БДР під керівництвом керівника	28.04.2021 - 05.05.2021	Виконано
5.	Завершення роботи над першим варіантом частини БДР	05.05.2021 -1 2.05.2021	Виконано
6.	Проведення роботи над експериментальною частиною БДР	05.05.2021 - 19.05.2021	Виконано
7.	Проведення роботи над програмним продуктом	19.05.2021 - 26.05.2021	Виконано
8.	Оформлення БДР та аналіз отриманих результатів	19.05.2021 - 26.05.2021	Виконано

Студент І. О. Собко

Керівник С. О. Кухарєв

*Якщо визначені консультанти. Консультантом не може бути зазначено керівника дипломної роботи

РЕФЕРАТ

Дипломна робота: 91 с., 24 рис., 6 табл., 2 додатки, 18 джерел.

НЕЙРОННІ МЕРЕЖІ, РОЗПІЗНАВАННЯ, СПЕКТРОГРАМИ, РЯДИ ФУР’Є, РОБОТА З АУДІО.

Об’єкт дослідження – класифікація мови

Мета роботи – проаналізувати існуючі рішення для розпізнавання мови та розробити власне рішення розпізнавання слів та мови.

Технологія розпізнавання мови - це те, про що мріяли і над яким працювали десятки років. Однак для всіх сучасних технологічних досягнень голосове управління було досить складною справою. Результат цього дослідження допоможе краще «зрозуміти» методи розпізнавання мови.

У цьому документі розглянута проблема сегментація аудіо сигналу з подальшою класифікацією коротких-слів команд. Розпізнавання слів є важливим кроком до розпізнавання мови. Класифікація різних слів є основою технології розпізнавання мови, яка дає можливість робити базове передбачення сказаного.

Метою дипломної роботи є розробка системи класифікації аудіо сигналів, за допомогою трансформування сигналу в ряд Фур’є та глибоких нейронних мереж.

ABSTRACT

Bachelor thesis: 91 p., 24 fig., 6 tabl., 2 append., 18 sources.

NEURAL NETWORKS, RECOGNITION, SPECTROGRAMS, FOURIER SERIES, AUDIO WORK.

The object of research is the classification of language

The aim of the work is to analyze the existing solutions for language recognition and to develop one's own solution for word and language recognition. Language recognition technology is something we have dreamed of and worked on for decades. However, for all modern technological advances, voice control was quite a difficult task. The result of this study will help to better "understand" language recognition methods.

This paper discusses the problem of audio signal segmentation with subsequent classification of short-word commands. Word recognition is an important step towards language recognition. Classification of different words is the basis of language recognition technology, which makes it possible to make a basic prediction of what is said.

The aim of the thesis is to develop a system for classifying audio signals by transforming the signal into a Fourier series and deep neural networks.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ASR - Automatic Speech Recognition

SR - Speech Recognition

HMM - Hidden Markov Model

DTW - Dynamic Time Warping

ANN - Artificial Neural Network

WER - Word Error Rate

SWER - Single Word Error Rate

CSR - Command Success Rate

STFT - Short-Time Fourier Transform

CNN - Convolutional Neural Network

VAD - Voice Activity Detector

ВСТУП

Технологія збільшила швидкість передачі даних які обробляються машинами в нашому житті. Ці машини включають комп'ютери, телевізори, мікрохвильові печі та багато інших. Зі збільшенням швидкості обробки даних машини починають чекати на введення даних людини, а не люди, які чекають відповіді машин.

Сьогодні існує безліч технологій введення, які включають блоки клавіатур (клавіатури, пульти дистанційного керування або матриці з мембранними перемикачами) та вказівні пристрої (миші, джойстики або цифрові планшети). Однак більшість з цих пристроїв мають повільний ввід даних. Отже, оскільки машини отримують функції та потужність обробки даних, вони витрачають менше часу на роботу та більше часу на очікування людського вкладу. Однією з технологій, яка існує вже деякий час, але все ще застосовується обмежено, є автоматичне розпізнавання мови (ASR).

Ця робота вивчає відомі техніки розпізнавання мови (зокрема, методи створення спектрограм та глибоке навчання) та застосовує нову обчислювальну потужність та зберігання даних для виконання цих методів. У цьому проекті було зроблено спробу розробити систему розпізнавання мови, яка має невеликі розміри, економічна і може швидко виконувати розпізнавання, щоб її можна було використовувати в реальному часі.

Пояснювальна записка складається з чотирьох розділів. У першому розділі проводиться огляд актуальності задачі, огляд вхідних даних для аналізу та здійснюється постановка задачі. У другому розділі розглянуто ряд математичних моделей, за допомогою яких відбувається класифікація та описуються методи алгоритму. Третій розділ описує архітектуру розробленої програми, також у ньому аналізуються результати роботи алгоритму. Четвертий розділ являє собою економічну частину, в якій розробляється кошторис витрат на розробку.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність задачі

Ближче до кінця XX століття системи ASR знайшли широке застосування в комп'ютеризованих іграх та іграшках, контролерів різних інструментів, збір даних та диктантах. Ця функція також виявилася дуже корисною для тих, хто не може користуватися клавіатурами, та серед тих, хто має певні вади. Голосові асистенти, які встановлені на останніх смартфонах, є одним з найвидатніших прикладів мобільного голосового інтерфейсу і демонструє вплив розпізнавання мови в сучасному суспільстві. Далі показано деякі наслідки розвитку розпізнавання мовлення в суспільстві [1]:

- Вплив на галузь охорони здоров’я - ця функція використовується у медичній звітності медичним персоналом.
- Використання при наданні послуг - клієнти можуть не хотіти розмовляти з оператором, тому вони можуть використовувати системи розпізнавання мови.
- Автоматизована ідентифікація - щоб уникнути надання конфіденційної та ризикованої особистої інформації, установи можуть вибрати розпізнавання мовлення для автентифікації особи своїх клієнтів.

Зростаюче розповсюдження голосових пристроїв, зростаючий попит споживачів на розумні пристрої та інтеграція голосових інформаційних та розважальних систем є ключовими факторами, що сприяють зростанню ринку розпізнавання мови та голосу. Однак висока вартість голосових або мовленневих смарт-пристроїв є одним з основних факторів, що стримують зростання цього ринку.

Зростаюче використання штучного інтелекту в автомобілях, охороні здоров’я та побутовій електроніці збільшило попит на голосові пристрої. Прогнозується, що світовий ринок розпізнавання мови та голосу, який у

2018 році оцінювався у 6,9 млрд. доларів США, становитиме 19,8% CAGR (Compound annual growth rate) і досягне 28,3 млрд. доларів до кінця 2026 року [2]. У 2018 році 9,5 мільйона людей у Великобританії використовували розумну колонку, що на 98,6% більше порівняно з 2017 роком [3]. Більше того, зростаючий попит на голосові додатки в пристроях, включаючи розумні колонки, побутову електроніку, гостинність, розумні носимі пристрої, підключені автомобілі, розумний дім та охорону здоров'я, є одним із ключових факторів, що рухають на ринку розпізнавання голосової мови.

1.2 Опис наявних даних для аналізу

Набір даних був сформований у рамках змагання «TensorFlow Speech Recognition Challenge» на сайті http://kaggle.com/. Набір даних включає в себе 8 коротких слів-команд та містить по 1000 аудіо записів на кожне слово-команду, сказані різними людьми у різних умовах.

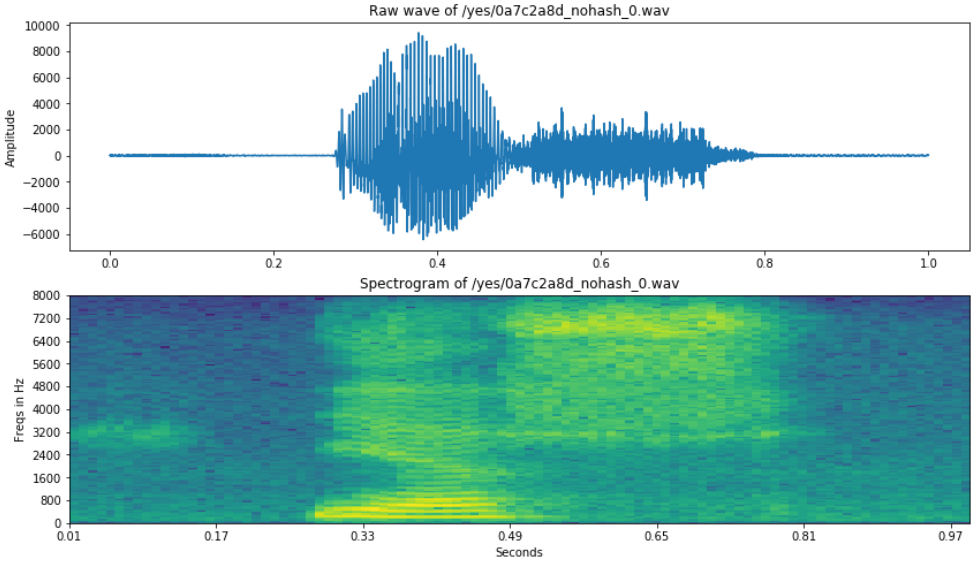


Рисунок 1.1 - Візуалізація слова "yes"

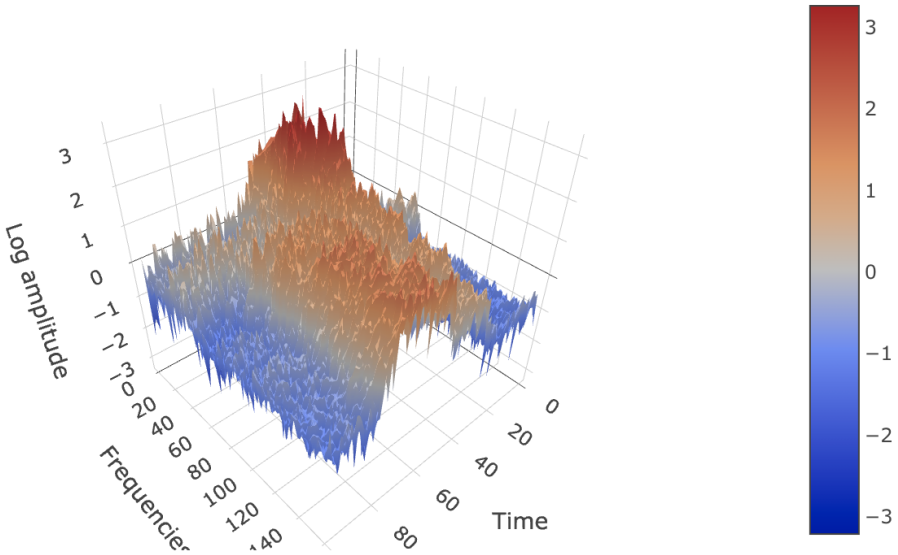


Рисунок 1.2 - 3D спектрограма слова “yes”

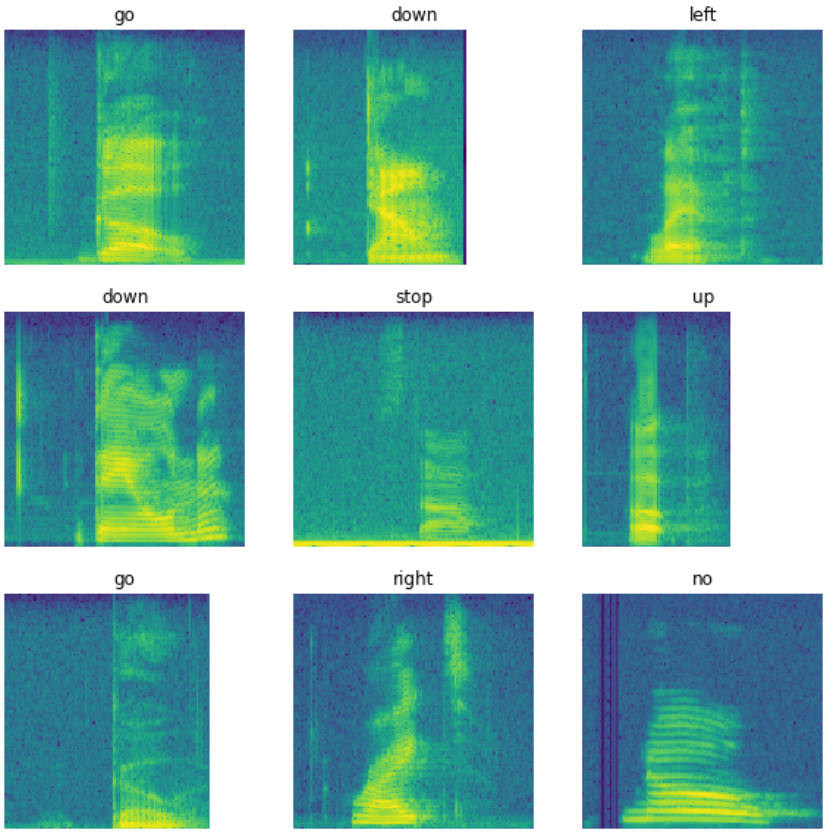


Рисунок 1.3 - Спектрограми усіх слів

Цей набір даних найкраще підходить для розробки та тестування системи розпізнавання мови, оскільки він легкий (набір даних важить лише 0.25 гігабайту), а отже майбутню модель буде легко навчити.

1.3 Проблеми та задачі розпізнавання мови

Основна задача розпізнавання мови - це, як не дивно, розпізнавання мови. ASR системи можуть будуватися на різних фундаментальних рівнях - розпізнавання фонем, слів або речень. Кожна система має розпізнати свою групу звуків.

В останні 5-10 років розвитку розпізнавання мови, переважна концентрація була мінімізації помилок під час оброблення звуку [4]. Сучасні проблеми розпізнавання мови зумовлені двома основними факторами - охопленням та гучним середовищем. Це вимагає ще більш точних систем, які можуть вирішити найамбітніші випадки використання ASR.

Окрім цього, розпізнавання мовлення має бути доступним для більшої кількості мов та охоплювати широкі теми. Оскільки зараз ASR потребує великої кількості даних, щоб добре працювати, а деякі з них просто не зібрані для певних мов та тем. Без їх додавання системи ASR залишаться помітно обмеженими.

Сучасні проблеми розпізнавання мовлення дуже різні, та потребують індивідуального підходи для вирішення. Серед таких проблем можно виділити:

- Неточність та хибні інтерпретації. Програмне забезпечення для розпізнавання мови не завжди може правильно інтерпретувати вимовлені слова. Це пов’язано з тим, що комп’ютери не співпадають з людьми, розуміючи контекстне відношення слів і речень, спричиняючи неправильне тлумачення того, що мовець мав сказати чи досягти.

- Час і недостатня ефективність систем ASR. Зазвичай ми вважаємо, що комп’ютеризація процесу пришвидшує його. На жаль, не завжди це стосується систем розпізнавання голосу. У багатьох випадках використання голосової програми займає більше часу, ніж використання традиційної текстової версії.
- Акценти та місцеві відмінності. Акценти можуть становити великий проблему для розпізнавання мови. Хоча системи покращуються, все одно існує велика різниця в їх здатності розуміти американську чи шотландську англійську. Навіть проста застуда може бути причиною для голосових команд не працювати, так як зазвичай.
- Фоновий шум і гучне середовище. При використанні систем ASR у гучному середовищі, частіше за все будуть неточності та помилки. Особливо ускладнює ефективне їх використання в міських умовах на відкритому повітрі або у великих громадських приміщеннях / офісах. Використовуючи певні мікрофони або гарнітури, обмеження можна зменшити, але для цього потрібен додатковий пристрій, що ніколи не бажано.

1.4 Постановка задачі

Для початку визначимо фундаментальну одиницю, на основі якої ми будемо будувати систему розпізнавання мови. Нам потрібно класифікувати певні короткі слова: up, down, right, left, stop, go, yes, no. В рамках даної задачі краще за все за фундаментальну одиницю взяти слова. В умовах розпізнавання речень найкращим вибором було б будувати систему на основі фонем, оскільки в кожній мові кількість фонем обмежено, а слів багато і літературний запас весь час поповнюється.

Далі, сира мова повинна бути спочатку опрацьована і стиснена, щоб спростити подальшу обробку. Доступно багато методів аналізу сигналів, які

можуть виділити корисні функції та стиснути дані в десять разів не втрачаючи жодної важливої інформації. Серед найпопулярніших:

- За допомогою Фур'є-аналізу або швидкого перетворення Фур'є (ШПФ), яке дає дискретні частоти з часом, які можна інтерпретувати візуально, у вигляді спектрограм. Частоти часто розподіляються за шкалою Мела, яка є лінійною у низькому діапазоні, але логарифмічна у високому діапазоні, що відповідає фізіологічним характеристикам людського вуха [5].
- Перцептивне лінійне прогнозування (PLP) - ця методика використовує три концепції з психофізики слуху, щоб отримати оцінку слухового спектру: (1) спектральна роздільна здатність критичної смуги, (2) крива рівної гучності та (3) закон потужності інтенсивності-гучності, але дає коефіцієнти, які не можна інтерпретувати візуально [5].
- Лінійне інтелектуальне кодування (LPC) - це один з найпотужніших методів аналізу мовлення та один з найкорисніших методів кодування мови високої якості з низькою швидкістю передачі даних і забезпечує дуже точні оцінки параметрів мови [5].

Для нашої задачі ми виберемо швидке перетворення Фур'є, оскільки воно дуже ефективне в сенсі часу, що дозволить нам розробити швидку систему, а також тому що це дозволить створювати спектрограми, які надалі допоможуть нам із створенням моделі розпізнавання мови. В рамках цього завдання необхідно:

- Побудувати систему попередньої обробки.
- Проаналізувати отримані дані.
- Побудувати нейронну модель
- Проаналізувати результати моделі
- Створити звіт про якість отриманих результатів.

Для досягнення даних цілей необхідно вирішити наступні завдання:

- Проаналізувати існуючі підходи, для обробки сигналів.
- Виділити окремі методи, які доцільно використовувати для розпізнавання мови.
- Створити програмний продукт, на базі отриманих алгоритмів.
- Проаналізувати результати роботи програми.

1.5 Висновки до розділу

Отже, у даному розділі була показана актуальність задачі розпізнавання мови та її перспективи. Також було проведено опис даних, з детальним поясненням файлів, проведено аналіз предметної області. Було показано усі проблеми та виклики, через які сфера розпізнавання мови досі не досягла своєї вершини В кінці розділу поставили основну задачу, вимоги та обмеження до системи, яка проектується.

РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ РОБОТИ

2.1 Дослідження існуючих підходів до вирішення задачі

В основному існує 3 алгоритми, які використовуються для ASR. Вони наведені нижче:

а) Прихована модель Маркова (HMM)

б) Динамічне викривлення часу (DTW)

в) Штучні нейронні мережі (ANN)

а) Прихована марковська модель (HMM)

Прихована марковська модель - це статистична марковська модель, в якій модель, що моделюється, вважається марковським процесом з неспостережуваними (прихованими) станами. HMM можна представити як найпростішу динамічну байєсівську мережу. Прихована модель Маркова - це сукупність станів, пов’язаних між собою переходами, як показано на рис. 2.1.

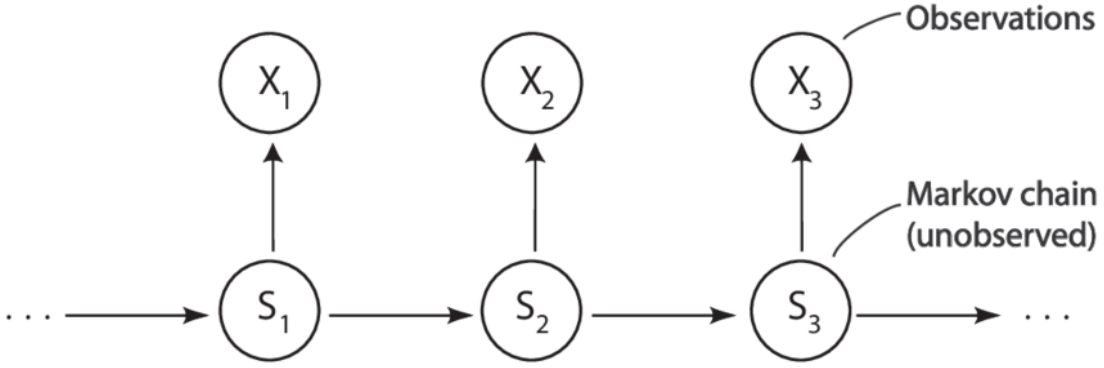


Рисунок 2.1 - Приклад структури прихованої марковської моделі

Вона починається у призначеному початковому стані. На кожному дискретному кроці часу перехід переходить у новий стан, а потім у цьому

стані генерується один вихідний символ. Вибір символу переходу та вихідного сигналу є випадковим і регулюється розподілом ймовірностей. HMM можна розглядати як чорний ящик, де спостерігається послідовність вихідних символів, що генеруються з часом, але послідовність станів, відвіданих з часом, прихована від очей. Ось чому її називають прихованою марковською моделлю. HMM мають різноманітне застосування. Коли HMM застосовується до розпізнавання мови, стани інтерпретуються як акустичні моделі, вказуючи, які звуки, ймовірно, будуть чути під час відповідних їм сегментів мови; тоді як переходи забезпечують часові обмеження, вказуючи, як стани можуть послідовно слідувати один за одним. Оскільки мова завжди рухається вперед у часі, переходи в мовленнєвій програмі завжди йдуть вперед. [6]

Існує три основних алгоритми, пов’язані з прихованими моделями Маркова:

- Forward алгоритм, корисний для ізольованого розпізнавання слів;
- Алгоритм Вітербі, корисний для безперервного розпізнавання мови;
- Forward-backwards алгоритм, корисний для навчання HMM.

Проте HMM мають деякі слабкі сторони, які обмежують їх застосування:

- Алгоритм Вітербі дорогий, як з точки зору пам'яті, так і часу обчислення. Для послідовності довжиною *n* динамічне програмування для пошуку найкращого шляху через модель із s-станами та e-ребрами займає пам'ять, пропорційну *sn* та час, пропорційний *en*. Для пошукових запитів пошук із прихованою моделлю Маркова приблизно в 10 разів повільніший, ніж використання простої моделі Маркова - для більших моделей (необхідні для довших цільових послідовностей) штраф збільшиться. Інші алгоритми прихованих марковських моделей, такі як алгоритм прямого і зворотного руху, ще дорожчі.

- HMM потрібно навчити набору seed sequence та, як правило, вимагає більшого seed, ніж прості моделі Маркова. Навчання передбачає багаторазові повторення алгоритму Вітербі, які можуть бути досить повільними.

б) Динамічне викривлення часу (DTW)

Найпростіший спосіб розпізнати ізольований зразок слова - порівняти його з кількістю збережених шаблонів слів та визначити, який "найкращий збіг".

Ця мета ускладнюється низкою факторів. По-перше, різні зразки даного слова матиме дещо іншу тривалість. Цю проблему можна усунути, просто нормалізувавши шаблони та невідому мову, щоб усі вони мали однакову тривалість. Однак інша проблема полягає в тому, що темп мови не може бути постійним протягом усього слова; іншими словами, оптимальне вирівнювання між шаблоном і зразком мовлення може бути нелінійним.

Динамічне викривлення часу (Dynamic Time Warping) є ефективним методом пошуку цього оптимального нелінійного вирівнювання. Динамічне викривлення часу - це добре відома техніка інтуїтивного пошуку оптимального вирівнювання між двома заданими (залежними від часу) послідовностями за певних обмежень; послідовності викривляються нелінійно, щоб збігатися між собою. Спочатку DTW використовувався для порівняння різних моделей мови при автоматичному розпізнаванні мови. У таких сферах, як видобуток даних та пошук інформації, DTW успішно застосовано для автоматичного подолання деформацій у часі та різних швидкостей, пов'язаних із даними, що залежать від часу.

При аналізі часових рядів динамічне перекручування часу (DTW) - це алгоритм вимірювання подібності між двома часовими послідовностями, які можуть змінюватися за часом або швидкістю. Наприклад, подібність

моделей ходьби можна виявити за допомогою DTW, навіть якщо одна людина йшла швидше, ніж інша.[6]

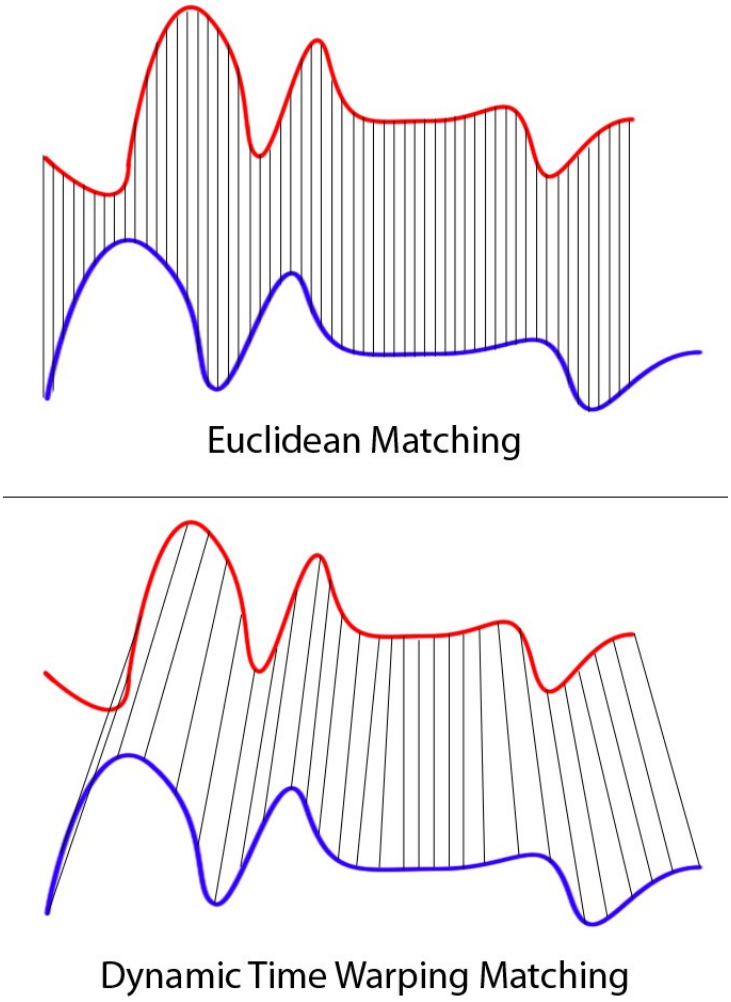


Рисунок 2.2 - Порівняння часових рядів за допомогою алгоритму Евкліда та алгоритму DTW.

У контексті розпізнавання мови алгоритм DTW мають такі недоліки:

- Він характерний тільки для конкретного мовця.
- Він може давати патологічні результати. Найважливішим моментом є те, що алгоритм може спробувати пояснити мінливість по осі Y шляхом викривлення осі X. Це може призвести до не інтуїтивних вирівнюванню, коли одна точка одного часового ряду накладається на велику ділянку іншого часового ряду. Це може призвести до не

інтуїтивних вирівнювання, коли одна точка одного часового ряду накладається на велику ділянку іншого часового ряду.

- Їх недолік полягає в тому, що вони можуть перешкодити знайти "правильне" викривлення. У змодельованих випадках правильне викривлення можна дізнатися, скрививши тимчасової ряд і спробувавши видалити оригінал.
- Додаткова проблема DTW полягає в тому, що алгоритм може не знайти очевидні природні вирівнювання в двох послідовностях тільки тому, що характеристика (тобто пік, долина, точка перегину, плато і т.д.) в одній послідовності трохи вище або нижче, ніж відповідна характеристика в іншій послідовності.
- Іншим основним недоліком DTW є збільшення простору пошуку для безперервних завдань розпізнавання і низька ефективність незалежності від диктора.
- Однією з основних проблем в системах розпізнавання мови на основі динамічного викривлення часу є підготовка надійних еталонних шаблонів для набору слів, що підлягають розпізнаванню.

в) Штучні нейронні мережі (ANN)

ANN - це біологічні комп’ютерні програми, призначені для імітації способу обробки мозку людини інформацією. ANN збирають свої знання, виявляючи закономірності та взаємозв'язки в даних, і навчаються (або навчаються) на основі досвіду, а не з програмування, і в цьому полягає основна різниця між ANN та іншими класичними комп'ютерними програмами. Ще однією суттєвою відмінністю програмного забезпечення ANN від інших комп'ютерних програм є те, що алгоритми, що використовуються для аналізу даних, є гнучкими. Вони можуть бути змінені в будь-який час під час аналізу. Відмінною особливістю ANN є їх здатність ефективно вирішувати багатовимірні проблеми, включаючи кілька тисяч функцій. ANN формується з сотень одиниць, тобто штучних нейронів або

процесорних елементів, пов'язаних з коефіцієнтами (вагами), які складають нейронну структуру і організовані шарами.

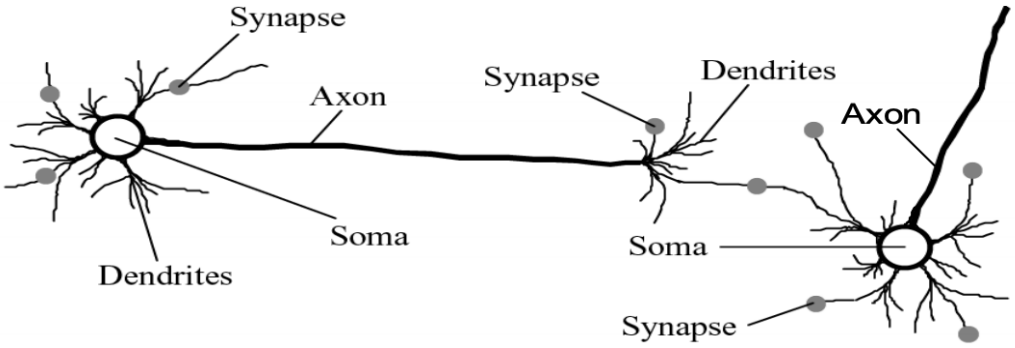


Рисунок 2.3 - Біологічна нейронна мережа

Здатність нейронних обчислень походить від з'єднання нейронів у мережі. Чим краще нейрони пов'язані в мережах, тим краще прогнозування як вихід. Активність нейронної мережі визначається передавальними функціями її нейронів, правилом навчання та самою архітектурою. Досягнення успішного результату досліджень ANNs залежить від мінімізації похибки прогнозування шляхом оптимізації міжз’єднаних зв’язків під час навчання. Роблячи це як спроби та помилки, мережа досягає заданого рівня точності. Після того, як мережа пройде навчання з мінімальною помилкою прогнозування та буде протестована, її можна буде використовувати з новою вхідною інформацією для прогнозування результату. Інформація в ANN кодується на основі міцності „синаптичних” зв’язків мережі. Останні дослідження щодо ANN зосереджені в основному на розробці нових типів мереж шляхом зміни передавальної зв'язку нейронів, зміни правила навчання та ініціювання нової формули зв'язку.[7]

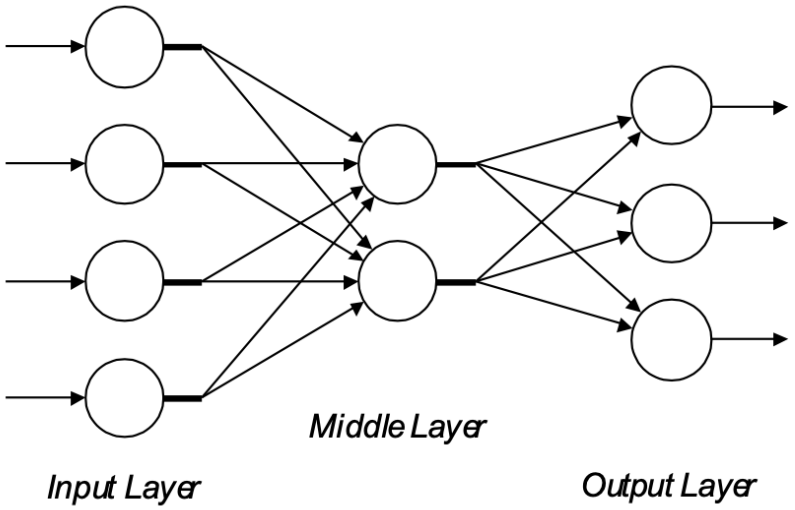


Рисунок 2.4 - Структура штучної нейронної мережі

Переваги ANN у контексті розпізнавання мови:

- ANN - це дуже нелінійне моделювання, що дозволяє гнучко підлаштовувати її під різні задачі.
- ANN - це нелінійна модель, яка проста у використанні та розумінні порівняно зі статистичними методами.
- ANN - це непараметрична модель, тоді як більшість статистичних методів - це параметрична модель, яка потребує вищого рівня статистики.
- Алгоритм навчання ANN із зворотним поширенням (Back propagation) широко використовується при вирішенні різних класифікацій та прогнозуванні проблеми. Хоча збіжність BP відбувається повільно, але вона гарантована. [8]
- Нейронні мережі пропонують ряд переваг, включаючи потребу в менш офіційній статистичній підготовці, можливість неявного виявлення складних нелінійних взаємозв’язків між залежними та незалежними змінними, здатність виявляти всі можливі взаємодії між змінними-предикторами та наявність множинних алгоритмів навчання [9].

Раніше, до стрімкого розвитку глибинного та машинного навчання, основними алгоритма у системах ASR були моделі Маркова та динамічне викривлення часу. Проте, на сьогоднішній день для задач ASR ANN є ефективним та дієвим способом, оскільки він має багатошарову мережу. Розпізнавання мови також використовується в смартфонах, де мовлення / вимовлені слова подаються як вхідні дані, а SR надає відповідний пошук або інформацію, яку користувач хоче отримати на виході. Нейронні мережі з їх надзвичайною здатністю виводити значення зі складних або неточних даних можна використовувати для вилучення закономірностей та виявлення тенденцій, які є занадто складними, щоб їх можна було помітити ні людям, ні іншим комп’ютерним методами. Навчену нейронну мережу можна вважати "експертом" у категорії інформації, яку вона отримала для аналізу. ANN має:

- Адаптивне навчання: здатність навчитися виконувати завдання на основі даних, наданих для навчання або початкового досвіду.
- Самоорганізація: АНН може створити власну організацію або представити інформацію, яку отримує під час час навчання.
- Операції в режимі реального часу: обчислення ANN можуть виконуватися паралельно, а також проводяться спеціальні апаратні пристрої розроблені та виготовлені, які використовують цю можливість.
- Толерантність до несправностей через надмірне кодування інформації: Часткове руйнування мережі призводить до відповідного погіршення продуктивності. Однак деякі можливості мережі можуть зберігатися навіть при серйозних пошкодженнях мережі.

Таким чином, для розпізнавання мови штучна нейронна мережа є найефективнішим та найдієвішим алгоритмом серед усіх алгоритмів.

2.2 Критерії якості вирішення задачі

Ефективність систем розпізнавання мови зазвичай оцінюється з точки зору точності та швидкості. Зазвичай точність оцінюється коефіцієнтом помилок слів (WER), тоді як швидкість вимірюється коефіцієнтом реального часу. Інші показники точності включають коефіцієнт помилок одного слова (SWER) та коефіцієнт успішності команд (CSR). [10] Однак розпізнавання мови (машиною) є дуже складною проблемою. Вокалізація варіюється з точки зору акценту, вимови, артикуляції, шорсткості, носальності, висоти, гучності та швидкості. Мова спотворюється фоновим шумом і відлунням, електричними характеристиками. Точність розпізнавання мови залежить від наступного: [11]

- Розмір словника та плутанина
- Залежність оратора від незалежності
- Ізольована, розривна або безперервна мова
- Завдання та мовні обмеження
- Читання проти спонтанного мовлення
- Несприятливі умови

2.3 Алгоритм розв’язку задачі

Алгоритм розпізнання мови складається з декількох частин. Грубо кажучи їх можно поділити на дві частини:

а) Алгоритм обробки сигналу

б) Алгоритм розпізнавання сигналу

а) Алгоритм обробки сигналу відповідає за трансформацію сигналу у той вигляд, який буде приймати алгоритм розпізнавання сигналу. У попередньому пункті ми визначились, що найкращим алгоритмом для вирішення задачі розпізнавання мови буде ANN або штучні нейронні

мережі. Отже, для початку нам потрібно вирішити як ми будемо “годувати” інформацію нашій мережі.

б) Для того, щоб розпізнавати сигнали, нам потрібно перетворити їх у щось, що зможе точно відображати характеристики аудіо-запису. Для цього нам будуть потрібні спектрограми.

Спектрограма - це візуальний спосіб представлення сили сигналу або “гучності” сигналу в часі на різних частотах, присутніх у певній формі хвилі. Можна не тільки побачити, чи є енергія більшою чи меншою, наприклад, 2 Гц проти 10 Гц, але можна також побачити, як рівні енергії змінюються з часом.

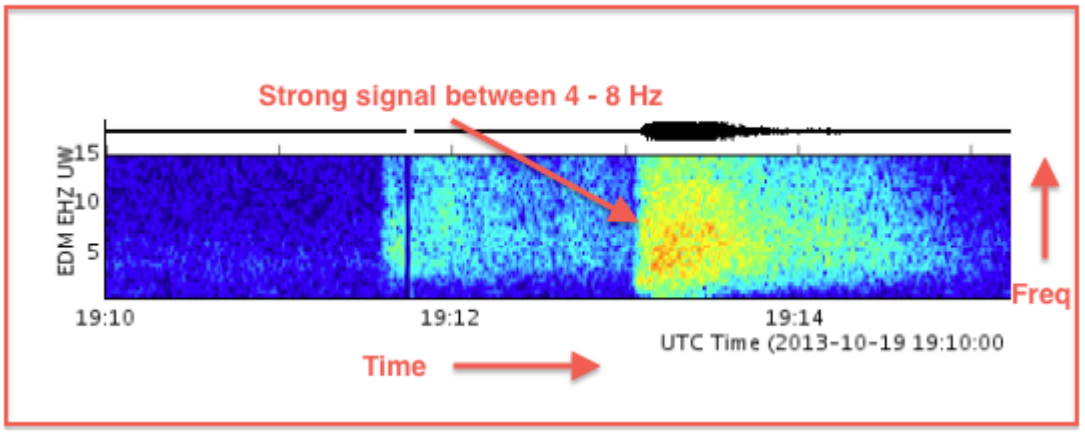


Рисунок 2.5 - Приклад спектрограми та її інтерпретації

Створення спектрограми за допомогою швидкого перетворення Фур’є - це цифровий процес. Дані цифрової вибірки у часовій області розбиваються на фрагменти, які зазвичай перекриваються, і перетворенням Фур'є трансформується для обчислення величини частотного спектру для кожного фрагмента. Потім кожен шматок відповідає вертикальній лінії на зображенні. Потім ці спектри або часові графіки «кладуть поруч», щоб сформувати зображення або тривимірну поверхню [12], або дещо перекриваються різними способами, тобто вікнами. Цей процес по суті відповідає обчисленню квадрата величини віконного перетворення Фур'є (STFT) сигналу $s(t)$, для ширини вікна w,

$$spectrogram(t,\ w)\ =\ |STFT(t,w)|^2.$$ [13]

Короткочасне перетворення Фур'є (STFT) - це послідовність перетворень Фур'є віконного сигналу. STFT забезпечує інформацію про частоту, локалізовану в часі, для ситуацій, коли частотні складові сигналу змінюються з часом, тоді як стандартне перетворення Фур'є забезпечує інформацію про частоту, усереднену за весь інтервал часу сигналу. Пара STFT задана як

$$\begin{cases} X_{STFT}\,[m,n] = \sum_{k=0}^{L-1} x\,[k]\,g\,[k-m]\,e^{-j2\pi nk/L} \\ x\,[k] = \sum_m \sum_n X_{STFT}\,[m,n]\,g\,[k-m]\,e^{j2\pi nk/L} \end{cases}$$

де x[k] - сигнал;

g[k] - функція вікна точки L.

З цього визначення STFT значення x[k] можна інтерпретувати як перетворення Фур’є добутку x [k]*g[k – m]. Рисунок 6 ілюструє обчислення STFT, беручи перетворення Фур'є віконного сигналу.

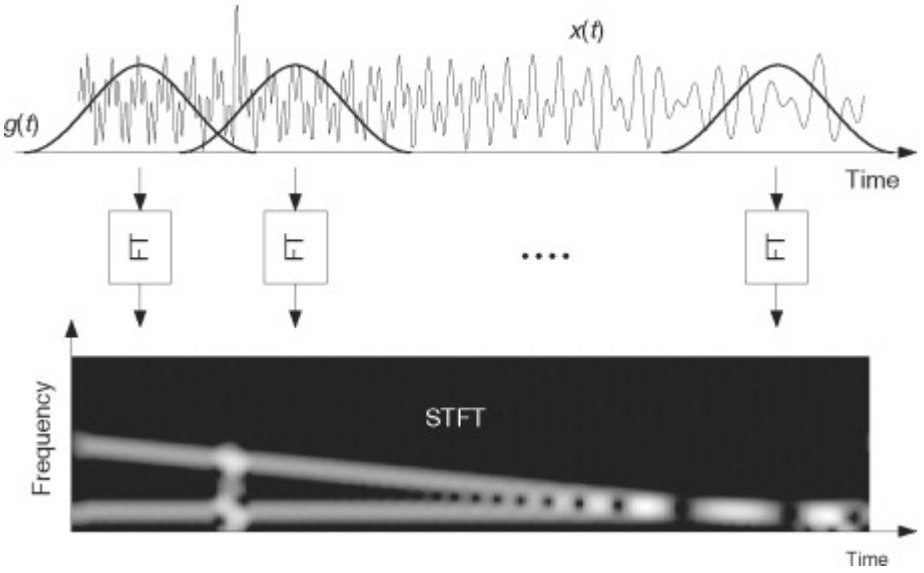


Рисунок 2.6 - Короткочасне перетворення Фур'є

Існує компроміс між часом та роздільною здатністю частоти в STFT. Іншими словами, хоча вікно вузької ширини призводить до кращої роздільної здатності у часовій області, воно породжує низьку роздільну здатність у частотній області, і навпаки.

Тепер, ми можемо надати нове визначення спектрограмми як графік інтенсивності величини STFT з часом.

Ми визначились з тим якими даними буде оперувати наша штучна мережа. Тепер необхідно зрозуміти яку саме нейронну мережу нам потрібно вибрати. Правильний вибір нейронної мережі для досягнення бажаного результату відіграє надзвичайно велику роль. Головна мета нашої роботи це класифікація зображення спектрограми. Для цього найкраще підходить мережа із згортковою архітектурою CNN(Convolution Neural Network).

CNN - це клас глибоких нейронних мереж, які можуть розпізнавати та класифікувати певні особливості зображень і широко використовуються для аналізу візуальних зображень. Їх застосування варіюється від розпізнавання зображень та відео, класифікації зображень, аналізу медичних зображень, комп’ютерного зору та обробки природної мови.[14]

Термін "convolution" в CNN позначає математичну функцію згортки, яка є особливим видом лінійної операції, коли дві функції множаться, утворюючи третю функцію, яка виражає, як форма однієї функції змінюється іншою. Простіше кажучи, два зображення, які можна представити у вигляді матриць, множать, щоб отримати вихідний результат, який використовується для вилучення рис із зображення.[15]

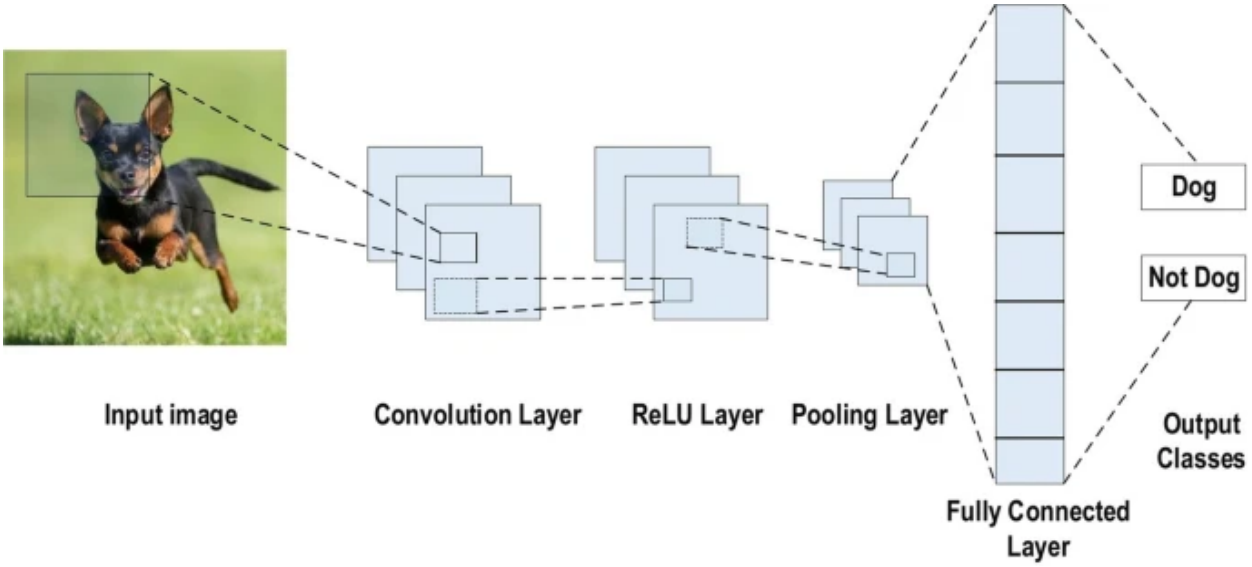


Рисунок 2.7 - Приклад архітектури CNN

Архітектура CNN складається з ряду шарів (або так званих багато компонентних блоків):

1. Convolutional Layer: У архітектурі CNN найбільш значущим компонентом є згортковий шар. Він складається з набору згорткових фільтрів (так званих ядер). Вхідне зображення, виражене у вигляді N-мірних метрик, складається з цими фільтрами для створення вихідної карти функцій.
    a. Ядро: Сітка дискретних чисел або значень описує ядро. Кожне значення називається вагою ядра. Випадкові числа призначаються як ваги ядра на початку навчального процесу CNN. Крім того, існує кілька різних методів, що використовуються для ініціалізації ваг. Далі ці ваги коригуються

в кожну епоху тренувань; таким чином, ядро вчиться виділяти суттєві особливості.

b. Згорткова операція: Спочатку формат вводу CNN описується. Векторний формат є входом традиційної нейронної мережі, тоді як багатоканальне зображення - входом CNN. Наприклад, одноканальний - це формат зображення в сірому масштабі, тоді як формат RGB - триканальний. Щоб зрозуміти згорткову операцію, давайте візьмемо приклад 4×4 піксельне зображення в сірому масштабі з ядром, що ініціюється випадковою вагою 2×2. Спочатку ядро ковзає по всьому зображенню по горизонталі та вертикалі. Крім того, визначається точковий добуток між вхідним зображенням і ядром, де їх відповідні значення множаться, а потім підсумовуються, створюючи єдине скалярне значення, обчислене одночасно. Потім весь процес повторюється, доки подальше ковзання неможливе. Кінцевий результат після підсумовування отриманих значень товару представляє значення входу на вихідну карту функцій.

2. Pooling Layer: Основним завданням шару об'єднання є субодискретизація карт об'єктів. Ці карти генеруються шляхом дотримання згорткових операцій. Іншими словами, такий підхід зменшує великі карти функцій, щоб створювати менші карти об’єктів. Одночасно він зберігає більшість домінуючої інформації (або особливостей) на кожному етапі етапу об'єднання. Подібно до згорткової операції, як крок, так і ядро спочатку призначаються за розміром перед тим, як буде виконана операція об'єднання. Кілька типів методів об'єднання доступні для використання в різних шарах об'єднання. Ці методи включають об'єднання дерев, об'єднане об'єднання, середнє об'єднання, мінімальне об'єднання, максимальне об'єднання, загальне середнє об'єднання (GAP) та глобальне максимальне об'єднання. Найбільш звичними та часто

використовуваними методами об'єднання є об'єднання max, min та GAP. Рисунок 8 ілюструє ці три операції об’єднання.

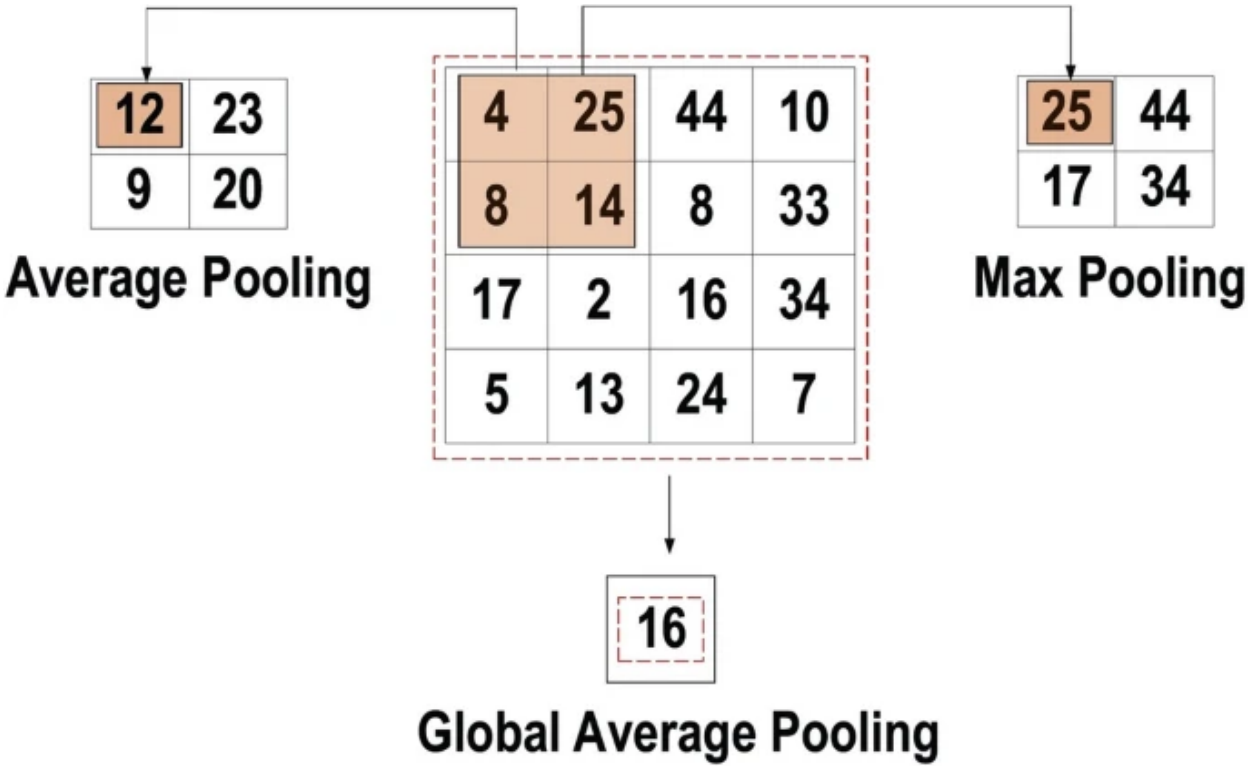


Рисунок 2.8 - Приклади операцій pooling layer

Іноді загальна продуктивність CNN в результаті знижується; це являє собою основний недолік шару об'єднання, оскільки цей рівень допомагає CNN визначити, чи є певна функція доступна на конкретному вхідному зображенні, але зосереджується виключно на визначенні правильного розташування цієї функції. Таким чином, модель CNN упускає відповідну інформацію.

3. Функція активації - відображення вхідних даних на виході є основною функцією всіх типів функції активації у всіх типах нейронної мережі. Вхідне значення визначається шляхом обчислення зваженого підсумовування вхідного нейрона разом з його зміщенням (якщо воно є). Це означає, що функція активації приймає рішення щодо запуску нейрону з посиланням на певний вхід шляхом створення відповідного виходу.

Нелінійні активаційні шари використовуються після всіх шарів з вагами в архітектурі CNN. Ця нелінійна ефективність активаційних шарів означає, що відображення вхідного та вихідного даних буде нелінійним; більше того, ці рівні дають CNN можливість вивчати надскладні речі. Функція активації також повинна мати здатність диференціюватись, що є надзвичайно важливою особливістю, оскільки вона дозволяє використовувати зворотне розповсюдження помилок для навчання мережі. Наступні типи функцій активації найчастіше використовуються в CNN та інших глибоких нейронних мережах:

a. Sigmoid: вхід цієї функції активації є дійсними числами, тоді як вихідний сигнал обмежений нулем та одиницею. Крива сигмоїдної функції має S-подібну форму і може бути представлена математично рівнянням (2.1).

$$f(x)_{sigm} = \frac{1}{1+e^{-x}} \quad (2.1)$$

b. Tanh: Це схоже на сигмоїдну функцію, оскільки її вхідні дані є дійсними числами, але вихідний сигнал обмежений між - 1 і 1. Його математичне подання представлено рівнянням (2.2).

$$f(x)_{tahn} = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad (2.2)$$

c. ReLU: Найчастіше вживана функція в контексті CNN. Вона перетворює цілі значення вхідних даних у додатні числа. Менша обчислювальне навантаження - головна перевага ReLU перед іншими. Його математичне подання представлено в рівнянні (2.3).

$$f(x)_{ReLU} = max(0, x) \quad (2.3)$$

Іноді під час використання ReLU може виникнути кілька важливих проблем. Наприклад, розглянемо алгоритм зворотного розповсюдження помилок з більшим градієнтом, що проходить через нього. Передача цього градієнта в функцію ReLU оновить ваги таким чином, що нейрон точно не активується ще раз. Цю проблему називають “Помираючим ReLU”. Для вирішення таких проблем існують деякі альтернативи ReLU.

4. Fully Connected Layer: Зазвичай цей шар розташований у кінці кожної архітектури CNN. Усередині цього шару кожен нейрон з'єднаний з усіма нейронами попереднього шару, так званий підключений повністю. Він використовується як класифікатор CNN. Це слідує основному методу звичайної багатошарової нейромережі персептрона, оскільки це тип ANN із зворотним зв'язком. Вхід рівня FC надходить від останнього об'єднаного або згорткового шару. Цей вхід має форму вектора, який створюється з карт об'єктів після згладжування. Вихід рівня FC представляє кінцевий вихід CNN, як проілюстровано на рис. 2.9.

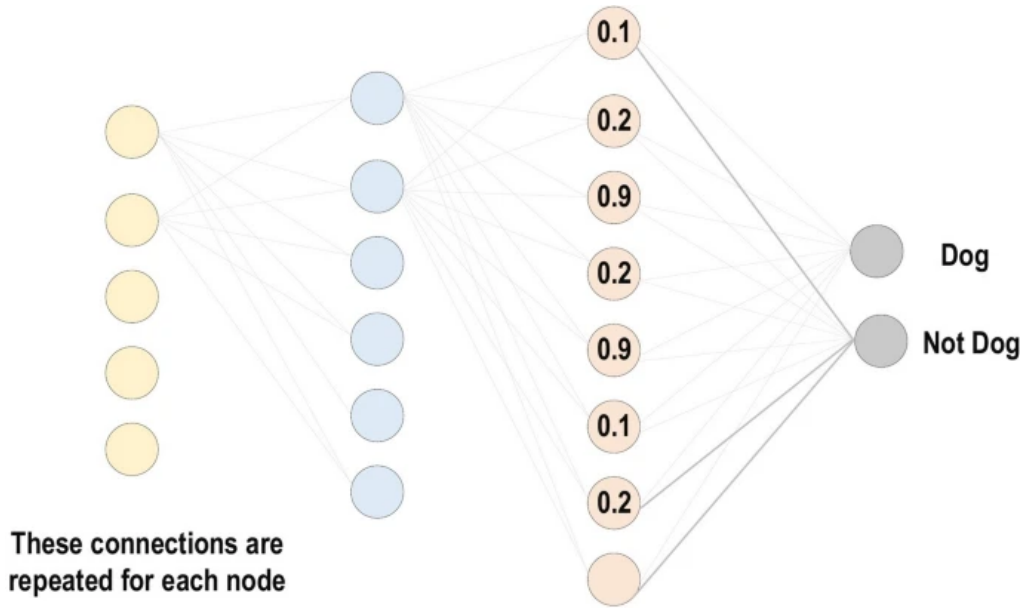


Рисунок 2.9 - Fully Connected Layer

5. Функції втрат: У попередньому розділі були представлені різні типи шарів архітектури CNN, крім того, остаточна класифікація досягається з вихідного рівня, який представляє останній шар архітектури CNN. Деякі функції втрат використовуються у вихідному рівні для обчислення передбачуваної помилки, створеної в навчальних зразках у моделі CNN. Ця помилка виявляє різницю між фактичним випуском та прогнозованим. Далі випуск буде оптимізовано за допомогою навчального процесу CNN. Два параметри використовуються функцією втрат для обчислення помилки. Оцінений вихід CNN (іменований як передбачення) є першим параметром. Фактичний результат (іменований ярликом) є другим параметром.

Для моделей CNN проблема over-fit є центральною проблемою, пов'язаною з отриманням добре вихованого узагальнення. Модель може бути over-fitted в тих випадках, коли модель особливо добре працює на навчальних даних і не дає успіху на тестових даних. Недоступна модель - це навпаки; цей випадок трапляється, коли модель не засвоює достатню кількість з навчальних даних.

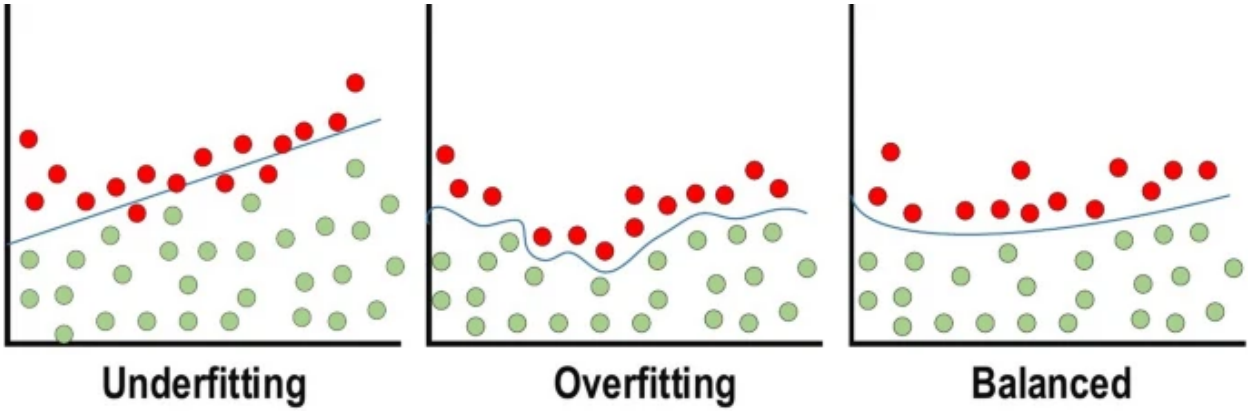


Рисунок 2.10 - Зображення різних станів навчання в моделі

Модель називається "just-fitted", якщо вона добре працює як на даних навчання, так і на тестуванні. Різні інтуїтивні поняття використовуються, щоб допомогти регуляризації уникнути надмірного припасування, серед них:

- Випадання: Це широко застосовувана техніка узагальнення. Протягом кожної навчальної епохи нейрони випадково скидаються. При цьому сила вибору ознак розподіляється порівну по всій групі нейронів, а також змушує модель вивчати різні незалежні особливості. Під час тренувального процесу нейрон, що випав, не буде частиною зворотного або прямого поширення. Навпаки, повномасштабна мережа використовується для прогнозування під час процесу тестування.
- Нормалізація batch: Цей метод забезпечує виконання вихідних активацій [16]. Цей показник відповідає одиничному гауссовому розподілу. Віднімання середнього та ділення на стандартне відхилення нормалізує вихід на кожному шарі. Хоча це можна розглядати як завдання попередньої обробки на кожному рівні мережі, можна також диференціювати та інтегрувати його з іншими мережами. Крім того, він застосовується для зменшення “внутрішнього зсуву коваріації” шарів активації. У кожному шарі зміна розподілу активації визначає

внутрішній зсув коваріації. Цей зсув стає дуже високим завдяки постійному оновленню ваги під час тренувань, яке може статися, якщо зразки даних тренувань збираються з численних різнорідних джерел (наприклад, денні та нічні зображення). Таким чином, модель буде витрачати додатковий час на конвергенцію, а в свою чергу, час, необхідний для навчання, також збільшиться. Для вирішення цієї проблеми в архітектурі CNN застосовується шар, що відображає операцію пакетної нормалізації.

Ми розглянули основні архітектурні компоненти згорткової нейронної мережі CNN. Нам відомо, що функція втрат потрібна для того, щоб нейронна мережа могла розуміти в яку сторону їй потрібно рухатися, щоб покращувати свій результат. Для того, щоб виконувати цей рух і потрібні оптимізатори. У поглибленому навчанні використовуються такі оптимізатори:

а) Градієнтний спуск.

Градієнтний спуск - це ітеративний алгоритм оптимізації першого порядку для знаходження мінімуму функції. Щоб знайти локальний мінімум функції за допомогою градієнтного спуску, робляться кроки, пропорційні негативу градієнта (або приблизного градієнта) функції в поточній точці. Якщо замість цього роблять кроки, пропорційні позитиву градієнта, наближаються до локального максимуму цієї функції, і процедура тоді називається підйомом градієнта.

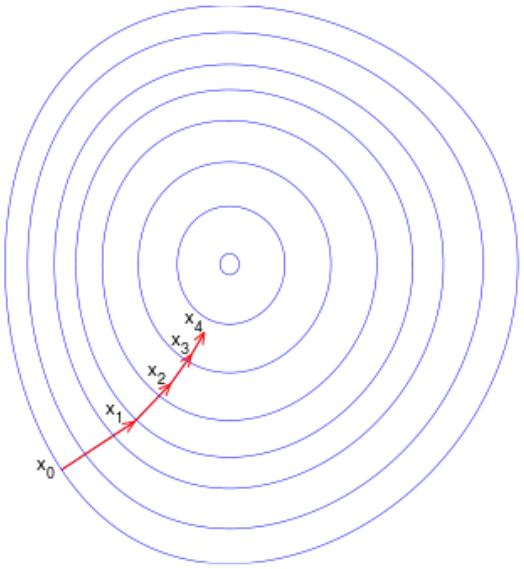


Рисунок 2.11 - Градієнтний спуск. Траєкторія рухається до мінімуму функції втрат

б) Стохастичний градієнтний спуск

Стандартним методом навчання нейронних мереж є метод стохастичного градієнтного спуску (SGD). Проблема градієнтного спуску полягає в тому, що для того, щоб визначити нове наближення вектора ваги, необхідно розрахувати градієнт з кожного елемента вибірки, що може значно уповільнити алгоритм. Ідея прискорення алгоритму стохастичного градієнтного спуску полягає у використанні лише одного елемента або деякої під вибірки для обчислення нового наближення ваг.

Нейронні мережі часто навчаються стохастично; тобто різні фрагменти даних використовуються на різних ітераціях. Це як мінімум з двох причин. По-перше, набори даних, що використовуються для навчання, часто надто великі, щоб повністю їх зберігати в оперативній пам’яті та/або ефективно виконувати обчислення. По-друге, оптимізована функція, як правило, не опукла, тому використання різних частин даних на кожній ітерації може допомогти затримати модель на локальному мінімумі.

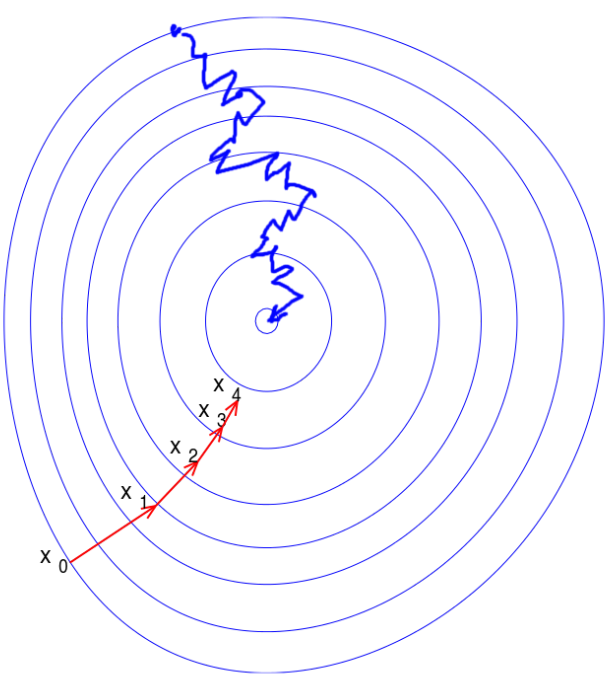


Рисунок 2.12 - Стохастичний градієнтний спуск (синій). Траєкторія не є прямою, але дозволяє навчати нейронні мережі на частині даних.

в) Adam

Адаптивна оцінка моменту (Adam) - це ще одна техніка оптимізації або алгоритм навчання, яка широко використовується [17]. Адам представляє останні тенденції в оптимізації глибокого навчання. Це представлено матрицею Гесса, яка використовує похідну другого порядку. Адам - це стратегія навчання, розроблена спеціально для навчання глибоких нейронних мереж. Більш ефективна пам’ять і менша обчислювальна потужність - дві переваги Адама. Механізм Адама полягає в розрахунку адаптивного LR для кожного параметра в моделі. Він об'єднує плюси як Momentum, так і RMSprop. Він використовує квадратичні градієнти для масштабування швидкості навчання як RMSprop, і це схоже на імпульс за допомогою ковзного середнього градієнта. Рівняння Адама:

$$w_{ij}{}^{t} = w_{ij}{}^{t-1} - \frac{\eta}{\sqrt{\widehat{E[\delta^2]^t} + \epsilon}} * E[\delta^2]^t$$

2.4 Висновки до розділу

В розділі виконано огляд існуючих підходів до побудови моделі для розпізнавання мови, та алгоритми обробки. Було викладено базове поняття про функцію активації, наведено декілька варіантів готових, вибрано найкращу з них, яка найкраще підходить під нашу задачу. Було викладено базові поняття про згортку, її структуру, принцип роботи в обробці зображень, оскільки вона часто використовується в такого роду задачах. Модель CNN, що використовує згорткову мережу для знаходження паттернів подібності має чіткий та теоретично обґрунтований алгоритм, та дозволяє побудувати надійну модель співставлень зображень, а точніше спектрограм. Досвід застосування моделі показує високі результати для задач візуальної відповідності.

РОЗДІЛ 3 ОПИС ПРОГРАМНОГО ПРОДУКТУ

3.1 Обґрунтування вибору платформи та мови програмування

Оскільки робота включає в себе інтегрування сторонніх реалізацій глибинних мереж, для створення продукту розглядалися дві мови програмування: C++ та Python. Розглянемо детальніше кожну з них, та проведемо порівняльний аналіз.

3.1.1 Вибір мови

3.1.1.1 Python

Python став широко використовуваною мовою для машинного навчання з найбільш підтримуваними бібліотеками для тих самих цілей. Легкий для розуміння синтаксис Python, вбудовані функції та широка підтримка пакетів зробили його загальновизнаною мовою програмування, а також найсильнішим гравцем у грі машинного навчання та науки про дані.

Мова підтримує два стилі програмування, а саме функціональний та об’єктно-орієнтовний, також можна комбінувати ці стилі. Варто згадати полегшений синтаксис цієї мови, яка відмовилася від зайвих дужок, замінивши їх символами табуляції. Python є скриптовою мовою, а це означає, що вона не вимагає етапу компіляції, що пришвидшує розробку. Мова є відносно повільною, проте є велика кількість модулів, написаних на швидких мовах програмування, які з легкістю інтегруються в Python. Варто зазначити, що більшість дослідників в галузі deep learning використовують саме Python.

Щоб отримати приблизну оцінку потужності Python, можна просто зрозуміти той факт, що ми можемо отримати доступ до понад 299,638 пакетів через менеджер пакетів PyPI [18]. “Python для всього” був би правильною фразою для його опису, а також має велику підтримку спільноти.

Деякі з пакетів, які підтримуються в Python для машинного навчання: - Tensorflow для глибокого навчання, Numpy для математичних операцій; Pandas для файлових операцій; Pytorch для пакету глибокого навчання; Sklearn для класифікації та алгоритмів регресії; OpenCV і Dlib для комп'ютерного зору; і Matplotlib для візуалізації даних.

З усіма цими перевагами Python також має кілька недоліків щодо відносної повільності роботи, ніж інші мови, такі як C ++, а також намагається підтримати багатопоточність.

3.1.1.2 C++

Що стосується побудови бібліотек, то тут бере участь C ++. Ми часто чули, що до розробки ігор та великих систем найбільше підходили до C ++. Це пов’язано з його функцією переносимості, а також забезпечує базове розуміння побудови логіки.

Деякі з пакетів, які підтримує C++, включають - Microsoft Cognitive Toolkit (CNTK) для глибокого навчання; Tensorflow для глибокого навчання; OpenCV для комп'ютерного зору; MLPack для машинного навчання; DyNet для нейронних мереж; OpenNN для нейронних мереж; Shogun для машинного навчання і FANN для нейронних мереж.

Однак, як і все інше, у C ++ теж є свої недоліки: Він дуже орієнтований на синтаксис, на відміну від Python, який справді зручний для початківців; Немає великої бібліотечної підтримки, як Python; Дуже складний процес інтеграції сторонніх бібліотек у проект; Відсутність стандартизованого пакетного менеджеру.

Враховуючи, що швидкість роботи є рівноцінно важливою з швидкістю розробки, було прийнято рішення вибрати мову програмування Python, з можливістю подальшої інтеграції побудованої моделі у C++.

3.1.2 Вибір платформи та бібліотек.

При роботі були використані різні бібліотеки для докладного аналізу даних, а саме Numpy, Pandas, які надають ефективні інструменти для розробці та підготовці даних. Бібліотеку Scikit-learn використовували власне для аналізу та препроцесінгу даних. Також було описано і використано бібліотеку для вирішення задач та побудови нейронної мережі TensorFlow.

3.1.2.1 TensorFlow

Бібліотека програмного забезпечення для машинного навчання є новим поколінням внутрішньої технології DistBelief, яка була розроблена командою Google Brain для безлічі завдань, включаючи пошук зображень і поліпшення алгоритмів розпізнавання мови.

TensorFlow - це нейронна мережа, яка вчиться вирішувати завдання шляхом позитивного посилення і обробляє дані на різних рівнях (вузлах), що допомагає знаходити вірний результат. Бібліотеки TensorFlow помітно спрощують вбудовування в додатки самонавчального елементів і функцій штучного інтелекту, призначених для розпізнавання мови, організації комп'ютерного зору або обробки природної мови.

Дослідники даних застосовують бібліотеку для самих різних цілей, починаючи від алгоритмів маршрутизації роботів, що переміщаються по складах, і закінчуючи поліпшенням прогнозування попиту і пропозицією покупцям товарів відповідно до їх недавніми уподобанням.

3.2 Опис архітектури та моделі

Основним результатом роботи є програмний продукт, який буде приймати на вхід аудіозапис та видавати результат, а також модель,

побудована на основі фреймворку Tensorflow з пайплайнами препроцессингу та обробки данних.

Програмний продукт складається з двох компонент:

1. GUI-частина
2. Data processing

Було прийнято рішення реалізувати GUI-частину за допомогою фреймворку Qt. Qt - це набір інструментів-віджетів для створення графічних користувальницьких інтерфейсів, а також міжплатформенних додатків, що працюють на різних програмних та апаратних платформах, таких як Linux, Windows, macOS, Android або вбудовані системи, з незначними або відсутніми змінами в базовій кодовій базі.

“Скрита” частина, яка опрацьовує дані та викликає модель та віддає ій дані і повертає оброблений результат реалізована за допомогою бібліотек numpy, pandas, matplotlib, seaborn та tensorflow. Для процесингу даних використовується така послідовність дій:

1. Завантаження аудіозапису у систему.
2. Базова обробка аудіозапису: приведення до єдиного фреймрейту, з’єднання каналів з усередненням результатів.
3. Створення waveform - графік, який відображає силу звуку відносно часу, та яка необхідна для побудови спектрограми.
4. Створення спектрограми.
5. Завантаження спектрограми у модель.
6. Обробка спектрограми моделлю.
7. Вивід результатів моделі у вигляді графіку з відображенням ймовірностей передбачення.

Далі результат роботи моделі відображається у GUI частині програми.

Для імплементації моделі використовується базовий класс `models.Sequential` пакету tensorflow. Цей класс дає можливість

вручну побудувати шари згорткової мережі та налаштувати її саме під наші потреби.

Наша модель складається з 10 шарів, детальний опис яких будет далі (див. Рис 13).

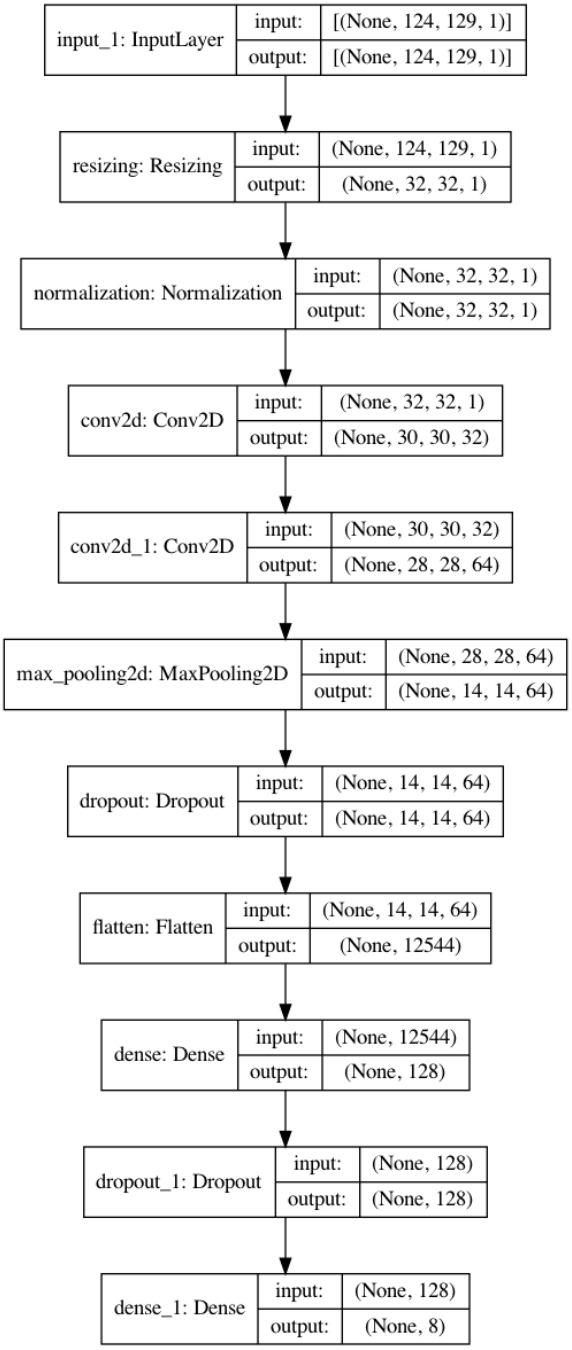


Рисунок 3.1 - Опис нейронної мережі

3.3 Результати роботи

Результатами роботи є готова модель, готова до завантаження у будь-яку систему та готова до використання. Перевіримо роботу моделі при навчанні з базовими параметрами. Результати тренування (кінцеві епохи):

```
Epoch 21/100
100/100 [==============================] - 14s 136ms/step - loss: 0.1907 - accuracy:
0.9334 - val_loss: 0.5015 - val_accuracy: 0.8687
Epoch 22/100
100/100 [==============================] - 12s 117ms/step - loss: 0.1844 - accuracy:
0.9356 - val_loss: 0.4665 - val_accuracy: 0.8662
Epoch 23/100
100/100 [==============================] - 11s 113ms/step - loss: 0.1948 - accuracy:
0.9324 - val_loss: 0.5315 - val_accuracy: 0.8600
Epoch 24/100
100/100 [==============================] - 11s 113ms/step - loss: 0.1883 - accuracy:
0.9333 - val_loss: 0.4823 - val_accuracy: 0.8600
```

Для порівняння основних метрик навчання моделі на тренувальних даних виведемо графіки value accuracy та value loss:

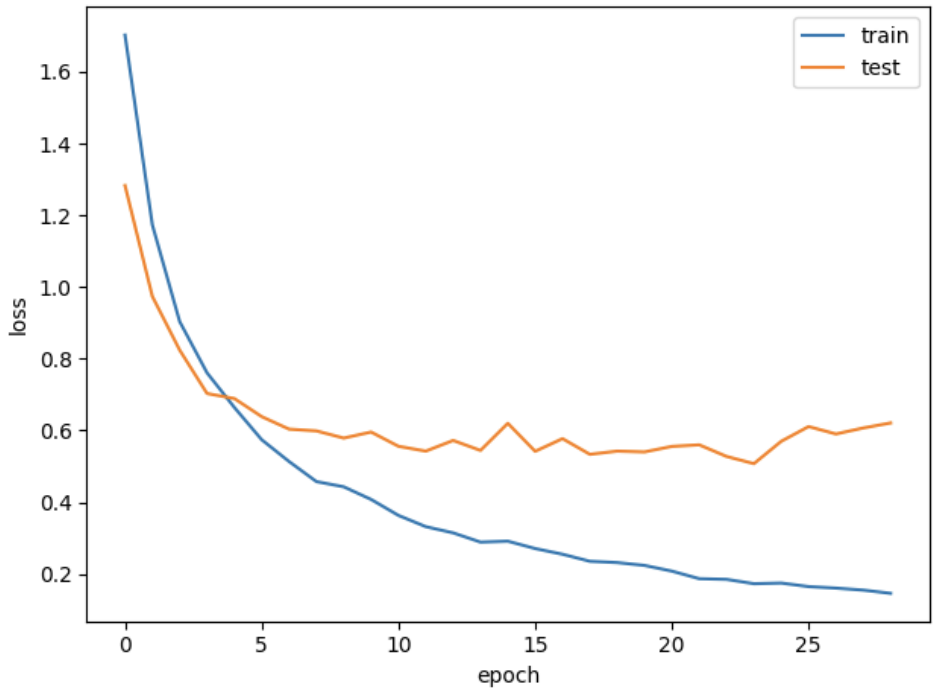


Рисунок 3.2 - Графік втрат валідаційних та тренувальних даних

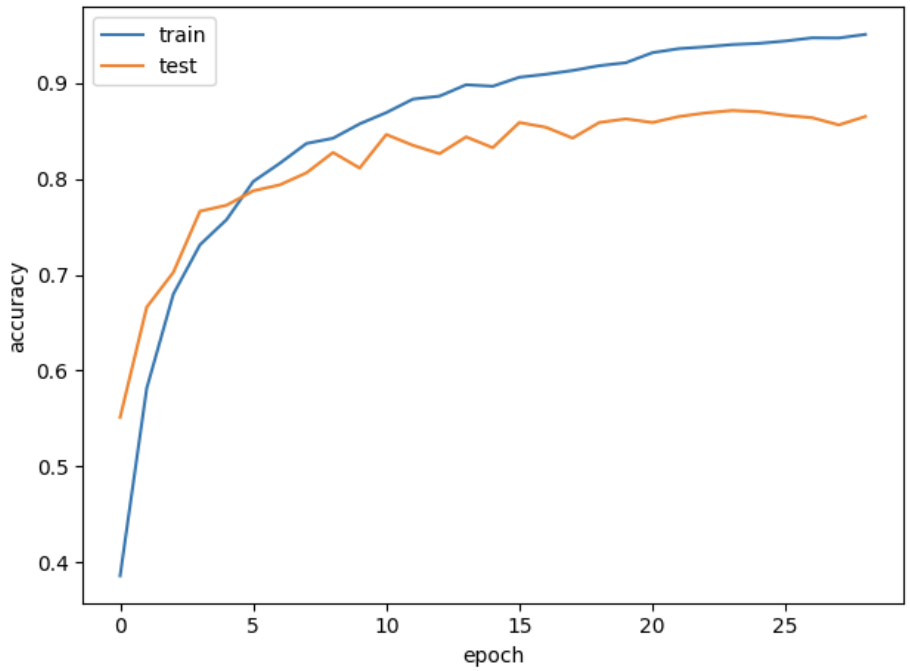


Рисунок 3.3 - Графік точності тренувальних та валідаційних даних

А також матрицю невідповідностей, яка дає можливість унаочнювати продуктивність алгоритму. Кожен з рядків цієї матриці представляє зразки прогнозованого класу, тоді як кожен зі стовпців представляє зразки справжнього класу (або навпаки). Її назва походить від того факту, що вона дає можливість просто бачити, чи допускає система невідповідності між цими двома класами (наприклад, часто помилково маркуючи один як інший):

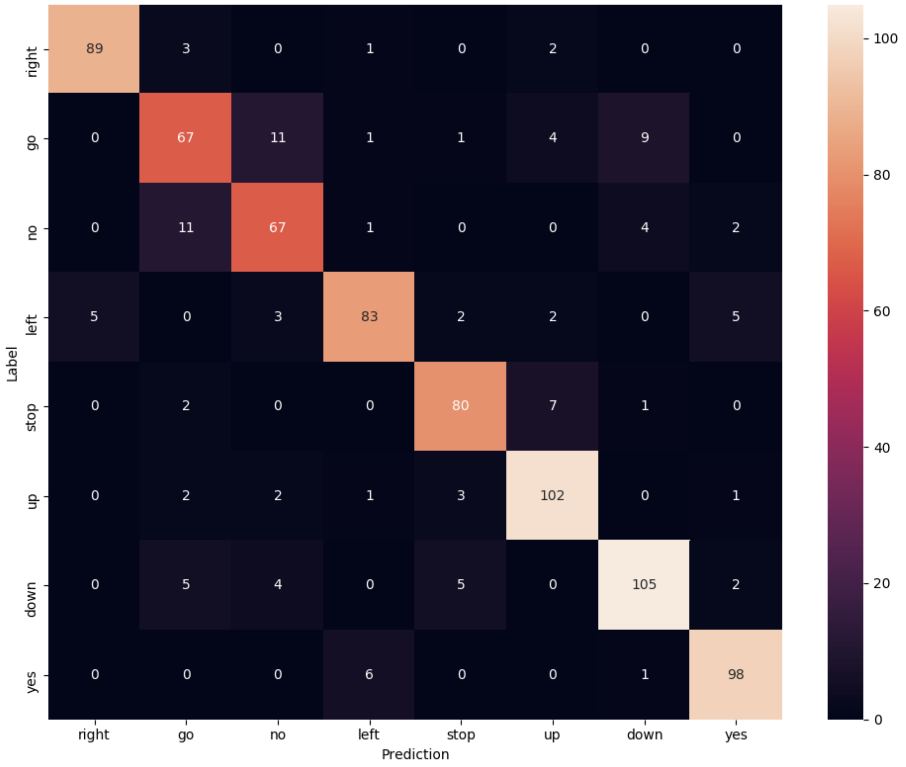


Рисунок 3.4 - Матриця невідповідностей

Як показують результати на тестовому датасеті, модель здатна визначати сказане слово з достатньою точністю. Також, можна зробити висновок що модель продемонструє найкращу поведінку в умовах, коли сказане слово найбільш відповідає стандартному звучаню і спікер немає акценту або інших неточностей вимови.

3.4 Висновки на основі результатів роботи

В даному розділі було проведено аналіз двох мов програмування, Python та C++, наведено недоліки та переваги кожної з них, був проведений порівняльний аналіз та вибрано кращу для побудови необхідної системи, а саме Python. Також було наведено саму архітектуру системи, архітектуру моделі, побудовано детальну схему моделі. В третьому підпункті було проведено аналіз роботи моділ, наведено графіки функції втрат та точності

для тренувальних та валідаційних даних, а також матрицю невідповідностей.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, розробленого для порівняння методів штучного інтелекту для відбору персоналу.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи порівняння методів штучного інтелекту для відбору персоналу. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому

фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних для відбору персоналу.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який вирішує задачу порівняння методів штучного інтелекуту та будує їхні моделі. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування;

F_2 – вибір фреймворку машинного навчання;

F_3 – вибір середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) Python

б) C++

Функція F_2:

а) Tensorflow;

б) PyTorch;

Функція F_3:

а) Jupyter Notebook;

б) Pycharm.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

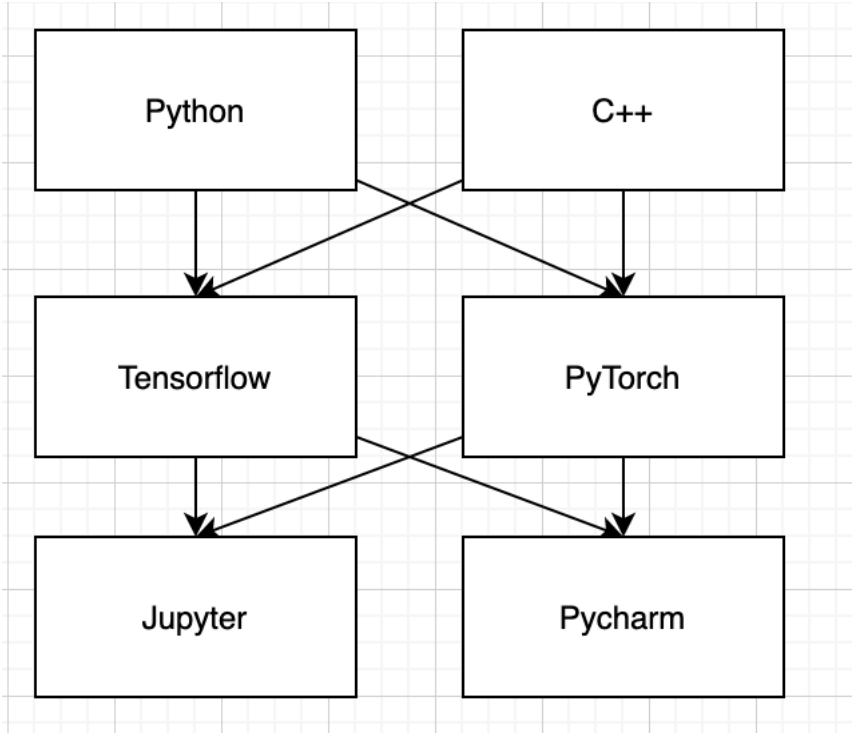


Рисунок 4. 1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіанти основних функцій. На основі цієї карти будуємо позитивно-негативну матрицю варіантів основних функцій (Таб. 4.1). Робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Перевагу віддаємо швидкості вивчення, простоті використання та наявності стандартних бібліотек для обчислення. Для спрощення роботи по написанню коду варіант Б має бути відкинутий.

Функція F_2:

Обидва варіанти можна використовувати в розробці.

Функція F_3:

Віддаємо перевагу варіанту Б в разі вибору мови програмування Python.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

F_1а – F_2а – F_3а

F_1а – F_2б – F_3а

Таблиця 4.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Швидка розробка програми, доступність бібліотек, кросплатформеність	Низька швидкість роботи, особливо, якщо потрібно обробляти велику кількість даних
	Б	Код швидко виконується	Іде багато часу на розробку програми
F_2	*А*	Надійно працює з складними проектами	Не підтримується багатьма мовами
	Б	Надійність та простота у використанні	Не має складних методів
F_3	*А*	Підтримується багатьма мовами програмування, легко запускається на будь-якому сервері	Відсутня можливість роботи без інтернету
	Б	Багато інструментів, безпечна	Підтримує одночасно лише одну мову програмування

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів ПП

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- *X1* – швидкодія мови програмування;
- *X2* – об’єм пам’яті для обчислень та збереження даних;
- *X3* – час навчання даних;
- *X4* – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у таблиці 4.3.

Таблиця 4.2 - Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	10000	14000	19000
Об’єм пам’яті	X2	Мб	420	128	64
Час попередньої обробки даних	X3	мс	4	3	2
Потенційний об’єм програмного коду	X4	кількість рядків коду	4000	2500	1000

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

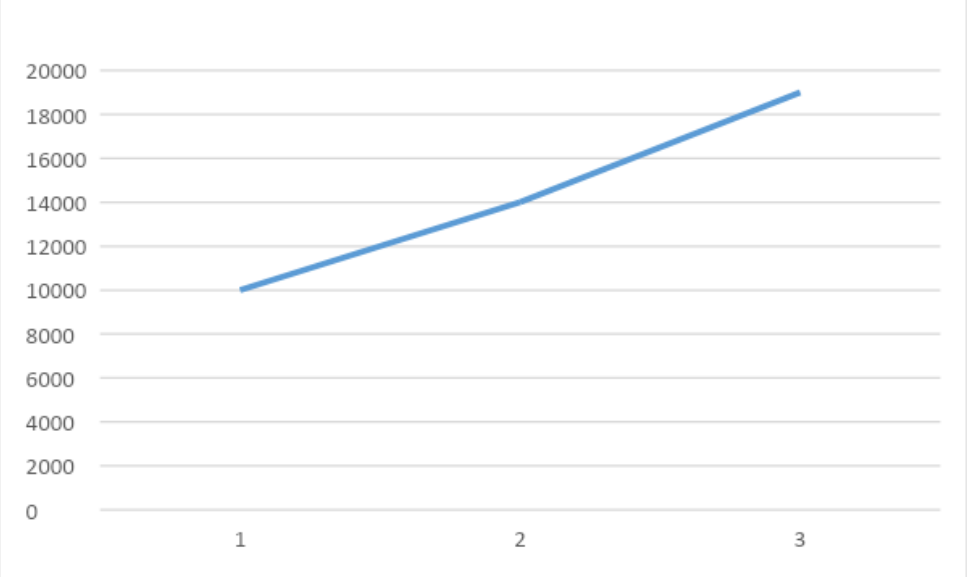


Рисунок 4.2 – X1, швидкодія мови програмування

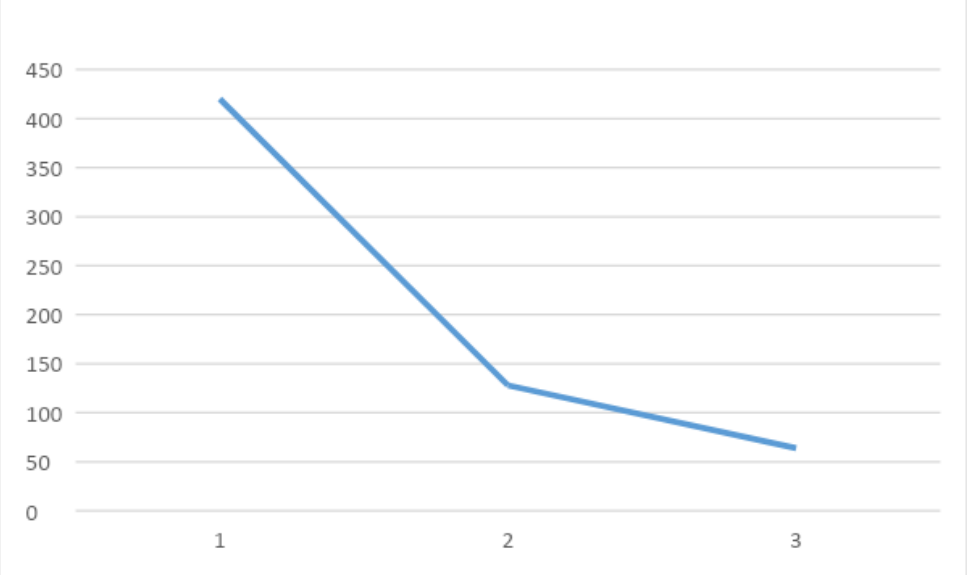


Рисунок 4.3 – X2, об’єм пам’яті

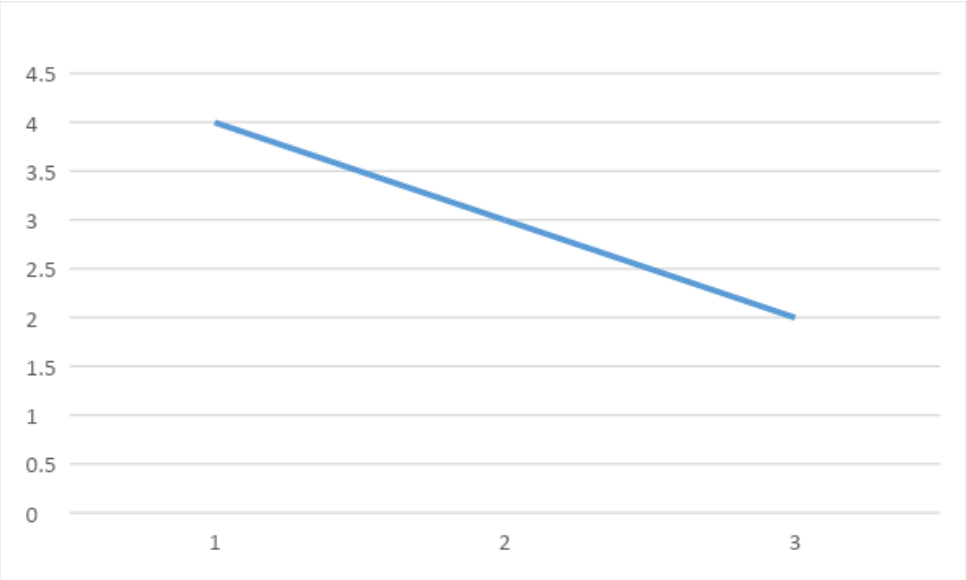


Рисунок 4.4 – X3, час попередньої обробки даних

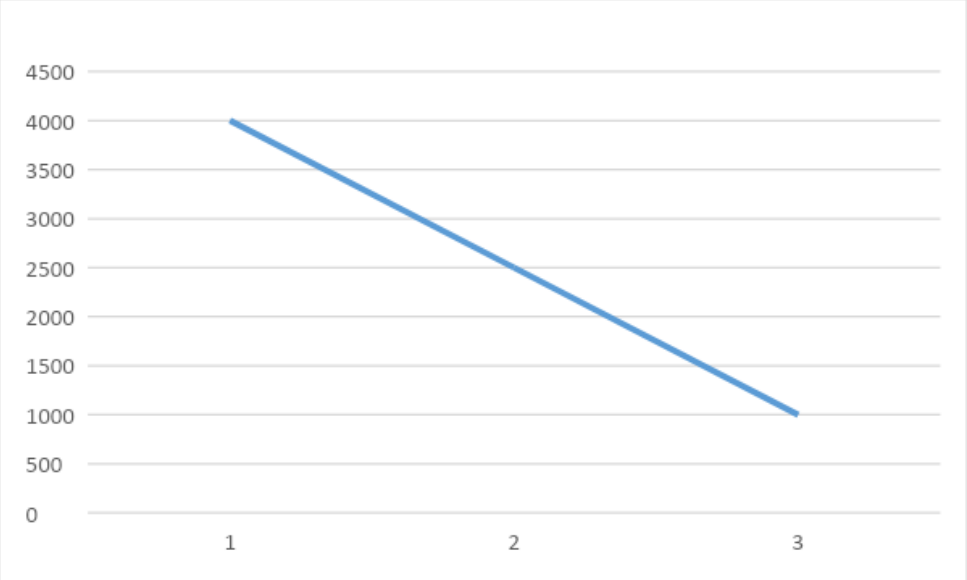


Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі –

розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	4	5	2	5	3	4	5	28	3,5	12,25
X2	Об’єм пам’яті	Mb	2	1	3	1	2	1	2	12	-12,5	156,25
X3	Час попередньої обробки даних	мс	5	3	5	5	4	5	3	30	5,5	30,25

Продовження таблиці 4.3

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X4	Потенційний об'єм програмного коду	Кількість рядків коду	3	5	4	3	5	4	4	28	3,5	12,25
	Разом		14	14	14	14	14	14	14	98	0	211

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 98,$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 24,5$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T.$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 211.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12 \cdot 211}{7^2(4^3-4)} = 0,86 > W_k = 0,67.$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	<	<	>	>	>	>	1,5
X1 і X3	<	>	<	=	<	<	>	<	0,5
X1 і X4	>	>	<	=	<	=	>	>	1,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	<	<	<	<	<	<	<	<	0,5
X3 і X4	>	<	>	>	<	>	<	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \{1.5 \text{ при } X_i > X_j \; 1.0 \text{ при } X_i = X_j \; 0.5 \text{ при } X_i < X_j .$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}$$

$$b_i = \sum_{i=1}^{N} a_{ij}$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'},$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.4 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1,0	1,5	0,5	1,5	4,5	0,36	17,75	0,25	73,38	0,25
X2	0,5	1,0	1,5	0,5	3,5	0,28	15,75	0,22	65,63	0,23
X3	1,5	1,5	1,0	1,5	5,5	0,44	22,75	0,33	93,6	0,32

X4	0,5	1,5	0,5	1,0	3,5	0,28	13,75	0,2	57,63	0,2
Всього:					12,5	1	70	1	290,25	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об'єм пам'яті), *X3* (час попередньої обробки даних) та *X4* (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 5.7):

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j},$$

де n – кількість параметрів;

$K_{\text{в}i}$ – коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	A	X1	10000	8	0.25	2
F2	A	X2	64	3	0,3	0,9

F3	Б	X2	128	4	0,25	1
	А	X3	1000	6	0,2	1,2

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}\left[F_{1k}\right] + K_{\text{ТУ}}\left[F_{2k}\right] + \ldots + K_{\text{ТУ}}\left[F_{zk}\right],$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 2+0,9+1,2 = 4.1,$$

$$K_{K2} = 1,05+1,6+0,69 = 4.2$$

Як видно з розрахунків, кращим є другий варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як

$$T_О = T_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М},$$

де $T_Р$ – трудомісткість розробки ПП;

$К_П$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність вхідної інформації;

$К_М$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_Р$ =90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_Р$ =27 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ =0.8:

$$T_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_1 = (122.4 + 19.44 + 4.8 + 19.44) \cdot 8 = 1328{,}64 \text{ людино-годин.}$$

$$T_2 = (122.4 + 19.44 + 6{,}91 + 19.44) \cdot 8 = 1345{,}52 \text{ людино-годин.}$$

В розробці беруть участь два програмісти з окладом 15000 грн., один аналітик в області даних з окладом 18000. Визначимо середню зарплату за годину за формулою:

$$\text{СЧ} = \frac{\text{М}}{T_m \cdot t} \text{грн.},$$

де М – місячний оклад працівників;

T_m – кількість робочих днів у місяць;

t – кількість робочих годин в день.

$$\text{СЧ} = \frac{15000+15000+18000}{3 \cdot 21 \cdot 8} = 95,24 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою:

$$\text{СЗП} = \text{С}_\text{ч} \cdot \text{Т}_i \cdot \text{КД},$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 95.24 \cdot 1328.64 \cdot 1.2 = 151847.61$ грн.

II. $С_{ЗП} = 95.24 \cdot 1345.52 \cdot 1.2 = 153776.79$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 151847.61 \cdot 0.22 = 33406.47$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 153776.79 \cdot 0.22 = 33830.89$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 15000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 15000 \cdot 0,2 = 36000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 36000 \cdot (1 + 0.2) = 43200 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 43200 \cdot 0,22 = 9504 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 30% та вартості ЕОМ – 30000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.3 \cdot 30000 = 10350 \text{ грн.},$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 30000 \cdot 0.05 = 1725 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.9 =$$

$$= 1677.6 \text{ годин},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1677.6 \cdot 0,3 \cdot 0.8 \cdot 3,1555 = 1270.48 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 30000 \cdot 0,67 = 20100 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{ЕКС} = C_{ЗП} + C_{ВІД} + C_{А} + C_{Р} + C_{ЕЛ} + C_{Н},$$

$C_{ЕКС}$ = 43200 + 9504 + 10350 + 1725 + 1150 + 1270.48 + 20100 = 87299.48 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{М\text{-}Г} = C_{ЕКС} / T_{ЕФ} = 87299.48 / 1677.6 = 52.04 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_{М} = C_{М-Г} \cdot T,$$

I. $C_{М} = 52.04 \cdot 1328.64 = 69142.42$ грн.

II. $C_{М} = 52.04 \cdot 1345{,}52 = 70020.86$ грн.

Накладні витрати складають 67% від заробітної плати:

$$C_{Н} = C_{ЗП} \cdot 0,67,$$

I. $C_{Н} = 151847.61 \cdot 0{,}67 = 101737.89$ грн.

II. $C_{Н} = 153776.79 \cdot 0{,}67 = 103030.44$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_{М} + С_{Н},$$

I. $С_{ПП} = 151847.61 + 33406.47 + 69142.42 + 101737.89 = 356134.39$ грн.

II. $С_{ПП} = 153776.79 + 33830.89 + 70020.86 + 103030.44 = 360658.98$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_j} = К_{К_j} / С_{Ф_j},$$

$$К_{ТЕР1} = 4.1 / 356134.39 = 11.51 \cdot 10^{-6},$$

$$К_{ТЕР2} = 4.2 / 360658.98 = 11.65 \cdot 10^{-6}$$

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР2} = 11.65 \cdot 10^{-6}$

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості
$К_{ТЕР} = 11.65 \cdot 10^{-6}$.

Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – Python;
- Використання моделей з великою ємністю
- Використання стандартного інтерфейсу візуалізації, швидкість

розробки

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, непоганий функціонал і швидкодію.

4.8 Висновки до розділу

Проведено повний функціонально-вартісний аналіз програмного продукту. Визначено та проведено оцінку основних функцій програмного продукту. Визначено параметри, які характеризують програмний продукт. Проведено експертне оцінювання параметрів та аналіз якості реалізації функцій.

Проведено економічний аналіз варіантів розробки – трудомісткість, витрати на заробітну плату та інші витрати.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

У даній дипломній роботі було розглянута задача розпізнавання мови, яка складається з побудови спектрограми, використовуючи швидке перетворення Фур'є, і класифікації отриманої спектрограми використовуючи штучну згорткову нейронну мережу CNN. В рамках даної роботи було створено програмний продукт для опрацювання аудіо сигналу, тренування моделі та побудови передбачення опрацьованого сигналу.

В останньому розділі було розглянуто економічні показники для двох варіантів реалізації та обрано найоптимальніший із них.

Результати роботи можуть широко застосовуватися у персональних віртуальних помічниках, журналістиці, заповненні документів та багато інших прикладів.

В подальшому можна покращити розроблену систему покращивши опрацювання даних, наприклад додавши детектор тиші (VAD) або додавши додатковий функціонал, наприклад можливість розпізнавання у реальному часі.

ПЕРЕЛІК ПОСИЛАНЬ

1. D. Geer. 5 impacts of speech recognition system in various fields. URL: https://thenextweb.com/news/5-impacts-speech-recognition-system-various-fields.
2. Fortune Business Insights. Speech and Voice Recognition Market to be Worth USD 28.3 Billion by 2026. URL: https://www.globenewswire.com/en/news-release/2021/04/22/2214930/0/en/Speech-and-Voice-Recognition-Market-to-be-Worth-USD-28-3-Billion-by-2026-Rising-at-a-CAGR-of-19-8.html.
3. Meticulous Market Research Pvt. Ltd. Speech and Voice Recognition Market. URL: https://www.globenewswire.com/news-release/2021/03/26/2199918/0/en/Speech-and-Voice-Recognition-Market-to-be-Worth-26-79-billion-by-2025-Market-Size-Share-Forecasts-Trends-Analysis-Report-by-Meticulous-Research.html
4. The Current Challenges Of Speech Recognition. URL: https://onlim.com/en/the-current-challenges-of-speech-recognition/
5. Pahini A. Trivedi V.V.P. Engineering College Rajkot. Introduction to Various Algorithms of Speech Recognition. URL: https://www.ijedr.org/papers/IJEDR1404035.pdf
6. Joe Tebelskis, "Speech Recognition using Neural Networks", May 1995, CMU-CS-95-142, School of Computer Science, Carnegie Mellon University Pittsburgh, Pennsylvania 15213-3890. URL: https://isl.anthropomatik.kit.edu/pdf/Tebelskis1995.pdf.

7. Satyajit D. Sarker, Lutfun Nahar. Computational Phytochemistry. URL: https://www.sciencedirect.com/book/9780128123645/computational-phytochemistry
8. Back Propagation Neural Network: Explained With Simple Example. URL: https://www.guru99.com/backpropogation-neural-network.html.
9. J. Mahanta. Introduction to Neural Networks, Advantages and Applications.
   URL:
   https://towardsdatascience.com/introduction-to-neural-networks-advantages-and-applications-96851bd1a207.
10. Deng Y. , Li X. , Kwan C.,Raj B.,Stern R., "Continuous Feature Adaptation for Non-Native Speech Recognition", International Journal of Computer, Information Science and Engineering ,Vol:1 No:6, 2007. URL: https://www.researchgate.net/publication/282157449_Continuous_Feature_Adaptation_For_Non-Native_Speech_Recognition.
11. National Institute of Standards and Technology, "The History of Automatic Speech Recognition Evaluation", at NIST.
12. Definition of Spectrogram. URL: https://ccrma.stanford.edu/~jos/mdft/Spectrograms.html.
13. STFT Spectrograms Details. URL: https://zone.ni.com/reference/en-XX/help/371361E-01/lvanls/stft_spectrogram_core/#details.
14. Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. URL: https://dl.acm.org/doi/10.1145/3065386.
15. Wang T, Lu C, Yang M, Hong F, Liu C. A hybrid method for heartbeat classification via convolutional neural networks, multilayer perceptrons and focal loss. URL: https://peerj.com/articles/cs-324/.

16. Ioffe S, Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift. URL: https://arxiv.org/abs/1502.03167.
17. Zhang Z. Improved Adam optimizer for deep neural networks. URL: https://www.researchgate.net/publication/331422684_Improved_Adam_Optimizer_for_Deep_Neural_Networks.
18. Analytics for PyPI packages. URL: https://pypistats.org/

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

gui_app.py

```
import pathlib
import time
import sys
from PyQt5 import QtWidgets
import matplotlib
from PyQt5.QtWidgets import QPushButton, QHBoxLayout, QVBoxLayout, QFileDialog, QLabel

matplotlib.use('Qt5Agg')
matplotlib.rcParams['toolbar'] = 'None'

import numpy as np
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.figure import Figure
import tensorflow as tf
from tensorflow.keras.models import load_model


from utils import get_spectrogram, get_waveform, preprocess_dataset

commands = ['right', 'go', 'no', 'left', 'stop', 'up', 'down', 'yes']

class MplCanvas(FigureCanvas):

    def __init__(self, parent=None, width=5, height=4, dpi=100):
        fig = Figure(figsize=(width, height), dpi=dpi)
        self.axes = fig.add_subplot(111)
        super(MplCanvas, self).__init__(fig)

class CentralWidget(QtWidgets.QWidget):

    def __init__(self, parent):
        super(CentralWidget, self).__init__(parent)
        self.model = load_model("speech-model")

        self.main_layout = QVBoxLayout(self)
```

```
        self.btn_layout = QHBoxLayout(self)
        self.filename_btn = QPushButton(self)
        self.filename_btn.setText("Select Audio")
        self.btn_layout.addWidget(self.filename_btn)

        self.predict_btn = QPushButton(self)
        self.predict_btn.setText("Predict")
        self.btn_layout.addWidget(self.predict_btn)

        self.main_layout.addLayout(self.btn_layout)

        self.filename_label = QLabel(self)
        self.main_layout.addWidget(self.filename_label)

        self.canvas_layout = QHBoxLayout(self)
        self.spec_canvas = MplCanvas(self, width=5, height=4, dpi=100)
        self.canvas_layout.addWidget(self.spec_canvas)

        self.predict_canvas = MplCanvas(self, width=5, height=4, dpi=100)
        self.canvas_layout.addWidget(self.predict_canvas)

        self.main_layout.addLayout(self.canvas_layout)

        self.show()

        self.filename_btn.clicked.connect(self.select_audio)
        self.predict_btn.clicked.connect(self.predict)

    def select_audio(self):
            self.filename, _ = QFileDialog.getOpenFileName(self, "Open Audio",
"~", "Audio Files (*.wav)")
        print(self.filename)
        if not self.filename:
            return
        self.filename_label.setText(self.filename)
        self.update_plot()

    def update_plot(self):
        waveform_ = get_waveform(self.filename)
        spectrogram_ = get_spectrogram(waveform_)
        self.spec_data = spectrogram_.numpy()

        log_spec = np.log(self.spec_data.T)
```

```
        height = log_spec.shape[0]
        width = log_spec.shape[1]
        X = np.linspace(0, np.size(self.spec_data), num=width, dtype=int)
        Y = range(height)

        self.spec_canvas.axes.cla()
        self.spec_canvas.axes.pcolormesh(X, Y, log_spec)
        self.spec_canvas.draw()

    def predict(self):
        if not self.filename:
            return
        sample_ds = preprocess_dataset([str(self.filename)])
        start_time = time.time()
        for spectrogram in sample_ds.batch(1):
            prediction = self.model(spectrogram)
            softmax = tf.nn.softmax(prediction[0])
            # # print(softmax.numpy())
            self.predict_canvas.axes.bar(commands, softmax)
            self.predict_canvas.axes.set_ylim(bottom=0, top=1)
            self.predict_canvas.axes.set_ylabel('Probability')
            self.predict_canvas.draw()
              print("---Prediction took: %s seconds ---" % (time.time() -
start_time))


class MainWindow(QtWidgets.QMainWindow):

    def __init__(self, *args, **kwargs):
        super(MainWindow, self).__init__(*args, **kwargs)
        self.central_widget = CentralWidget(self)
        self.resize(800, 400)
        self.setCentralWidget(self.central_widget)


app = QtWidgets.QApplication(sys.argv)
w = MainWindow()
w.show()
app.exec_()
```

utils.py

```
import numpy as np
import tensorflow as tf

# Set seed for experiment reproducibility
seed = 42
tf.random.set_seed(seed)
np.random.seed(seed)


def get_waveform(file_path):
    audio_binary = tf.io.read_file(file_path)

    audio, _ = tf.audio.decode_wav(audio_binary)
    audio = tf.squeeze(audio, axis=-1)
    return audio


def get_spectrogram(waveform):
    waveform = tf.cast(waveform, tf.float32)
    spectrogram = tf.signal.stft(
        waveform, frame_length=255, frame_step=128)

    spectrogram = tf.abs(spectrogram)

    return spectrogram

def preprocess_dataset(files):
    files_ds = tf.data.Dataset.from_tensor_slices(files)
                            output_ds       =       files_ds.map(get_waveform,
num_parallel_calls=tf.data.AUTOTUNE)
    output_ds = output_ds.map(
        get_spectrogram, num_parallel_calls=tf.data.AUTOTUNE)
    return output_ds
```

model.py

```
import os
import pathlib

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

```
import tensorflow as tf

from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras import layers
from tensorflow.keras import models
from IPython import display

# Set seed for experiment reproducibility
seed = 42
tf.random.set_seed(seed)
np.random.seed(seed)

data_dir = pathlib.Path('data/train')
# if not data_dir.exists():
#   tf.keras.utils.get_file(
#       'mini_speech_commands.zip',
#
origin="http://storage.googleapis.com/download.tensorflow.org/data/mini_speech
_commands.zip",
#       extract=True,
#       cache_dir='.', cache_subdir='data')

commands = np.array(tf.io.gfile.listdir(str(data_dir)))
commands = commands[commands != '.DS_Store']
print('Commands:', commands)

filenames = tf.io.gfile.glob(str(data_dir) + '/*/*')
filenames = tf.random.shuffle(filenames)
num_samples = len(filenames)
print('Number of total examples:', num_samples)
print('Number of examples per label:',
      len(tf.io.gfile.listdir(str(data_dir / commands[0]))))
print('Example file tensor:', filenames[0])

train_files = filenames[:6400]
val_files = filenames[6400: 6400 + 800]
test_files = filenames[-800:]

print('Training set size', len(train_files))
print('Validation set size', len(val_files))
print('Test set size', len(test_files))
```

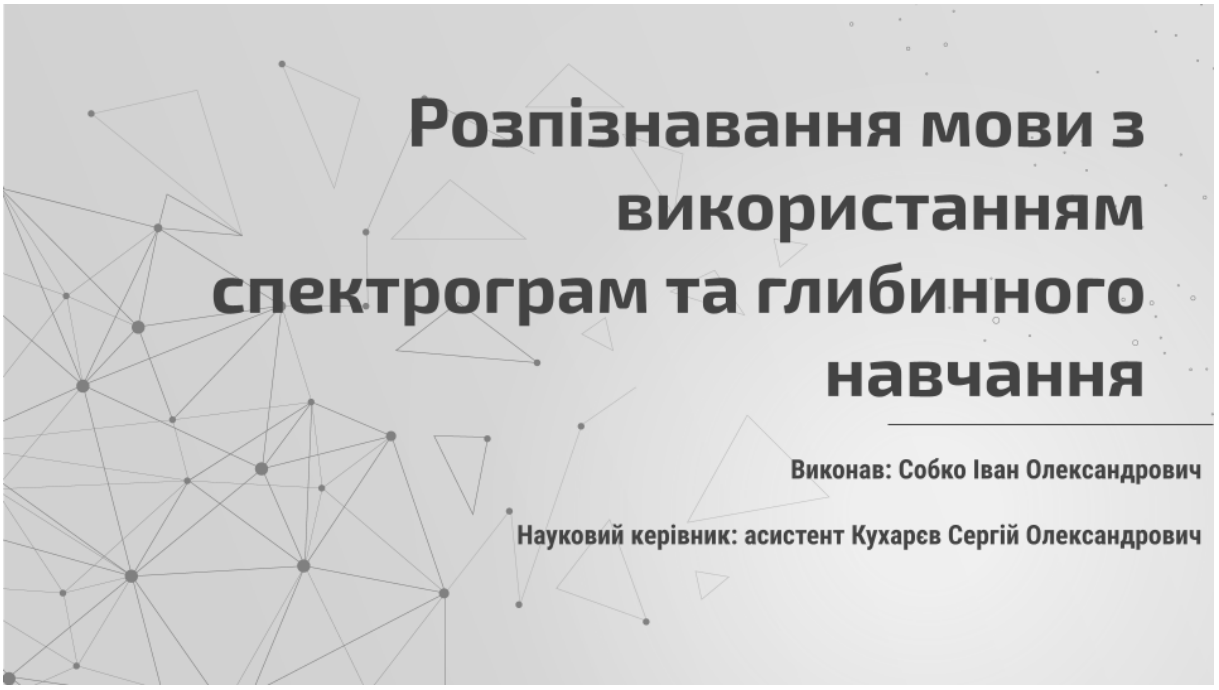
```
def decode_audio(audio_binary):
    audio, _ = tf.audio.decode_wav(audio_binary)
    return tf.squeeze(audio, axis=-1)


def get_label(file_path):
    parts = tf.strings.split(file_path, os.path.sep)

    # Note: You'll use indexing here instead of tuple unpacking to enable this
    # to work in a TensorFlow graph.
    return parts[-2]


def get_waveform_and_label(file_path):
    label = get_label(file_path)
    audio_binary = tf.io.read_file(file_path)
    waveform = decode_audio(audio_binary)
    return waveform, label


AUTOTUNE = tf.data.AUTOTUNE
files_ds = tf.data.Dataset.from_tensor_slices(train_files)
waveform_ds                =                files_ds.map(get_waveform_and_label,
num_parallel_calls=AUTOTUNE)

def get_spectrogram(waveform):
    # Padding for files with less than 16000 samples
    zero_padding = tf.zeros([16000] - tf.shape(waveform), dtype=tf.float32)

    # Concatenate audio with padding so that all audio clips will be of the
    # same length
    waveform = tf.cast(waveform, tf.float32)
    equal_length = tf.concat([waveform, zero_padding], 0)
            spectrogram   =   tf.signal.stft(equal_length,   frame_length=255,
frame_step=128)

    spectrogram = tf.abs(spectrogram)

    return spectrogram


def plot_spectrogram(spectrogram, ax):
    # Convert to frequencies to log scale and transpose so that the time is
```

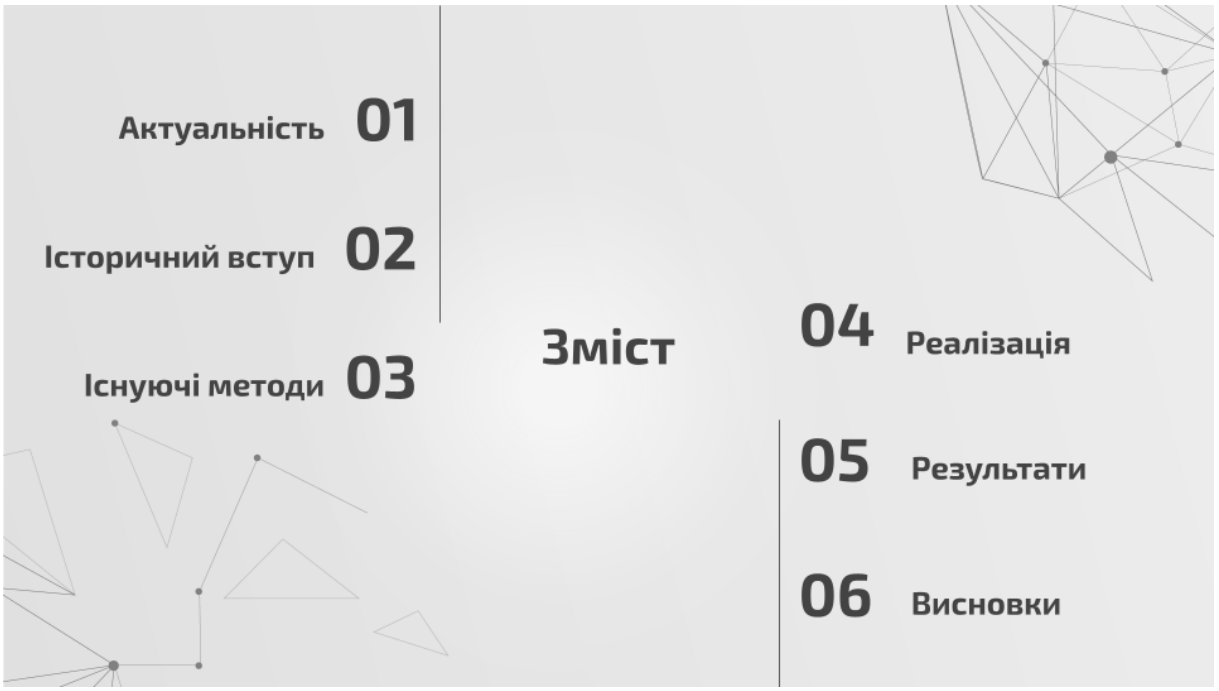
```
    # represented in the x-axis (columns).
    log_spec = np.log(spectrogram.T)
    height = log_spec.shape[0]
    width = log_spec.shape[1]
    X = np.linspace(0, np.size(spectrogram), num=width, dtype=int)
    Y = range(height)
    ax.pcolormesh(X, Y, log_spec)


def get_spectrogram_and_label_id(audio, label):
    spectrogram = get_spectrogram(audio)
    spectrogram = tf.expand_dims(spectrogram, -1)
    label_id = tf.argmax(label == commands)
    return spectrogram, label_id


spectrogram_ds = waveform_ds.map(get_spectrogram_and_label_id,
num_parallel_calls=AUTOTUNE)

def preprocess_dataset(files):
  files_ds = tf.data.Dataset.from_tensor_slices(files)
                output_ds = files_ds.map(get_waveform_and_label,
num_parallel_calls=AUTOTUNE)
  output_ds = output_ds.map(
      get_spectrogram_and_label_id,  num_parallel_calls=AUTOTUNE)
  return output_ds

train_ds = spectrogram_ds
val_ds = preprocess_dataset(val_files)
test_ds = preprocess_dataset(test_files)

batch_size = 64
train_ds = train_ds.batch(batch_size)
val_ds = val_ds.batch(batch_size)

train_ds = train_ds.cache().prefetch(AUTOTUNE)
val_ds = val_ds.cache().prefetch(AUTOTUNE)


num_labels = len(commands)

norm_layer = preprocessing.Normalization()
norm_layer.adapt(spectrogram_ds.map(lambda x, _: x))
```

```
for spectrogram, _ in spectrogram_ds.take(1):
  input_shape = spectrogram.shape
print('Input shape:', input_shape)
model = models.Sequential([
    layers.Input(shape=input_shape),
    preprocessing.Resizing(32, 32),
    norm_layer,
    layers.Conv2D(32, 3, activation='relu'),
    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_labels),
])

print(model.summary())

model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'],)

EPOCHS = 100
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=EPOCHS,
    callbacks=tf.keras.callbacks.EarlyStopping(verbose=1, patience=5),
)

metrics = history.history
print(metrics.keys())
plt.plot(history.epoch, metrics['loss'], metrics['val_loss'])
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'test'])
plt.show()

plt.plot(metrics['accuracy'])
```

```
plt.plot(metrics['val_accuracy'])
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

test_audio = []
test_labels = []

for audio, label in test_ds:
  test_audio.append(audio.numpy())
  test_labels.append(label.numpy())

test_audio = np.array(test_audio)
test_labels = np.array(test_labels)

y_pred = np.argmax(model.predict(test_audio), axis=1)
y_true = test_labels

test_acc = sum(y_pred == y_true) / len(y_true)
print(f'Test set accuracy: {test_acc:.0%}')

confusion_mtx = tf.math.confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx, xticklabels=commands, yticklabels=commands,
            annot=True, fmt='g')
plt.xlabel('Prediction')
plt.ylabel('Label')
plt.show()


model.save("speech-model")
```

ДОДАТОК Б ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ





01
АКТУАЛЬНІСТЬ

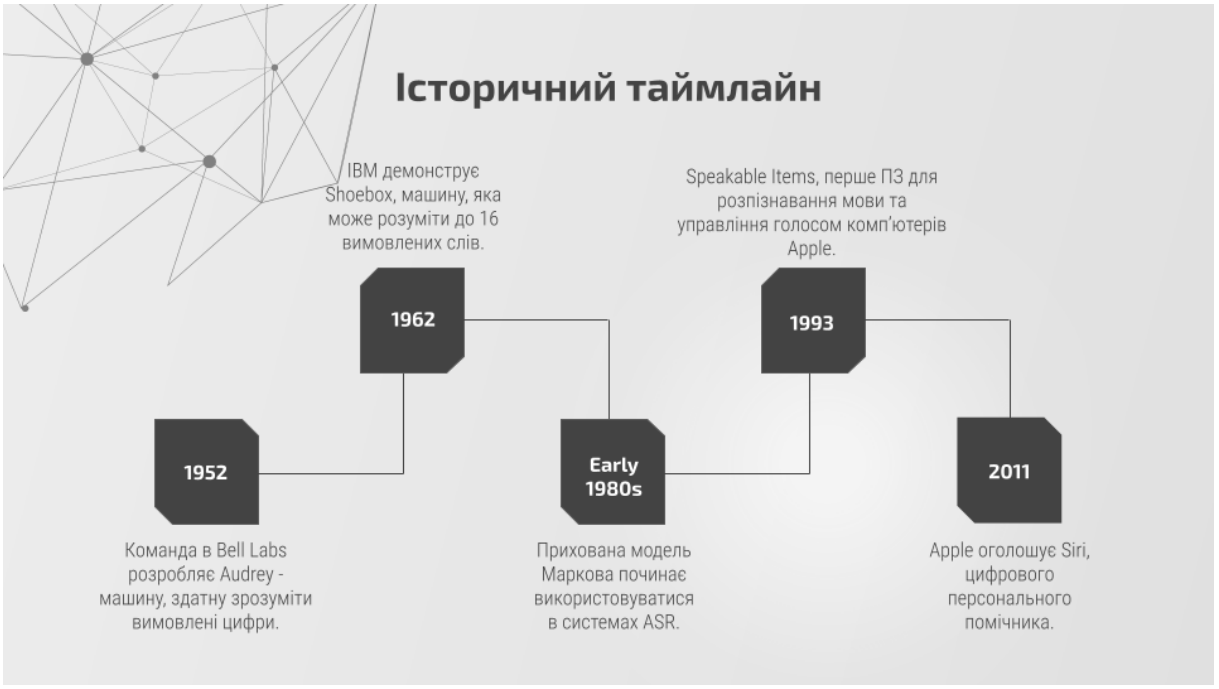
Чи це справді потрібно?

Де використовуються системи speech-to-text?

1. Введення адреси голосом в навігатор Google Maps
2. голосовий пошук Google Now.
3. Голосові асистенти Alexa, Cortana, Siri, та інші
4. Системи голосового пошуку

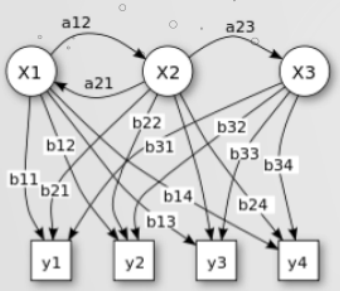

~~Чи набирає популярність~~ Як швидко набирають популярність системи ASR?
У 2018 році ринок оцінювався у 6,9 млрд. $. До кінця 2026 року має досягти 28,3 млрд. $.
Використання голосових асистентів зросло на 103% з 21,5 млн у 2018 році до 43,7 млн у 2020 році.
27% глобальної онлайн популяції використовує voice search на телефонах.


02
ІСТОРІЯ
Хто, коли та як?


Історичний таймлайн
1952
Команда в Bell Labs розробляє Audrey - машину, здатну зрозуміти вимовлені цифри.
1962
IBM демонструє Shoebox, машину, яка може розуміти до 16 вимовлених слів.
Early 1980s
Прихована модель Маркова починає використовуватися в системах ASR.
1993
Speakable Items, перше ПЗ для розпізнавання мови та управління голосом комп'ютерів Apple.
2011
Apple оголошує Siri, цифрового персонального помічника.


03
ІСНУЮЧІ МЕТОДИ
Що вже зробили до нас?

Hidden Markov Model

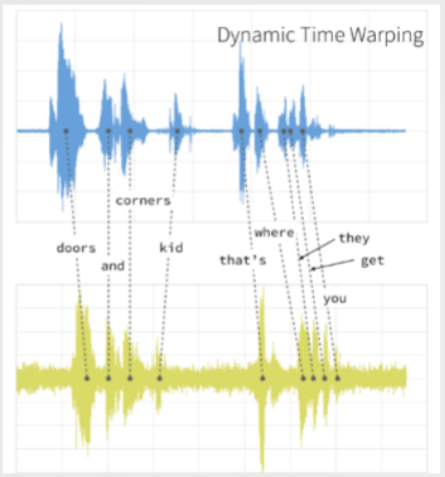


Імовірнісні параметри прихованої моделі Маркова:
X - стани
y - можливі спостереження
a - ймовірності переходу стану
б - вихідні ймовірності

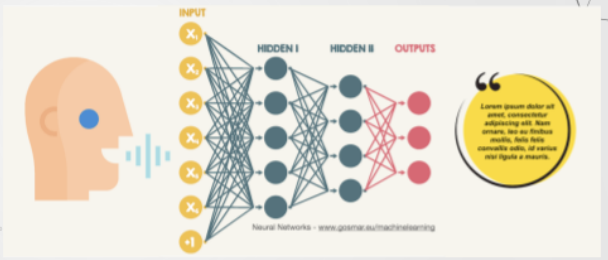
HMM застосовується до розпізнавання мови: стани інтерпретуються як акустичні моделі, вказуючи, які звуки, ймовірно, будуть чути під час відповідних їм сегментів мови; тоді як переходи забезпечують часові обмеження, вказуючи, як стани можуть послідовно слідувати один за одним.

Dynamic Time Warping

DTW створює зсув у часі і відображає кожен елемент серії до найближчого елемента іншої серії. Іншими словами, DTW знаходить оптимальну відстань (тут оптимальна відстань - це найкоротша відстань) між елементами, виконуючи це відображення.



Artificial Neural Networks

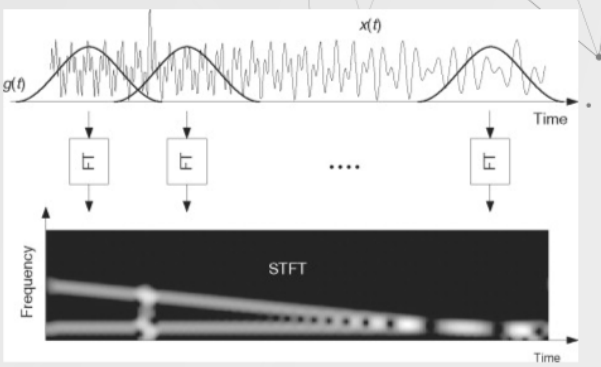


ANN витягує так звані характеристики спектрограми амплітудної модуляції (AM), які є, по суті, специфічними для звуку характеристиками.

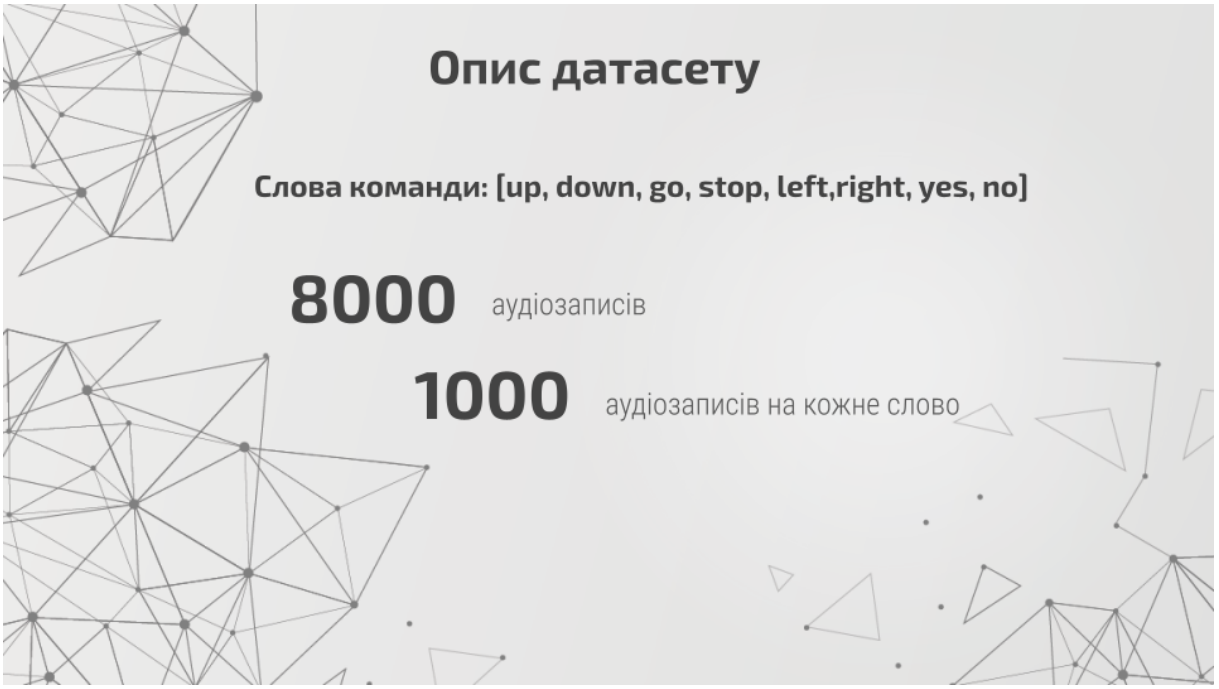
Функції, витягнуті за допомогою моделі потім використовуються для навчання ANN розпізнавати людську мову.

Fourier Transform

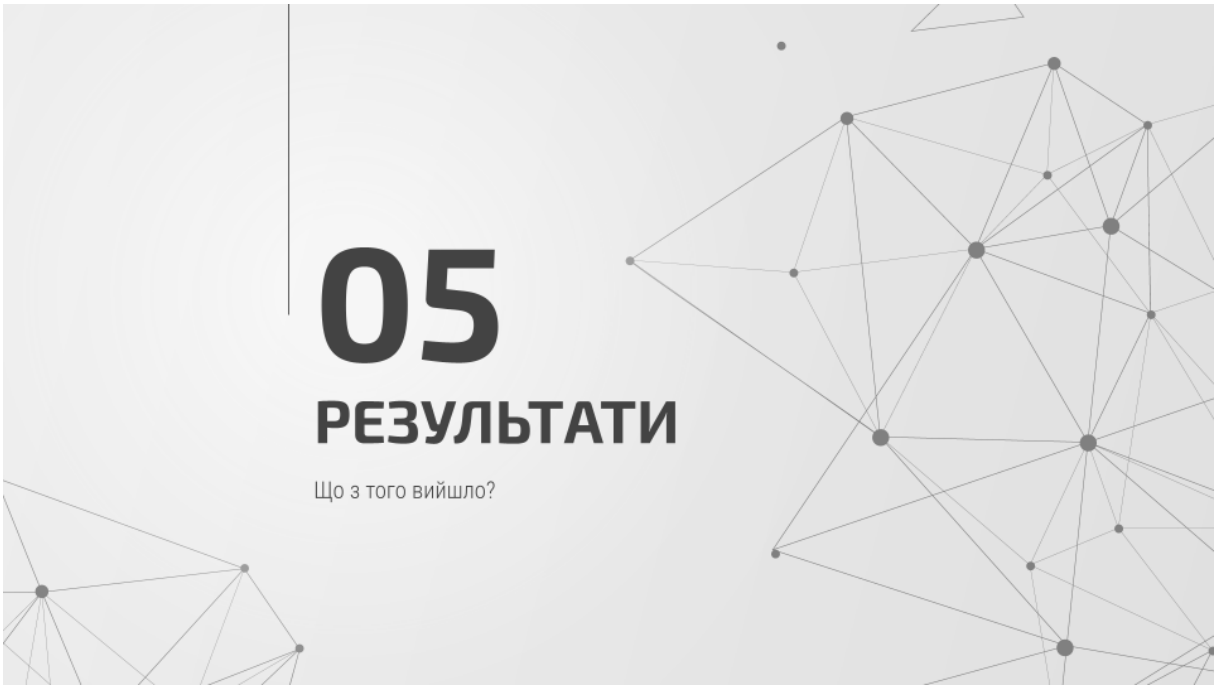
Короткочасне перетворення Фур'є (STFT) - це послідовність перетворень Фур'є віконного сигналу. STFT забезпечує інформацію про частоту, локалізовану в часі, для ситуацій, коли частотні складові сигналу змінюються з часом, тоді як стандартне перетворення Фур'є забезпечує інформацію про частоту, усереднену за весь інтервал часу сигналу.




04
РЕАЛІЗАЦІЯ
Що було зроблено?

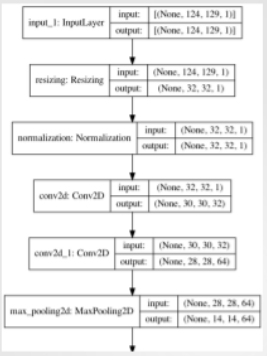

Опис датасету
Слова команди: [up, down, go, stop, left,right, yes, no]
8000 аудіозаписів
1000 аудіозаписів на кожне слово

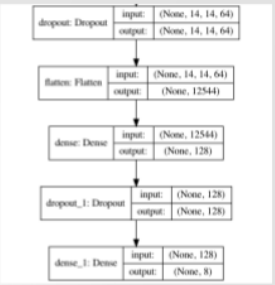

Пайплайн системи
Парсинг сигналу
Завантаження аудіо сигналу в систему та уніфікація його каналів
Створення waveform
Побудова аудіо хвилі з сигналу
Створення спектрограми
Побудова спектрограми для завантаження в модель
Завантаження моделі
Завантаження побудованої моделі у систему
Побудова передбачення
Обробка спектрограми моделлю та побудова передбачення
Вивід результатів
Побудова візуалізації результатів роботи моделі


05
РЕЗУЛЬТАТИ
Що з того вийшло?

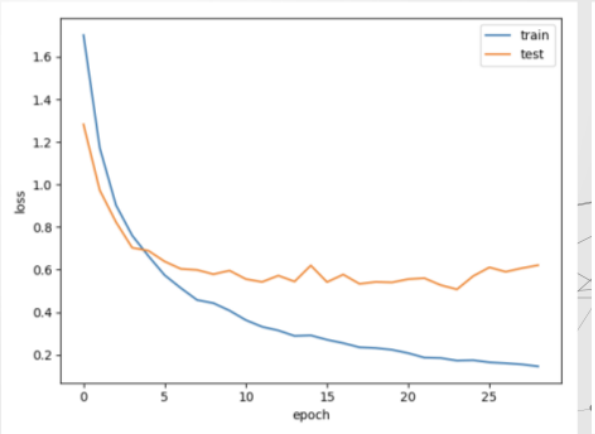
Результати побудови моделі

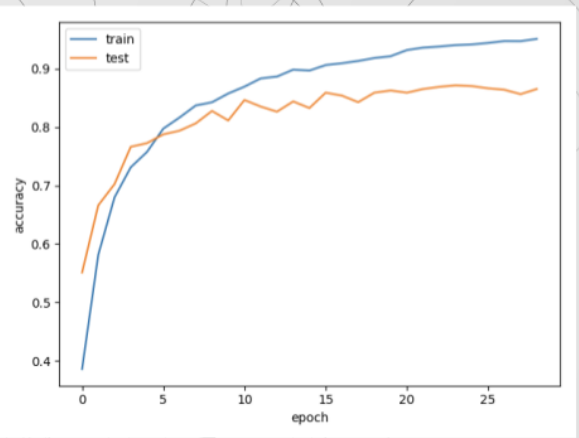
Для імплементації моделі використовується базовий класс models.Sequential пакету tensorflow. Цей класс дає можливість вручну побудувати шари згорткової мережі та налаштувати її саме під наші потреби.



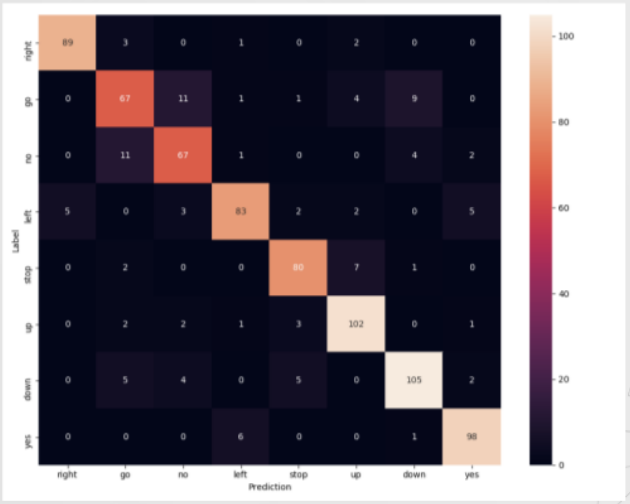


Learning Curves





Confusion Matrix



06

ВИСНОВКИ

То що ж ми дізнались?

Висновки

1. Реалізований функціонал обробки аудіо та перетворення в спектрограму
2. Побудована згорткова нейронна мережа CNN
3. Проаналізовані результати роботи мережі
4. Побудоване ПЗ для розпізнавання мови

Дякую за увагу!