Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Побудова нейронної мережі для розпізнавання мови.

Виконав: студент IV курсу, групи СНс-42
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Мартинюк В.Л.
(підпис) (прізвище та ініціали)

Керівник _____ Приймак М.В.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Бойко І. В.
(підпис) (прізвище та ініціали)

Тернопіль
2022

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Мартинюк Володимир Любомирович
(прізвище, ім'я, по батькові)

1. Тема роботи Побудова нейронної мережі для розпізнавання мови.

Керівник роботи Приймак Микола Володимирович, доктор технічних наук, професор кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*16*» березня 2022 року № *4*/7-161

2. Термін подання студентом завершеної роботи 2022р.

3. Вихідні дані до роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ 1. Аналіз предметної області за темою роботи. 1.1 Історія розвитку методів розпізнавання мови. 1.2 Класифікація нейронних мереж. 1.2.1 Приховані Марковські моделі. 1.2.2 Рекурентні нейронні мережі. 1.2.3 Довга короткострокова пам'ять. 1.3 Згорткові нейронні мережі. 1.4 Connectionist Temporal Classification. 1.5 Пакетна нормалізація. 1.6 Висновок до першого розділу. Розділ 2. Проектування та навчання нейронної мережі. 2.1 Архітектура. 2.2 Інструменти. 2.2.1 Python. 2.2.2 Pandas. 2.2.3 NumPy. 2.2.4 LibROSA. 2.2.5 Tensorflow. 2.3 Навчальна вибірка. 2.4 Попередня обробка даних. 2.4.1 Семплювання. 2.4.2 Спектрограма. 2.4.3 Аугментація. 2.5 Процес навчання. 2.6 Результати. 2.7 Висновок до другого розділу. Розділ 3. Безпека життєдіяльності, основи хорони праці. Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці	Гурик О. Я., доцент кафедри МТ		

7. Дата видачі завдання 24 січня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	24.01.2022	*Виконано*
2.	Підбір джерел про нейронні мережі	04.01.2022-30.01.2022	*Виконано*
3.	Переклад та опрацювання джерел про розпізнавання	31.01.2022-06.02.2022	*Виконано*
	мови		
4.	Розроблення та навчання моделі нейронної мережі	07.02.2022-13.02.2022	*Виконано*
	розпізнавання мови		
5.	Оформлення розділу «Аналіз предметної області за	14.02.2022-06.03.2022	*Виконано*
	темою роботи»		
6.	Оформлення розділу «Проектування та навчання	07.03.2022-03.04.2022	*Виконано*
	нейронної мережі »		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	04.04.2022-17.04.2022	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	18.04.2022-01.05.2022	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	02.05.2022-15.05.2022	*Виконано*
10.	Нормоконтроль	16.05.2022-22.05.2022	*Виконано*
11.	Перевірка на плагіат	09.06.2022	*Виконано*
12.	Попередній захист кваліфікаційної роботи	10.06.2022	*Виконано*
13.	Захист кваліфікаційної роботи	23.06.2022	

Студент _____ Мартинюк В.Л.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Приймак М.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Побудова нейронної мережі для розпізнавання мови. // Кваліфікаційна робота освітнього рівня «Бакалавр» // Мартинюк Володимир Любомирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНс-42 // Тернопіль, 2022 // С. 80, рис. – 10, табл. – 2, кресл. – 0, додат. – 1, бібліогр. – 27.

Ключові слова: нейронна мережа, машинне навчання, розпізнавання мови, згорткова нейронна мережа, рекурентна нейронна мережа, марковська модель.

Мета роботи – розробка нейронної мережі розпізнавання мови.

В першому розділі кваліфікаційної роботи розгянуто сучасні дослідження, коротка історія розвитку розпізнавання мови, основна теорія нейронної мережі.

В другому розділі кваліфікаційної роботи розглянуто архітектуру нейронної мережі, обрано інструменти для реалізації нейронної мережі, проведено попередню обробку даних, описано процес навчання, приведено результати роботи.

ANNOTATION

A neural network development for speech recognition // Qualification work of Bachelor educational degree // Martyniuk Volodymyr // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, SNs-42 Group // Ternopil, 2022 // Pages – 80, figures – 10, tables – 2, sketches – 0, addendums – 1, references – 27.

Keywords: neural network, machine learning, speech recognition, convolutional neural network, recurrent neural network, markov model.

Thesis deals with neural network development for speech recognition.

The first section of the qualification work deals with modern research, a brief history of the development of speech recognition, the basic theory of the neural network.

In the second section of the qualification work the architecture of the neural network is considered, the tools for the implementation of the neural network are selected, preliminary data processing is carried out, the learning process is described, the results of the work are given.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШНМ – Штучна нейронна мережа.

ШІ – Штучний інтелект.

ПММ – Прихована марковська модель.

РНМ – Рекурентна нейронна мережа.

ЗМІСТ

ВСТУП

Актуальність теми. На цей момент завдання по розпізнаванню мови є однією із найактуальніших в сучасному світі. Мільярди людей по всьому світу використовують розпізнавання мови у своєму повсякденному житті, починаючи з голосового пошуку в мережі Інтернет, спілкування з голосовим асистентом, керування розумним будинком і закінчуючи валідацією користувача для доступу до його особистої інформації, і сервісами для людей з обмеженими можливостями. З кожним роком підходи до рішення цього завдання все більше покращують свою точність розпізнавання, впритул приближуючись в цьому показнику до людини, проте, досі не доганяючи її, але тенденція така, що подолання цього порогу не за горами.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є:

– проаналізувати сучасні підходи до вирішення проблеми;

– на основі існуючих досліджень побудувати оригінальну архітектуру;

– отримати модель розпізнавання мови за допомогою машинного навчання;

– протестувати і оцінити роботу отриманої моделі розпізнавання мови.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗА ТЕМОЮ РОБОТИ

1.1 Історія розвитку методів розпізнавання мови

Штучна нейронна мережа (ШНМ, англ. artificial neural networks, ANN) – це математичні обчислювальні системи на основі нейронних мереж мозку тварин. Такі системи вивчають завдання і поступово покращують продуктивність, розглядаючи приклади, і зазвичай не вимагають спеціального програмування для виконання завдання.

Перші серйозні досягнення для розв’язання задач по розпізнавання мови були отримані з застосуванням марковський випадкових процесів, зокрема прихованих марковських моделей. Проте, з розвитком моделей розпізнавання використання тільки ПММ не давало приросту в точності.

Наступним важливим кроком у підвищенні точності стала рекурентна нейронна мережа, яка може опрацьовувати послідовність подій, при цьому використовуючи власну пам’ять для обчислення наступних кроків. Всупереч тому, що дані мережі дійсно ефективні їм властива серйозна проблема у вигляді зникаючого градієнту. Тому, було розроблено цілу низку варіацій архітектур на основі РНМ, метою яких було розв’язання проблем оригіналу, одною із таких стала LSTM, яка вже не мала проблем із градієнтом і могла запам’ятовувати доволі довгі послідовності подій. Проте, у 2013 році дані мережі досягли порогу помилки в розпізнаванні мови в 17.7% [1].

На цю мить всі сучасні нейронні мережі мають в основі не єдину архітектуру, а композицію декількох різних структур. Так, наприклад, в 2015 році дослідники із Microsoft зуміли зменшити помилку в розпізнаванні до 8% в роботі [2] за допомогою моделі, базованій на взаємодії розгорнутої рекурентної та згорткової нейронних мереж. У 2016 році цю похибку вдалось знизити до 5.8% уже на основі згорткової, LSTM та LACE мереж, описаних в дослідженні [3]. А у 2017 році був досягнутий результат у 5.1%, ця модель уже може

враховувати контекст сказаної фрази, аби не плутати слова схожі по звучанню. Хоча і на цей момент ця модель все одно програє людині в точності розпізнаванні, вона показує приголомшливі результати.

1.2 Класифікація нейронних мереж

1.2.1 Приховані Марковські моделі

Перші серйозні досягнення в області розпізнавання мови в 80-х роках XX століття були пов’язані із прихованою марковською моделлю (ПММ). Даний підхід являв собою перехід від простих методів розпізнавання образів, які базуються на шаблонах та вимірюванні спектрального стану до статистичного методу розпізнавання мови, що привело до значного стрибка в точності.

По своїй суті ПММ – це статистична модель, в якій модельована система вважається марковським процесом з невідомим параметром. Завдання в тому, аби визначити приховані параметри із спостережуваних даних. В прихованій марковській моделі стан не відображається напряму, проте змінні, на які впливає стан, є видимими. Кожен стан має розподілення ймовірностей по можливих вихідним токенам (словам). Тому, послідовність токенів, які генеруються ПММ, дає певну інформацію про послідовність станів.

ПММ створює стохастичні моделі із відомих виразів і порівнює ймовірність того, що невідомий вираз було згенеровано кожною моделлю. При цьому використовується статистична теорія, щоб впорядкувати вектори ознак в матрицю Маркова (ланцюги), в якій зберігаються ймовірності переходів станів. Тобто, якби кожне із кодових слів представляло якийсь стан, ПММ слідувала б послідовність змін станів і будувала модель, яка містить в собі ймовірності переходу кожного стану в інший.

Приховані марковські моделі так популярні, тому що їм можна навчати автоматично, також вони прості в використанні. ПММ розглядає мовний сигнал як квазістатичний протягом коротких проміжків часу і моделює фрейми для

розпізнавання. Вона розбиває вектор ознак сигналу на декілька станів і визначає ймовірність переходу сигналу з одного стану в інший.

1.2.2 Рекурентні нейронні мережі

Найбільш поширеним алгоритмом для розпізнавання мови є рекурентні мережі (Recurrent neural network; RNN) і його покращені різновиди довга короткострокова пам’ять (Long short-term memory; LSTM) і керовані рекурентні блоки (Gated Recurrent Units; GRU).

Суть рекурентних мереж полягає у використані послідовних ланцюгів інформації. Дані мережі мають так звану “пам’ять”, яка “пам’ятає” значення попереднього елементу в ланцюгу і базуючи на цьому може робити висновок про вид наступного елементу, що особливо корисно в розпізнавані мови, адже наша мова – це послідовність слів або символів (далі ми будемо розглядати мову саме як послідовність символів). Проте, на практиці рекурентні мережі можуть враховувати лише невелику кількість попередніх елементів.

Глянувши на архітектуру рекурентної мережі (див. рисунок 1.1), легко можна зрозуміти, що для кожного окремого елементу буде свій шар, тобто, наприклад, якщо слово складається з 3 букв, то і мережу розгорнеться на 3 шари.

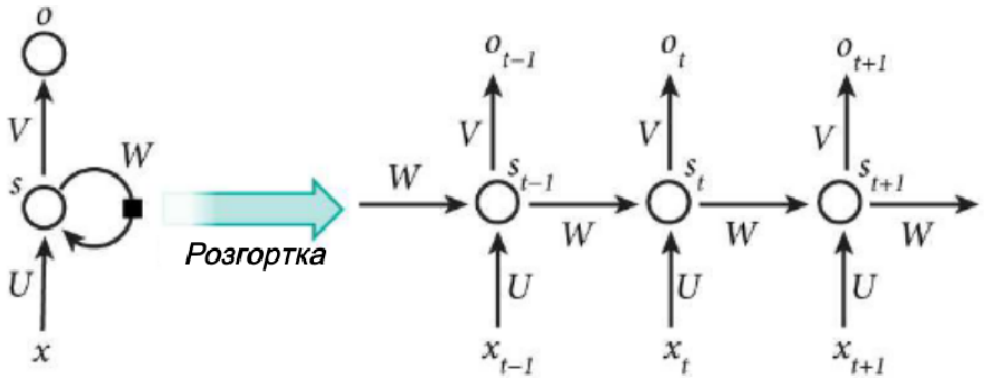

Рисунок 1.1 – Архітектура рекурентної мережі

Глянемо, що відбувається на окремо взятому шарі:

1. На кожному часовому кроці t на вхід подається вектор x_t, який відповідає символу в слові.

2. S_t – це прихований стан на кроці t. По суті, s_t можна назвати тою самою “пам’яттю” шару. Вона обчислюється з допомогою попереднього прихованого стану і входу на текучому кроці.

3. O_t – вихід на кроці t.

Проте, рекурентні мережі володіють доволі великою проблемою зникаючого градієнту. Алгоритм градієнтного спуску знаходить глобальний мінімум функції вартості, що є оптимальним налаштуванням для мережі. Також, ми обчислюємо помилку і розповсюджуємо її назад мережею, аби обновити ваги. Виходить, що мережа порівнює нашу обчислену функцію вартості з бажаним виходом.

Обчислена помилка буде розповсюджуватися не тільки на попередній шар, але і на всі попередні тимчасові кроки, а оскільки на початку ми виставляли ваги близькі до нуля, то з часом градієнт стане все меншим із кожною операцією множення.

1.2.3 Довга короткострокова пам’ять

LSTM (довга короткочасна пам’ять) мережа виправляє проблеми рекурентних мереж. В ній немає фундаментальної різниці з рекурентними мережами, проте змінено обчислення виходів і прихованих станів, які використовують вхідні значення.

Архітектуру даної мережі зображено на рисунку 1.2.

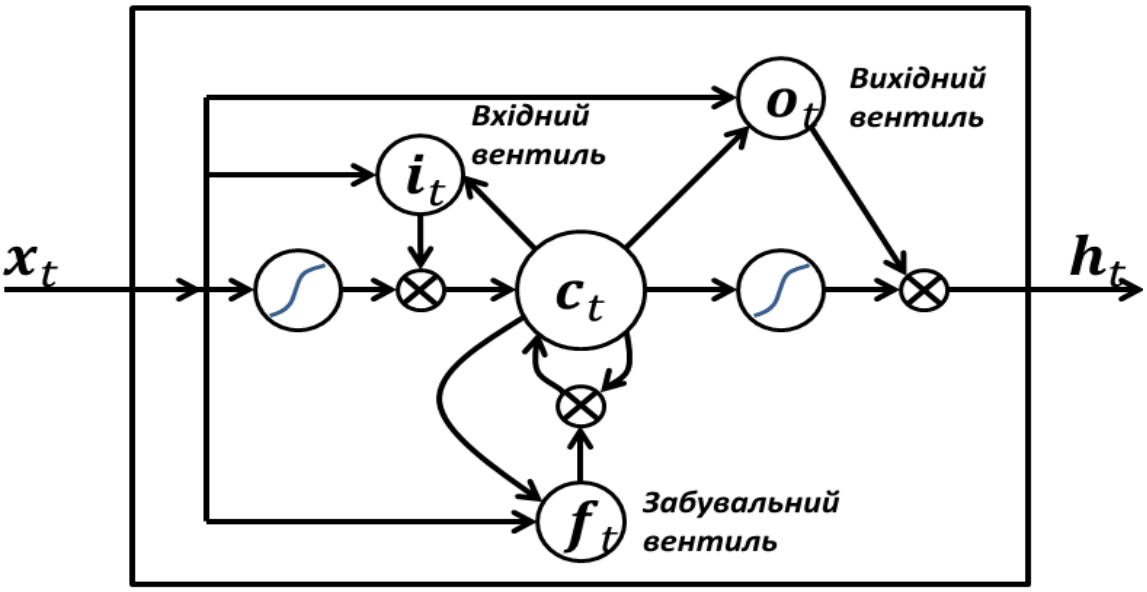

Рисунок 1.2 – Архітектура LSTM

Дані мережі пристосовані для навчання довгочасними залежностями, до того ж вона не має проблеми зникаючого градієнту.

1.3 Згорткові нейроні мережі

Зазвичай, згорткові мережі асоціюються із комп'ютерним зором, оскільки, двовимірну згортку можна побачити на високих позиціях в рейтингах змагань по розпізнаванню образів. Проте, одновимірна згортка може конкурувати із рекурентними мережами в завданнях обробки послідовностей, при цьому маючи меншу обчислювальну вартість (див. рисунок 1.3) [4].

По аналогії з двовимірною згорткою, яка отримує двовимірні ділянки і застосовує ідентичне перетворення до кожної ділянки, так і одновимірна згортка отримує локальні підпослідовності з послідовностей.

Рисунок 1.3 – Одновимірна згортка

Дані шари можуть розпізнавати локальні ознаки в послідовності. Оскільки одне і теж вхідне перетворення застосовується до кожної послідовності, ознаки, вивчені в одній позиції, можуть бути розпізнані в іншій позиції.

Нехай $x_i \in R$– k-вимірний вектор, який є елементом послідовності. Тоді послідовність довжини n представляється як:

$$x = x_1 \oplus x_2 \oplus ... \oplus x_n \tag{1.1}$$

Тепер, кожна ознака c_i буде обчислюватись за формулою [5]:

$$c_i = f(wx_{i:i+h-1} + b), \tag{1.2}$$

де $x_{i:i+h-1}$ – об'єднання елементів від i до $i+h-1, w \in R^{hk}$– фільтр згортки, який застосовується до вікна з h елементів, отримуючи нові ознаки, b – відхилення, f – нелінійна функція. На виході ми отримуємо вектор із ознак, обчислених по кожному вікні, який називається картою ознак [6]:

$$c = [c_1, c_2, ..., c_{n-h+1}] \tag{1.3}$$

Зазвичай фільтр w ініціалізується випадково. Не можна встановлювати всі ваги в нуль. Якщо всі нейрони будуть мати однакову вагу, то вони будуть створювати одні й ті самі результати та мережа не буде вивчати нові ознаки. Всі ваги під час оновлення будуть залишатись однаковими. Проте, якщо на початку взяти випадковий розподіл, то з високою ймовірністю всі ваги нейронів будуть різними і мережа зможе вивчати різні ознаки.

При цьому, ми будемо використовувати згортку із розширеним каузальним фільтром. Розширення фільтру означає розширення його розміру, заповнюючи порожні позиції нулями. На практиці такий фільтр не створюється, замість цього, ваги порівнюються із видаленими елементами у вхідному векторі. Відстань визначається коефіцієнтом розширення. При цьому, фільтр називається каузальним, якщо вихід фільтру не залежить від наступних входів.

1.4 Connectionist Temporal Classification

На виході нашої мережі ми отримуємо оцінки вимови символу в момент часу *t,* цей вихід являє собою матрицю (див. Рисунок 1.4). У цей момент виникають дві задачі, які ми хочемо вирішити:

– Розрахувати значення втрат під час навчання.

– Декодувати матрицю, аби отримати текст, який розпізнала наша модель.

Для розв'язання цих завдань ми будемо використовувати функцію втрат CTC (Connectionist Temporal Classification).

Нам необхідно повідомити функції втрат CTC лише текст, який являє собою розшрифрування нашого звукового файлу, нема потреби турбуватися про тривалість звуків для кожної букви, щобільше, наступна обробка розпізнаного тексту не потрібна [7].

	1	2	3	4	5	6	7	8	9	10
A	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
B	**0.20**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C	0.0	0.0	0.0	0.0	**0.53**	0.0	0.0	0.0	0.0	0.0
D	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**0.69**
...	...	...	...	...	...	...	...	...	...	...
Z	**0.46**	0.0	0.0	0.0	0.0	0.0	0.0	**0.78**	0.0	0.0
'	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
space	0.0	0.0	0.0	0.0	0.0	**0.92**	0.0	0.0	0.0	0.0

Рисунок 1.4 – Матриця оцінок ймовірностей появи символу в момент часу t

Для розв'язання першої проблеми розробники CTC визначили метрику кількості невірних міток (label error rate) для часового класифікатора h як середня нормалізована відстань Левенштейна між отриманим результатом і справжнім текстом [8]:

$$LER(h,S') = \frac{1}{|S'|}\sum_{(x,z)\in S'} \frac{ED(h(x),z)}{|z|}, \quad (1.4)$$

де $ED(p,q)$ – відстань Левенштейна між послідовностями p і q, S' – тестовий набір даних, що складається з пар (x, z). Цю метрику ми і будемо оптимізувати при навчанні моделі.

Другу проблему вирішує метод декодування, який базується на алгоритмі променевого пошуку (beam search), також цей пошук називається префіксним, адже на часовому кроці t вибирається найбільш відповідний префікс.

Нехай $P_b(t,l)$ – ймовірність того, що префікс *l*, на певному часовому кроці *t* походить від одного або декількох шляхів, які закінчуються порожнім символом, $P_{nb}(t,l)$ – є аналогом порожньої ймовірності, тому враховує всі основні шляхи, які закінчуються не порожнім символом, $P_{ctc}(t,c)$ – ймовірність появи послідовності слів *x* в мові за оцінкою мовної моделі [9].

Крок 1. Ми починаємо з порожніх префіксів. Досить просто зрозуміти, що порожній префікс за визначенням не зможе з'явитися з шляху, який проходить через не порожні символи. Список *R* містить всі префікси, які ми намагатимемося розширити на заданому часовому кроці, і буде оновлений, аби містити *k* найбільш імовірних префіксів в кінці кожної ітерації.

Крок 2. Для кожного часового кроку ми оцінюємо кожен префікс *R* і намагаємось розширити його кожним символом в нашому алфавіті.

Крок 3. Якщо *c* – пустий символ, то $P_b(t,l)$ обчислюється за формулою:

$$P_b(t,l) = P_b(t,l) + P_{ctc}(t,c)\big(P_b(t-1,l) + P_{nb}(t-1,l)\big). \qquad (1.5)$$

Крок 4. В іншому випадку *c* з алфавіту, тому розглядаємо префікс $l_1 = l + c$ і тоді виникає 3 можливі ситуації:

c рівне останньому символу в послідовності *l*, тому збільшуємо ймовірність того, що попереднім символом в шляху був символ пропуску і ймовірність того, що попереднім символом був символ *c*.

c рівний пробілу, або ми знаходимось на останньому кроці *t*:

$$P_{nb} = P_{nb} + (P_{ctc}(t,c)P_{nb}(t-1,l) + P_b(t-1,l))P_{lm}(l_1) \qquad (1.6)$$

c з алфавіту, виключаючи символ пробілу. Тому, збільшуємо ймовірність префіксу l_1:

$$P_{nb} = P_{nb} + (P_{ctc}(t,c)P_{nb}(t-1,l) + P_b(t-1,l)) \qquad (1.7)$$

Крок 5. Виявляється, тільки тому що префікс був відкинутий на попередньому кроці, він все ще може бути корисним для нас.

Крок 6. Після обчислення ймовірності префіксів на кроці *t*, ми сортуємо префікси по зменшенню $P_b(t,l) + P_{nb}(t,l)$. Після цього ми вибираємо k найбільш ймовірних префіксів.

Найбільша ширина променя (k – ширина променя) приводить до кращої продуктивності моделі, оскільки багато послідовності-кандидати збільшують ймовірність кращої відповідності цільовій послідовності. Це збільшення продуктивності приводить до зниження швидкості декодування. В цій роботі ширина променя була вибрана рівною 100, як було встановлено за замовчуванням у використаній реалізації.

Процес пошуку може бути зупинений для кожного кандидату окремо або шляхом досягнення максимальної довжини, або шляхом отримання маркера кінця послідовності, або шляхом досягнення порогової ймовірності.

1.5 Пакетна нормалізація

Навчання глибоких нейронних мереж ускладнюється тим фактом, що розподілення вхідних даних кожного шару змінюється під час навчання у міру змінення параметрів попередніх шарів. Це сповільнює навчання, вимагаючи зниження коефіцієнту швидкості навчання, а також акуратного налаштування параметрів, і робить доволі складним навчання моделі, перенасичених нелінійними залежностями. Це явище називають внутрішнім коваріантним зсувом (internal covariate shift). Іншими словами, коваріантним зсувом називають таку ситуацію під час навчання мережі, коли розподілення ознак в тестовій і навчальній вибірці мають різні параметри, такі як математичне сподівання, дисперсію тощо.

Очевидним розв’язанням цієї проблеми є перемішування вихідних даних перед формуванням пакету. Проте, таке рішення не підходить для прихованих

шарів, оскільки розподілення вхідних даних кожного шару змінюється під час навчання у міру змінення параметрів попередніх шарів.

Іншим методом розв'язання проблем є метод пакетної нормалізації, він дозволяє використовувати високий коефіцієнт швидкості навчання, а також бути менш уважним при налаштуванні параметрів. При цьому він діє як регулятор, усуваючи необхідність в Dropout.

Для кожного шару, в якому на вхід подається вектор $x = (x^{(1)},x^{(2)},...,x^{(n)})$ буде нормалізуватись кожна компонента [10]:

$$x^k = \frac{x^k - E[x^{(k)}]}{\sqrt{D[x^{(k)}]}}, \tag{1.8}$$

де математичне сподівання E і дисперсія D обчислюються по всьому набору даних. Необхідно звернути увагу на те, що ця нормалізація може змінити представлення шару. Для розв'язання цієї проблеми потрібно ввести пару параметрів $\gamma^{(k)}$ і $\beta^{(k)}$ для кожної компоненти $\boldsymbol{x^{(k)}}$, які стискають і зсувають нормалізоване значення [11]:

$$y^{(k)} = \gamma^{(k)}x^{(k)} + \beta^{(k)}. \tag{1.9}$$

Дані параметри навчаються разом із вихідними параметрами моделі. Встановивши $\gamma^{(k)} = \sqrt{D[x^{(k)}]}$ і $\beta^{(k)} = E[x^{(k)}]$ ми відновимо оригінальні активації, якщо буде необхідно.

1.6 Висновок до першого розділу

В першому розділі кваліфікаційної роботи ми розглянули сучасні дослідження, коротку історію розвитку розпізнавання мови, основну теорію нейронної мережі.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА НАВЧАННЯ НЕЙРОННОЇ МЕРЕЖІ

2.1 Архітектура

Як вже було сказано, рекурентні нейронні мережі і їх покращена версія – LSTM мережа є найбільш популярними архітектурами для розпізнавання мови, які обробляють послідовність і мають можливість враховувати величезний контекст. В залежності від завдання ми обмежуємо контекст лише минулим – вивчанням причинно-наслідкових зв’язків в даних. При цьому, ми можемо враховувати як минуле, так і майбутнє. Сучасним рішенням цієї задачі є двонапрямні рекурентні шари. Їх один прохід вивчає відношення зліва-направо, а іншої – справа-наліво. Результати потім об’єднуються [12].

Проте, ми можемо включити такий двосторонній контекст і в одновимірні згортки, проблема складається в тому, щоб вхідні дані залежали від великого контексту, необхідно використовувати фільтри все більшого розміру, а це тягне за собою великі потреби пам’яті.

Ми дуже обмежені в обчислювальних ресурсах, тому необхідно максимально скоротити використання пам’яті. Вище ми вже розглянули згортку із розширеним каузальним фільтром, які є набагато менш обчислювально витратні за рахунок своєї структури, адже ми зберігаємо в пам’яті не весь великий фільтр, а всього лиш розширену версію доволі невеликих фільтрів, в яких порожні місця заповнені нулями.

Зазвичай, за згортковими шарами йде функція активації ReLU і вона доволі непогано підходить для парадигми згорткової мережі, тоді як рекурентні шари використовують підходи забування і згадування, які реалізовані в фільтрі забування.

Було б непогано використовувати цю здібність рекурентних мереж в нашій роботі. Для цього ми будемо використовувати принцип активаційних вентилів (gated activation) [13].

Ідея є доволі простою: ми будемо пропускати вхідні дані через одновимірну згортку і потім використовувати функції *tanh* і σ, а результатом буде поелементне множення. При цьому, ми підемо ще далі і будемо використовувати функцію *tanh* ще раз в кінці.

На рисунку 2.1 представлена архітектура мережі в загальному вигляді, на рисунку 2.2 структура стеку залишкових блоків і залишкового блоку, які ми розглядали вище.

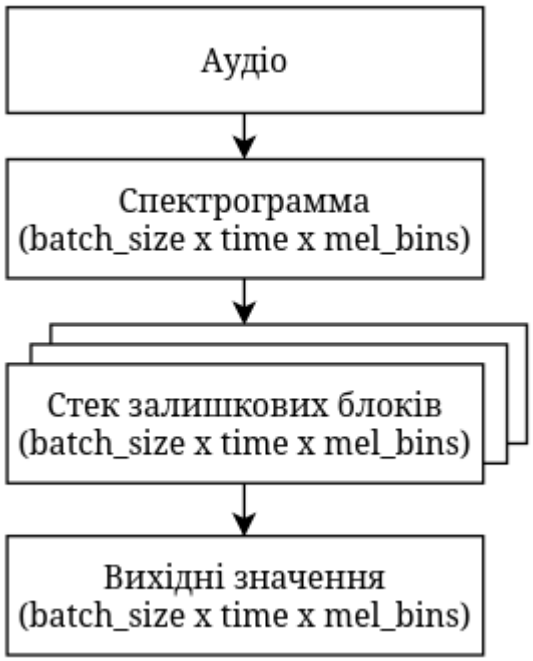

Рисунок 2.1 – Архітектура мережі в загальному вигляді

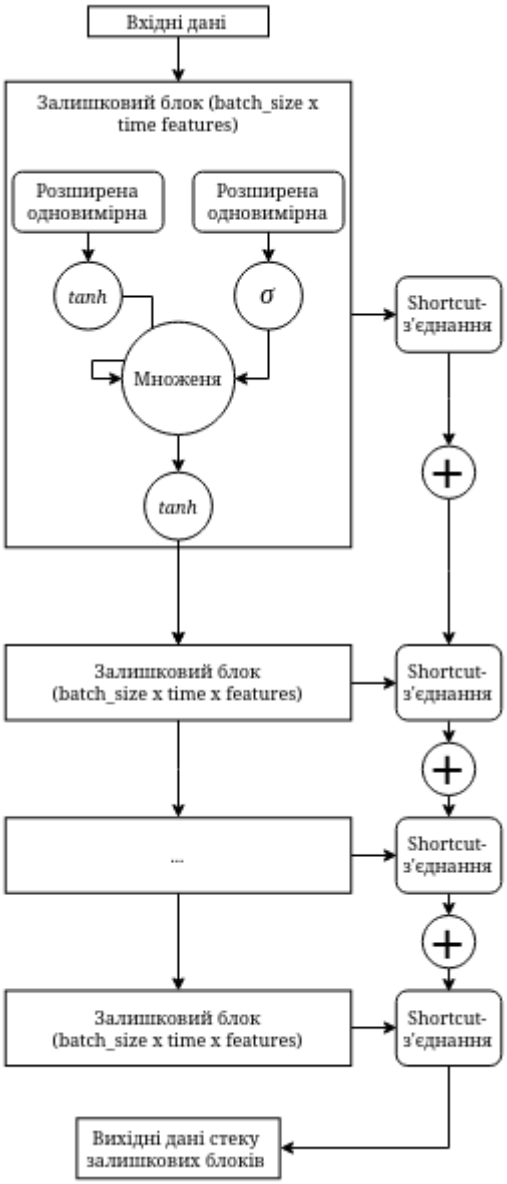

Рисунок 2.2 – Структура стеку залишкових блоків і його блоку

На основі цих рисунків і будується програмна структура нейронної мережі.

2.2 Інструменти

2.2.1 Python

Для реалізації роботи було обрано мову програмування Python, адже ця мова має велику кількість сторонніх бібліотек для розробки нейронних мереж і

аналізу даних. Вона є передовою мовою в галузі машинного навчання, при цьому надає простий синтаксис для роботи [14].

2.2.2 Pandas

Pandas – програмна бібліотека на мові Python для обробки і аналізу даних. З її допомогою відбувалась обробка даних звукових файлів і відбувався аналіз інформації про набір даних [15].

2.2.3 NumPy

NumPy – бібліотека з відкритим вихідним кодом для мови програмування Python. Можливості:

– підтримка багатовимірних масивів (включаючи матриці);

– підтримка високорівневих математичних функцій, призначених для роботи із багатовимірними масивами.

Дана бібліотека застосовувалась для попередньої обробки звукових файлів, зокрема для аугментації, а також був використаний контейнер npy, який дозволяв завантажувати і обробляти дані в декілька разів швидше, ніж стандартні csv та mp3, внаслідок чого було прискорено навчання моделі [16].

2.2.4 LibROSA

LibROSA – бібліотека для мови програмування Python, яка призначена для обробки і аналізу аудіофайлів. Головною відмінністю є те, що присутня можливість працювати із форматом mp3, що було необхідним для цієї роботи [17].

Всі аудіозаписи були завантажені при використанні функції librosa.core.load.

2.2.5 Tensorflow

Tensorflow відкрита програмна бібліотека для машинного навчання, яка розроблена компанією Google для вирішення завдань побудови і тренування нейронної мережі з метою автоматичного знаходження і класифікації образів, досягаючи якості людського сприйняття. Використовується як для досліджень, так і для розробки власних продуктів Google. Основний API для роботи із бібліотекою реалізований для Python, також існують реалізації для C++, Haskell, Java, Go та Swift [18].

Причиною вибору цієї бібліотеки стала можливість написання власних нейронних шарів, тонкого налаштування параметрів і наявність доступної та розгорнутої документації. Також, Tensorflow має підтримку обчислення на відеокартах, що суттєво прискорює процес навчання моделей.

Для того, аби створити і навчити свою власну мережу на TensorFlow ми виконали 4 пункти:

1. Визначили модель.
2. Визначили функцію подачі даних на вхід і генератор даних.
3. Визначили функцію моделі, функцію втрат, оцінювач.
4. Завантажили дані і використали оптимізатор для навчання моделі.

Для створення свого власного шару необхідно його описати за допомогою класу, який наслідується від класу tf.layers.Layer. Оскільки базові елементи мережі не є унікальними, то ми використовуємо їх реалізацію із цієї бібліотеки.

2.3 Навчальна вибірка

Для навчання мережі використовується відкритий набір даних Common Voice від Mozilla, який містить велику кількість аудіофайлів українською мовою в форматі mp3, загальною тривалістю 79 годин від 696 осіб [19].

Кожному аудіофайлу відповідає транскрипт у вигляді тексту, а також, приблизно в 50% записів присутня демографічна інформація, така як вік, стать, акцент.

У зв’язку із обмеженістю обчислювальних ресурсів використовується лише інформація про відповідний текст аудіофайлу.

Вся вибірка розділена на три частини: тренувальну, тестову і перевіркову.

Навчальна вибірка – вибірка, за допомогою якої ми налаштовуємо параметри нашої моделі.

Тестова вибірка – вибірка, за допомогою якої ми оцінюємо якість навченої моделі.

Перевіркова вибірка – вибірка, за допомогою якої ми вибираємо найкращу побудовану модель.

Усі вибірки незалежні, тобто не мають спільних записів. Тому, таким чином ми добиваємось незміщеної оцінки на тестових даних.

2.4 Попередня обробка даних

2.4.1 Семплювання

Для того, аби мережа могла опрацьовувати аудіофайли, які їй подали на вхід, і успішно навчалася на них, необхідно представити кожну аудіодоріжку у вигляді набору цифр.

Відомо, що звук передається як хвиля, тобто необхідно перетворити хвилю в цифри.

Не зважаючи на те, що нейронні мережі досить непогані в автоматичному вивчанні ознак, краще використовувати відомі ознаки, які несуть інформацію, необхідну для вирішення нашої проблеми.

Для більшості застосунків, зокрема, розпізнавання мови, ознаки, які нам потрібні містяться в представленні звуку за допомогою частоти.

Для цього дискредитується (семплюється від англ. sampling) звукова хвиля, тобто на висоті хвилі зіставляється точка в кожен момент часу. Зазвичай це робиться декілька десятків тисяч разів в секунду [20].

Частоту вибору точок називаються частотою дискретизації і виміряють в герцах. Найчастіше аудіофайли семплюють в 44.1 кГц, 16 кГц і 8 кГц. В цій роботі використовується частота 16 кГц.

2.4.2 Спектрограма

На цьому моменті дані уже можна передавати на вхід нейронній мережі, проте отримання ознак із них буде доволі довгим і складним. Тому, необхідно виконати певну попередню обробку для спрощення завдання. Для цього використовується спектрограма (див. рисунок 2.3), на основі якої будуть обчислені мелчастотні кепстральні коефіцієнти (MFCC).

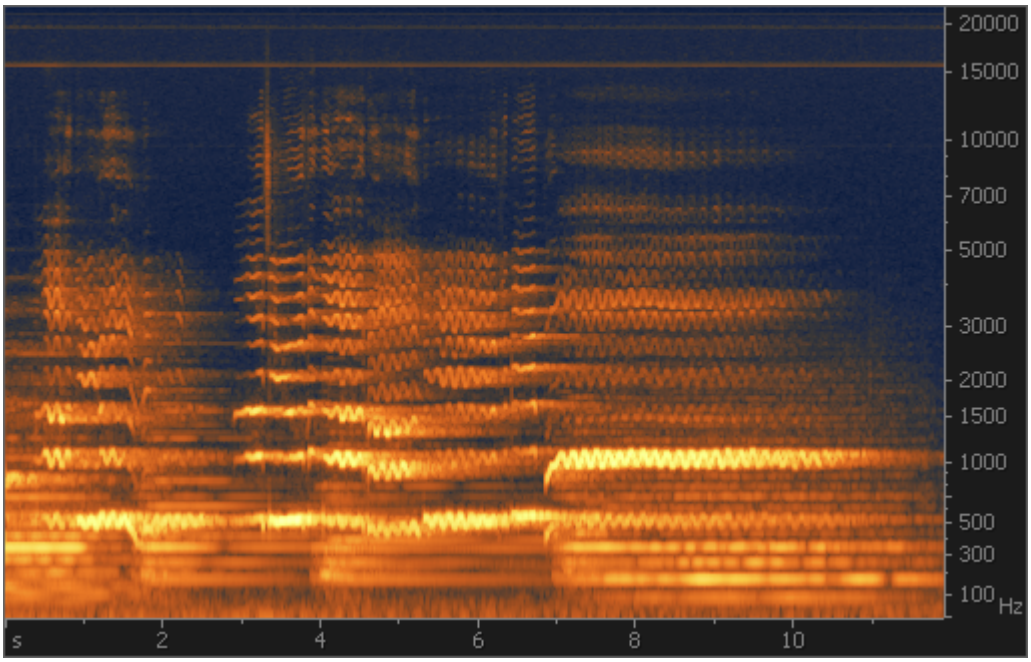

Рисунок 2.3 – Спектрограма

Спектрограма – зображення, яке показує залежність спектральної потужності сигналу від часу [21].

2.4.3 Аугментація

Аугментація даних є методом отримання додаткових тренувальних даних, шляхом змінення вихідних. Не зважаючи на те, що даний метод більше корисний в умовах серйозної обмеженості тренувальної вибірки він також допомагає зменшити дисперсію [22].

Аугментація семплів відбувається трьома способами:

1. Додавання шуму.
2. Зсув (shift).
3. Розтягнення (stretch).

Варто зауважити, що “спотворення” семплу відбувається до обчислення мелчастотних кепстральних коефіцієнтів. Для кращого ефекту додавати шум можна із зазделегідь приготованих семплів, таким чином, ми могли б додати записи слів, яких немає в оригіналі, аби модель могла відділяти мову користувача від сторонніх джерел звуку. Для спрощення ми зашумлюємо наші вихідні дані випадковими значеннями.

Аугментація була проведена із використанням функції random uniform, pa, time_stretch із бібліотек NumPy і LibROSA.

2.5 Процес навчання

Навчання моделі відбувалось за допомогою сервісу Google Colab протягом близько 15 діб із невеликими перервами. Colab дозволяє безплатно використовувати серверні обчислювальні ресурси, зокрема відеокарту NVIDIA Tesla K80, що значно спростило і пришвидшило навчання. Також, Tensorflow дозволяє зберігати стан мережі, що дає можливість після зупинки навчання запустити його із того ж моменту, на якому воно було збережене в попередній раз. Так ми не ризикуємо втратити наш прогрес через непередбачуваних проблем, зокрема втрата Інтернет з’єднання.

2.6 Результати

Не зважаючи на обмеження у часі результати даної роботи можна вважати доволі успішними. При цьому, теоретично, результати можна покращити, адже під час дослідження були знайдені мінуси в прийнятих рішеннях, з’явились нові ідеї для розвитку. Після 15 днів навчання ми отримали результати, представлені на рисунках 2.4 та 2.5. На них зображені графіки функції втрат під час навчання для тренувальної та валідаційної вибірки відповідно.

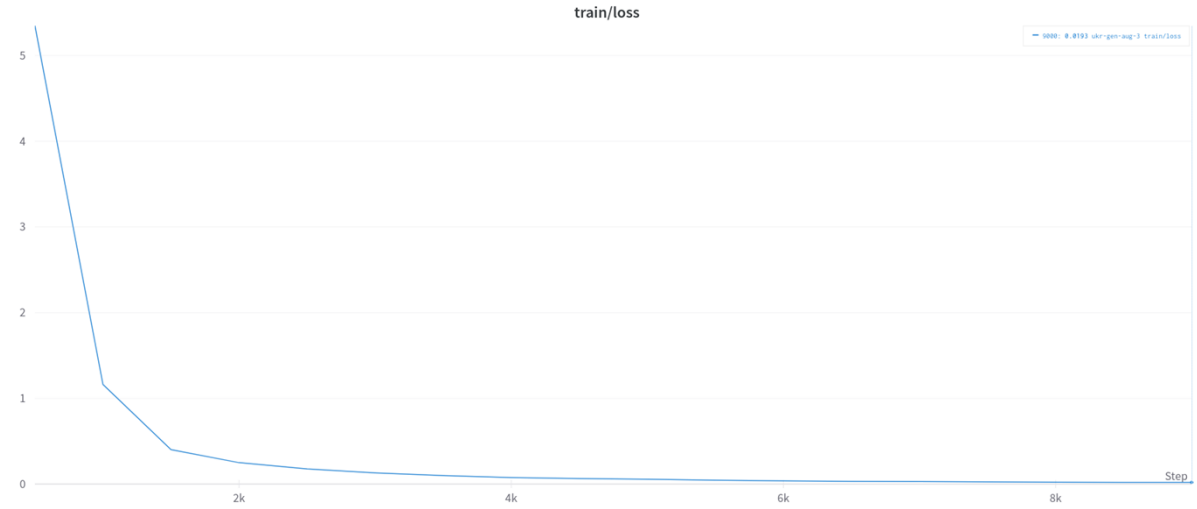

Рисунок 2.4 – Графік функції втрат тренувальної вибірки

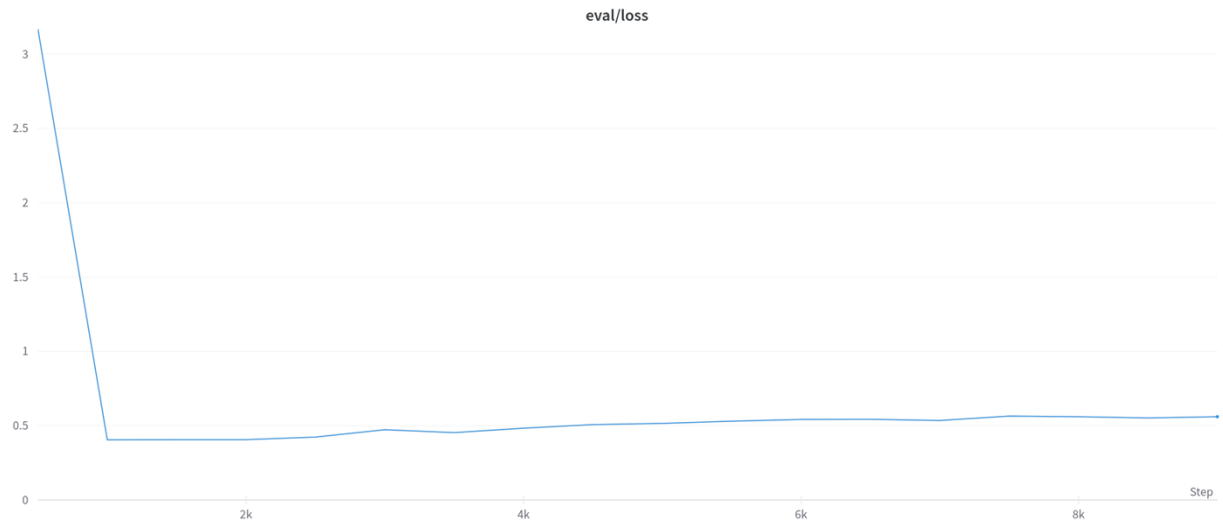

Рисунок 2.5 – Графік функції втрат валідаційної вибірки

На рисунку 2.6 можна побачити графік відсотка неправильно розпізнаних слів (WER). Таким чином, ми досягнули 77% точності на валідаційній вибірці, що є доволі непоганим результатом, враховуючи обмежений час навчання.

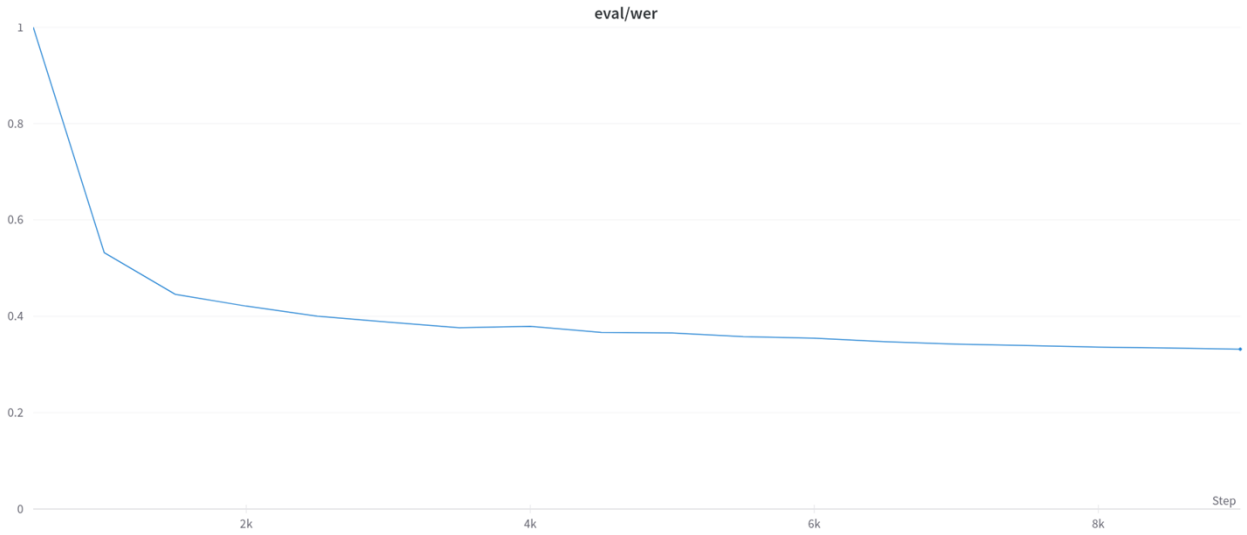

Рисунок 2.6 – Графік відсотку неправильно розпізнаних слів (WER)

В більшості випадків помилки виникають із словами паронімами. До прикладу: хутровий та хутряний. Це пов’язано із наближеним звучанням цих слів та, зазвичай, із низькою якістю аудіозапису.

2.7 Висновок до другого розділу

В другому розділі кваліфікаційної роботи ми розглянули архітектуру нейронної мережі, обрали інструменти для реалізації нейронної мережі, проведели попередню обробку даних, описали процес навчання, привели результати роботи.

РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Оцінка дії шуму і його нормування

При встановленні нормативів щодо обмеження шуму виходять, як правило, не з оптимальних (комфортних), а з припустимих умов, при яких шкідливий вплив шуму на людину або не виявляється, або є незначним. Таке гігієнічне (санітарне) нормування, встановлюється органами охорони здоров'я.

Допустимі норми виробничого шуму визначені в державному стандарті ДСН 3.3.6.037-99 "Санітарні норми виробничого шуму, ультразвуку та інфразвуку". Нормованими параметрами постійного чи переривчастого виробничих (транспортних) шумів є рівні звукового тиску в октавних смугах частот (граничні спектри, які вимірюються в дБ, позначення спектру відповідає рівню звука у смузі 1 кГц) і рівні звука, скориговані по шкалі „А" стандартного вимірювача шуму (дБА). Постійним вважається шум, рівні якого з часом змінюються не більше, ніж на 5 дБ. Непостійним вважається шум, рівні якого з часом змінюються більше ніж на 5 дБ [23]. Переривчастий шум переривається паузами тривалістю в кілька годин, хвилин чи секунд.

Непостійний шум оцінюється в еквівалентних рівнях звуку (дБА) [24]:

$$L_{\text{екв}}(A) = 10lg\left(\frac{1}{100}\sum_{i=1}^{n} t_i 10^{0,1L_i}\right), \qquad (3.1)$$

де: L_i – середній рівень класу i, дБА;

t_i – час впливу шуму класу i від загального часу контролю, %.

Розраховані значення Lекв(А) зіставляються з нормованими рівнями звука (дБА). Для дискретного та імпульсного шуму допустимі рівні знижуються на 5 дБ.

Нормування шуму в приміщеннях і на території житлових будівель. У державному стандарті ДСН 3.3.6.037-99 внесені поправки на характер шуму

(для тонального чи імпульсного – -5 дБ), час доби (для денного часу – +10 дБ), місце розташування об'єкта (для курортного району – -5 дБ) і сумарний час впливу шуму. Наявність таких поправок обумовлена впливом різних факторів на сприймання звуку людиною.

Нормування ультра- та інфразвука. Допустимі рівні звукового тиску для робочих місць ультразвукових установок визначені в державному стандарті ДСН 3.3.6.037-99 ” Санітарні норми виробничого шуму, ультразвуку та інфразвуку”. Нормовані величини мають наступні значення: при середньогеометричній частоті 1/3 октанової смуги 12,5 кГц – 75 дБ, при 16 кГц – 85 дБ і при частотах вище 20 кГц – 110 дБ. Якщо сумарний час впливу ультразвука менше чотирьох годин за зміну, то допустимі рівні збільшуються так само, як і під дією шуму.

Рівні експозиції і тривалість – професійну експозицію шуму необхідно контролювати таким чином, щоб працююча людина не піддавалася надмірній експозиції, яка визначається рівнем та тривалістю дії звука на людину. Значення припустимої комбінації рівня L і тривалості T розраховуються за формулою, хв. [25]:

$$T = 480/2^{(L-85)/3} \qquad (3.2)$$

Таблиця 3.1 – Залежність допустимого рівня звуку від тривалості його дії

***L*, дБА**	**T**		
	Год	Хв.	Сек
80	25	24	
90	2	31	
100		15	
110		1	29
120			9

Продовження таблиці 3.1

***L*, дБА**	**T**		
	Год	Хв.	Сек
130-140			<1

Відповідні значення дози експозиції шуму наводяться у таблиці 3.2, де *TWA* – зважений і усереднений за 8 годин рівень шуму [26]:

$$TWA = 10\lg\left(\frac{D}{100}\right) + 85, \quad (3.3)$$

де *D* – доза шуму.

Таблиця 3.2 – Залежність усередненого рівня звуку від дози шуму

D %	***TWA***
50	82.0
100	85.0
1000	95.0
10000	105.0
100000	115.0
1000000	125.0

Вибір ліміту експозиції залежить від визначень двох параметрів:

1. максимального прийнятного рівня порога слуху (РПС), понад який спостерігається погіршення слуху та нижче від якого вважається, що слух знаходиться у нормі;

2. частки експонованого шумом населення, яке захищається від погіршення слуху.

Нормативів щодо обмеження інфразвука поки що немає. Рекомендується використовувати як орієнтовний гранично допустимий рівень інфразвуку 95 дБ, якщо час впливу ультразвуку більше чотирьох годин. Крім гігієнічного нормування, існує нормування технічне, мета якого полягає у встановленні на основі відомих і технічно здійсненних методів гасіння шуму граничних характеристик шуму для визначеного типу машин і устаткування. У той час як санітарні норми визначають необхідні рівні шуму, технічні нормативи вказують на його можливі значення з технічної і економічно обґрунтованої точки зору.

3.2 Долікарська допомога при вивихах

Вивих – різновид травм, що характеризується як порушення конфігурації суглобових поверхонь. Це вид травми, за якої суглобовий кінець однієї кістки зміщується за межі суглобової поверхні іншої, котра разом із першою утворює суглобову пару.

Також термін «вивих» і «підвивих» застосовують до деяких уражень кришталика ока. Коли відбувається повний розрив зонулярних волокон, а кришталик в результаті цього розміщений за межами зіниці – іде мова про вивих кришталика. Як відбувається частковий розрив зонулярних волокон, а кришталик децентралізований, але частково розміщений у межах зіниці, то тоді це трактують як підвивих кришталика.

Вивихи за часом поділяються на [27]:

– свіжі – до 3-х діб після травми;

– несвіжі – від 3-х діб до 3-х тижнів;

– застарілі – більше 3-х тижнів.

Також вивихи за особливістю поділяються на:

– звичний вивих – це багаторазові вивихи без надмірних фізичних зусиль;

– невправний вивих – це, як правило, свіжий вивих, який через різні причини не вдається усунути. Причиною може бути інтерпозиція капсули, зв’язок, сухожилків;

Ознаки вивиху:

– сильний біль у кінцівці у ділянці суглоба, особливо при рухах;

– значний набряк ураженого суглоба;

– блідність шкіри навколо суглоба;

– деформація ділянки суглоба;

– відсутність активних і неможливість пасивних рухів у суглобі;

– кінцівка зафіксована у неприродньому положенні;

– зміна довжини кінцівки (видиме укорочення або подовження).

Надання домедичної допомоги при вивихах:

1. при наявності кровотечі із судин – зупинка кровотечі джгутом чи імпровізованими джгутами (закрутка, перетягання ременем), накладання стискальної пов'язки.

2. при відкритому вивиху накладання первинної (асептичної) пов'язки з метою профілактики вторинного мікробного забруднення.

3. транспортна іммобілізація: аутоіммобілізація, підручними засобами.

Задачі транспортної іммобілізації:

1. кінцівка зафіксована у неприродньому положенні;

2. зупинка кровотечі (артеріальної);

3. попередження травматичного шоку;

4. накладання стерильної пов’язки на рану;

5. проведення іммобілізації табельними або підручними засобами.

Основні принципи транспортної іммобілізації:

1. Для знерухомлення ушкодженої кінцівки необхідно іммобілізувати два суміжних суглоби, а при вивиху плечової або стегнової кістки три суглоби.

2. Моделювання шини слід проводити по здоровій кінцівці, або на тому, хто її буде накладати, тобто на медпрацівнику.

3. При іммобілізації кінцівки необхідно надати їй фізіологічне положення, а якщо це неможливо, то таке положення, при якому кінцівка менш за ве травмується.

4. При відкритих вивихах вправлення не виконують, накладають стерильну пов'язку і кінцівку фіксують в тому положенні, в якому вона знаходилась в момент ушкодження.

5. Іммобілізацію накладають поверх одягу і взуття потерпілого, між шиною і кінцівкою потерпілого кладуть м'яку ватно-марлеву підкладку.

6. При відкритих вивихах на рану слід накласти стерильну пов'язку.

7. При фіксації шини не можна закрити місце накладання джгута, щоб була можливість коригувати стан джгута.

8. Іммобілізована кінцівка перед транспортуванням в холодну пору року повинна бути обов'язково утеплена з метою профілактики відмороження.

9. Під час перекладання потерпілого з нош, ушкоджену кінцівку повинен тримати помічник.

3.3 Висновок до третього розділу

В третьому розділі кваліфікаційної роботи описані питання з безпеки життєдіяльності та охорони праці.

В першому питанні розглядається оцінка дії шуму і його нормування. Оскільки негативні наслідки шуму впливають напряму на продуктивність і здоров’я. Також шум знижує якість аудіозаписів, що негативно впливає на результат нейронної мережі.

В другому питанні розглядається долікарська допомога при вивихах, що є важливим питанням, оскільки особа, що працює із даною нейронною мережею, повинна вміти надавати домедичну допомогу у випадку травмування.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було розроблено нейрону мережу розпізнавання мови, проведено навчання та тестування мовної моделі.

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

– Подано коротку історію розвитку розпізнавання мови.

– Розглянуто сучасні дослідження розпізнавання мови, основна теорія нейронної мережі.

– Висвітлено основні переваги та недоліки різних типів нейронних мереж.

В другому розділі кваліфікаційної роботи:

– Розроблено архітектуру нейронної мережі розпізнавання мови.

– Обрано інструменти для реалізації нейронної мережі.

– Описано процес навчання.

– Протестовано та приведено результат роботи.

У розділі «Безпека життєдіяльності, основи хорони праці» було описано питання оцінки дії шуму і його нормування, долікарської допомоги при вивихах. Висвітлено негативні наслідки шуму на продуктивність і здоров’я. Розглянуто першу долікарську допомогу при вивихах, їхні різновиди, можливі сценарії та план дій.

ПЕРЕЛІК ДЖЕРЕЛ

1. Speech Recognition with Deep Recurrent Neural Networks [Електронний ресурс] // Режим доступу: https://arxiv.org/abs/1303.5778/.

2. George Saon, Hong-Kwang J. Kuo, Steven Rennie and Michael Picheny. The IBM 2020 English Conversational Telephone Speech Recognition System [Електронний ресурс] // Режим доступу: https://arxiv.org/abs/1811.02058/.

3. W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu and G. Zweig. Achieving Human Parity in Conversational Speech Recognition [Електронний ресурс] // Режим доступу: https://arxiv.org/abs/1610.05256/.

4. M. Habiba, B. A. Pearlmutter. Continuous Convolutional Neural Networks: Coupled Neural PDE and ODE [Електронний ресурс] // Режим доступу: https://arxiv.org/abs/2111.00343/.

5. J. Wu. Introduction to Convolutional Neural Networks [Електронний ресурс] // Режим доступу: https://cs.nju.edu.cn/wujx/paper/CNN.pdf/.

6. A study on data augmentation of reverberant speech for robust speech recognition [Електронний ресурс] // Режим доступу: https://ieeexplore.ieee.org/abstract/document/7953152/.

7. Listen, Attend and Spell [Електронний ресурс] // Режим доступу: https://arxiv.org/abs/1508.01211/.

8. Normalization of Transliterated Words in Code-Mixed Data Using Seq2Seq Model & Levenshtein Distance [Електронний ресурс] // Режим доступу: https://arxiv.org/abs/1805.08701/.

9. Levenshtein Transformer [Електронний ресурс] // Режим доступу: https://arxiv.org/abs/1905.11006/.

10. Exponential convergence rates for Batch Normalization: The power of length-direction decoupling in non-convex optimization [Електронний ресурс] // Режим доступу: https://arxiv.org/abs/1805.10694/.

11. A Mean Field Theory of Batch Normalization [Електронний ресурс] // Режим доступу: https://arxiv.org/abs/1902.08129/.

12. Long short-term memory [Електронний ресурс] // Режим доступу: https://en.wikipedia.org/wiki/Long_short-term_memory/.

13. Conditional Image Generation with PixelCNN Decoders [Електронний ресурс] // Режим доступу: https://arxiv.org/abs/1606.05328/.

14. Python For Artificial Intelligence [Електронний ресурс] // Режим доступу: https://wiki.python.org/moin/PythonForArtificialIntelligence/.

15. Pandas [Електронний ресурс] // Режим доступу: https://pandas.pydata.org/docs/.

16. Numpy [Електронний ресурс] // Режим доступу: https://numpy.org/doc/stable/.

17. LibROSA [Електронний ресурс] // Режим доступу: https://librosa.org/doc/latest/index.html/.

18. Tensorflow [Електронний ресурс] // Режим доступу: https://www.tensorflow.org/

19. Mozilla Common Voice Datasets [Електронний ресурс] // Режим доступу: https://commonvoice.mozilla.org/en/datasets

20. How to Listen? Rethinking Visual Sound Localization [Електронний ресурс] // Режим доступу: https://arxiv.org/abs/2204.05156/.

21. Spectograms [Електронний ресурс] // Режим доступу: https://ccrma.stanford.edu/~jos/mdft/Spectrograms.html

22. Speech Augmentation Based Unsupervised Learning for Keyword Spotting [Електронний ресурс] // Режим доступу: https://arxiv.org/abs/2205.14329/.

23. Запорожець О.І. Основи охорони праці / О. І. Запорожець, О. С. Протоєрейський, Г. М. Франчук, І. М. Боровик. – Київ: Основа, 2020. – 264 с.

24. Наказ Міністерства охорони здоров’я України “Про затвердження Державних санітарних норм допустимих рівнів шуму в приміщеннях житлових та громадських будинків і на території житлової забудови” від 22.02.2019 № 281-33252 // Міністерство юстиції України. 2019.

25. Третьяков О. В. Охорона праці 2020 / О. В. Третьяков, В. В. Зацарний, В. Л. Безсонний. – Київ: Основа, 2020. – 361 с.

26. Жидецький В. Ц. Основи охорони праці / В. Ц. Жидецький. — Львів: Афіша, 2018. — 349 с.

27. Долікарська допомога при вивихах [Електронний ресурс] // Режим доступу: https://moz.gov.ua/article/health/travmuvannja-vidi-persha-dopomoga-ta-poradi/.

ДОДАТКИ

Додаток А

Фрагмент програмного коду нейронної мережі

```
import torch

from dataclasses import dataclass, field
from typing import Any, Dict, List, Optional, Union

@dataclass
class DataCollatorCTCWithPadding:
    """
    Data collator that will dynamically pad the inputs received.
    Args:
        processor (:class:`~transformers.Wav2Vec2Processor`)
            The processor used for proccessing the data.
        padding         (:obj:`bool`,         :obj:`str`         or
    :class:`~transformers.tokenization_utils_base.PaddingStrategy`
    , `optional`, defaults to :obj:`True`):
            Select a strategy to pad the returned sequences
    (according to the model's padding side and padding index)
            among:
            * :obj:`True` or :obj:`'longest'`: Pad to the longest
    sequence in the batch (or no padding if only a single
              sequence if provided).
            *  :obj:`'max_length'`:  Pad  to  a  maximum  length
    specified with the argument :obj:`max_length` or to the
              maximum acceptable input length for the model if
    that argument is not provided.
            * :obj:`False` or :obj:`'do_not_pad'` (default): No
    padding (i.e., can output a batch with sequences of
              different lengths).
        max_length (:obj:`int`, `optional`):
            Maximum length of the ``input_values`` of the returned
    list and optionally padding length (see above).
        max_length_labels (:obj:`int`, `optional`):
            Maximum length of the ``labels`` returned list and
    optionally padding length (see above).
        pad_to_multiple_of (:obj:`int`, `optional`):
            If set will pad the sequence to a multiple of the
    provided value.
            This is especially useful to enable the use of Tensor
    Cores on NVIDIA hardware with compute capability >=
            7.5 (Volta).
    """

    processor: Wav2Vec2Processor
    padding: Union[bool, str] = True
    max_length: Optional[int] = None
    max_length_labels: Optional[int] = None
    pad_to_multiple_of: Optional[int] = None
    pad_to_multiple_of_labels: Optional[int] = None
```

```
    def __call__(self, features: List[Dict[str, Union[List[int],
    torch.Tensor]]]) -> Dict[str, torch.Tensor]:
        # split inputs and labels since they have to be of
    different lenghts and need
        # different padding methods
        input_features = [{"input_values":
    feature["input_values"]} for feature in features]
        label_features = [{"input_ids": feature["labels"]} for
    feature in features]

        batch = self.processor.pad(
            input_features,
            padding=self.padding,
            max_length=self.max_length,
            pad_to_multiple_of=self.pad_to_multiple_of,
            return_tensors="pt",
        )
        with self.processor.as_target_processor():
            labels_batch = self.processor.pad(
                label_features,
                padding=self.padding,
                max_length=self.max_length_labels,
                pad_to_multiple_of=self.pad_to_multiple_of_labels,
                return_tensors="pt",
            )

        # replace padding with -100 to ignore loss correctly
        labels =
    labels_batch["input_ids"].masked_fill(labels_batch.attention_m
    ask.ne(1), -100)

        batch["labels"] = labels

        return batch

from transformers import Wav2Vec2ForCTC

model = Wav2Vec2ForCTC.from_pretrained(
        base_model,
        attention_dropout=0.1,
        activation_dropout=0.05,
        hidden_dropout=0.1,
        feat_proj_dropout=0.008,
        mask_time_prob=0.05,
        layerdrop=0.05,
        final_dropout=0.1,
        gradient_checkpointing=True,
        ctc_loss_reduction="mean",
        ctc_zero_infinity=True,
        pad_token_id=processor.tokenizer.pad_token_id,
        vocab_size=len(processor.tokenizer)
    )
```

```
model.freeze_feature_extractor()
from transformers import Trainer
from transformers.trainer_pt_utils import LengthGroupedSampler,
DistributedLengthGroupedSampler
from torch.utils.data import DataLoader
import collections

training_args = TrainingArguments(
        output_dir=output_models_dir,
        group_by_length=True,
        per_device_train_batch_size=64,
        per_device_eval_batch_size=64,
        gradient_accumulation_steps=1,
        dataloader_num_workers=8,
        evaluation_strategy="steps",
        save_strategy="steps",
        lr_scheduler_type='cosine',
        max_steps=10000,
        num_train_epochs=1,
        fp16=True,
        save_steps=500,
        eval_steps=500,
        logging_steps=500,
        learning_rate=1e-4,
        warmup_steps=500,
        save_total_limit=2,
        report_to='none',
        run_name = 'ukr-gen-aug-3'
)
    class GroupedLengthsTrainer(Trainer):
        # length_field_name should possibly be part of
    TrainingArguments instead
        def __init__(self, length_field_name='length', *args,
    **kwargs):
            super().__init__(*args, **kwargs)
            self.length_field_name = length_field_name

        def _get_train_sampler(self) ->
    Optional[torch.utils.data.sampler.Sampler]:
            if isinstance(self.train_dataset,
    torch.utils.data.IterableDataset) or not isinstance(
                self.train_dataset, collections.abc.Sized
            ):
                return None

            # Build the sampler.
            if self.args.group_by_length:
                lengths =
    self.train_dataset[self.length_field_name] if
    self.length_field_name is not None else None
                model_input_name =
    self.tokenizer.model_input_names[0] if self.tokenizer is not
    None else None
```

```
                if self.args.world_size <= 1:
                    return LengthGroupedSampler(
                        self.train_dataset,
    self.args.train_batch_size,                    lengths=lengths,
    model_input_name=model_input_name
                    )
                else:
                    return DistributedLengthGroupedSampler(
                        self.train_dataset,
                        self.args.train_batch_size,
                        num_replicas=self.args.world_size,
                        rank=self.args.process_index,
                        lengths=lengths,
                        model_input_name=model_input_name,
                    )

            else:
                return super()._get_train_sampler()

    # Build trainer indicating the name of the field that contains
    the lengths
    trainer = GroupedLengthsTrainer(
        model=model,
        data_collator=data_collator,
        args=training_args,
        compute_metrics=compute_metrics,
        train_dataset=train_ds,
        eval_dataset=test_ds,
        tokenizer=processor.feature_extractor,
)

trainer.train()
```