Київський національний університет імені Тараса Шевченка Факультет інформаційних технологій
Кафедра програмних систем і технологій

На правах рукопису

УДК 004.42

ВИПУСКНА КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

“АДАПТИВНА СИСТЕМА КОРЕКЦІЇ ВАД МОВИ В РЕАЛЬНОМУ ЧАСІ”

Спеціальність 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

Студент групи ІПЗм-21
_____ Пелипенко І.Г.
(підпис) (дата)

Науковий керівник
_____ доцент кафедри ПСТ, д.т.н. Порєв Г.В.
(підпис) (дата)

Допускається до захисту з питань нормоконтролю
_____ завідувач кафедри ПСТ д.т.н. О.С.Бичков
(підпис) (дата)

2022

Рішенням Екзаменаційної комісії
випускна кваліфікаційна робота студента
Пелипенка Іллі Григоровича
захищена з оцінкою

_____

Голова Екзаменаційної комісії
професор, доктор техн. наук Онищенко В.В

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ:
Завідувач кафедри програмних систем і технологій
_____(О.С.Бичков)
2022 _____”____“р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ
СТУДЕНТУ
Пелипенку Іллі Григоровичу

1. Тема випускної кваліфікаційної магістерської роботи “Адаптивна система корекції вад мови в реальному часі” та керівник роботи Порєв Геннадій Володимирович, доцент кафедри КПСТ, д.т.н., затверджені наказом вищого навчального закладу від «__»____20__ р. №_____
2. Строк здачі студентом закінченої роботи 04.05.2022
3. Вихідні дані до роботи: підручники, навчальні посібники, статті, Інтернет-ресурси, стандарт ДСТУ 3008:2015 «Інформація та документація, звіти у сфері науки і техніки.
4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)

Аналіз характеристик звуку у цифровому виді. Знаходження правильного датасету для навчання. Аналіз та порівняння найкращих характеристик мови для навчання. Розгляд та порівняння нейронних мереж та методів класифікації. Проектування архітектури нейронної мережі та її навчання. Програмування застосунку для корекції мови та аналіз результатів його роботи.

5. Перелік графічного матеріалу: 16 рисунків, 2 таблиці

6. Консультанти з роботи із зазначенням розділів роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Підготовка датачету	Порєв Г.В.	20.11.2022	28.01.2022
Розробка нейронної мережі	Порєв Г.В.	03.02.2022	09.03.2022
Програмна реалізація	Порєв Г.В.	12.03.2022	26.04.2022

7. Дата видачі завдання 16.10.2021

Керівник Порєв Г.В.

(підпис) (розшифровка підпису)

Завдання прийняв до виконання Пелипенко І.Г.

(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Студент – магістр _____ Пелипенко І.Г.

(підпис) (розшифровка підпису)

Керівник роботи _____ Порєв Г.В.

(підпис) (розшифровка підпису)

Номер і назва етапів роботи	Термін виконання етапів роботи	Примітка
1. Ознайомлення з предметною галуззю	24.10.2021	"виконано"
2. Визначення структури магістерської дисертації; вивчення літератури	12.11.2021	"виконано"
3. Робота над першим розділом; Збір та обробка даних	28.01.2022	"виконано"
4. Робота над другим розділом; Розробка нейронної мережі	09.03.2022	"виконано"
5. Робота над третім розділом; Розробка програмної частини	26.04.2022	"виконано"
6. Оформлення текстової частини магістерської роботи	01.05.2022	"виконано"
7. Оформлення презентаційної частини магістерської роботи	15.05.2022	"виконано"

Анотація

Випускна кваліфікаційна магістерська робота: містить 71 сторінку, 16 рисунків, 2 таблиці, один додаток та 9 джерел використаних в роботі.

Тема: Адаптивна система корекції вад мови в реальному часі.

Об'єкт дослідження: Аудіопотік мови із порушенням плавності (заїканням)

Мета роботи: Корекція аудіопотоку мови із заїканням у реальному часі

Предмет дослідження: Згорткова нейронна мережа у роботі із аудіоданими

Результати дослідження: Розглянули основні поняття та здійснили огляд звуку у цифровому представленні. Порівняли характеристики звуку та їх вплив на машине навчання на кореляцію із заїканням. Проаналізували та порівняли методи які вирішують проблему класифікації звуку. Вибравши найкращий метод класифікації ми дослідили структуру згорткової моделі нейронної мережі, та її модифікацію із рекурентними шарами. Після аналізу та поставленні експериментів над гіперпараметрами, ми досягли точності класифікації у 93 відсотка. Це також потребувало експериментів над пропорцією та розмірністтю даних для навчання. Найкращий результат показав датасет із сегментів по одній секунді та пропорцією із 25% заїкання та 75% мови без заїкання. Модель яку ми розробили, ми порівняли із іншими відомими моделями класифікації звуку, такими як SVM, CNN, HMM та RNN. Наша модель показала найкращі результати. Отримавши модель яка може розпізнавати заїкання у вигляді повторення звуків, ми спроектували на розробили програму, яка використовує цю модель для розпізнавання звуку у реальному часі.

Висновок: Нам вдалося досягнути поставленої мети у розробці адаптивної системи корекції мови у реальному часі. Хоча результати роботи ПЗ можна покращувати багатьма шляхами, розроблену програму вже можна використовувати для зменьшення заїкання у дітей.

ЗАЇКАННЯ, ЗГОРТКОВА НЕЙРОНА МЕРЕЖА, АУДІОПОТІК, КЛАСИФІКАЦІЯ, MFCC, РЕАЛЬНИЙ ЧАС

Annotation

The final qualifying master's thesis: contains 71 pages, 16 figures, 2 tables, one appendix and 9 sources used in the work.

Topic: Adaptive system of correction of speech defects in real time.

Object of research: Audio flow of speech with impaired fluency (stuttering)

Objective: Correction of audio flow of speech with stuttering in real time

Subject of research: Convolutional neural network in working with audio data

Research results: Considered the basic concepts and reviewed the sound in digital representation. The characteristics of sound and their effect on the learning machine on the correlation with stuttering were compared. Analyzed and compared methods that solve the problem of sound classification. Having chosen the best method of classification, we investigated the structure of the convolutional model of the neural network and its modification with recurrent layers. After analyzing and experimenting with hyperparameters, we achieved a classification accuracy of 93 percent. It also required experiments on the proportion and dimension of data for training. The best result was shown by a dataset of segments of one second and a proportion of 25% stuttering and 75% speech without stuttering. We compared the model we developed with other well-known sound classification models, such as SVM, CNN, HMM and RNN. Our model showed the best results. Having obtained a model that can recognize stuttering in the form of repetition of sounds, we designed to develop a program that uses this model to recognize sound in real time. Through experiments, we were able to achieve optimization of the time of work with the sound segment up to 1240 milliseconds, which is an acceptable result for working with sound.

Conclusion: We managed to achieve the goal of developing an adaptive real-time speech correction system. Although the results of the software can be improved in many ways, the developed program can already be used to reduce stuttering by children.

STUTTERING, CNN, AUDIOSTREAM, CLASSIFICATION, MFCC, REAL-TIME

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ

A/D - аналогово-цифрове перетворення

MFCC - Мелчастотні кепстральні коефіцієнти (Mel-frequency cepstral coefficients)

CNN - Згорткова нейронна мережа (convolutional neural network)

GRCNN - Convolutional Neural Networks with Gated Recurrent Connections (Згорткова нейронна мережа з рекурентними зв’язками із закритим доступом)

ШНМ- Штучна нейрона мережа

ШН - Штучний нейрон

ReLU — випрямлений лінійний вузол (Rectified Linear unit)

ЗМІСТ

ВСТУП

Багато людей мають ті чи інші вади мови. Одні вади майже не впливають на життя, інші роблять життя дуже складним. Однією із найпоширеніших вад мови є заїкання. Заїкання - це розлад спілкування, що супроводжується порушеннями плавності у мовленні людини. Крім того, що люди, які заїкаються, часто відчувають фізичну напругу і боротьбу в мовних м’язах, а також збентеження, тривогу і страх перед розмовою. Разом ці симптоми можуть ускладнювати розмову людям, які заїкаються, і це ускладнює їм ефективне спілкування з іншими. Існує стільки різних моделей заїкання, скільки людей, які заїкаються, і багато різних ступенів заїкання, від легкого до сильного. Тяжкість заїкання у людей сильно варіює. Він також може змінюватися в однієї і тієї ж людини з дня на день і залежно від мовної ситуації. Заїкання характеризується повторенням звуків, складів або слів, подовження звуків, і переривання в мовленні, відомі як блоки. Найчастіший випадок заїкання, особливо у дітей - це несвідоме повторення звуків. Підраховано, що близько одного відсотка дорослого населення заїкається. Це становило б майже 400 тисяч, людей які заїкаються лише в Україні. Заїкання зустрічається приблизно в три-чотири рази частіше у чоловіків, ніж у жінок.

Ситуація погіршується коли розмова перетікає у цифровий формат. Людям із заїканням завжди було важко розмовляти по телефону, бо в такому випадку людина може комунікувати лише мовою, співбесідники не бачать друг друга, тому можуть виникати непорозуміння. А у нас час, все більше і більше спілкування переносяться в онлайн формат. Зустрічі у реальному житті потребуються все меньше і меньше. І у людей із проблемами заїкання виникає багато проблем у спілкувані в такому форматі. Так як єдиним засобом спілкування є голос, люди починають триважитися та соромитися свого заїкання, що робить його більш тяжким.

Ідея моєї роботи полягає у розробці системи допомоги людям які страждають від заїкань(повторень звуків) у онлайн форматі в реальному часі. Ця система повинна перехоплювати вхідні звуки із системи вводу звуку,

обробляти сигнал та виявляти, чи є заїкання-повторення у мові. Якщо такі ознаки є, то програма посилає на вихід звук, який би приховував заїкання (наприклад звук роздумів), або просто не посилала звук до співбесідника. А якщо ознак заїкання немає, то звук без перепон проходить від мікрофону до програми виводу.

Отже метою моєї роботи є корекція аудіопотоку мови із заїканням у реальному часі.

Для досягнення мети роботи ми повинні виповнити такі задачі:

- аналіз характеристик звуку та методів роботи з ним
- створення нейронної мережі для оцінки наявності ознак заїкання в мові
- збір датасету для вивчення нейронної мережі
- створення програмного забезпечення яке б аналізувало та модифікувало аудіосигнал в реальному часі

Об’єкт дослідження - аудіопотік мови із порушенням плавності (заїканням).

Предмет дослідження - Згорткова нейронна мережа у роботі із аудіоданими.

Для досягнень цих задач ми використали такі методи: аналіз, порівняння, вимірювання, моделювання, експеримент, узагальнення результатів і формування висновків.

Наукова новизна полягає в удосконаленні моделі згорткової нейронної мережі для Розпізнавання заїкання, та розробці ПЗ яке би використовувало цю модель для аналізу даних в реальному часі.

Практична цінність роботи полягає у можливості використання розробленої моделі нейронної мережі Розпізнавання заїкання, для різних задач, наприклад для аналізу прогрессу лікування, або для оцінки ступенню заїкання лікарями у лікарнях. Також створене ПЗ можна використовувати на будь-якому ПК для покращення якості спілкування в онлайні, та зменьшення психологічного навантаження людини із заїканням.

РОЗДІЛ 1: ПІДГОТОВКА ДАТАСЕТУ

Для побудування нейронної мережі для Розпізнавання заїкання, нам треба спочатку зібрати датасет, за яким буде навчатися наша нейронна мережа. Так як основою нашого датасету є аудіодані, нам треба розібратися, які дані ми будемо використовувати для навчання, провести аналіз та порівняння характеристик звуку та дізнатися які з них найбільше підходять для навчання.

1.1 Концепції цифрового аудіо

Аудіо за своєю суттю є аналоговою ознакою світу природи. Коли об’єкт вібрує, він змушує вібрувати й навколишні його молекули. Ці молекули впливають на прилеглі до них і так далі, поширюючи вібрацію у вигляді хвилі назовні від джерела, поки амплітуда хвилі (її об’єм) не згасне з відстанню. Таким чином, зернистість аудіохвилі в реальному світі - це структура окремої молекули середовища, через яку проходить звукова хвиля.
На Землі середовищем, через яке передається більшість аудіо, є повітря. Деякі звуки подорожують через воду або навіть через камінь, що складається з самої планети, але майже всі звуки, які ви чуєте щодня, подорожують до вух через повітря.

1.1.1 Вибірка аудіо

Отже, звуки, які людина чує щодня, насправді є вібраціями в повітрі, які викликають внутрішню роботу вуха. Чим далі рухаються молекули повітря з кожним імпульсом хвилі, тим вища амплітуда хвилі і гучніший звук. Чим швидше вібрують молекули, тим вище частота хвилі.

Чим вище амплітуда (висота) хвилі, тим гучніший звук у цей момент. Чим коротша довжина хвилі (чим ближче один до одного розташовані гребені хвилі), тим вище частота (або висота) звуку, який виробляється.

Комп’ютери, однак, цифрові. Щоб відобразити звукову хвилю так, як комп’ютери можуть маніпулювати та працювати з ними (не кажучи вже про

передачу через мережу), звук має бути перетворений у цифрову форму. Цей процес називається аналого-цифровим перетворенням (скорочено A/D).

Першим фактором, що впливає на точність записаного звуку, є пропускна здатність звуку, тобто діапазон звукових частот, який АЦП здатний захопити та перетворити в цифрову форму. На пропускну здатність аудіо також впливає кодек, якщо він вирішить відкинути будь-які діапазони частот під час кодування звуку.

Звук надходить у комп'ютер через мікрофон або інший вхід у вигляді потоку електронів, напруга якого змінюється відповідно до амплітуди звукової хвилі. Цей аналоговий сигнал потім перетворюється в цифрову форму за допомогою схеми, яка фіксує амплітуду вхідної хвилі через регулярні проміжки часу, перетворюючи ці дані в число у формі, зрозумілій системі аудіозапису. Кожен з цих знятих моментів є зразком(sample). З'єднавши всі зразки разом, ви можете приблизно представити вихідну хвилю, як показано на схемі нижче.

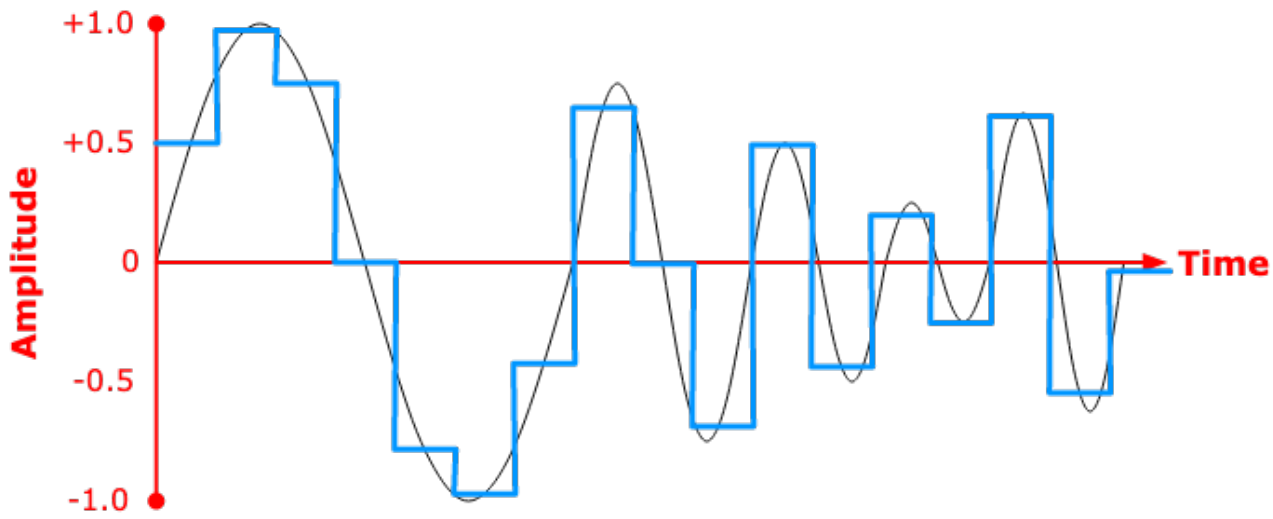


Рис. 1.1 Представлення звуку в цифровому форматі

У цьому прикладі синя лінія представляє вибірки, взяті з аудіо осцилограми, яка має чорний колір. Через регулярні проміжки часу схема АЦП зчитує напругу сигналу як значення між (у цьому випадку) -1,0 і +1,0. Оскільки амплітуда змінюється протягом цього періоду часу, аналого-цифровий

перетворювач повинен вибрати значення для представлення цього зрізу, чи взявши значення в певний момент (на діаграмі вище середина кожного фрагмента використовується як значення) або усереднюючи амплітуду за тривалість кожної вибірки. Ці значення вибірки потім записуються як амплітуда сигналу в цей час.

Коли приходить час відтворити цей звук пізніше, ці амплітуди використовуються для генерування наближення вихідної форми сигналу; замість відтворення точної копії вихідної гладкої хвилі відтворюється більш груба синя хвиля.

Чим частіше беруться зразки оригінального аудіо, тим ближче до оригіналу ви можете отримати. Кількість вибірок за секунду називається частотою дискретизації (sample rate). Чим більше зразків береться, тим плавніше стає хвиля.

1.1.2 Формат і структура аудіоданих

На найпростішому рівні аудіо представлено потоком вибірок, кожен із яких визначає амплітуду аудіосигналу, виміряну для даного фрагмента загальної форми звукового сигналу. Існує кілька форматів, що використовуються для окремих зразків у аудіофайлі. Більшість аудіофайлів використовують 16-розрядні цілі числа зі знаком для кожного зразка, але інші використовують 32-розрядні значення з плаваючою комою або 24-розрядні чи 32-розрядні цілі числа. Крім того, зразки також можуть використовувати знакові або беззнакові значення. Розмір окремої вибірки називається розміром вибірки (sample size).

Положення кожного джерела звуку в звуковому сигналі називається каналом . Кожен канал містить вибірку, яка вказує амплітуду звуку, що створюється цим джерелом у певний момент часу. Наприклад, у стереозвуку є два джерела звуку: один динамік ліворуч і один праворуч. Кожен з них представлений одним каналом, а кількість каналів, що містяться в аудіосигналі, називається кількістю каналів .

Під час запису або створення багатоканальних аудіофайлів канали збираються в серію аудіокадрів , кожен з яких складається з одного зразка для

кожного з аудіоканалів. Окремий зразок — це числове значення, що представляє амплітуду звукової форми в певний момент часу і може бути представлене в різних форматах.

Стерео-аудіо є, ймовірно, найбільш часто використовуваним розташуванням каналів у веб-аудіо, а 16-бітові зразки використовуються для більшості щоденних аудіо, які використовуються сьогодні. Для 16-бітового стереоаудіо кожна вибірка, взята з аналогового сигналу, записується у вигляді двох 16-бітових цілих чисел, одне для лівого каналу і одне для правого. Це означає, що для кожного зразка потрібно 32 біти пам’яті. При загальній частоті дискретизації 48 кГц (48 000 вибірок в секунду) це означає, що кожна секунда аудіо займає 192 КБ пам’яті. Таким чином, типова трихвилинна пісня вимагає близько 34,5 МБ пам’яті. Це багато пам’яті, але, що ще гірше, це шалений обсяг пропускної здатності мережі, який можна використовувати для відносно короткого фрагмента аудіо. Ось чому більшість цифрового аудіо стискається.
Процес стиснення та декомпресії аудіо виконується шляхом його кодування та декодування за допомогою аудіокодека (CO der/ DE coder).

1.1.3 Аудіоканали та кадри

Існує два типи аудіоканалів. Стандартні аудіоканали використовуються для передачі більшості звукового сигналу. Звук для лівого та правого основних каналів, а також усіх ваших динаміків об’ємного звучання (центральний, лівий і правий задні, лівий і правий бічні, стельові канали тощо) є стандартними аудіоканалами. Спеціальні канали низькочастотного покращення (LFE) забезпечують сигнал для спеціальних динаміків, призначених для створення низькочастотних звуків і вібрації для створення нутрощного відчуття під час прослуховування аудіо. Канали LFE зазвичай керують сабвуферами та подібними пристроями.

Монофонічний звук має один канал, стереозвук має два канали, об’ємний звук 5.1 має 6 каналів (п’ять стандартних і один LFE) і так далі. Кожен аудіокадр є записом даних, який містить вибірки для всіх каналів, доступних в аудіосигналі. Розмір аудіокадру обчислюється шляхом множення розміру

вибірки в байтах на кількість каналів, тож один кадр стереофонічного 16-бітового аудіо має довжину 4 байти, а один кадр звуку з плаваючою комою 5.1 - 24 (4 байти). за вибірку, помножену на 6 каналів).

Кількість кадрів, які становлять одну секунду аудіо, залежить від частоти дискретизації, яка використовується під час запису звуку. Оскільки частота дискретизації відповідає кількості «зрізів», на які ділиться звукова хвиля за кожну секунду часу, її іноді вважають частотою (у тому сенсі, що це опис чогось, що періодично повторюється, а не в термінах фактичного частота звуку), і тому вимірювання вибірок за секунду використовує герц як одиницю.

1.2 Обробка початкового набору аудіоданих

Після аналізу характеристик цифрового звуку та отримання розуміння, як комп’ютер працює із звуком, ми повинні знайти та підготувати набір даних, на основі яких ми будемо навчати нейрону мережу для Розпізнавання заїкання. Проблемою є те, що знайти аудіофайли людей із заїканням дуже важко, особливо жінок. Такі люди не дуже люблять записувати себе, тим паче викладати в мережу. І звичайно в відкритих джерелах майже немає записів мови із заїканням українською. А так як нам треба мати багато різних екземплярів голосу, с різним тембром та від різних людей, то ми можемо використовувати лише записи англійською мовою, які все ж вдалося знайти.

1.2.1 Інформація про початковий набір даних

Набір даних, який ми використовуємо для навчання наших моделей для виявлення заїкання - це архів заїкання Університетського коледжу Лондона. Він містить записи фактичних людей, які заїкаються, різного віку (в основному шкільного віку) та різного ступеня тяжкості. Записи зроблені під час монологу та читання. Папка монологу складається з 82 записів (6 жіночих і 76 чоловічих), а папка для читання - 108 записів (15 жіночих і 93 чоловічих). Ми припускаємо, що цей дисбаланс у вибірках даних для жінок і чоловіків цілеспрямовано створений через те, що заїкання частіше зустрічається серед хлопчиків, ніж у дівчат, а співвідношення чоловіків і жінок велике, приблизно 4 до 1. Нажаль,

достатній набір даних із мовою різних людей із заїканням є лише у форматі записів дітей. Це означає, що у майбутньому нейронна мережа вивчена на ціх даних буде мати складнощі із Розпізнавання заїкання у дорослих. Але це не зашкодить використання цієї програми дітьми, адже зараз багато учнів вчаться із дому, використовуючи програмні записи. А для дітей із заїканням це набагато більший стресс ніж для дорослих, тому створена програма буде все ще дуже користною. Аудіофайли були у 2 форматах – .mp3 та .wav. Ми віддали перевагу .wav перед .mp3, оскільки файли .wav не мають втрат і не стиснуті, таким чином зберігаючи оригінальну якість записів.

Всі аудіозаписи мають такі характеристики:

Формат: .wav

Частота дискретизації: 44100Гц

Кількість каналів: 1

Sample width: 2

Frame width: 2

1.2.2 Сегментація та класифікація датасету

Ми вирізали із набору даних найчіткіші та найрепрезинтативніші частини мовлення із заїканням та без. Обрізані аудіо із заїканням містять частини мови із повторенням звуків, вони розділені на набори із різним рівнем серйозності заїкання, легким та важким. Також існує один звичайний набір обрізаних аудіофайлів, без заїкань. Цей набір також містить кілька зразків лише фонового шуму. В результаті ми отримали три набори аудіофайлів: з легким ступеннем заїканням, важким ступенем заїкання та без заїкання(набір містить як плавну мову, так і фоновий шум).

Обрізані аудіофайли далі розрізаються на сегменти фіксованого розміру по одній секунді кожен, і кожному сегменту надається мітка в залежності від того у якій папці був початковий аудіофайл. Мітки бувають трьох видів:

0 - сегмент звуку із папки без заїкань

1 - сегмент звуку із папки із слабким заїканням

2 - сегмент звуку із папки із сильним заїканням

У всіх етапах розробки ПЗ, будування моделі, підготовці датасету та ін, ми використовували мову Python – це інтерпретована, об’єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Його високорівневі вбудовані структури даних у поєднанні з динамічним типізацією та динамічним зв’язуванням роблять його дуже привабливим для швидкої розробки додатків, а також для використання в якості мови сценаріїв або склеювання для з’єднання існуючих компонентів. Простий синтаксис Python, який легко вивчати, підкреслює читабельність і, отже, знижує витрати на обслуговування програми. Python підтримує модулі та пакунки, що сприяє модульності програм і повторному використанню коду. Інтерпретатор Python має стандартні бібліотеки доступні як у скомпільованій, так і у чистій формі на всіх основних платформах. В мові програмування Python підтримується багато парадигм програмування, таких як: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована. Серед переваг мови програмування можна визначити такі:

- Python працює на різних платформах (Windows, Mac, Linux, Raspberry Pi тощо).
- Python має простий синтаксис, схожий на англійську мову.
- Python має синтаксис, який дозволяє розробникам писати програми з меншою кількістю рядків, ніж деякі інші мови програмування.
- Python працює на системі інтерпретатора, що означає, що код може бути виконаний, як тільки він буде написаний. Це означає, що створення прототипу може бути дуже швидким.
- Python можна обробляти процедурним, об’єктно-орієнтованим або функціональним способом.

Ми вибрали мову Python бо вона краще всього підходить для наших задач. Вона надає багато інструментів які будуть корисні у нашій роботі, а саме: інструменти для роботи із аудіо (як файлами так і потоковим звуком), інструменти для роботи із даними та їх перетворення, інструменти для програмування моделей та їх використання.

Для сегментації наших аудіовідрізків ви використали бібліотеку PyDub - бібліотека яка надає широкий функціонал для роботи із аудіофайлами та звуком. Останній сегмент аудіозапису майже завжди видяляється тому, що він менший за одну секунду і з ним складно буде працювати у подальшому.

У результаті ми отримали купу аудіосегментів по одній секунді кожен, та документ, де ми позначаємо ступінь заїкання у кожному сегменті. Ці дані будуть нам потрібні для подальшої роботи.

Також ми використали декілька стандартних бібліотек для полегшення роботи із велики масивами даних. Це такі бібліотеки як Pandas та NumPy.

NumPy - це бібліотека Python, яка використовується для роботи з масивами, розшифровується як числовий Python. Вона також має функції для роботи в області лінійної алгебри, перетворення Фур'є та матриць. У Python у нас є списки, які служать меті масивів, але вони повільно обробляються.
NumPy має на меті надати об’єкт масиву до 50 разів швидше, ніж традиційні списки Python. Масиви дуже часто використовуються в науці про дані, де швидкість і ресурси дуже важливі.

Pandas - це бібліотека Python, яка використовується для роботи з наборами даних. Він має функції для аналізу, очищення, дослідження та маніпулювання даними. Pandas дозволяє нам аналізувати великі дані та робити висновки на основі статистичних теорій, може очищати безладні набори даних, робити їх читабельними та релевантними. Релевантні дані дуже важливі в науці про дані.

1.3 Збір кепстральних характеристик

У якості даних для вивчення нейронної мережі можна взяти одну із характеристик звуку. Таких характеристик, що підходять для навчання є багато, серед популярніших визначають такі: MFCC, LPC, LPCC, LSF, PLP і DWT. Але найгрунтованішою на найрепрезентативнішою характеристикою є MFCC. Саме її ми і будемо використовувати для навчання мережі у майбутньому.

MFCC - мелчастотні кепстральні коефіцієнти. Мел-шкала є емпіричною шкалою, що ґрунтується на людському відчутті частоти звуку. На основі MFCC

розраховуються ознаки кольоровості для нейронних мереж при розпізнаванні конкретної голосової команди.

Висота тону є однією з характеристик мовного сигналу і вимірюється як частота сигналу. Шкала Mel – це шкала, яка пов’язує сприйняту частоту тону з фактично виміряною частотою. Вона масштабує частоту, щоб точніше відповідати тому, що може почути людське вухо (люди краще визначають невеликі зміни в мовленні на нижчих частотах). Ця шкала була отримана на основі експериментів на людях.

Діапазон слуху людини становить від 20 Гц до 20 кГц. Уявіть собі мелодію на 300 Гц. Це звучало б приблизно на зразок стандартного сигналу набору номера стаціонарного телефону. Тепер уявіть собі мелодію з частотою 400 Гц (трохи більш високий тон набору номера). Тепер порівняйте відстань між цими двома, як би це не сприймалося вашим мозком. Тепер уявіть собі сигнал частотою 900 Гц (схожий на звук зворотного зв’язку мікрофона) і звук 1 кГц. Відстань між цими двома звуками може здатися більшою, ніж між першими двома, хоча фактична різниця така ж (100 Гц). Шкала mel намагається вловити такі відмінності.

Будь-який звук, який створює людина, визначається формою їх голосового тракту (включаючи язик, зуби тощо). Якщо цю форму можна визначити правильно, будь-який звук, який виробляється, можна точно відобразити. Огинаюча спектру часової потужності мовного сигналу є репрезентативною для голосового тракту, і MFCC (який є не що інше, як коефіцієнти, що складають кепстр частоти Mel) точно представляє цю огинаючу. Кепстр – це результат дискретного косинусного перетворення логарифма амплітудного спектру сигналу. Мел-шкала описує частотну чутливість людського слуху – враховується, що зміна частоти в два рази в діапазоні низьких та високих частот людина сприймає порізному. Мел-частотні кепстральні коефіцієнти - це розподілені по мел-шкалі значення кепстру із використанням банку фільтрів.

Алгоритм знаходження MFCC такий:

1) Попередньо оброблений мовний сигнал розбивається на певну кількість відрізків тривалістю 20 мс;
2) До кожного відрізку застосовується дискретне перетворення Фур'є;
3) Знаходиться спектральна потужність щільності отриманого сигналу;

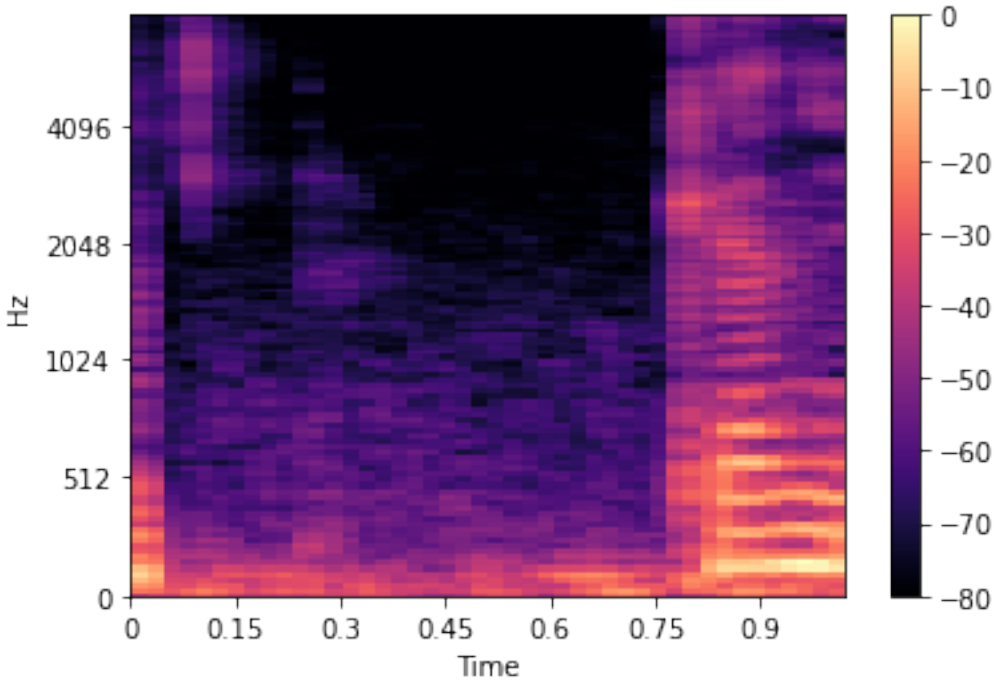


Рис. 1.2 Діаграма спектральної потужності

4) Застосовуючи банк фільтрів, спектр розподіляється за мел-шкалою (кількість каналів – 20, кількість кепстральних коефіцієнтів – 13, частотний діапазон – 300-3700 Гц);
5) Логарифмуємо результат:

$$Xn[i] = \ln(Xn[i]),\ i = 1,\ldots,P;$$

(1.1)

6) Здійснюємо дискретне косинусне перетворення результату

$$C_n[j] = \sum_{k=1}^{P} Xn[k]\cos\left(j\left(k-\frac{1}{2}\right)\frac{\pi}{P}\right),\ i = 1,\ldots,P,\ j = 1,\ldots,J,$$

(1.2)

де Cn[j] – масив кепстральних коефіцієнтів,

k – кількість відрізків;

P – кількість фільтрів;

J – бажане число коефіцієнтів, J < P . Для прикладу отриману матрицю кепстральних коефіцієнтів можна зобразити у вигляді бітової карти, зображену на рис нижче.

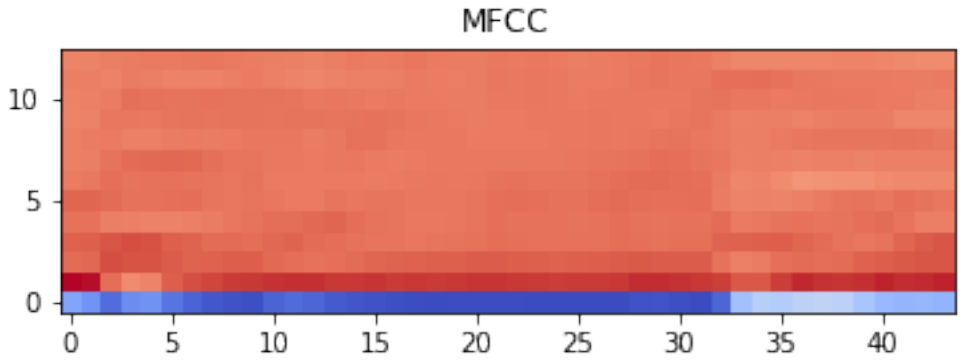


Рис. 1.3 матриця кепстральних коефіцієнтів

Треба зазначити, що ми будемо використовувати лише перші 13 коефіцієнтів MFCC, бо саме вони корелюють із заїканням та мовленням взагалі. Використання інших характеристин не показало бажаного результату, тому було вирішено не брати їх до розрахунків, щоб не збільшувати масив даних та час навчання моделі.

Після знаходження всіх коефіцієнтві ми зглажуємо результат та знаходимо дельту характеристик за допомогою фільтру Савіцького - Голая.

Це цифровий фільтр, який можна застосувати до набору точок цифрових даних з метою згладжування даних, тобто підвищення точності даних без спотворення тенденції сигналу. Це досягається в процесі, відомому як згортка, шляхом підбору успішних підмножин суміжних точок даних з поліномом низького ступеня за методом лінійних найменших квадратів. Коли точки даних розташовані на однаковому відстані, можна знайти аналітичне рішення рівнянь найменших квадратів у вигляді єдиного набору «коефіцієнтів згортки», які можна застосувати до всіх підмножин даних, щоб дати оцінки згладженого сигнал (або похідні від згладженого сигналу) у центральній точці кожного підмножини.

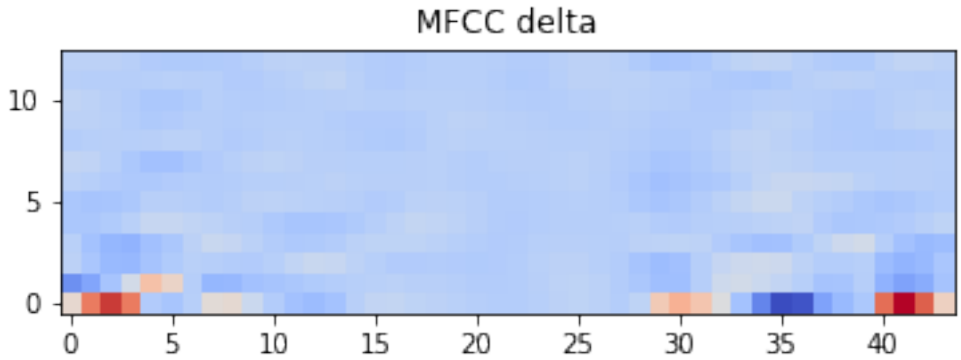


Рис. 1.4 матриця кепстральних коефіцієнтів після фільтраці

У результаті проведених операцій ми отривали дані для навчання нашої моделі Розпізнавання заїкання. Ми отримали набір із близько тисячи ста матриць кепстральних коефіцієнтів розмірності 13 x 44, та окремий набір із мітками класифікації сегменту аудіодоріжки на ступінь заїкання. Ці дані ми використаємо для навчання нашої мережі.

1.4 Висновок до розділу

У даному розділі ми розглянули характеристики цифрового звуку, які використовуються в машиному навчані нейронних мереж для роботи із аудіоданими. Найбільш ефективним є використання Mel-частотних кепстральних характеристик звуку. Ми розглянули математичну основу на процес витягу та обробки цих коефіцієнтів.

Також ми знайшли достатній набір даних різних людей із заїканням, який ми будемо використовувати для навчання. Обробили цей набір, оцінили ступінь заїкання, та розбили на сегменти по одній секунді, з яких потім витягли MFCC характеристики.

Таким чином ми отримали два набори. Перший це набір із близько 1100 матриць розмірності 13 x 44, у якому ми зберігаємо Mel-коефіцієнти. Другий же набір містить дані про нашу оцінку ступеня заїкання для фрагментів. Ці дані ми використовуємо для подальшої роботи та навчання нейронної мережі.

РОЗДІЛ 2: РОЗРОБКА НЕЙРОННОЇ МЕРЕЖІ ДЛЯ РОЗПІЗНАВАННЯ ЗАЇКАННЯ

Тепер коли ми маємо датасет для навчання, наша задача на данному етапі це розробити модель програми яка б визначала, чи є ознаки заїкання у мові, та на скільки великі. Існує багато способів машиного навчання для класифікації даних, які ми і розглянимо далі.

2.1 Вибір методу класифікації даних

Класифікація - це завдання аналізу даних, тобто процес пошуку моделі, яка описує та розрізняє класи та концепти даних. Класифікація - це проблема визначення, до якого з набору категорій належить нове спостереження, на основі навчального набору даних, що містить спостереження, і належність до категорій яких відома. Цей метод часто передбачає використання алгоритмів, які можна легко адаптувати для покращення якості даних. Ось чому контрольоване навчання тісно пов’язане з процесом класифікації в аналізі даних. Кінцева мета класифікації полягає в тому, щоб зв'язати змінну, що цікавить, із змінними, що спостерігаються. Алгоритм, необхідний для виконання класифікації, відомий як класифікатор. І зроблені спостереження відомі як випадки. Модель класифікації в аналізі даних використовується, коли змінна фокусу є більш «якісною». Існує кілька різних типів класифікації. І кожен з цих алгоритмів використовується для вилучення корисної інформації з набору даних.

Алгоритми класифікації можна розділити дві категорії:

- Алгоритм генеративної класифікації моделює розподіл окремих класів. Він намагається вивчити модель, яка створює дані через оцінку розподілів і припущень моделі. Можна використовувати генеративні алгоритми для прогнозування невидимих даних. Відомим генеративним алгоритмом є наївний байєсовий класифікатор.

- Дискримінаційний - Це рудиментарний алгоритм класифікації, який визначає клас для рядка даних. Він моделює з використанням спостережуваних

даних і залежить від якості даних, а не від їх розподілу. Логістична регресія є відмінним типом дискримінаційних класифікаторів.

Класифікація є дуже популярним аспектом інтелекту даних. В результаті машинне навчання має багато класифікаторів:

- Логістична регресія дозволяє моделювати ймовірність певної події або класу. Він використовує логістику для моделювання двійкової залежної змінної. Це дає вам ймовірності одного випробування. Оскільки логістична регресія була створена для класифікації та допомагає зрозуміти вплив кількох незалежних змінних на одну змінну результату. Проблема логістичної регресії полягає в тому, що вона працює лише тоді, коли ваша прогнозована змінна є двійковою, а всі провісники незалежні. Крім того, передбачається, що дані не мають відсутніх значень, що може бути серйозною проблемою.

- Лінійна регресія заснована на навчанні під керівництвом і виконує регресію. Вона моделює значення прогнозу відповідно до незалежних змінних. Насамперед, ми використовуємо її, щоб з'ясувати зв'язок між прогнозом і змінними. Він прогнозує значення залежної змінної відповідно до конкретної незалежної змінної. Зокрема, він знаходить лінійну залежність між незалежною змінною та залежною змінною. Це відмінно підходить для даних, які ви можете розділити лінійно, і є високоефективним. Але це метод спирається на припущення, що незалежні та залежні змінні пов’язані лінійно.

- Дерево рішень є найнадійнішим методом класифікації в аналізі даних. Це блок-схема, схожа на структуру дерева. Тут кожен внутрішній вузол посилається на перевірку умови, а кожна гілка означає результат тесту (істинний він чи хибний). Кожен листовий вузол дерева рішень має мітку класу. Можна розділити дані на різні класи відповідно до дерева рішень. Він передбачатиме, до яких класів буде належати нова точка даних відповідно до створеного дерева рішень. Межами його прогнозування є вертикальні та горизонтальні лінії.

- Класифікатор випадкових лісів відповідає кільком деревам рішень на різних підвибірках набору даних. Він використовує середнє значення для підвищення точності прогнозування та управління переобладнанням. Розмір підвибірки

завжди дорівнює розміру вхідної вибірки; проте зразки відбираються із заміною. Особлива перевага класифікатора випадкових лісів полягає в тому, що він зменшує переобладнання. Більше того, цей класифікатор має значно більшу точність, ніж дерева рішень. Однак це набагато повільніший алгоритм для прогнозування в реальному часі і є дуже складним алгоритмом, тому його дуже складно ефективно реалізувати.

- Алгоритм наївного Байєса передбачає, що кожна ознака незалежна одна від одної і що всі ознаки однаково вносяться в результат. Інше припущення, на яке спирається цей алгоритм, полягає в тому, що всі функції мають однакову важливість. У сучасному світі він має багато застосувань, таких як фільтрація спаму та класифікація документів. Наївному Байєсу потрібна лише невелика кількість навчальних даних для оцінки необхідних параметрів. Більше того, наївний байєсівський класифікатор значно швидше, ніж інші складні та просунуті класифікатори. Однак наївний байєсівський класифікатор відомий своєю поганою оцінкою, оскільки передбачає, що всі ознаки однаково важливі, що не відповідає дійсності в більшості реальних сценаріїв.

- Машинний алгоритм опорного вектора , також відомий як SVM, представляє навчальні дані в просторі, розділені на категорії за великими пробілами. Нові точки даних потім відображаються в тому самому просторі, а їхні категорії прогнозуються відповідно до сторони розриву, в який вони потрапляють. Цей алгоритм особливо корисний у просторах високої розмірності і є досить ефективним для пам’яті, оскільки використовує лише підмножину навчальних точок у своїй функції прийняття рішень. Цей алгоритм відстає в наданні оцінок ймовірності. Потрібно буде розрахувати їх за допомогою п’ятикратної перехресної перевірки, що дуже дорого.

- Алгоритм k-найближчого сусіда має нелінійні межі передбачення, оскільки він є нелінійним класифікатором. Він прогнозує клас нової точки даних тесту, знаходячи клас її k найближчих сусідів. Ви б вибрали k найближчих сусідів контрольної точки даних, використовуючи евклідову відстань. У k найближчих сусідів вам потрібно буде підрахувати кількість точок даних, присутніх у різних категоріях, і призначити нову точку даних категорії з найбільшою кількістю

сусідів. Це досить дорогий алгоритм, оскільки для пошуку значення k потрібно багато ресурсів. Крім того, він також повинен розраховувати відстань кожного екземпляра до кожної навчальної вибірки, що ще більше збільшує його обчислювальні витрати.

- Нейронні мережі є непараметричними моделями, що не вимагають припущень про імовірнісне розподілення даних, але при цьому не використовують міри відстаней. Це робить їх універсальними класифікаторами, дозволяючи отримувати результати навіть у випадках, коли параметричні та метричні класифікатори не забезпечують прийнятного рішення. Вони мають такі переваги: є самонавчальними моделями, робота яких практично не вимагає втручання користувача; є універсальними апроксиматорами, що дозволяють апроксимувати будь-яку безперервну функцію з прийнятною точністю; є нелінійними моделями, що дозволяє ефективно вирішувати завдання класифікації навіть за відсутності лінійної роздільності класів.

Для того щоб зрозуміти наскільки нам вдалася класифікація - треба оцінити її точніть. Така оцінка може бути проведена за допомогою крос-перевірки. Кросперевірка є процедурою оцінки точності класифікації для даних з тестової множини, їх також можна назвати кросперевірочною множиною.

Точність класифікації такої тестової множини зазвичай порівнюється з точністю класифікації множини із навчальної виборки. Якщо така класифікація тестової виборки має такі ж результати, як і класифікація навчальної виборки, то будемо вважати, що модель пройшла кросперевірку успішно. Наш розділ на навчальну і тестову виборку здійснюється шляхом певного ділення множини у пропорції, наприклад скажемо що, навчальна множина - це дві третини даних і тестова множина - це одна третина даних. Цей спосіб можна використовувати для вибірок із великою кількістю екземплярів. Якщо ж наша вибірка має малі обсяги, то треба застосовувати спеціальні методи, при використанні яких навчальна і тестова вибірки можуть частково перетинатися.

Оцінювання методів класифікації слід проводити, виходячи з певних характеристик: надійність, стійкість, швидкість, інтерпретованість.

Швидкість характеризується часом який потрібен на створення такої

моделі та використання цієї моделі. Стійкість до порушень вихідних умов, означає можливість праці з зашумленими даними і пропущеними значеннями в них. Інтерпретованість - це можливість розуміння моделі аналітиком даних. Надійність таких методів класифікації має передбачати можливість роботи цих методів при наявності в наборі шумів та викидів.

Отже для найбільш точного результату ми будемо проводути класифікацію даних за допомогою штучної нейронної мережі, яка може дати нам найбільш успішний результат.

2.2 Аналіз типів штучних нейроних мереж

Нейронні мережі - це набір алгоритмів, змодельованих за моделлю людського мозку, які призначені для розпізнавання шаблонів. Вони інтерпретують сенсорні дані за допомогою свого роду машинного сприйняття, маркування або групування вихідних даних. Патерни, які вони розпізнають, є числовими, містяться у векторах, у які мають бути переведені всі дані реального світу, будь то зображення, звук, текст чи часові ряди.

Нейронні мережі допомагають нам групувати та класифікувати дані. Можна розглядати їх як рівень кластеризації та класифікації поверх даних, які збираються та коригуються. Вони допомагають групувати немарковані дані відповідно до подібності між прикладами вхідних даних і класифікують дані, коли у них є позначений набір даних для навчання. Нейронні мережі також можуть витягувати функції, які подаються в інші алгоритми для кластеризації та класифікації; тому можна розглядати глибокі нейронні мережі як компоненти більших програм машинного навчання, що включають алгоритми навчання з підкріпленням, класифікації та регресії.

Глибоке навчання - не навчання нейронних мереж що складаються з кількох шарів. Шари складаються з вузлів . Вузол - це просто місце, де відбуваються обчислення, нещільно нанесені на нейрон в мозку людини, який спрацьовує, коли зустрічає достатню кількість подразників. Вузол поєднує вхідні дані з набором коефіцієнтів або ваг, які або посилюють, або гасять цей вхід, тим самим призначаючи значення вхідних даних щодо завдання, яке

намагається вивчити алгоритм. Ці вхідні вагові продукти підсумовуються, а потім сума передається через так звану функцію активації вузла, щоб визначити, чи повинен цей сигнал просуватися далі через мережу, щоб вплинути на кінцевий результат, скажімо, акт класифікації. Якщо сигнали проходять, нейрон був «активований».

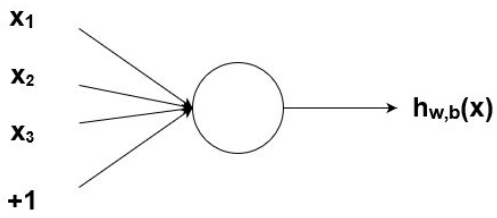


Рис. 2.1 Вузли із входами

Вузловий шар - це ряд нейроноподібних перемикачів, які вмикаються або вимикаються, коли вхід подається через мережу. Вихід кожного шару одночасно є введенням наступного шару, починаючи з початкового вхідного рівня, який отримує ваші дані. Поєднання регульованих ваг моделі з вхідними функціями – це те, як ми приділяємо значення цим характеристикам щодо того, як нейронна мережа класифікує та кластерує вхідні дані.

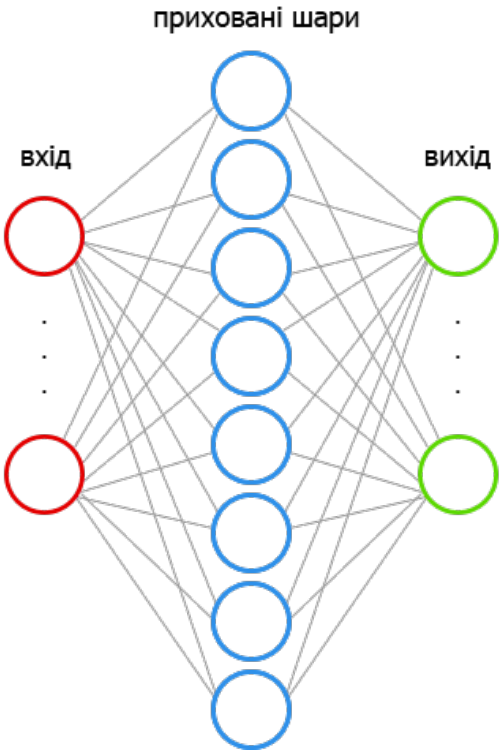


Рис. 2.2 абстрактна архітектура нейронної мережі

Мережі глибокого навчання відрізняються від більш звичайних одношарових прихованих нейронних мереж своєю глибиною, тобто кількість шарів вузлів, через які дані повинні проходити в багатоетапному процесі розпізнавання образів. Попередні версії нейронних мереж, такі як перші персептрони, були неглибокими, складалися з одного вхідного та вихідного шарів і щонайбільше одного прихованого шару між ними. Більше трьох рівнів (включаючи вхідний і вихідний) кваліфікується як «глибоке» навчання. Тож глибокий - це строго визначений термін, який означає більше ніж один прихований шар. У мережах глибокого навчання кожен рівень вузлів тренується на певному наборі функцій на основі результатів попереднього рівня. Чим далі ви просуваєтеся в нейронну мережу, тим складніші функції можуть розпізнати ваші вузли, оскільки вони об’єднують і рекомбінують функції з попереднього шару.

Це явище відоме як ієрархія функцій , і це ієрархія зростаючої складності та абстракції. Це робить мережі глибокого навчання здатними обробляти дуже великі масиви даних з мільярдами параметрів, які проходять через нелінійні функції. Перш за все, ці нейронні мережі здатні виявляти приховані структури в немаркованих, неструктурованих даних, які є переважною більшістю даних у світі. Під час навчання на немаркованих даних кожен вузловий шар у глибокій мережі автоматично вивчає особливості, багаторазово намагаючись відновити вхідні дані, з яких він бере свої вибірки, намагаючись мінімізувати різницю між припущеннями мережі та розподілом ймовірностей самих вхідних даних.

У процесі ці нейронні мережі вчаться розпізнавати кореляції між певними релевантними характеристиками та оптимальними результатами – вони створюють зв’язки між сигналами ознак і тим, що ці функції представляють, чи то повна реконструкція, чи з позначеними даними.

Мережу глибокого навчання, навчену поміченими даними, потім можна застосувати до неструктурованих даних, надаючи їй доступ до набагато більшої кількості вхідних даних, ніж мережі машинного навчання. Це рецепт вищої продуктивності: чим більше даних може тренуватися мережа, тим точнішою

вона буде. (Погані алгоритми, навчені на великій кількості даних, можуть перевершити хороші алгоритми, навчені на дуже малому.) Здатність глибокого навчання обробляти та навчатися на величезній кількості немаркованих даних дає йому явну перевагу над попередніми алгоритмами. Мережі глибокого навчання закінчуються вихідним рівнем: логістичним, або softmax, класифікатором, який призначає ймовірність певного результату або мітки. Ми називаємо це прогнозним, але воно є прогнозним у широкому сенсі. Враховуючи вихідні дані у вигляді зображення, мережа глибокого навчання може вирішити, наприклад, що вхідні дані на 90 відсотків представляють людину.

Отже можна сказати, що нейронна мережа - це коригувальний цикл зворотного зв’язку, який нагороджує ваги, які підтримують її правильні припущення, і карають вагові показники, які призводять до помилок.

Для того, щоб мережа виконувала складні завдання, її необхідно навчити. Нейронні мережі не програмуються, а навчаються. Наявність навчання - головна перевага штучних нейронних мереж перед традиційними алгоритмами. У штучному інтелекті і машинному навчанні є два основних підходи: навчання з наглядом і навчання без нагляду. Основна відмінність полягає в тому, що один використовує мічені дані, щоб спрогнозувати результати, а інший - ні. Однак між цими двома підходами є певні нюанси.

Кероване навчання(навчання з вчителем) - це підхід машинного навчання, який визначається використанням позначених наборів даних. Ці набори даних призначені для навчання чи «нагляду» алгоритмів щодо класифікації даних або точного прогнозування результатів. Використовуючи позначені вхідні та вихідні дані, модель може вимірювати свою точність і навчатися з часом. Кероване навчання можна розділити на два типи проблем під час аналізу даних - класифікація та регресія.

Рис. 2.3 процес навчання з вчителем

Неконтрольоване навчання(навчання без вчителя) використовує алгоритми машинного навчання для аналізу та групування нерозмічених наборів даних. Ці алгоритми виявляють приховані закономірності в даних без необхідності втручання людини. Моделі навчання без нагляду використовуються для трьох основних завдань: кластеризації, асоціації та зменшення розмірності.

Основна відмінність між цими двома підходами полягає у використанні позначених наборів даних. Простіше кажучи, контрольоване навчання використовує позначені вхідні та вихідні дані, а алгоритм навчання без нагляду - ні. Під час навчання з керівництвом алгоритм «вчиться» з навчального набору даних шляхом ітераційного прогнозування даних і коригування для правильної відповіді. Хоча моделі навчання під керівництвом, як правило, більш точні, ніж моделі навчання без нагляду, вони вимагають попереднього втручання людини для належного позначення даних.

Також з’явилося найбільш продвинутий спосіб навчання з підкріпленням. Воно менше схоже на попередні види, бо нагадує штучний інтелект. Воно використовується не там, де потрібно проаналізувати дані, а там, де потрібно вижити у якомусь середовищі. Середовищем може бути багато речей: наприклад гра в шахи або автопілот. Знання про середовище таким мережам буде дуже корисне, але знати усе їм не обов'язково. Завдання таких мереж -

мінімізувати помилки та збільшити вигоду. Таке навчання дуже схоже на реальне навчання людей.

Так як нас цікавить проблема класифікації даних, ми будемо використовувати навчання з учителем.

2.3 Огляд моделі CNN

Для нашої задачі класифікації звуку на сьогодні краще всього підходить згорткова нейрона мережа (CNN). Штучний інтелект став свідком монументального зростання в подоланні розриву між можливостями людей і машин. Порядок денний у цій галузі полягає в тому, щоб машини могли бачити світ, як люди, сприймати його подібним чином і навіть використовувати знання для багатьох завдань, таких як розпізнавання зображень і відео, аналіз і класифікація зображень, відтворення медіа, системи рекомендацій, обробка природної мови тощо. Удосконалення комп'ютерного зору з глибоким навчанням було створено та вдосконалено з часом, насамперед за допомогою одного конкретного алгоритму - згорткової нейронної мережі.

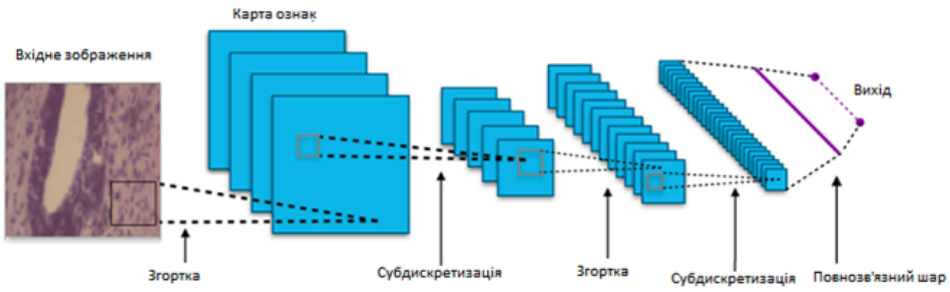


Рис. 2.4 Структура CNN

Згорткова нейронна мережа (CNN) — це алгоритм глибокого навчання, який може приймати вхідне зображення, призначати важливість різним аспектам/об'єктам зображення та мати можливість відрізнити один від іншого. Попередня обробка, необхідна в ConvNet, набагато нижча в порівнянні з іншими алгоритмами класифікації. Хоча в примітивних методах фільтри

розробляються вручну, при достатньому підготовці, ConvNets мають можливість вивчати ці фільтри/характеристики. Архітектура ConvNet аналогічна структурі зв’язку нейронів у людському мозку і була натхненна організацією зорової кори. Окремі нейрони реагують на подразники лише в обмеженій області поля зору, відомому як рецептивне поле. Набір таких полів перекривається, щоб охопити всю візуальну область.

У випадках надзвичайно простих двійкових зображень метод може показувати середню оцінку точності під час виконання передбачення класів, але не матиме практичної точності, коли справа доходить до складних зображень, що мають піксельну залежність у всьому. ConvNet може успішно фіксувати просторові та часові залежності в зображенні за допомогою застосування відповідних фільтрів. Архітектура краще відповідає набору даних зображення завдяки зменшенню кількості задіяних параметрів і можливості повторного використання ваг. Іншими словами, мережу можна навчити краще розуміти витонченість зображення.

CNN мають три основних типи шарів, які:

- Згортковий шар
- Об’єднуючий шар
- Повністю підключений (FC) шар

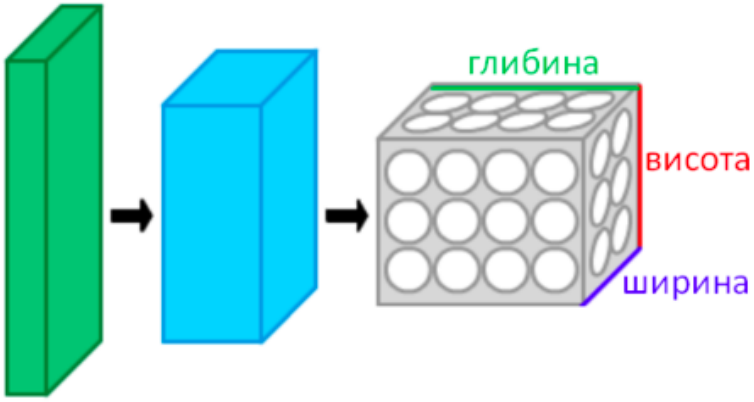


Рис. 2.5 Абстрактні шари CNN

Згортковий шар - це перший шар згорткової мережі. У той час як за згортковими шарами можуть слідувати додаткові згорткові шари або шари

об’єднання, повністю пов’язаний шар є останнім шаром. З кожним шаром CNN збільшується у своїй складності, ідентифікуючи більші частини зображення. Більш ранні шари зосереджені на простих елементах, таких як кольори та краї. Коли дані зображення просуваються через шари CNN, він починає розпізнавати більші елементи або форми об’єкта, поки нарешті не ідентифікує передбачуваний об’єкт.

Згортковий шар є основним будівельним блоком CNN, і саме там відбувається більшість обчислень. Для цього потрібно кілька компонентів, а саме вхідні дані, фільтр і карта об’єктів. Припустимо, що вхідним буде кольорове зображення, яке складається з матриці пікселів у 3D. Це означає, що вхідні дані будуть мати три виміри — висоту, ширину та глибину — які відповідають RGB зображення. У нас також є детектор функцій, також відомий як ядро або фільтр, який буде переміщатися по сприйнятливих полях зображення, перевіряючи, чи є функція. Цей процес відомий як згортка.

Детектор ознак - це двовимірний (2-D) масив ваг, який представляє частину зображення. Хоча вони можуть відрізнятися за розміром, розмір фільтра зазвичай є матрицею 3x3; це також визначає розмір рецептивного поля. Потім фільтр застосовується до області зображення, і між вхідними пікселями та фільтром обчислюється крапковий добуток. Цей точковий добуток потім подається у вихідний масив. Після цього фільтр зміщується на один крок, повторюючи процес, поки ядро не охопить усе зображення. Остаточний вихід із ряду точкових добутків із входу та фільтра відомий як карта ознак, карта активації або згорнута функція. Зрештою, згортковий шар перетворює зображення в числові значення, дозволяючи нейронній мережі інтерпретувати та витягувати відповідні шаблони.

Другий шар це шар об’єднання зменшує дискретизацію та розмірність, зменшуючи кількість параметрів у вхідних даних. Подібно до згорткового шару, операція об’єднання об’єднує фільтр по всьому входу, але різниця в тому, що цей фільтр не має жодних ваг. Замість цього ядро застосовує функцію агрегації до значень у прийнятному полі, заповнюючи вихідний масив. Існує два основних типи об’єднання:

- Максимальне об’єднання: під час переміщення фільтра по входу він вибирає піксель із максимальним значенням для відправки на вихідний масив. Крім того, цей підхід, як правило, використовується частіше в порівнянні зі середнім об’єднанням.
- Середнє об’єднання: коли фільтр переміщується по вхідному сигналу, він обчислює середнє значення в поле сприйняття для надсилання до вихідного масиву. Хоча багато інформації втрачається на рівні об’єднання, це також має ряд переваг для CNN. Вони допомагають зменшити складність, підвищити ефективність та обмежити ризик переобладнання.

Останній шар - повнов’язний. У ньому кожен вузол вихідного шару підключається безпосередньо до вузла попереднього шару. Цей шар виконує завдання класифікації на основі ознак, витягнутих через попередні шари та їх різні фільтри. У той час як рівні згортки та об’єднання, як правило, використовують функції ReLu, рівні FC зазвичай використовують функцію активації softmax для належної класифікації вхідних даних, створюючи ймовірність від 0 до 1.

2.4 Розробка архітектури та навчання нейронної мережі

Тепер коли ми зрозуміли як влаштована згорткова нейрона мережа, можна переходити до конструювання нашої моделі на основі CNN.

2.4.1 Огляд програмних інструментів для розробки нейронної мережі

Для розробки моделі нейронної мережі на основі CNN, ми будемо використовувати мову Python та її бібліотеки. Найпоширеніша бібліотека для створення моделей нейроних мереж - TensorFlow.

TensorFlow - це бібліотека з відкритим кодом для чисельних обчислень і великомасштабного машинного навчання. TensorFlow об’єднує безліч моделей і алгоритмів машинного навчання та глибокого навчання (він же нейронних мереж) і робить їх корисними за допомогою загальної метафори. Він використовує Python, щоб забезпечити зручний інтерфейсний API для

створення додатків з фреймворком, виконуючи ці програми на високопродуктивному C++. TensorFlow може навчати та запускати глибокі нейронні мережі для рукописної класифікації цифр, розпізнавання зображень, вбудовування слів, повторюваних нейронних мереж, моделей від послідовності до послідовності для машинного перекладу, обробки природною мовою та моделювання на основі PDE (диференціальне рівняння з частковими частинами). Найкраще те, що TensorFlow підтримує прогнозування виробництва в масштабі, з тими ж моделями, що використовуються для навчання. Програми TensorFlow можна запускати на будь-якій зручній цілі: локальній машині, кластері в хмарі, пристроях iOS і Android, ЦП або графічних процесорах. Найбільшою перевагою, яку TensorFlow надає для розвитку машинного навчання, є *абстракція*. Замість того, щоб мати справу з дрібними деталями реалізації алгоритмів або з'ясовувати правильні способи прив'язати вихід однієї функції до входу іншої, розробник може зосередитися на загальній логіці програми. TensorFlow подбає про деталі за кадром.

Як допоміжну бібліотеку для будування шарів можелі ми використовуємо бібліотеку Keras - один з провідних API високого рівня для нейронних мереж. Він написаний на Python і підтримує кілька внутрішніх механізмів обчислень нейронних мереж. Keras був створений для зручності користування, модульності, простоти розширення та роботи з Python. API був розроблений для людей, а не для машин, і відповідає найкращим практикам зменшення когнітивного навантаження. Нейронні рівні, функції витрат, оптимізатори, схеми ініціалізації, функції активації та схеми регуляризації - все це окремі модулі, які можна об'єднати для створення нових моделей.

Керас власне не виконує власні операції на низькому рівні, такі як тензорні вироби та звивини; для цього він покладається на внутрішній двигун. Незважаючи на те, що Keras підтримує декілька внутрішніх двигунів, його основним (і за замовчуванням) є TensorFlow, а основним - Google. API Keras поставляється в TensorFlow as tf.keras, який, як уже згадувалося раніше, стане основним TensorFlow API з TensorFlow 2.0. Модель є ядром Keras структури

даних. У Keras доступні два основних типи моделей: Sequential модель та Model клас, що використовується з функціональним API.

2.4.2 Розробка архітектури

Після аналізу нейроних мереж, створення датасету для навчання нейронної мережі та вибору типу нейронної мережі, ми переходими до розробки архітектури. У якості базової моделі ми взяли модифікацію одної із популярніших моделей, CNN яка називається GRCNN.

GRCNN - Згортка нейронна мережа (CNN) стала базовою моделлю для вирішення багатьох проблем комп'ютерного зору. В останні роки був запропонований новий клас CNN, нейронна мережа з рекурентною згорткою (RCNN), натхненна численними повторюваними зв’язками в зорових системах тварин. Критичним елементом RCNN є рекурентний згортковий шар (RCL), який включає повторювані зв’язки між нейронами в стандартному згортковому шарі. Зі збільшенням кількості повторюваних обчислень рецептивні поля (RF) нейронів у RCL необмежено розширюються, що не відповідає біологічним фактам. Ця модель модифікує RF нейронів, вводячи ворота до повторюваних з'єднань. Ворота контролюють кількість контекстної інформації, що вводиться до нейронів, і тому радіочастотні сигнали нейронів стають адаптивними. Отриманий шар називається шаром рекурентної згортки (GRCL). Кілька GRCL складають глибоку модель, яка називається RCNN зі стробуванням (GRCNN). GRCNN був оцінений для кількох завдань комп’ютерного зору, включаючи розпізнавання об’єктів, розпізнавання тексту сцени та виявлення об’єктів, і отримав набагато кращі результати, ніж RCNN. Крім того, у поєднанні з іншими адаптивними радіочастотними методами, GRCNN продемонстрував конкурентоспроможну продуктивність найсучасніших моделей на еталонних наборах даних для цих завдань. У поєднанні з іншими адаптивними радіочастотними методами GRCNN продемонстрував конкурентоспроможну продуктивність найсучасніших моделей на еталонних наборах даних для цих завдань.

Для початку ми завантажили підготовлені датасети. Перший датасет містить Mel-коефіцієнти та має розмір (13 x 44 x 1118). Це означає що ми маємо 13мел коефіцієнтів на кожні 44 герца у одній мілісекунді для одного сегменту. А таких сегментів ми маємо 1118. Отже наша мережа буде навчатися на майже двадцяти хвилинах аудіо. Звичайно це недостатньо для точного навчання для класифікації звуку, але в подальшому після розробки моделі, тестову вибірку можна збільшити.

Також для кожного аудіосегмента ми маємо дані про ступінь заїкання у цьому сегменті, одже маємо ще один набір даних розміром 1118 x 1. Дані зберігаються у форматі .npy. Такий формат дає впевненість, що матриця даних буде виглядати однакова на будь-яких пристроях.

Спочатку ми беремо перший масив із даними та нормалізуємо його за допомогою tf.keras.utils.normalize. Нормалізація даних - це техніка, яка допомагає швидше отримати результат, оскільки машині доводиться обробляти менший діапазон даних. Це буде шар попередньої обробки, який нормалізує безперервні ознаки. Цей рівень буде зміщувати та масштабувати вхідні дані в розподіл з центром навколо 0 зі стандартним відхиленням 1. Він досягає цього шляхом попереднього обчислення середнього значення та дисперсії даних та виклику (input - mean) / sqrt(var)під час виконання.

Нормолізуємо дані по вісі -1. Вісь - ціле число, кортеж цілих чисел або немає. Вісь або осі, які повинні мати окреме середнє значення та дисперсію для кожного індексу у формі. Наприклад, якщо shape має значення (None, 5)і axis=1, шар відстежуватиме 5 окремих середніх значень і значень дисперсії для останньої осі. Якщо axisвстановлено значення None, шар нормалізує всі елементи у вхідних даних за скалярним середнім значенням і дисперсією. За замовчуванням -1, де остання вісь вхідних даних вважається розміром об’єкта і нормалізується для кожного індексу.

Також нормалізуємо за нормою L2 - довжина вектора називається векторною нормою або величиною вектора. Довжина вектора завжди є додатним числом, за винятком вектора з усіма нульовими значеннями. Він обчислюється за допомогою деякої міри, яка підсумовує відстань вектора від

початку координат векторного простору. Наприклад, початок векторного простору для вектора з 3 елементами є (0, 0, 0). Позначення використовуються для представлення векторної норми в більш широких обчисленнях, а тип обчислення векторної норми майже завжди має своє унікальне позначення.

Довжину вектора можна обчислити за допомогою норми L2, де 2 є верхнім індексом L, наприклад L^2. Позначенням для L2-норми вектора є ||v|| 2 , де 2 — індекс. Норма L2 обчислює відстань векторної координати від початку координат векторного простору. Таким чином, вона також відома як евклідова норма, оскільки вона обчислюється як евклідова відстань від початку координат. Результатом є позитивне значення відстані. Як і норма L1, норма L2 часто використовується при підборі алгоритмів машинного навчання як методу регуляризації, наприклад, метод, щоб коефіцієнти моделі були невеликими і, у свою чергу, модель була менш складною.

У результаті нормалізації матриця розмірності (1118, 13, 44) стала розміром (1995, 3, 44). А самі нормалізовані дані можна побачити нижче.

```
[-349.38297733 -339.46299205 -335.9631354  -332.85456749 -335.33701397
 -338.99786663 -335.77088785 -333.44439033 -329.44559082 -322.16420586
 -319.92502913 -316.64271557 -311.43076845 -308.73829721 -309.3209236
 -307.67348592 -303.83712975 -305.39611913 -310.7827827  -312.72902822
 -320.60781548 -332.39978589 -345.72288923 -354.45209134 -345.82691536
 -222.58115458 -144.63731484 -104.18134994  -94.19182443 -125.13382514
 -176.27034638 -187.67775668 -142.13757249 -106.01491192 -103.24835776
 -105.74849192 -113.21542036 -124.97840797 -130.46651078 -136.70618579
 -157.23289474 -202.31283386 -221.40212359 -224.78087842]
after
[-0.19771582 -0.1921021  -0.19012153 -0.18836239 -0.18976721 -0.19183888
 -0.19001274 -0.18869617 -0.18643325 -0.18231272 -0.18104557 -0.17918811
 -0.17623866 -0.174715   -0.1750447  -0.17411242 -0.17194142 -0.17282366
 -0.17587197 -0.17697335 -0.18143196 -0.18810503 -0.19564457 -0.20058443
 -0.19570344 -0.12595867 -0.08185025 -0.05895622 -0.05330315 -0.07081323
 -0.09975138 -0.10620684 -0.08043565 -0.05999384 -0.05842824 -0.05984307
 -0.0640686  -0.07072528 -0.073831   -0.07736203 -0.08897809 -0.11448883
 -0.12529146 -0.12720349]
```

Рис. 2.6 дані до нормалізації та після

Після нормалізації даних ми зробили тестову вибірку у розмірності сорока елементів, що є майже двома відсотками від загальної кількості елементів для навчання. Треба зазначити, що така тестова вибірка може бути надто мала для точної оцінки результатів навчання.

Найпростіший спосіб побудувати нейронну мережу за допомогою TensorFlow - це за допомогою класу Sequential Keras. Нижче опишемо параметри які ми налаштували при створені моделі.

У якості згорткового шару ми використали keras.layers.Conv2D - двовимірний (працюючий з двовимірними даними) згортковий шар. Цей шар створює ядро згортки, яке згортається з вхідним шаром для створення тензора вихідних даних. Якщо use_bias значення True, створюється вектор зміщення та додається до вихідних даних. Нарешті, якщо activationні None, він також застосовується до виходів. Використовуючи цей шар як перший шар у моделі, аргумент ключового слова input_shape (кортеж цілих чисел або None, не включає вісь зразка), ви можете використовувати None, коли розмір має змінний розмір. Цей шар ми використовуємо параметер strides (1, 1) - число або кортеж/ список з 2 цілих чисел, що визначає кроки згортки по висоті та ширині. Може бути одним цілим числом для визначення однакового значення для всіх просторових вимірів. Будь-яке значення кроку != 1 несумісне з визначенням будь-якого dilation_rate значення != 1.

На всіх згорткових шарах крім першого ми використовуємо параметр padding “valid”. Він означає відсутність заповнення. Значення “same" призводить до рівномірного заповнення нулями ліворуч/праворуч або вгору/ вниз від введення. Коли padding="same"і strides=1, вихід має той же розмір, що і вхідний.

Після кожного згорткового шару ми використовуємо шар активації, у якості функції активації ми вибрали найпоширенішу функцію активації ReLU. Це кусково-лінійна функція, яка повертає результат, тільки якщо він позитивний, і в основному використовується для прихованих шарів. Крім того, вихідний рівень повинен мати активацію, сумісну з очікуваним

виходом. Наприклад, лінійна функція підходить для задач регресії, тоді як сигмоїд часто використовується для класифікації.

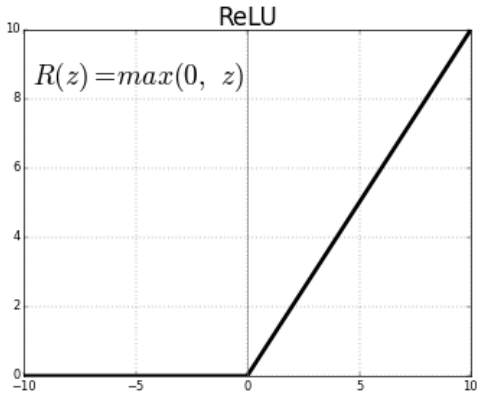


Рис. 2.7 функція активації ReLU

Застосовує функцію активації випрямленої лінійної одиниці.

Зі значеннями за замовчуванням це повертає стандартну активацію ReLU: max(x, 0), поелементний максимум 0 і вхідний тензор. Змінення параметрів за замовчуванням дозволяє використовувати відмінні від нуля пороги, змінювати максимальне значення активації та використовувати ненульове значення, кратне входу для значень нижче порогу.

Також ми намагалися використовувати шар нормалізації BatchNormalization. Шар, який нормалізує свої вхідні дані. Пакетна нормалізація застосовує перетворення, яке підтримує середнє вихідне значення близько 0, а вихідне стандартне відхилення близьке до 1. Важливо, що пакетна нормалізація працює по-різному під час навчання та під час висновку. Під час навчання (тобто під час використання fit()або виклику шару/моделі з аргументом training=True) рівень нормалізує свій вихід, використовуючи середнє значення та стандартне відхилення поточної партії вхідних даних. Але використання даного шару не надало нам необхідну точність, яки ми хотіли, тому було вирішено не використовувати даний шар.

Шар Reshape можна використовувати для зміни розмірів його вхідних даних, не змінюючи його даних, змінюються лише розміри; в процесі не копіюються дані. Додатні числа використовуються безпосередньо, встановлюючи відповідний розмір вихідного blob.

Особливісттю нашої нейромережі є шар GRU, який ми взяли із RNN - рекурентних згорткових мереж, він дозволяє краще навчити модель. Цей шар являє собою простий повністю пов’язаний шар із рекурентними одиницями із строєм замість простих нейронів. Закриті рекурентні блоки (GRU) є покращеною версією стандартної рекурентної нейронної мережі. GRU схожий на довготривалу короткочасну пам'ять (LSTM), але з меншою кількістю параметрів. Це дійсно корисно під час прогнозування часових рядів або класифікації послідовних даних.

Шар Dropout випадково встановлює вхідні одиниці на 0 з частотою rate на кожному кроці протягом часу навчання, що допомагає запобігти перенавчанню. Входи, не встановлені на 0, збільшуються на 1/(1 - швидкість), так що сума по всіх входах не змінюється. Введено новий гіперпараметр, який визначає ймовірність вилучення вихідних даних шару або, навпаки, ймовірність збереження вихідних даних шару. Загальним значенням є ймовірність 0,3 для збереження вихідних даних кожного вузла в прихованому шарі.

Шар Dense - звичайний щільно пов’язаний шар NN. Нейрон щільного шару в моделі отримує вихідні дані від кожного нейрона свого попереднього шару, де нейрони щільного шару виконують множення матриці-вектора.

Останій шар це шар активації sigmoid. Сигмовидна активаційна функція, sigmoid(x) = 1 / (1 + exp(-x)). Застосовує функцію сигмовидної активації. Для малих значень (<-5) sigmoidповертає значення, близьке до нуля, а для великих значень (>5) результат функції наближається до 1. Сигмовидна еквівалентна 2-елементному Softmax, де другий елемент вважається нульовим. Сигмовидна функція завжди повертає значення від 0 до 1.

Вихідний рівень , який повертає кінцевий вихід нейронної мережі. Якщо ми робимо просту двійкову класифікацію або регресію, вихідний шар повинен мати лише 1 нейрон (тобто він повертає лише 1 число).

В процесі експерименту використано 64 епохи на етапі навчання мережі. З кожним етапом епохи точність моделі зростала. Чим ближче до завершення тим більше точність.

Останім шагом було налаштування гіперпараметрів, це один з надважливих етапів навчання моделі. Різні гіперпараметри можуть бути розглядаємими, наприклад підвибірки,повнозв'язні шари, епохи, додавання згортальних шарів, шари з максимізацією пулінгу. Після додавання повнозв'язного шару з багатьма нейронами з функцією активації ReLU і коефіцієнтом дропаутів, 0,3, ми виявили, що точність спала. Також був протестований інший варіант налаштування гіперпараметрів, який починається з більшої кількості нейронів в першому шарі, а потім зменшується до у другому шарі. В результаті ми обрали саме цей підхід, враховуючи цю ідею організації нейронної мережі, в якій кожен шар змішує елементи з попереднього шару. Але точність знизилася коли ми зменшили вдвічі кількість параметрів. Це може бути пов'язано з тим, що вхідний сигнал низький у порівнянні з першим шаром.

Layer (type)	Output Shape	Param #
conv2d_120 (Conv2D)	(None, 3, 37, 32)	288
activation_151 (Activation)	(None, 3, 37, 32)	0
conv2d_121 (Conv2D)	(None, 3, 30, 32)	8224
activation_152 (Activation)	(None, 3, 30, 32)	0
conv2d_122 (Conv2D)	(None, 3, 23, 48)	12336
activation_153 (Activation)	(None, 3, 23, 48)	0
conv2d_123 (Conv2D)	(None, 3, 16, 48)	18480
activation_154 (Activation)	(None, 3, 16, 48)	0
conv2d_124 (Conv2D)	(None, 3, 9, 64)	24640
activation_155 (Activation)	(None, 3, 9, 64)	0
reshape_36 (Reshape)	(None, 27, 64)	0
gru1 (GRU)	(None, 27, 32)	9312
gru2 (GRU)	(None, 32)	6240
dropout_32 (Dropout)	(None, 32)	0
dense_32 (Dense)	(None, 1)	33
activation_156 (Activation)	(None, 1)	0

Total params: 79,553

Рис 2.8 розроблена архітектура нейронної мережі

2.4.3 Навчання моделі

При розробці моделі ми також використовували batch size = 64. Batch size - це розмір пакету що визначає кількість зразків, які будуть розповсюджуватися мережею. Наприклад, скажімо, що у вас є 1995 навчальних вибірок, і ми хочемо встановити batchsize рівний 100. Алгоритм бере перші 100 вибірок (від 1-го до 100-го) з навчального набору даних і навчає мережу. Далі він бере другі 100 вибірок (від 101-го до 200-го) і знову навчає мережу. Ми можемо продовжувати виконувати цю процедуру, поки не поширимо всі зразки по мережі. Проблема може виникнути з останнім набором зразків. У нашому прикладі ми використали 1995, яке не ділиться на 100 без залишку. Найпростішим рішенням є просто отримати остаточні 5 зразків і навчити мережу. Переваги використання розміру партії < кількість всіх зразків:

- Для цього потрібно менше пам’яті. Оскільки ви навчаєте мережу, використовуючи менше вибірок, загальна процедура навчання вимагає менше пам’яті. Це особливо важливо, якщо ви не можете вмістити весь набір даних у пам’ять вашої машини.
- Зазвичай мережі навчаються швидше за допомогою міні-пакетів. Це тому, що ми оновлюємо ваги після кожного поширення. У нашому прикладі ми розповсюдили 11 пакетів (10 з них мали 100 зразків, а 1 - 50 зразків), і після кожного з них ми оновили параметри нашої мережі. Якби ми використали всі вибірки під час поширення, ми б зробили лише 1 оновлення для параметра мережі.

Також ми використовуємо 64 епохи. Епоха - це термін, який використовується в машинному навчанні і вказує на кількість проходів усього набору навчальних даних, який завершив алгоритм машинного навчання. Набори даних зазвичай групуються в пакети (особливо коли обсяг даних дуже великий). Деякі люди використовують термін “ітерація” вільно і називають ітерацію одну партію через модель. Якщо розмір пакету - це весь навчальний набір даних, то кількість епох - це кількість ітерацій. З практичних міркувань це зазвичай не так. Багато моделей створюються з більш ніж однією

епохою. Загальне співвідношення, де розмір набору даних дорівнює d , кількість епох — e , кількість ітерацій — i, а розмір пакету — b , буде d*e = i*b . Визначення того, скільки епох повинна виконуватися модель для навчання, ґрунтується на багатьох параметрах, пов’язаних як з самими даними, так і з метою моделі, і хоча були спроби перетворити цей процес на алгоритм, часто глибоке розуміння самих даних є незамінним.

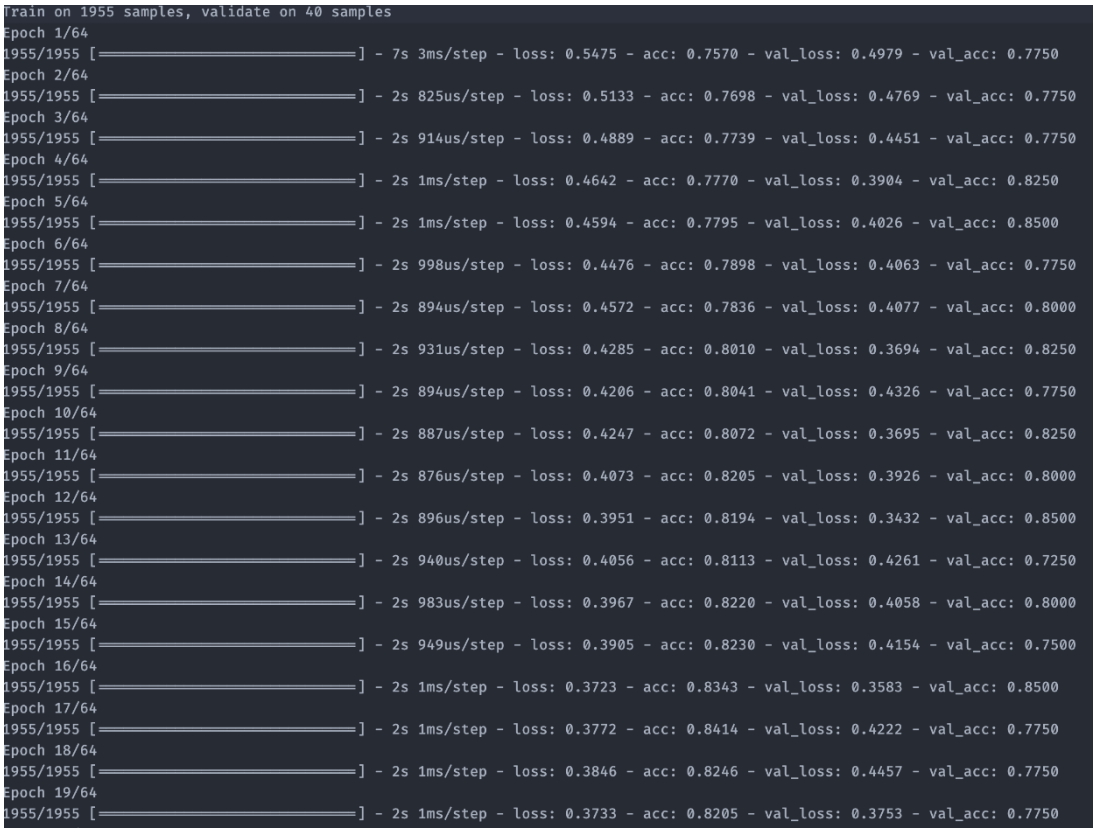


Рис 2.9 процес навчання моделі

Тестування мережі відбувалася на виборці із сорока екземплярів. У результаті ми отримали точність у 92 відсотків для визначення заїкань-повторень у аудіосигналі. Такий результат дав нам використання мережі GRCNN яка поєднує у собі переваги як CNN так і RNN, та налаштування гіперпараметрів.

Давайте порівняємо результати нашої моделі із конкурентами:

Таблиця 2.1 порівняння різних моделей

Модель	SVM	HMM	RNN	CNN	GRCNN
Точність	63%	65%	81%	86%	92%

Набір даних з великою кількістю вибірок класу без заїкання та меншою кількістю вибірок класу заїкання покращив точність. Бо в реальному світі випадків не заїкання набагато більше, ніж випадків заїкання, тому набір даних має бути двійковоасиметричним. Таким чином, ми довели співвідношення зразків мовлення із заїканням до зразків мовлення без заїкання з 50:50 до 25:75 відсотків. Це покращило результати навчання.

Найкращі результати дали аудіо сегменти довжиною в одну секунду. Випробовували різні розміри сегментів, наприклад 0,5 сек, 1 сек, 1,5 сек, 2 сек, серед яких розмір в одну секунду був найкращим для нас.

Навчена модель також враховує основну якість голосу. Під час тестування наших моделей на випадкових записах мовлення людей із заїканням, а також тих, хто штучно заїкається, ми помітили, що моделі завжди передбачали низький індекс серйозності заїкання для штучної мови із заїканням, ніж прогнозований на записах справжніх заїкаючих. Це вказує на те, що моделі також навчилися враховувати впевненість та силу голосу. Це пов’язано з тим, що плавність голосу природного заїкаючегося постійно погана (тремтливий голос), на відміну від штучно заїкання.

Система розпізнає заїкання у 20% або нижче майже на всіх аудіофайлах, в незалежності від плавності мови. Тому було вирішити не враховувати відсоток менший за 25% як аудіозапис із заїканням. Перевіримо результати роботи нейромережі на двух 40-ка секундних файлах. Один із вираженим заїканням, інший без заїкання.

Перевірка навченої моделі на 40-ка секундному аудіофайлі із заїканнями-повтореннями:

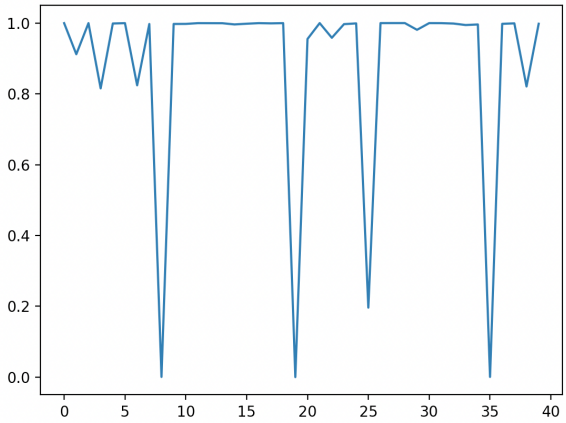


Рис 2.10 графік відношення сегментів в 1 секунду до їх коефіцієнтів заїкань

Програма виявила 88% заїкань-повторень у аудіофайлі.

Перевірка навченої моделі на 40-ка секундному аудіофайлі без повторень:

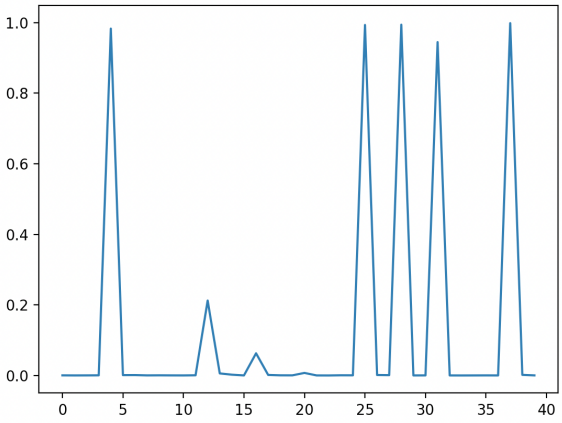


Рис 2.11 графік відношення сегментів в 1 секунду до їх коефіцієнтів заїкань

Програма виявила 12% заїкань-повторень у аудіофайлі.

Модель знаходить невеликі ознаки заїкання у всіх аудіофайлах, тому поки що доречно говорити, що в аудіозаписі є заїкання, тільки коли його відсоток перевищить 25-30%. Але навіть зараз таку модель можна використовувати наприклад у терапії для відсідковування прогрессу.

2.5 Висновок до розділу

У цьому розділу ми розглянули типи методів класифікації даних, проаналізували типи штучних нейроних мереж, так як вони є найкращим методом класифікації даних на цей час. Серед багатьох моделей ми обрали модель CNN з модифікаціями, які були взяті із моделі RNN. Взята за основу модель GRCNN, після налаштування гіперпараметрів показала гарні результати класифікації заїкань-повторень у 92 відсотки. Модель навчили на 1118 екзамплярах з характеристиками MFCC.

Розроблена модель передбачає невелике заїкання у всіх аудіофайлах, тому вирішено було не брати до уваги аудіофрагменти у яких меньше 25-30 відсотків заїкання, як аудіозаписи людей із вадами мовлення.

Навчена модель також враховує основну якість голосу. Це пов’язано з тим, що плавність голосу природного заїкаючегося постійно погана (тремтливий голос), на відміну від штучно заїкання.

Тепер отриману модель можна використати у нашому ПЗ для корекції.

РОЗДІЛ 3: ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ КОРЕКЦІЇ МОВИ

3.1 Проектування та розробка ПЗ

Основна ідея ПЗ - це корегувати аудіопотік мови із заїканням (повторенням звуків) так, щоб на виході співрозмовник не чув самого заїкання. Тому для роботи з аудіопотоком, нам треба створити систему, що працює у рамках м’якого реального часу.

Термін «система реального часу» відноситься до будь-якої системи обробки інформації з апаратними та програмними компонентами, які виконують прикладні функції в режимі реального часу і можуть реагувати на події в межах передбачуваних і конкретних часових обмежень. Системи реального часу мають ряд переваг:

- Більш точний час: Системи реального часу призначені для виконання завдань, які повинні виконуватися в чіткі терміни циклу (до мікросекунд)
- Вища передбачуваність і надійність: Оскільки системи реального часу обробляють дані у визначені, передбачувані часові рамки, виконання завдань або робочих навантажень практично гарантується, тим самим підвищуючи надійність критичних систем для бізнесу.
- Пріоритетність робочих навантажень у режимі реального часу: Коли конкретні робочі навантаження в режимі реального часу повинні бути виконані протягом встановленого терміну, щоб уникнути критичних системних збоїв, здатність визначати пріоритет одних робочих навантажень над іншими має першочергове значення. Деякі, але не всі системи реального часу мають таку можливість визначення робочого навантаження або пріоритету завдань.

Іншою важливою характеристикою систем реального часу є їх здатність одночасно виконувати робочі навантаження в режимі реального часу і не в реальному часі, щоб уникнути критичного збою системи. Важливо зрозуміти, як системи реального часу зазвичай класифікуються. Вони позначаються як м’яка система реального часу або жорстка система реального часу на основі обмежень часу.

Так як затримка у роботі нашої програми не може привести до критичних наслідків, то вона не повинна виповнюватися у режимі жорсткого реального часу. Нам треба щоб система працювала саме у режимі м’якого реального часу - це коли система продовжує функціонувати, навіть якщо вона не в змозі виконати протягом відведеного часу.

Якщо система пропустила свій термін,це не призведе до критичних наслідків. Система може продовжувати функціонувати, хоча і з небажаним зниженням продуктивності.

Для роботи із звуком на комп’ютері у режимі реального часу ми використаємо бібліотеку PyAudio - це набір абстракцій Python над PortAudio, написаного за допомогою кросплатформної бібліотеки C++, яка взаємодіє з аудіодрайверами. За допомогою PyAudio можна легко використовувати Python для відтворення та запису аудіо на різних платформах.

Для отримання аудіопотоки ми відкриємо вхідний порт для нашої програми. Отримання звуку буде мати такі параметри:

- Sample rate: 22050hz
- Channels: 1
- Format: Float32
- Chunk: 8 frames
- Frame width: 2bits

Після кожної секунди вхідного аудіо, програма буде викликати функцію callback для обробки цього аудіо. У цій функції ми будемо мати звук у виді байткоду який ми отримали за цю секунду.

Далі нам треба виявити чи є ознаки заїкання у цьому фрагменті. Для цього ми витягуємо MFCC коефіцієнти з цього сегменту, як ми це робили при формувані датасету для навчання моделі. Далі ми використовуємо нашу створену модель для оцінки заїкання у фрагменті. Це потребує невеликого проміжка часу.

Після того як ми отримали коефіцієнт вірогідності заїкання-повторення звуків у фрагменті мови, ми вирішуємо що з цим фрагментом робити далі, на основі цього коефіцієнту. Якщо коефіцієнт меньший за 0.3, то це означає, що

скоріше за всього у цьому фрагменті немає ознак заїкання, тому даний фрагмент можна пропускати до аудіовиходу. Якщо ж коефіцієнт більший за 0.3, то ми не викидаємо такий фрагмент аудіо до аудіовиходу. Якщо ми не будемо видавати звук співбесіднику, коли наша людина заїкається, то такий співбесідник може подумати, що людина його ігнорує або має проблему із зв'язком. Тому краще за все заміняти проміжки мови із заїканням на звуки роздумів. При такому сценарії, співбесідник буде думати що людина замислюється, а не заїкається. Це дозволить не тільки підвищити якість бесіди, але й зменшити заїкання у людини з часом. Бо людина не буде так хвилюватися за свою мову і як її будуть оцінювати.

Коли ми зібрали аудіо, яке хочемо передати співбесіднику, будь це аудіо із звуками роздумів або з мовою без заїкань, ми передаємо це аудіо на аудіовихід. У якості аудіовходи та аудіовиходу може слугувати безліч речей. Звичайно, найчастіший варіант використання, це використання мікрофону комп'ютера як аудіовходу, а програму для спілкування таку як Zoom чи Skype, як аудіовихід для звуку. Але можна розширити використання і на інші сценарії.

Загальний алгоритм роботи програми можна побачити нижче.

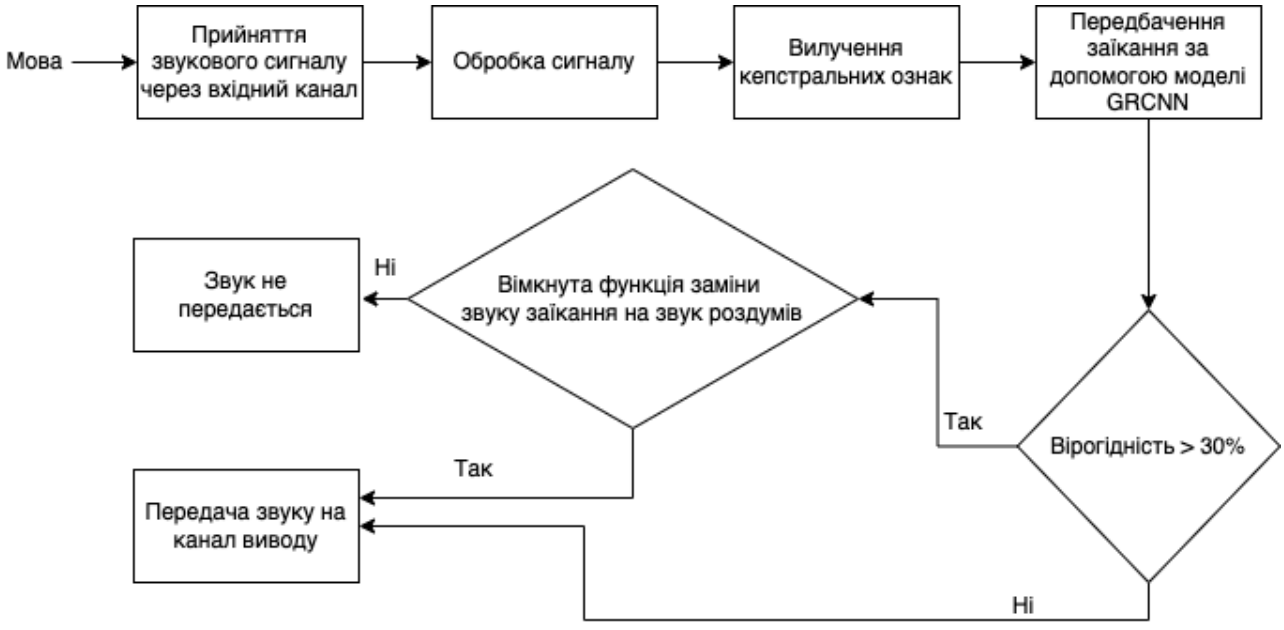


Рис. 3.1 алгоритм роботи ПЗ

3.2 Огляд отриманих результатів

У резльтаті ми отримали програмну систему, яка обробляє та виводить модифікований звук мови у реальному часі. Обробка та модицікація проходить за допомогою нейронної мережі на основі CNN.

Після розробки програми, її налагодження та тестування ми отримали деякі результати роботи програмного забезпечення. По-перше, програмна затримка при вводі та виводі звуку є близько 1.24 секунди, або 1240 мілісекунд. Така затримка більш ніж підходить для обробки звуку у реальному часі. Але треба мати на увазі, що поки звук дійде до, наприклад блютуз-навушника, пройде ще невеликий час. Тому можна сказати, що 1.24 секунди це не затримка звуку взагалі, а додаткова затримка звуку, поки він обробляється програмою.

Таку затримку, можна прискорити використовуючи більш низькорівневу мову програмування, наприклад C++, яка зможе швидше обробляти звук.

Разрублена програма аналізує звукові фрагменти за допомогою навченої моделі. Модель була навчена на 1118 екземплярах, в основному були задіяні записи дітей та підлітків. У випадку використання програмою цієї моделі для обробки голосів, було показано що модель не так гарно справляється с реальними голосами, як показала тестова вибірка. Основний пласт проблем відбувається із тестуванням програми на чоловічих голосах. С жіночими голосами ситуація набагато краще. Ми гадаємо, що причиною цього є нерепрезинтабильна вибірка, яка складається із голосів дітей і підлітків, у котрих голос набагато м’якший за дорослого чоловіка. Інші проблеми із Розпізнаванням відбуваються скоріше із за маленької кількості екземплярів для навчання. У подальшому вдосконалені програми, треба перевчити модель із більшою кількістью даних, та з більшим різновидом даних.

Загалом програма виконує свої цілі, та успішно справлюється із задачою корекції мови в аудіопотоці. Найкраще всього програма показує себе на великих відрізках мовлення із заїканням, на малих відрізках, інколи можуть бути похибки. Ми вважаємо, що причиною цього є те, що працювати та знаходити чи заїкається людина, програма може лише із сегментами в одну секунду. Тому, якщо наприклад перші 250 мілесекунд людина не заїкається, а у інші 750

мілісекунд сегменту заїкання є, то програма вирізає увесь сегмент і ми можемо вирізати сегмент із нормальним мовленням. Наприклад ми можемо вирізати початок або кінець слова.

Таблицю із результатами тестування програми можна подивитися нижче. Ми оцінили результати роботи програми, як вона знаходить та вирізає сегменти із заїканням за такою шкалою:

0 - програма модифікує аудіопотік так, що комунікація стає неможливою

1 - комунікація можлива, але програма вирізає багато сегментів мовлення без заїкання

2 - програма працює гарно та вирізає лише сегменти із заїканням

Таблиця 3.1 оцінка якості роботи програми

	Чоловік	Жінка	Дитина
Короткий сегмент заїкання	0	1	1
Середній сегмент заїкання	1	1	2
Довгий сегмент заїкання	1	2	2

Після тестування програми, ми бачимо, що найбільшу ефективність вона може проявити, коли нею користуються діти із заїканням. Це дуже важливо, бо саме діти більше всього страждають під заїкання та його психологічного аспекту. Дане ПЗ можна використовувати дитині на онлайн уроках у школі, щоб зменшити вплив заїкання на його навчання та відносини із товаришами.

3.3 Висновок до розділу

У даному розділи, ми описали процесс створення програмного забезпечення та його архітектуру. Провели експерименти над створиним ПЗ, щоб підвищити його ефективність та швидкість. В рамках розділу було оцінено ефективність роботи ПЗ, його сильні та слабкі сторони. Критично підійденно до проблем та недоліків програма, та надано шляхи для їх виправлення майбутньому.

ВИСНОВКИ

Нашою задачою була розробка адаптивної системи корекції вад мови у реальному часі, яка б вирішувала проблему спілкування через мережу інтернет людини яка має проблеми із заїканням, ненавмисним повторенням звуків. Ця проблема малодосліджена, бо людей із заїканням небагато, а спілкування через інтернет як щоденна практика не була такою росповсюдженною ще декілька років назад. Щоб розробити ПЗ, яке б могло хоча б частково вирішити цю проблему, ми розробили програмне забезпечення, яке у реальному аналізує вхідний сигнал за допомогою нейронної мережі, та модифікує вихідний звуковий сигнал, забираючи або заміняючи сегменти із заїканням.

Щоб досягти такого результати ми використали багато методів дослідження. Спочатку розглянули основні поняття та здійснили огляд звуку у цифровому представленні. Після цього порівняли характеристики звуку та їх вплив на машине навчання на кореляцію із заїканням. Потім проаналізували та порівняли методи які вирішують проблему класифікації звуку. Вибравши найкращий метод класифікації ми дослідили структуру згорткової моделі нейронної мережі, та її модифікацію із рекурентними шарами. Після аналізу та поставленні експериментів над гіперпараметрами, ми досягли точності класифікації у 93 відсотка. Це також потребувало експериментів над пропорцією та розмірністтю даних для навчання. Найкращий результат показав датасет із сегментів по одній секунді та пропорцією із 25% заїкання та 75% мови без заїкання. Модель яку ми розробили, ми порівняли із іншими відомими моделями класифікації звуку, такими як SVM, CNN, HMM та RNN. Наша модель показала найкращі результати.

Отримавши модель яка може розпізнавати заїкання у вигляді повторення звуків, ми спроектували на розробили програму, яка використовує цю модель для розпізнавання звуку у реальному часі. Шляхом експериментів ми змогли досягти оптимізації часу роботи із сегментом звуку до 1240 мілісекунд, що є прийнятним результатом для роботи із звуком.

Після проведень експериментів та тестування розробленого ПЗ у реальних умовах, було виявлено що програма ще не готова до використання у реальних умовах. Основною проблемою є доволі часте неправильне визначення заїкання у чоловіків, та робота із занадто великим сегментом звуку в одну секунду. Для вирішення цієї проблеми, потрібно перенавчити модель на більш маленьких сегментах, змінивши параметри навчання. А проблема із поганою ідентифікацією чоловіків виникає тому, що у датасеті на якому ми навчали модель, переважна більшість була голосами дітей та підлітків, які мають набагато меньш грубий голос ніж чоловіки. Вирішення цих проблем може бути подальшою задачою для опрацювання.

Але дана робота має не втрачає від цього сенс та ціннсіть. Важливою частиною роботи є розробка моделі для виявлення заїкань-повторень. Цю модель можна використовувати не тільки в нашій програмі, але й багатьма іншими способами. Наприклад, можна відслідковувати прогресс лікування заїкання у лікаря, кожен сеанс перевіряючи відсоток заїкання у мові.

Так як програма показує достатньо гарні результати для дітей, то її можна використовувати під час онлайн занять школярів у інтернеті, що у нас час є дуже актуальним. Використовуючи дане ПЗ для маскування заїкання, із часом дитина почне меньше соромитися своєї мови, більше відповідати на онлайн заняттях та більше спілкуватися із однолітками. Таке використання не тільки може покращити психологічний стан дитини із часом, а й зменшити чи навіть вилікувати заїкання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Emre Cakır - “Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection” - 2017
2. Girish. M. - “Word Repetition Analysis in Stuttered Speech Using MFCC and Dynamic Time Warping” - 2017
3. Real Time Signal Processing in Python [Електронний ресурс] – Режим доступу: https://bastibe.de/2012-11-02-real-time-signal-processing-in-python.html
4. Lin Yang - “GRCNN: Graph Recognition Convolutional Neural Network for Synthesizing Programs from Flow Charts” - 2021
5. PyAudio documentation [Електронний ресурс] – Режим доступу: https:// people.csail.mit.edu/hubert/pyaudio/docs/#pyaudio.Stream.__init__
6. G. Manjula - “Overview of analysis and classification of stuttered speech”- 2016
7. Librosa documentation [Електронний ресурс] – Режим доступу: https:// librosa.org/doc/latest/generated/librosa.feature.delta.html?highlight=feature%20delta#librosa.feature.delta
8. K.M. Ravikumar - “An Approach for Objective Assessment of Stuttered Speech Using MFCC Features” - 2009
9. FluencyBank, a shared database for the study of fluency development [Електронний ресурс] – Режим доступу: https://fluency.talkbank.org/

ДОДАТОК А

Taras Shevchenko National University of Kyiv

Adaptive system of correction of speech defects in real time

Software Architecture Document (SAD)

CONTENT OWNER: Illia Pelypenko

REVISION HISTORY

DOCUMENT NUMBER:	RELEASE/REVISION:	RELEASE/REVISION DATE:
1	V1	Wednesday, May 4th

Table of Contents

1. Documentation Roadmap

The Documentation Roadmap should be the first place a new reader of the SAD begins. But for new and returning readers, it is intended to describe how the SAD is organized so that a reader with specific interests who does not wish to read the SAD cover-to-cover can find desired information quickly and directly.

Sub-sections of Section 1 include the following.

- Section 1.1 ("Document Management and Configuration Control Information") explains revision history. This tells you if you're looking at the correct version of the SAD.
- Section 1.2 ("Purpose and Scope of the SAD") explains the purpose and scope of the SAD, and indicates what information is and is not included. This tells you if the information you're seeking is likely to be in this document.
- Section 1.3 ("How the SAD Is Organized") explains the information that is found in each section of the SAD. This tells you what section(s) in this SAD are most likely to contain the information you seek.
- Section 1.4 ("Stakeholder Representation") explains the stakeholders for which the SAD has been particularly aimed. This tells you how you might use the SAD to do your job.
- Section 1.5 ("Viewpoint Definitions") explains the *viewpoints* (as defined by IEEE Standard 1471-2000) used in this SAD. For each viewpoint defined in

Section 1.5, there is a corresponding view defined in Section 3 (“Views”). This tells you how the architectural information has been partitioned, and what views are most likely to contain the information you seek.

- Section 1.6 (“How a View is Documented”) explains the standard organization used to document architectural views in this SAD. This tells you what section within a view you should read in order to find the information you seek.

1. Document Management and Configuration Control Information

- Revision Number: 1
- Revision Release Date: 04.03.2022
- Purpose of Revision: Initial release
- Scope of Revision: Initial release

1.2 Purpose and Scope of the SAD

This SAD specifies the software architecture for mobile application for real estate search. All information regarding the software architecture may be found in this document, although much information is incorporated by reference to other documents.

What is software architecture? The software architecture for a system is the structure or structures of that system, which comprise software elements, the externally-visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible” properties refers to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. This definition provides the basic litmus test for what information is included in this SAD, and what information is relegated to downstream documentation.

Elements and relationships. The software architecture first and foremost embodies information about how the elements relate to each other. This means that architecture specifically omits certain information about elements that does not pertain to their interaction. Thus, a software architecture is an *abstraction* of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements. Elements interact with each other by means of interfaces that partition details about an element into public and private parts. Software architecture is concerned with the public side of this division, and that will be documented in this SAD accordingly. On the other hand, private details of elements—details having to do solely with internal implementation—are not architectural and will not be documented in a SAD.

Multiple structures. The definition of software architecture makes it clear that systems can and do comprise more than one structure and that no one structure holds the irrefutable claim to being the architecture. The neurologist, the orthopedist, the hematologist, and the dermatologist all take a different perspective on the structure of a human body. Ophthalmologists, cardiologists, and podiatrists concentrate on subsystems. And the kinesiologist and psychiatrist are concerned with different aspects of the entire arrangement's behavior. Although these perspectives are pictured differently and have very different properties, all are inherently related; together they describe the architecture of the human body. So it is with software. Modern systems are more than complex enough to make it difficult to grasp them all at once. Instead, we restrict our attention at any one moment to one (or a small number) of the software system's structures. To communicate meaningfully about an architecture, we must make clear which structure or structures we are discussing at the moment—which *view* we are taking of the architecture. Thus, this SAD follows the principle that documenting a software architecture is a matter of documenting the relevant views and then documenting information that applies to more than one view.

For example, all non-trivial software systems are partitioned into implementation units; these units are given specific responsibilities, and are the basis of work assignments for programming teams. This kind of element will comprise programs and data that software in other implementation units can call or access, and programs and data that are private. In large projects, the elements will almost certainly be subdivided for assignment to sub-teams. This is one kind of structure often used to describe a system. It is a very static structure, in that it focuses on the way the system's functionality is divided up and assigned to implementation teams.

Other structures are much more focused on the way the elements interact with each other at runtime to carry out the system's function. Suppose the system is to be built as a set of parallel processes. The set of processes that will exist at runtime, the programs in the various implementation units described previously that are strung together sequentially to form each process, and the synchronization relations among the processes form another kind of structure often used to describe a system.

None of these structures alone is *the* architecture, although they all convey architectural information. The architecture consists of these structures as well as many others. This example shows that since architecture can comprise more than one kind of structure, there is more than one kind of element (e.g., implementation unit and processes), more than one kind of interaction among elements (e.g., subdivision and synchronization), and even more than one context (e.g., development time versus runtime). By intention, the definition does not specify

what the architectural elements and relationships are. Is a software element an object? A process? A library? A database? A commercial product? It can be any of these things and more.

These structures will be represented in the views of the software architecture that are provided in Section 3.

Behavior. Although software architecture tends to focus on structural information, *behavior of each element is part of the software architecture* insofar as that behavior can be observed or discerned from the point of view of another element. This behavior is what allows elements to interact with each other, which is clearly part of the software architecture and will be documented in the SAD as such. Behavior is documented in the element catalog of each view.

1.3 Viewpoint Definitions

The SAD employs a stakeholder-focused, multiple view approach to architecture documentation, as required by ANSI/IEEE 1471-2000, the recommended best practice for documenting the architecture of software-intensive systems [IEEE 1471].

As described in Section 1.2, a software architecture comprises more than one software structure, each of which provides an engineering handle on different system qualities. A *view* is the specification of one or more of these structures, and documenting a software architecture, then, is a matter of documenting the relevant views and then documenting information that applies to more than one view [Clements 2002].

ANSI/IEEE 1471-2000 provides guidance for choosing the best set of views to document, by bringing stakeholder interests to bear. It prescribes defining a set of viewpoints to satisfy the stakeholder community. A viewpoint identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view. A view, then, is a viewpoint applied to a system. It is a representation of a set of software elements, their properties, and the relationships among them that conform to a defining viewpoint. Together, the chosen set of views show the entire architecture and all of its relevant properties. A SAD contains the viewpoints, relevant views, and information that applies to more than one view to give a holistic description of the system.

The remainder of Section 1.5 defines the viewpoints used in this SAD. The following table summarizes the stakeholders in this project and the viewpoints that have been included to address their concerns.

Table 1: Stakeholders and Relevant Viewpoints

Stakeholder	Viewpoint(s) that apply to that class of stakeholder's concerns
Illia Pelypenko	Student and project developer

1. "Student and project developer" Viewpoint Definition

1. Abstract

This viewpoint summarizes the views of a student developing this project and their concerns

2. Stakeholders and Their Concerns Addressed

The main concern of a student is to get the develop the best abb for real estate search.

3. Elements, Relations, Properties, and Constraints

Project consists of 3 primary components:

- Dataset development
- CNN neural network development
- Software development

The idea of work is to develop a system to help people suffering from stuttering (repetition of sounds) in online format in real time. This system must intercept input sounds from the audio input system, process the signal and detect if there is stuttering-repetition in the language. If there are such signs, the program sends a sound that would hide stuttering (such as the sound of reflection), or simply did not send the sound to the interlocutor. And if there are no signs of stuttering, the sound passes unhindered from the microphone to the output program.

4. Language(s) to Model/Represent Conforming Views

Segments to MFCC coefficients code

```
dir = "data_test"

for filename in os.listdir(dir):
    path = dir + "/" + filename


    soundFile = AudioSegment.from_wav(path)
```

```
segments = soundFile[::1000]

segmentDirPath = path + "_segments"

os.makedirs(segmentDirPath)

for i, segment in enumerate(segments):
    if len(segment) < 100:
        continue

    segmentPath = segmentDirPath + "/segment{0}.wav".format(i)

    segment.export(segmentPath, format="wav")

    # Load an audio file as a floating point time series.
    # sr - target sampling rate
    # y - audio time series
    y, sr = librosa.load(segmentPath)

    os.remove(segmentPath)

    # Compute MFCC features from the raw signal
    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
    # delta matrix of data at specified order
    delta = librosa.feature.delta(mfcc, order=2, mode='constant')
```

```
df = pd.DataFrame(delta)
```

2. Architecture Background

2.1 Problem Background

2.1.1 System Overview

The main idea of the software is to adjust the audio flow of speech with stuttering (repetition of sounds) so that the interlocutor does not hear the stuttering itself.
Therefore, to work with the audio stream, we need to create a system that works in soft real-time.

2.1.2 Goals and Context

So the goal of my work is to correct the audio flow of speech with stuttering in real time.

To achieve the goal, we must perform the following tasks:

- analysis of sound characteristics and methods of working with it

- creation of a neural network to assess the presence of signs of stuttering in speech

- collecting a dataset to study the neural network

- creation of software that would analyze and modify the audio signal in real time

2.1.3 Significant Driving Requirements

To build a neural network to recognize stuttering, we must first assemble a dataset that will study our neural network. Since the basis of our dataset is audio data, we need to understand what data we will use for learning, analyze and compare the characteristics of sound and find out which of them are most suitable for learning.

Now that we have a dataset for learning, our task at this stage is to develop a program model that would determine whether there are signs of stuttering in the language, and how big. There are many ways of machine learning to classify data, which we will consider below.

2.2 Solution Background

2.1.1 Architectural Approaches

Since a delay in our program can not lead to critical consequences, it should not be performed in hard real time. We need the

system to run in soft real time - this is when the system continues to operate, even if it is unable to run for the allotted time.

If the system misses its deadline, it will not lead to critical consequences. The system can continue to operate, albeit with an undesirable decrease in performance.

After each second of incoming audio, the program will call the callback function to process this audio. In this function we will have the sound in the form of a bytecode that we received for this second.

As a convolution layer, we used keras.layers.Conv2D - two-dimensional (working with two-dimensional data) convolution layer. This layer creates a convolution core that collapses with the input layer to create an output data tensor. If use_bias is True, an offset vector is created and added to the source data. Finally, if the activation is None, it is also applied to the outputs. Using this layer as the first layer in the model, the keyword argument input_shape (tuple of integers or None, does not include the sample axis), you can use None when the size is variable. This layer we use the parameter strides (1, 1) - a number or tuple / list of 2 integers, which determines the steps of convolution in height and width. Can be an integer to determine the same value for all spatial dimensions. Any value of step! = 1 is incompatible with the definition of any dilation_rate value! = 1.

On all convolutional layers except the first we use the padding "valid" parameter. It means no filling. A value of "same" results in a uniform fill with zeros to the left / right or up / down of the input. When padding = "same" and strides = 1, the output is the same size as the input.

After each convolution layer we use the activation layer, as the activation function we have chosen the most common ReLU activation function. This is a piecewise linear function that returns the result only if it is positive, and is mainly used for hidden layers. In addition, the output level must have activation compatible with the expected output. For example, the linear function is suitable for regression problems, while the sigmoid is often used for classification.

Uses the activation function of a rectified linear unit.

With default values, this returns the standard ReLU activation: max (x, 0), element-by-element maximum 0, and input tensor. Changing the default settings allows you to use non-zero thresholds, change the maximum activation value, and use a non-zero value that is a multiple of the input for values below the threshold.

We also tried to use the BatchNormalization layer. A layer that normalizes its input. Batch normalization applies a transformation that

maintains an average output value of about 0 and an output standard deviation close to 1. It is important that batch normalization works differently during training and output. During training (ie when using fit () or calling a layer / model with the argument training = True), the level normalizes its output using the mean and standard deviation of the current batch of input data. But the use of this layer did not give us the necessary accuracy we wanted, so it was decided not to use this layer.

The Reshape layer can be used to resize its input data without changing its data, only resizing it; data is not copied in the process. Positive numbers are used directly, setting the appropriate size of the original blob.

A feature of our neural network is the GRU layer, which we took from RNN - recurrent convolutional networks, it allows you to better teach the model. This layer is a simple fully connected layer with recurring units with a system instead of simple neurons. Closed Recurrent Blocks (GRUs) are an improved version of the standard recurrent neural network. GRU is similar to long-term short-term memory (LSTM), but with fewer parameters. This is really useful when forecasting time series or classifying sequential data.

The Dropout layer randomly sets the input units to 0 with a rate rate at each step during the training time, which helps prevent overtraining. Inputs not set to 0 are increased by 1 / (1 - speed), so that the sum of all inputs does not change. A new hyperparameter is introduced, which determines the probability of extracting the source data of the layer or, conversely, the probability of saving the original data of the layer. The total value is a probability of 0.3 to save the original data of each node in the hidden layer.

The Dense layer is a regular tightly bound NN layer. The dense layer neuron in the model receives the original data from each neuron of its previous layer, where the dense layer neurons perform multiplication of the matrix vector.

The last layer is the sigmoid activation layer

2.2.2 Analysis Results

In this section, we have considered the types of data classification methods, analyzed the types of artificial neural networks, as they are the best method of data classification at this time. Among the many models, we chose the CNN model with modifications taken from the RNN model. Based on the GRCNN model, after adjusting the hyperparameters showed good results in the classification of stuttering-repetitions in 92 percent. The model was trained on 1118 copies with MFCC characteristics.

The developed model assumes a small stuttering in all audio files, so it was decided not to take into account audio fragments with less than 25-30 percent stuttering, as audio recordings of people with speech defects.

The trained model also takes into account the basic quality of the voice. This is due to the fact that the smoothness of the voice of a natural stutterer is constantly bad (trembling voice), in contrast to artificial stuttering.

Now the obtained model can be used in our software for correction.

Conducted experiments on the created software to increase its efficiency and speed. The section assessed the effectiveness of the software, its strengths and weaknesses. The program is critical of the problems and shortcomings, and provides ways to correct them in the future.

2.2.3 Requirements Coverage

To build a neural network to recognize stuttering, we must first assemble a dataset that will study our neural network. Since the basis of our dataset is audio data, we need to understand what data we will use for learning, analyze and compare the characteristics of sound and find out which of them are most suitable for learning.

Now that we have a dataset for learning, our task at this stage is to develop a program model that would determine whether there are signs of stuttering in the language, and how big. There are many ways of machine learning to classify data, which we will consider below.

3 Solution Background

Emre Cakır - “Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection”
Girish. M. - “Word Repetition Analysis in Stuttered Speech Using MFCC and Dynamic Time Warping”
Real Time Signal Processing in Python - https://bastibe.de/2012-11-02-real-time-signal-processing-in-python.html
Lin Yang - “GRCNN: Graph Recognition Convolutional Neural Network for Synthesizing Programs from Flow Charts”
PyAudio documentation - https://people.csail.mit.edu/hubert/pyaudio/docs/#pyaudio.Stream.__init__
G. Manjula - “Overview of analysis and classification of stuttered speech”
Librosa documentation - https://librosa.org/doc/latest/generated/librosa.feature.delta.html?highlight=feature%20delta#librosa.feature.delta
K.M. Ravikumar - “An Approach for Objective Assessment of Stuttered Speech Using MFCC Features”
FluencyBank, a shared database for the study of fluency development - https://fluency.talkbank.org/

4.3 Acronym List

A / D - analog-to-digital conversion

MFCC - Mel-frequency cepstral coefficients

CNN - Convolutional neural network

GRCNN - Convolutional Neural Networks with Gated Recurrent Connections

ANN- Artificial neural network

SHN - Artificial neuron

ReLU - Rectified Linear unit

5. Figures

Figure 1 MFCC matrix

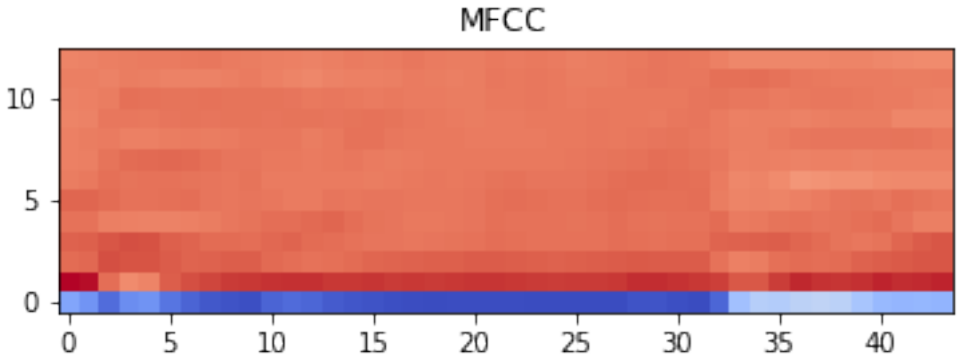


Figure 2 Neural network architecture

Layer (type)	Output Shape	Param #
conv2d_120 (Conv2D)	(None, 3, 37, 32)	288
activation_151 (Activation)	(None, 3, 37, 32)	0
conv2d_121 (Conv2D)	(None, 3, 30, 32)	8224
activation_152 (Activation)	(None, 3, 30, 32)	0
conv2d_122 (Conv2D)	(None, 3, 23, 48)	12336
activation_153 (Activation)	(None, 3, 23, 48)	0
conv2d_123 (Conv2D)	(None, 3, 16, 48)	18480
activation_154 (Activation)	(None, 3, 16, 48)	0
conv2d_124 (Conv2D)	(None, 3, 9, 64)	24640
activation_155 (Activation)	(None, 3, 9, 64)	0
reshape_36 (Reshape)	(None, 27, 64)	0
gru1 (GRU)	(None, 27, 32)	9312
gru2 (GRU)	(None, 32)	6240
dropout_32 (Dropout)	(None, 32)	0
dense_32 (Dense)	(None, 1)	33
activation_156 (Activation)	(None, 1)	0

Total params: 79,553

Figure 3 Stutter detection

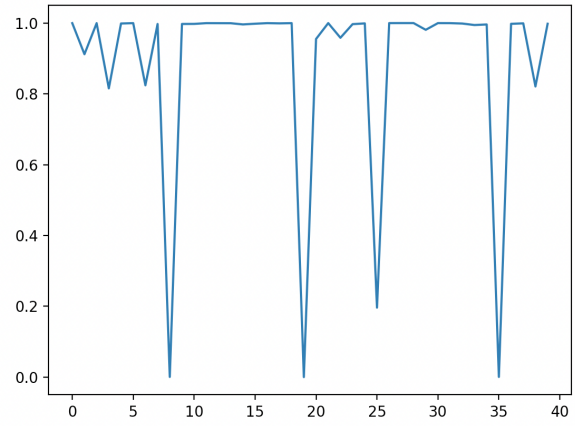
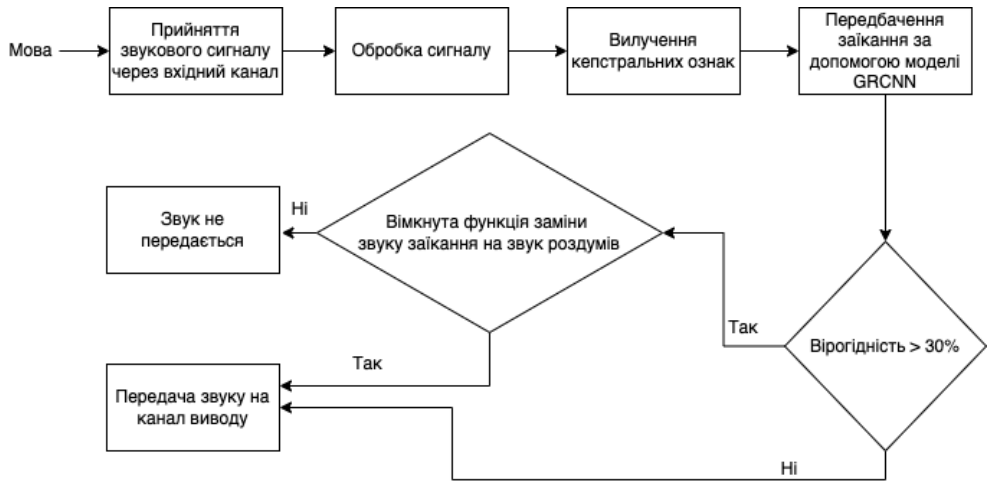


Figure 4 Software architecture



UNICHECK

Ім'я користувача:
Поляков Сергій ФІнфТехнол

ID перевірки:
1011280115

Дата перевірки:
21.05.2022 21:05:04 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
22.05.2022 08:21:30 EEST

ID користувача:
100002815

Назва документа: ПЕЛИПЕНКО-ІЛЛЯ_Порєв (1)

Кількість сторінок: 60 Кількість слів: 12185 Кількість символів: 85030 Розмір файлу: 2.64 MB ID файлу: 1011168948

10.2%
Схожість

Найбільша схожість: 2.27% з джерелом з Бібліотеки (ID файлу: 1009584587)

0.65% Джерела з Інтернету	56	Сторінка 62
10.1% Джерела з Бібліотеки	321	Сторінка 62

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	13