

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное  
учреждение высшего образования  
**«Южно-Уральский государственный университет  
(национальный исследовательский университет)»**  
Высшая школа электроники и компьютерных наук  
Кафедра системного программирования

**Разработка приложения для идентификации человека по голосу  
на основе нейросетевых технологий**

КУРСОВАЯ РАБОТА  
по дисциплине «Программная инженерия»  
ЮУрГУ – 02.03.02.2021.308-115.КР

Нормоконтролер,  
Доцент кафедры СП, к.ф - м.н.  
\_\_\_\_\_ С.А. Иванов  
« \_\_\_\_ » \_\_\_\_\_ 2021 г.

Научный руководитель:  
Доцент кафедры СП, к.ф - м.н.  
\_\_\_\_\_ С.А. Иванов

Автор работы:  
студент группы КЭ-301  
\_\_\_\_\_ А.С. Попов

Работа защищена  
с оценкой: \_\_\_\_\_  
« \_\_\_\_ » \_\_\_\_\_ 2021 г.

Челябинск 2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное учреждение  
высшего образования

**«Южно-Уральский государственный университет  
(национальный исследовательский университет)»**

**Высшая школа электроники и компьютерных наук  
Кафедра системного программирования**

УТВЕРЖДАЮ

Зав. кафедрой СП

\_\_\_\_\_  
Л.Б. Соколинский  
10.02.2021

### **ЗАДАНИЕ**

**на выполнение курсовой работы**

по дисциплине «Программная инженерия»

студенту группы КЭ-301

Попову Андрею Сергеевичу,

обучающемуся по направлению

02.03.02 «Фундаментальная информатика и информационные технологии»

#### **1. Тема работы**

Разработка приложения для идентификации человека по голосу на основе нейросетевых технологий.

**2. Срок сдачи студентом законченной работы:** 31.05.2021 г.

#### **3. Исходные данные к работе**

- 3.1. A. Poddar, M. Sahidullah, G. Saha / Speaker verification with short utterances: a review of challenges, trends and opportunities // *IET Biometrics*, vol. 7, no. 2, pp. 91–101, Mar. 2018, doi: 10.1049/iet-bmt.2017.0065.
- 3.2. B. Desplanques, J. Thienpondt, K. Demuynck / ECAPA-TDNN: Emphasized Channel Attention, Propagation and Aggregation in TDNN Based Speaker Verification // *Proc. Annu. Conf. Int. Speech Commun. Assoc. INTERSPEECH*, vol. 2020-October, pp. 3830–3834, May 2020, doi: 10.21437/Interspeech.2020-2650.

#### **4. Перечень подлежащих разработке вопросов**

- 4.1. Выполнить анализ предметной области.

4.2. Обучить современную нейронную сеть для задачи распознавания человека по голосу.

4.3. Спроектировать приложение, предназначенное для идентификации людей по их голосу.

4.4. Реализовать приложение.

4.5. Провести тестирование нейронной сети и приложения.

**5. Дата выдачи задания:** 9 февраля 2021 г.

**Научный руководитель**  
Доцент кафедры СП, к.ф - м.н.

С.А. Иванов

**Задание принял к исполнению**

А.С. Попов

## ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ .....	6
1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ .....	8
1.1. Способы классификации человека по голосу .....	8
1.2. Проблемы, возникающие при идентификации голоса .....	9
1.3. Обзор аналогичных работ .....	10
2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ.....	12
2.1 Предобработка аудиофайла .....	12
2.2. Нейронные сети с временной задержкой .....	14
3. ПРОЕКТИРОВАНИЕ .....	18
3.1. Топология нейронной сети .....	18
3.2. Определение требований .....	19
3.3. Варианты использования системы .....	20
3.4. Архитектура системы .....	21
3.5. Диаграмма деятельности.....	23
3.6. Схема базы данных .....	25
4. РЕАЛИЗАЦИЯ.....	26
4.1. Средства реализации .....	26
4.2. Предобработка аудиофайла .....	27
4.3. Получение информации о голосе .....	27
4.4. Обучение нейронной сети .....	28
4.5. Реализация чат бота .....	30
4.6. Реализация менеджера хранилища зарегистрированных голосов .....	32
4.7. Создание докер контейнера .....	32
5. ТЕСТИРОВАНИЕ.....	34

ЗАКЛЮЧЕНИЕ .....	37
ЛИТЕРАТУРА .....	38
ПРИЛОЖЕНИЯ .....	40
ПРИЛОЖЕНИЕ А. Спецификация вариантов использования системы .....	40
ПРИЛОЖЕНИЕ В. Код класса нейронной сети ЕСАРА-TDNN .....	42

## **ВВЕДЕНИЕ**

Распознавание человека по голосу – это одна из форм биометрии, которая рассматривает голос человека как уникальную биологическую характеристику для его идентификации [1].

Технология распознавания голоса имеет историю, насчитывающую около четырех десятилетий, и использует акустические особенности речи человека. Эти акустические паттерны отражают как анатомию, так и усвоенные поведенческие паттерны индивида.

С помощью извлечения данных паттернов исследователи могут упростить задачу перевода речи в системах, обученных на определенных голосах, или его можно использовать для аутентификации и проверки личности говорящего в сферах защиты информации и в системах доступа, также данная технология может пригодиться для анализа диалогов и кластеризации голосов на аудиозаписи.

### **Актуальность**

В настоящее время системы распознавания голоса занимают всего около 4% на рынке биометрии [2], но в рамках пандемии коронавируса становятся популярнее за счет бесконтактной проверки и относительной дешевизны решений. Также достижения в нейросетевых технологиях за последние 2-3 года позволяют разработать алгоритмы голосовой биометрии, которые являются более быстрыми, точными и могут идентифицировать пользователей используя меньший отрывок речи нежели ранние версии, основанные на классических подходах.

### **Цели и задачи данной работы**

Целью данной работы является разработка веб-сервиса для идентификации человека по аудиозаписи с помощью нейронных сетей. Для выполнения поставленной цели необходимо решить следующие задачи.

1. Произвести обзор литературы и существующих приложений в данной предметной области.
2. Подготовить обучающую и тестовую выборки.

3. Спроектировать нейронную сеть для решения задачи классификации человека по аудиозаписи.
4. Провести обучение и тестирование спроектированной нейронной сети.
5. Разработать веб-сервис для идентификации человека по голосу, протестировать и отладить его работу.

### **Структура и содержание работы**

Работа состоит из введения, 5 глав, заключения, списка литературы и приложений. Объем работы составляет 43 страницы, объем библиографии – 20 источников, объем приложений – 4 страницы.

В первой главе приводятся теоретические сведения о предметной области и осуществляется обзор существующих подходов к идентификации человека по его голосу.

Во второй главе описывается алгоритм предобработки аудиофайла и приводятся теоретические сведения о нейронных сетях с временной задержкой.

В третьей главе описывается топология нейронной сети, ее архитектура и требования к приложению.

В четвертой главе описывается реализация приложения и обучение нейронной сети.

В пятой главе описывается тестирование нейронной сети и чат-бота.

В заключении представлены основные результаты работы и направления, в которых возможны дальнейшие исследования.

В приложениях содержатся листинги с реализацией архитектуры нейронной сети.

# **1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

## **1.1. Способы классификации человека по голосу**

Каждая система распознавания говорящих разделяется на две фазы: регистрация пользователя и его проверка.

Во время регистрации голос говорящего записывается, и из него, как правило, извлекается ряд характеристик для формирования дескриптора, голосового слепка, шаблона или модели.

На стадии проверки фрагмент речи говорящего или «высказывание» в зависимости от рассматриваемой системы сравнивается или с одним ранее созданным дескриптором голоса (аутентификация), или со всеми, записанными в системе (идентификация). За счет меньшего количества сравнений процесс аутентификации проходит быстрее, чем идентификация.

Все системы распознавания голоса делятся на две группы: зависящие и независящие от текста (text-dependent, text-independent) [3].

### **Тексто-зависимые**

Если текст во время записи и проверки должен быть идентичным, то речь идет о распознавании, зависящим от текста [4]. В тексто-зависимой системе фразы могут быть либо общими для всех пользователей, либо уникальными. Чаще всего такие системы используются для аутентификации.

### **Тексто-независимые**

Системы независящие от слов говорящего на практике обычно используются для идентификации людей, поскольку они практически не требуют действий со стороны говорящего. В этом случае фразы пользователя при регистрации и тестировании могут отличаться. Иногда, запись голоса может происходить и без ведома пользователя, как в случае с криминалистическими приложениями для идентификации человека.

На данный момент существует множество подходов к решению проблемы распознавания человека по записи его голоса. Для хранения и выявления уникальных характеристик конкретного голоса используют следующие технологии.



1. Частотная оценка (frequency estimation).
2. Скрытые Марковские модели.
3. Гауссовские смешанные модели (Gaussian mixture models) [5].
4. Алгоритмы сопоставления шаблонов (pattern matching algorithms).
5. Нейронные сети [6].
6. Деревья решений.
7. Векторная квантизация (vector quantization) [3].

Для сопоставления зарегистрированных дескрипторов голосов с новым представлением голоса, поступившим в систему, используют методы, основанные на подсчете различных метрик, например косинусного расстояния.

## **1.2. Проблемы, возникающие при идентификации голоса**

Существует несколько причин способных ухудшить качество определения говорящего.

Окружающий шум может попасть как на базовую запись голоса при регистрации, так и при проверке. Если система будет учитывать различные посторонние шумы при генерации цифрового представления голоса, это может повлиять на точность распознавания. Одним из возможных решений данной проблемы является использование различных алгоритмов шумоподавления.

Различные изменения характеристик голоса со временем также могут ухудшить качество работы системы. Возможным решением выступает автоматическое дообучение системы при каждом ее использовании пользователем.

Проблемы может вызвать использование разных устройств для регистрации голоса и для последующего использования. Различное качество и характеристики динамиков могут повлиять на производительность системы.

### 1.3. Обзор аналогичных работ

Во время анализа предметной области не было встречено много аналогов, использующихся в коммерческих проектах для идентификации человека по голосу.

**Speaker-Recognition** [7] – программный продукт, предоставляющий приложение, позволяющее в режиме реального времени идентифицировать говорящего.

Для извлечения первичных признаков аудиозаписи данное ПО использует мел-частотные кепстральные коэффициенты (MFCC – Mel-Frequency Cepstral Coefficients) и кодирование с линейным прогнозированием (LPC – Linear Predictive Coding).

Классификация реализуется с помощью таких технологий как Gaussian Mixture Model (GMM), Universal Background Model (UBM) и др.

**SincNet** [8] – библиотека предоставляющая глубокую нейронную сеть, способную извлекать уникальный вектор признаков говорящего для последующего анализа (классификация, кластеризация говорящих и др.). Для извлечения первичных признаков используется спектральная информация (спектрограммы), MFCC и др.

**3D Convolutional Neural Networks for Speaker Verification** [9] – метод верификации говорящего основанный на применении трехмерной свертки в нейронных сетях. В качестве входных данных для нейронной сети могут быть использованы различные представления спектральной информации аудиозаписи.

Данный подход распознает намного точнее по сравнению с методами, основанными на обычных сверточных нейронных сетях, однако не является одним из лучших подходов для решения задачи распознавания голоса. Не показывает меньше 20% ошибки на тестовых данных.

**Pytorch xvectors** [10] – библиотека, предоставляющая предобученные модели и код для их дообучения на своих данных. Используют один из лучших на сегодняшний день подходов к задаче классификации говорящего. Данный

подход основан на использовании нейронной сети с временной задержкой (TDNN – Time Delay Neural Network) и называется x-vector. На вход также подается любое представление спектральной информации (спектрограмма, MFCC и др.). Показывает менее 3% ошибки на тестовых данных набора Voxseleb-1 [11].

### **Выводы по первой главе**

В ходе анализа предметной области были найдены продукты, предоставляющие идентификацию говорящих в режиме реального времени, но основанные на устаревших моделях, показывающих плохое качество в зашумленных условиях. Также были рассмотрены библиотеки предоставляющие интерфейсы для обучения одних из лучших моделей, но в проектах отсутствовало внедрение данных моделей в реальную систему. Современные проекты используют различные методы, основанные на нейронных сетях. Однако, в зависимости от рабочих условий системы, могут применяться и статистические методы с меньшим качеством распознавания.

При извлечении первичных признаков обычно используют мел-частотные кепстральные коэффициенты или мел-спектрограммы, однако могут применяться и другие методы.

## 2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

### 2.1 Предобработка аудиофайла

Любой процесс обработки аудиосигнала включает извлечение акустических характеристик, которые далее обрабатываются другими алгоритмами с целью получить какой-либо результат для текущей задачи.

В первоначальном варианте аудио сигнал представляется в виде семплов, показывающих положение звуковой волны в данный момент времени; в задаче распознавания голоса используют частоту семплирования равную 16 кГц. Нейронные сети способны работать и с первоначальной версией аудио сигнала, однако данный подход требует много вычислительных ресурсов, поэтому в большинстве подходов аудиозапись подвергают предобработке и извлечению первичных признаков, занимающих меньше вычислительных ресурсов.

В данной работе для извлечения первичных признаков будут использоваться мел-спектрограммы. Для начала необходимо получить спектрограмму с помощью кратковременного преобразования Фурье. Для этого по аудио сигналу проходят с окном определенного размера, деля звуковую последовательность на сегменты, и выполняют быстрое преобразование Фурье для каждого сегмента. Также во время формирования спектрограммы ось частоты преобразуется в логарифмическую шкалу, а цветовое измерение преобразуется в децибелы (можно считать, что это логарифмическая шкала амплитуды). Спектрограмма для аудиофрагмента изображена на рисунке 1.

Исследования показали, что человеческое ухо воспринимает частоты не по линейной шкале, а по логарифмической и лучше улавливает изменения на более низких частотах, чем на высоких [12]. В 1937 году Стивенс, Фолькманн и Ньюманн предложили такую единицу измерения высоты тона, при которой расстояния на высоких шкалах частоты были бы различны для слушателя. Эта единица называется шкалой мел. Таким образом, чтобы получить мел-спектрограмму достаточно перевести шкалу частот в шкалу мелов. Пример мел-спектрограммы можно увидеть на рисунке 2.

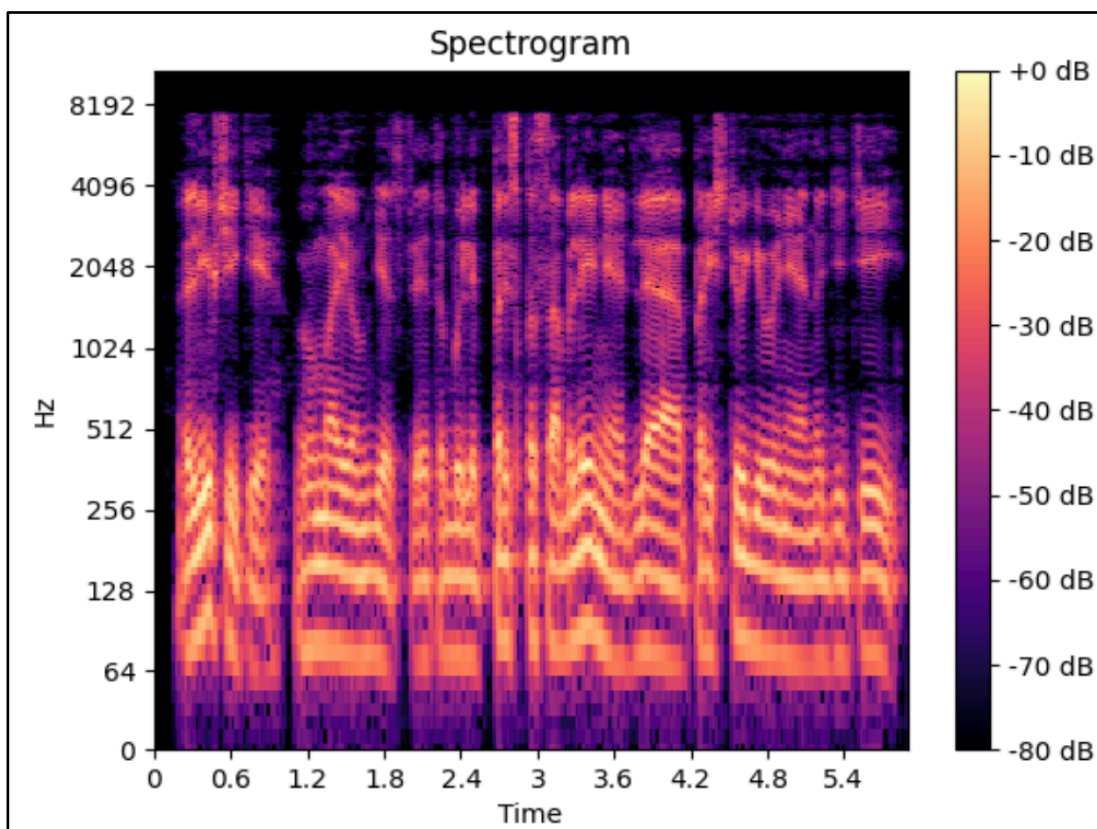


Рисунок 1 – Спектрограмма аудиофайла

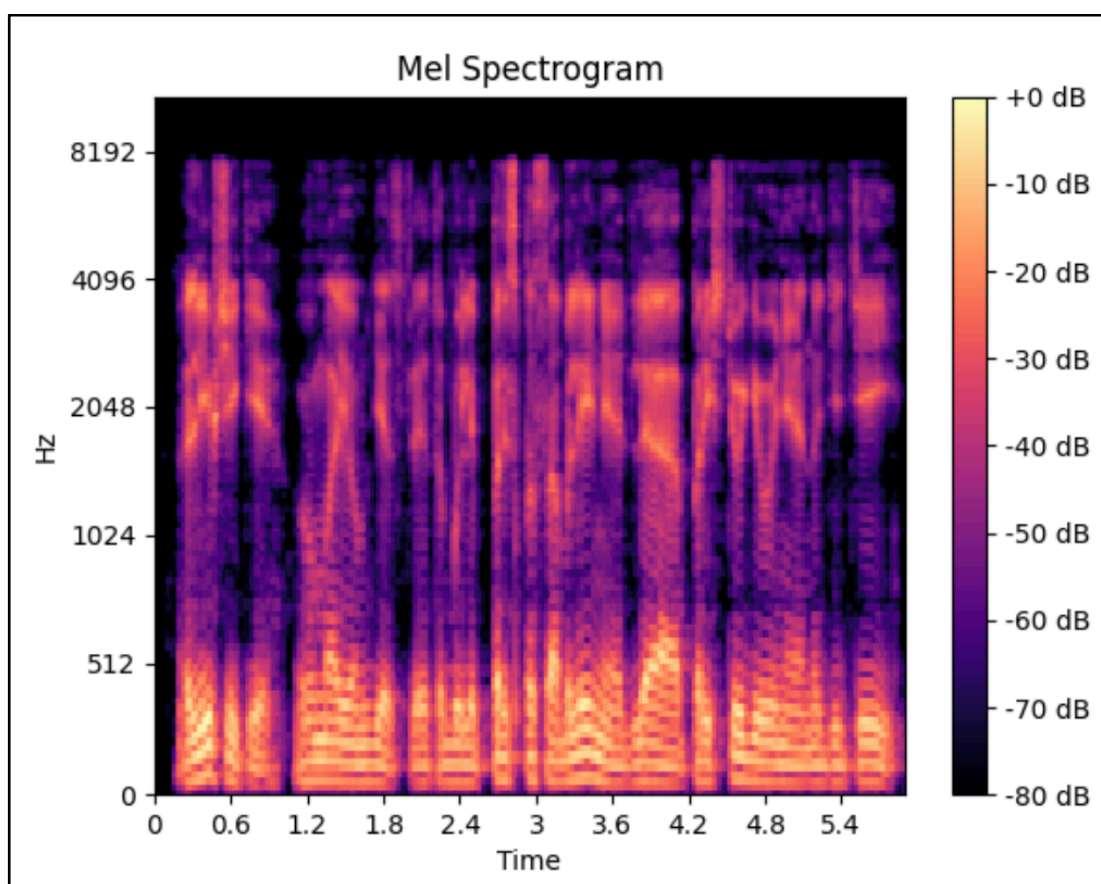


Рисунок 2 – Мел-спектрограмма аудиофайла

## 2.2. Нейронные сети с временной задержкой

Нейронная сеть с временной задержкой (TDNN – time delay neural network [13]) – это многослойная архитектура искусственной нейронной сети, целью которой является классификация паттернов независимо от их положения на временном промежутке и моделирование локального контекста на каждом слое сети. Данная архитектура была разработана 80-х годах XX века [13] и до сих пор находит свое применение в современных архитектурах. Нейронные сети с временной задержкой предназначены для обработки последовательностей, но при этом они не являются рекуррентными. Структура данной нейронной сети представлена на рисунке 3.

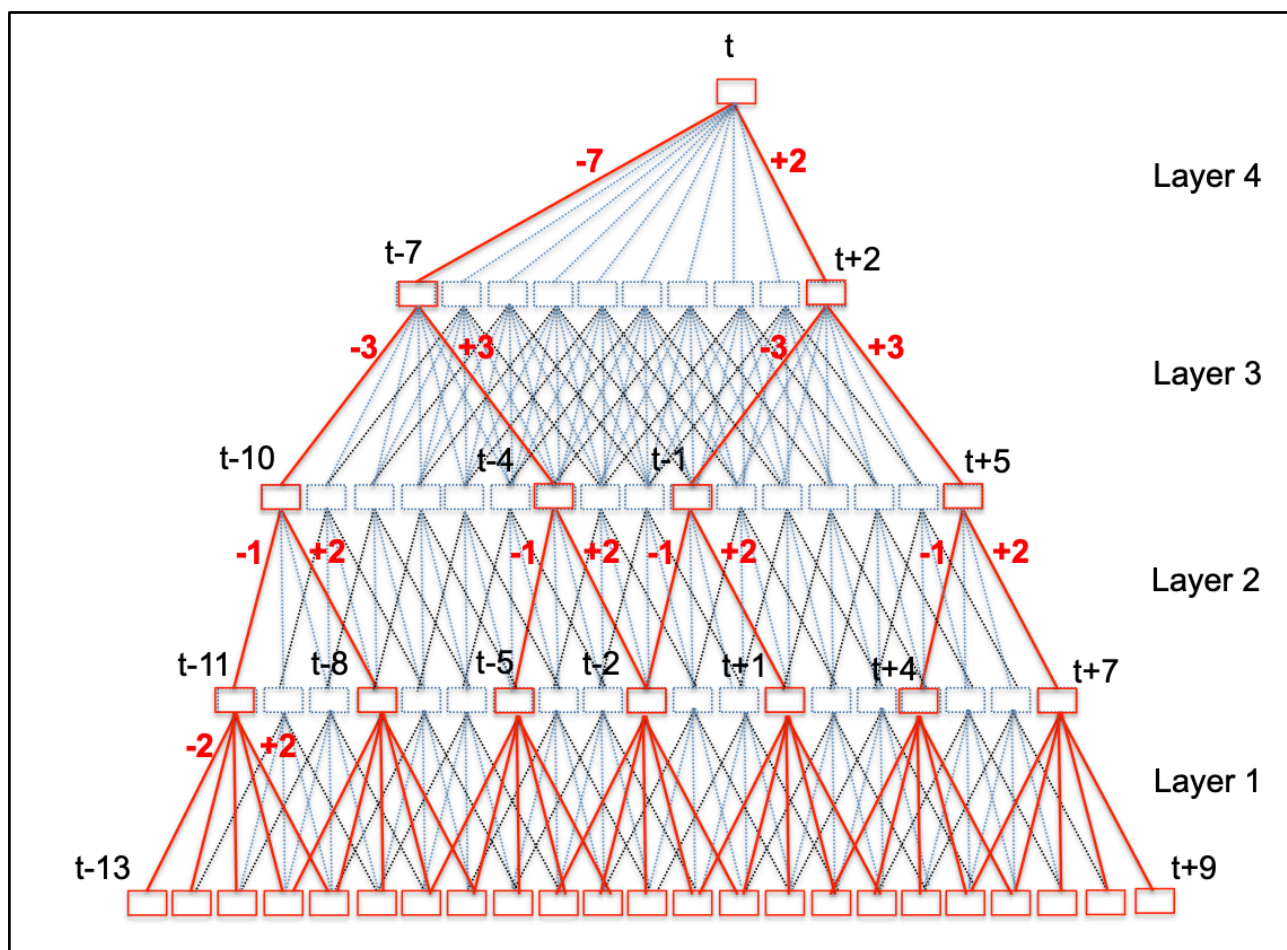


Рисунок 3 – Структура TDNN модели [14]

В архитектуре TDNN начальные преобразования обучаются на узких временных контекстах, а более глубокие слои обрабатывают скрытые активации

из более широкого контекста. Следовательно, более глубокие слои способны выявить более широкие временные связи в последовательности. Каждый слой в TDNN работает с разным временным разрешением, которое увеличивается по мере перехода к более глубоким слоям сети.

Часто TDNN рассматриваются как предшественники сверточных нейронных сетей, однако «свертка» здесь происходит по времени, а не по размерности признаков. В процессе обратного распространения, нижние слои сети обновляются градиентом, накопленным за все временные шаги входного временного контекста. Таким образом, нижние слои сети вынуждены обучаться преобразованиям признаков, инвариантных к положению во всем временном контексте.

Гиперпараметрами, определяющими работу TDNN, являются диапазоны временных контекстов для каждого слоя.

Нейронная сеть с временной задержкой нашла свое применение в x-vector архитектурах, предназначенных формировать уникальный дескриптор фиксированной длины для любого аудиосигнала. Дескрипторы, принадлежащие одному типу сигнала или голоса будут лежать достаточно близко в метрическом пространстве. Данное свойство позволит классифицировать полученные дескрипторы с помощью подходов классического машинного обучения, таких как K-nearest neighbors, SVM и других основанных на поиске «близких» друг к другу векторов. Схема базовой x-vector архитектуры представлена на рисунке 4.

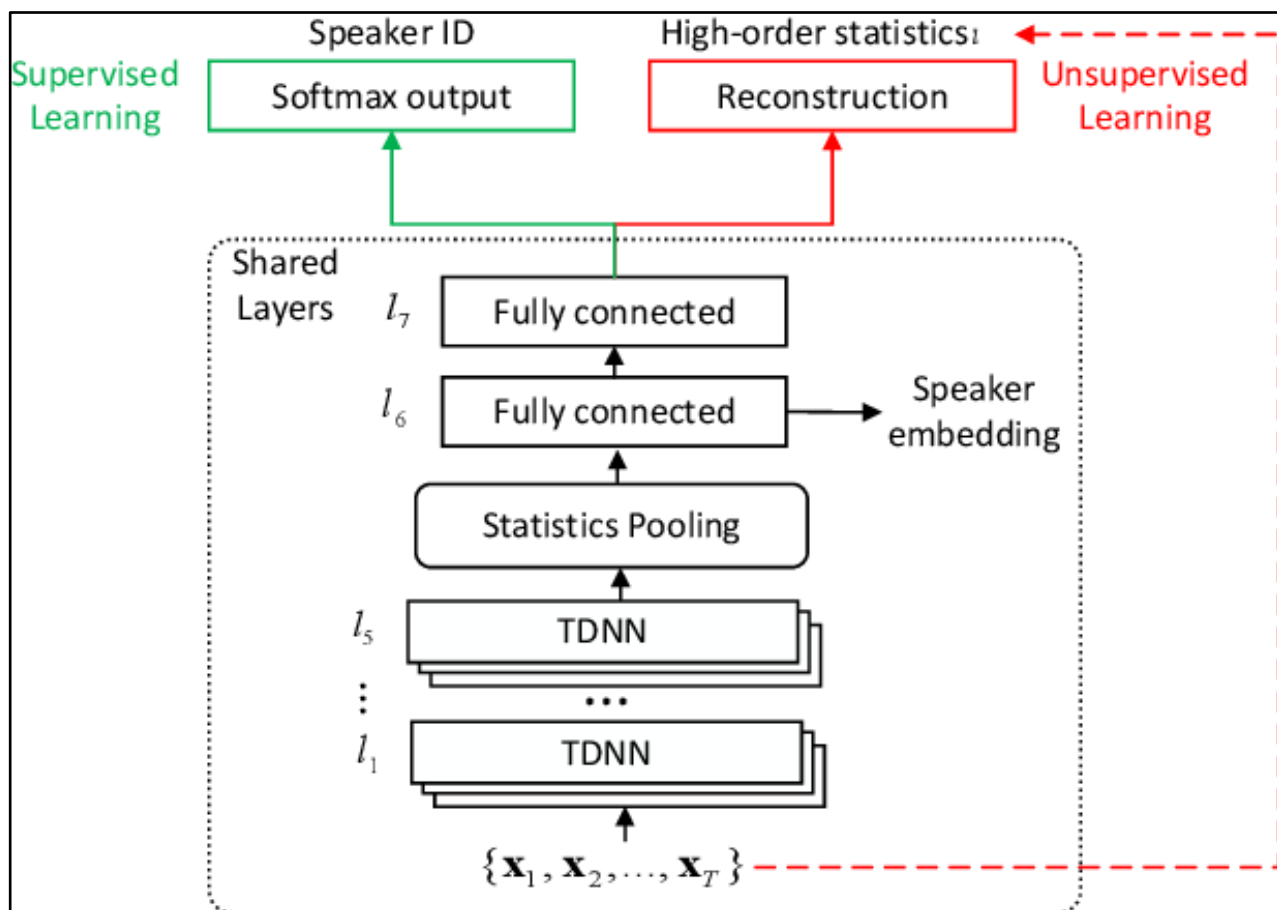


Рисунок 4 – x-vector архитектура для контролируемого и неконтролируемого обучения [15]

$\{x_1, x_2, \dots, x_T\}$  – набор входных признаков.

*TDNN* – слой сети с временной задержкой.

*Statistic Pooling* – слой усредняющий выходы со всех *TDNN* слоев.

*Fully connected* – полносвязный слой конструирующий вектор фиксированной размерности, который используется для классификации.

*Speaker embedding* – итоговое векторное представление голоса.

Сейчас проводится множество исследований на тему улучшения данной архитектуры. Одной из последних значимых работ является статья, в которой представили улучшение TDNN модели путем добавления в нее механизма канального внимания (emphasized channel attention) [16], что позволило достигнуть процента ошибки близкого к нулю (0.87%) на датасете из 1251 уникального голоса [11].



## **Выводы по второй главе**

В данном разделе был представлен один из подходов к предобработке аудиосигнала и извлечения первичных признаков. Также была рассмотрена нейронная сеть с временной задержкой и ее применение в x-vector архитектуре для решения задачи классификации человека по голосу.

### 3. ПРОЕКТИРОВАНИЕ

#### 3.1. Топология нейронной сети

Для решения поставленной задачи было решено использовать нейронную сеть с архитектурой, представленной на рисунке 5.

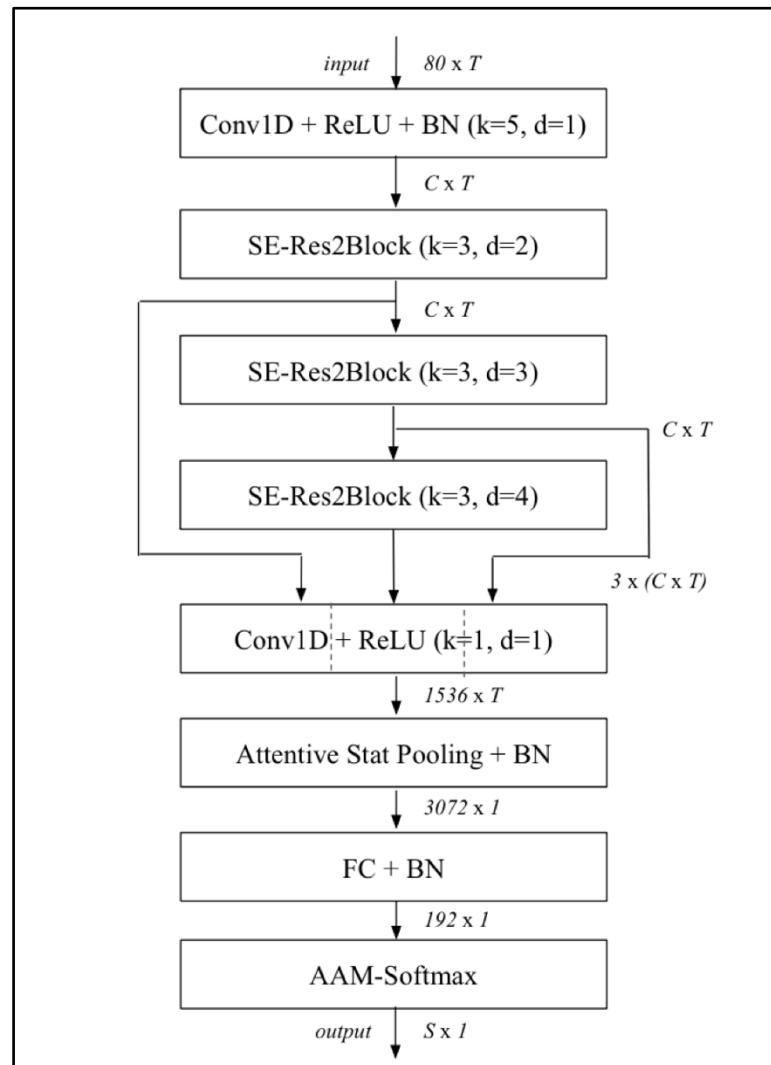


Рисунок 5 – Топология сети ECAPA-TDNN [16].

1.  $k$  – размер ядра свертки.
2.  $d$  – промежуток расширения (dilation spacing) в Conv1D или SE-Res2Block [16].
3.  $C$  – канальная размерность карт признаков.
4.  $T$  – темпоральная размерность карт признаков (количество временных фреймов).
5.  $S$  – количество тренировочных аудиозаписей.

Архитектура сети состоит из следующих слоев.

1. Сверточный одномерный слой (*Conv1d*) с функцией активации (*ReLU*) и пакетной нормализацией (*BN*) для ускорения обучения и сходимости сети. На вход принимает матрицу размерности  $80 \times T$  (80 коэффициентов MFCC и произвольное количество временных отрезков).
2. Три слоя SE-Res2Block [16],  $k=3$  и  $d=2$ ,  $d=3$ ,  $d=4$  соответственно. Данные слои реализуют нейронную сеть с временной задержкой внутри нашей представленной архитектуры.
3. *Conv1d + ReLU* с  $k=3$ ,  $d=1$ . На вход принимает карты признаков с трех предыдущих слоев.
4. Слой, вычисляющий среднее значение и стандартное отклонение входного массива векторов (*Attentive Stat Pooling*), *BN*.
5. Полносвязный слой (*FC*) и *BN*. На выходе получается вектор размерности  $(192 \times 1)$ .
6. *AAM-Softmax* – слой, использующийся при обучении, для того, чтобы похожие векторы конечной модели лежали максимально близко, а непохожие на них, наоборот, максимально далеко [17]. Это необходимо для эффективной работы модели, классифицирующей итоговые векторы.

### 3.2. Определение требований

В ходе проектирования приложения были определены функциональные и нефункциональные требования к разрабатываемой системе.

#### Функциональные требования

Можно определить следующий набор функциональных требований к проектируемой системе.

1. Система должна поддерживать загрузку аудиофайлов в формате mp3, wav, flac с частотой дискретизации 16 кГц.

2. Система должна предоставлять возможность запуска процесса классификации аудиозаписи и добавления аудиозаписи в общую базу данных.

### Нефункциональные требования

Можно определить следующий набор нефункциональных требований к проектируемой системе.

1. Система должна быть написана на языке программирования Python 3.
2. Должны быть предустановлены следующие библиотеки PyTorch, SpeechBrain, NumPy, Pandas, SQLite, Peewee, LibROSA.
3. Для процесса классификации голоса должна использоваться предобученная нейронная сеть.

### 3.3. Варианты использования системы

Для проектирования приложения был использован язык UML. Была построена модель взаимодействия актера «Пользователь» с приложением в виде диаграммы вариантов использования, которая представлена на рисунке 6.

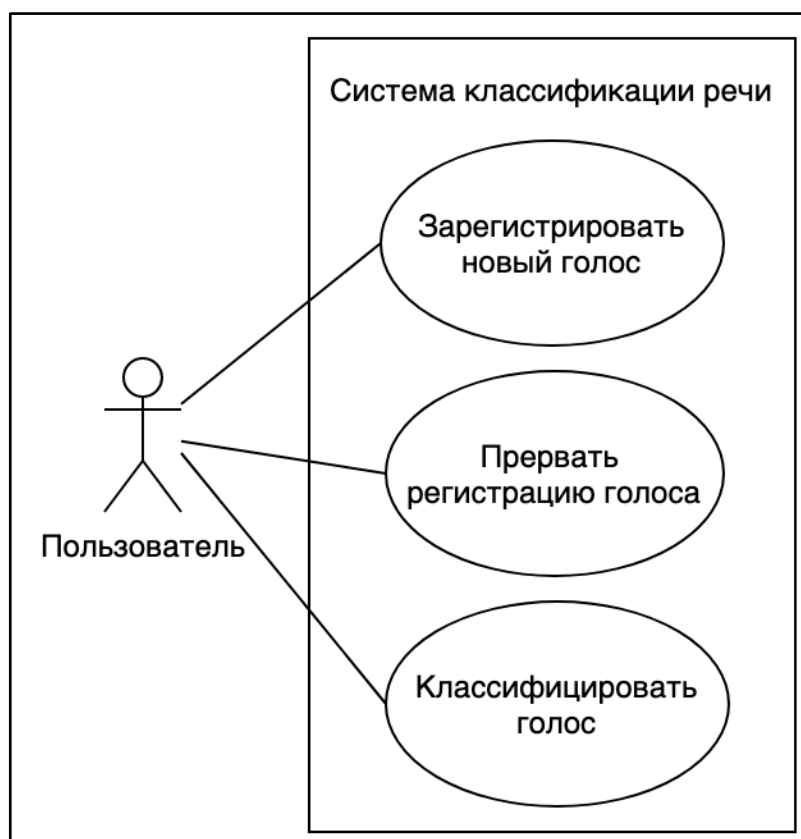


Рисунок 6 – Диаграмма вариантов использования

Пользователю доступны следующие варианты использования системы:

- 1) вариант использования «Зарегистрировать новый голос» позволяет добавить голос нового пользователя в систему классификации;
- 2) вариант использования «Классифицировать голос» позволяет запустить процесс классификации голоса.

Спецификация вариантов использования системы подробно описана в приложении А.

### 3.4. Архитектура системы

В ходе проектирования системы была разработана диаграмма классов, представленная на рисунке 7. Система содержит 6 классов: App, Bot, SpeakerRecognizer, EcapaTdnn, StorageManager, Speaker.

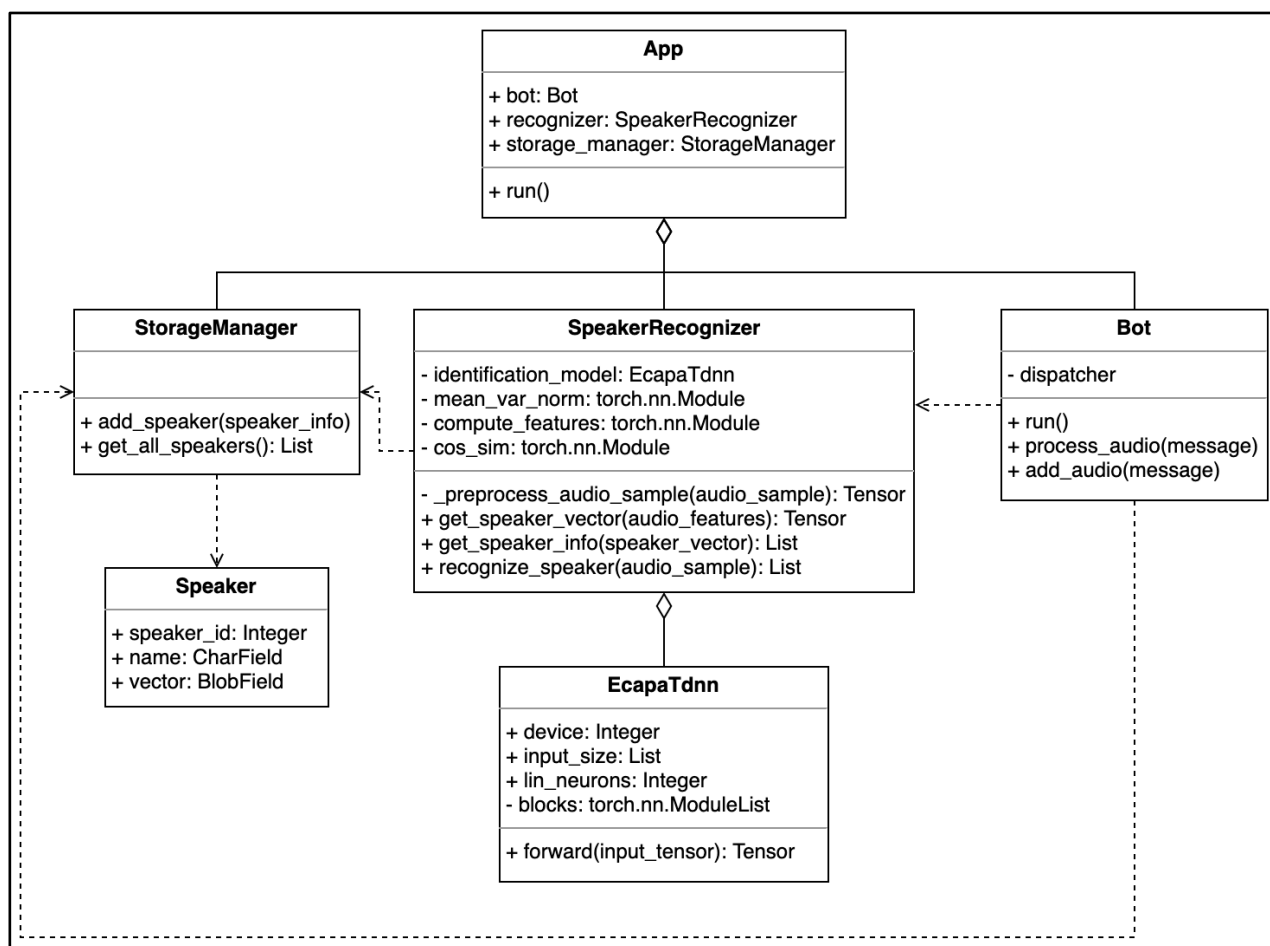


Рисунок 7 – Диаграмма классов

App – главный класс обеспечивающий работу системы. Класс содержит атрибуты, являющиеся объектами классов Bot, SpeakerRecognizer и StorageManager.

Класс содержит следующие методы.

1. `__init__(config)` – метод, инициализирующий параметры системы, которые описаны в переменной `config`.
2. `run()` – метод, запускающий систему.

Класс Bot принимает запросы пользователя на регистрацию голоса, распознавание и прерывание процесса регистрации. Атрибут `dispatcher` используется для обработки входных сообщений от пользователя через telegram api.

Класс содержит следующие методы.

1. `process_audio(message)` обрабатывает аудиосообщения пользователя на этапе идентификации голоса и отправляет пользователю сообщения о процессе и результате идентификации.
2. `add_audio(message)` обрабатывает аудиосообщения пользователя отправленные на этапе регистрации.
3. `run()` запускает работу бота.

Класс SpeakerRecognizer принимает запросы на распознавание от бота. Атрибут `identification_model` является объектом нейронной сети, конструирующей векторное представление голоса, `mean_var_norm` и `compute_features` используются на этапе предобработки аудиосигнала, `cos_sim` используется для подсчета «близости» векторов голосов.

Класс содержит следующие методы.

1. `_preprocess_audio_sample(audio_sample)` на вход получает аудиофрагмент и возвращает первичные признаки, полученные из этого фрагмента.
2. `get_speaker_vector(audio_features)` на вход получает массив первичных признаков аудиофрагмента и возвращает вектор фиксированной размерности в формате torch тензора.

3. `get_speaker_info(speaker_vector)` на вход получает вектор аудиофрагмента и возвращает информацию о данном векторе в виде списка в котором хранится уникальный идентификатор голоса, имя человека, которому принадлежит голос и уверенность модели в том, что голос действительно принадлежит этому человеку (значение от 0 до 1)
4. `recognize_speaker(audio_sample)` объединяет метод получения векторного представления голоса и метод получения информации об этом векторном представлении.

Класс `StorageManager` необходим для работы с хранилищем зарегистрированных голосов.

Класс содержит следующие методы.

1. `add_speaker(speaker_info)` реализует добавление нового голоса в систему. На вход получает информацию о регистрируемом голосе.
2. `get_all_speakers()` реализует получение данных обо всех зарегистрированных голосах (уникальный идентификатор, имя, векторное представление голоса).

`EsapaTdnn` – класс нейронной сети. Атрибут `device` определяет вычислительные мощности GPU или CPU будут задействованы, `input_size` – размерность входных параметров, `lin_neurons` – количество выходных нейронов сети, `blocks` – список слоев нейронной сети. Метод `forward(input_tensor)` реализует прямой проход по нейронной сети.

`Speaker` – класс реализующий взаимодействие с таблицей `Speaker` базы данных, содержит поля `id`, `name`, `vector`.

### 3.5. Диаграмма деятельности

Для процесса добавления голоса в систему была разработана диаграмма деятельности. Процесс добавления начинается после того, как пользователь отправляет команду «`/add_voice`», после этого отправляет имя и аудио и получает сообщение о том, что голос успешно добавлен. Также пользователь может

прервать добавление голоса с помощью команды «/cancel». Более подробно процесс добавления голоса можно посмотреть на рисунке 8.

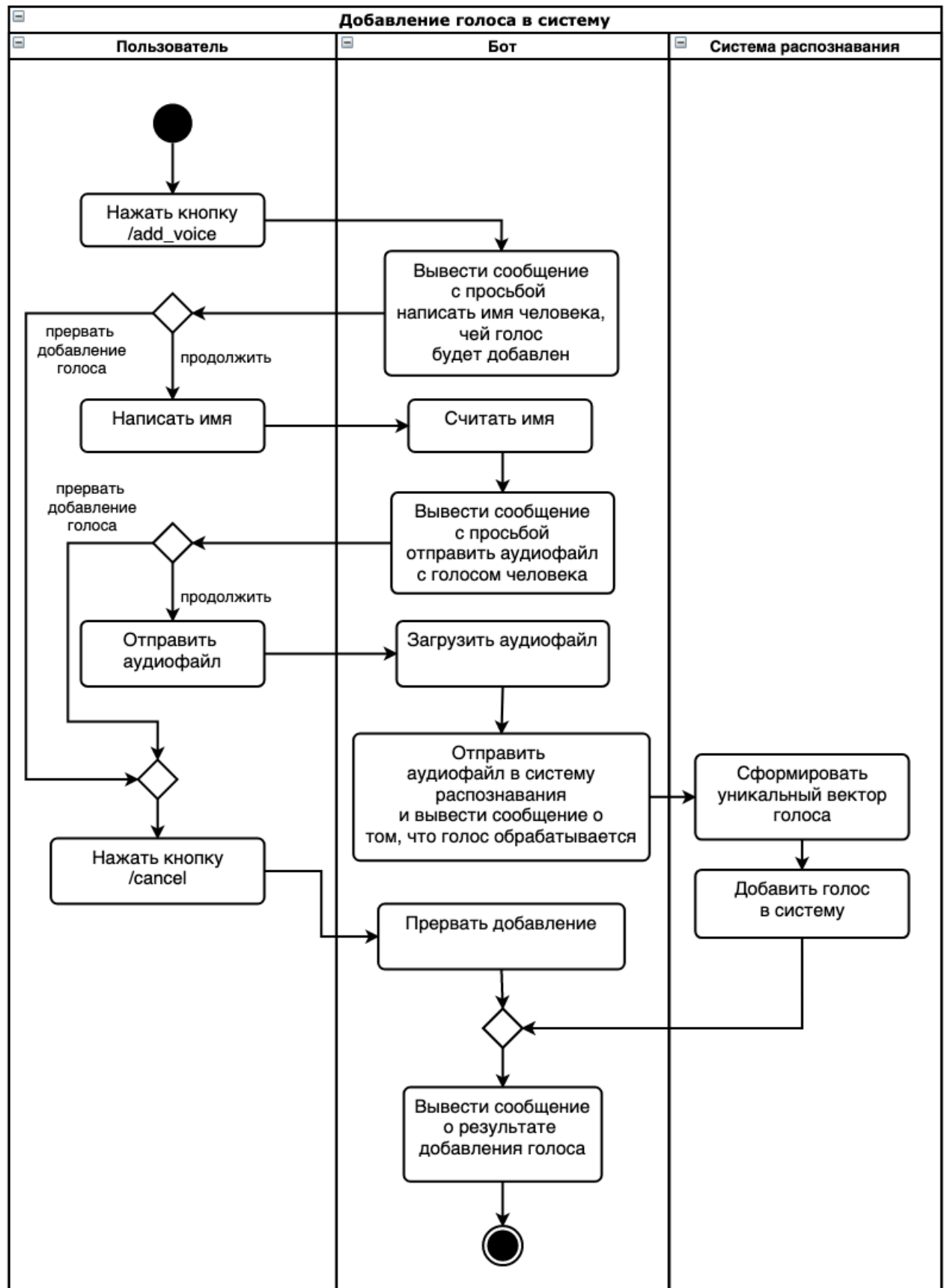
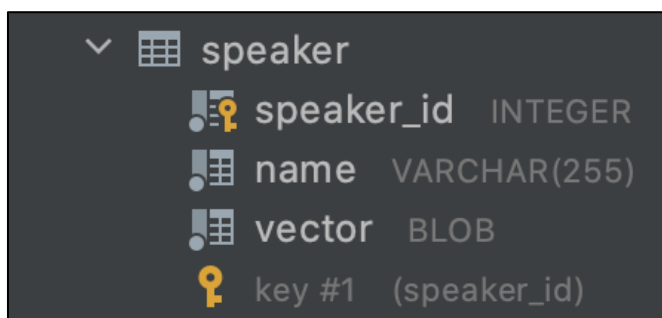


Рисунок 8 – Диаграмма деятельности добавления голоса



### 3.6. Схема базы данных

Для создания базы использовалась библиотека `peewee`. Создание базы происходит с помощью кода на языке программирования `python`. На локальном сервере была использована база данных SQLite с названием «`speakers.db`» и с одной таблицей, содержащей информацию о зарегистрированных голосах, а именно уникальный идентификатор голоса, имя, указанное при регистрации и векторное представление голоса. Подробнее поля таблицы можно рассмотреть на рисунке 9.



```
▼ speaker
  speaker_id INTEGER
  name VARCHAR(255)
  vector BLOB
  key #1 (speaker_id)
```

Рисунок 9 – Поля таблицы «`speaker`»

Поле «`speaker_id`» хранит идентификатор голоса и является первичным ключом, имеет тип данных «`INTEGER`». Поле автоматически инкрементируется при добавлении новой записи в таблицу.

Поле «`name`» хранит имя, которое пользователь указал при регистрации голоса, имеет тип данных «`VARCHAR`» с максимальной длиной 255 символов.

Поле «`vector`» хранит вектор пользователя в байтовом представлении, имеет тип данных «`BLOB`».

#### Выводы по третьей главе

В данном разделе была приведена топология используемой в системе нейронной сети, рассмотрена диаграмма вариантов использования системы и описаны спецификации ее прецедентов. Также были созданы диаграмма классов всей системы и диаграмма деятельности процесса добавления голоса.

## **4. РЕАЛИЗАЦИЯ**

### **4.1. Средства реализации**

Для реализации системы идентификации человека по голосу было решено использовать язык программирования Python 3.8. Для написания и отладки кода использовалась среда Pycharm Professional 2021.1.

В процессе разработки были использованы следующие программные продукты и библиотеки.

#### **PyTorch (1.8.1)**

Библиотека с открытым исходным кодом, используемая для решения задач машинного обучения.

#### **SpeechBrain (0.5.7)**

Библиотека с открытым исходным кодом, в которой описаны популярные методы предобработки аудиофайлов и представлены современные архитектуры нейронных сетей, для различных задач машинного обучения, связанных с обработкой аудио.

#### **Peewee (3.14.4)**

Библиотека с открытым исходным кодом, предоставляющая возможность работать с базами данных, используя синтаксис языка python.

#### **AIOgram (2.13)**

Библиотека с открытым исходным кодом, позволяющая работать с telegram api.

#### **TorchAudio (0.8.1)**

Библиотека с открытым исходным кодом, предназначенная, для обработки аудио данных на pytorch.

#### **LibROSA (0.8.1)**

Библиотека с открытым исходным кодом, предназначенная, для работы с аудиоданными.

## 4.2. Предобработка аудиофайла

На этапе предобработки происходит извлечение первичных признаков из аудиосигнала с заданной частотой дискретизации 16кГц. Сначала тензор представляющий аудиосигнал портируется на устройство, на котором будут производиться дальнейшие вычисления (GPU или CPU). Потом по аудиофрагменту строится мел-спектрограмма с параметрами, заданными при запуске системы:

- 1) частота дискретизации – 16кГц;
- 2) величина окна – 25 мс;
- 3) величина сдвига – 10 мс;
- 4) количество мел фильтров и соответственно выходная размерность мел шкалы спектрограммы – 80.

Далее ось амплитуд мел-спектрограммы переводится в децибелы, и спектрограмма транспонируется для последующей работы с моделью. Код, реализующий данный процесс приведен в листинге 1.

Листинг 1 – Функция извлечения первичных признаков

```
def _preprocess_audio_sample(self, audio_sample):
    waveform, sample_rate = audio_sample
    if len(waveform.shape) == 1:
        waveform = waveform.unsqueeze(0)
    waveform = waveform.to(self.device)
    # Computing features
    melspec = self.compute_features(waveform)
    melspec = torch.from_numpy(librosa.power_to_db(melspec))
    features = melspec.transpose(1, 2)

    return features
```

## 4.3. Получение информации о голосе

На этапе распознавания голоса после получения вектора из предобработанного аудио система сравнивает полученный вектор с остальными, уже лежащими в хранилище, векторами. Для этого используется метрика косинусной близости по следующей формуле [18].

$$similarity = \frac{x_1 \cdot x_2}{\max(\|x_1\|_2, \|x_2\|_2, \epsilon)}, \text{ где}$$

- 1)  $x_1$  и  $x_2$  массивы векторов с размерностями (1, 192) и (N, 192) (N – количество зарегистрированных голосов);
- 2)  $\epsilon$  – малая величина, чтобы избежать деления на ноль;
- 3)  $\|x_1\|_2$  и  $\|x_2\|_2 - l_2$  нормы векторов.

Близость изменяется в диапазоне от 0 до 1. Чем больше значение, тем ближе находятся векторы.

После определения вектора, который ближе всего находится к проверяемому вектору, система оценивает степень уверенности, т.е. на сколько большим является значение схожести. При значении схожести меньше определенного порога, который задается в конфигурационном файле, система выдает пользователю соответствующее сообщение. Код с получением информации о голосе представлен в листинге 2.

#### Листинг 2 – Функция получения информации о голосе

```
def get_speaker_info(self, speaker_vector):
    speaker_info = [[], []]
    with torch.no_grad():
        all_speakers = self.storage_manager.get_all_speakers()
        all_vectors = torch.stack([i[2] for i in all_speakers])
        similarities = self.cos_sim(speaker_vector, all_vectors)
        max_similar_idx = similarities.argmax()
        if similarities[max_similar_idx] > self.config['threshold']:
            # id, name, confidence
            speaker_info[0] = all_speakers[max_similar_idx][: -1] +
[similarities[max_similar_idx].item()]
            speaker_info[1] = all_speakers[max_similar_idx][: -1] +
[similarities[max_similar_idx].item()]
    return speaker_info
```

## 4.4. Обучение нейронной сети

После изучения предметной области для данной задачи была выбрана нейронная сеть, описанная в пункте 3.1. Для обучения сети использовалась библиотека `speechbrain`, предоставляющая удобный формат для обучения и проведения экспериментов, базовая архитектура сети также была взята в данной библиотеке.

На вход нейронной сети подается тензор размерностью (batch\_size, time, num\_features), на выходе сеть выдает вектор размерностью (1, 192). Код,

показывающий основные слои нейронной сети и функцию реализующую прямой проход по сети можно посмотреть в приложении В.

В качестве данных для обучения и тестирования было решено взять открытый набор данных русской речи voxforge [19] с 206 голосами разных людей. Так как датасет является достаточно небольшим, принято решение расширить его с помощью различных аугментаций, связанных с изменением скорости аудио, добавлением фонового шума, изменением интенсивности как основного аудио, так и фонового шума. Также, так как в основном исследовании по ESCAPE-TDNN [16] указывается, что при обучении сети достаточно использовать аудиофрагменты размером по 3 секунды, исходные аудио были нарезаны на более мелкие фрагменты, что тоже позволило увеличить обучающую выборку. Однако количество уникальных голосов по прежнему оставалось небольшим, поэтому было решено не обучаться на полученных данных с нуля, а использовать веса модели предобученной на наборе данных voxceleb-1 [11] с более чем 1000 уникальных англоязычных голосов.

Данные были поделены на две части таким образом, что в тренировочных и тестовых оказалось по 103 голоса. Голоса из тренировочных в тестовых данных не повторяются. Обучение проводилось на 50 эпохах с сохранением моделей, показывающих лучшее качество. Какая модель показала лучшее качество и ее процент ошибки на тестовых данных можно прочесть в разделе тестирования. Код с обучением нейронной сети представлен в листинге 3.

### Листинг 3 – Функция для обучения нейронной сети

```
def train_escapa(hparams, run_opts, datasets):
    # Initialize the Brain object to prepare for mask training.
    spk_id_brain = SpeakerBrain(
        modules=hparams["modules"],
        opt_class=hparams["opt_class"],
        hparams=hparams,
        run_opts=run_opts,
        checkpointer=hparams["checkpointer"],
    )

    spk_id_brain.fit(
        epoch_counter=spk_id_brain.hparams.epoch_counter,
        train_set=datasets["train"],
        valid_set=datasets["valid"],
        train_loader_kwargs=hparams["dataloader_options"],
        valid_loader_kwargs=hparams["dataloader_options"],
    )

    # Load the best checkpoint for evaluation
    test_stats = spk_id_brain.evaluate(
        test_set=datasets["test"],
        min_key="error",
        test_loader_kwargs=hparams["dataloader_options"],
    )
```

## 4.5. Реализация чат бота

Telegram предоставляет зарегистрированным пользователям возможность создавать специальные аккаунты чат-боты, которые способны автоматически обрабатывать входящие запросы от любых пользователей, за исключением других ботов, и отправлять сообщения. Логика бота контролируется при помощи HTTPS запросов на API для бота, развернутое на нашем сервере. С ботом можно общаться как с помощью сообщений, так и с помощью команд. Любая команда должна начинаться с символа косой черты «/» и не может быть длиннее 32 символов [20].

Для того, чтобы создать бота и начать с ним работу необходимо выполнить следующие шаги.

1. Написать специальному чат-боту, разработанному Telegram (@BotFather), команду /start и посмотреть список доступных команд и их описание. BotFather представляет собой чат-бота для администрирования других ботов.
2. Отправить команду /newbot.

3. Написать имя будущего чат-бота (Speaker Recognitor).
4. Написать никнейм бота, так чтобы он обязательно оканчивался на bot (speaker\_recognitor\_bot).
5. Сохранить токен, который пришлет BotFather. Токен представляет собой строку, которая в зашифрованном виде содержит необходимую информацию для аутентификации и авторизации.
6. При необходимости провести дополнительные настройки.

При обработке аудиосообщения или аудиофайла бот получает уникальный идентификатор файла, скачивает файл, отправляет его в систему распознавания и получает ответ с информацией о проверяемом голосе, далее чат-бот отправляет пользователю ответ системы распознавания. Код данного процесса можно посмотреть в листинге 4.

#### Листинг 4 – Код обработки аудио при классификации

```
async def process_audio(self, message: types.Message):
    audio = message.audio or message.voice
    await message.reply("Голос обрабатывается...")

    file = await self.bot.get_file(audio.file_id)
    file_path = file.file_path
    audio_path = f"{audio_data_path}/user_{audio.file_unique_id}.wav"
    await self.bot.download_file(file_path, audio_path)

    await message.answer("Голос проверяется нейросетью...")
    speaker_info = self.speaker_recognizer.recognize_speaker(audio_path)
    os.remove(audio_path)

    if speaker_info[0]:
        await message.answer(f"ID: {speaker_info[0][0]}\n"
                             f"NAME: {speaker_info[0][1]}\n"
                             f"CONFIDENCE: {speaker_info[0][2]}")
    elif speaker_info[1]:
        await message.answer(f"Система не достаточно уверена, но возможно это
Вы:\n"
                             f"ID: {speaker_info[1][0]}\n"
                             f"NAME: {speaker_info[1][1]}\n"
                             f"CONFIDENCE: {speaker_info[1][2]:.4f}")
    else:
        await message.answer("В системе нет зарегистрированных голосов")
```

## 4.6. Реализация менеджера хранилища зарегистрированных голосов

Для взаимодействия с таблицей зарегистрированных голосов в системе используется библиотека Peewee ORM, которая позволяет взаимодействовать с базами данных используя синтаксис языка программирования python. С помощью peewee можно описывать таблицы используя классы. Код класса `Speaker`, реализующего работу с таблицей `Speaker` нашей базы данных, представлен в листинге 5.

Листинг 5 – Описание таблицы через класс языка python

```
class Speaker(Model):
    speaker_id = IntegerField(primary_key=True)
    name = CharField(max_length=255)
    vector = BlobField()

    class Meta:
        database = db
```

Менеджер хранилища данных имеет две функции: добавление информации в таблицу `Speaker` и считывание всей информации, хранящейся в этой таблице. Код для данных функций представлен в листинге 6.

Листинг 6 – Функции менеджера хранилища данных

```
@classmethod
def add_speaker(cls, speaker_name, speaker_vector):
    speaker_vector_dump = pickle.dumps(speaker_vector)
    Speaker.create(name=speaker_name,
                   vector=speaker_vector_dump)

@classmethod
def get_all_speakers(cls):
    all_speakers_query = Speaker.select()
    speakers = [[s.speaker_id, s.name, pickle.loads(s.vector)] for s in
all_speakers_query]
    return speakers
```

## 4.7. Создание докер контейнера

Docker контейнер разрабатываемого сервиса построен на основе образа `python:3.8`. При создании контейнера в него копируется файл с зависимостями, которые необходимо установить. Установка зависимостей происходит с помощью команды «`pip install -r requirements.txt`». После этого в контейнер



копируются все файлы проекта, путь до которых не указан в файле «.dockerignore». Запуск системы внутри контейнера производится командой «python app.py». Команды для сбора образа контейнера и для его запуска представлены в листинге 3. Содержимое файла Dockerfile показано в листинге 4.

#### Листинг 7 – Команды для создания образа и запуска Docker контейнера

```
docker build -f Dockerfile -t speaker_rec_bot .
docker run --rm \
    -v "$(pwd)/data:/app/data" \
    -v "$(pwd)/config.yml:/app/config.yml" \
    -v "$(pwd)/src/recognition/weights:/app/src/recognition/weights" \
    --name speaker_bot \
    -d \
    speaker_rec_bot
```

#### Листинг 8 – Содержимое файла Dockerfile

```
FROM python:3.8
WORKDIR /app
COPY requirements.txt .
RUN apt-get update && apt-get install -y libsndfile1 ffmpeg
RUN pip install -r requirements.txt && rm requirements.txt
COPY . .
CMD ["python", "app.py"]
```

### Выводы по четвертой главе

В данной главе описаны средства разработки, используемые для создания приложения, предоставлена реализация алгоритма предобработки данных, код обучения нейронной сети и информация о тренировочных данных. Также рассмотрены реализации менеджера хранилища и бота и описано создание докер контейнера для данной системы.

## 5. ТЕСТИРОВАНИЕ

Тестирование проводилось в системе со следующими техническими характеристиками.

1. Architecture: x86\_64
2. CPU(s): 4.
3. Thread(s) per core: 1.
4. CPU model name: Intel(R) Core(TM) i5-6600K CPU @ 3.50GHz.
5. NVIDIA Corporation TU106 [GeForce RTX 2060 SUPER] 8GB.

### Тестирование нейронной сети

Для тестирования нейронной сети было случайно выбрано 103 уникальных голоса и разделены на несколько фрагментов. Всего в тестовой выборке насчитывается 1383 файла формата wav, что составляет примерно 22 % от количества всех аудиофайлов. График изменения ошибки на тестовой выборке на протяжении 50 эпох изображен на рисунке 10. Графики функций потерь для тренировочной и для тестовой выборок представлены на рисунке 11.

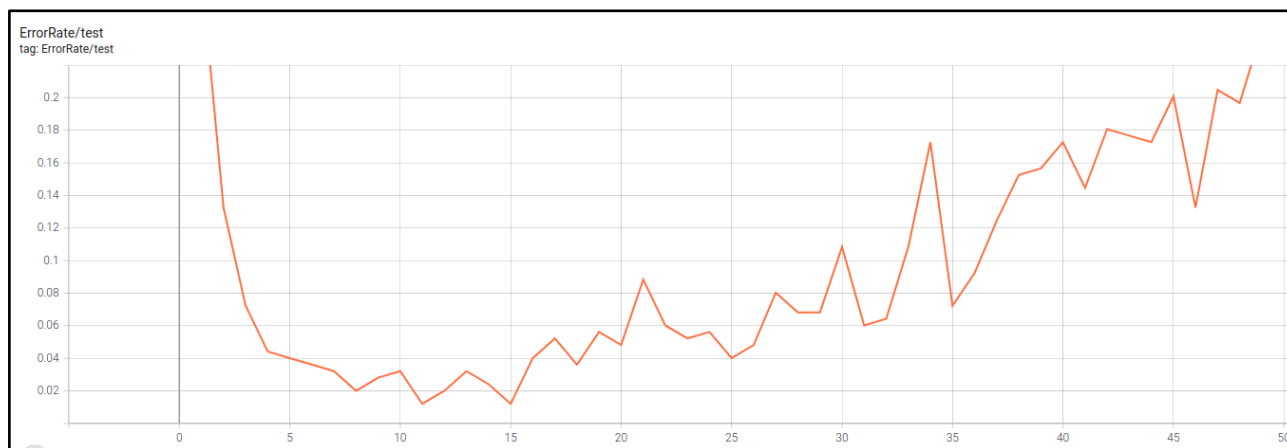


Рисунок 10 – Значение ошибки на тестовой выборке на протяжении обучения

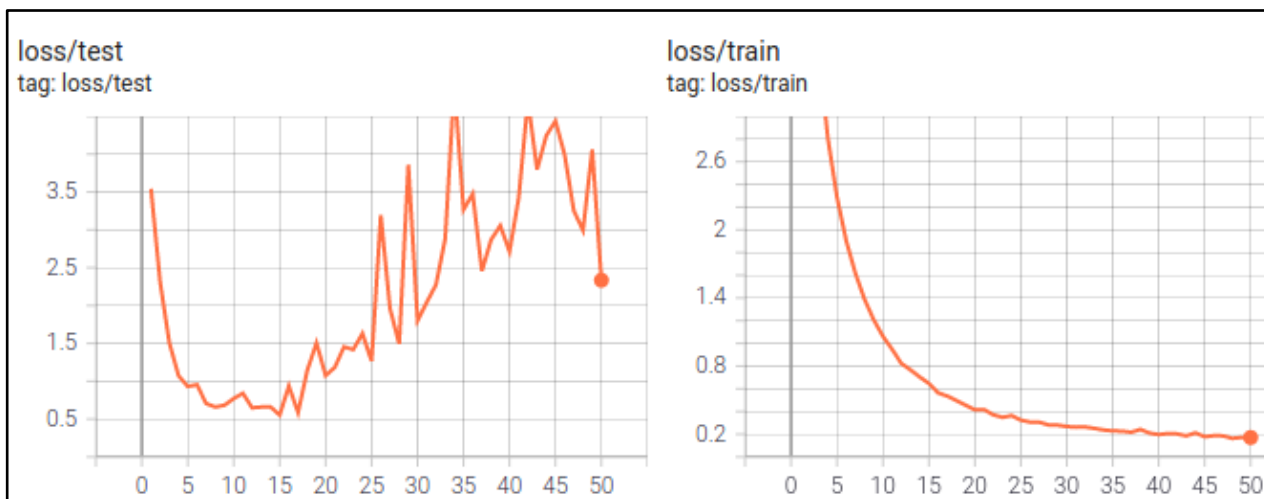


Рисунок 11 – Значение функции потерь на тренировочных и тестовых данных

На данных графиках хорошо прослеживается переобучение модели после 15 эпохи – значение функции потерь на тренировочных данных продолжает падать, а на тестовых падение сменяется на рост. Значение ошибки на тестовых данных также начинает расти после 15 эпохи. Поэтому в качестве конечных весов для модели были выбраны веса, сформированные на 15 эпохе обучения и итоговое значение функции потерь на тестовых данных оказалось равно 0.49, значение ошибки – 1.8 %.

### Функциональное тестирование

Функциональное тестирование необходимо для проверки реализуемости функциональных требований, составленных на этапе спецификации, т.е. функциональное тестирование определяет возможность программного обеспечения в определенных условиях решать задачи, поставленные пользователем. Результаты тестирования приведены в таблице 4.

Таблица 4 – Результаты функционального тестирования

№	Цель	Действие	Ожидаемый результат	Результат теста
1	Проверить работу бота	Пользователь отправил команду /start	Бот отправил приветствие и описал свое назначение	Пройден
2	Проверить работу бота	Пользователь отправил команду /help	Бот вывел список доступных команд и их описание	Пройден
3	Проверить процесс добавления нового голоса через загрузку аудиофайла формата wav/mp3 или аудиосообщения	Пользователь отправил команду /add_voice, после ответа бота вводит имя, потом отправляет аудиофайл или аудиосообщение	Бот попросил написать имя, после попросил отправить аудио. В итоге отправил сообщение о том, что голос успешно добавлен	Пройден
4	Проверить процесс прерывания добавления голоса	Пользователь отправил команду /add_voice и следом за ней /cancel	Бот написал, что добавление голоса прекращено	Пройден
5	Проверить процесс прерывания добавления голоса	При регистрации голоса на этапе отправки аудио пользователь написал /cancel	Бот написал, что добавление голоса прекращено	Пройден
6	Проверить процесс отправки неверных команд боту	Пользователь написал неверную команду	Бот ответил соответствующим сообщением	Пройден
7	Проверить процесс классификации голоса	Пользователь отправил аудиофайл	Бот написал информацию о результате классификации	Пройден
8	Проверить процесс классификации голоса при пустом хранилище	Пользователь отправил аудиофайл при отсутствии каких-либо голосов в хранилище	Бот ответил соответствующим сообщением	Пройден

### Вывод по пятой главе

В данной главе было рассмотрено функциональное тестирование системы и тестирование нейронной сети. Были представлены итоговые значения ошибки и функции потерь на тестовой выборке данных. Система выполняет свои задачи и успешно добавляет новые голоса и классифицирует уже добавленные.

## ЗАКЛЮЧЕНИЕ

В рамках данной работы была разработана и протестирована система идентификации человека по голосу. При этом были решены следующие задачи.

1. Изучены методы автоматического распознавания голоса и архитектуры современных нейронных сетей, применяемых для данной задачи.
2. Обучена нейронная сеть, архитектура которой показывает одни из лучших результатов на популярных датасетах.
3. Нейронная сеть протестирована на предварительно выделенных в тестовую выборку голосах.
4. Спроектировано приложение для идентификации человека по голосу.
5. Разработан и протестирован чат-бот для идентификации голосов.

В дальнейшем можно улучшить качество классификации, расширив набор данных и проведя настройку гиперпараметров модели, например увеличить размер выходного вектора голоса, количество каналов внимания, добавить функцию подавления посторонних шумов и функцию детектирования голоса перед его идентификацией. Также можно разделить бота и систему классификации в независимые модули и организовать их общение посредством запросов. Это позволит масштабировать систему и удобно организовать ее работу в облаке.

## ЛИТЕРАТУРА

1. A. Poddar, M. Sahidullah, G. Saha / «Speaker verification with short utterances: a review of challenges, trends and opportunities» // *IET Biometrics*, vol. 7, no. 2, pp. 91–101, Mar. 2018, doi: 10.1049/iet-bmt.2017.0065.
2. «Российский биометрический рынок в 2019–2022 годах» / URL: <https://www.secuteck.ru/articles/rossijskij-biometricheskij-rynok-v-2019-2022-godah> (дата обращения 31.03.2021).
3. A. E. Rosenberg and F. K. Soong / «Evaluation of a vector quantization talker recognition system in text independent and text dependent modes» // *Comput. Speech Lang.*, vol. 2, no. 3–4, pp. 143–157, Sep. 1987, doi: 10.1016/0885-2308(87)90005-2.
4. M. Hébert / «Text-Dependent Speaker Recognition» // *Springer Handbooks*, Springer, 2008, pp. 743–762.
5. J. Yuan, M. Liberman / «Speaker identification on the SCOTUS corpus» // *Proceedings - European Conference on Noise Control*, May 2008, vol. 123, no. 5, pp. 5687–5690, doi: 10.1121/1.2935783.
6. F. Richardson, D. Reynolds, N. Dehak / «A Unified Deep Neural Network for Speaker and Language Recognition» Apr. 2015 // URL: <http://arxiv.org/abs/1504.00923> (дата обращения: 06.04.2021).
7. «Speaker-recognition.» // URL: <https://github.com/ppwwyyxx/speaker-recognition> (дата обращения: 06.04.2021).
8. «SincNet.» // URL: <https://github.com/mravanelli/SincNet> (дата обращения: 06.04.2021).
9. A. Torfi, J. Dawson, N. M. Nasrabadi / «Text-Independent Speaker Verification Using 3D Convolutional Neural Networks» // *arXiv*, May 2017, URL: <http://arxiv.org/abs/1705.09422> (дата обращения: 10.04.2021).
10. «Pytorch Xvectors» // URL: [https://github.com/manojpamk/pytorch\\_xvectors](https://github.com/manojpamk/pytorch_xvectors) (дата обращения: 10.04.2021).
11. «Voxceleb Dataset» // URL: <https://www.robots.ox.ac.uk/~vgg/data/voxceleb/> (дата обращения: 10.04.2021).

12. S. S. Stevens, J. Volkman, E. B. Newman / «A Scale for the Measurement of the Psychological Magnitude Pitch» // *J. Acoust. Soc. Am.*, vol. 8, no. 3, pp. 185–190, 1937, doi: 10.1121/1.1915893.
13. A. Waibel, G. Hinton, K. Shikano, K. J. Lang / «Phoneme Recognition Using Time-Delay Neural Networks» // 1989.
14. V. Peddinti, D. Povey, S. Khudanpur / «A time delay neural network architecture for efficient modeling of long temporal contexts» // 2015.
15. L. You, W. Guo, L. Dai, J. Du / «Deep Neural Network Embedding Learning with High-Order Statistics for Text-Independent Speaker Verification» // 2019.
16. B. Desplanques, J. Thienpondt, K. Demuynck / «ECAPA-TDNN: Emphasized Channel Attention, Propagation and Aggregation in TDNN Based Speaker Verification» // *Proc. Annu. Conf. Int. Speech Commun. Assoc. INTERSPEECH*, vol. 2020-October, pp. 3830–3834, May 2020, doi: 10.21437/Interspeech.2020-2650.
17. J. Deng, J. Guo, N. Xue, S. Zafeiriou / «ArcFace: Additive angular margin loss for deep face recognition» // *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2019, vol. 2019-June, pp. 4685–4694, doi: 10.1109/CVPR.2019.00482.
18. «CosineSimilarity — PyTorch 1.8.1 documentation» // URL: <https://pytorch.org/docs/stable/generated/torch.nn.CosineSimilarity.html> (дата обращения 03.05.2021).
19. «Russian - voxforge dataset» // URL: <http://www.voxforge.org/ru> (дата обращения 03.05.2021).
20. «Документация Telegram: Боты» // URL: <https://tlgrm.ru/docs/bots> (дата обращения 03.05.2021).

## ПРИЛОЖЕНИЯ

### ПРИЛОЖЕНИЕ А. Спецификация вариантов использования системы

В таблицах 1–3 представлены спецификации вариантов использования системы.

Таблица 1 – Спецификация прецедента «Зарегистрировать новый голос»

<i>Прецедент:</i> «Зарегистрировать новый голос»
<i>ID:</i> 1
<i>Аннотация:</i> предоставляет возможность пользователю добавить аудиофайл в систему и имя человека, чей голос представлен на аудио.
<i>Главные актеры:</i> Пользователь
<i>Второстепенные актеры:</i> нет
<i>Предусловия:</i> приложение запущено.
<i>Основной поток:</i> <ol style="list-style-type: none"><li>1) прецедент начинается, когда пользователь нажимает кнопку «/add_voice»;</li><li>2) система предлагает отправить аудиофайл с голосом или записать аудиосообщение;</li><li>3) пользователь отправляет файл или аудиосообщение;</li><li>4) если системой получено некорректное аудиосообщение, то система отправляет пользователю сообщение о некорректном аудиосигнале и завершает прецедент иначе переходит к пункту 5;</li><li>5) система обрабатывает аудио, формирует дескриптор голоса и просит пользователя написать имя человека, чей голос представлен на аудиозаписи;</li><li>6) пользователь пишет имя человека;</li><li>7) система сохраняет имя и вектор человека;</li><li>8) система уведомляет пользователя об успешном добавлении человека.</li></ol>
<i>Постусловия:</i> сообщение о статусе добавления человека.
<i>Альтернативные потоки:</i> нет

Таблица 2 – Спецификация варианта использования «Классифицировать голос»

<i>Прецедент:</i> «Классифицировать голос»
<i>ID:</i> 2
<i>Аннотация:</i> классифицирует аудиосообщение и отправляет пользователю имя и id человека, которого система распознала на аудиозаписи или сообщает о том, что на аудио нет зарегистрированного человека.
<i>Главные актеры:</i> Пользователь
<i>Второстепенные актеры:</i> нет
<i>Предусловия:</i> приложение запущено.
<i>Основной поток:</i> <ol style="list-style-type: none"><li>1) пользователь отправляет аудиосообщение или аудиофайл в систему;</li><li>2) если отправлено некорректное аудио, система возвращает сообщение об ошибке пользователю и завершает прецедент иначе переходит к пункту 3;</li><li>3) система классифицирует аудио и отправляет пользователю имя распознанного человека, если он добавлен в систему или сообщение о том, что человек с данным голосом не зарегистрирован в системе.</li></ol>
<i>Постусловия:</i> сообщение со статусом классификации человека.
<i>Альтернативные потоки:</i> нет



Таблица 3 – Спецификация прецедента «Прервать добавление голоса»

<i>Прецедент:</i> «Прервать добавление голоса»
<i>ID:</i> 3
<i>Аннотация:</i> предоставляет возможность пользователю прервать добавление голоса на этапе введения имени человека – хозяина голоса и на этапе отправки аудиозаписи голоса.
<i>Главные актеры:</i> Пользователь
<i>Второстепенные актеры:</i> нет
<i>Предусловия:</i> пользователь находится на этапе введения имени или отправки аудиозаписи или аудиосообщения.
<i>Основной поток:</i> <ol style="list-style-type: none"> <li>1) прецедент начинается, когда пользователь получает сообщение от бота с просьбой написать имя хозяина голоса или отправить аудиофайл;</li> <li>2) пользователь нажимает на кнопку «/cancel»;</li> </ol> система прекращает процесс добавления и выводит сообщение о прерывании операции;
<i>Постусловия:</i> сообщение о прерывании операции добавления голоса.
<i>Альтернативные потоки:</i> нет

## ПРИЛОЖЕНИЕ В. Код класса нейронной сети ECAPA-TDNN

### Листинг В.1 – Класс нейронной сети с инициализирующим методом

```
class ECAPA_TDNN(torch.nn.Module):
    def __init__(self, input_size, device="cpu", lin_neurons=192, res2net_scale=8,
        activation=torch.nn.ReLU,
        channels=[512, 512, 512, 512, 1536],
        kernel_sizes=[5, 3, 3, 3, 1],
        dilations=[1, 2, 3, 4, 1],
        attention_channels=128,
        se_channels=128,
        global_context=True
    ):
        super().__init__()
        self.channels = channels
        self.blocks = nn.ModuleList()
        self.blocks.append( # The initial TDNN layer
            TDNNBlock(
                input_size,
                channels[0],
                kernel_sizes[0],
                dilations[0],
                activation,
            )
        )
        # SE-Res2Net layers
        for i in range(1, len(channels) - 1):
            self.blocks.append(
                SERes2NetBlock(
                    channels[i - 1],
                    channels[i],
                    res2net_scale=res2net_scale,
                    se_channels=se_channels,
                    kernel_size=kernel_sizes[i],
                    dilation=dilations[i],
                    activation=activation,
                )
            )
        # Multi-layer feature aggregation
        self.mfa = TDNNBlock(
            channels[-1],
            channels[-1],
            kernel_sizes[-1],
            dilations[-1],
            activation,
        )
        # Attentitive Statistical Pooling
        self.asp = AttentiveStatisticsPooling(
            channels[-1],
            attention_channels=attention_channels,
            global_context=global_context,
        )
        self.asp_bn = BatchNorm1d(input_size=channels[-1] * 2)
        # Final linear transformation
        self.fc = Conv1d(
            in_channels=channels[-1] * 2,
            out_channels=lin_neurons,
            kernel_size=1,
        )
    )
```

## Листинг В.2 – метод реализующий прямой проход сети

```

def forward(self, x, lengths=None):
    """Returns the embedding vector.
    Arguments
    -----
    x : torch.Tensor
        Tensor of shape (batch, time, channel).
    """
    # Minimize transpose for efficiency
    x = x.transpose(1, 2)

    xl = []
    for layer in self.blocks:
        try:
            x = layer(x, lengths=lengths)
        except TypeError:
            x = layer(x)
        xl.append(x)

    # Multi-layer feature aggregation
    x = torch.cat(xl[1:], dim=1)
    x = self.mfa(x)

    # Attentive Statistical Pooling
    x = self.asp(x, lengths=lengths)
    x = self.asp_bn(x)

    # Final linear transformation
    x = self.fc(x)

    x = x.transpose(1, 2)
    return x

```