Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерних наук та технологій
Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Олександр ЛИТВИНЕНКО

«____» _____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Програмний модуль розпізнавання голосових повідомлень з використанням нейронної мережі

Виконавець:_____Георгій ВОВК

Керівник: _____Дмитро КУЧЕРОВ

Нормоконтролер: _____Євгеній ТУПОТА

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп’ютерних наук та технологій
Кафедра комп’ютеризованих систем управління
Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»_____
Форма навчання денна_____

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____Олександр ЛИТВИНЕНКО

«___»_____ 2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
Вовк Георгія Романовича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема кваліфікаційної роботи: «Програмний модуль розпізнавання голосових повідомлень з використанням нейронної мережі» затверджена наказом ректора від «28» серпня 2023р. №1497/ст.

2. Термін виконання роботи (проєкту): з 02.10.2023 р. по 24.12.2023 р._____

3. Вихідні дані до роботи (проєкту): мова програмування *Python*, середовище розробки *Google Colab*, бібліотека *TensorFlow*, згорткова нейронна мережа

4. Зміст пояснювальної записки: Доцільність голосового розпізнавання. Розпізнавання нейромережею за технологією глибокого навчання. Розроблення системи розпізнавання. Тренування нейромережі. Тестування системи розпізнавання.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1. Схема алгоритму;

2. Анатомія моделі класів мережі;

3. Приклад інтерфейсу користувача.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Ознайомитись з постановкою задачі дипломного проектування	13.10.2023	
2	Вивчити спеціальну наукову літературу і технічну документацію	20.10.2023	
3	Проаналізувати методи розпізнавання голосу його передачі, дослідити VoIP протоколи	27.102023	
4	Написати розділ 1.	03.11.2023	
5	Провести огляд існуючих розробок. Визначити актуальність систем із звукового розпізнавання. Дослідити нейронні мережі та бібліотеки для їх застосування	10.11.2023	
6	Написати розділ 2.	24.11.2023	
7	Розробити програмний модуль розпізнавання голосу із застосуванням згорткових нейронних мереж	01.12.2023	
8	Написати розділ 3.	08.12.2023	
9	Оформити пояснювальну записку	14.12.2023	
10	Підготовити графічний демонстраційний матеріал	18.12.2023	

7. Дата видачі завдання: “2” жовтня 2023 р.

Керівник кваліфікаційної роботи _____ Дмитро КУЧЕРОВ
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Георгій ВОВК
(підпис студента) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Програмний модуль розпізнавання голосових повідомлень з використанням нейронної мережі»: 80 сторінок, 29 рисунків, 2 таблиці, 1 додаток, 31 інформаційне джерело.

НЕЙРОННІ МЕРЕЖІ, ТЕНЗОР, СПЕКТРОГРАМА, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, *Python, Google Colab.*

Об’єктом дослідження є процес розпізнавання голосових повідомлень на основі нейромереж.

Предметом дослідження є розпізнавання голосових команд із використанням згорткової нейронної мережі, та перетворення звукового сигналу в спектрограми.

Метою кваліфікаційної роботи є розробити програмно-технологічне рішення для розпізнавання голосових повідомлень, використовуючи сучасні нейронні мережі.

Обробка звукових сигналів є ключовим елементом багатьох сучасних систем обслуговування, в різних можливих сферах. Забезпечення користувачів стабільним програмними рішенням, позбавленим недоліків великих корпоративним рішенням допомагають продовжувати роботу багатьох сервісів, забезпечуючи при цьому їхню швидкодію, автономність, багатоплатформність та конфіденційність.

Прогнозами та припущеннями щодо розвитку об‘єкта дослідження є запропонований метод для розпізнавання голосових повідомлень буде мати ряд переваг перед поширеними методами із розпізнавання голосових повідомлень.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ОС – Операційна система

ПЗ – Програмне забезпечення

***IDE* (*Integrated Development Environment*) –** це середовище для розробки програмного забезпечення, яке об'єднує в собі різні інструменти та сервіси для полегшення роботи розробників. Основні компоненти, які можуть входити в *IDE*, включають текстовий редактор для написання коду, компілятор або інтерпретатор для виконання коду, налагоджувач для виявлення та виправлення помилок, інструменти для керування версіями, а також інші допоміжні сервіси.

***CNN* (*Convolutional neural network*) –** Згорткова нейронна мережа, яка використовує різновид багатошарових перцептронів, розроблений так щоб вимагати використання мінімального обсягу попередньої обробки.

Спектрограма – візуальне зображення спектру частот сигналу в часі. Найбільш поширеним уявленням спектрограми є двовимірна діаграма: на горизонтальній осі представлено час, вертикальної осі – частота

IoT (Internet of Things) – це концепція, що описує мережу фізичних об'єктів (пристроїв, автомобілів, будинків, датчиків та інших об'єктів), які з'єднані між собою і з Інтернетом. Ці об'єкти оснащені технологією збору та обміну даними, що дозволяє їм взаємодіяти, обмінюватися інформацією та виконувати певні завдання без прямого втручання людини.

ВСТУП

Актуальність теми. Сучасний стан систем із розпізнавання голосових повідомлень піддається критиці через ряд факторів, таких як залежність від хмарних сервісів та необхідність постійного стабільного підключення до глобальної мережі інтернет. Також необхідно виділити що при користуванні такими системами часто доводиться погоджуватись на користувацькі умови цих сервісів, які не завжди є оптимальними, особливо для великих компаній та корпорацій з власною політикою конфіденційності. Багато компаній саме через відсутність вибору використовують поширені системи для розпізнавання голосу від великих корпорацій. Відсутність власних рішень може заважати їх повноцінному та автономному функціонуванню, без залежності від доступу до мережі, і неможливості швидко та стабільно працювати незалежно від навантаження на хмарні сервіси, та їх працездатність в даний момент.

Більшість систем голосового розпізнавання створені для загального користування, що звісно не завжди заважає використовувати їх в вузько направлених сферах, але інколи заважає ефективному використанню ресурсів та коштів.

Основним застосуванням розпізнавання голосових повідомлень та команд являються переважно *IoT* системи та голосові асистенти, розпочинаючи від лампочок, телевізорів та іншої домашньої техніки яка вмикається по голосовій команді, закінчуючи портативними асистентами, які в будь-який момент дадуть відповідь на будь-яке ваше запитання, достатньо лише назвати ім’я помічника, та вимовити запитання. Сфера застосування також розповсюджується на медичні системи та системи безпеки, а в сфері логістики уже можна зустріти навіть голосове управління обладнанням.

Враховуючи сучасні особливості розвитку технологій із розпізнавання голосу, та їх адаптації в усіх можливих сферах використання, мати власний інструмент для

виконання задач із голосового розпізнавання, адаптований сугубо під ваші потреби та запити, допоможе зберегти час, та більш ефективно виконувати поставлені задачі.

Мета і завдання виконання кваліфікаційної роботи. Мета виконання кваліфікаційної роботи – розробити мультиплатформенний програмний модуль для розпізнавання голосових команд, який відповідає сучасним вимогам, може працювати автономно, або виконувати функції розпізнавання на власному хмарному сервісі.

При розгляді програмного модуля, що обробляє звукові сигнали і таким чином розпізнає голосові повідомлення доволі часто можна наткнутись на зв’язок програм розпізнавання не лише із більш звичними методами розпізнавання сигналів, а із доволі таки новими нейронними мережами. Нейронні мережі в таких запитаннях являються доволі універсальним та абсолютним рішенням, адже їх використання дозволяє програмному модулю розвиватись і навчатись разом із мережею, так ще й не виключає можливості застосування інших методів обробки сигналів. Актуальність застосування нейронних мереж в таких питаннях порівняно із іншими існуючими методами полягає в кількох аспектах, а саме можливість навчання, швидкість, зручність використання і інтеграції, гнучкість, а також можливість врахування різних перепон в процесі навчання моделі.

Завдання роботи включають:

1. Дослідити особливості мереж із передавання мультимедіа;
2. Аналіз існуючих рішень для розпізнавання природної мови;
3. Вивчити та дослідити процес розпізнавання голосових сигналів;
4. Дослідити можливості розпізнавання голосу за допомогою нейромереж;
5. Розробити ефективний програмний модуль.

Мета кваліфікаційної роботи – створення програмного модуля, який зможе із високою точністю розпізнавати голосові команди.

Об’єкт і предмет дослідження. Об’єкт кваліфікаційної роботи – створення програмного модулю розпізнавання голосових команд із застосуванням нейронної мережі.

Предмет кваліфікаційної роботи – модуль розпізнавання голосових команд із застосуванням нейронної мережі.

Методи дослідження. Щоб досягти виконання зазначених завдань кваліфікаційної роботи було вирішено працювати в інтегрованому середовищі розробки *Google Colab*. Дане середовище було обрано через його здатність об'єднувати процеси розробки, підтримувати різні технології, проводити хмарні розрахунки на виділених серверах з сучасним апаратним обладнанням, яке забезпечує високу швидкість навчання та загальну швидкодію нейронних мереж.

Мовою програмування для реалізації модуля було обрано *Python*. Мова була вибрана через велику кількість підтримуваних технологій та бібліотек, а також чудову підтримку бібліотеки *TensorFlow*, яка дозволяє працювати із нейронними мережами будь-якої складності та масштабу.

Наукова новизна отриманих результатів. Новизна отриманих результатів полягає в застосуванні згорткових нейронних мереж для розпізнавання голосу. Такий підхід дозволяє забезпечити оптимальне використання ресурсів та підвищити загальну точність розпізнавання голосових команд.

Використання нейронних мереж завдяки можливості легко адаптуватися до різних умов, обмежень та цілей користувача, підкреслюють своє безспірне лідерство серед усіх методів розпізнавання та обробки звуко-голосових повідомлень.

Нейронні мережі, завдяки можливості їх навчання та тренування, здатні аналізувати інформацію при нечітких або невикористовуваних раніше параметрах або вхідних даних. Саме ця їх особливість надає їм перевагу над звичними методами, і дозволяє доволі ефективно працювати навіть в умовах з далеко не найкращою якістю вхідних даних, дозволяючи швидко обробляти інформацію і використовувати її в подальшому.

Системи розпізнавання інформації з пристроїв вводу на основі нейронних мереж доволі часто є менш складними в реалізації, саме ця перевага робить їх привабливими для більш широкого кола застосувань, а в довгостроковій перспективі якість розпізнавання в таких системах може лише покращуватись завдяки новим навчальним даним.

Особистий внесок випускника. Всі результати, представлені у кваліфікаційній роботі, отримані випускником особисто. Особливої уваги заслуговує налаштування згорткової нейронної мережі та процес її навчання, задля досягнення високих показників точності, навіть при високій подібності слів.

Прогнозні припущення про розвиток об’єкту та предмету дослідження. Полягають у можливості застосування розробленого програмного модулю як системи для розпізнавання голосових команд. Подальший розвиток даного модулю відкриває багато перспектив та можливостей користуватися власними бібліотеками для ускладнення задачі із розпізнавання.

РОЗДІЛ 1
АНАЛІЗ СТАНУ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1. Мережі передавання мультимедійних даних

Мережі передавання даних це свого роді певні термінали зв’язку, функціонал яких полягає в передачі певних видів інформації на різні відстані, зазвичай це передавання різних символів. Основними складовими будь-якої системи для передачі даних є комутатори та канали передачі даних. Комутатор працює на канальному рівні моделі *OSI*, об’єднуючи пристрої в одну мережу і створює між ними канал зв’язку по якому передається інформація.

Процес передавання даних може відбуватись як послідовно, так і паралельно. У першому випадку біти відправляються одним шляхом, і це вимагає меншої обробки сигналу, а появлення помилки мінімізоване завдяки тому що можна легко передати контрольну цифру. Під час паралельного передавання одночасно пересилаються елементи сигналу що дозволяє досягти більшої швидкості передавання інформації, такий метод можна зустріти в архітектурі сучасних комп’ютерів.

Передавання даних також можна поділити на два різні підходи в організації комунікації, різниця в яких полягає в їхньому виконанні операції та керуванні часом. Синхронне передавання даних несе у собі часову узгодженість, що означає що відправка та отримання відбувається одночасно. А клієнт який надсилає інформацію блокується, та не може продовжувати користуватись каналом до завершення передачі інформації на сторону отримувача. Асинхронний метод передавання ж навпаки не встановлює ніяких часових обмежень між відправником та отримувачем, і відправник може надсилати інформацію в будь-який час не дочікуючись завершення попередньої передачі. Завдяки цьому відправник може продовжувати

виконання будь-яких інших задач, не дочікуючись завершення операції яка йде в даний момент часу.

Саму ж мультимедіа являє собою комбінування форм представлення інформації таких як звук, текст, зображення, відео та анімацію. В випадку передачі мультимедіа слід виділити протоколи, зображені у таблиці 1.1.

Таблиця 1.1

Протоколи передачі мультимедіа

Протокол	**Місце застосування**
HTTP/HTTPS	Веб-сервери
FTP (File Transfer Protocol)	Передача великих файлів між серверами та/або клієнтами
RTSP (Real-Time Streaming Protocol)	Передача відео або аудіо в реальному часі за допомогою технології стрімінгу
RTMP (Real-Time Messaging Protocol)	Передача відео або аудіо в реальному часі за допомогою технології стрімінгу, краще адаптований для онлайн-трансляцій
WebSockets	протокол двосторонньої взаємодії між клієнтом та сервером в реальному часі
Peer-to-Peer (P2P)	Протокол обмін даних в якому відбувається безпосередньо між двух клієнтів, завдяки встановленому між ними безпосередньому зв'язку
Bluetooth або Wi-Fi Direct	Протокол для обміну даними між пристроями які знаходяться або в радіусі дії *Bluetooth*, або підключені до однієї *Wi-Fi* мережі
MQTT (Message Queuing Telemetry Transport)	Протокол для передачі мультимедійних повідомлень в *IoT* системах, та системах негайного сповіщення

В залежності від вимог користувача і конкретної системи в якій буде проводитись передача мультимедійних файлів слід вибрати протокол який задовільнить вимоги клієнтів, між якими буде проводитись обмін даними.

1.2. Технологія підтримки пакетної передачі мови

Пакетна передача мови - це метод передачі аудіоданих, де звуковий сигнал розбивається на невеликі пакети, які передаються мережею окремо. Цей підхід сприяє ефективнішому використанню пропускної здатності мережі та забезпечує покращену якість передачі звуку у порівнянні з іншими методами.

Однак у певних ситуаціях може виникнути виклик, коли розмір пакету перевищує пропускну здатність вузлів зв'язку в системі. У протоколі *IP* така ситуація вирішується шляхом фрагментації пакету на менші частини. Під час фрагментації одного пакету може бути розділено на кілька менших пакетів, які відправляються окремо через мережу.

Клієнт, якому призначено отримати інформацію, отримує ці розділені пакети і повинен їх зібрати в один пакет перед подальшим аналізом. У мережах *IP* цей процес відомий як *IP*-фрагментація, і він забезпечує правильну передачу даних в умовах обмеженої пропускної здатності.

Розбиття на пакети для передачі мови складається з таких етапів:

1)Аналогове або цифрове записування- початковий запис сигналу, та за потреби його оцифровування;

2) Кодування або компресія- виконується для зменшення обсягу даних за допомогою застосування таких кодеків, як *MP3, AAC, Opus*;

3) Розбиття на пакети- поділ сигналу на невеликі пакети даних, кожен з яких містить фрагмент аудіо та додаткові метадані, зокрема порядкова нумерація, чи час створення пакетів, для зручності подальшої обробки пакетів;

4) Формування пакетів для мережі- підготовка для передачі пакетів через налаштовані канали зв'язку, за потреби додавання додаткових метаданих, та іншої інформації для подальшої маршрутизації пакетів на рівні мережевих протоколів;

5) Передача через мережу- передача пакетів за допомогою протоколів передачі даних, *RTP* (*Real-Time Transport Protocol*), *HTTP/HTTPS* чи інших протоколів які здатні забезпечити можливість передачі в реальному часі із високою якістю та низькою затримкою;

6) Розпакування та відтворення- пакети розпаковуються та відтворюються на клієнті отримувача.

Пакетна передача мови доволі широко використовується в сучасних системах телефонії, відеоконференцій, стрімінгових сервісів та навіть в *IoT* мережах, де важлива реальність часу та ефективність передачі даних.

1.3. Протоколи *VoIP* з'єднань

VoIP (Voice over Internet Protocol) — це технологія передачі голосу, звуків та відео через мережі, які працюють за *IP* протоколом. Початково термін *VoIP* використовувався для опису внутрішньокорпоративного зв'язку у формі цифрових телефонних дзвінків.. Проте з розвитком інформаційних технологій та доступності інтернету, *VoIP* став не просто засобом корпоративного зв'язку, але й універсальним інструментом комунікації для користувачів у всьому світі, дозволяючи здійснювати голосові та відеодзвінки з будь-якої точки планети за допомогою Інтернет-з'єднання.

Протоколи *VoIP* визначають стандарти та правила, що стосуються передачі звукових повідомлень через мережу інтернет. Ці протоколи стали невід'ємною частиною сучасних технологій телефонного зв'язку в інтернеті. Однією з основних переваг є економія витрат на телефонний зв'язок для компаній та домогосподарств. VoIP дозволяє здійснювати голосові та відеодзвінки, надсилати голосові та відео повідомлення, а також здійснювати голосовий пошук через мережу інтернет.

Серед основних протоколів забезпечення ефективного та стандартизованого голосового та відеозв'язку через мережу інтернет варто виділити протоколи зображені в таблиці 1.2.

Початок таблиці 1.2

VoIP протоколи

Протокол	**Опис**
SIP (Session Initiation Protocol),	Найпоширеніший протокол *VoIP*. Використовується для ініціювання, зміни та завершення сесій між двома і більше клієнтами. *SIP* також дозволяє передачу голосових, відео та інших мультимедійних даних.
H.323	Стандарт для передачі голосу та відео через мережу Інтернет. Використовується для відеоконференцій, голосового зв'язку та інших мультимедійних додатків.
RTP (*Real-Time Transport Protocol*)	Використовується для передачі аудіо- та відеопотоків в реальному часі. В основному використовується в поєднанні з іншими протоколами, такими як *SIP* чи *H.323*.
RTCP (*Real-Time Control Protocol*)	Часто використовується разом з *RTP* для надання обміну контрольною інформацією про якість обслуговування, пакети і т. д.
MGCP (Media Gateway Control Protocol)	В використовується для керування мультимедійним обладнанням, таким як шлюзи між публічною телефонною мережею (*PSTN*) та *IP*-мережею.
H.248	Відомий як *Megaco*, використовується для управління обладнанням шлюзів та контролю мережевих ресурсів.
IAX (Inter-Asterisk eXchange)	Використовується головним чином в системах *VoIP* на основі *Asterisk*. Він об'єднує сигнальні та мультимедійні потоки в одному з'єднанні.

Використання *VoIP*-телефонії відкриває широкі можливості: компанії можуть ефективно взаємодіяти та здійснювати зв'язок за межами територій, а користувачі можуть використовувати *VoIP*-телефони в будь-якому місці, де є доступ до Інтернету.

Ця технологія стає важливою складовою сучасного бізнесу та побутового зв'язку, забезпечуючи ефективну та доступну засоби комунікації.

1.4. Характеристики IP-мережі та параметрів, що негативно впливають на якість відтворення мови. Затримки та джиттер в мережі Інтернет

Набір мережевих протоколів мережі інтернет, включаючи *IP* (англ. *Internet Protocol* — «міжмережевий протокол»), взаємодіє з *TCP* протоколом (англ. *Transmission Control Protocol* — «протокол керування передаванням») для ефективного управління передачею даних. *IP* є ключовим елементом, що об'єднує чотири сегменти мережі в єдину систему, забезпечуючи передачу пакетів даних між різними вузлами мережі через мережеві маршрутизатори. У мережевій моделі *OSI* він класифікується як протокол мережевого рівня.

Важливо відзначити, що, хоча *IP* забезпечує основну інфраструктуру для передачі даних, він сам не гарантує стовідсоткової доставки пакету до призначення, збереження його оригінального вигляду чи порядку. Іноді пакети можуть бути продубльовані, пошкоджені або навіть не прибути. Однак для забезпечення надійної доставки та інших гарантій використовується *TCP* протокол, який оперує на рівні транспортного рівня мережевої моделі *OSI*. Такі протоколи вищого рівня вводять додаткові можливості для забезпечення цілісності та послідовності передачі даних у мережі.

Затримка в мережі інтернет виникає, коли пакети з інформацією передаються від одного клієнта до іншого, а потім повертаються до відправника з непередбачуваною затримкою. Цей явище може відображати швидкість, з якою пристрій взаємодіє з іншими клієнтами або серверами. Затримка в мережі прямо залежить від якості з'єднання, а не від потокової швидкості обміну інформацією з мережею інтернет. Затримка може виникнути з різних причин, таких як обмежена пропускна здатність мережі, перевантажені маршрутизатори чи комутатори, а також географічні відстані між пристроями. Затримка в мережі може впливати на

продуктивність та ефективність обміну інформацією, особливо у випадках, коли необхідно швидко відповідати на запити чи виконувати реального часу дії.

Затримку прийнято називати словом *Ping*, а вимірювати її в мілісекундах. На більшості сучасних пристроїв присутня однойменна утиліта (Рис 1.1), яка відправляє запити *ICMP Echo-Request* на вказаний користувачем вузол, і фіксує наявність відповідей *ICMP Echo-Reply*. За допомогою цієї утиліти також можна визначити затримку (*Ping*), двосторонні затримки (*RTT*, від англ. *Round Trip Time*) за маршрутом, а також обрахувати частоту втрати пакетів під час їх повернення.

Джиттер- коливання затримки, при наявності яких можна зробити висновок про те що в роботі мережі є якісь недоліки.

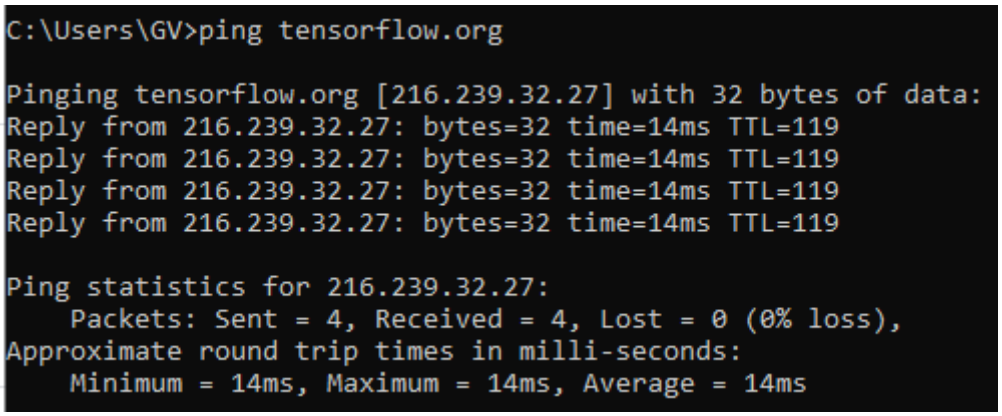

Рис. 1.1 Робота команди «*ping*»

Низька затримка та відсутність джиттеру в сучасних мережах мають критичне значення для покращення користувацького досвіду, особливо при використанні стрімінгових сервісів та *VoIP*-протоколів. Затримка, яка майже непомітна, сприяє плавній та миттєвій передачі даних, а відсутність джиттеру, або нерівномірності в часовому інтервалі передачі даних, дозволяє уникнути перерв та розривів у трансляціях або в дзвінках через мережу інтернет.

Висока затримка в свою чергу негативно впливає на можливість передачі мультимедіа в реальному часі у використанні різноманітних сервісів, адже затримка стає відчутною та заважає сприйняттю подій у режимі реального часу. Присутність високого джиттеру стає причиною збоїв у дзвінках, викривлення аудіо та відео, а

іноді навіть може призводити до появи ехо під час *VoIP*-дзвінка. Зокрема, висока затримка та джиттер можуть спричиняти втрату пакетів, коли частина інформації не надходить до отримувача, що порушує цілісність передачі даних. Ці фактори стають важливими при взаємодії з мережами та визначають, наскільки ефективно можна використовувати різноманітні онлайн-сервіси та технології для комунікації у реальному часі.

1.5. Висновки до розділу

За результатами проведеного дослідження мереж передавання мультимедійних даних надано визначення поняття мережі, розглянуто процес послідовної та паралельної передачі інформації між клієнтами мережі, та визначено їх недоліки та переваги один над одним. Визначено термін мультимедіа, та розглянуто основні протоколи, які використовуються для передачі мультимедіа даних між пристроями в мережі, визначено їх особливості.

У процесі аналізу технології пакетної передачі мови, важливим аспектом є розуміння етапів, які відбуваються під час поділу та передачі аудіоінформації у вигляді пакетів через Інтернет-протокол (*IP*). Фрагментація пакетів у *IP* мережах відіграє ключову роль у забезпеченні ефективної та надійної передачі звукових даних.

На початковому етапі аудіосигнал розбивається на невеликі частини або пакети, які подальше піддаються фрагментації. Фрагментація полягає у розбитті кожного пакету на менші фрагменти для адаптації до обмежень мережі та підвищення ефективності передачі. Цей процес дозволяє оптимально використовувати ресурси мережі та забезпечує надійну передачу даних. Після фрагментації, отримані фрагменти пакетів передаються через мережу за допомогою Інтернет-протоколу. Передача відбувається незалежно для кожного фрагмента, що дозволяє оптимізувати використання пропускної здатності та прискорює процес обміну інформацією.

Призначенням цього процесу є забезпечення стабільної та неперервної передачі голосу через мережу, при цьому система гарантує відновлення та збереження цілісності аудіоданих навіть у випадку втрати чи затримки певних пакетів. Це робить технологію пакетної передачі мови ефективним інструментом для голосового зв'язку через інтернет.

Під час ретельного аналізу *VoIP* протоколів були визначені та розглянуті найпопулярніші серед них, висвітлено їхні ключові особливості та функціонал. Важливим етапом вивчення стала розглядана проблематика використання *VoIP* у мережі інтернет як засобу передачі аудіо та відео інформації між користувачами.

Додатково, зосереджено увагу на тому, як *VoIP* може бути використаний в інтернет мережі для передачі аудіо та відео інформації між користувачами. Вивчено різні аспекти взаємодії та обміну даними, а також розглянуто можливості оптимізації якості звуку та відео в рамках *VoIP* технологій. Підкреслено те що *VoIP* протоколи є ключовим елементом сучасних комунікаційних технологій та відіграють важливу роль у забезпеченні ефективного та надійного обміну голосовою та відео інформацією у мережі інтернет.

Дано коротку характеристику *IP* мереж. Виділено основні проблеми передачі інформації в *IP* мережах. Розглянуто *TCP* протокол який виступає ефективним рішенням для управління передачею даних в *IP* мережах, забезпечуючи надійність і відновлення в разі втрати пакетів. Це робить його особливо корисним у вимогливих до якості зв'язку додатках, таких як *VoIP*. Використання *TCP* дозволяє ефективно передавати інформацію в мережі, забезпечуючи послідовність та точність, що робить його ідеальним для вимогливих до послідовності даних застосувань, таких як передача файлів чи відеоконференції.

Визначено наслідки затримки та джиттера в *IP* мережі, як вони впливають на мережу, і проблеми які вони викликають у користувачів. Визначено кінцеві проблеми з якими зіштовхнуться користувачі *VoIP*, якщо в їх мережі присутня велика затримка, або високі значення джиттеру.

У реальних умовах використання *VoIP*-технологій, важливою є не лише стійкість до затримок та джиттеру, але й вирішення кінцевих проблем, які можуть

виникнути у користувачів при великих значеннях затримки та високому рівні джиттеру в їхній мережі. Забезпечення якості голосового зв'язку через Інтернет вимагає комплексного підходу до оптимізації та управління мережею, враховуючи особливості передачі голосових даних в режимі реального часу.

Використання технологій, таких як *VoIP*, стає все більш важливим у сучасному світі, адже вони надають ефективний та економічний засіб для здійснення голосового зв'язку через Інтернет. Порозуміння основних аспектів передачі даних в мережах та їх вплив на якість голосового зв'язку дозволяє розробникам та адміністраторам ефективно вдосконалювати та оптимізувати мережеві інфраструктури для найкращого використання технологій *VoIP*.

РОЗДІЛ 2
НЕЙРОМЕРЕЖІ ДЛЯ РОЗПІЗНАВАННЯ ГОЛОСУ. МОДЕЛЬ *TENSORFLOW*

2.1. Огляд існуючих розробок

Сьогодні на ринку існує безліч різних застосунків для розпізнавання голосу та голосових повідомлень, з різним функціоналом. Розглянемо найпопулярніші із них.

Із готових рішень для розпізнавання тексту можна виділити такі системи:

1. Голосові помічники: Популярні голосові помічники, такі як *Siri* від *Apple*, *Google Assistant* від *Google, Alexa* від *Amazon* і *Cortana* від *Microsoft*, використовують розпізнавання голосу для розуміння та виконання голосових команд користувачів.
2. Технології розпізнавання голосу в мобільних додатках: Багато мобільних додатків ігнорують текстовий ввід і надають можливість користувачам взаємодіяти з ними через голос. Це може бути використано для пошуку, навігації, введення тексту та інших завдань.
3. Транскрипція голосових записів: Бізнеси і індивіди використовують технології розпізнавання голосу для автоматизованої транскрипції голосових записів, таких як інтерв'ю, конференції або нотатки зустрічей.
4. Системи безпеки та контролю доступу: Розпізнавання голосу використовується в системах біометричної автентифікації для забезпечення

безпеки, таких як системи входу до будівель або доступ до банківських рахунків.

5. Медичні застосування: Розпізнавання голосу може бути використано для медичних додатків, таких як реєстрація медичних даних або розпізнавання мовленнєвих порушень.
6. Автомобільна індустрія: Системи розпізнавання голосу вбудовуються в сучасні автомобілі для управління різними функціями, такими як навігація, виклик голосових команд і відтворення музики.
7. Комп'ютерні ігри і розваги: У деяких комп'ютерних іграх і розважальних додатках використовується розпізнавання голосу для взаємодії з персонажами та управління грою.
8. Системи автоматизації та домашньої автоматизації: Деякі системи домашньої автоматизації використовують розпізнавання голосу для керування освітленням, температурою та іншими пристроями.

В наш час технології розпізнавання голосу здобули широке застосування у різних галузях, відзначаючись ключовою роллю в еволюції віртуальних асистентів, систем управління, аудіоаналітики та інших сучасних застосувань. Надійне та точне розпізнавання голосу визначається за допомогою високотехнологічних методів, зокрема використання потужних алгоритмів глибокого навчання.

Особливо важливими у цьому контексті є рекурентні нейронні мережі (*RNN*) та згорткові нейронні мережі (*CNN*). Вони володіють унікальними властивостями, що дозволяють ефективно опрацьовувати та аналізувати акустичні шаблони голосу. *RNN*, завдяки своїй здатності враховувати контекст і залежності між послідовними елементами, забезпечують високу точність у завданнях розпізнавання мови, а *CNN* використовуються для виділення ключових ознак в аудіосигналах, що робить їх ефективними у виявленні особливостей голосу.

Загальна тенденція в галузі полягає в використанні цих методів для створення потужних систем розпізнавання голосу, які здатні забезпечувати високий рівень точності та швидкість обробки. Такі технології визначають майбутнє розвитку

інтерфейсів з користувачем та інновацій у сферах автоматизованого управління та обробки аудіоданих.

Рекурентні нейронні мережі (*RNN*) використовують зворотний зв'язок, що дозволяє враховувати контекст та залежності між послідовними фрагментами голосу. Це особливо корисно при розпізнаванні мовлення та інших виразів, де важливий контекстуальний аналіз.

Згорткові нейронні мережі (*CNN*), у свою чергу, можуть виявляти важливі звукові ознаки в голосі, розпізнаючи шаблони і характеристики звукового сигналу. Це дозволяє ефективно аналізувати частотні характеристики голосу та розпізнавати його особливості.

Великі технологічні компанії, такі як *Google*, *Amazon*, *Microsoft* і *IBM*, активно вкладають зусилля у розвиток та надання інструментів для розпізнавання голосу в хмарному середовищі. Це відкриває нові перспективи для розробників, які можуть легко інтегрувати функції розпізнавання голосу у свої додатки та системи.

Інфраструктура хмарних сервісів, надана такими компаніями, стає каталізатором для доступу до високоефективних та високоточних рішень у сфері розпізнавання голосу. Це не лише спрощує розробку голосових додатків, але й робить їх доступними для широкого кола користувачів. Інтеграція таких інструментів відзначається зручністю використання та можливістю розширення функціоналу додатків.

Ці хмарні сервіси надають розробникам інноваційні інструменти для створення інтерактивних та інтелектуальних голосових інтерфейсів. Вони відкривають нові горизонти для розробки додатків, які взаємодіють з користувачами у найбільш природний та зручний спосіб, створюючи унікальні та прогресивні можливості взаємодії.

Не ігноруючи те що розпізнавання голосу в хмарному середовищі відкриває безліч переваг, також це супроводжується і рядом мінусів. Зокрема, однією з головних проблем є конфіденційність та безпека даних. Збереження голосових даних в хмарних сервісах може викликати занепокоєння щодо можливості несанкціонованого доступу до цих даних або їхнього використання без належного

контролю. Додатковим аспектом є залежність від стабільного і швидкого інтернет-з'єднання. Відсутність з'єднання або його недостатня швидкість може призвести до недостатньої ефективності роботи голосових сервісів, що становить виклик для користувачів. Також питання етичності та приватності стають дедалі більш актуальними в контексті збору і обробки голосових даних. Важливо вирішувати етичні аспекти та гарантувати правильність управління приватністю користувачів. Крім того, існують виклики з точки зору локалізації даних, оскільки різні країни можуть мати різні вимоги щодо збереження та обробки особистої інформації. І нарешті обмежена налаштовуваність може бути проблемою для тих, хто прагне максимально адаптувати розпізнавання голосу під свої конкретні потреби чи бізнес-вимоги.

2.2. Актуальність систем із розпізнавання голосових повідомлень

Системи розпізнавання голосу мають велику актуальність у багатьох сферах життя і сучасному технологічному середовищі з численними важливими застосуваннями. Ось деякі з ключових аспектів їх актуальності:

1. Голосові помічники та інтерфейси: Голосові помічники, такі як *Siri, Google Assistant, Alexa* і *Cortana*, набули великої популярності в розумних домівках, мобільних пристроях, автомобілях і інших пристроях. Вони дозволяють користувачам взаємодіяти з технологією із зручністю голосових команд і запитів.
2. Медичні застосування: Системи розпізнавання голосу можуть використовуватися для реєстрації медичних даних, діагностики мовленнєвих розладів, розпізнавання ритму та інших медичних завдань.
3. Автоматизація індустрії: У виробництві і логістиці системи розпізнавання голосу можуть бути використані для керування пристроями та машинами через голосові команди, що підвищує продуктивність та безпеку робочого процесу.

4. Особиста безпека та автентифікація: Розпізнавання голосу може бути використано для систем автентифікації, що дозволяє підвищити рівень безпеки доступу до пристроїв і даних.
5. Освіта та доступність: Голосові інтерфейси спрощують доступ до технології для людей з різними видами обмежень та забезпечують нові можливості в освіті та навчанні.
6. Системи автомобільної безпеки та навігації: В автомобільній промисловості розпізнавання голосу дозволяє керувати автомобілем через голосові команди, що покращує безпеку та зручність під час водіння.
7. Автоматизована обробка голосу і даних: Розпізнавання голосу може бути використано для автоматизованої обробки голосових записів, транскрипції, вилучення інформації та аналізу великих обсягів голосових даних.
8. Пошук інформації: Голосовий пошук і голосові помічники дозволяють швидко та зручно знаходити інформацію в мережі та здійснювати запити без використання клавіатури або сенсорного введення.

З розвитком технологій штучного інтелекту, глибокого навчання та обробки природної мови системи розпізнавання голосу стають все точнішими і різноманітними в своїх можливостях. Це робить їх актуальними і корисними в багатьох сферах нашого життя.

2.3. Бібліотека *TensorFlow*, тензори та підтримка нейронних мереж

TensorFlow - це відкрита бібліотека для машинного навчання і глибокого навчання, розроблена компанією *Google* в 2015 році як відкрита бібліотека. Вона стала однією з найпопулярніших бібліотек для роботи з нейронними мережами і широко використовується у наукових дослідженнях та промисловому застосуванні. Модель широко застосовується компанією *Google* у багатьох своїх сервісах, таких як *Google* фото та голосовий асистент.

TensorFlow визначається своєю унікальною концепцією графів потоку даних, яка служить основою для моделювання та виконання операцій машинного навчання. Граф потоку даних є потужним інструментом, який визначає, як дані переміщуються між різними операціями, створюючи ефективний шлях для обчислень у нейронних мережах та інших моделях машинного навчання.

Ця концепція реалізується через визначення графа, де вузли представляють операції, а ребра - потік даних між цими операціями. *TensorFlow* використовує граф потоку даних для опису обчислень та взаємодії між різними частинами моделі. Кожна операція в графі відповідає конкретному обчислювальному етапу, а переміщення даних між цими операціями визначає потік інформації вздовж графа.

Нижче наведено приклад типового коду для побудови графа потоку даних у *TensorFlow*:

```
import tensorflow as tf

# Визначення констант та операцій
a = tf.constant(2)
b = tf.constant(3)
multiply_op = tf.multiply(a, b)

# Створення сеансу TensorFlow
with tf.Session() as sess:
    # Виконання графа потоку даних
    result = sess.run(multiply_op)
    print(result)
```

У цьому прикладі граф містить константи a та b, а операція *multiply_op* визначає множення цих констант. Під час виклику *sess.run(multiply_op)* граф потоку даних виконується, і результат обчислення виводиться.

Такий підхід дозволяє *TensorFlow* оптимально оптимізувати та розпаралелювати обчислення, роблячи його ефективним для широкого спектру завдань у галузі машинного навчання та глибокого навчання.

TensorFlow підтримує роботу з широким спектром моделей машинного навчання, включаючи звичайні нейронні мережі, рекурентні нейронні мережі, згорткові нейронні мережі та мережі трансформери.

В квітні 2016 р. *Google* випустив розширену версію *TensorFlow* з розподіленими можливостями глибокого навчання, трішки пізніше у *TensorFlow* додано підтримку *HDFS* (*TFoS*), і можливість паралелізму даних (Рис. 2.1-2.2).

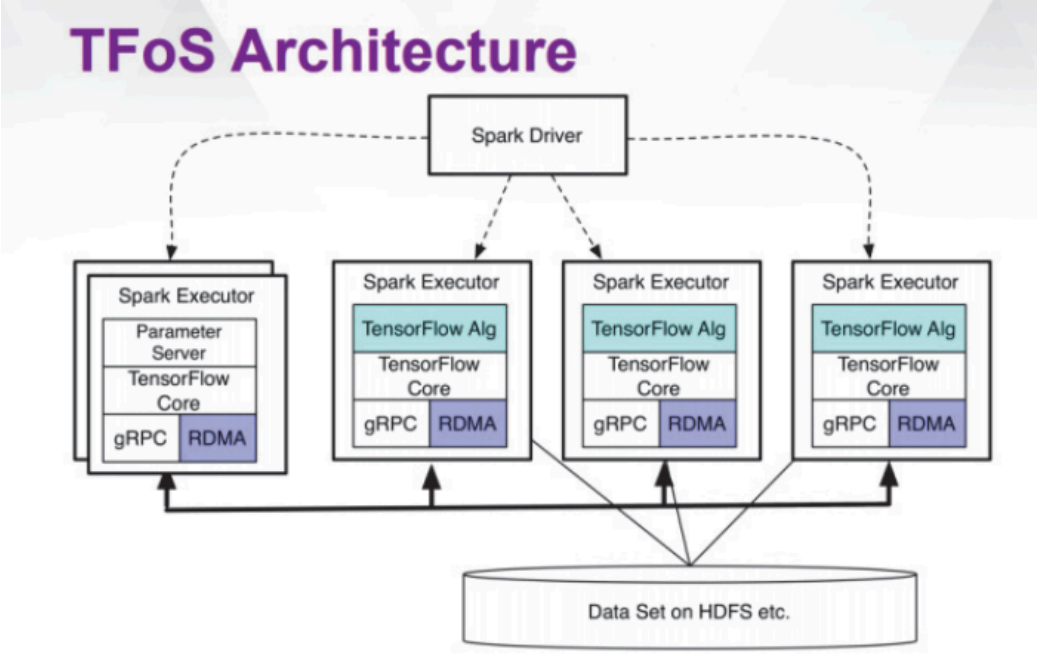

Рис. 2.1 Архітектура *TFoS*

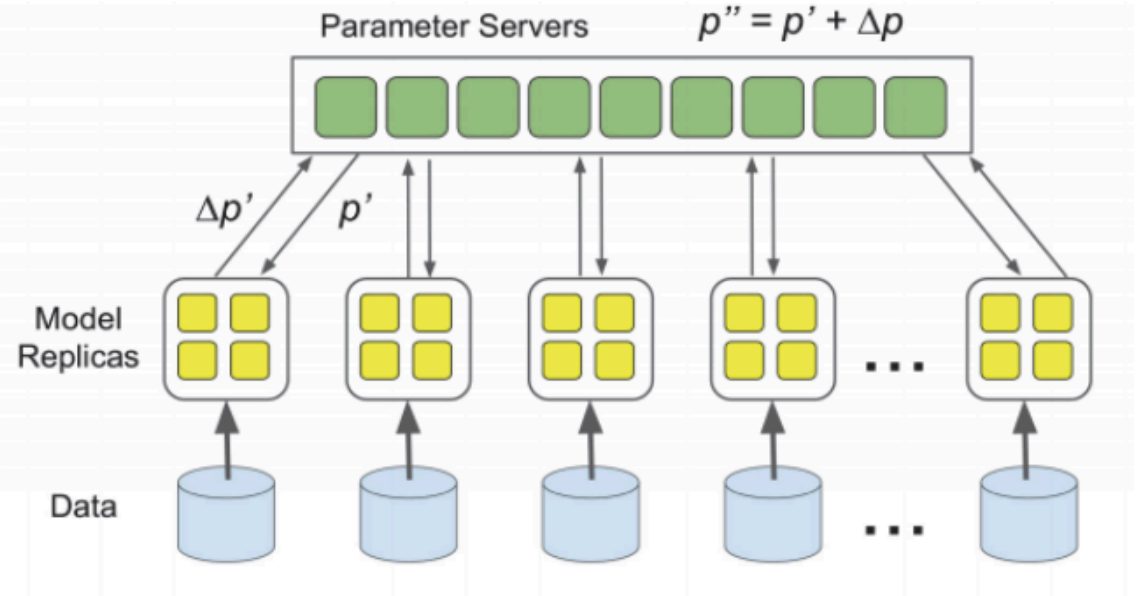

Рис. 2.2 Модель паралелізму даних у *TensorFlow*

В основі бібліотеки лежать обчислювальні графи та тензори. Графи мають вигляд колекції вузлів, які виконують операції, та ребер, через які проходять дані у вигляді тензорів. Самі тензори можна описати як *N*-мірні вектори, які в свою чергу подають *N*-вимірні набори інформації.

Тензори широко використовуються в науці та інженерії для представлення різноманітної інформації, такої як зображення, звуки, часові ряди, та інше. Вони забезпечують універсальний спосіб вираження складних даних та можливість виконання математичних операцій над ними. У глибокому навчанні, наприклад, нейронні мережі працюють з великими тензорами, представляючи ваги, вхідні дані та вихідні значення моделі.

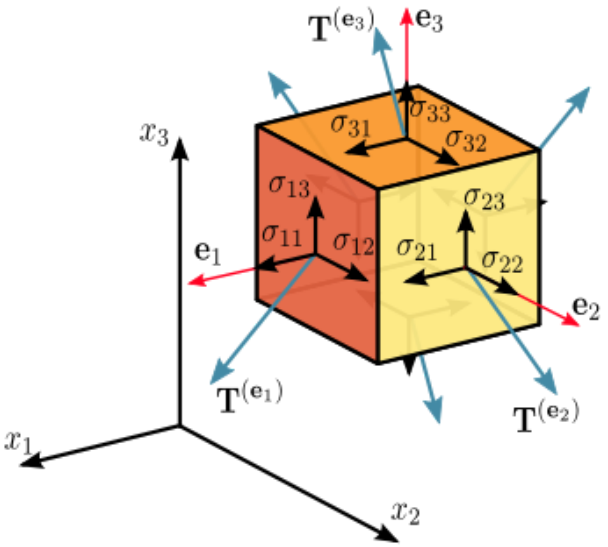

Рис. 2.3 Зображення тензору другого порядку

Математично тензор являє собою *N*-мірний вектор, який перетворюється певним чином при зміні системи координат. Це означає, що компоненти тензора залежать від вибору системи координат. Завдяки цій особливості він і може представляти собою *N*-мірні набори даних. Тензор узагальнює такі поняття як скаляр, вектор, ковектор, лінійний оператор і білінійна форма. Тензор являє собою абстракцію, що охоплює об'єкти різного рангу, такі як скаляри (тензори рангу 0), вектори (тензори рангу 1), матриці (тензори рангу 2) та більш складні структури вищого порядку, такі як тензори кривизни, які мають 100 компонентів, і завдяки цьому часто використовуються для зображення кривизни простору-часу . На рис. 2.4 зображено візуалізацію тензорів різних порядків.

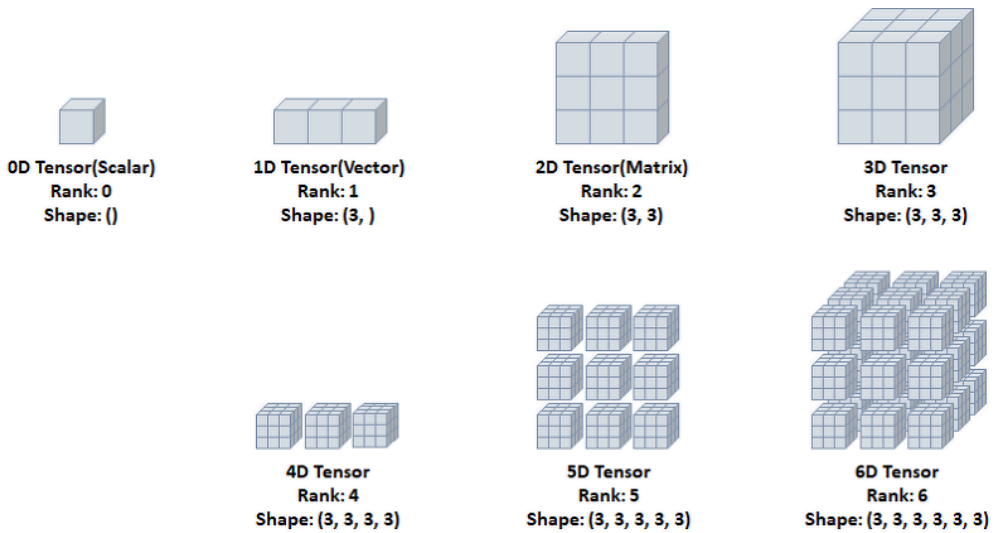

Рис. 2.4 Різні форми тензорів включно до 6ти вимірного.

Оскільки розміри даних продовжують зростати, використання тензорів стає невід'ємною частиною в сучасних дисциплінах, що використовують обробку та аналіз великих обсягів інформації. Тензори забезпечують універсальний та ефективний метод представлення різноманітної структурованої інформації, включаючи такі складні об'єкти, як зображення, аудіо, текстові дані та числові параметри.

При розгляді тензора у вигляді (3x3), його можна інтерпретувати як матрицю з 3 рядками і 3 стовпцями. Збільшуючи розмір до (1000x3x3), тензор можна уявляти як набір з 1000 матриць розміром 3x3 або як вектор, що складається з 1000 елементів, при цьому кожен елемент є матрицею розміром 3x3. Тут термін "форма" або "розмір" вказує на структуру результативного тензора, який у цьому випадку має форму (1000x3x3).

Важливою характеристикою тензорів є їхня гнучкість. Тензори можуть бути постійними (незмінними) або змінними, здатними динамічно адаптуватися до змін у вхідних даних або внутрішніх умов. Це робить їх особливо корисними в контексті алгоритмів машинного навчання, де моделі можуть адаптувати свої параметри під час навчання для досягнення кращої ефективності на різних завданнях. Обчислювальні графи. Термін "потік" відноситься до обчислювального графа, який ніколи не може містити циклічних залежностей. У графі кожен вузол представляє операцію, таку як додавання або віднімання, а кожна операція породжує новий тензор.

Початкові вузли в обчислювальних графах завжди представлені тензорами, що є основними структурами для представлення даних та виконання операцій. Це обумовлено тим, що операції, які відбуваються в мережах, вимагають наявності вхідних тензорів для виконання конкретної операції. Такий підхід гарантує, що всі необхідні дані наявні до того, як буде виконана обчислювальна операція, забезпечуючи логічний порядок виконання.

Графи обчислень завжди організовані ієрархічно, що означає, що операції розташовані в графі в певній ієрархічній структурі або послідовності. Починаючи з початкових вузлів, які представляють вхідні дані, граф веде до кінцевих вузлів, які можуть представляти вихідні дані чи результати певних обчислень. Це організаційне вирішення структури графа дозволяє ефективно визначати порядок виконання операцій та оптимізувати обчислення.

У графах обчислень вузли представляють операції, а ребра - передачу даних між цими операціями. З таким підходом ієрархічної організації, важливою є не лише наявність вхідних тензорів для кожної операції, але і правильний порядок їх обробки, що забезпечує коректне та ефективне виконання обчислень в мережах глибокого навчання та інших схожих застосувань.

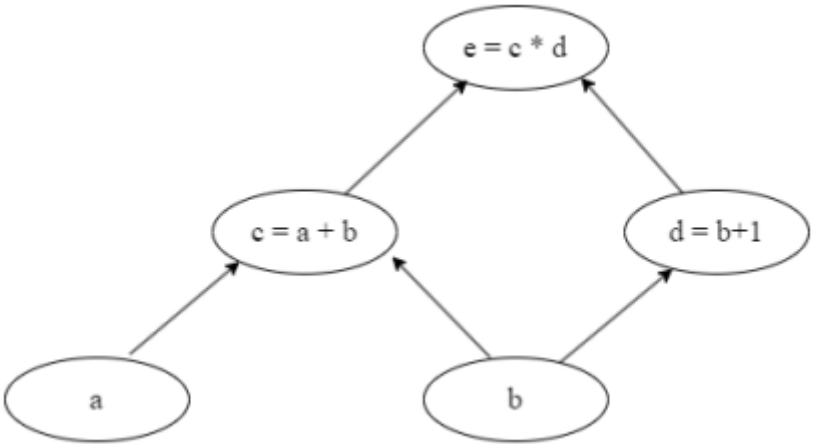

Рис. 2.3 Звичайний обчислювальний граф

Обертання графа в зворотному порядку є ключовим етапом у процесі обчислень та утворює важливу структуру для формування кінцевого виразу. Під час цього процесу граф розбивається на субвирази, які потім об'єднуються для створення остаточного виразу чи результату.

Під час руху вперед у графі, кожен вузол стає залежним від попереднього та впливає на наступний вузол. Наприклад, якщо ми маємо вузли *a, b, c, d* та *e*, де *c = a + b, a d = c* * 2, то в процесі обчислень *e = d + c* вимагатиме вже попереднього обчислення c та d. Така залежність між вузлами під час руху вперед ілюструє, як

кожен вузол в графі потребує результатів своїх попередніх вузлів для коректного обчислення.

Застосування зворотного порядку у графі дозволяє ефективно визначити, які вузли необхідно обчислити в першу чергу, аби забезпечити належний порядок виконання операцій. Наприклад, якщо ми хочемо обчислити e виходячи з графа, обертання графа в зворотному порядку допоможе визначити, що спочатку потрібно обчислити *d*, потім *c*, і тільки після цього можна перейти до обчислення e.

Варто відмітити що дії на вузлах одного рівня графа являються повністю незалежними від інших дій на тому ж рівні. Ця властивість може спостерігатись у графі на рис. 2.3, де вузли *c* і *d* на одному рівні не залежать один від одного, тому їх можна обчислювати паралельно.

Завдяки незалежності вузлів на одному рівні, вузли можна обчислювати паралельно. Ця властивість позитивно впливає на швидкодію моделі. Також *TensorFlow* здатна розподіляти навантаження між різними апаратними пристроями, тому одна задача може виконуватись на центральному процесорі, (*CPU*), а інша на графічному процесорі (*GPU*), звісно якщо обладнання відповідає необхідним вимогам. Розподілення операцій між різними пристроями керується потоками. Зазвичай інформація у вигляді тензорів передається між ними. На рисунку 2.4 зображено як форма тензора на пристрої *A*, надходить на пристрій *B*. Це створить певну затримку в системі, залежну від розміру тензору. *A* пристрій *B* не зможе працювати як належно, допоки не отримає інформацію с пристрою *A*.

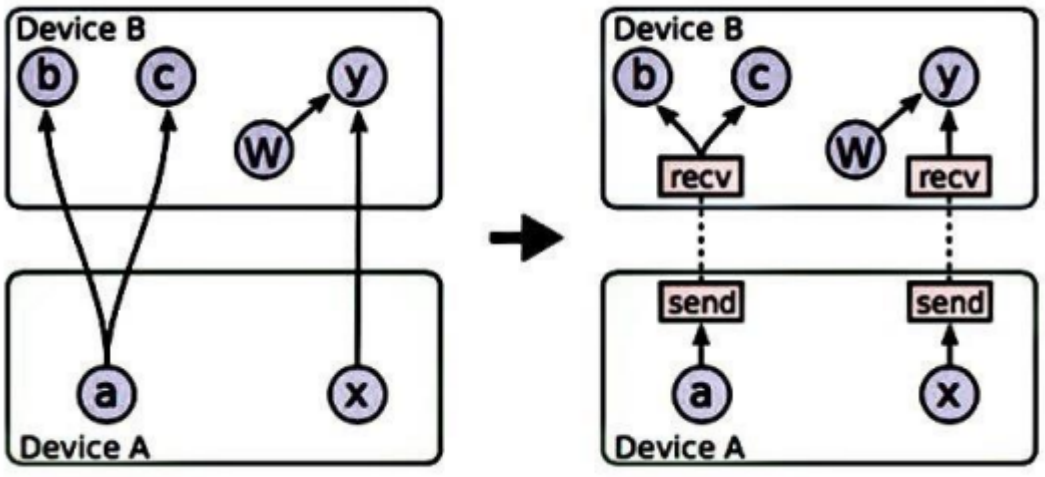

Рис. 2.4 Передача тензорів між різними вузлами.

TensorFlow має дві основні версії: *TensorFlow* 1.x і *TensorFlow* 2.x.. Версія *TensorFlow* 2.x. була важливим оновленням, яке принесло багато змін для поліпшення досвіду використанням бібліотекою. Що в свою чергу принесло неабиякий вплив в простому розуміння бібліотеки, і навіть людина яка лише розпочинає знайомство з нейронними мережами, зможе зрозуміти як саме працює мережа. Станом на сьогоднішній день, очікується реліз теперішньої бета версії 2.14.0-*rc1*, яка знаходиться в бета доступі з 28 серпня 2023 року.

До ключових особливостей *TensorFlow* слід віднести використання графа обчислень. Побудова обчислювальних графів, які представляють необхідну модель, і послідовне виконання обчислень на цих графах. Це дозволило користуватись паралельними обчисленнями, цим самим збільшивши продуктивність моделі. Завдяки ресурсу *TensorFlow Hub* у користувачів є можливість спробувати використати макети різних готових моделей та компоненти мережі. А також застосовувати їх у інших проектах. Це також покращує користувацький досвід, адже застосування готового шаблону значно спрощує початок роботи з мережею, та дозволяє ознайомитись з уже готовими до роботи мережами, а коли користувач зможе самостійно працювати і створювати власні моделі то вже зможе поділитись своїм досвідом та напрацюваннями із іншими користувачами. Варто відмітити що велика спільнота користувачів і розробників також постійно добавляє нові матеріали для роботи з мережею, тому навряд-чи можна опинитись в ситуації коли не знайдеться потрібної документації.

TensorFlow, крім того, що є однією з провідних бібліотек для роботи з глибинним навчанням, надає широкий спектр можливостей для розробки та навчання різноманітних моделей. Зокрема, *TensorFlow* відзначається високою ефективністю у підтримці нейронних мереж, включаючи застосування згорткових мереж (*CNN*) та рекурентних мереж (*RNN*).

Згорткові мережі (*CNN*) в *TensorFlow* використовуються для обробки зображень, де їхні особливості виявляються та використовуються для різноманітних

завдань, таких як класифікація об'єктів чи визначення паттернів. *TensorFlow* забезпечує потужні функції для роботи зі зображеннями, включаючи операції навчання та обробки даних, що робить його ідеальним інструментом для завдань, пов'язаних із зображеннями.

Рекурентні мережі (*RNN*) у *TensorFlow* використовуються для роботи з послідовними даними, такими як тексти або часові ряди. Це дозволяє створювати моделі, які можуть розуміти та враховувати контекст в даних залежно від попередніх обчислень. *TensorFlow* надає різноманітні шари та операції для роботи з рекурентними мережами, що полегшує їхню реалізацію та навчання.

TensorFlow включає в себе потужний інструмент *TensorBoard*, який відкриває широкі можливості для візуалізації та аналізу процесу навчання моделей глибинного навчання. *TensorBoard* є необхідним допоміжним інструментом, що надає розробникам та дослідникам зручні та ефективні засоби для візуалізації різних аспектів навчання.

Однією з ключових можливостей *TensorBoard* є його здатність надавати візуальне представлення метрик та параметрів моделі під час навчання. Розгортання графіків допомагає спостерігати за змінами в ході епох та з'ясовувати ефективність моделі. Крім того, *TensorBoard* надає можливість візуалізувати структуру моделі, що сприяє кращому розумінню її архітектури та потужності.

До інших корисних функцій *TensorBoard* входить відслідковування втрат та точності, вивчення розподілу ваг, відображення зображень та векторів, а також відстеження градієнтів під час навчання. Всі ці можливості дозволяють розробникам ефективно аналізувати та вдосконалювати моделі, а також виявляти можливі проблеми у процесі навчання.

Широкий спектр застосувань: *TensorFlow* може бути використано в різних галузях, включаючи машинне навчання, обробку природної мови, обробку зображень, комп'ютерне зорове сприйняття, обробку звуку та багато інших. Він використовується як для досліджень, так і для створення продуктів.

Версія *TensorFlow Lite*: *TensorFlow* також має версію під назвою *TensorFlow Lite*, призначену для використання на мобільних пристроях та вбудованих системах, де обмежені ресурси.

TensorFlow є не лише потужним, але й вкрай важливим інструментом для вивчення, розробки та впровадження моделей машинного навчання та глибокого навчання. Заснований на відкритому вихідному коді, цей фреймворк став ключовим інструментом для науковців, інженерів та розробників у галузі штучного інтелекту.

2.4. Нейронні мережі в *TensorFlow*

Нейронні мережі, також відомі як штучні нейронні мережі, складаються з взаємопов'язаних штучних нейронів або процесорів, які співпрацюють для вирішення задач. Штучні нейрони є простими обчислювальними одиницями, і їх взаємодія в мережі дозволяє виконувати складні обчислення. Нейронні мережі можуть бути навчені розв'язувати різноманітні завдання шляхом адаптації своїх ваг і структури під час процесу навчання.

TensorFlow підтримує широкий спектр нейронних мереж, включаючи мережі глибокого навчання, згорткові мережі, рекурентні мережі та мережі з довгою короткочасною пам’яттю. Це дозволяє вибрати найбільш відповідну архітектуру під потреби користувача.

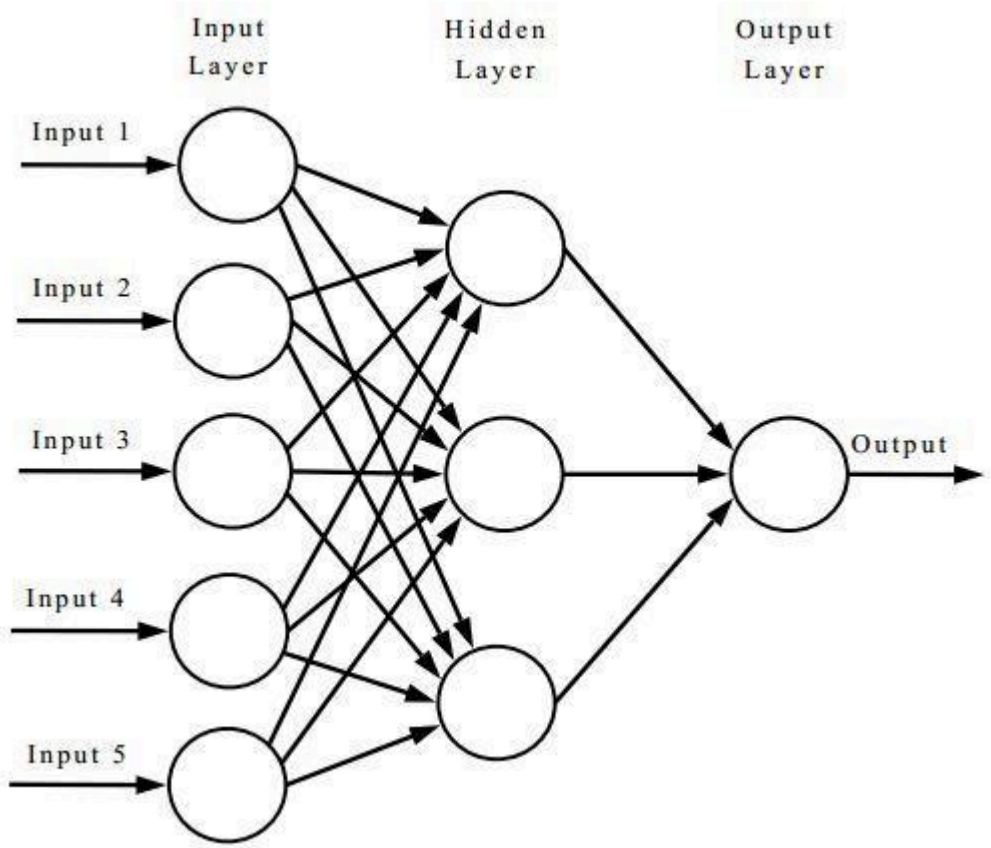

Рис. 2.5 Схема нейронної мережі

Кожна нейронна мережа складається з таких компонентів як:

1. Вхідний шар- отримує вхідні дані, розпочинаючи з числових характеристик, закінчуючи різними мультимедіа.
2. Прихований шар- отримує сигнали з входу, та виконує обчислення.
3. Вихідний шар- генерує вихідні дані чи прогнози на основі обчислень проведених в прихованому шарі.

Процес навчання в нейронних мережах проходить з учителем або без учителя. В першому випадку необхідно мати приклади з вірними відповідями (як приклад- мітки класів). В такому випадку мережа вивчить відповідність між вхідними даними та правильними відповідями. Навчання з учителем слід використовувати для покращення здібностей до класифікації, як приклад розпізнання зображень та мови.

Сам процес навчання зазвичай складається із двох етапів:

1. Пряме поширення (*forward propagation*) помилок (рис. 2.6);
2. Зворотне поширення (*backward propagation*) помилок (рис. 2.7).

У прямому поширенні мережа на вхід отримує дані та передає їх через внутрішні шари до вихідного шару, при цьому ще й генерує прогнозовані вихідні значення. Цим етапом користуються ваги, для саморегулювання, щоб обчислювати значення у прихованих та вихідному шарах.

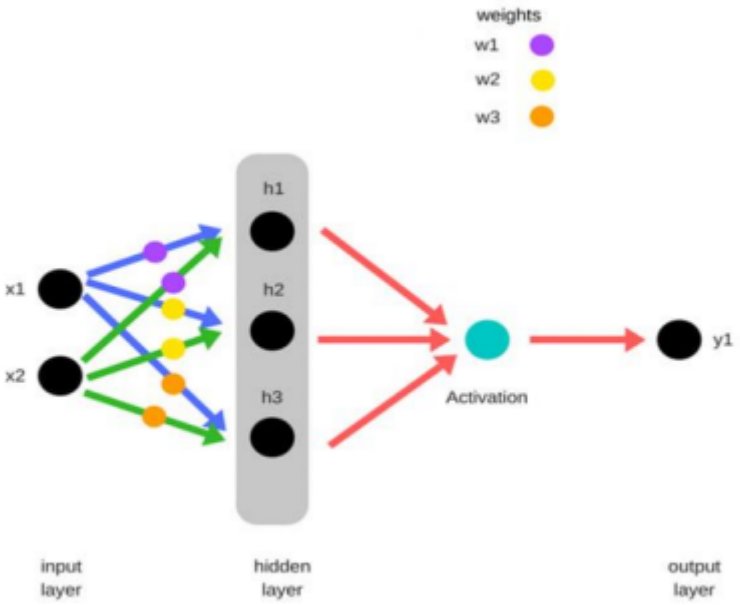

Рис. 2.6 Схема прямого поширення помилки

У зворотному поширенні помилок коригування ваг відбувається шляхом оцінювання помилок між правильними відповідями та прогнозованими результатами. Помилки поширюються назад по мережі, а ваги для мінімізації помилок, оновлюються з допомогою алгоритму градієнтного спуску. Оновлення ваг при такій схемі навчання проходить з урахуванням швидкості навчання (*learning rate*), яка фіксує ступінь коригування ваг в процесі навчання .

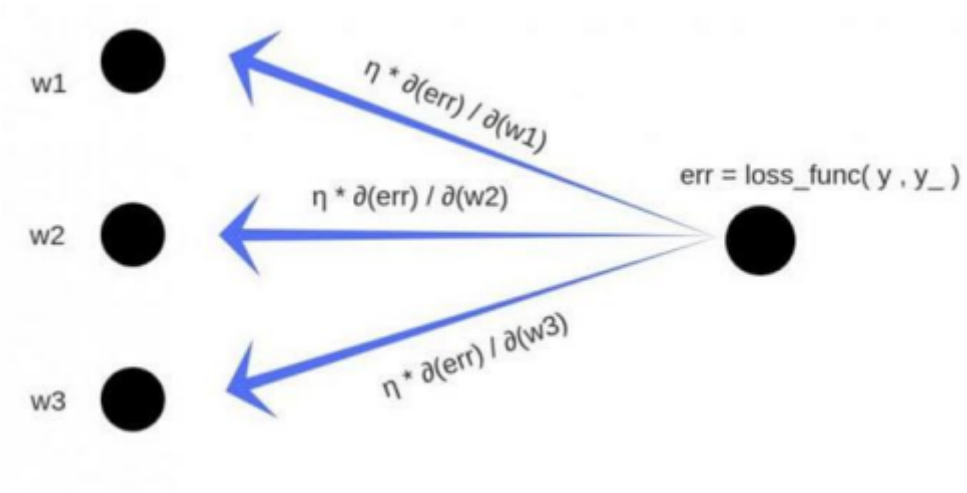

Рис. 2.7 Схема зворотного поширення помилки

В загальному процес навчання нейронної включає в себе ітерації прямого та зворотного поширення помилок допоки не буде досягнуто задовільний рівень точності мережі. Кількість ітерацій та інші параметри навчання можуть варіюватись залежно від конкретних задач та вхідних даних. Застосування нейронних мереж широко розповсюджене і застосовується в таких областях, як комп'ютерне зорове сприйняття, обробка природної мови, рекомендаційні системи, автономні автомобілі та літальні апарати і багато інших.

2.5. Згорткові та рекурентні нейронні мережі

1. Згорткові нейронні мережі (*Convolutional Neural Network, CNN, ЗНМ*) це нейронні мережі із спеціалізованою архітектурою, яка зазвичай використовується для обробки зображень і розпізнавання образів. Вона має певні унікальні властивості, які дозволяють ефективно виконувати завдання з обробки зображень та мови.

Однією з ключових відмінностей згорткових нейронних мереж (*CNN*) від звичайних нейронних мереж є їхній особливий підхід до обробки даних, який включає в себе дві основні операції: згортку та пулінг. Ці операції грають важливу роль у виявленні та використанні ключових ознак у вхідних даних, особливо у візуальних завданнях, таких як обробка зображень.

Згортка є першим етапом у відмінності згорткових нейронних мереж. Ця операція включає перемноження елементів матриці вхідних даних з невеликим фільтром або ядром. Цей фільтр ковзає по вхідних даних, виконуючи локальні операції з елементами, і створює карту ознак. Карта ознак визначає різні характеристики зображення, такі як контури, текстури або інші важливі деталі, що визначають його зміст.

Другим важливим етапом є пулінг, що здійснюється для зменшення розміру карти ознак. Операція пулінгу включає обчислення максимального або середнього значення в малих областях карти ознак. Це дозволяє зберігати ключові ознаки, але в меншому розмірі, що робить обробку даних ефективнішою та обчислювально менш витратною.

Двоетапний процес згортки та пулінгу в згорткових нейронних мережах дозволяє автоматично виділяти та використовувати важливі ознаки в даних, забезпечуючи ефективну та робочу архітектуру для різноманітних завдань обробки інформації.

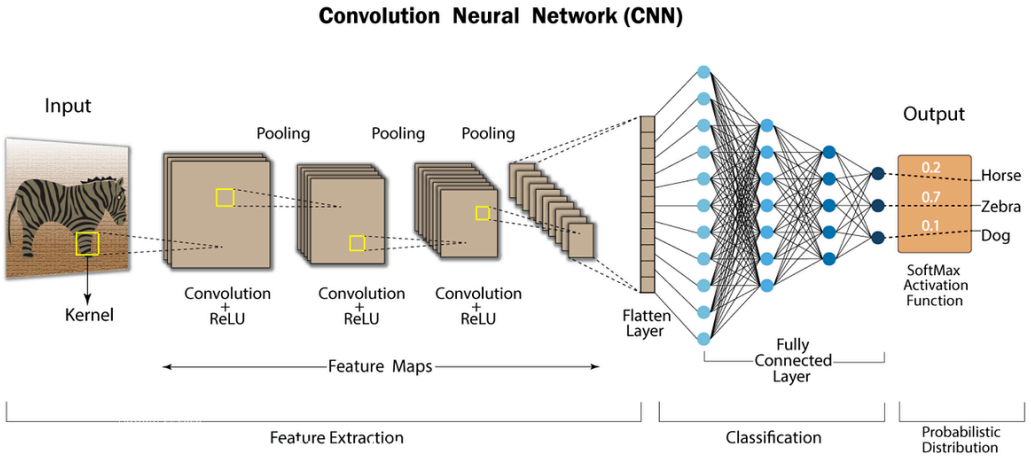

Рис. 2.8 Згорткова нейронна мережа

До основної переваги згорткових нейронних мереж слід віднести їх здатність до автоматичного виявлення різних рівнів абстракції в зображеннях. Нижні шари мережі виявляють прості шаблони, такі як границі і форми, тоді як вищі шари виявляють складніші об'єкти, наприклад, обличчя або тварини. Це робить їх особливо потужними для завдань розпізнавання образів.

Процес навчання згорткових нейронних мереж (*CNN*) також представляє собою ітеративний процес, але відрізняється своєю специфікою та особливостями, спрямованими на обробку візуальної інформації, зокрема зображень.

По-перше, дані вводяться в мережу у вигляді зображень. Згорткова операція, що є ключовою частиною архітектури *CNN*, застосовує фільтри або ядра до різних частин вхідного зображення, виконуючи операцію згортки. Це дозволяє виділяти різні візуальні ознаки, такі як границі, форми або текстури. Після кожної згортки створюється карта ознак, яка реагує на конкретні аспекти вхідного зображення.

Далі застосовуються операції пулінгу, які зменшують розмір карти ознак, зберігаючи ключові візуальні ознаки. Це сприяє зменшенню обчислювальних витрат та використанню більш абстрактних представлень вхідної інформації.

Після згорток та пулінгу використовуються повнозв'язані шари, які об'єднують вищезгадані ознаки та виконують прогнози на основі зібраних характеристик. Як і в інших типах нейронних мереж, порівняння проводиться з фактичними значеннями, і розраховується похибка моделі.

Для коригування ваг використовується алгоритм зворотного поширення помилок (*Backpropagation*), що дозволяє оптимізувати параметри мережі так, щоб зменшити величину похибки. При цьому враховується вся структура мережі, включаючи згорткові та повнозв'язні шари.

Цей процес навчання згорткової нейронної мережі підкреслює ефективність моделей у вирішенні завдань комп'ютерного зору та обробки зображень, забезпечуючи високу точність та здатність автоматично виокремлювати та розпізнавати важливі візуальні ознаки, що буде пізніше використано в ході аналізу спектрограм аудіофайлів.

Згорткові нейронні мережі використовуються у багатьох застосуваннях, таких як комп'ютерне зорове сприйняття, розпізнавання обличь, класифікація зображень, виявлення об'єктів, сегментація зображень та багато інших. Вони продемонстрували вражаючі результати в багатьох задачах машинного навчання і їхній здатності автоматично виявляти та використовувати важливі візуальні ознаки в навчальному

матеріалі. Завдяки цим перевагам вони стали одним з основних інструментів глибокого навчання в області обробки зображень та графічних матеріалів.

2. Рекурентні нейронні мережі (*Recurrent Neural Networks, RNN*) це особливий тип нейронних мереж, які мають власну внутрішню пам'ять та можуть обробляти послідовні дані, такі як текст, мову, часові ряди тощо. Вони використовують зв'язки з власним попереднім станом, що дозволяє враховувати контекст і історію вхідних даних.

Вони відзначаються важливою особливістю, яка визначає їхню унікальність у порівнянні з іншими архітектурами нейронних мереж - це наявність зворотного зв'язку. Цей аспект дозволяє інформації переміщатися не лише вперед, від вхідного до вихідного шару, а й у зворотньому напрямку, від вихідного шару до вхідного, враховуючи контекст і залежності між послідовними елементами даних.

Одна з ключових переваг використання зворотного зв'язку в *RNN* полягає в тому, що вихідне значення на кожному часовому кроці може служити вхідним значенням для наступного часового кроку. Це інтегрує інформацію з попередніх часових кроків у поточний, надаючи моделі можливість "пам'ятати" попередні стани та використовувати їх у подальших обчисленнях. Наприклад, при вирішенні задачі прогнозування наступного слова в реченні *RNN* може використовувати контекст попередніх слів для кращого прогнозування. Це дозволяє моделі розуміти взаємозв'язки та логічні зв'язки в послідовностях даних, що є важливим у великому спектрі завдань, таких як машинний переклад, генерація тексту та обробка мови.

У рекурентних нейронних мережах використовуються два типи вузлів: вузли введення (*input nodes*) і вузли прихованого стану (*hidden state nodes*). Вузли введення приймають вхідні дані, а вузли прихованого стану зберігають попередній стан мережі. Кожен вузол прихованого стану має свою внутрішню пам'ять, що дозволяє йому зберігати інформацію про попередні стани і використовувати її при обробці нових вхідних даних.

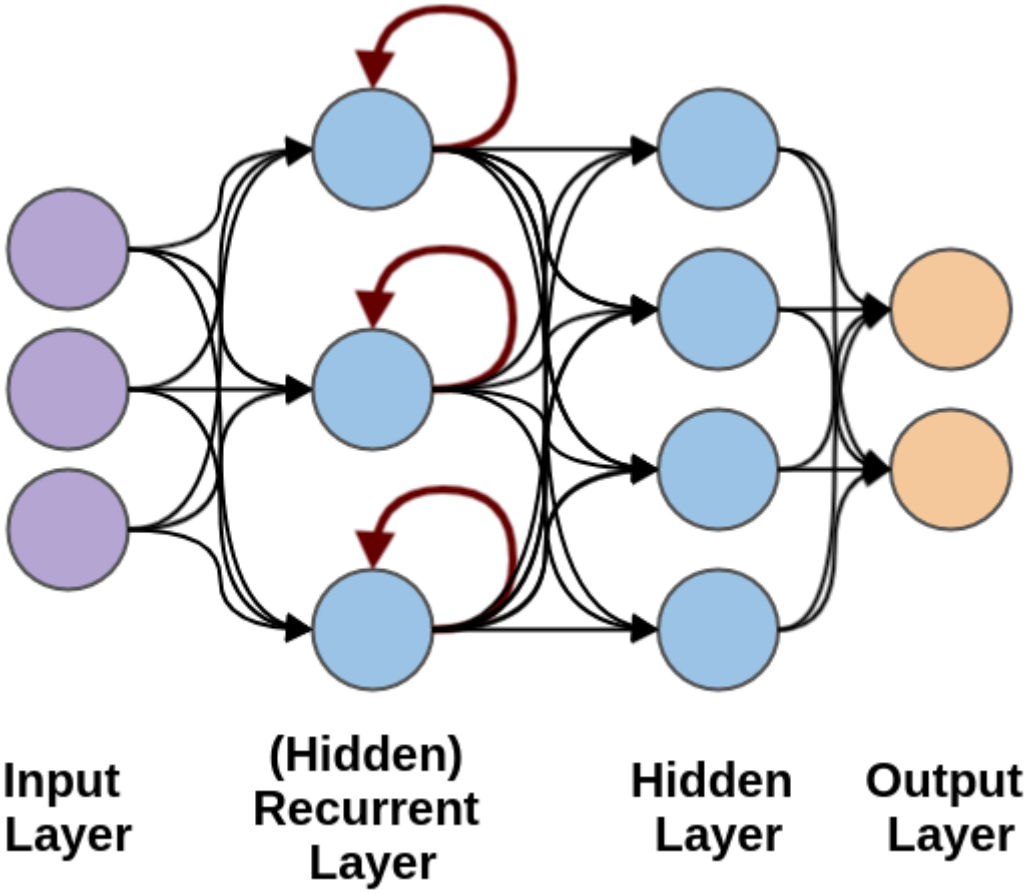

Рис. 2.9 Рекурентна нейронна мережа

Процес навчання рекурентних нейронних мереж включає у себе ітеративний цикл, який враховує послідовність вхідних даних, виконує прогнози для вихідних значень та коригує внутрішні ваги моделі для покращення її точності. Цей процес дозволяє *RNN* враховувати залежності в часі та використовувати контекст попередніх вхідних елементів для кращого розуміння та прогнозування майбутніх значень.

Спочатку дані вводяться в мережу послідовно, частинами або по одному часовому кроці. Кожен часовий крок розглядається окремо, а внутрішні стани мережі оновлюються відповідно до вхідних даних та попереднього стану. Потім, за використанням цих оновлених станів, модель генерує вихідне значення.

Після отримання вихідного значення, порівняння проводиться з фактичними (очікуваними) значеннями, і визначається похибка моделі. Для коригування ваг та оптимізації моделі використовується алгоритм зворотного поширення по часу

(*Backpropagation Through Time, BPTT*). Цей алгоритм враховує похибки на кожному часовому кроці та поширює їх назад через всю мережу, щоб коригувати ваги.

Важливим аспектом цього процесу є розуміння та управління затуханням та вибухом градієнтів, що може виникнути через тривалі послідовності. Це може вимагати використання додаткових технік, таких як обрізка градієнтів, для забезпечення стабільності та ефективності процесу навчання.

Таким чином, процес навчання рекурентних нейронних мереж є складним та деталізованим, але він забезпечує мережу здатністю враховувати динаміку даних та робити точні прогнози для послідовних вхідних елементів.

2.6. Висновки до розділу

У даному розділі розглянуто готові рішення, які пропонують функціонал з розпізнавання природньої мови. Виділено їх переваги та недоліки перед власними рішеннями. Розглянуто сфери застосування цих рішень.

В ході аналізу визначено актуальність систем із розпізнаванням мови, проаналізовано можливі варіанти їх застосування. Виділено основні типи нейронних мереж на яких будуються системи із голосового розпізнавання.

Розглянуто бібліотеку *TensorFlow* як основний інструмент для реалізації нейронних мереж, охарактеризовано тензори як масиви даних, описано роботу графів та вузлів в нейронних мережах. Дано коротку характеристику моделі *TensorFlow* та суміжної екосистеми. Виділено зручність та простоту роботи з моделлю та суміжними сервісами, можливість подальшої швидкої інтеграції, та переходу на іншу мову програмування, що в свою чергу позитивно вплинуло на вибір бібліотеки для подальшого використання.

Особливо сильним *TensorFlow* робить його універсальність та здатність працювати з різними типами моделей. Він підтримує важливі архітектури, такі як згорткові мережі (*CNN*) та рекурентні мережі (*RNN*), що відкриває безмежні можливості для розв'язання завдань обробки зображень, обробки мовлення, класифікації та багатьох інших.

Загальною перевагою використання *TensorFlow* є його універсальність, охоплюючи різні аспекти глибинного навчання, включаючи обробку зображень, обробку мовлення та інші завдання, що визначає його як потужний інструмент для вивчення та застосування в сучасних дослідженнях та розробках.

Окрім своєї потужності у розробці моделей, *TensorFlow* пропонує розширені інструменти для візуалізації та аналізу процесу навчання через *TensorBoard*. Це сприяє вдосконаленню моделей, шляхом аналізу метрик, втрат та оптимізації гіперпараметрів, що робить його невід'ємним інструментом для кращого розуміння та налаштування моделей.

У наукових та промислових застосуваннях *TensorFlow* залишається невід'ємним інструментом, що користується великою популярністю та довірою. Він успішно використовується для розв'язання актуальних наукових завдань, де вимагається високий рівень точності та ефективності. Незалежно від області застосування - від обробки зображень та аудіо до природної мови та глибокого навчання - *TensorFlow* став надійним компаньйоном для дослідників та інженерів.

Активна спільнота розробників, яка регулярно обмінюється знаннями та ресурсами, робить *TensorFlow* екосистемою, що живе та постійно розвивається. Це дозволяє розробникам залишатися на передових позиціях у галузі штучного інтелекту, отримуючи доступ до новітніх інструментів та технологій. Оновлення фреймворку регулярно впроваджують нові функціональності та виправлення, що допомагає тримати проекти на кращому технічному рівні.

Розробники, які обирають *TensorFlow* для своїх проектів, можуть бути впевнені в його здатності впроваджувати передові інновації у сфері машинного навчання. Його відкрита архітектура і гнучкість дозволяють створювати високоефективні та науково обґрунтовані рішення, що відповідають високим стандартам якості та результативності. Таким чином, *TensorFlow* продовжує служити каталізатором для інновацій та вдосконалення у галузі машинного навчання, підтримуючи прогрес та розвиток цієї динамічної області.

Охарактеризовано процес навчання моделей, які використовуються в *TensorFlow*, подано схеми рекурентних та згорткових нейронних мереж, описано

способи їх навчання, а також методи навчання із учителем та без нього. В ході аналізу згорткових і рекурентних нейронних мереж та аналізу їх можливих способів використання безпосередньо виділено особливості роботи згорткових нейронних мереж із візуальними даними, які вплинули на їх вибір для використання у модулі розпізнавання. Описано процес коригування коефіцієнтів вхідних даних під час навчання та тренування мережі.

РОЗДІЛ 3
РОЗПІЗНАВАННЯ ГОЛОСУ В НЕЙРОМЕРЕЖІ *TENSORFLOW*. ПІДГОТОВКА НАВЧАЛЬНИХ ДАННИХ ТА ПРОЦЕС НАВЧАННЯ

3.1. Початок тренування мережі.

Для початку роботи з мережею необхідно підготувати аудіодані, у форматі *.wav* на основі яких відбуватиметься розпізнавання та тренування мережі. Для цього буде використано набір *speech_commands*, а саме слова *«no», «yes», «down», «go», «left», «up», «right»*, та *«stop».* Також для спрощення розуміння того, що буде робити мережа створено спектрограми базових слів, які система буде розпізнавати. Візуалізація частот за допомогою двовимірної діаграми також безпосередньо відбувається в нейромережі, за допомогою короткочасного перетворення Фур'є. І на основі змін частоти на протязі часового проміжку мережа може зробити висновок яке слово було подано на вхід

На рис. 3.1-3.6 продемонстровано спектрограми власного запису слів, які використовувались для тренування мережі. Запис проводився на мікрофон «*YTOM M1PRO»* в режимі запису 16-*bit* 44.1*kHz*, гучність пристрою зчитування в налаштуваннях звуку windows була зафіксована на 70%. Запис проводився в кімнаті з відсутньою шумоізоляцією та акустичною корекцією. Навіть без використання спеціальних програм для аналізу та порівняння двовимірних діаграм, можна знайти подібність у графіках слів. Також можна прослідкувати деяку подібність між графіками слів *«Go»* та *«No»*, що може викликати неточності в роботі програми.

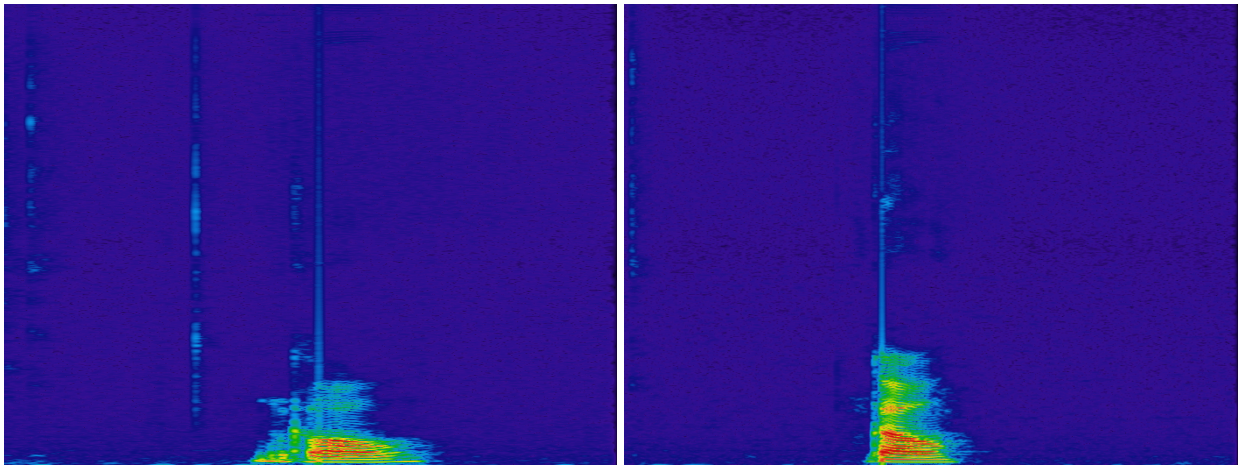

Рис 3.1 Спектрограми слова «Go»

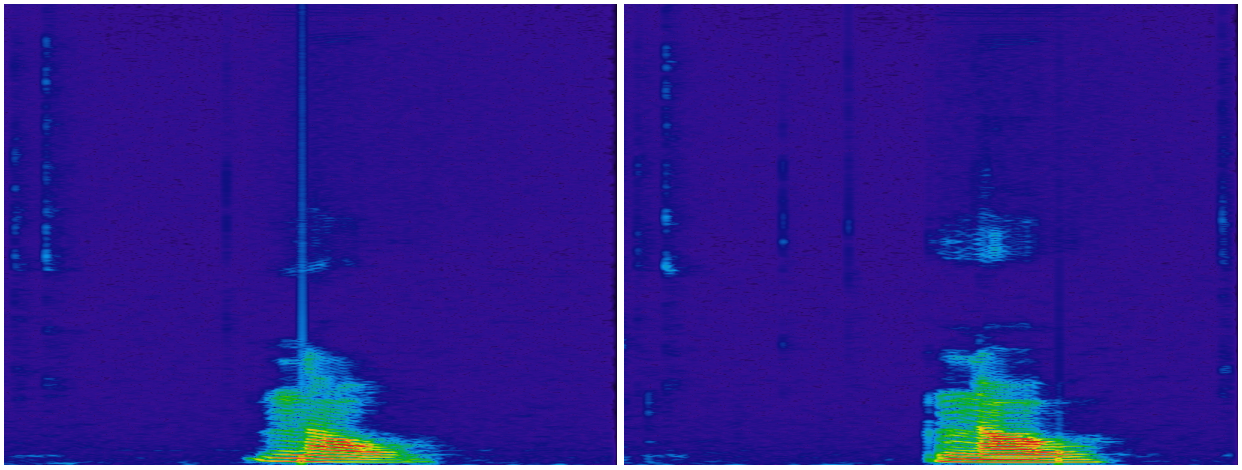

Рис 3.2 Спектрограми слова «*No*»

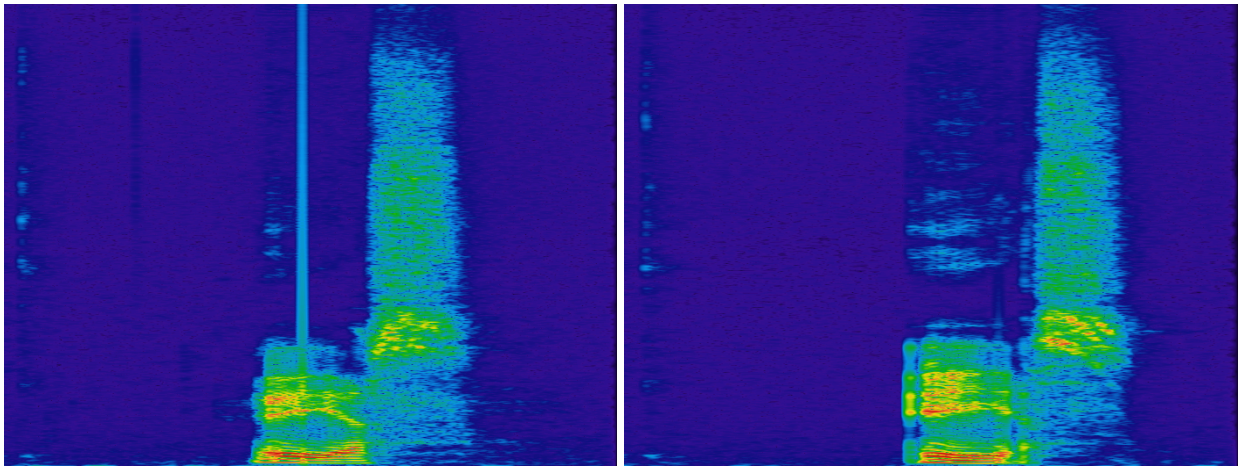

Рис 3.3 Спектрограми слова «*Yes*»

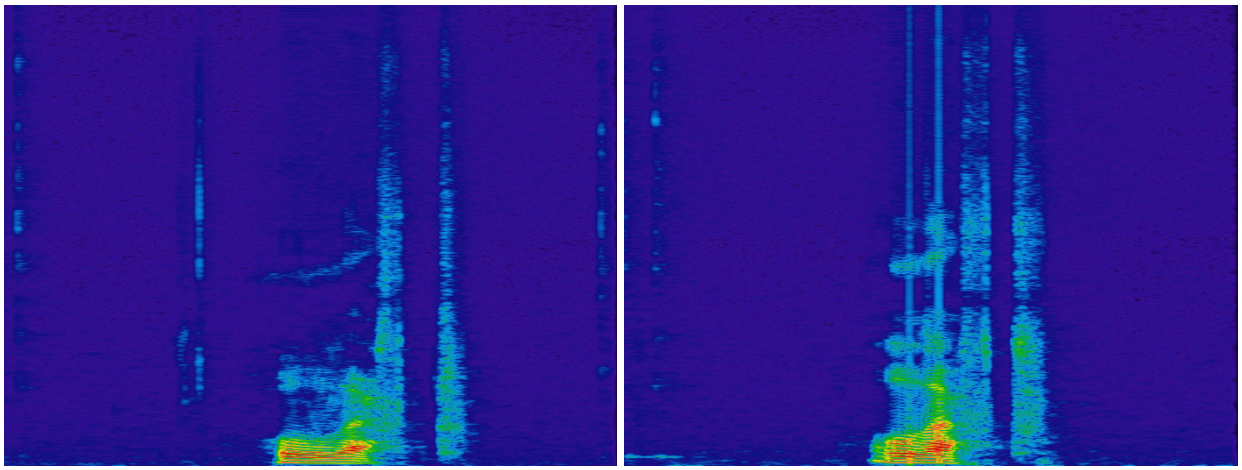

Рис 3.4 Спектрограми слова «*Left*»

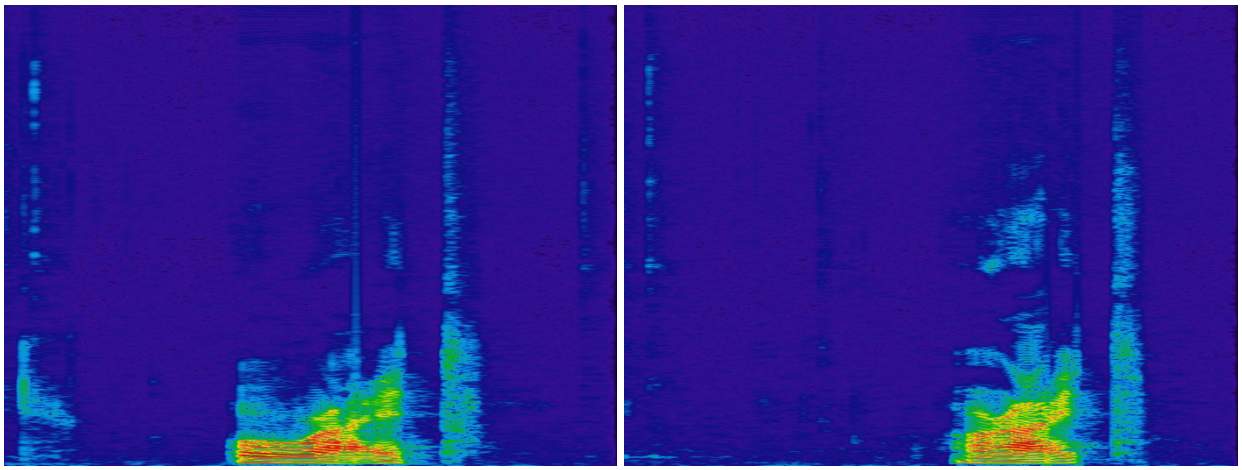

Рис 3.5 Спектрограми слова «*Right*»

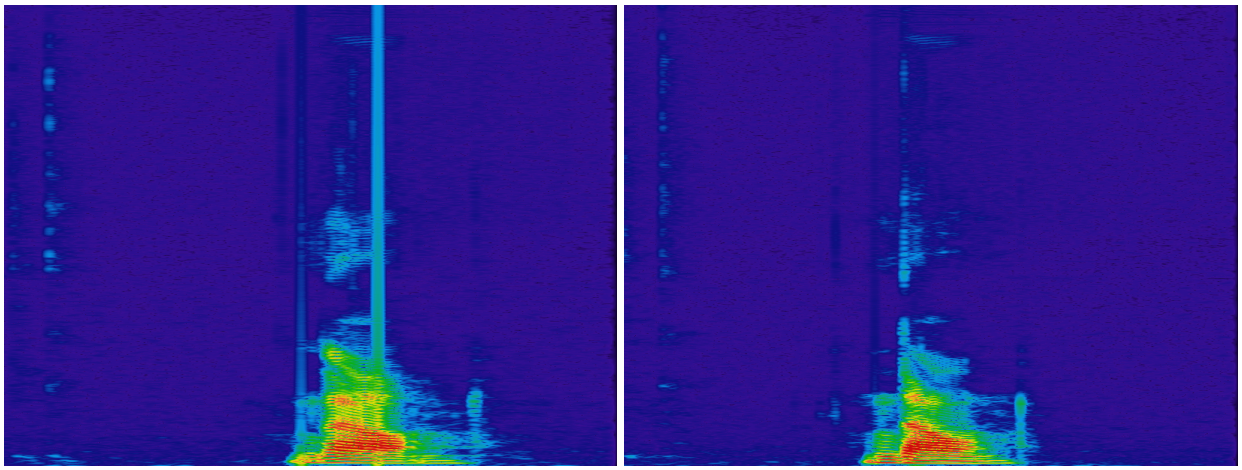

Рис. 3.6 Спектрограми слова «*Down*»

До всіх зображених спектрограм прикріплено відповідні *.wav* файли, на основі яких власне і було побудовано кожну із них. Також для встановлення відповідності між ними та за бажання проведення кращого аналізу, відповідні файли та спектрограми було пронумеровано.

3.2. Вибір мови програмування, середовища розробки та його налаштування

Оскільки найпопулярнішою та відповідно найпоширенішою мовою для роботи з *TensorFlow* являється *Python*, в користь останньої було зроблено вибір, адже завдяки цьому більша кількість зацікавлених людей зможе розуміти як працює застосунок, та за потреби вносити власні правки та корективи. Як наслідок це позитивно відобразиться на можливості застосунку до модифікацій та інтеграцій. Варто відмітити що сама *TensorFlow* може використовувати також з *C++* і *JavaScript*. Для роботи з *TensorFlow* 2.0 потрібна версія *Python* від 3.7 до 3.12.1, та Менеджер пакетів *pip* версії 19.0 або новіше.

В якості середовища розробки було обрано *Google Colab*. *Colab* або «*Colaboratory*» дозволяє писати та виконувати *Python* прямо у вашому браузері мінімізуючи вимоги до вашого апаратного забезпечення, та надаючи безкоштовний доступ до новітніх *GPU* від компанії «*NVidia*».

Сам *Colab* можна назвати доволі цікавим інструментом, який нагадує текстовий редактор. Інтерфейс складається із клітинок, які можуть містити в собі як код, чи частини коду програми, так і пояснювальний текст, в який навіть можна імпортувати математичні формули за допомогою *LaTeX*, а завдяки підтримці *MathJax* формули будуть виглядати зручними для читання та розуміння.

Colaboratory створено на базі *Jupyter Notebook*, тому в *Colab* можна використовувати звичні для нього скорочені анотації, які змінюють спосіб виконання тексту в клітині. Також *Colab* забезпечує автоматичне завершення для вивчення атрибутів об’єктів *Python*, а також швидкий перегляд рядків документації. Введення

відкритої дужки після будь-якого класу чи функції в модулі відобразить спливаюче вікно рядка документації (Рис 3.7).

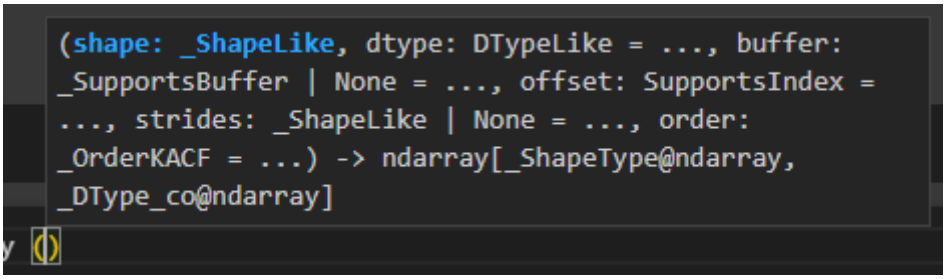

Рис. 3.7 спливаюче вікно рядка документації

Документацію завжди можна відкрити знову за допомогою комбінації *Ctrl+Shift+Space*, або помістивши курсор миші на назву методу.

Google Colaboratory надає неабиякі переваги для роботи з *TensorFlow*, і однією з найцінніших можливостей є зручна візуалізація результатів навчання за допомогою цифрових та текстових діаграм. Це відкриває широкі можливості для зрозумілого представлення та демонстрації процесу навчання моделей, зробивши його більш доступним і наглядним.

У контексті *TensorFlow*, де складні обчислення та величезний обсяг даних можуть бути важко охопити сухим текстом, використання діаграм стає неоціненно важливим. Це дозволяє дослідникам, розробникам та інженерам не лише аналізувати результати, а й ефективно обмінюватись їхніми знаннями та навичками.

Щодо інструментів візуалізації, *Google Colaboratory* дає можливість використовувати різноманітні бібліотеки, такі як *Seaborn, Matplotlib, Altair, Plotly* та інші. Вибір конкретної бібліотеки може залежати від особистих вподобань автора, його потреб у конкретних графічних ефектах чи від задач, які ставляться перед проектом.

Велика перевага полягає в тому, що ця можливість візуалізації не лише полегшує розуміння процесу навчання для самого автора, але й робить його результати більш доступними для співробітників та інших зацікавлених сторін. Такий підхід покращує комунікацію в команді та полегшує обмін інформацією, що є ключовим для ефективної роботи у сучасному науковому та технічному середовищі.

Також слід врахувати що оскільки *Colab* являється розробкою *Google*, то є можливість інтеграції даних з *Google Drive*, що дозволяє ділитися, коментувати та співпрацювати над одним документом з кількома людьми. А сам функціонал дозволяє створювати контрольні версії, зберігати безпосередньо як їх так і їх копії, так ще й коментувати клітини так само, як і будь який інший *Google* документ. Коментарі відображаються лише поруч з клітинами до яких вони прикріплені. Автори роботи можуть відповідати на коментарі, і їх відповіді буде надіслано безпосередньо як і в *Colab*, так і авторам коментарів на пошту.

Для тих хто більше звик працювати з *GitHub* передбачена і інтеграція з цим сервісом. Навіть без авторизації блокноти з *GitHub* можуть бути відкриті в *Colaboratory,* для ведення подальшої роботи над ними. *Colab* також підтримує спеціальні *URL*-адреси, які посилаються безпосередньо на браузер *GitHub* для будь-якого користувача/організації чи сховища. Для тих хто хоче працювати в *Colab* з приватними блокнотами, також передбачена можливість підтягувати інформацію з приватного сховища *GitHub*.

Готові або відредаговані проекти можна зберігати як і в сховищі *Google Drive*, так і в *GitHub*.

Для розробки програмного модуля в *Colab* буде використано мову *Python*. Ця мова славиться своєю простотою та читабельністю коду, що в свою чергу полегшує розробку та робить її зрозумілою для широкого кола користувачів. *Python* є універсальною та високопродуктивною мовою, широко використовуваною в різних сферах розпочинаючи з веб-розробки, а закінчуючи машинним навчанням та повноцінними науковими дослідженнями. Велика та активна спільнота розробників сприяє обміну досвідом та створенню різноманітних бібліотек та інструментів. *Python* має екосистему, що включає *NumPy*, *Pandas*, *TensorFlow*, *Django* та інші, що робить його потужним інструментом для вирішення різних завдань.

Python є платформенно-незалежною мовою, що дозволяє виконувати програми на різних операційних системах. Динамічна типізація спрощує роботу з типами даних, а автоматичне управління пам'яттю полегшує розробку, зменшуючи необхідність ручного видалення об'єктів.

Для візуалізації в ході роботи використано бібліотеку *Seaborn*, яка представляє собою повноцінний інструмент для спрощення візуалізації даних в середовищі мови програмування *Python*. Вона визначається своєю зручністю використання та стилістичними можливостями, що дозволяють створювати естетичні та інформативні графіки. Бібліотека вирізняється високорівневим інтерфейсом для створення різноманітних видів графіків, таких як гістограми, діаграми розсіювання, ящики з вусами та інші, що дозволяє швидко вивчати та розуміти дані. Ключовою перевагою *Seaborn* є її можливість автоматично застосовувати стилі та кольорові палітри, що дозволяє створювати привабливі графіки з мінімальними зусиллями. Крім того, бібліотека дозволяє працювати з великими наборами даних та використовувати різні функції агрегації для підготовки даних до візуалізації. Також *Seaborn* інтегрується з бібліотекою *Matplotlib,* яка теж являє собою бібліотеку для створення статичних, анімованих та взаємодіючих візуалізацій в середовищі мови програмування *Python*.

Для навчання моделі використано набір даних *Speech Commands*, в якому 35 різних англійських слів та команд вимовлено 3000 раз з різними голосами, тембрами та акцентами. Також записи проведено в різних умовах, для покращення можливостей мереж для розпізнавання голосу в різних середовищах. Цей набір даних для вивчення та розробки моделей розпізнавання голосу знаходиться у відкритому доступі, і є доволі популярним серед дослідників та розробників, які працюють у сфері машинного та глибокого навчання.

Варто відмітити що за потреби користувача можливо завантажити не лише цю, але й будь-яку іншу доступну бібліотеку, або самотужки додати нові слова на будь-якій мові, чи взагалі створити власну бібліотеку. Також варто відмітити що в мережі інтернет вже є кілька бібліотек із українськими словами, але все ж вони потребують розширення, для підвищення точності розпізнавання.

3.3. Формування системних та апаратних вимог застосунку

Після визначення середи розробки, мови програмування можна встановити мінімальні вимоги для системи на якій програмний модуль зможе працювати автономно та коректно. Для демонстрації роботи достатньо лише звичайного персонального комп’ютера чи мобільного пристрою з доступом до мережі інтернет та сервісу *Google Colab*. Проте, якщо ми говоримо про можливість автономного запуску та роботи на пристрої, то необхідно розглянути вимоги для компонентів без яких програма просто не буде працювати.

В першу чергу, якщо ми говоримо про безпосереднє розпізнавання в режимі реального часу, то виникає необхідність у мікрофоні підключеному безпосередньо до пристрою із встановленим програмним модулем за рахунок цифрового інтерфейсу *USB*. Основним критерієм тут можна визначити те, що мікрофон повинен бути направленим, а отже не буде уловлювати звуки оточення, які зіпсують точність чи навіть швидкість розпізнавання. В такому випадку під вимоги підійдуть мікрофони кардіоїдної, суперкардіоїдної або гіперкардіоідної направленості, на рис. 3.8 вони зображені зеленим, блакитним та рожевим кольором відповідно.

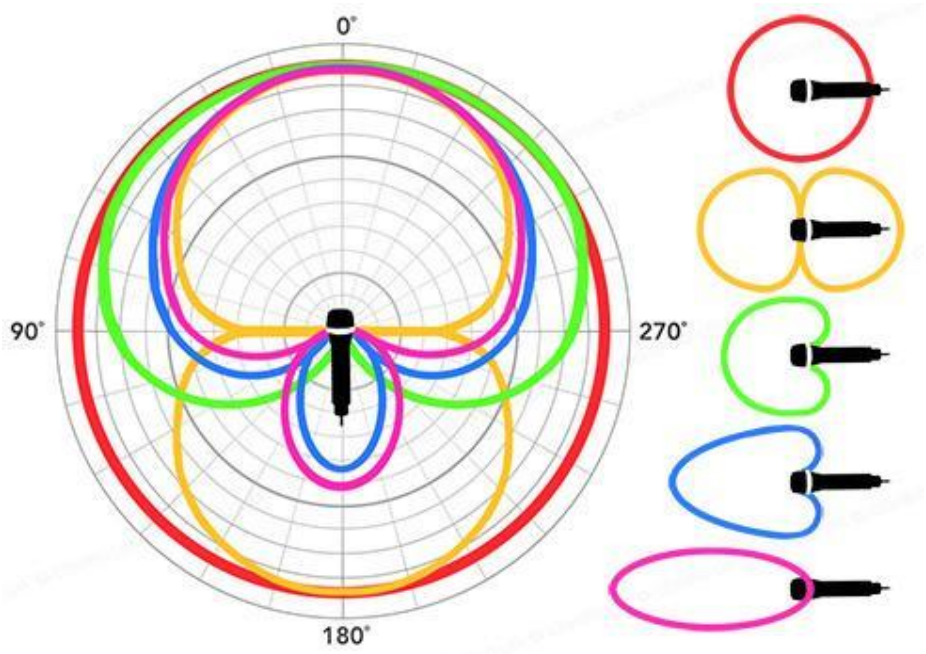

Рис. 3.8 Діаграма направленості мікрофонів (Червоний- кругова, помаранчевий- двонаправлений, зелений- кардіоїдна, синій- суперкардіоїдна, рожевий- гіперкардіоїдна)

Варто відмітити також що важливим параметром у мікрофонів є рівень чутливості, але знайшовши розрахунки чутливості слуху людини при синусоїдальній хвилі частотою 1*kHz*, слух здорового молодого чоловіка буде відповідати чутливості мікрофона в 94 або 74 децибели, в першому випадку тиск вимірюється в паскалях, а в іншому в динах. Але якщо брати в розрахунок можливість запису тихого голосу, то підійдуть і конденсаторні мікрофони з чутливістю –60 — –35 децибел, а електричні конденсаторні із чутливістю –45 — –35 децибел. До того ж конденсаторні мікрофони мають набагато нижчий опір, близько 50 — 200 Ом, що дозволить користуватись ними не використовуючи при цьому фантомне живлення для підсилення сигналу. Такі конденсаторні мікрофони зазвичай працюють в діапазоні від 30 до 15 000 Гц.

Для самої роботи *TensorFlow* можна знайти такі системні вимоги для ПК: Операційні системи, підійдуть лише 64-розрядні: *Ubuntu* 16.04, *macOS* 10.12.6 *Sierra*, *Windows* 7 або новіші версії цих систем.

Python 3.7 або новішої версії

pip 19.0 або новішої версії

TensorFlow 2

Вимоги до апаратної частини обладнання:

Центральний процесор з підтримкою *AVX* інструкцій. (*Intel Haswell* або *AMD Zen* та всі новіші покоління)

Для підтримки *GPU* потрібна відеокарта з підтримкою *CUDA API* (Лише *Ubuntu* та *Windows*).

Окремо варто відмітити що для кращої швидкодії варто використовувати *GPU* обчислення, тому *macOS* покаже себе не найкращим чином, але при наявності можливості модифікацій там все ж можна запустити відеокарти з підтримкою *CUDA*. Сама бібліотека *CUDA* і однойменна технологія також грає ключову роль у використанні відеокарт для обчислень загалом, включаючи роботу з тензорами. CUDA забезпечує високопродуктивні обчислення та дозволяє використовувати відому програмну платформу для паралельного програмування.

Оскільки найбільший стрибок розвитку тензорних обчислень прийшовся на 10 покоління відеокарт *NVidia* побудованих на архітектурі *Pascal*, в яких вперше були

вперше впроваджені тензорні ядра, які спеціально оптимізовані для виконання операцій над тензорами, що використовуються в глибокому навчанні. Це призвело до помітного зростання продуктивності для завдань, пов'язаних з обробкою великих обсягів даних. Тому варто взяти до уваги що використання відеокарт *NVidia* 10го або більш нового покоління позитивно вплине на швидкість роботи.

Тензорні ядра які використовуються в відеокартах *NVidia* використовують паралельність для ефективної обробки даних, що дозволяє прискорити виконання розрахунків в глибоких нейронних мережах. Використання нових відеокарт може значно покращити швидкість завдяки оптимізаціям та підтримці оновлених функцій. Тензорні ядра визначають спеціалізовані операції, які можуть бути виконані на відеокарті для ефективного обчислення тензорних операцій, які часто зустрічаються в глибокому навчанні. Оптимізовані тензорні ядра грають ключову роль у забезпеченні швидкості та продуктивності обчислень.

В новіших поколіннях відеокарт *NVidia* можуть впроваджуватися покращені функції для підтримки тензорних операцій, що також може позитивно позначитися на продуктивності при використанні тензорних операцій у фреймворках машинного навчання, таких як *TensorFlow*.

Архітектура *Pascal* від *NVidia* відзначається високою продуктивністю та ефективністю для паралельних обчислень. Це дозволяє відеокартам ефективно виконувати паралельні обчислення, які є ключовим елементом глибокого навчання.

Таким чином, використання відеокарт *NVidia* 10-го покоління чи новіших може великою мірою підвищити продуктивність при обчисленнях, пов'язаних із тензорами, особливо в глибокому навчанні. Оптимізовані тензорні ядра та висока продуктивність архітектури *Pascal* роблять ці відеокарти важливим інструментом для задач, що вимагають великого обсягу обчислень.

Також окремо потрібно підкреслити що відеокарти компанії *AMD* також здатні працювати з *CUDA*, але для цього потрібно використовувати програмний стек *ROCm*, який працює з відеокартами на архітектурі *GCN* починаючи з 4 покоління. Сам *ROCm* (*Radeon Open Compute*) являє собою платформу відкритого коду для обчислювального зору та глибокого навчання, розроблена компанією AMD

(*Advanced Micro Devices*). *ROCm* надає інструменти та бібліотеки для розробки програмного забезпечення, яке використовує графічні процесори (*GPU*) *AMD* для вирішення завдань обчислювального зору та машинного навчання.

Але при цьому навіть аналогічні по швидкості обчислення *fp16* та *fp32* відеокарти від *NVidia*, будуть швидші від двох до восьми раз (в залежності від покоління).

3.4. Підготовка нейромережі

В ході налаштування мережі використовувалось середовище виконання з застосуванням апаратного прискорення *T4 GPU*. Це дозволяє швидше виконання операцій, пов'язаних з нейронними мережами та іншими завданнями, які можуть бути розпаралелені і використовувати велику кількість обчислювальних ядер, що притаманна графічним процесорам.

Середовище виконання *T4 GPU* забезпечує оптимізовану підтримку обчислень у паралельному режимі, що робить його ефективним для завдань глибокого навчання та інших сценаріїв, де використання *GPU* може значно покращити продуктивність.

T4 GPU (графічний процесор *T4*) - це графічний процесор виробництва *Nvidia*, спеціально розроблений для високопродуктивних обчислювальних завдань та глибокого навчання. Він входить до складу серії *GPU Tesla* і має особливість бути оптимізованим для роботи з обчислювальними задачами, зокрема, задачами, пов'язаними з штучним інтелектом та нейронними мережами.

Графічний процесор *T4* використовується у великої кількості високопродуктивних серверів і систем для прискорення обчислень, зокрема у сферах глибокого навчання, наукових досліджень та обчислювальних завдань, які вимагають великої обчислювальної потужності.

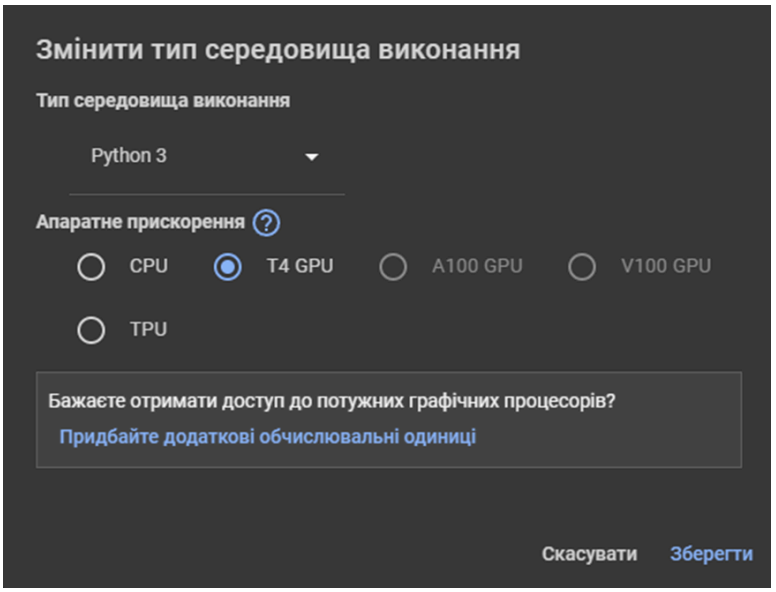

Рис. 3.9 Налаштування середовища виконання в *Google Colab*

Для початку імпортуймо необхідні модулі та залежності

Код:

```
pip install -U -q tensorflow tensorflow_datasets
import os
import pathlib

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import tensorflow as tf

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

# Set the seed value for experiment reproducibility.
seed = 42
```

tf.random.set_seed(seed)
np.random.seed(seed)

Завантаження та розпаковка набору *speech_commands.zip*, що містить набори даних мовних команд, які були зібрані *Google* і опубліковані за ліцензією *CC BY*. Далі за допомогою утиліти *tf.keras.utils.get_file*:

Код:

```
DATASET_PATH = 'data/speech_commands'

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

Аудіозаписи набору даних зберігаються у папках, які відповідають кожній мовленнєвій команді, в нашому випадку ми скористаємося командами: *no, yes, down, go, left, up, right*, та *stop*.

Код:

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

Код визначення масиву:

```
Commands: ['stop' 'up' 'yes' 'down' 'no' 'left' 'right' 'go']
```

Поділені таким чином на каталоги команди можна легко завантажити за допомогою *keras.utils.audio_dataset_from_directory.* Аудіофайли в бібліотеці тривають близько секунди з довжиною 16*kHz*. Для легшого розподілення обріжемо довжину аудіофайлів тривалість яких перевищує одну секунду.

Код:

```
train_ds, val_ds = tf.keras.utils.audio_dataset_from_directory(
    directory=data_dir,
    batch_size=64,
    validation_split=0.2,
    seed=0,
    output_sequence_length=16000,
    subset='both')

label_names = np.array(train_ds.class_names)
print()
print("label names:", label_names)
```

Після виконання отримаємо відповідь

```
Found 8000 files belonging to 8 classes.
Using 6400 files for training.
Using 1600 files for validation.

label names: ['down' 'go' 'left' 'no' 'right' 'stop' 'up' 'yes']
```

Рис. 3.10 Відповідь програми

Тепер визначений набір даних містить групи аудіо-файлів та їх змінних заголовків. Аудіо файли характеризуються так: *(batch, samples, channels)*. Оскільки наш набір даних містить в собі лише файли записані в одноканальному аудіо режимі, використаємо функцію *tf.squeeze*

Код:

```
def squeeze(audio, labels):
  audio = tf.squeeze(audio, axis=-1)
  return audio, labels

train_ds = train_ds.map(squeeze, tf.data.AUTOTUNE)
val_ds = val_ds.map(squeeze, tf.data.AUTOTUNE)
```

Оскільки функція *utils.audio_dataset_from_directory* повертає лише до двох сплітів, то збереження тестових даних окремо від даних для перевірки допоможе краще зрозуміти процес навчання іншим користувачам. Для того щоб зберігати цю інформацію в окремому каталозі можна використовувати *Dataset.shard,* і розділити дані навчання на дві половини. Але в будь-якому випадку кожна ітерація в будь-якому «шарді» завантажить всі дані перевірки, та збереже лише їх фрагмент.

Код:

```
test_ds = val_ds.shard(num_shards=2, index=0)
val_ds = val_ds.shard(num_shards=2, index=1)

for example_audio, example_labels in train_ds.take(1):
  print(example_audio.shape)
  print(example_labels.shape)
```

```
(64, 16000)
(64,)
```

Рис. 3.11 Відповідь програми

За допомогою *Seaborn* побудуємо звукові хвилі, які далі перетворимо на спектрограми.

Код:

```
plt.figure(figsize=(16, 10))
rows = 3
cols = 3
n = rows * cols
for i in range(n):
  plt.subplot(rows, cols, i+1)
  audio_signal = example_audio[i]
  plt.plot(audio_signal)
  plt.title(label_names[example_labels[i]])
  plt.yticks(np.arange(-1.2, 1.2, 0.2))
  plt.ylim([-1.1, 1.1])
```

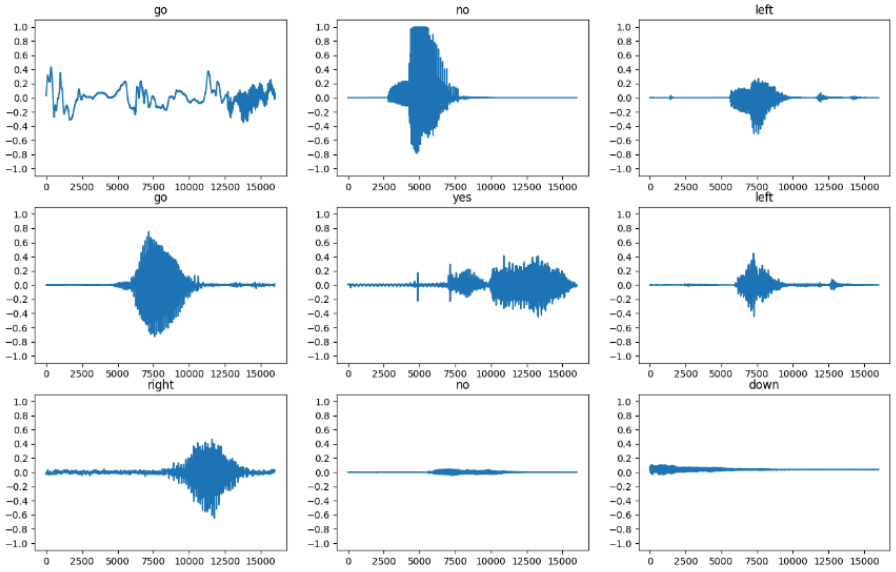

Рис. 3.12 Звукові хвилі візуалізовані за допомогою *Seaborn*

Отримані осцилограми представлені в часово-частотній області. Обрахувавши короткочасне перетворення Фур'є, перетворимо звукові хвилі на спектрограми за допомогою *Seaborn* , щоб представити зміни частоти з часом та підкреслити амплітуду на певній частоті в конкретний момент часу. Завдяки цифровій вибірці даних в часовій області сигнал розбивається на частини, які зазвичай перекриваються, а потім за допомогою перетворення Фур'є, розраховується

величина частотного спектру для кожної частини проміжку. Створені спектрограми передаються в нейронну мережу для навчання моделі.

Перетворення Фур'є - це математичний метод, який дозволяє розкладати сигнал на його складові частоти. У контексті *TensorFlow*, *tf.signal.fft* застосовує цей метод до вхідного сигналу. Однак варто відзначити, що при використанні цього методу вся інформація про час втрачається. Іншими словами, результат цього перетворення показує, які частоти присутні в сигналі, але не вказує, коли ці частоти виникають.

Для вирішення обмежень перетворення Фур'є, застосовується метод короткочасного перетворення Фур'є (*STFT*). Цей метод розбиває вхідний сигнал на невеликі вікна часу і виконує перетворення Фур'є для кожного з цих вікон. Такий підхід зберігає інформацію про час, оскільки вікна стискаються вздовж осі часу. Результатом є *2D*-тензор, в якому одна ось відповідає часу, інша - частотам. Цей тензор може бути використаний для подальших аналізів та обробки, наприклад, для виконання стандартних операцій згортки.

Отже, використання *tf.signal.stft* дозволяє зберігати інформацію про час, що є важливим для багатьох задач обробки сигналів, і водночас використовувати перетворення Фур'є для аналізу частотних компонентів сигналу.

Для створення спектрограми потрібно створити допоміжну функцію, яка буде отримувати на вхід сигнали однакової довжини, щоб фінальні спектрограми на основі яких буде проводитись навчання. Для цього достатньо використати функцію *tf.zeros*, яка додає нуль до всіх аудіо-файлів, які по часу займають менше часу ніж одну секунду.

Під час виклику *tf.signal.stft*, вибираємо параметри *frame_length* та *frame_step* так, щоб створене «зображення» спектрограми було майже квадратним.

Функція створює масив комплексних чисел, що представляють собою величину та фазу, і переводить їх в графік.

Код:

```
def get_spectrogram(waveform):
  # Convert the waveform to a spectrogram via a STFT.
```

```
  spectrogram = tf.signal.stft(
      waveform, frame_length=255, frame_step=128)
  # Obtain the magnitude of the STFT.
  spectrogram = tf.abs(spectrogram)
  # Add a `channels` dimension, so that the spectrogram can be used
  # as image-like input data with convolution layers (which expect
  # shape (`batch_size`, `height`, `width`, `channels`).
  spectrogram = spectrogram[..., tf.newaxis]
  return spectrogram
```

Далі розпишемо форми тензорованої хвилі одного файлу та відповідної до нього спектрограми, і відтворимо оригінальний звук.

Код:

```
for i in range(3):
  label = label_names[example_labels[i]]
  waveform = example_audio[i]
  spectrogram = get_spectrogram(waveform)

  print('Label:', label)
  print('Waveform shape:', waveform.shape)
  print('Spectrogram shape:', spectrogram.shape)
  print('Audio playback')
  display.display(display.Audio(waveform, rate=16000))
```

Код визначення функції для відображення спектрограми:

```
def plot_spectrogram(spectrogram, ax):
  if len(spectrogram.shape) > 2:
    assert len(spectrogram.shape) == 3
    spectrogram = np.squeeze(spectrogram, axis=-1)
  # Convert the frequencies to log scale and transpose, so that the time is
```

```
  # represented on the x-axis (columns).
  # Add an epsilon to avoid taking a log of zero.
  log_spec = np.log(spectrogram.T + np.finfo(float).eps)
  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)
```

Код форми сигналу прикладу в часі (спектрограми):

```
fig, axes = plt.subplots(2, figsize=(12, 8))
timescale = np.arange(waveform.shape[0])
axes[0].plot(timescale, waveform.numpy())
axes[0].set_title('Waveform')
axes[0].set_xlim([0, 16000])


plot_spectrogram(spectrogram.numpy(), axes[1])
axes[1].set_title('Spectrogram')
plt.suptitle(label.title())
plt.show()
```

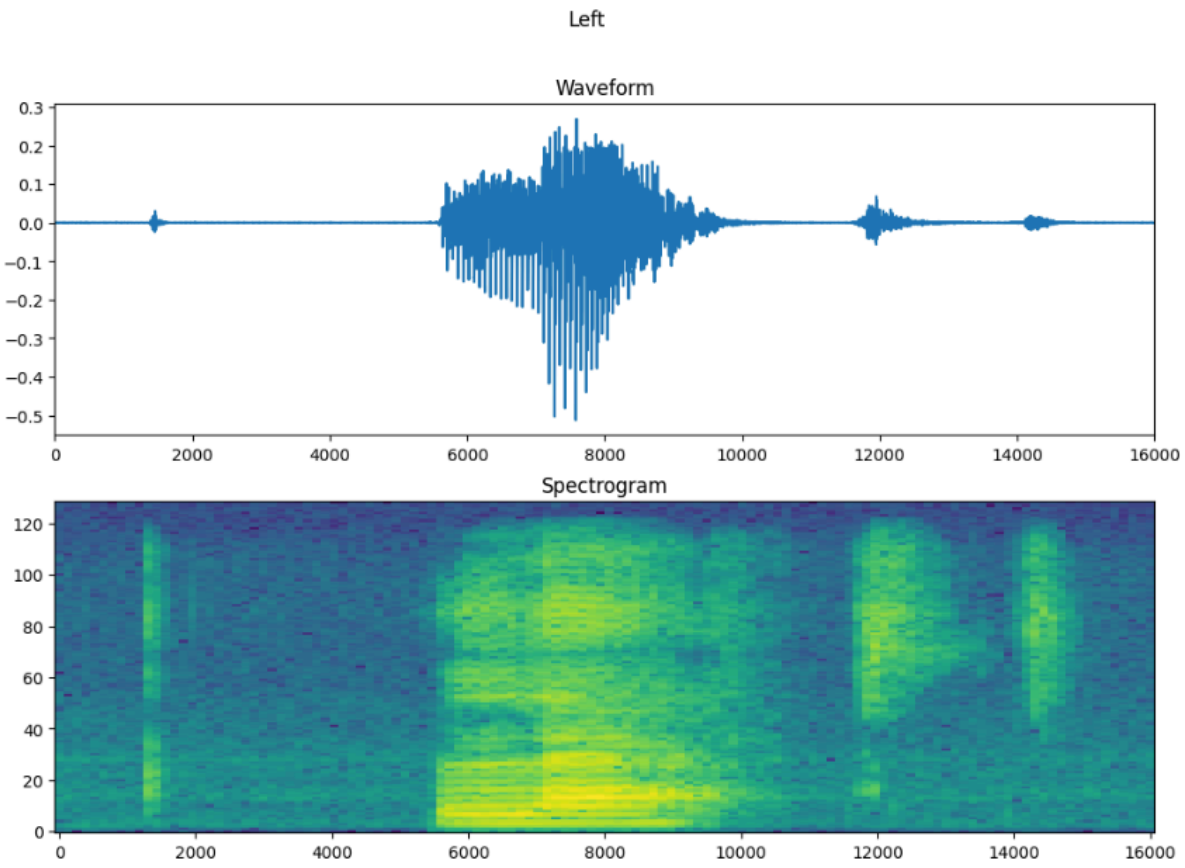

Рис. 3.13 Голосова хвиля перетворена в спектрограму за допомогою перетворення Фур'є

Далі створюймо набори даних спектрограм із наявних в нашій базі звукових даних

Код:

```
def make_spec_ds(ds):
  return ds.map(
      map_func=lambda audio,label: (get_spectrogram(audio), label),
      num_parallel_calls=tf.data.AUTOTUNE)


train_spectrogram_ds = make_spec_ds(train_ds)
val_spectrogram_ds = make_spec_ds(val_ds)
test_spectrogram_ds = make_spec_ds(test_ds)
```

Код створення спектрограм для 9 прикладів наборів звукових файлів:

```
rows = 3
cols = 3
n = rows*cols
fig, axes = plt.subplots(rows, cols, figsize=(16, 9))


for i in range(n):
    r = i // cols
    c = i % cols
    ax = axes[r][c]
    plot_spectrogram(example_spectrograms[i].numpy(), ax)
    ax.set_title(label_names[example_spect_labels[i].numpy()])


plt.show()
```

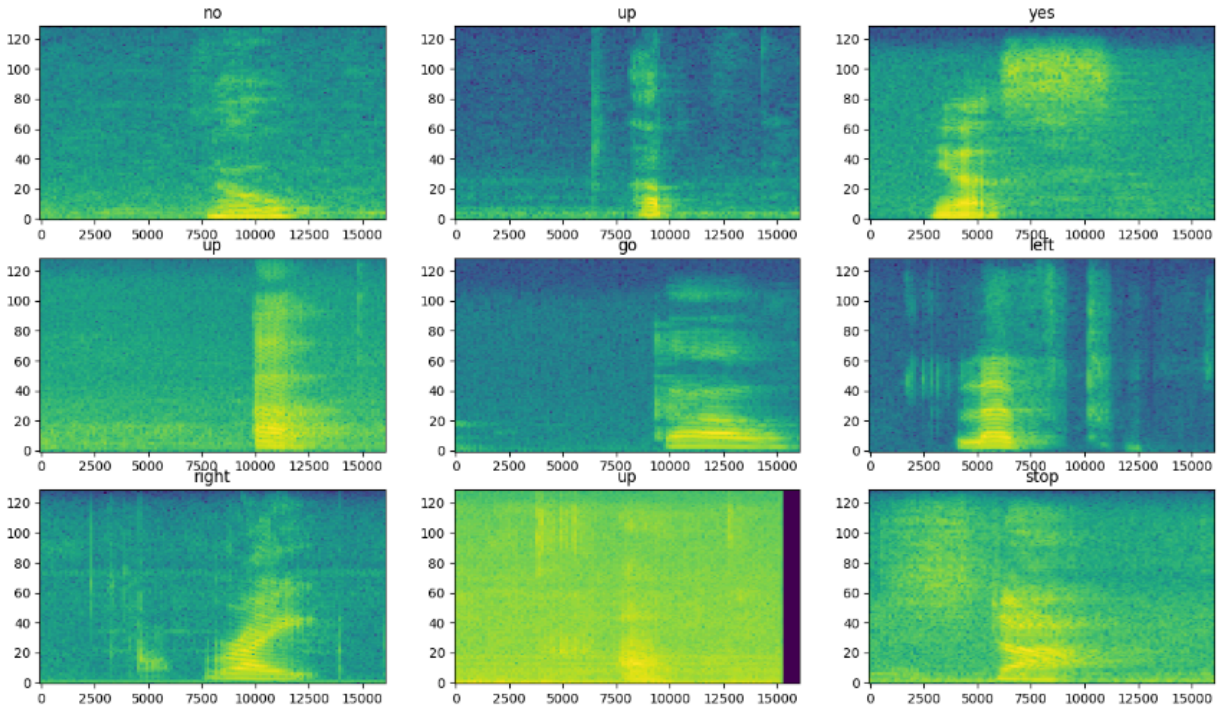

Рис. 3.14 Спектрограми слів які будуть використані в ході навчання

3.5. Побудова та навчання моделі

Для моделі використаєм згорткову нейронну мережу *CNN*, оскільки вже було підготовано аудіофайли та створено зображення спектрограм на їх основі.

Для зменшення затримки читання під час навчання моделі слід використати операції *Dataset.cache* і *Dataset.prefetch.*

Код прикладу використання операцій *Dataset.cache* і *Dataset.prefetch*:

```
train_spectrogram_ds = train_spectrogram_ds.cache().shuffle(10000).prefetch(tf.data.AUTOTUNE)
val_spectrogram_ds = val_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)
test_spectrogram_ds = test_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)
```

Побудована модель буде використовувати такі шари попередньої обробки *Keras* - *tf.keras.layers.Resizing* для зменшення дискретизації вхідних даних, щоб модель могла тренуватися швидше, а *tf.keras.layers.Normalization* нормалізує кожен піксель на зображенні, на основі середнього значення та стандартного відхилення спектрограми.

Для нормалізації слою слід викликати метод *adapt* перед тим як використовувати дані для тренування, щоб розрахувати середньо статистичну похибку в процесі навчання.

Код:

```
input_shape = example_spectrograms.shape[1:]
print('Input shape:', input_shape)
num_labels = len(label_names)


norm_layer = layers.Normalization()
norm_layer.adapt(data=train_spectrogram_ds.map(map_func=lambda spec, label: spec))
```

```
model = models.Sequential([
    layers.Input(shape=input_shape),
    layers.Resizing(32, 32),
    # Нормалізація
    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),
])

model.summary()
```

```
Input shape: (124, 129, 1)
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resizing (Resizing)         (None, 32, 32, 1)         0

 normalization (Normalizati  (None, 32, 32, 1)         3
 on)

 conv2d (Conv2D)             (None, 30, 30, 32)        320

 conv2d_1 (Conv2D)           (None, 28, 28, 64)        18496
```

Рис. 3.15 Відповідь програми

Налаштуймо модель *Keras* за допомогою оптимізатора *Adam* та кросс-ентропійних втрат, щоб запобігти випадкам, коли модель буде впевнена в тому

що неправильний варіант буде правильним вибором. Ця функція дуже ефективно працює для моделей глибокого навчання, які працюють з завданнями класифікації, тому вона чудово підійде для конкретно цієї задачі.

Код:

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

Код для демонстрації роботи кросс-ентропійної функції на прикладі тренування на протязі 10 епох за допомогою кривих втрат побудованих за допомогою *Seaborn*:

```
EPOCHS = 10
history = model.fit(
    train_spectrogram_ds,
    validation_data=val_spectrogram_ds,
    epochs=EPOCHS,
    callbacks=tf.keras.callbacks.EarlyStopping(verbose=1, patience=2),
)

metrics = history.history
plt.figure(figsize=(16,6))
plt.subplot(1,2,1)
plt.plot(history.epoch, metrics['loss'], metrics['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.ylim([0, max(plt.ylim())])
plt.xlabel('Epoch')
plt.ylabel('Loss [CrossEntropy]')
```

```
plt.subplot(1,2,2)
plt.plot(history.epoch,100*np.array(metrics['accuracy']),
100*np.array(metrics['val_accuracy']))
plt.legend(['accuracy', 'val_accuracy'])
plt.ylim([0, 100])
plt.xlabel('Epoch')
plt.ylabel('Accuracy [%]')
```

За допомогою бібліотеки *Seaborn* візуалізуємо схему точності розпізнавання з ходом епох навчання в нашій мережі (зображено на рис. 3.16). По осі *X* відображено кількість епох (повторень навчального набору), а по осі *Y* – відсоток точності розпізнавання. Схема проявляє динаміку змін точності від початкового етапу навчання до завершення всього процесу.

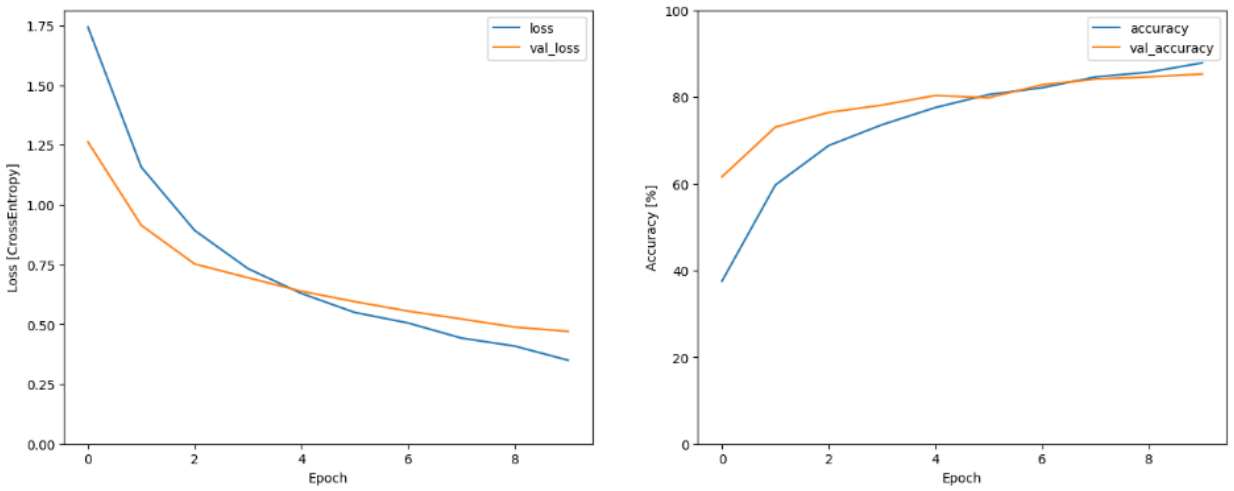

Рис. 3.16 Візуалізація схеми, яка відображає зміну точності з ходом епох навчання

На даній схемі можна виділити такі етапи навчання:

1. Початковий етап (*Epoch* 1-5): На початку навчання точність розпізнавання ще відносно низька, оскільки модель ще не вивчила складнощі завдання. Точність буде зростати поступово з кожною епохою, оскільки мережа адаптується до особливостей даних;

2. Середні епохи (*Epoch* 5-8): У цьому етапі точність розпізнавання стабілізується і продовжить зростати. Мережа покращує свої параметри, уточнює ваги та адаптується до нюансів даних;
3. Етап навчання (*Epoch* 8+): На цьому етапі, якщо навчання продовжується, точність може досягти свого піку. Мережа вивчає найбільше з інформації та стає більш узгодженою;

Графік такого роду відображає, як точність змінюється в залежності від етапу навчання, допомагаючи аналізувати ефективність та стабільність навчання нейронної мережі.

3.6. Оцінка точності створеної моделі

Для оцінки продуктивності моделі, запустимо модель а на вхід подамо підготовані тестові дані.

Для візуалізації результату використаємо *Seaborn* для відображення матриці помилок, для того, щоб зобразити як модель класифікувала кожну із поданих команд з тестових даних.

Код:

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

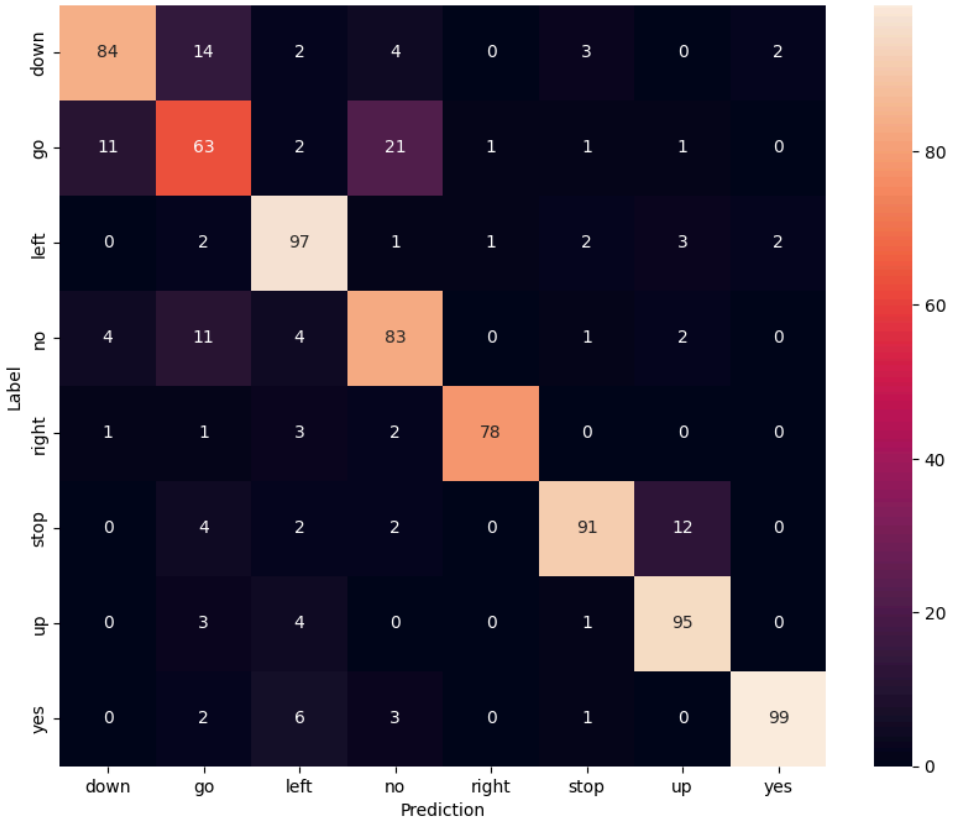

Рис. 3.16 Матриця помилок отримана в ході тестування моделі. На вертикальній осі зображено подані на вхід слова, а на горизонтальній варіант передбачений нейромережею.

Якщо уважно розглянути матрицю, можна помітити уже згадувану в пункті 3.1 схожість в звучанні та спектрограмах для слів «*go*» та «*no*». В даному випадку для створених тест кейсів модель оцінила ймовірність того що невірні відповіді підходять в 21% та 11%.

3.7. Висновки

В ході роботи створено тестовий кейс на основі попередньо записаних слів, за допомогою яких відбудеться тестування працездатності моделі після її тренування. Для полегшення візуального аналізу та порівняння результатів, були створені власноруч спектрограми. Спектрограми - це візуальне представлення звукового сигналу у вигляді тривимірного графіку, де осі відповідають часу, частоті та

інтенсивності звуку. Це корисний інструмент для аналізу аудіосигналів та їхнього використання в обробці мовлення.

Далі було проведено розбір створених спектрограм та аналізовано методи та способи їх створення. Аналіз спектрограм дозволяє отримати уявлення про те, як модель сприймає та обробляє звукову інформацію на різних етапах навчання. Цей підхід допомагає не лише перевірити ефективність моделі, а й отримати інформацію щодо її роботи на різних етапах тренування. Розглядання власноруч створених спектрограм дозволяє визначити, наскільки точно модель відтворює аудіосигнал та визначає звукові характеристики вхідних даних. Також було візуалізовано зміну точності в ході 10 епох навчання моделі.

В ході роботи було використано короткочасне перетворення Фур'є для створення спектрограм є отримання візуального представлення часового аудіосигналу в частотному просторі. Цей процес є ключовим у розумінні частотного складу сигналу та дозволяє виявляти зміни у спектральній характеристиці залежно від часу. за допомогою методу Фур'є можливе більш детальне вивчення вивчення часових та частотних характеристик аудіосигналів, для подальшого застосування у сферах обробки сигналів і штучного інтелекту.

Для створення прототипу програмного модуля вибрано середовище *Google Colab*, основною мовою програмування- *Python*, візуалізацію роботи моделі проведено за допомогою бібліотеки візуалізації *Seaborn*, аргументовано причини вибору. На основі зробленого вибору, відповідно до мінімальних вимог та потреб для роботи в середовищі і з бібліотеками, визначено системні та апаратні вимоги програмного модуля для автономного запуску. Також визначено та встановлено вимоги для мікрофона, який підійшов би для проведення запису голосових команд і повідомлень для їх розпізнавання в реальному часі.

Визначено що використання відеокарт *NVidia* 10-го покоління чи новіших може великою мірою підвищити продуктивність при обчисленнях, пов'язаних із тензорами, особливо в глибокому навчанні. Оптимізовані тензорні ядра та висока продуктивність архітектури *Pascal* роблять ці відеокарти важливим інструментом для задач, що вимагають великого обсягу обчислень. Також варто виділити що ці

відеокарти легко інтегруються в популярні фреймворки глибокого навчання, такі як *TensorFlow* та *PyTorch*, дозволяючи розробникам зручно використовувати їхню продуктивність у своїх проектах.

Проведено підготовку нейромережі, завантажено бібліотеку з файлами для тренування модуля. Реалізовано переведення звукових хвиль в спектрограму за допомогою перетворення Фур'є. Далі на основі спектрограми підготовано згорткову мережу, яка буде працювати з графічними даними та надавати відповіді. Проведено навчання моделі, схематично відображено зміну результатів точності моделі після періоду навчання в 10 епох.

В результаті проведено тестування мережі підготованими файлами. Також проведено аналіз точності моделі, та створено матрицю неточності, яка відображає відсоткову схожість, яку нейромережа встановлює після порівняння спектрограми отриманої нею із тестового аудіофайлу із інформацією отриманою з даних бібліотеки, яку було використано при навчанні.

В результаті навчання моделі було встановлено, що в випадках високої подібності у звучанні команд точність з якою модель віднесла тест файл до відповідної команди відповідає 63%, що є доволі непоганим результатом для такого доволі короткого періоду навчання. У інших випадках середня точність моделі дорівнює 90% що є доволі хорошим результатом, який можна покращити і далі використавши більшу кількість навчальних даних та часу.

ВИСНОВКИ

Дослідження та використання систем для розпізнавання голосу є актуальним та перспективним напрямком в сучасних технологіях. Ці системи відкривають нові можливості для зручної та ефективної взаємодії між людьми та комп'ютерами, а також мають широкий спектр застосувань у багатьох галузях життя та промисловості. А отже, даний програмний модуль може знайти своє застосування як в сфері інтернету речей, так і в корпоративних та промислових цілях. Використання системи із розпізнавання голосових команд може спростити життя, як в побуті так і в робочому житті. Особливістю розробки є її можливість до цілком автономної роботи, і незалежності від наявності мережі інтернет, чи/або доступності хмарних сервісів, які надають послуги з розпізнавання голосової інформації. А оскільки не використовуються сторонні сервіси, то варто виділити і підвищену конфіденційність і безпеку використання такого модуля.

Завдяки особливостям *TensorFlow*, яка надає не лише надає потужні інструменти для розробки та впровадження моделей машинного та глибокого навчання, але й відзначається своєю високою гнучкістю у плані розгортання. Однією з ключових переваг є можливість легкої інтеграції додатків на різних пристроях. Це означає, що одну й ту ж модель можна використовувати як на мобільних пристроях, так і на робочих станціях чи серверах, забезпечуючи єдність та синхронізацію функціоналу на різних платформах.

Окрім цього, *TensorFlow* активно підтримує хмарні розробки та обчислення. Це дозволяє розробникам використовувати власні додатки в хмарному середовищі, забезпечуючи високу масштабованість та потужність обчислень. Враховуючи те, що хмарні сервіси можуть надавати значні обчислювальні ресурси, використання такого підходу дозволяє впроваджувати та обслуговувати моделі, які можуть вимагати значних обчислювальних потужностей. Можливість хмарного використання власних рішень забезпечує ефективність додатків, які мають високі вимоги до обчислень чи великі обсяги даних, і водночас зберігає прозорість і доступність для

кінцевих користувачів навіть у випадках, коли їхні пристрої мають обмежені ресурси. Такий підхід робить *TensorFlow* універсальним і ефективним інструментом для широкого кола розробок та застосувань.

Зазвичай для розпізнавання голосу використовуються глибокі нейронні мережі, зокрема рекурентні нейронні мережі (*RNN*), згорткові нейронні мережі (*CNN*) і моделі *Transformer*. А наявність інструментів та сервісів для розпізнавання голосу в режимі хмари, дозволяє розробникам легко інтегрувати ці функції у свої додатки та системи. В ході роботи було використано згорткові нейронні мережі, та їх переваги для роботи з візуальними даними. В результаті модуль здатен створювати спектрограми з вхідних аудіофайлів, і шляхом їх порівняння із отриманих в процесі навчання даних, з достатньо високою точністю здатен проаналізувати вхідну інформацію, та розпізнати команду подану на вхід.

Основний об'єкт дослідження бібліотека *TensorFlow* є потужним та універсальним інструментом для розробки та впровадження моделей машинного навчання та глибокого навчання. Ця модель вирізняється такими перевагами, як дуже висока точність розпізнавання, здатність до навчання на великих обсягах даних та інформації, зокрема і в реальному часі, підтримкою різних архітектур та можливістю інтеграції інших інструментів *TensorFlow*. Все це можливе завдяки використанню потужних алгоритмів та оптимізації у процесі навчання моделей. Загалом, використання моделі *TensorFlow* для розпізнавання голосових повідомлень дозволяє створити потужну та точну систему, від якої вимагається висока якість результатів розпізнавання голосу. *TensorFlow* проявляє вражаючу здатність до навчання на великих обсягах даних та інформації в реальному часі. Це дозволяє створювати моделі, які не лише демонструють високу точність, але й адаптуються до змінних умов та швидко реагують на нові дані. Нарешті, можливість інтеграції з іншими інструментами робить *TensorFlow* привабливим вибором для розробників, які вже використовують певні технології та потребують сумісності з іншими інструментами. Завдяки цьому інтеграція модуля стає тривіальною задачею, яка не складе проблем в разі такої необхідності.

Використання моделі *TensorFlow* для розпізнавання голосових повідомлень є універсальним і відкриває широкі можливості застосування в різних сферах. Починаючи з побутових завдань, таких як розпізнавання голосових команд у домашніх асистентах чи системах управління, і завершуючи промисловим використанням у сферах, де точність і ефективність грають важливу роль.

У побутовому використанні, такі моделі можуть допомагати створювати інтуїтивно зрозумілі та зручні голосові інтерфейси для щоденного використання. Наприклад, вони можуть бути використані для розпізнавання голосових команд у смарт-домашніх системах, автоматизації побутових пристроїв та забезпечення комфорту користувача.

У промисловому використанні, розпізнавання голосу може знаходити застосування в сферах, де важлива автоматизація та оптимізація робочих процесів. Наприклад, в області телекомунікацій або клієнтського обслуговування, системи голосового розпізнавання можуть оптимізувати обробку звернень та забезпечити швидкий та ефективний обмін інформацією.

Таким чином, використання моделі *TensorFlow* для розпізнавання голосу можуть бути важливим інструментом, який може одночасно покращити як якість та зручність взаємодії з технологіями в різних аспектах життя.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: НАУ 2017. – 63 с
2. ДСТУ ГОСТ 7.1:2006 «Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання».
3. ДСТУ 3582: 2013 «Бібліографічний опис скорочення слів і словосполучень в українській мові».
4. ДСТУ 8302-2015 «Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання».
5. Прохоренко Є. І. «Про пакетну передачу мови» *https://repository.kpi.kharkov.ua*
6. *Hope T., Resheff Y., Lieder I. Learning TensorFlow: A Guide to Building Deep Learning Systems. O'Reilly Media,* 2017. 240 *p.*
7. *Hala N. Essential Math for AI: Next-Level Mathematics for Efficient and Successful AI Systems. O'Reilly Media,* 2023. 602 *p.*
8. *McClure N. TensorFlow Machine Learning Cookbook: Over* 60 *recipes to build intelligent machine learning systems with the power of Python, 2nd Edition. Packt Publishing,* 2018. 422 *p.*
9. *Ravichandiran S. Hands-On Deep Learning Algorithms with Python: Master deep learning algorithms with extensive math by implementing them using TensorFlow. Packt Publishing,* 2019. 512 *p.*
10. *Gerrish S. How Smart Machines Think. The MIT Press,* 2018. 358 *p.*
11. *Chollet F. Deep Learning with Python. Manning; First Edition,* 2017. 384 *p.*
12. *Chollet F. Deep Learning with Python. Manning; Second Edition,* 2021. 504 *p.*
13. *Géron A. Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media,* 2017. 572 *p.*
14. *Goodfellow I., Bengio Y., Courville A. Deep Learning (Adaptive Computation and Machine Learning series). The MIT Press,* 2016. 800 *p.*

15. *Introduction to TensorFlow [Electronic resource]*. – 2023. *Access mode: www.tensorflow.org (lastaccess:*21.12.23*). – Title from the screen.*
16. *Lewis C. How to Create & Understand Mel-Spectrograms [Electronic resource]. – 2021. Access mode: importchris.medium.com (lastaccess:*21.12.23*). – Title from the screen.*
17. Козінцев І. Протоколи *RTP* і *RTCP*. [Електронний ресурс]. – 2010. Режим доступу: *wiki.cuspu.edu.ua* – Назва з екрана.
18. *IP Packet Delay Variation Metric for IP Performance Metrics (IPPM) [Electronic resource]. – 2002. Access mode: datatracker.ietf.org (lastaccess:*21.12.23*). – Title from the screen.*
19. CrossEntropyLoss *[Electronic resource]*. – 2023. *Access mode: pytorch.org (lastaccess:*21.12.23*). – Title from the screen.*
20. Overview of Colaboratory Features *[Electronic resource]*. – 2023. *Access mode:colab.research.google.com (lastaccess:*21.12.23*). – Title from the screen.*
21. Charts in Colaboratory *[Electronic resource]*. – 2023. *Access mode:colab.research.google.com (lastaccess:*21.12.23*). – Title from the screen.*
22. Audio Recognition in Tensorflow *[Electronic resource]*. – 2023. *Access mode: geeksforgeeks.org (lastaccess:*21.12.23*). – Title from the screen.*
23. Seaborn: statistical data visualization *[Electronic resource]*. – 2012. *Access mode: seaborn.pydata.org (lastaccess:*21.12.23*). – Title from the screen.*
24. Schartz R. The Shape of Tensor *[Electronic resource]*. – 2019. *Access mode: medium.com (lastaccess:*21.12.23*). – Title from the screen.*
25. *Cuantum Technologies. Natural Language Processing with Python: Building your Own Customer Service ChatBot: Unleash the Power of AI (Masteting AI and Python). Cuantum Technologies,* 2023. 504 *p.*
26. *Cuantum Technologies. Machine Learning with Python: Keras, PyTorch, and TensorFlow: Unlocking the Power of AI and Deep Learning (Masteting AI and Python). Cuantum Technologies,* 2023. 564 *p.*
27. *Liu G. R. Machine Learning with Python: Theory and Applications. World Scientific Publishing Company,* 2022. 692 *p.*

28. *Melehi D. The Rise of Machine Learning- What It Means for the World Kindle Edition.* 2023. 28 *p.*
29. *Ganegedara T. TensorFlow in Action. Manning,* 2022. 680 *p.*
30. *Desai S. Introduction to TensorFlow Using Python.* 2021. 95 *p.*
31. *Lingfei W., Peng C., Jian P., Liang Z. Graph Neural Networks: Foundations, Frontiers, and Applications. Springer,* 2022. 689 *p.*

ДОДАТОК

Додаток А

Матриці помилок

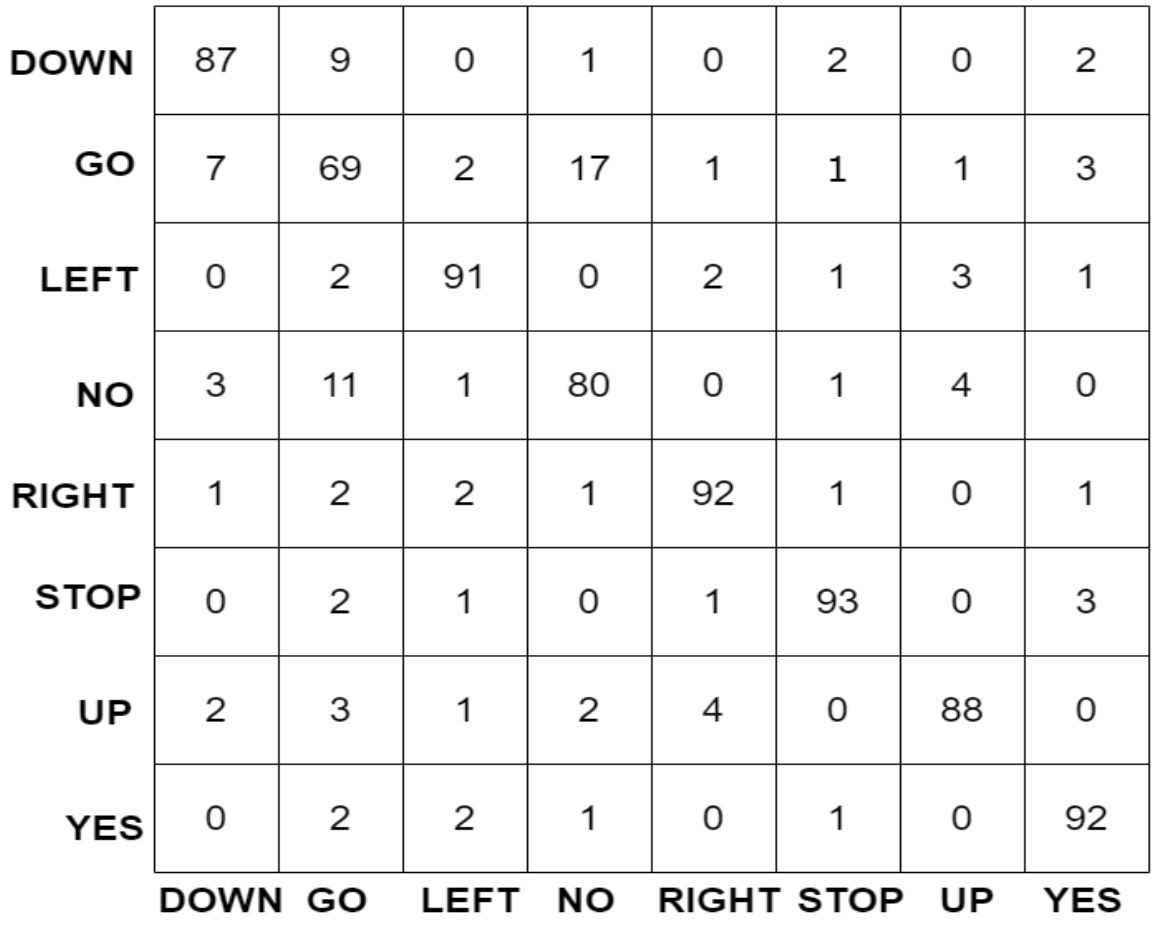

Рис А.1 Матриця помилок складена в результаті тестування моделі першим тест-кейсом

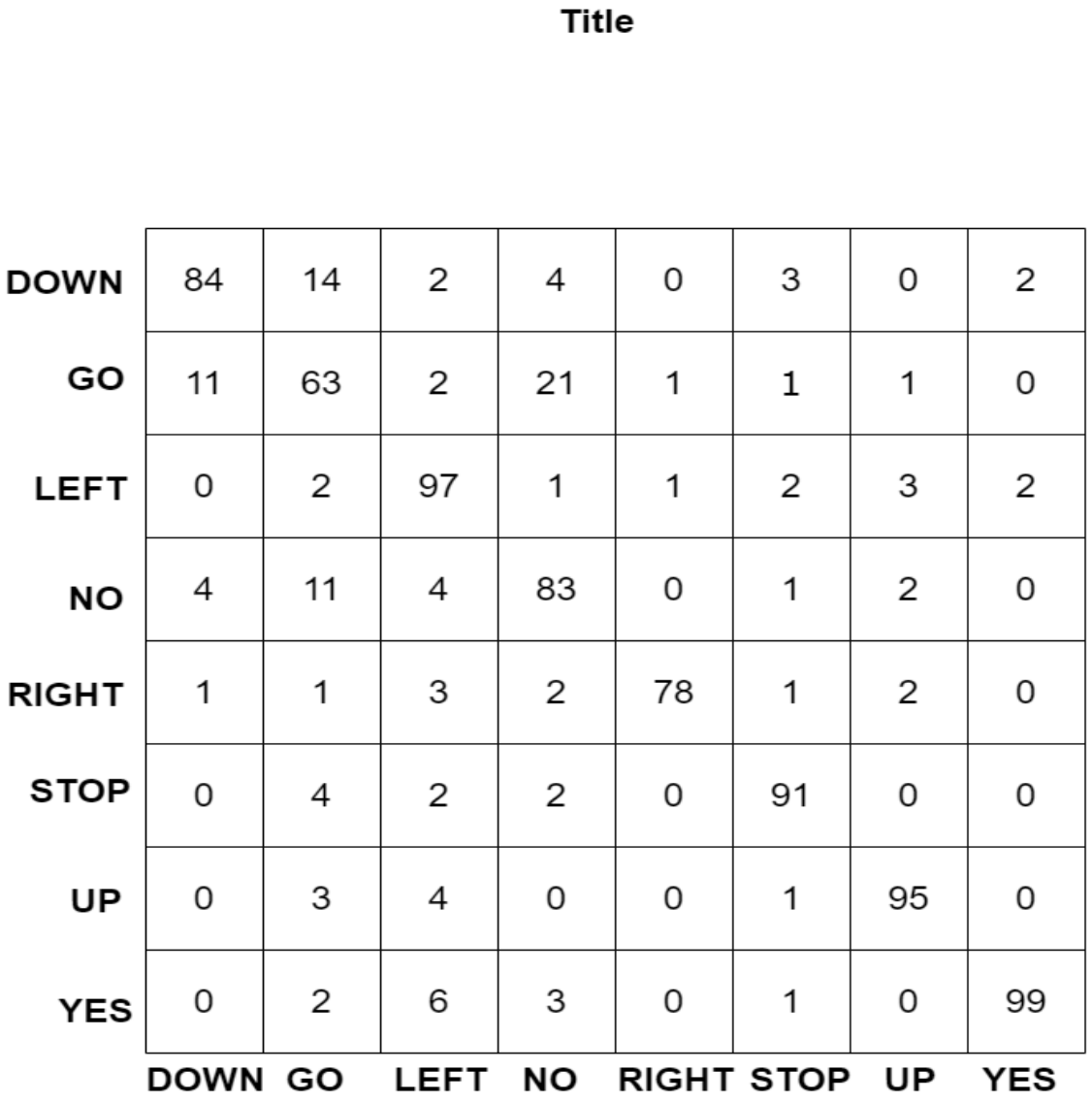

Рис А.2 Матриця помилок складена в результаті тестування моделі другим тест-кейсом