Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ» (НИЯУ МИФИ)

ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ

КАФЕДРА КИБЕРНЕТИКИ

На правах рукописи

УДК 004.934

БАРАНОВА Д.Д.

Разработка системы двухфакторной аутентификации на основе голосовой биометрии

Выпускная квалификационная работа бакалавра

Направление подготовки 01.03.02 Прикладная математика и информатика

|  |
| --- |
| Выпускная квалификационная работа защищена  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 г. |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Секретарь ГЭК \_\_\_\_\_\_\_\_\_\_\_\_\_ |

г. Москва

2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**Национальный исследовательский ядерный университет «МИФИ»**

|  |  |  |
| --- | --- | --- |
|  | | |
| **Институт  интеллектуальных кибернетических систем** | | |
| **Кафедра №22 «Кибернетика»** | | |
| Направление подготовки 01.03.02 Прикладная математика и информатика | | |
| **Пояснительная записка** | | |
| к ВКР на тему: | | |
| Разработка системы двухфакторной аутентификации на основе голосовой биометрии | | |
|  |  |  |
| Группа | Б17-501 |  |
| Студент | (подпись) | Баранова Д.Д.  (ФИО) |
|  |  |  |
| Руководитель | (подпись) | Козин Р.Г.  (ФИО) |
|  |  |  |
| Научный консультант | (подпись) | (ФИО) |
|  |  |  |
|  |  |  |

**Москва 2021**

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**Национальный исследовательский ядерный университет «МИФИ»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Институт | Интеллектуальных кибернетических систем | Кафедра | №22 «Кибернетика» | |
| Группа | Б17-501 | |
|  |  |  |  | |
| Специальность  (направление) | Прикладная математика и информатика  (01.03.02) | «Утверждаю» | | |
| Зав. кафедрой | |  |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | | Загребаев А.М. |
|  |  | «\_\_\_» \_\_\_\_\_\_\_\_\_\_» 20\_\_г. | | |

**ЗАДАНИЕ НА ДИПЛОМНУЮ РАБОТУ**

**(выпускную квалификационную работу ВКР)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Фамилия, имя, отчество студента | | | | | | | | | | | | |
|  |  | | Баранова Дарья Дмитриевна | | | | | |  | | |  | |
|  |  | | (ФИО) | | | | | |  | | | (подпись) | |
|  | Тема работы (ВКР) | | | | « | Разработка системы двухфакторной | | | | | | | |
|  | аутентификации на основе голосовой биометрии | | | | | | | | | | | | » |
|  |  | | | | | | | | | | | | |
|  | Срок сдачи студентом готовой работы: 14 июня 2021 г. | | | | | | | | | | | | |
|  |  | | | | | | | | | | | | |
|  | Место выполнения | | | НИЯУ МИФИ, кафедра «Кибернетика» | | | | | | | | | |
|  |  | | | | | | | | | | | | |
|  | Руководитель работы | | | | | | | | | | | | |
|  |  | Козин Рудольф Глебович | | | | |  | к.т.н., доцент ИИКС | |  |  | | |
|  |  | (ФИО) | | | | |  | (уч. степень, должность) | |  | (подпись) | | |
|  |  | | | | | | | | | | | | |
|  | Соруководитель работы от НИЯУ МИФИ | | | | | | | | | | | | |
|  |  |  | | | | |  |  | |  |  | | |
|  |  | (ФИО) | | | | |  | (уч. степень, должность) | |  | (подпись) | | |
|  |  | | | | | | | | | | | | |
|  | Консультант работы | | | | | | | | | | | | |
|  |  |  | | | | |  |  | |  |  | | |
|  |  | (ФИО) | | | | |  | (уч. степень, должность) | |  | (подпись) | | |

1. **Аналитическая часть**

Изучение и сравнительный анализ методов распознавания речи

Изучение и сравнительный анализ методов голосовой идентификации

Обзор особенностей аудиозаписи и речи, затрудняющих работу методов

Изучение существующих программных решений

1. **Теоретическая часть**

Подготовка данных для обучения моделей

Разработка метода распознавания речи

Разработка и модификация метода идентификации диктора по голосу

Разработка структуры системы аутентификации

1. **Технологическая и практическая часть**

Выбор языка и вспомогательных модулей для реализации системы

Программная реализация разработанных моделей и алгоритмов

Экспериментальное исследование влияния архитектуры модели на точность распознавания речи

Экспериментальное исследование влияния архитектуры модели на точность идентификации по голосу

Программная реализация API

Тестирование системы

Дата выдачи задания «01» апреля 2021 г.

**РЕФЕРАТ**

Пояснительная записка содержит 108 страниц, 46 рисунков, 15 таблиц, 29 формул, 49 использованных источников, 9 приложений.

ГОЛОСОВАЯ ИДЕНТИФИКАЦИЯ, РАСПОЗНАВАНИЕ РЕЧИ, РЕКУРРЕНТНАЯ НЕЙРОННАЯ СЕТЬ, АКУСТИЧЕСКАЯ МОДЕЛЬ, ЯЗЫКОВАЯ МОДЕЛЬ, РАССТОЯНИЕ ЛЕВЕНШТЕЙНА, МОДЕЛЬ ГАУССОВЫХ СМЕСЕЙ

Целью данной работы является повышение защищенности систем от несанкционированного доступа с помощью автоматической двухфакторной аутентификации на основе распознавания речи и идентификации диктора по голосу.

В первой главе рассматриваются различные существующие подходы к решению задачи идентификации по голосу, а также к решению задачи распознавания речи. Исследуются особенности речи и всей аудиозаписи, затрудняющих работу методов. Описываются существующие системы и программные решения на рынке. Затем выбираются методы, которые будут использоваться в работе.

Во второй главе описываются обучающие и тестовые данные. После этого приводится описание разработанных методов для выполнения идентификации по голосу и для распознавания речи, выделяются подзадачи и приводятся способы их решения. Описываются подходы к модификации методов. Формулируются критерии качества.

В третьей главе для определения оптимальных архитектур моделей проводятся экспериментальные исследования на полученной выборке данных. Также описываются средства разработки, вспомогательные библиотеки и модули, реализованные функции. Формулируются требования к всей системе и реализуются все компоненты сервиса. Приводятся результаты тестирования готовой системы. После этого подводятся итоги проведённой работы с кратким описанием результатов по каждому разделу.

В приложениях А, Б, В приведен программный код метода распознавания речи.

В приложениях Г, Д приведен код метода голосовой идентификации.

В приложениях Е, Ж, И, К приведен программный код системы двухфакторной аутентификации, реализованной в виде API.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 8](#_Toc61907954)

[1 Анализ существующих подходов к двухфакторной аутентицикации по голосу 10](#_Toc61907955)

[1.1 Понятие аутентификации 10](#_Toc61907956)

[1.2 Известные методы распознавания речи 11](#_Toc61907956)

[1.3 Классификация особенностей речи, затрудняющих её распознавание 24](#_Toc61907962)

[1.4 Известные сервисы распознавания речи 25](#_Toc61907962)

[1.5 Выбор метода распознавания речи 26](#_Toc61907962)

[1.6 Известные методы идентификации по голосу 27](#_Toc61907962)

[1.7 Особенности аудиозаписи, затрудняющие идентификацию по голосу 31](#_Toc61907962)

[1.8 Существующие программные решения и области применения голосовой идентификации 32](#_Toc61907962)

[1.9 Выбор метода идентификации по голосу 33](#_top)

[1.10 Постановка задачи и выводы 33](#_Toc61907969)

[2 Разработка методов и проектирование системы аутентификации 35](#_Toc61907974)

[2.1 Обучающая выборка и входные данные 35](#_Toc61907986)

[2.2 Метод работы системы распознавания речи 35](#_Toc61907976)

[2.3 Метод работы системы идентификации голоса 46](#_Toc61907977)

[2.4 Структура системы аутентификации 53](#_Toc61907975)

[2.5 Оценка эффективности и метрики качества 55](#_Toc61907984)

[2.6 Выводы 56](#_Toc61907984)

[3 Программная реализация системы аутентификации 58](#_Toc61907985)

[3.1 Выбор языка и вспомогательных модулей для реализации системы 58](#_Toc61907987)

[3.2 Программная реализация моделей и алгоритмов 58](#_Toc61907988)

[3.3 Экспериментальное исследование влияния архитектуры модели на точность распознавания речи 62](#_Toc61907989)

[3.4 Экспериментальное исследование влияния архитектуры модели на точность идентификации по голосу 67](#_Toc61907990)

[3.5 Выводы по результатам экспериментальных исследований 68](#_Toc61907991)

[3.6 Программная реализация API 69](#_Toc61907988)

[3.7 Тестирование системы 75](#_Toc61907992)

[3.8 Выводы 79](#_Toc61907993)

[ЗАКЛЮЧЕНИЕ](#_Toc61907994) 80

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](#_Toc61907995) 81

[ПРИЛОЖЕНИЕ А 85](#_Toc61907996)

[ПРИЛОЖЕНИЕ Б](#_Toc61907997) 89

[ПРИЛОЖЕНИЕ В 92](#_Toc61907998)

[ПРИЛОЖЕНИЕ Г 94](#_Toc61907998)

[ПРИЛОЖЕНИЕ Д 96](#_Toc61907998)

[ПРИЛОЖЕНИЕ Е 97](#_Toc61907998)

[ПРИЛОЖЕНИЕ Ж 99](#_Toc61907998)

[ПРИЛОЖЕНИЕ И 100](#_Toc61907998)

[ПРИЛОЖЕНИЕ К 105](#_Toc61907998)

**ВВЕДЕНИЕ**

В настоящее время аутентификация используется повсеместно. Идет постоянное противостояние методов защиты и методов взлома систем, поэтому технологии и способы аутентификации все время развиваются и совершенствуются. С развитием компьютерных технологий и появлением устройств получения биометрических данных возникла возможность проводить аутентификацию на основе биометрии человека. Её принцип работы основан на биологических и физических характеристиках человека, а не на знании паролей, шифров, ключей.

Биометрическая аутентификация обладает рядом преимуществ. Биометрические системы доступа являются очень удобными для пользователей. Данные не могут быть скопированы. Пропадает возможность взлома путем перебора. Также, в отличие от паролей или ключей, которые могут быть потеряны, украдены или скомпрометированы, биологические характеристики человека, на которых основана биометрическая система, всегда находятся вместе с владельцами, и проблемы нарушения сохранности не возникает. Потерять данные или передать их третьим лицам почти невозможно.

Из недостатков использования биометрии в системе аутентификации называют обычно невозможность изменить данные в случае, если они были скомпрометированы. Другими словами, если злоумышленник может каким-то образом воссоздать характерные биологические черты объекта, то объект не имеет возможности изменить свои биометрические данные и вследствие этого использовать их в дальнейшем.

Одним из видов биометрических данных является голос человека. Голос говорящего содержит в себе его личные черты, обусловленные уникальным строением органов произношения (уникальной голосовой формой тракта, размером гортани), особенностями дикции, акцентом, ритмом, и индивидуальной манерой речи говорящего. Таким образом, можно автоматически идентифицировать говорящего по его голосу через компьютер.

Для повышения надежности проведения аутентификации системы создаются многофакторными. В данной работе первым фактором будет являться голос говорящего. Вторым фактором будет выступать фраза-пароль, произнесенная человеком в том отрывке аудиозаписи, который передан системе для проведения аутентификации. Многофакторность позволит увеличить надежность системы и компенсировать недостатки биометрических данных, которые в случае взлома невозможно изменить. Чтобы получать фразу-пароль из аудиозаписи, необходимо дополнительно построить систему распознавания речи и интегрировать её в систему аутентификации.

Таким образом, целью данной работы является создание двухфакторной системы аутентификации, факторами которой выступают:

* свойство, которым обладает объект (уникальной голос человека);
* знание информации объектом (фраза-пароль, которая была произнесена).

Поставленная цель предполагает решение следующих задач:

* выполнить обзор существующих методов распознавания речи;
* выполнить обзор существующих методов идентификации по голосу;
* получить размеченные обучающие данные;
* разработать метод распознавания речи;
* экспериментально определить оптимальную реализацию метода распознавания речи;
* разработать метод идентификации по голосу;
* экспериментально определить оптимальную реализацию метода идентификации по голосу;
* реализовать систему аутентификации в виде API;
* выполнить тестирование системы и определить степень надежности.

**1. Анализ существующих подходов к двухфакторной аутентификации по голосу**

**1.1 Понятие аутентификации**

Понятие аутентификации тесно связано с идентификацией и авторизацией. Их часто путают между собой.

*Идентификация* говорящего направлена на определение личности говорящего по тестовому высказыванию из сформированной базы данных по формуле:

(1)

где обозначает количество имеющихся в базе говорящих, - функция для вычисления меры сходства, - сохраненный на сервере образец голоса k-ого говорящего. Процесс идентификации заключается в поиске в базе и присвоении пользователю идентификатора, которому он соответствует. Если никогда не может быть вне зарегистрированных в базе говорящих, тогда задача идентификации говорящего является задачей закрытого типа; в противном случае это открытая задача.

Понятие *авторизации* затрагивает уже не только пользователя, но и определяет его права. Допустим, в системе есть какой-то ограниченный список возможностей или функций. В зависимости от пользователя, ему может быть доступен либо весь набор этих возможностей, либо частично ограниченный. Авторизация представляет из себя процесс определения прав доступа пользователя к ресурсам. Именно на её основе происходит предоставление конкретному пользователю возможностей в соответствие с положенными ему правами.

*Аутентификация* представляет из себя проверку подлинности. Например, проверка подлинности пользователя – соответствия человека, который хочет войти в систему, и того человека, которым он представляется. Аутентификация возможна посредством предоставления пароля – тогда проверка подлинности будет осуществляться путем сравнения введенного пароля с паролем, сохраненным в базе данных. Если сравнение прошло успешно, то система может поверить указанному юзернейму (логину), которым представлялся пользователь. Аутентифицированный пользователь имеет право пользоваться этим юзернеймом и получить набор прав, которые приписаны этому юзернейму – пройти авторизацию системой. Пароль в этом примере называется фактором аутентификации.

*Факторы аутентификации* – это категории свойств пользователя, которые доказывают, что он это он, свойства, которые делают человека уникальным. Обычно выделяют три фактора:

* что пользователь знает – фактор знания. Первый и самый распространенный фактор. К нему относятся пароль или ответ на секретный вопрос;
* чем пользователь владеет – вещественный фактор. Здесь ключом является что-то, чем пользователь обладает – какой-то уникальный вещественный предмет. Ключ от механического замка, карточка с чипом, электронный токен, сим карта;
* чем пользователь является – биофактор или фактор свойства. Характеристикой является физическая особенность пользователя. Это может быть любая биометрическая характеристика - отпечаток пальца или ладони, голос или сетчатка глаза.

Иногда выделяют дополнительные факторы - где пользователь находится, что пользователь делает (поведенческий фактор), кто пользователя знает (социальный фактор).

Таким образом, расширенная (или многофакторная) аутентификация построена на совместном использовании нескольких факторов аутентификации. Для подтверждения подлинности необходимо одновременно удовлетворить несколько требований системы. Это значительно повышает её защищённость.

**1.2 Известные методы распознавания речи**

Сейчас существует несколько методов решения задачи распознавания речи в аудиозаписи. Успешность и скорость их работы во многом зависит от дикции автора, качества записи аудиофайла, поданного на вход, фонового шума, а также от качества работы спроектированной системы распознавания, и, как следствие, от выбранного метода.

Разнообразие особенностей речи, а также реализуемых подходов к решению задачи распознавания ведут к многообразию существующих решений. Все они борются за лидерство по качеству работы, но все еще имеют свои индивидуальные достоинства и недостатки.

Рассмотрим подробнее самые популярные существующие решения, а также особенности речи, затрудняющие распознавание.

**1.2.1 Скрытое Марковское моделирование (HMM)**

До сих пор наиболее успешный [6] и часто используемый метод для распознавания речи - это математическая модель, полученная на основе модели Маркова.

Скрытая Марковская модель представляет собой статистическую модель, моделирующую процесс Маркова с неизвестными параметрами. Пусть имеет N состояний модели. Каждое из состояний имеет свою вероятность наступления. Обозначим вероятности перехода между состояний как матрицу A={aij}, где aij – вероятность перехода из i-го в j-е состояние. В каждом из своих состояний система принимает одно из M значений какого-то своего параметра. Обозначим вероятность выпадения каждого из M значений параметра в каждом из N состояний системы через B={bj(k)}, где bj(k) – вероятность выпадения k-го значения параметра в j-м состоянии. Также введем вектор вероятности наступления начального состояния через π={πi}, где πi – вероятность того, что в начальный момент система окажется в i-м состоянии. На рисунке 1 представлена схема строения скрытой Марковской модели. Состояния представлены в виде голубых кружков (N штук), значения скрытого параметра представлены с помощью желтых квадратов (M штук). Каждая стрелка символизирует возможный переход и имеет свой вес (значение вероятности). Таким образом, скрытой Марковской моделью называется тройка λ = {A, B, π} [7].

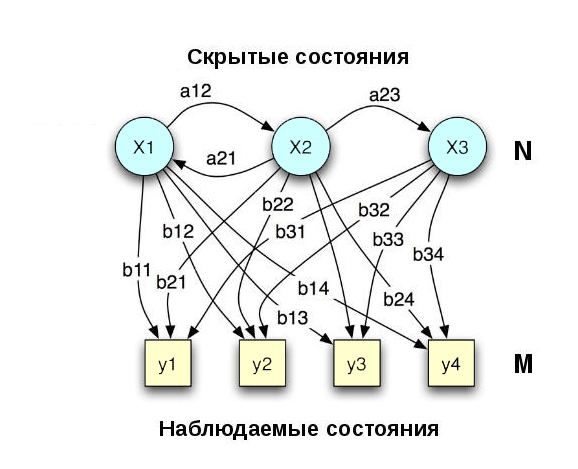


Рисунок 1 - Общая схема устройства скрытой Марковской модели [8]

Использование скрытых Марковских моделей для распознавания речи основано на трех допущениях [9]:

1. речь можно разбить на фрагменты (соответствующие состояниям в СММ) так, что внутри каждого фрагмента речевой сигнал можно рассматривать как стационарный. Параметры речи в его пределах постоянны;
2. переход между состояниями осуществляется мгновенно;
3. вероятность каждого фрагмента зависит только от текущего состояния модели и не зависит от предыдущих состояний.

При распознавании речь разбита на мельчайшие фрагменты (не только гласные и согласные, но и сопряженные сложные звуки). Все эти фрагменты представлены как состояния в Марковской модели. При распознавании слова необходимо определить наиболее подходящую модель, которая может ему соответствовать. В зависимости от уровня, на котором выбирается проведение распознавания, каждая модель может обозначать один из звуков русского языка (или отсутствие звука - одна из моделей), или одно из изученных слов. Состояние также может иметь переход в собственное состояние, если звук повторяется.

Для осуществления распознавания с помощью Марковских моделей предварительно строят множество моделей на каждый элемент области значений. Так, если распознавание проводится на уровне слов, то необходимо построить Марковские модели для каждого слова, которые мы хотим распознать.

**1.2.2 Использование N-грамм**

N-грамма – это модель представления данных, которая также может использоваться для распознавания речи [10]. N-граммная модель рассчитывает вероятность последнего элемента N-граммы при условии, что известны все предыдущие. В задаче распознавания речи элементами могут выступать фонемы, если задача решается на более низком уровне, или же целые слова. При использовании этого подхода предполагается, что появление каждого слова (фонемы) зависит только от предыдущих слов (фонем), причем значение N указывает, от скольких предыдущих слов оно зависит.

Рассмотрим пример биграммной модели. В ней N = 2. Будем рассматривать фразу «мой дядя самых честных правил» на уровне слов. Другими словами, целью построения модели будет являться определение вероятности употребления этой фразы. Вероятность можно задать формально как вероятность возникновения именно этой последовательности слов в каком-то тексте. Вероятность фразы можно вычислить как произведение вероятностей каждого из слов этой фразы:

(2)

Здесь обозначает условную вероятность возникновения «самых» при условии появления «мой дядя». Чтобы определить , нужно посчитать, сколько раз это слово встретилось в тексте, и поделить это значение на общее число слов. Рассчитать вероятность уже сложнее.

Чтобы упростить эту задачу, примем, что вероятность слова в тексте зависит только от предыдущего слова. Именно здесь проявляет себя выбранное значение N. Тогда формула для расчета фразы примет вид:

(3)

Рассчитать условную вероятность несложно. Для этого считаем количество пар 'мой дядя', и делим на количество в тексте слова 'мой'.

В результате, если мы посчитаем все пары слов в некотором тексте, мы сможем вычислить вероятность произвольной фразы. Этот набор рассчитанных вероятностей и будет биграммной моделью.

Степень модели указывает на количество слов, учитываемых при расчете вероятностей. Рисунок 2 показывает несколько N-грамм для небольших значений N. Униграмма имеет степень 1, биграмма – степень 2, триграмма – степень 3.

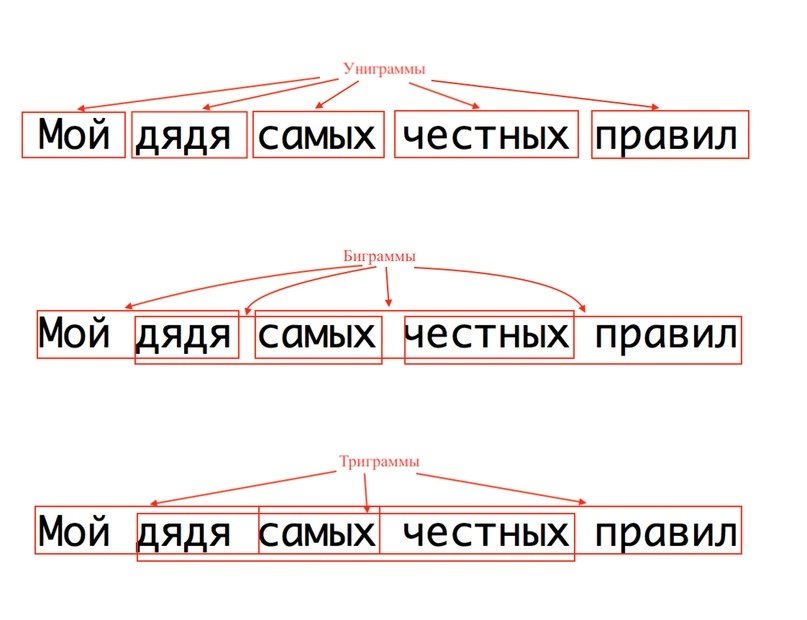


Рисунок 2 - N-грамма для N=1,2,3

Представим фразу **w** как последовательность слов Тогда более формально N-граммная модель рассчитывает вероятности по формуле:

(4)

(5)

Тогда формула вычисления вероятностей биграммы представляется в виде:

(6)

**1.2.3 Обзор подхода искусственного интеллекта**

Данный подход получил широкую известность [11] и использует методы машинного обучения в своей основе, в частности – нейронные сети.

**1.2.3.1 Рекуррентные сети**

Рекуррентные нейронные сети (RNN) - это класс нейронных сетей, обладающих искусственной внутренней памятью. Такие сети используются в случаях, когда важна сама последовательность данных и та временная динамика, которая соединяет данные. Последовательные данные – это, в основном, просто упорядоченные данные, в которых связанные объекты следуют друг за другом. Примерами могут служить тексты, финансовые данные, последовательность ДНК, или данные временных рядов. Различные архитектуры рекуррентных сетей позволяют работать с разной необходимой глубиной памяти, в следствие чего спектр задач, в которых такие сети могут применяться, достаточно велик. Возможность учитывать порядок входных данных делает рекуррентные сети идеально подходящими для задач машинного обучения, связанных с обработкой текстовых данных.

В RNN информация проходит циклически и с каждым циклом может затухать – запоминать более новую информацию, обновляя память. Когда принимается решение, сеть учитывает текущие входные данные, а также то, что она узнала на предыдущих слоях. Рисунок 3 иллюстрирует разницу в потоке информации между RNN и обычной нейронной сетью.

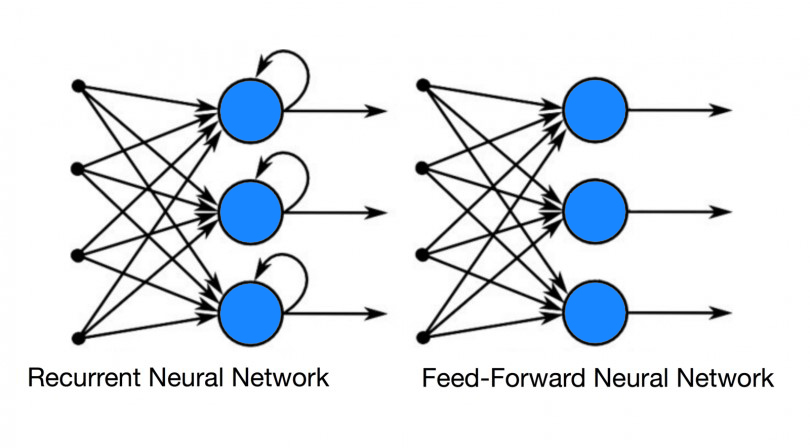


Рисунок 3 - Сравнение структуры рекуррентной нейронной сети (справа) и cети прямого распространения (слева) [12]

 Нейроны сети производят вывод, а также копируют этот вывод и зацикливают его обратно на свой вход. Так они получают информацию не только от предыдущего слоя, но и от самих себя предыдущего прохода. Это означает, что порядок, в котором подаются данные и обучается сеть, становится важным.

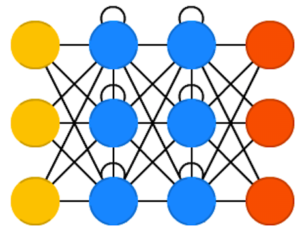


Рисунок 4 - Схема строения рекуррентной нейронной сети [13]

На рисунке 4 изображена простая архитектура рекуррентной нейронной сети. Желтым цветом отмечены нейроны входного слоя, красным – выходного слоя. Синие нейроны позволяют сети носить название рекуррентной – их выход поступает на свой же вход. Именно такая архитектура используется в исследованиях [14] [15].

Большой проблемой при работе сетей RNN является проблема исчезающего градиента, которая заключается в быстрой потере информации с течением времени. Сети способны хранить только самую недавнюю информацию, что не всегда является достаточным для данной задачи. Допустим, мы хотим предсказать последнее слово в тексте “*Я вырос в России… Мой родной язык русский*”. Ближайший контекст предполагает, что последним словом будет называние языка, но, чтобы установить, какого именно языка, нам нужен контекст «России» из более отдаленного прошлого. Таким образом, разрыв между актуальной информацией и точкой ее применения может стать очень большим, что не всегда допустимо. В таких задачах необходимо уметь сохранять более глубокую память.

**1.2.3.2 Рекуррентные сети с долгой краткосрочной памятью**

Сети с долгой краткосрочной памятью(Long Short Term Memory, LSTM) стараются решить вышеупомянутую проблему RNN потери информации. Для этого в сети используются фильтры и клетки памяти.

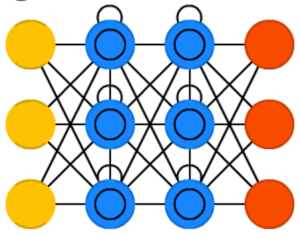


Рисунок 5 - Схема строения рекуррентной нейронной сети с долгой краткосрочной памятью [13]

Память в LSTM называется ячейками (синие нейроны на рисунке 5 имеют в себе черный круг, символизирующий внутреннюю ячейку памяти). Помимо стандартной для рекуррентных нейронов передачи данных - принимают в качестве входных данных текущий входной параметр и предыдущее состояние - они имеют специальное внутреннее строение, что отличает их от обычной рекуррентной сети. Внутри у каждой клетки памяти есть три фильтра: входной, выходной и забывающий, которые решают, какую память сохранить и какую стереть.

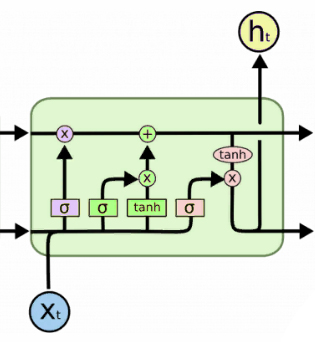


Рисунок 6 - Ячейка LSTM в развертке

На рисунке 6 изображена ячейка LSTM в развертке. Обозначим - входное значение (голубой кружок), – выходной вектор (желтый кружок). Ячейка состоит из нескольких компонентов, разделенных на три группы по цвету. Входной фильтр (компоненты зеленого цвета) определяет, сколько информации из предыдущего слоя будет храниться в клетке. Выходной фильтр (компоненты розового цвета) определяет, сколько информации получат следующие слои. Ну и забывающий фильтр (компоненты фиолетового цвета) – какую часть информации можно отсечь. Целью этих фильтров является защита информации внутри. Затем они объединяют предыдущее состояние, текущую память и входной параметр. Существуют различные модификации этой «классической» схемы ячейки. Сети LSTM способны научиться создавать более сложные структуры, чем классические рекуррентные сети. Решение задачи распознавания речи с использованием этой архитектуры было применено в исследовании [16].

**1.2.3.3 Двунаправленные рекуррентные сети**

Двунаправленные RNN(Bidirectional Recurrent Neural Networks, BRNN) основаны на той идее, что выход в момент времени t может зависеть не только от предыдущих элементов в последовательности, но и от будущих. Другими словами, сеть будет рассматривать данные как две последовательности – в прямом и в обратном направлении. Двунаправленные рекуррентные нейронные сети (BRNN) соединяют два скрытых слоя, работающих в противоположных направлениях, с одним выходом, позволяя им получать информацию как из прошлых, так и из будущих состояний. BRNN разбивает нейроны классической рекуррентной сети на два направления, одно для прямых состояний (положительное направление времени), а другое для обратных состояний (отрицательное направление времени). Выходы из прямых состояний не связаны со входами обратных состояний, и наоборот. Это приводит к общей структуре, которую можно увидеть на рисунке ниже (сеть развернута на 4 временных шага).

Например, если нужно предсказать недостающее слово в середине последовательности (предложения), то нужно учитывать и левый, и правый контекст. Двунаправленная рекуррентная нейронная сеть представляет собой две рекуррентные сети, уложенные друг на друга. Без обратных состояний эта структура упрощается до обычной однонаправленной прямой RNN. Если прямые состояния исключены, получается классическая рекуррентная сеть с перевернутой временной осью.

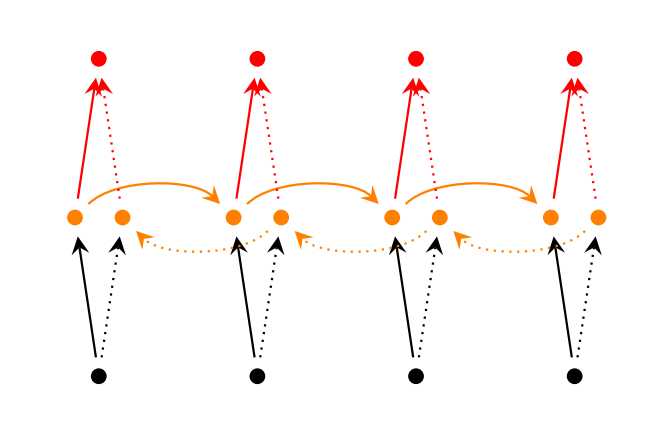


Рисунок 7 - Общая структура двунаправленной рекуррентной нейронной сети [17]

На рисунке 7 изображена общая структура двунаправленной рекуррентной нейронной сети (BRNN). В процессе её обучения прямые и обратные состояния обрабатываются сначала в прямом проходе (на рисунке сплошной линией изображено положительное/прямое направление), выходные нейроны вычисляются последними (на рисунке пунктирной линией изображено отрицательное/обратное направление). При обратном проходе происходит обратное: сначала обрабатываются выходные нейроны, затем передаются состояния вперед и назад. Выход вычисляется на основе скрытого состояния обоих сетей RNN - веса обновляются только после завершения прямого и обратного проходов.

Двунаправленная рекуррентная нейронная сеть находит свое применение в задачи распознавания речи во многих успешных исследованиях [18].

**1.2.3.4 Свёрточная нейронная сеть**

В последнее время для решения задачи распознавания речи обретает популярность [19][20][21] метод, использующий свёрточные нейронные сети. Архитектура сети изображена на рисунке 8.

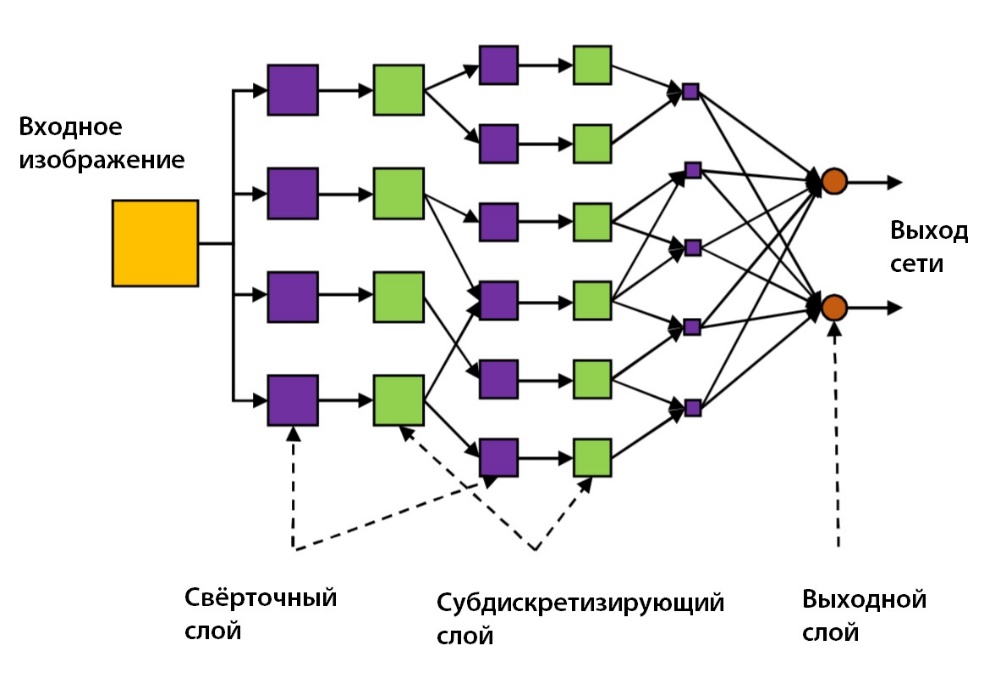


Рисунок 8 - Архитектура свёрточной нейронной сети

Для описания модели введем обозначения:

* - слой нейронной сети, где – количество слоев в сети;
* - количество скрытых слоёв в сети;
* - количество карт признаков (фильтров) на слое ;
* - -ая карта признаков на слое .

Карта признаков является тензором и представляет собой выход после каждого слоя сети. К примеру, при применении N ядер (свёрток) к одному входному изображению получим N карт признаков. Обычно происходит чередование свёрточных и субдискретизирующих (пуллинга) слоёв. Тогда свёрточные слои стоят на нечётных позициях , слои субдискретизации на четных позициях .

Опишем устройство *свёрточного* слоя. Будем рассматривать слой , где принимается нечётный числом . Тогда для карты признаков введем следующие обозначения:

* – свёртка (ядро, фильтр), применяемая к карте признаков слоя , на слое с картой признаков ;
* – пороговые значения, которые присоединяются к карте признаков на слое;
* – список всех карт признаков слоя , которые соединяются с картой признаков слоя.

Таким образом, карта признаков свёрточного слоя будет вычисляться следующим образом:

, (7)

где оператором обозначена математическая операция двумерной свёртки. Пример вычисления изображен на рисунке 9.

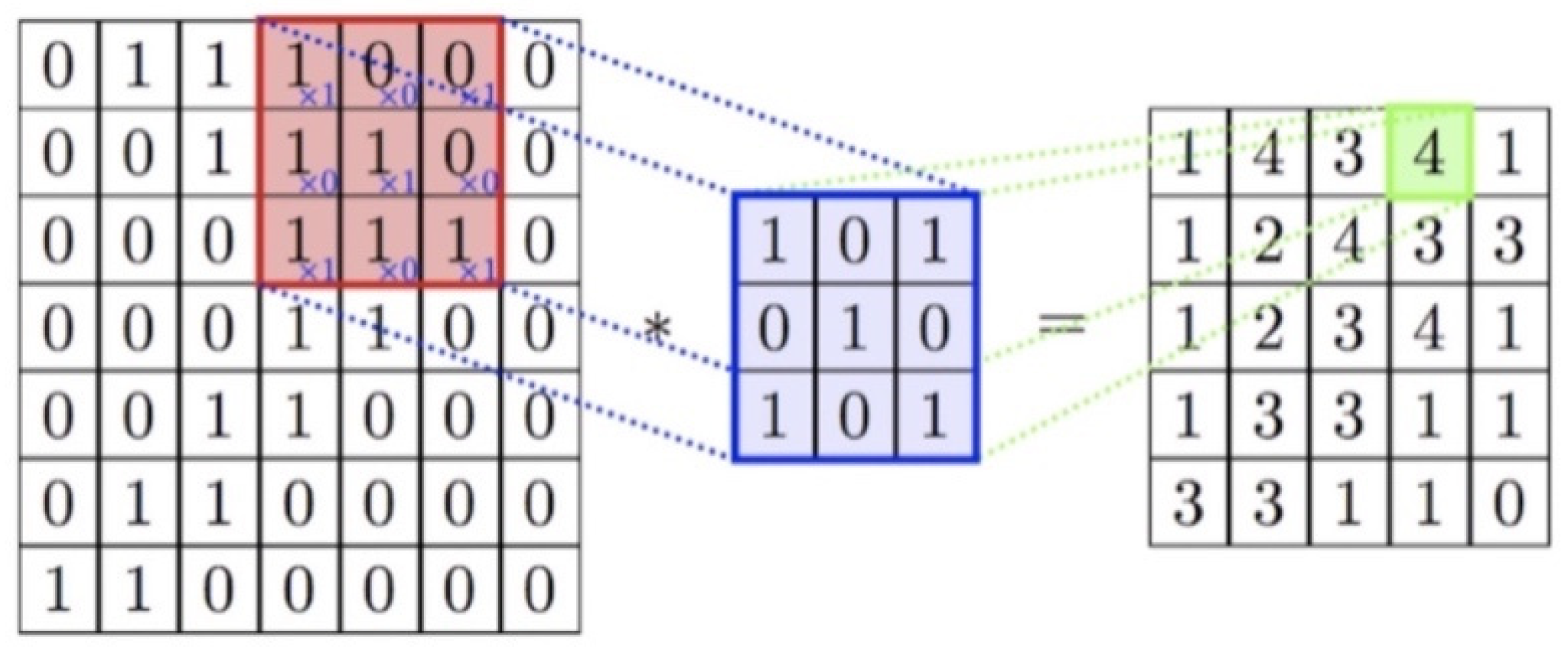


Рисунок 9 - Двумерная свёртка карты признаков (слева) и ядра (голубая матрица) [22]

Предположим, что размер входных карт признаков равен , а размер применяемой к ним свёртки равен . Тогда размер выходной карты признаков вычисляется как:

(8)

Подробная схема устройства слоя изображена на рисунке 10. Входная карта признаков представлена зеленым квадратом слева, свертка бежевым кругом, результат свертки – желтым квадратом. Затем, в случае многомерного фильтра (например, в случае работы с многоканальным изображением, когда каждый канал обрабатывается отдельно), результаты суммируются по формуле выше и применяется функция активации. Иначе – желтый квадрат уже является ответом.

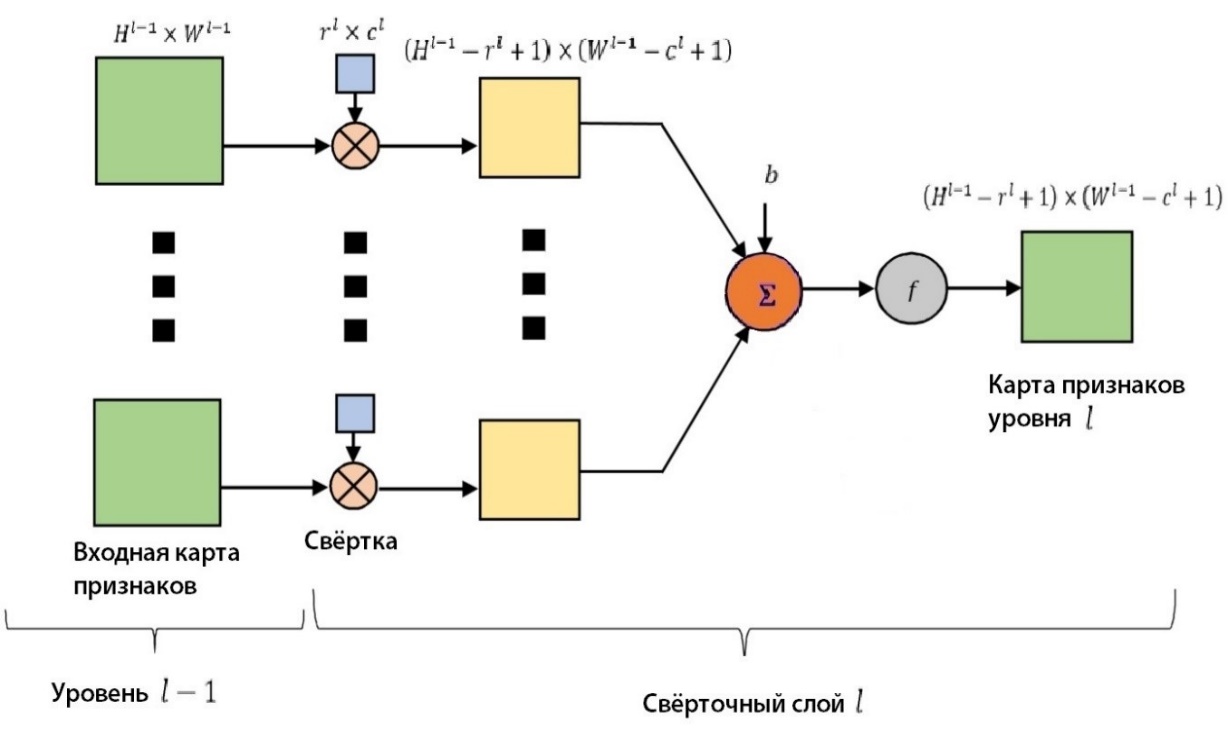


Рисунок 10 - Схема свёрточного слоя l

Введём в рассмотрение *субдискретизирующий* слой (пуллинговый). Его основная цель – уменьшение размерности входной карты признаков. В свёрточной нейронной сети номер субдискретизирующего слоя принято принимать чётный числом, то есть .

Разделим карту признаков -ого слоя на непересекающиеся блоки размером пикселя (Для простоты описания используется размер , однако на практике может использоваться и отличающийся размер). Затем просуммируем значения четырёх пикселей в каждом блоке и в результате получим матрицу . Её элементами будут являться соответствующие значения сумм. Таким образом, формула для вычисления имеет следующий вид:

(9)

Картой признаков субдискретизирующего слоя будет являться полученная матрица . При этом вместо суммирования может использоваться любая другая функция (взятие максимума, взятие среднего, суммирование, разность и т.д.). Пример вычисления субдискретизирующего слоя с функцией взятия максимума и разбиением на блоки представлен на рисунке 11.

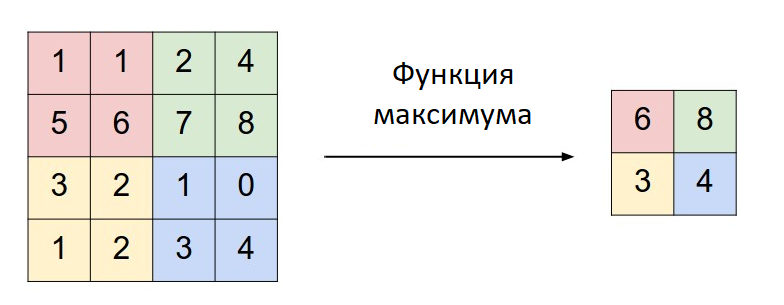


Рисунок 11 - Вычисление пуллингового слоя с функцией максимума и блоками 2×2 [23]

Таким образом, размер карты признаков субдискретизирующего слоя при блоках будет равен:

(10)

Существуют модификации слоя субдискретизации. Например, при разбиении карты признаков на блоки, последние могут перекрываться. Таким образом, можно задавать шаг смещения следующего блока относительно предыдущего. Другая модификация заключается в добавлении дополнительной «пустой рамки» к карте признаков. Так, при выполнении операции над блоком, оригинальные (не пустые) ячейки карты признаков могут вносить больший вклад в результат. Также, из-за добавления дополнительных ячеек выходная карта признаков будет иметь тот же размер, что и входная.

Выходной слой имеет номер , и состоит из нейронов. Он представляет собой слой обычного многослойного персептрона. Формула для расчета значения выходного нейрона :

, (11)

где

– фильтр, применяемый к карте признаков последнего свёрточного слоя для получения перехода к нейрону выходного слоя (матрица весовых коэффициентов);

– пороговое значение, добавляемое к нейрону (коэффициент сдвига).

Таким образом, выходом свёрточной нейронной сети является вектор следующего вида:

(12)

Для применения свёрточных сетей в задаче распознавания речи первоначально используется спектральное представление звукового потока.

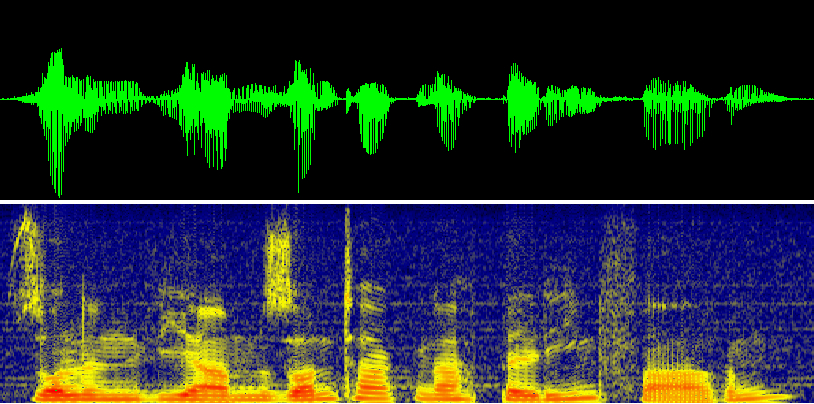


Рисунок 12 - Изображение звуковой дорожки и соответствующего ей спектрального представления [24]

На основе полученного спектрального изображения (на рисунке 12 изображена звуковая дорожка и соответствующее ей спектральное представление) стандартная свёрточная сеть для обработки спектрального представления сигнала выдает готовый распознанный результат.

**1.2.3.5 Другие архитектуры**

Также построение нейронной сети для может происходить без использования в основе какой-либо определенной архитектуры. Примером этого может быть определение оптимальной архитектуры нейронной сети экспериментально в процессе работы генетического алгоритма. Такой подход также встречается в некоторых методах распознавания речи [25] [26].

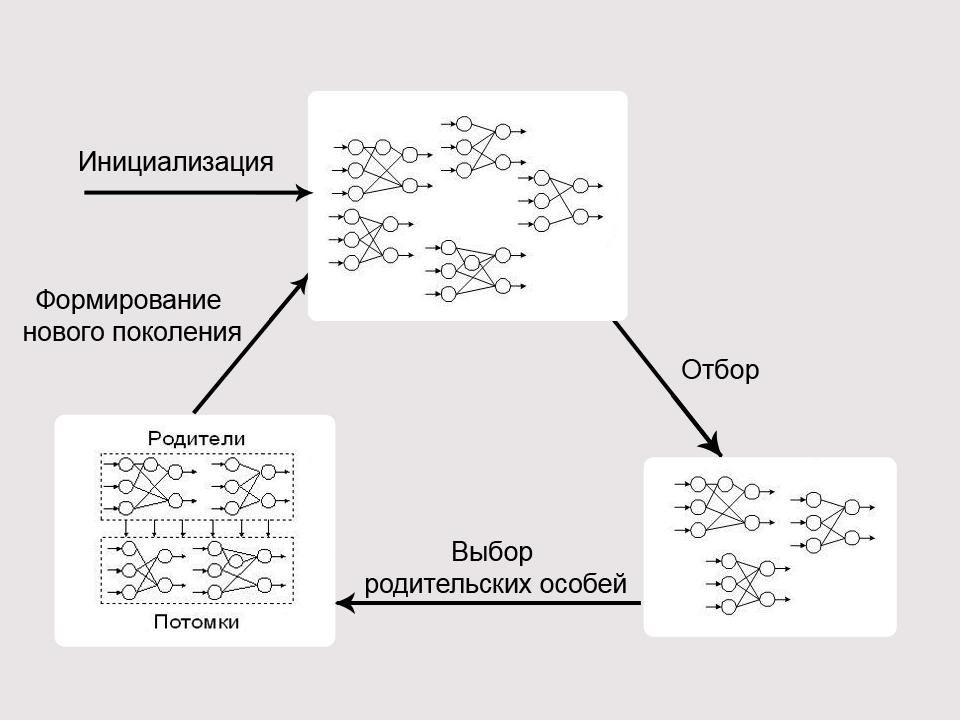


Рисунок 13 - Схема нейроэволюционного алгоритма

При использовании генетических алгоритмов создаются правила отбора, позволяющие определить, лучше или хуже справляется новая нейронная сеть с решением задачи. Кроме того, определяются правила модификации нейронной сети. В результате этого на основе эволюционного алгоритма выявляется оптимальная архитектура сети.

Генетический алгоритм заключается в параллельной обработке множества альтернативных решений и постепенном отсеивании сильно неоптимальных. Процесс поиска сосредоточен на наиболее перспективных, их развитие продолжается. Алгоритм работы алгоритма изображен на рисунке 13. Обычно для работы генетического алгоритма структура нейронной сети каким-то образом кодируется в массив значений (хромосому). Популяцией будет являться набор нескольких хромосом (решений, нейронных сетей). Эволюция популяций - это последовательность поколений, в которых хромосомы (сети) изменяют свои признаки, чтобы каждая новая популяция наилучшим способом приспосабливалась к внешней среде (лучше решала поставленную задачу). Таким образом, в конце получим наиболее приспособленную особь (нейронную сеть с самой высокой точностью).

**1.3 Классификация особенностей речи, затрудняющих её распознавание**

Несмотря на хороший прогресс, достигнутый в этой области, процесс распознавания речи по-прежнему сталкивается с множеством проблем, большинство из которых связано с тем, что речь может сильно отличаться. К наиболее распространенным особенностям речи, вызывающим ее вариативность, относятся:

* вариация говорящего - одно и то же слово по-разному произносится разными людьми в зависимости от возраста, пола, анатомических особенностей, скорости речи, эмоционального состояния говорящего и диалектных вариаций;
* фоновый шум: окружающая среда может добавлять шум к сигналу. Даже сам говорящий может добавить шум своей манерой речи;
* произношение слова (влияние интонации и ударения на слоги);
* характер речи: непрерывная речь представляет из себя поток трудно сегментирующихся сливающихся звуков, между словами редко бывает перерыв. Из-за этого очень сложно определить отдельные слова;
* неограниченный словарный запас – повседневная речь иногда содержит слова из других языков;
* другие внешние факторы: положение микрофона по отношению к говорящему, направление и настройка микрофона и многие другие.

**1.4 Известные сервисы распознавания речи**

Системы распознавания речи в наше время наиболее часто встречаются в виде интеграции в цифрового ассистента (виртуального цифрового помощника, Virtual Digital Assistant). Это полноценная система, принимающая информацию в виде звукового сигнала, содержащего речь человека, и возвращающая полноценный не бессмысленный ответ, а также способная выполнять определенные заданные действия (по планированию графика, выполнению ежедневных дел, поиска информации). Одной из ключевых функций цифрового помощника является распознавание человеческой речи (обычно записывающейся на микрофон). Он может быть интегрирован в мобильные устройства или компьютеры в качестве дополнительной программы или в так называемые «умные колонки», которые представляют собой смесь акустической системы и микрофонов работы с информацией Speech-to-speech. В таблице 1 представлены наиболее популярные бренды, остающиеся примерно на одинаково высоком уровне качества работы. Во всех компаниях цифровой ассистент имеет закрытый исходный код. В некоторых случаях компании лишь упоминают общие методы их работы.

Таблица 1 - Список лидирующих цифровых помощников на мировом рынке

|  |  |
| --- | --- |
| Компания (бренд) | Цифровой помощник |
| Amazon | Alexa |
| Google | Google Assistant |
| Apple | Siri |
| Microsoft | Cortana |
| Яндекс | Алиса |
| Mail | Маруся |
| Xiaomi | 小爱同学 |

Также существуют менее популярные программные компоненты, не предусматривающие интегрирование в цифровых помощников. К ним относятся программы, представленные в таблице 2.

Таблица 2 - Сравнение существующих программных компонентов по распознаванию речи

| Программа | Достоинства | Недостатки |
| --- | --- | --- |
| Dragon Naturally Speaking | * Удобство в работе * Поддержка распознавания аудиофайлов (но ограниченная) * Возможность повышения качества работы за счет обучения в процессе распознавания | * Невысокое качество распознавания без обучения * Отсутствие поддержки русского языка |
| Braina | * Простота * Поддержка множества языков, в том числе русского | * Нестабильное качество распознавания, которое в среднем является неприемлемо низким * Невозможность непосредственной работы с аудиофайлами * Не поддерживается обучение |
| Voco Professional | Пренебрежимо малы в связи с крайне низким качеством распознавания | Крайне низкое качество распознавания речи |
| Real Speaker | Обещают качество работы 90-100% | Пока только в демо-режиме |
| Speechlogger и Speechpad | Доступность | * Невысокое качество распознавания речи * Необходимость использования сторонних средств («виртуального кабеля») |
| Сервис Go Transcribe | Высокое качество | Платный (98$ за 10 часов речи), но качество превосходит стоимость |

Ни одна из них не представлена в виде открытого исходного кода, а также не описывает даже примерный принцип работы. Каждая из них имеет собственные достоинства и недостатки [27].

**1.5 Выбор метода распознавания речи**

На основе проведенного анализа можно составить следующую таблицу для описанных методов. Таблица 3 включает в себя достоинства и возможные недостатки, а также пометку, будет ли метод использоваться в данной работе.

Таблица 3 - Сравнение и выбор используемых в работе методов

| Метод | Достоинства | Недостатки | Будет использоваться |
| --- | --- | --- | --- |
| Скрытые Марковские модели | Очень хорошее качество работы | Нужно строить модель для каждого звука/слова | Нет |
| N-граммы | Простота | Используется для предугадывания на основе статистических данных, но не распознает | Да |

Продолжение таблицы 3

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Достоинства | Недостатки | Будет использоваться |
| Рекуррентная сеть | Обладает памятью | Память может быть недостаточно глубокой | Да |
| Рекуррентная сеть с долгой краткосрочной памятью | Обладает более долгой памятью | Не выявлено | Да |
| Двунаправленная рекуррентная сеть | Обрабатывает последовательность с двух сторон | Не выявлено | Да |
| Свёрточная сеть | Хорошо выделяет признаки из изображения | Не обладает памятью | Да |

Для распознавания речи будут использоваться несколько методов. Некоторые из них будут сравниваться и исключаться, другие будут дополнять систему. Было принято решение не использовать скрытые Марковские модели из-за того, что для их построения требуются специфичные обучающие данные.

**1.6 Известные методы идентификации по голосу**

Существует два типа систем идентификации говорящего:

* Текстозависимая. Если произнесенный текст должен быть одинаковым для регистрации диктора в базе и для идентификации, то этот подход называется текстозависимым распознаванием. В такой системе произносимое ключевое слово/фраза могут быть либо общими для всех дикторов, либо уникальными (например, уникальный пароль пользователя);
* Текстонезависимая. Независимые от текста системы чаще используются для идентификации диктора, поскольку они требуют меньшего внимания со стороны говорящего. В этом случае текст при регистрации и идентификации отличается, система считается инвариантна к речи.

Задачу идентификации человека по голосу, независимо от выбранного пути ее решения, разбивают на следующие основные этапы:

1) Извлечение признаков из аудио сигнала,

2) Построение модели диктора, на основе полученных на предыдущем шаге признаков.

Процесс определения человека по голосу из представленных в системе образцов голосов, во всех методах состоит в поиске более подходящего по модели диктора, на основании выбранных критериев.

Речевые сигналы - это звуковые колебания, которые распространяются в воздушной среде. Они характеризуются частотой (числом колебаний в секунду), интенсивностью (амплитудой колебаний) и длительностью. На протяжении всего речевого сигнала эти характеристики подвергаются изменениям, и фиксируются с помощью электронно-акустических приборов, таких осциллограф, спектрограф. Затем, при помощи аналого-цифрового преобразователя, аналоговый речевой сигнал переводится в цифровой, который на ЭВМ представлен в виде wav-файла, с которого уже происходит сбор речевых признаков.

**1.6.1 Выбор системы признаков**

Звуковой сигнал представляет собой упорядоченный массив значений амплитуды звука, к которому добавлен заголовок, содержащий в себе количество каналов, частоту дискретизации и другую информацию. Однако анализировать данные в таком виде не представляется возможным – они не содержат ключевых признаков, на основе которых метод сможет дать хороший результат. Существует несколько способов представления звукового сигнала в более удобной форме. Самыми распространенными способами представления аудиоданных и извлечения из них признаков является получение спектрограмм и получение мел-частотных кепстральных коэффициентов (Mel Frequency Cepstral Coefficients).

В задачах идентификации по голосу самым распространенным способом представления голосов являются MFCC. Рассмотрим подробнее их построение.

Исходный сигнал нарезают на перекрывающиеся фреймы (окна) небольшой длины (10-40 мс). Затем к получившимся фреймам применяют окно Хемминга (чтобы сгладить значения на границах фреймов и уменьшить утечку спектра), делают быстрое преобразование Фурье (FFT):

(13)

Таким образом получают спектральную плотность мощности (распределение мощности сигнала в зависимости от частоты, то есть мощность, приходящаяся на единичный интервал частоты).

Затем специальной «гребёнкой» фильтров, расположенных равномерно по мел-шкале (рисунок 14) делают мел-спектрограмму. При этом, каждый мел-фильтр - это треугольная оконная функция, которая суммирует количество энергии на определённом диапазоне частот и тем самым дает мел-коэффициент. Полученные результаты логарифмируют. Считается, что таким образом понижается чувствительность коэффициентов к шумам. В конце применяют дискретное косинусное преобразование (DCT) — алгоритм сжатия данных.

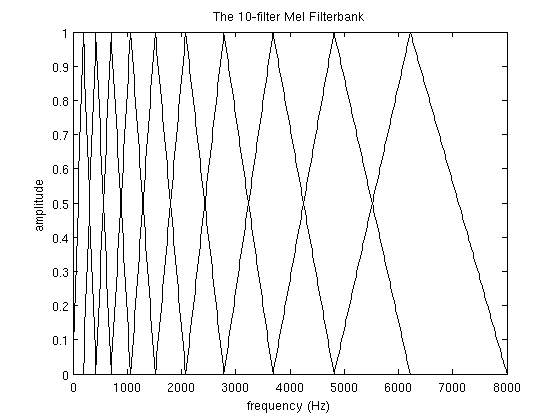


Рисунок 14 - Десять мел-фильтров - треугольных оконных функций

Полученные таким образом коэффициенты представляют из себя некую сжатую характеристику фрейма. При этом, поскольку фильтры, которые мы применяли, были расположены в мел-шкале, коэффициенты учитывают особенности восприятия человеческого уха, а значит несут больше полезной информации. Принято использовать от 13 до 25 MFCC на фрейм.

Так как индивидуальность голоса часто зависит от скорости речи, особенностей произношения и ускорения, будем также учитывать производные. Визуализация примера полученных коэффициентов изображена на рисунке 15.

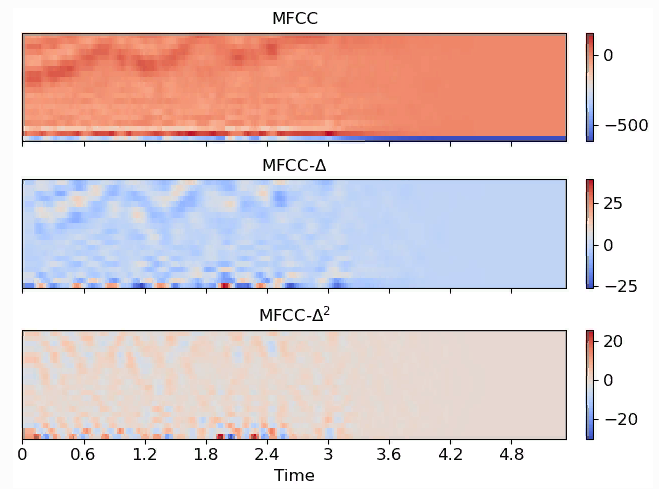


Рисунок 15 - Визуализация MFCC и дельта-функций первого, второго порядков [28]

Практически во всех исследованиях работа ведется с MFCC, а также иногда дополняется набором из первых и вторых дельта-функций – локальных оценок производных (в дискретном случае производная функции равна разности значений функции в последовательных точках).

**1.6.2 Выбор классификатора признаков**

Современные системы голосовой идентификации и верификации работают в двух режимах:

* режим обучения. Выделяются характерные признаки голоса человека, формируется его голосовая модель (голосовой отпечаток) на основе этих признаков и выполняется сохранение модели в базе данных;
* рабочий режим. Выделяются характерные признаки голосового сигнала человека и выполняется поиск в базе данных голосовой модели, соответствующей этим признакам (идентификация личности).

**1.6.2.1 GMM**

В настоящее время наиболее результативным подходом к решению задач текстонезависимой идентификации личности является построение голосовых моделей на основе моделей гауссовых смесей (Gaussian Mixture Model, GMM) [29] [30]. Сами модели строятся на основе некоторого набора голосовых признаков. Наиболее распространенным методом построения голосовых признаков является формирование вектора мел-частотных кепстральных коэффициентов (Mel-Frequency Cepstral Coefficient, MFCC) из голосовой записи [29] [31].

Однако, несмотря на достаточно хорошие результаты работы в «лабораторных условиях», метод GMM-MFCC не может быть использован для построения реальных систем голосовой идентификации, так как система имеет сильную зависимость результатов от вида обучающего материала (на основе которого составляется база голосовых моделей и фоновая модель), и условий записи голосового сигнала. Также недостатком являются потребность в большом количестве обучающего материала.

**1.6.2.2 Сеть Кохонена**

В качестве альтернативного способа сравнения голосовых отпечатков может использоваться нейросетевое сравнение при помощи самоорганизующейся сети Кохонена. Сеть обучается без учителя и также имеет две фазы работы – обучение и идентификацию. В процессе обучения обучающий алгоритм подстраивает веса сети так, чтобы получились согласованные выходные векторы. Таким образом, процесс обучения выделяет статистические свойства обучающего множества и группирует сходные векторы в классы. Предъявление на вход вектора из данного класса дает определенный выход. В ходе идентификации будет происходить вычисление сумм в узлах сети после подачи вектора на вход, затем активация найденного узла и получение классифицируемого вектора.

**1.6.2.3 Глубокие нейронные сети**

Кроме сети Кохонена существующие работы [32] основаны на использовании в качестве классификатора глубокой нейронной сети и нейронной сети со свёрточным слоем [33]. Она имеет несколько скрытых слоев, входной слой того же размера, что и вектор признаков, и выходной слой с количеством нейроном, равным количеству зарегистрированных дикторов в системе. Несмотря на различные способы оптимизации способа обучения сети, такой подход делает невозможным быстрое добавление новых дикторов в систему. Для этого необходимо не только заново обучить всю сеть, но и изменить её архитектуру (как минимум – увеличить выходной слой).

**1.7 Особенности аудиозаписи, затрудняющие идентификацию по голосу**

В отличие от аудиозаписи речи, которую нужно распознавать, на аудиозапись голоса накладывается куда меньше ограничений для её использования в системе идентификации. Особенности произношения и дикции, особенности голоса в зависимости от возраста, пола, анатомических особенностей, скорости речи, эмоционального состояния говорящего необходимо стараться сохранить, ведь именно на этих особенностях и строится работа системы. Словарный запас не имеет значения, так как работа системы никак не зависит от произнесенной фразы.

Однако сохраняются сильные ограничения на фоновый шум и другие голоса в аудиозаписи. Их наличие может сильно ухудшить работу системы идентификации. Также повлиять на качество идентификации может звукозаписывающее устройство – чтобы этого избежать, желательно использовать одно устройство для записи голоса в обучающем и в тестовом режимах.

На качество работы системы практически не оказывает влияние имитация голоса. Эксперименты с профессиональными имитаторами показали, что успех такой подделки голоса невелик [34]. Как показали исследования France Telecom на большой базе данных, имитация почти не увеличивает вероятность ошибки определения личности. Это связано с тем, что имитаторы подделывают голос лишь внешне – его интонационный контур, дикцию и особенности произношения, но глубинные факторы, зависящие от голосовой формы тракта, размера гортани, особенностей ритма и определяющие индивидуальные свойства речи, имитировать не удается. Даже голоса близких родственников, близнецов будет иметь различие, как минимум, в динамике управления.

**1.8 Существующие программные решения и области применения голосовой идентификации**

Существуют готовые промышленные системы голосовой идентификации. Ограничение в том, что все сервисы платные и имеют закрытый исходный код. Качество работы систем не идеально и продолжает совершенствоваться в настоящее время. Интерес к задаче растет.

Несмотря на существование готовых решений, все системы остаются закрыты, их методы работы нигде не описаны и сохраняются в тайне. Описанные в этом разделе существующие методы имеют в открытых источниках лишь краткое описание принципов работы без деталей реализации.

Идентификация по голосу всё чаще используется в системах безопасности банков, телекомов и других организаций в виде компонента аутентификации, которая выступает дополнительным фактором парольной защиты. Пользователь должен произнести по телефону пароль, при этом система определяет правильность парольной фразы и дополнительно проверяет уникальный голосовой отпечаток данного пользователя.

Ярким примером является «Сбербанк России» («Сбер»), который внедрил в свою систему возможность записи голоса (сбор голосов начался еще в 2018 году с целью создания Единой биометрической системы (ЕБС) [35] и Единой системы идентификации и аутентификации (ЕСИА)) и возможность голосового подтверждения операций (технология внедрена с 21 января 2021 года [36]). Благодаря новой технологии организация надеется защитить клиентов от мошенничества. Также компания начала предлагать клиентам на добровольной основе сдать образец голоса через мобильное приложение «Сбербанк онлайн» или записать голосовой образец при звонке в call-центр. Предварительно нужно дать согласие на обработку персональных данных.

Созданная ЕСИА сейчас встроена в систему авторизации пользователей на портале Госуслуг и обеспечивает доступ граждан по его желанию и с его согласия к порталам Госуслуг. После сдачи биометрических данных любой житель страны может пользоваться услугами банков и государственных учреждений без личного присутствия. Биометрическую базу можно будет применять для дистанционной сдачи экзаменов, для прохождения транспортного контроля в аэропортах, для подтверждения права бесплатного проезда на общественном транспорте или оплаты проезда [37].

Из крупнейших решений на мировом рынке голосовой биометрии можно выделить следующих [38]:

* Agnitio (Испания),
* Auraya Systems (Австралия),
* Authentify (ОАЭ),
* KeyLemon (Швейцария),
* Nuance (США),
* ValidSoft (Великобритания),
* Verint Systems (США),
* VoiceTrust (Германия),
* VoiceVault (Великобритания).

Лидером рынка сейчас признается Nuance. Ее решения установлены в Аэрофлоте, а также она известна всем пользователям iPhone, так как Siri построена именно на технологиях Nuance.

**1.9 Выбор метода идентификации по голосу**

После описания методов необходимо их сравнить и выбрать те, которые будут использоваться в работе. Сравнение представлено в таблице 4.

Таблица 4 - Сравнение и выбор используемого в работе метода

| Метод | Достоинства | Недостатки | Будет использоваться |
| --- | --- | --- | --- |
| GMM | Наиболее результативен | Зависимость от условий записи голосового сигнала. потребность в большом количестве обучающего материала | Да, с модификацией |
| Сеть Кохонена | Неплохие результаты, быстрая работа | Нужно часто перестраивать систему | Нет |
| Нейронные сети | Неплохие результаты, быстрая работа | Нужно часто менять архитектуру и переобучать модель | Нет |

Среди описанных методов был выбран только один. При этом, метод будет использоваться с некоторой модификацией, которая будет описана в следующем разделе. Модификация позволит избавиться от недостатков метода в данной подзадаче идентификации по голосу.

**1.10 Постановка задачи и выводы**

В приведенном обзоре были описаны самые распространенные методы для распознавания речи и для идентификации говорящего по голосу. Установлено, что в настоящее время существует несколько систем, использующих различные методы распознавания речи, однако все исходные коды систем закрыты. Чтобы построить систему, в работе будут сравниваться различные методы распознавания. Какие-то методы будут исключать друг друга, другие, наоборот, дополнять собой систему. Основой распознавания выбрана нейронная сеть. В связи с этим ставится цель создания нейросетевой оптимальной системы для автоматического распознавания английской речи. Для достижения цели предполагается решить следующие задачи:

1. Получение размеченной обучающей выборки,
2. Изучение и разработка методов распознавания речи,
3. Выбор и разработка оптимальной архитектуры нейросети,
4. Изучение и проектирование оптимальной архитектуры системы.

Также в разделе был проведен анализ существующих типов систем голосовой идентификации и изучены подходы к решению задачи идентификации. На основе этого сделан вывод, что наиболее интересной и практически ценной является задача текстонезависимой идентификации. Также система должна иметь возможность быстро добавлять нового диктора в свою базу (без изменения архитектуры всей системы, которое может повлечь ухудшение качества или требовать большое время работы) и не требовать большого объема обучающих данных (нельзя требовать от нового диктора несколько часов его записанной речи).

Задача идентификации говорящего по голосу является в настоящее время очень актуальной и продолжает набирать популярность. Рынок нуждается в новых более качественных решениях. Все существующие системы идентификации диктора закрыты и не предоставляют описания принципа своей работы.

В данной работе идентификация по голосу будет строиться на основе использования модели гауссовых смесей. Чтобы система соответствовала требованиям, метод будет специально модифицирован. Для построения системы необходимо решить следующие задачи:

1. Получение обучающей выборки,
2. Изучение и разработка метода идентификации диктора,
3. Проектирование и разработка оптимальной архитектуры выбранной модели,
4. Тестирование работы реализованной системы.

Цель работы состоит в создании двухфакторной (свойства и знания) системы аутентификации с использованием нейросетевой системы автоматического распознавания английской речи и системы текстонезависимой голосовой идентификации на основе моделей гауссовых смесей.

**2. Разработка методов и проектирование системы аутентификации**

В этом разделе приводится описание предлагаемого метода распознавания речи и метода голосовой идентификации, а также используемых метрик оценки качества работы. Задача разбивается на несколько подзадач, выделяются этапы. Приводятся способы решения поставленных подзадач.

**2.1 Обучающая выборка и входные данные**

Обучающая и тестовая выборки состоят из базы данных LibriSpeech. Она включает в себя несколько сотен часов английской речи с различными диалектами, записанными на разные устройства. Содержит более 300 голосов. Аудиофайлы чистые, без фонового шума, посторонних голосов и дополнительных артефактов.

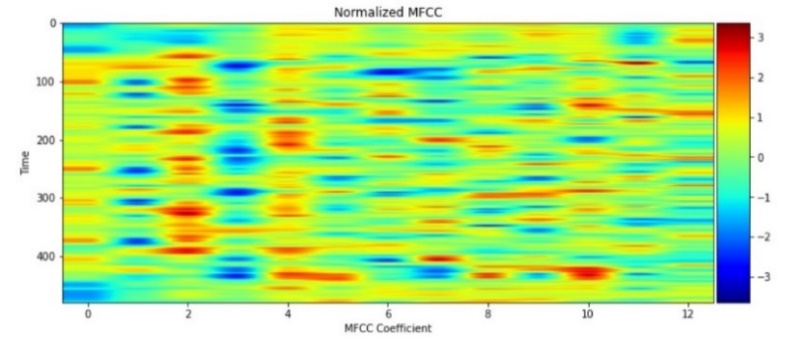
 

Рисунок 16 - Спектрограмма и мел-коэффициенты для одного аудио сигнала

Для построения системы распознавания речи будем представлять обучающие данные в виде спектрограмм и мел-коэффициентов – рисунок 16.

Для проведения идентификации по голосу будем дополнительно предварительно обрабатывать аудиофрагменты – удалять из них паузы и небольшие куски тишины, склеивать полученные фрагменты и нарезать на элементы с заданной длительностью для валидации и обучения GMM.

**2.2 Метод работы системы распознавания речи**

**[2.2.1 Постановка задачи обработки и распознавания речи](#page12)**

Задача распознавания речи заключается в автоматическом определении однозначной последовательности слов (или пустой последовательности), которая присутствует в исходном аудиофайле. Можно выделить несколько этапов в процессе распознавания:

1. получение аудиофайла (сюда может входить процесс записи речи на микрофон, аналогово-цифровые преобразования, транскодирование, получение файла через Интернет);
2. обработка речевого сигнала, очистка от шума, усиление/ослабление фоновых звуков;
3. получение акустических представлений (извлечение признаков), которые будут использоваться для разделения классов звуков речи (к примеру, построение спектограмм);
4. распознавание:
   * + 1. акустическая модель (encoder),
       2. языковая модель (decoder).

В данной работе подробно рассматривается последние два пункта – перевод аудиоданных в оптимальную форму представления, а также распознавание и определение текста, присутствующего на входных аудиоданных.

Последний этап чаще всего делят на два элемента – encoder, decoder. Грубо говоря, encoder – языковая модель – позволяет переводить сегменты звука в последовательность фонем. В то же время, языковая модель из последовательности фонем формирует слова. На рисунке 17 изображена схема работы метода.

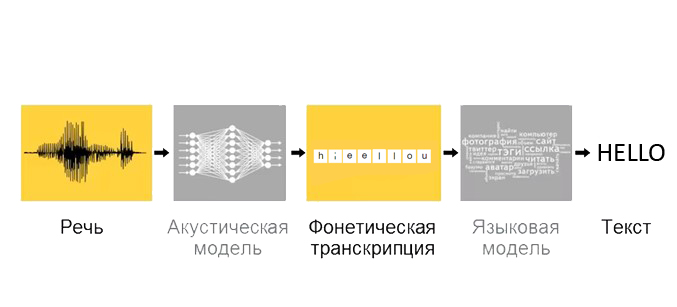


Рисунок 17 - Представление системы распознавания с разделением на encoder, decoder

Формально, будем считать, что входная последовательность признаков для акустической модели размерности должна на выходе языковой модели получить корректное распознавание в виде последовательности размерности . Тогда:

(14)

, (15)

где

и – функции, задаваемые акустической и языковой моделями;

– какое-то представление (последовательность морфем);

– условная вероятность верного распознавания.

**2.2.2 Ограничения**

Зададим определенные ограничения задачи распознавания. В качестве языка распознавания выбран английский, хотя, вообще говоря, сам алгоритм работы никак не зависит от языка, кроме как от количества классов разбиения для классификации, так как основывается на фонетическом распознавании. Обучение будет проходить на готовых датасетах без проведения этапа очищения от шумов, так как размер датасета и имеющиеся вычислительные мощности не позволяют проводить обработку по очищению данных.

[**2.2.3 Описание работы выбранных методов распознавания речи**](#page6) **для акустической модели**

Обучающие данные для акустической модели могут быть представлены в разной форме. Для выявления оптимального способа представления данных будет проведено сравнение работы нейронных сетей, обученных на данных, представленных в виде спектограмм, а также в виде мел-кепстральных коэффициентов (MFCC).

**2.2.3.1 CTC**

Коннекционистская временная классификация (CTC) – это тип выхода нейронной сети и связанная с ним функция оценки, предназначенная для обучения рекуррентных нейронных сетей.

Представим задачу распознавания речи более формально. Необходимо найти отображение входной последовательности (аудиофайла) к соответствующему выходу (транскрипция, последовательность фонем). С помощью CTC можно для заданного X найти выходное распределение по всем возможным Y без решения задачи сегментации входа.

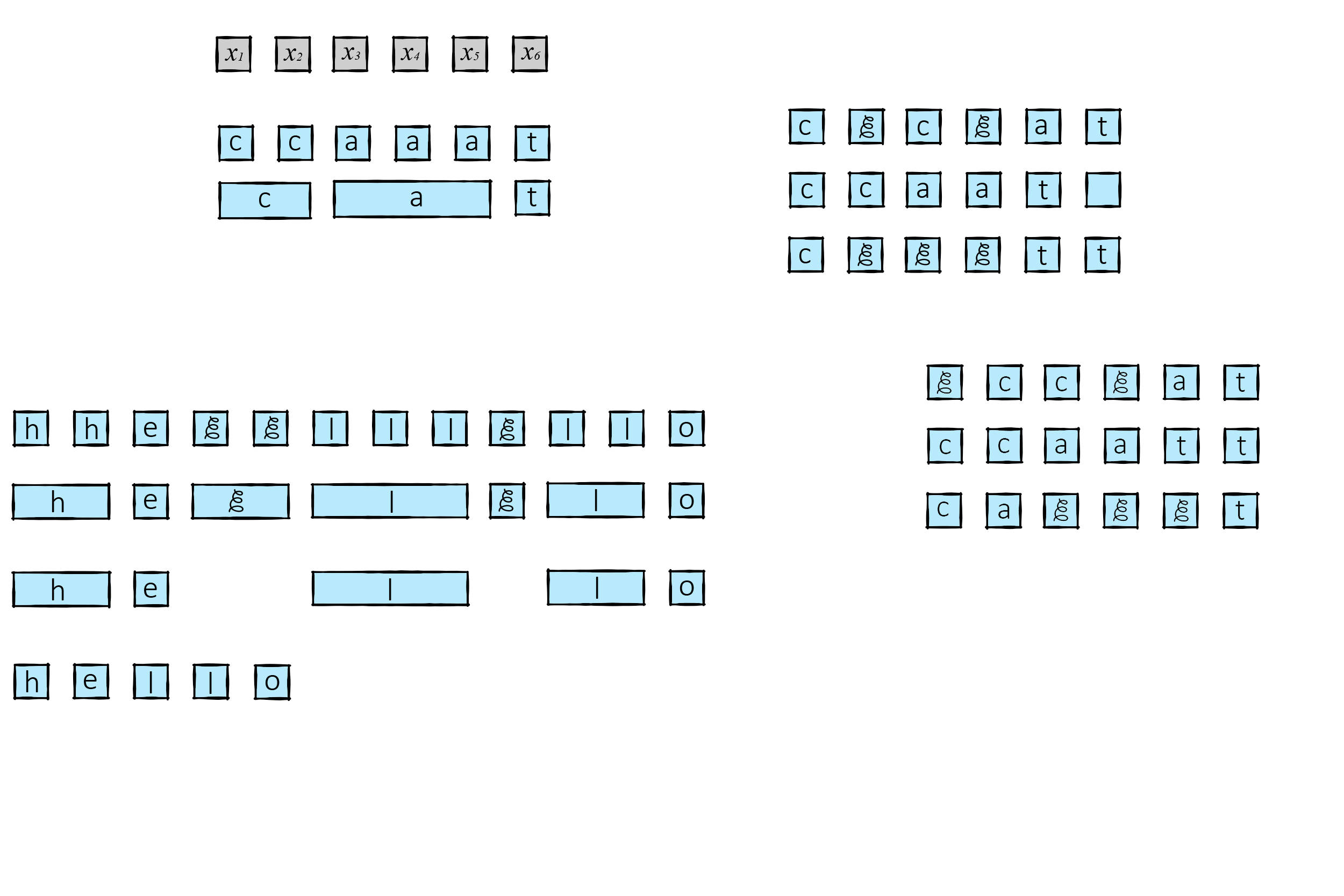


Рисунок 18 - Вектор X и свертка соответствующего ему ответа Y слова “cat”

Допущения:

1. Будем назначать выходной символ для каждого элемента входной последовательности (сделаем X и Y одной длины);
2. Будем сворачивать повторы символов в выходе Y для получения ответа (на рисунке 18 представлен пример для распознавания слова “cat”. Операция свертки переводит “ccaaat” в “cat”. Вектор X представляет входную последовательность);
3. Введем дополнительную «пустую фонему» в алфавит, чтобы корректно распознавать слова с удвоенной буквой (на рисунках ниже она обозначена как ℰ);
4. Если слово имеет две одинаковых буквы в ответе, то в выходе обязательно должна присутствовать «пустая фонема» между распознанными фонемами (на рисунке 19 представлена свертка слова “hello” с пустой фонемой. Сначала проходит операция свертки, затем удаляются символы ℰ, затем получаем ответ);
5. Соответствие X к Y «многие к одному»;
6. Длина Y не может быть больше, чем длина X.

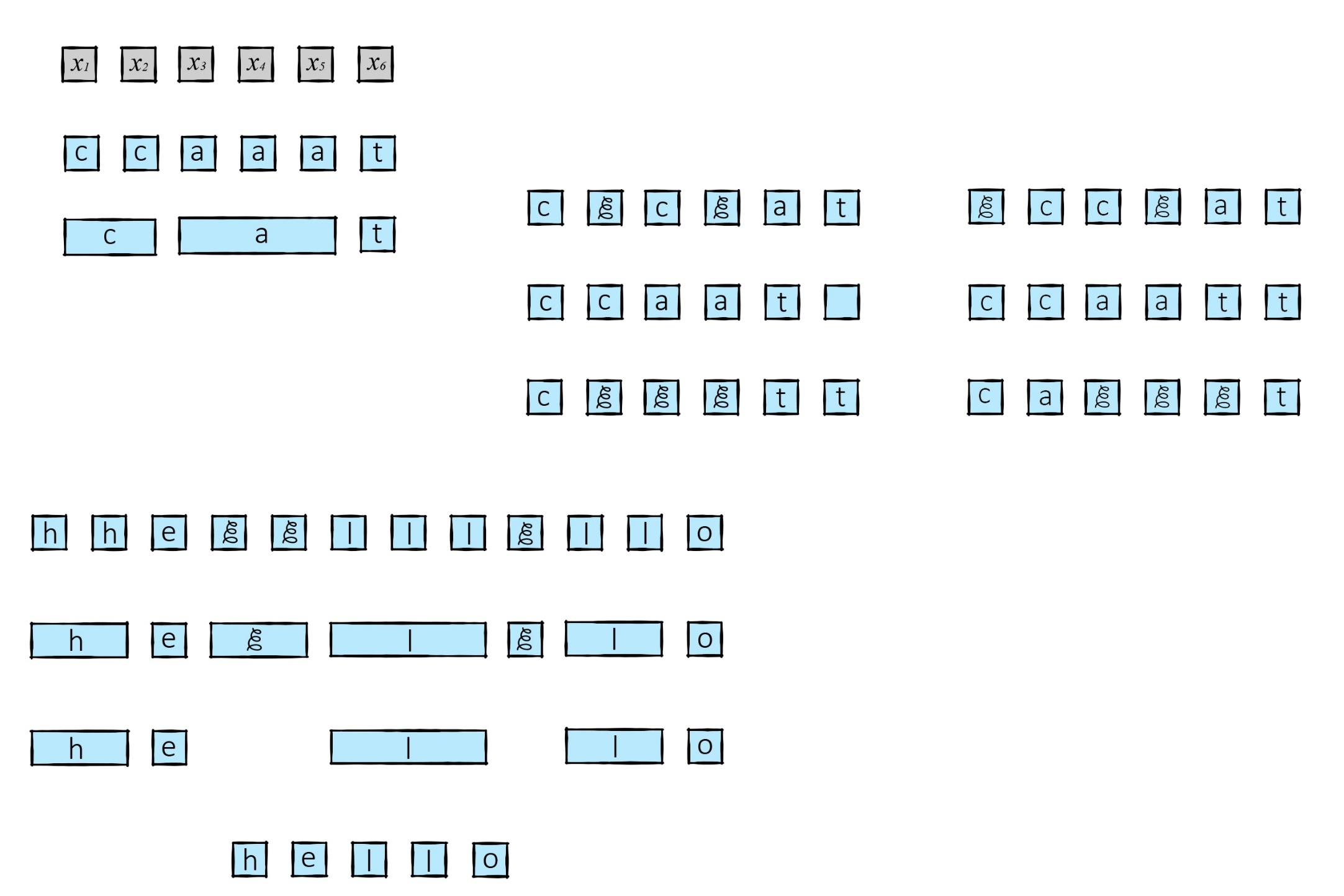


Рисунок 19 - Шаги алгоритма CTC сверху вниз. ℰ - пустая фонема

На рисунке 20 представлены примеры корректной и ошибочной работы алгоритма для слова “cat”. Слева для каждой распознанной последовательности алгоритм после свертки получит слово “cat”. Справа в первом случае работа алгоритма даст результат “ccat”, в третьем случае результат “ct”, а во втором случае ошибка произошла в первом допущении - выходной символ назначен не для каждого элемента входной последовательности.

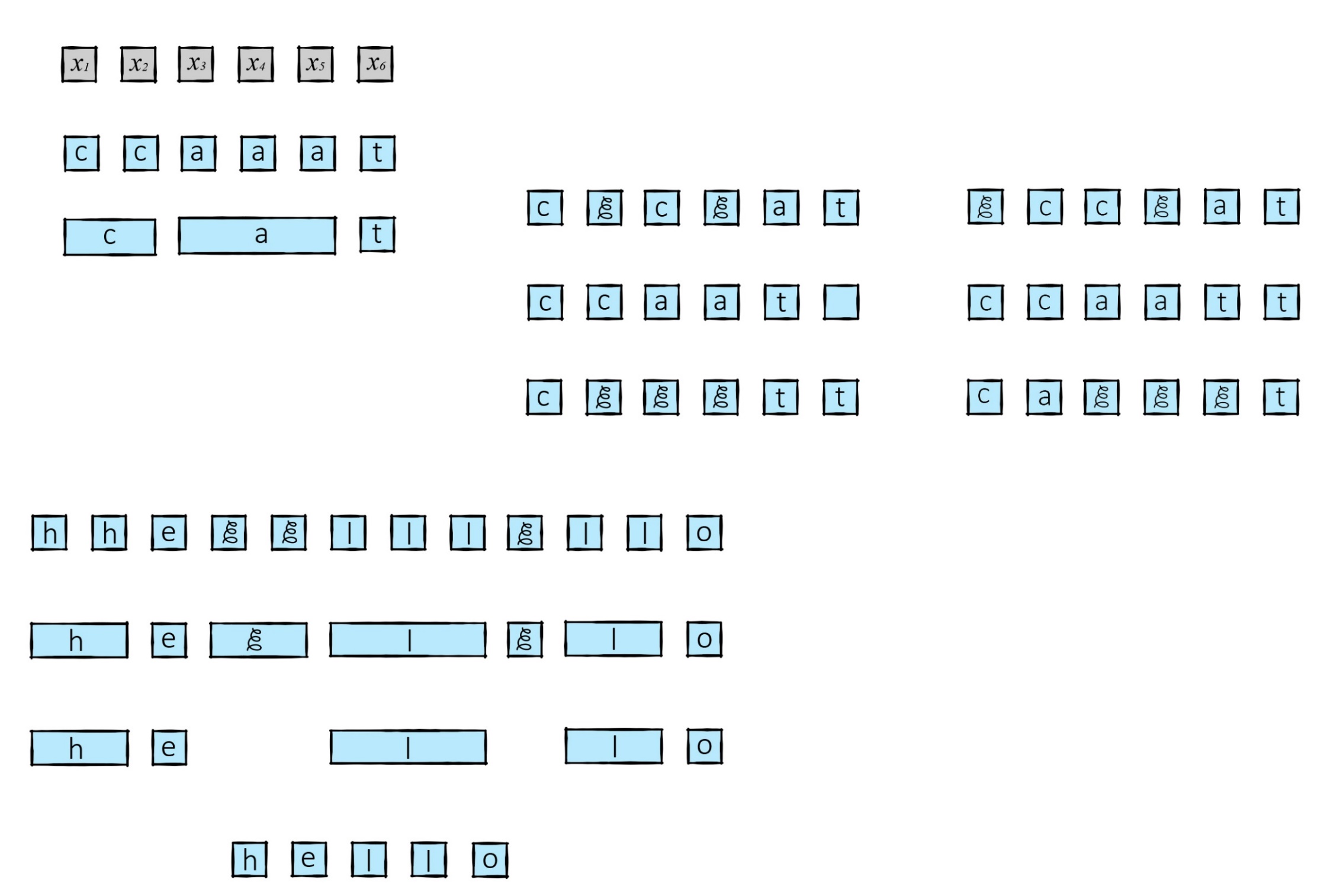
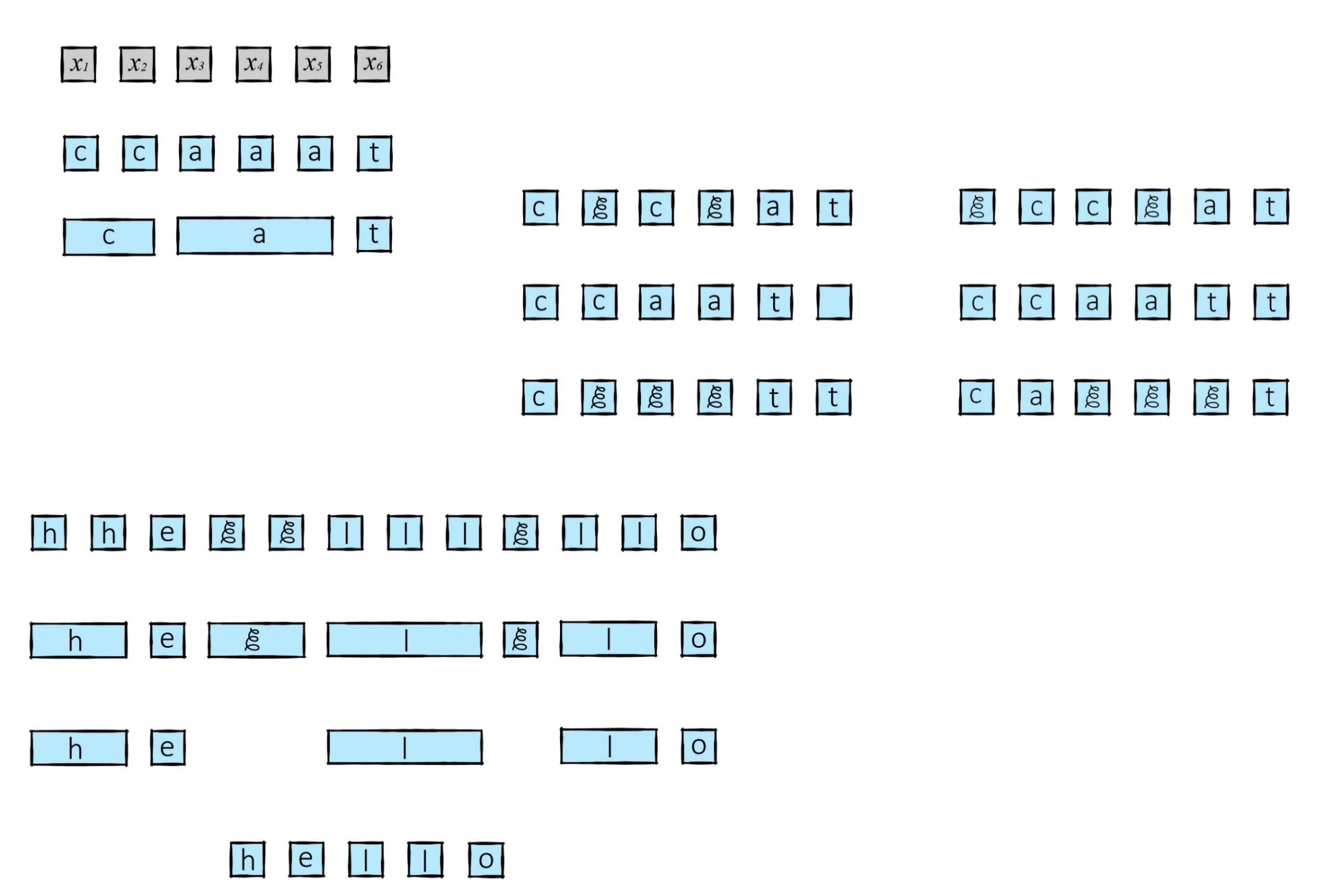


Рисунок 20 - Корректное (слева) и ошибочное (справа) распознавание для слова “cat”

Введем функцию потерь. Выход CTC представляет собой матрицу, где столбцы соответствуют временным шагам, а каждая строка соответствует букве в алфавите. Сумма вероятностей по столбцам равна 1. Используя матрицу оценок для каждого символа на каждом временном шаге, рассчитываем вероятность различных последовательностей выходов (используя динамическое программирование для облегчения вычислений). Затем суммируем по всем допустимым последовательностям. Таким образом, потеря рассчитывается путем суммирования всех вероятностей всех возможных выходных последовательностей текста:

, (16)

где

– условная вероятность CTC,

- сумма по набору допустимых распознаваний,

- вычисление вероятности для одного распознавания шаг за шагом.

Приведем пример и рассмотрим алфавит из 2 символов {a,b}. На рисунке 21 представлена матрица CTC для данного примера. Каждый кружок символизирует вероятность распознавания конкретной буквы на конкретном временном шаге. По строкам располагаются символы алфавита (последний символ является пустой фонемой), по столбцам – временные интервалы. Рассчитаем вероятности, для каждого возможного распознавания (пути).

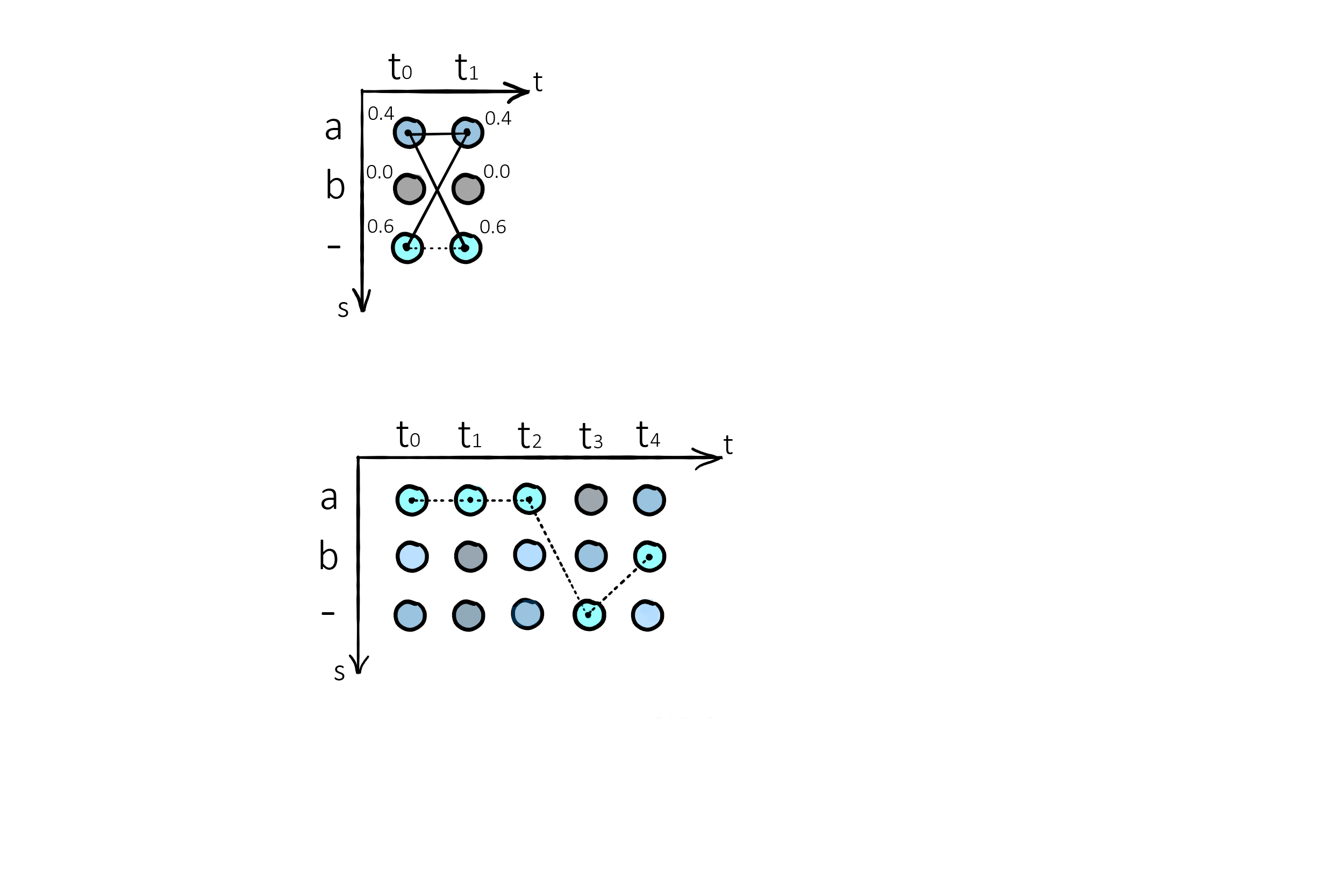


Рисунок 21 - Пример выходной матрицы. Более светлые ячейки для больших вероятностей

В приведенном выше примере оценка для пути “аа” равна 0,4·0,4=0,16, в то время как для “а-” она равна 0,4·0,6=0,24, а для “-а” - 0,6·0,4=0,24. Чтобы получить оценку для какого-то ответа Y мы суммируем вероятности (оценки) всех путей, соответствующих этому ответу. Предположим, что ответ Y - это “а“. Чтобы получить такой ответ, нужно вычислить все возможные пути длины 2 (потому что матрица имеет 2 временных шага), которые являются - ”аа“, ”а-“ и ”-а". Просуммируем вероятности каждого пути и получим: 0.4·0.4+0.4·0.6+0.6·0.4=0.64. Если ответ предполагается равным “”, то существует только один соответствующий путь, а именно “--”, который дает общую оценку 0,6·0,6=0,36.

Необходимо обучить NN присваивать высокую вероятность (в идеале, значение 1) для правильных классификаций. Поэтому будем максимизировать произведение вероятностей правильных классификаций для обучающего набора данных. Сформулируем эквивалентную задачу - минимизировать потерю обучающего набора данных, где потеря представляет собой отрицательную сумму логарифмических вероятностей. Для вычисления потери для одного ответа нужно вычислить вероятность, взять логарифм и поставить минус перед результатом.

**2.2.3.2 Декодирование CTC**

Рассмотрим теперь способ определения наиболее вероятного текста по выходной матрице вероятностей CTC. Простым и очень быстрым алгоритмом является лучшее декодирование пути, которое состоит из двух шагов:

1. вычислить наилучший путь, принимая наиболее вероятный символ за временной шаг;
2. отменить кодировку, сначала удаляя дубликаты символов, а затем удаляя все «пустые фонемы» из пути. То, что остается, представляет собой распознанный текст.

В качестве примера рассмотрим следующую матрицу CTC – рисунок 22.

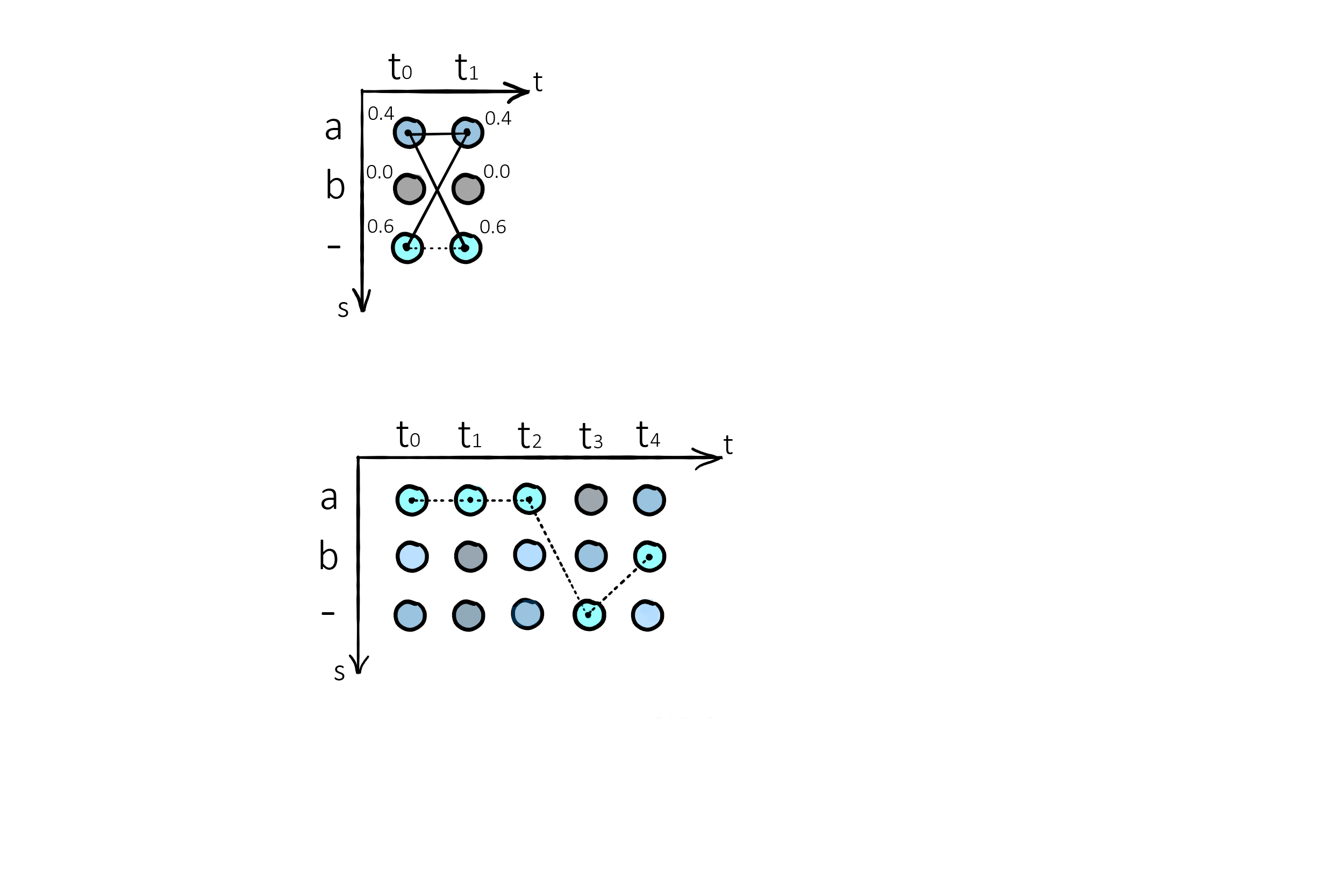


Рисунок 22 - Пример выходной матрицы. Пунктирная линия показывает наилучший путь

Алфавит содержит “a”, “b” и "-". Есть 5 временных шагов.  Применим алгоритм лучшего декодирования пути к этой матрице: наиболее вероятным символом в момент , и является “а”. Пустой символ имеет самый высокий балл в . В момент наибольшую вероятность имеет символ “b”. По алгоритму, получается путь “aaa-b". Теперь удаляем повторяющиеся символы, это дает “a-b“, затем удаляем пустой символ из оставшегося пути, это дает нам текст “ab”, который мы выводим как распознанный ответ.

Лучшее декодирование пути состоит в поиске символа с самым высоким баллом для каждого временного шага. Если у нас есть символы C и временные шаги T, то алгоритм имеет время выполнения

[**2.2.4 Описание работы выбранных методов распознавания речи**](#page6) **для языковой модели**

Будем улучшать имеющийся транскодированный ответ с помощью языковой модели, использующей нечеткий поиск в основе. Задачу нечеткого поиска можно сформулировать так: по заданному слову найти в тексте или словаре размера n все слова, совпадающие с этим словом (или начинающиеся с этого слова) с учетом k возможных различий.

**2.2.4.1 Расстояние Левенштейна**

В числе наиболее известных метрик — расстояние Левенштейна, которое является наиболее часто применяемой метрикой и будет использоваться в работе.

Пусть есть расстояние между префиксами строк x и y, длины которых равны, соответственно, i и j, то есть . Цену преобразования символа a в символ b обозначим через . Таким образом, – это цена замены одного символа на другой, когда , – цена удаления a, а – цена вставки b. Заметим, что в случае, когда выполнены нижеследующие условия, d является расстоянием Левенштейна:

(17)

(18)

(19)

(20)

В процессе вычислений значения записываются в массив , а вычисляются они с помощью следующего рекуррентного соотношения.

(21)

Оно выводится следующим образом. Если предположить, что известна цена преобразования в , то цену преобразования в мы получим, добавив к ней цену удаления . Аналогично, цену преобразования ) в можно получить, прибавив цену вставки к цене преобразования в . Наконец, зная цену преобразования в , цену преобразования в мы получим, прибавив к ней цену замены на . Вспомним, что расстояние является минимальной ценой преобразования в , поэтому из трех указанных выше операций надо выбрать самую дешевую.

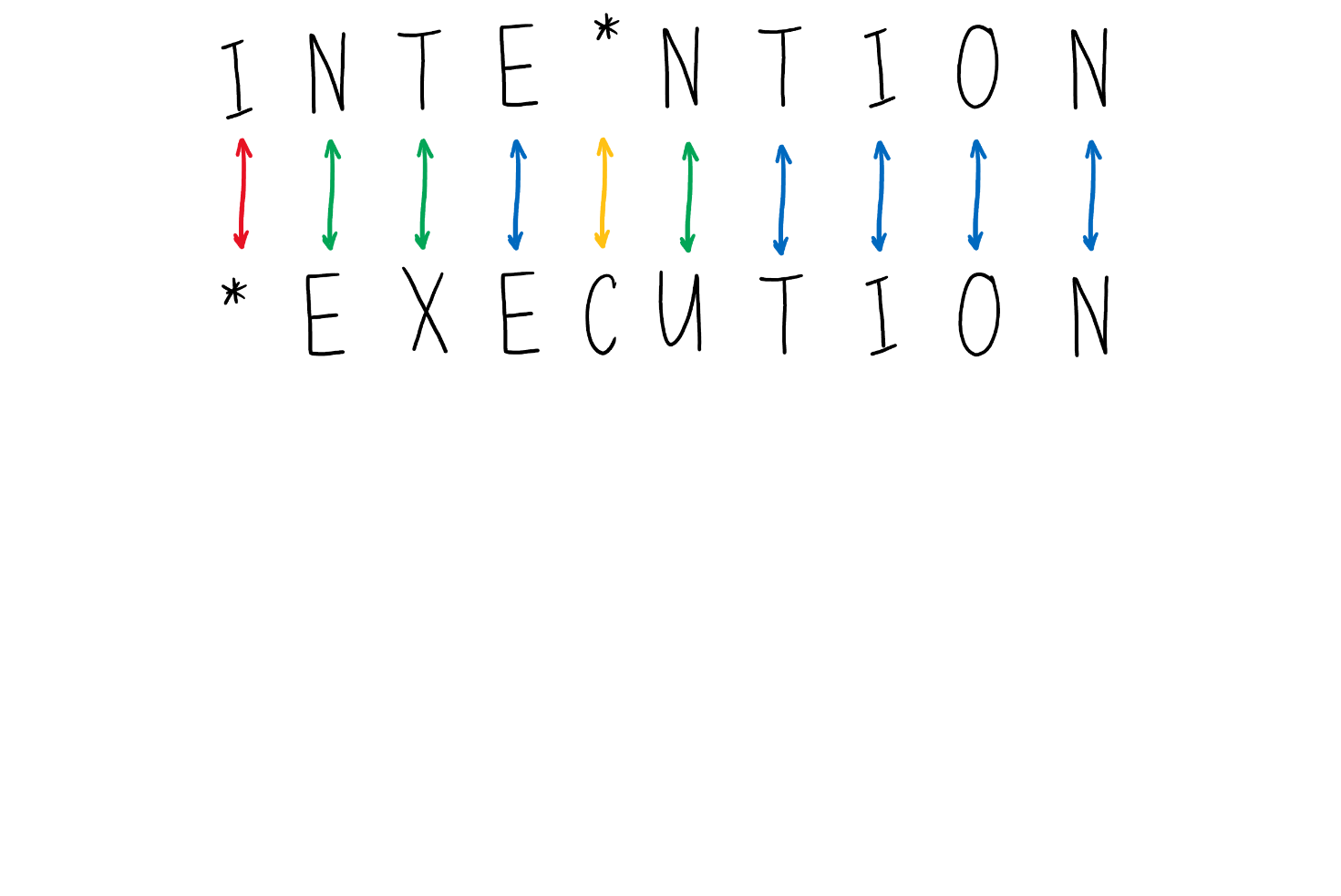
**

Рисунок 23 - Визуализация определения расстояния Левенштейна между двумя строками

На рисунке 23 показан пример определения расстояния Левенштейна между двумя строками «intention» и «execution». На рисунке синие стрелочки соответствуют одинаковым буквам в строках, красные отражают операцию удаления, желтые – добавления, а зеленые – замены.

**2.2.4.2 Мера Жаккара**

Мера Жаккара является мерой сходства, определяющаяся через размер пересечения и размер объединения двух наборов меток. Вычисляется по формуле:

, (22)

где

— количество элементов, общих для 1-го и 2-го множества,

— количество элементов из первого множества,

— количество элементов второго множества.

[**2.2.5 Модель нейронной сети**](#page9) **акустической модели**

Для построения акустической модели будут реализованы нейронные сети разной архитектуры, проведено сравнение качества работы и определение оптимальной сети, а также оптимального способа представления данных.

**2.2.5.1 RNN (One layer GRU –управляемый рекуррентный нейрон)**

Построим первую нейронную сеть как рекуррентную сеть, состоящую из одной ячейки GRU. На рисунке 24 показано ее примерное строение. В каждый момент времени на вход подается вектор .

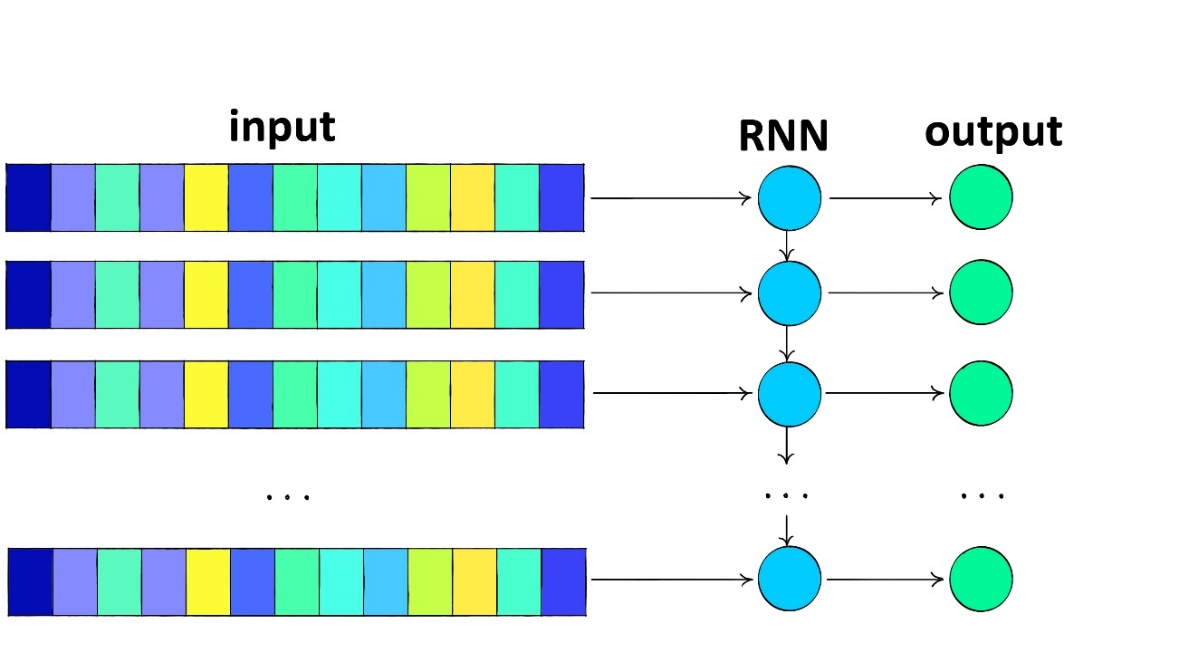


Рисунок 24 - Схема строения первой модели NN

На каждом временном шаге говорящий произносит один из 28 возможных символов, включая каждую из 26 букв английского алфавита, а также пробел («») и апостроф (').

Выходные данные RNN на каждом временном шаге - это вектор вероятностей с 29 записями, где j-я запись кодирует вероятность того, что j-й символ произносится во временной последовательности. (Дополнительный 29-й символ - это пустой «символ», используемый для дополнения обучающих примеров в пакетах, содержащих неравную длину.) Выходной вектор output состоит из массива вероятностей распознавания каждой буквы на каждом шаге, т.е. выхода CTC матрицы.

**2.2.5.2 RNN + TimeDistributed Dense**

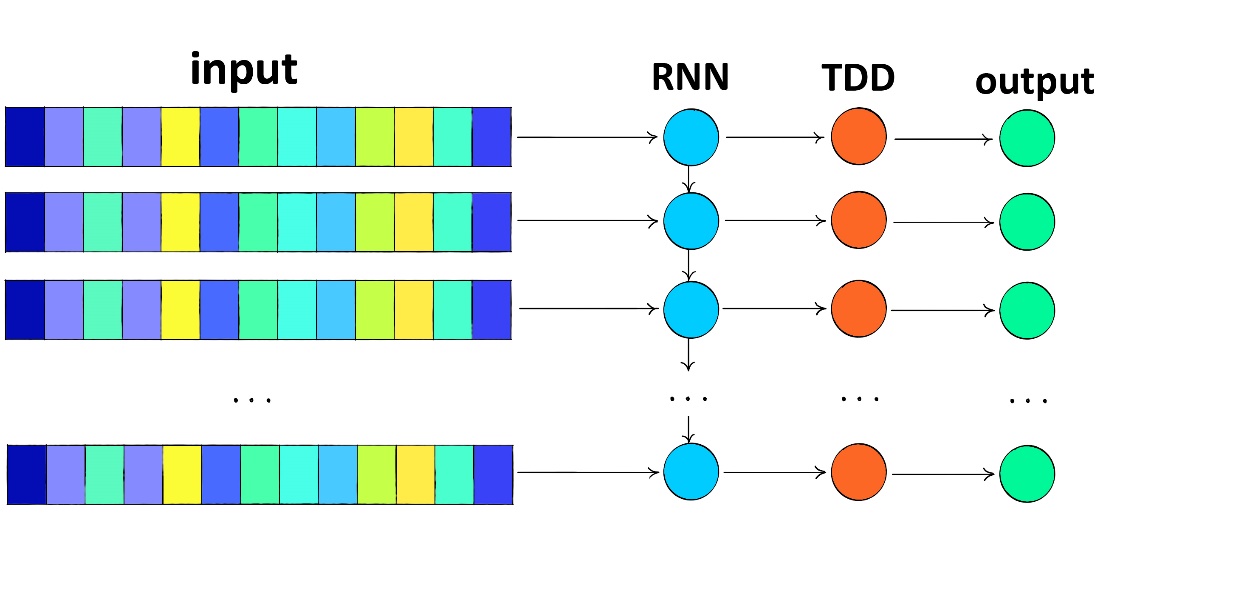


Рисунок 25 - Схема строения второй модели NN

TimeDistributed Dense позволяет применить слой к каждому временному срезу входного сигнала. Другими словами – это промежуточный слой между выходами RNN на каждом временном шаге и ответом (рисунок 25). Здесь и далее в качестве RNN будет использоваться LSTM.

На рисунке этот слой изображен красным.

**2.2.5.3 CNN + RNN + TimeDistributed Dense**

В третьей модели добавим использование свёрточной нейронной сети на входных данных. Архитектура изображена на рисунке 26.

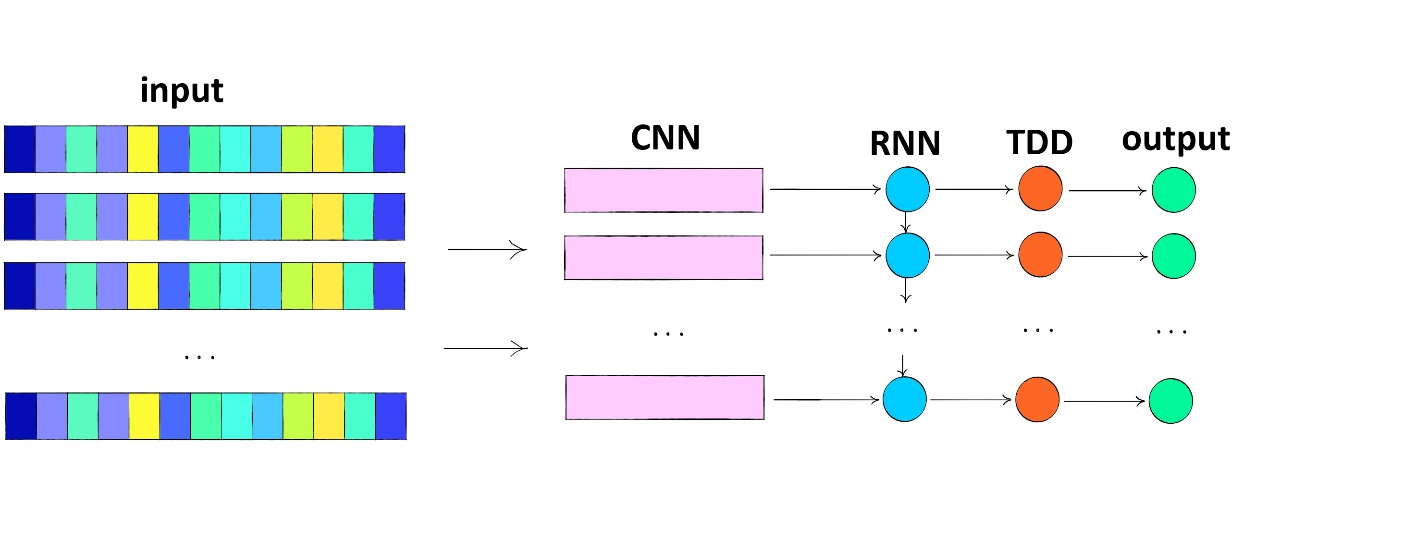


Рисунок 26 - Схема строения третьей модели NN

Свёрточная нейронная сеть будет иметь следующие подобранные экспериментально параметры:

* *filters=200,*
* *kernel\_size=11,*
* *strides=2,*
* *padding='valid'.*

Выход CNN уже будет подаваться на вход рекуррентной сети. CNN имеет 1D измерение.

**2.2.5.4 Deeper RNN + TimeDistributed Dense**

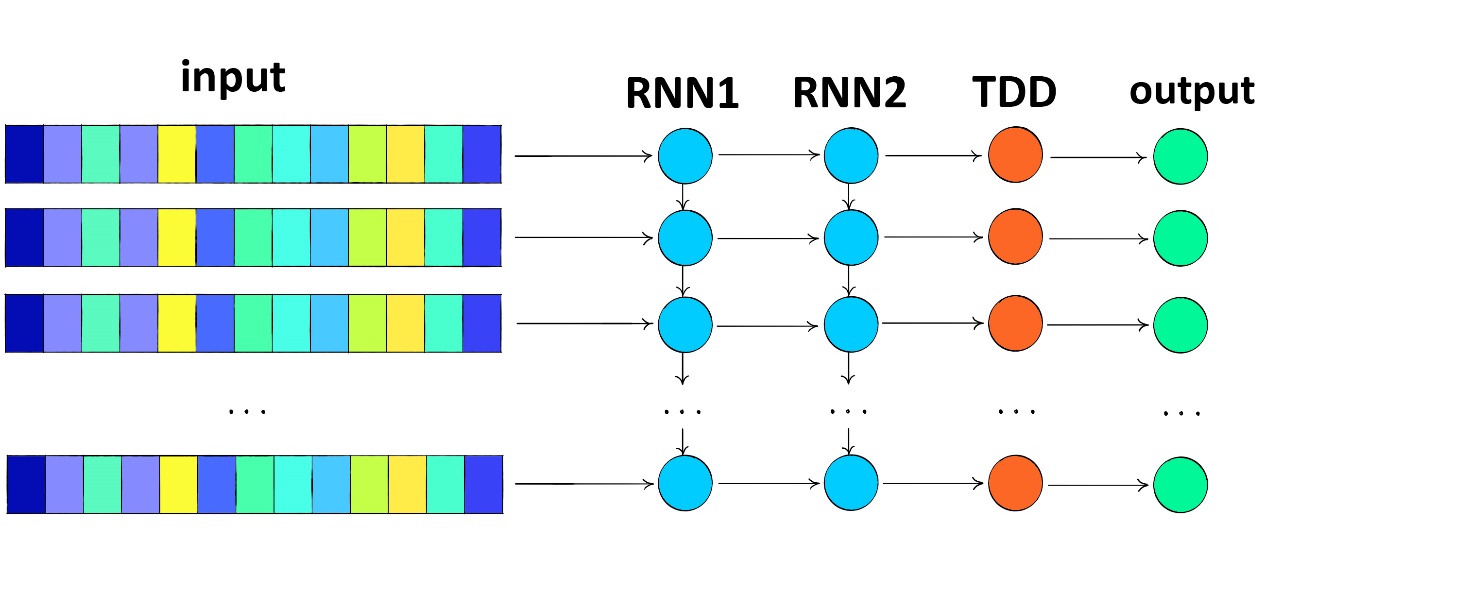
**

Рисунок 27 - Схема строения четвертой модели NN

Четвертая сеть будет состоять из глубокой рекуррентной сети DRNN. В качестве DRNN может задаваться переменное количество слоев RNN, к примеру, два слоя - RNN1 и RNN2. Выходная последовательность первого рекуррентного слоя используется как входные данные для следующего рекуррентного слоя. На рисунке 27 изображена схема строения сети.

**2.2.5.5 Bidirectional RNN + TimeDistributed Dense**

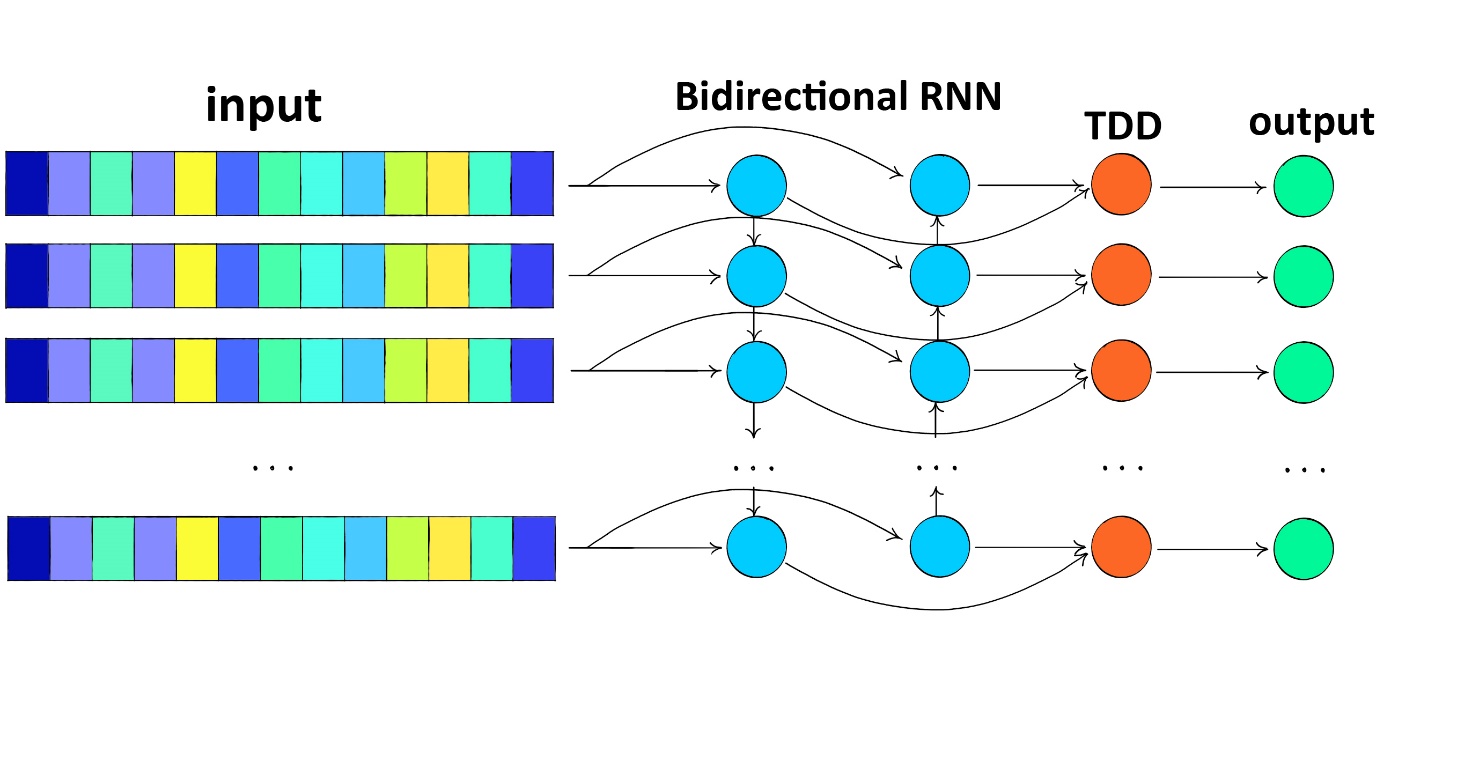
**

Рисунок 28 - Схема строения пятой модели NN

Одним из недостатков обычных RNN является то, что они могут использовать только предыдущий контекст. В задаче распознавании речи на вход подается уже полностью записанный аудиофайл, поэтому нет причин не использовать будущий контекст. Двунаправленные RNN (Brnn) делают это, обрабатывая данные в обоих направлениях с помощью двух отдельных скрытых слоев, которые затем подаются вперед на один и тот же выходной слой. Для построения пятой сети будет использоваться двунаправленная сеть в качестве части архитектуры. Схема представлена на рисунке 28.

**2.3 Метод работы системы идентификации голоса**

[**2.3.1 Постановка задачи идентификации**](#page12) **по голосу**

Поставим задачу идентификации по голосу. Необходимо создать систему идентификации личности диктора по его голосу в полученной аудиозаписи. Система должна иметь возможность сохранять голос диктора в своей базе и выполнять идентификацию по запросу и получению новой аудиозаписи голоса. Допускается, что идентификация будет происходить по закрытому множеству сохраненных голосов – в этом случае ответ всегда будет существовать и диктор всегда будет найден. Также необходимо, чтобы на ответ системы влиял только голос диктора и никак не влиял произнесенный на аудиозаписи текст – система будет являться текстонезависимой.

Всю задачу идентификации можно разбить на несколько этапов:

* предобработка сигнала,
* выделение признаков,
* распознавание диктора.

После записи голоса средствами записи и получения его на вход системы голос необходимо предварительно обработать. В этот этап входит чистка - удаление посторонних шумов, сетевого гула, выделение главного и удаление других голосов, удаление пауз, усиление высоких частот и нормализация сигнала. Для лучшего описания процесса предобработки процесс изображен на рисунке 29. Однако не все из изображенных этапов будут реализованы в системе (в частности, не будет происходить очистка от шумов).

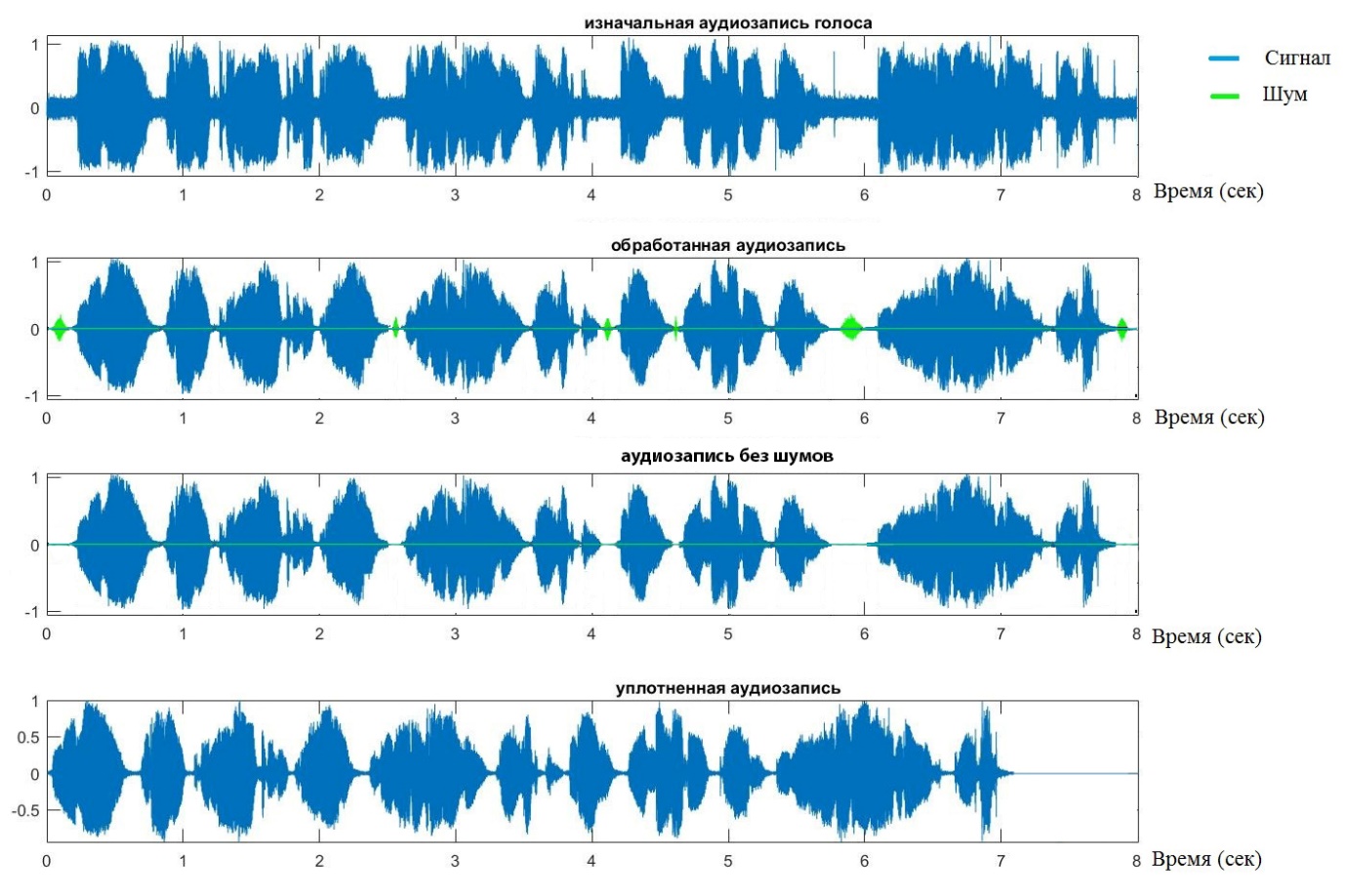


Рисунок 29 - Визуализация этапов предобработки аудиосигнала

Затем запись необходимо представить в виде многомерного вектора признаков, на основе которого будет уже выполняться распознавание диктора.

Таким образом, задача голосовой идентификации состоит в идентификации диктора по полученному аудиофайлу с его речью без учета произнесенного текста (в пассивном режиме). При этом задача является закрытой – диктор всегда будет выбираться из дикторов, зарегистрированных в системе.

**2.3.2 Ограничения**

В данной работе основное внимание будет уделяться последним двум этапам, первый этап предобработки будет рассматриваться вскользь, будут выполнены лишь ограниченное количество необходимых преобразований.

Предполагается, что фильтрация посторонних шумов, выделение главного голоса и удаление второстепенных и другие подзадачи этапа предобработки входного сигнала могут быть добавлены в систему в будущем при наличии такой необходимости. Это является одним из этапов развития системы.

Также важно отметить, что для представления голосов будут выбраны мел-кепстральные коэффициенты и их производные без сравнения качества работы на спектрограммах.

[**2.3.3 Описание работы выбранного метода идентификации по голосу**](#page6)

На данном этапе необходимо, имея новый набор признаков неизвестного диктора и множество сохраненных в базе наборов известных дикторов, отнести новые данные к верной категории - диктору. Сформулированная задача будет решаться через построение оптимальных для каждого диктора моделей гауссовской смеси распределений.

Модель гауссовой смеси распределений (Gaussian Mixture Model) - это функция, состоящая из нескольких гауссианов, каждый из которых идентифицируется как , где - количество кластеров в нашем наборе данных [39]. Другими словами, Модель Гауссовой смеси (GMM) представляет собой смесь многомерных гауссовых распределений вероятностей (многомерных нормальных распределений), которые наилучшим образом моделируют входной набор данных.

Хотя GMM часто классифицируется как алгоритм кластеризации, в нашем случае она будет представлять оценку плотности. Другими словами, результатом подгонки GMM к данным является технически не кластерная модель, а вероятностная модель, описывающая распределение данных.

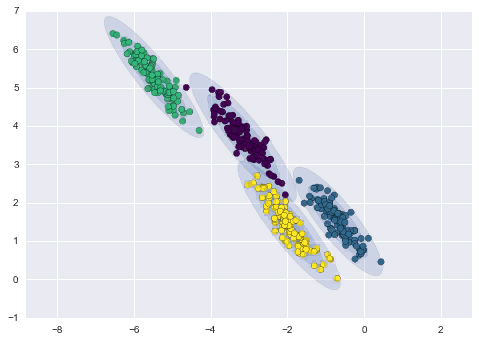
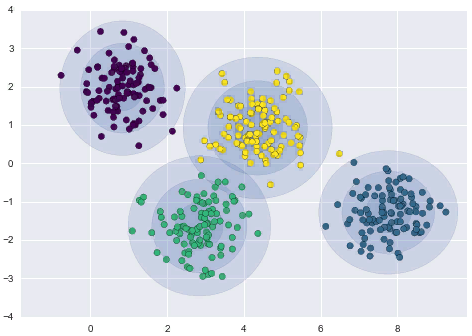


Рисунок 30 - Визуальное представление компонент GMM в зависимости от данных и типа ковариации [40]

На рисунке 30 данные представлены в виде точек, которые разбиты на кластеры. Точки каждого кластера окрашены своим цветом. Голубые окружности показывают смоделированное GMM распределение входных данных. Показаны результаты работы для двух типов ковариации - covariance\_type="spherical" (слева) и covariance\_type="full" (справа) [40].

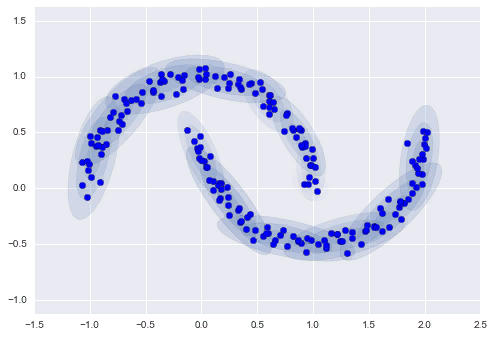
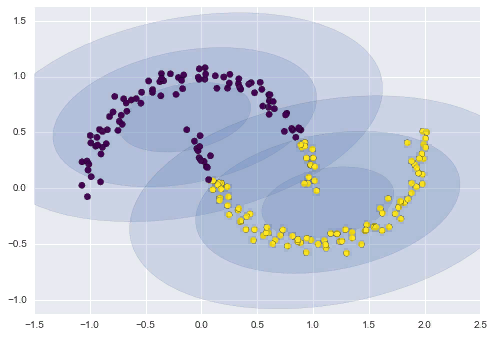


Рисунок 31 - Плохое описание данных (слева – 2 компоненты) и хорошее (справа – 16) [40]

Важно выбрать достаточное количество компонент GMM, иначе она будет давать плохой результат (рисунок 31).

[**2.3.4 Описание способа сравнения голосов**](#page6)

GMM удобна, так как представляет собой гибкое средство моделирования произвольного многомерного распределения данных. После этого моделирования будем сохранять все полученные модели GMM для каждого диктора.

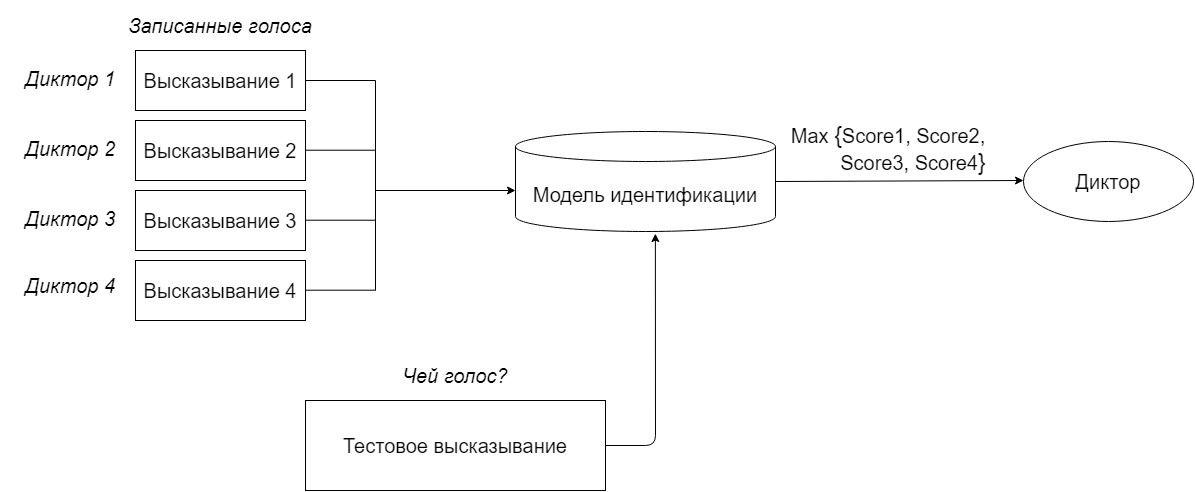


Рисунок 32 - Функциональная схема идентификации диктора

Получая на вход новый набор признаков, который необходимо отнести к верной категории – диктору, будем для каждого сохраненной модели GMM диктора определять вероятностные значения присутствия среди данного распределения записанного вектора признаков. Другими словами, будем сравнивать результат от каждой сохраненной GMM. Верной категорией будет признан диктор с максимальным значением вероятности. Описанная схема системы изображена на рисунке 32.

**2.3.5 Оптимизация метода**

Для того, чтобы сохранить образец голоса нового диктора, необходимо создать и обучить GMM, а затем сохранить ее в базе. Однако в процессе обучения возникает трудность – голосовых данных слишком мало. Для хорошего обучения GMM желательно иметь несколько часов речи диктора. В данной задаче такое требование невозможно. Нужно уметь строить модель на небольшом объеме данных – паре слов или паре фраз.

Будем решать эту проблему с помощью Универсальной фоновой модели (Universal Background Model, UBM, УФМ) [41].

Универсальная фоновая модель (UBM) в задаче распознавания диктора представляет собой некоторую модель, обученную на конечном большом множестве голосов определённых дикторов и содержащую в себе апостериорные знания об устройстве человеческого голоса в целом. UBM также является большой моделью гауссовой смеси (GMM), обученной для представления дикторонезависимого распределения признаков. Модель обучается методом Expectation-maximization (EM) на обучающем множестве. Когда необходимо получить признаки для вновь пришедшего произнесения, параметры UBM подстраиваются методом оценки апостериорного максимума (maximum a posteriori probability estimate, MAP) и полученная подстроенная модель будет являться конечной GMM для нового диктора. Она и будет сохранена. Описанная функциональная схема представлена на рисунке 33.

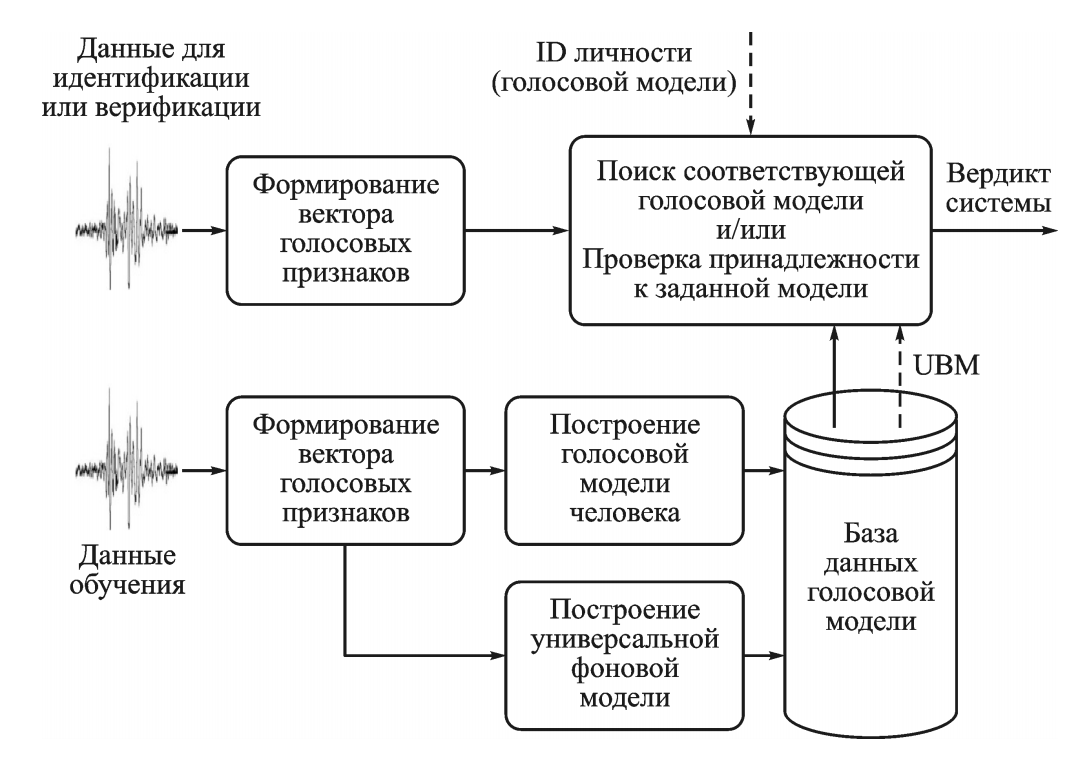
****

Рисунок 33 - Функциональная схема системы голосовой идентификации личности

Таким образом, UBM позволяет реализовывать GMM на небольшом количестве входных данных нового диктора, а также позволяет увеличить точность распознавания диктора по сравнению с неадаптированными моделями, которые можно построить на небольшой имеющейся обучающей выборке.

Системы, созданные с использованием UBM, называются системами идентификации диктора на основе модели Гауссовых смесей и универсальной фоновой модели (GMM-UBM). Первые варианты подобных систем предложены в [42][43][44][45].

Принцип обучения и работы UBM изображён на рисунке 34.

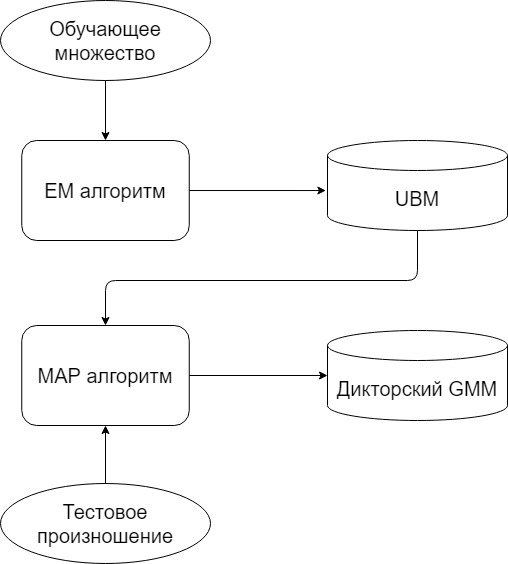


Рисунок 34 - Схема обучения и функционирования UBM-GMM

При создании UBM необходимо, чтобы данные, используемые для обучения модели, были сбалансированными по отношению к дальнейшему применению системы. Другими словами, голоса дикторов должны сильно отличаться, в выборке должны присутствовать как высокие голоса, так и низкие, с разной скоростью речи и разными записывающими системами - сбалансированы и по типу используемых при записи дикторов микрофонов.

В [43] показано, что системы голосовой верификации, производящие адаптацию модели диктора из универсальной фоновой модели, имеют намного лучшую точность, чем системы, в которых модель диктора обучается отдельно от УФМ. Это можно объяснить тем, что УФМ покрывает большинство классов акустических событий, появляющихся в речи дикторов. Соответственно, во время адаптации модели диктора, часть таких событий, появившихся в речи диктора, изменяют и подстраивают компоненты смеси под конкретного диктора. Оставшаяся часть событий, не встречающаяся в обучающей выборке, копируется из УФМ. Таким образом, это добавляет устойчивости модели к тем акустическим событиям конкретного диктора, которые отсутствовали в обучающей выборке.

Для -мерного вектора признаков , поданного на вход, плотность смеси рассчитывается как:

, (23)

где

– является D-мерным вектором признаков;

– является ли смесь весами, в сумме они = 1;

– матожидание каждой Гауссианы;

– параметры каждой GMM;

– ковариация каждой Гауссианы;

– плотности Гауссиан, определяемые как:

(24)

Обычно используется диагональная ковариационная матрица, а не полная ковариационная матрица, поскольку она более удобна с вычислительной точки зрения и эмпирически работает лучше.

GMM обучается на наборе обучающих векторов. Параметры GMM вычисляются итеративно с использованием алгоритма максимизации ожидания (EM), и поэтому нет никаких гарантий, что он дважды преобразуется к одному и тому же решению в зависимости от инициализации.

Алгоритм MAP представляет собой тот же алгоритм EM с начальными параметрами модели GMM, равными параметрам UBM. На каждой итерации происходит пересчет и изменение матожидания модели, сохраняя значение ковариации, так как изменение ковариации не улучшает качество работы модели.

Для пересчета матожидания используется максимальная апостериорная адаптация (maximum a posteroiri adaptation):

, (25)

где

– коэффициент адаптации матожидания,

– количество адаптационных данных,

- фактор соответствия (relevance factor).

**2.4 Структура системы аутентификации**

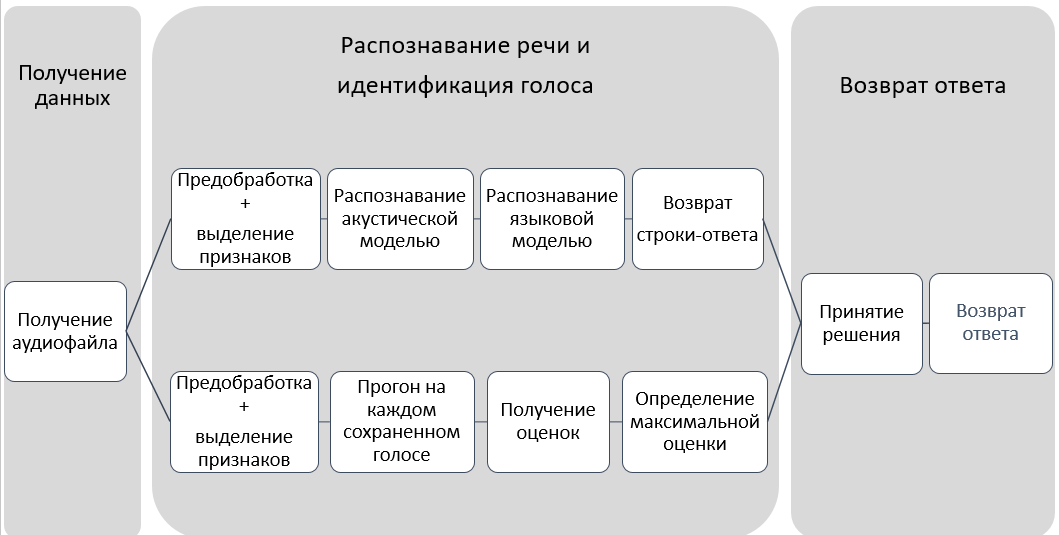
****

Рисунок 35 - Функциональная структура системы

Система аутентификации будет состоять из двух алгоритмов, выполняющихся параллельно и независимо – алгоритма по распознаванию речи и алгоритма по идентификацию диктора по голосу. Функциональная структура системы изображена на рисунке 35.

После получения аудиофайла система приступает к его обработке. Предобработка аудиофайла выполняется двумя компонентами независимо, так как механизмы предобработки для алгоритма распознавания речи и для алгоритма идентификации по голосу отличаются.

Принятие решения об успешности или не успешности аутентификации принимается системой на основе результатов, полученных от обоих компонентов. Если пройдены были оба фактора аутентификации, то система возвращает утвердительный ответ – пользователь прошел аутентификацию успешно.

Если рассматривать систему в целом, то помимо запросов на аутентификацию она должна выполнять и ряд других функций.

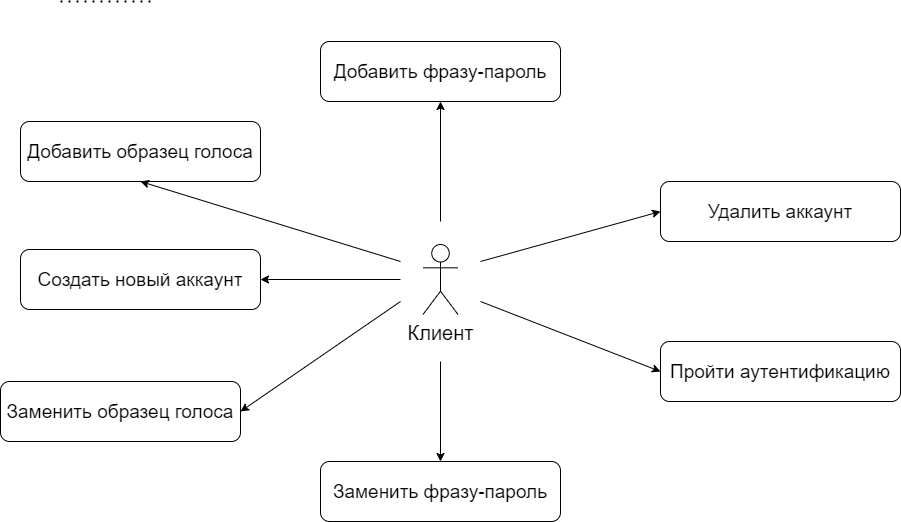


Рисунок 36 - UML-Диаграмма вариантов использования

Система должна обрабатывать запросы клиента на создание нового аккаунта (учетной записи), на добавление/изменение хранящейся в аккаунте информации о клиенте, на прохождение аутентификации, а также на удаление аккаунта. Диаграмма вариантов использования системы представлена на рисунке 36.

Таким образом, система будет представлять из себя вспомогательный компонент, который можно подключить к любой рабочей системе. Обращения к этому компоненту должны вестись из защищенной системы, так как компонент не содержит каких-либо ограничений и все запросы к системе будут выполнены без проверки на обладание запрашивающим соответствующих прав. К примеру, компонент может быть добавлен в систему авторизации к стороннему личному кабинету с уже реализованным интерфейсом. Предполагается, что это основная сфера применения разрабатываемой системы.

Таким образом, сторонний интерфейс будет дополнен созданным компонентом, что позволит наполнить функциональностью ПО без изменения уникальности его внешнего вида. Интеграция компонента во внешнюю среду будет проходить легче, если всю систему разработать в виде API. Связь с компонентом будет осуществляться только посредством запросов к нему и получению ответов. Для еще большей универсальности ответы от компонента будут возвращаться в формате JSON.

**2.5 Оценка эффективности и метрики качества**

**2.5.1 Метрики качества метода распознавания речи**

Для оценки качества распознавания существуют определенные метрики. Одна из наиболее популярных – WER (Word Error Rate) – мера оценки ошибки на уровне слов.

В системах автоматического распознавания речи основным показателем качества является точность распознавания, которая определяется как процент правильно распознанных слов (WRR – Word Recognition Rate) или, наоборот, неправильно распознанных слов (WER – Word Error Rate).

Метрика WER является коэффициентом ошибки в словах и вычисляется по формуле:

, (26)

где

S (substitutions) – количество замен слов (“twinkle” “crinkle”),

D (deletions) – количество операций удаления слов – (“get it done” “get done”),

I (insertions) – количество вставок - это всякий раз, когда добавляется слово, которое не было сказано (“trailblazers” “tray all blazers”),

N (number of words spoken) – количество произнесенных слов.

По некоторым оценкам, на большом наборе произвольных аудиозаписей с речью даже человек допускает ошибки вплоть до WER = 0.05. Согласно недавним исследованиям [46] для современных систем распознавания сечи коэффициент составляет около 20 процентов - для Google ≈18.3 процентов. Также можно рассматривать метрику CER (или LER, Character Error Rate/Letter Error Rate), которая рассчитывается аналогично, но на уровне букв, и по сути своей является расстоянием по Левенштейну.

Будем использовать обе эти метрики. Первая поможет анализировать качество работы на уровне слов, вторая – ниже, на уровне букв.

**2.5.2 Метрики качества метода идентификации по голосу**

Допустим, необходимо определить, принадлежит ли образец речи Y диктору S (из набора сохраненных дикторов). Сформулируем необходимые гипотезы:

* H0 – речь Y принадлежит диктору S,
* H1 – речь Y принадлежит не диктору S.

Матрица ошибок классификации представлена в таблице 5. В ней – ответ алгоритма на объекте, – истинная метка объекта.

Таблица 5 - Матрица ошибок классификации

|  |  |  |
| --- | --- | --- |
|  | True Positive (TP) | False Positive (FP) |
|  | False Negative (FN) | True Negative (TN) |

Тогда ошибка первого рода будет являться вероятностью «отказа своему» (False Negative), и ошибка второго рода – вероятность «допуска чужого» (False Positive).

Введем метрику точности. Она определяется по формуле:

(27)

Для оценки качества работы алгоритма на каждом из классов по отдельности введем метрики precision (точность) и recall (полнота). Они определяются по формулам:

(28)

(29)

можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными, а показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм. Введение не позволяет нам записывать все объекты в один класс, так как в этом случае мы получаем рост уровня False Positive (FP). демонстрирует способность алгоритма обнаруживать данный класс вообще, а — способность отличать этот класс от других классов.

**2.5.3 Метрики качества работы всей системы аутентификации**

В качестве показателей точности работы всей системы аутентификации будут использоваться метрики, описанные выше для метода идентификации. Для их измерения будет использоваться матрица ошибок аутентификации, совпадающая по структуре с матрицей ошибок идентификации. Таким образом, главными показателями качества работы будут считаться , , .

**2.6 Выводы**

Задача создания системы двухфакторной системы аутентификации будет состоять из нескольких этапов и включать несколько моделей, решающих определенную задачу в процессе принятия решения. Аутентификация будет разбита на две части – прохождения аутентификации по фактору свойства (идентификация по голосу говорящего) и по фактору знания (распознавание фразы-пароля, произнесенного говорящим в аудиозаписи).

Основной моделью при распознавании речи будет являться акустическая модель, использующая в основе нейронную сеть. Выходной сигнал акустической RNN - это вероятность символов, которые затем декодируется с помощью различных алгоритмов для получения наилучшего ответа. Чтобы улучшить полученный ответ, будет использоваться языковая модель, использующая информацию о семантическом сходстве слова ответа с набором часто встречающихся слов в данном языке.

Для реализации системы идентификации по голосу будет использоваться метод GMM-UBM на признаках MFCC. Оптимизация в виде UBM позволит избавиться от описанных недостатков и трудностей построения GMM. GMM, являясь средством моделирования произвольного многомерного распределения данных, будет выполнять функцию классификатора, выдавая плотность смеси на входном векторе признаков для каждого из сохраненных дикторов.

Сама система будет представлять из себя API. Система должна брать на себя весь процесс аутентификации и иметь возможность обрабатывать такие запросы как создание нового аккаунта (учетной записи пользователя), добавление и изменение информации о пользователе, на основе которой будет осуществляться аутентификация, запрос на аутентификацию, а также запрос на удаление аккаунта.

**3. Программная реализация** **системы аутентификации**

**3.1 Выбор языка и вспомогательных модулей для реализации системы**

Реализация всей системы происходила на языке Python 3.7. Разработка проводилась в программной среде Jupyter Notebook и PyCharm. В качестве вспомогательных библиотек использовались:

* TensorFlow и Keras (для построения нейронных сетей),
* Numpy (для реализации функций и алгоритмов),
* Scipy (для перевода аудиозаписей в .wav формат),
* Json (для получения данных из запроса и формирования ответа от системы в этом формате; для построения вспомогательных данных – словарей соответствия имени аудиофайла расшифровке текста, который присутствует в аудиофайле),
* Matplotlib (для построения графиков),
* Librosa, Pydub и Contextlib (для загрузки и работы с аудиофайлом),
* Python\_speech\_features (для построения MFCC),
* Webrtcvad (для реализации функции детектирования наличия речи в аудиосигнале).

**3.2 Программная реализация моделей и алгоритмов**

**3.2.1 Реализованные методы и классы в рамках компонента по распознаванию речи**

Опишем реализованный функционал системы распознавания подробнее. Система включает несколько классов и функций, сформированных в общие блоки:

1. class AudioGenerator - класс, который используется для работы с аудиофайлами, выделения признаков и предоставления их в сеть для обучения или тестирования. Код приведен в приложении А. Класс включает:

* def get\_batch - формирует группы (batch) обучающих, проверочных или тестовых данных;
* def sort\_data\_by\_duration - сортировка обучающих или проверочных наборов по (возрастающей) продолжительности;
* def shuffle\_data\_by\_partition - перемешивает обучающие или проверочные данные;
* def shuffle\_data - перемешивает данные. Вызывается после полного прохождения обучения или для валидации данных в процессе обучения;
* def sort\_data - сортировка данных по длительности;
* def plot\_spectrogram\_feature – функция построения спектрограмм. ;
* def plot\_mfcc\_feature – функция построения Mel-frequency cepstrum;

1. class Models – класс, в котором описывается строение нейронных сетей, использующихся в работе. Код приведен в приложении Б. Включает:

* def simple\_rnn\_model – строит модель NN, использующую GRU;
* def rnn\_model – строит модель NN, использующую рекуррентный слой RNN и слой временного распределения TDD;
* def cnn\_rnn\_model – строит модель NN, состоящую из сверточного слоя между входными данными и RNN + TDD;
* def deep\_rnn\_model – строит глубокую нейросетевую модель с возможностью создания заданного количества подряд идущих слоев RNN;
* def bidirectional\_rnn\_model – строит двунаправленную рекуррентную нейронную сеть (BRNN), соединяющую два скрытых слоя противоположных направлений с одним и тем же выходом;

1. CTC training. Включает:

* def ctc\_lambda\_func – функция для расчета потерь CTC;
* def add\_ctc\_loss – функция, возвращающая модель с CTC потерей;
* def train\_model – функция для обучения модели;

1. class LanguageModel. Код приведен в приложении В. Включает:

* def levenshtein\_distance – функция вычисления расстояния Левенштейна для двух строк;
* def ngram – функция для построения N-gram. Возвращает их как список;
* def jaccard – функция вычисления коэффициента Жаккара;
* def minimumEditDistance\_spell\_corrector – основная функция класса. Возвращает результат языковой модели – слово, для которого расстояние Левенштейна между ним и входящим словом минимально;
* def ngram\_spell\_corrector – основная функция класса. Возвращает результат языковой модели – слово с максимальным коэффициентом Жаккара от него и входящего слова.

Таким образом, структура системы распознавания речи представлена на рисунке 37.

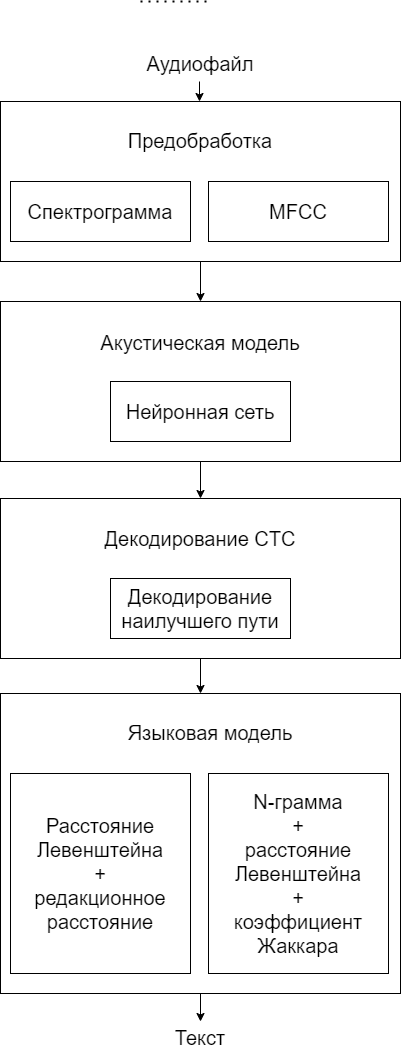


Рисунок 37 - Диаграмма структуры системы распознавания речи

Отдельные блоки представляют отдельные логические компоненты системы. Стрелками указана очередность выполнения отдельных блоков при поступлении на вход системе нового аудиофайла для распознавания. Блок Acoustic model включает в себя одну нейронную сеть NN, которая будет признана оптимальной экспериментально в разделе 3.3. Блок CTC decoding состоит из метода поиска лучшего пути (beam search decoding). Остальные блоки включают в себя пару методов, среди которых также необходимо выделить один наиболее подходящий для формирования наибольшей точности распознавания в данной задаче.

**3.2.2 Реализованные методы и классы в рамках компонента идентификации по голосу**

В процессе работы был реализован класс Frame. Код всех функций предобработки аудиофайла представлен в приложении Г. Программный код построения моделей GMM приведен в приложении Д. Реализованные функции:

1. def extract\_features – функция, вычисляющая признаки из поданного на вход аудиофрагмента. Функция вычисляет MFCC и добавляет первую и вторую производную;
2. def normalize – функция нормализации поданного на вход вектора признаков. Многие методы машинного обучения могут вести себя плохо, если отдельные признаки не будут более или менее похожи на стандартные нормально распределенные данные: Гауссовы с нулевым средним и единичной дисперсией. Фукция возвращает нормализованный вектор. Центрирует признаки, удаляя среднее значение каждого признака, а затем масштабирует, деля признаки на стандартное отклонение;
3. def write\_wave – функция, записывающая обработанный аудиофрагмент в .wav файл по указанному пути;
4. class Frame – класс, представляющий собой фрейм, на множество которых нарезается аудиофрагмент в процессе выделения наполненных голосом фрагментов и удаления тишины;
5. def frame\_generator – функция, генерирующая аудиофреймы (аудиокадры) из поданого на вход аудиофрагмента. Принимает желаемую длительность кадра в миллисекундах, данные и частоту дискретизации. Дает кадры требуемой длительности;
6. def vad\_collector – отфильтровывает аудиофреймы тишины (не наполненные голосом), выдает только озвученный аудиофрагмент. Использует алгоритм скользящего окна с дополнениями по аудиокадрам. Когда более 90 процентов кадров в окне озвучены, устанавливается триггер , функция начинает записывать звуковые кадры. Пока 90 процентов кадров в окне не будут озвучены, тригер сохраняет состояние и функция продолжает запись;
7. def map\_adaptation – функция MAP адаптации UBM к GMM для конкретного диктора. Принимает на вход предобученную модель, максимальное количество итераций. Выполняет итерационно подгон параметров UBM, пока не превысит максимальное количество итераций или не будет достигнута указанная точность;
8. def get\_data – загружает аудиофайл и предобрабатывает его. Удаляет тишину и паузы, склеивает и сохраняет. Работает в двух режимах – сохраняет либо все данные, которые получила после удаления пауз, либо какую-то заданную часть. Принимает на вход имя файла, который нужно загрузить и путь, по которому нужно сохранить обработанный файл;
9. def get\_features – функция извлекает вектор признаков заданной размерности из аудиофайла либо загружает сохраненные признаки из файла. В качестве параметров передается имя файла, количество MFCC. Возвращает извлеченные признаки;
10. def get\_ubm – создает модель UBM с заданными параметрами и обучает ее на входных признаках либо загружает уже существующую сохраненную модель из файла. Возвращает модель UBM;
11. def get\_gmm – создает модель GMM из поданной на вход UBM и обучает ее на входных признаках либо загружает уже существующую сохраненную модель из файла. Возвращает модель GMM;
12. get\_recording – загружает несколько аудиофайлов из заданной директории, склеивает их и нарезает. Возвращает путь, по которому сохранен обработанный файл;
13. get\_speaker – извлекает признаки из аудиофайла диктора, расположенного по определенному пути. Возвращает вектор признаков;
14. create\_gmms – извлекает признаки из указанного файла, загружает модель UBM, обучает UBM с помощью MAP адаптации. Сохраняет обученную модель;
15. test\_gmms – функция создана для тестирования существующих моделей. Считает время работы, а также метрики качества работы каждой модели, расположенной в определенной заданной директории.

**3.3 Экспериментальное исследование влияния архитектуры модели на точность распознавания речи**

* + 1. [**Экспериментальное исследование влияния архитектурных параметров нейросетевых моделей на точность распознавания**](#page6)

Для определения оптимальной архитектуры было протестировано много различных архитектур нейронных сетей.

Будем сравнивать влияние входных данных на качество работы сети. Для этого обучим описанные архитектуры на данных, представленных в виде спектрограмм и в виде мел-коэффициентов. На графиках ниже (рисунки 38 и 39) показаны результаты обучения сетей на разных данных. А именно, отображен validation loss на протяжении 20 эпох.

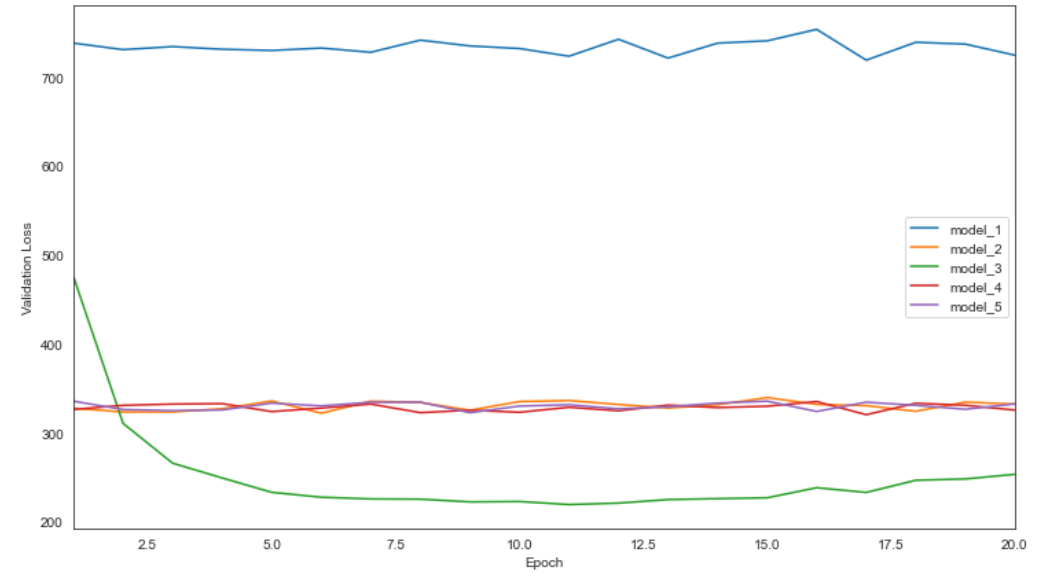


Рисунок 38 - Функция потерь валидации при обучении на спектрограммах

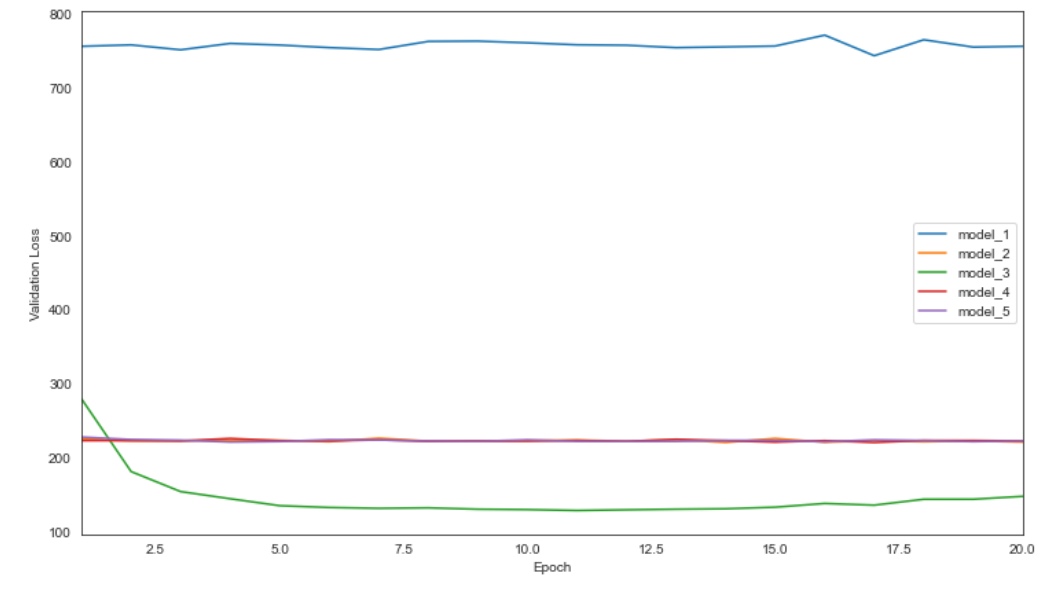


Рисунок 39 - Функция потерь валидации при обучении на мел-коэффициентах

Можно видеть, что при обучении на спектрограммах большинство сетей дает результат хуже, чем на мел-коэффициентах, кроме первой модели (которая остается примерно на том же уровне и не улучшает прогресс в обоих случаях). Третья сеть является наиболее точной в обоих случаях, однако на мел-коэффициентах дает результат точнее. Будем использовать мел-коэффициенты как вариант представления входных данных.

При сравнении описанных выше моделей на мел-коэффициентах был получен следующий результат обучения (рисунок 40). Первая модель, состоящая их одного элемента RNN, дала самый плохой результат (голубая кривая). Модель, использующая CNN, давала наименьшее значение функции потерь (зеленая кривая), и наилучшую точность.

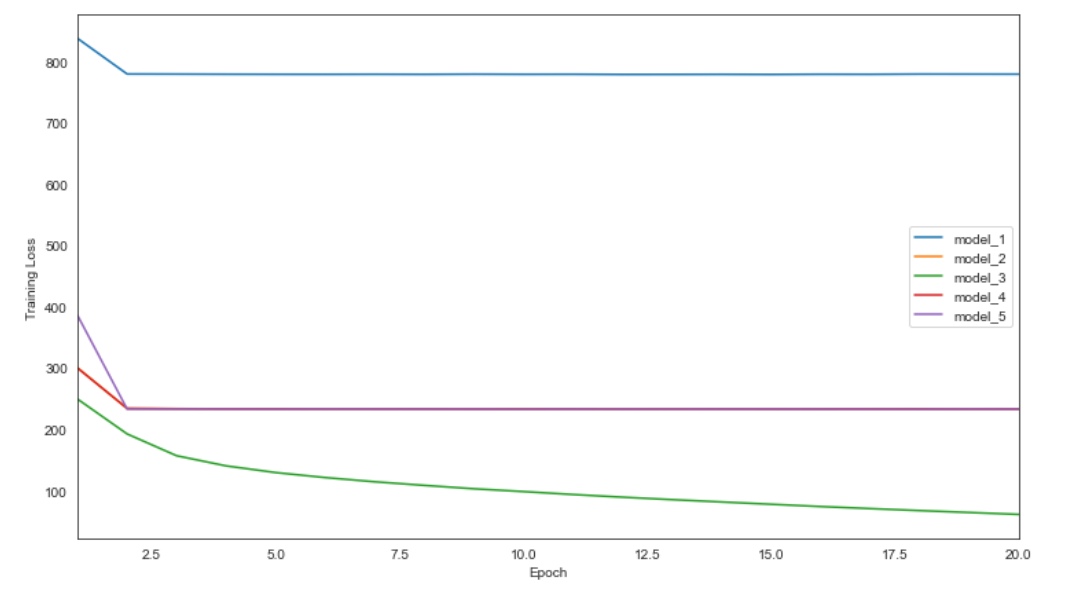


Рисунок 40 - Зависимость функции потерь обучения от количества эпох обучения разных моделей

Также из графика видно, что модель 3 продолжает обучаться на протяжении всех эпох (зеленая кривая стремится вниз, в то время как все остальные после первых 2-3 эпох не улучшают показатели обучения). Однако если посмотреть на график валидации (рисунок 38, 39 и более детальный процесс обучения - рисунок 41), то можно сказать, что после 15 эпохи сеть 3-ей архитектуры начинает немного переобучаться и адаптироваться под обучающую выборку, хотя все равно является самой оптимальной среди всех архитектур. Это может быть вызвано тем, что сверточный слой лучше находит на изображении дополнительные признаки, использующиеся потом RNN при обучении.

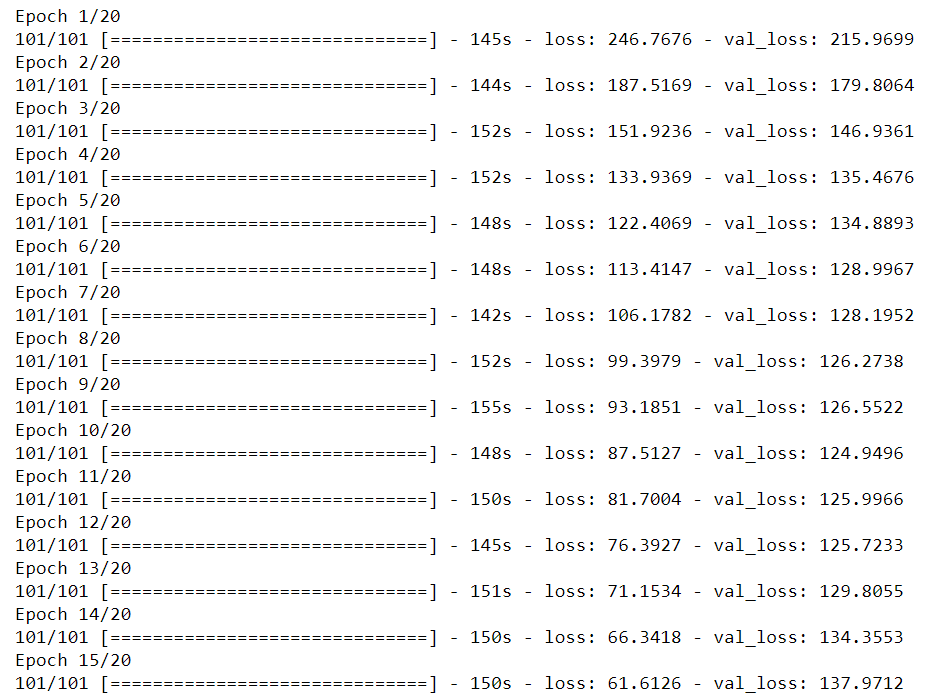


Рисунок 41 - Обучение сети 3 в течение 15 эпох

При изменении параметров сверточного слоя оптимальными стали параметры:

* *filters=200,*
* *kernel\_size=11,*
* *strides=2,*
* *padding='valid'.*

Параметр strides может также быть также равен единице. Изменение показателя никак не повлияло на качество сети.

В нейронной сети Deep Bidirectional RNN использование большого количества слоев сети приводило к большим значениям функции потерь. Это может быть связано с небольшим количеством обучающих данных, в то время как более глубокие сети требуют большее количество данных. При обучении трехслойной сети loss функция давала значения около 500 на протяжении 20 эпох. Для пяти слоев loss функция имела значения еще больше - около 750 (и уменьшилась к 700 после 20 эпох).

Таким образом, было принято решение принять нейронную сеть номер 3 за оптимальную и использовать ее в системе распознавания как акустическую модель.

**3.3.2 Экспериментальное сравнение качества работы систем, использующих разные методы распознавания**

Построенная оптимальная архитектура нейронной сети после обучения дает результат распознавания тестового примера, представленный на рисунке 42.

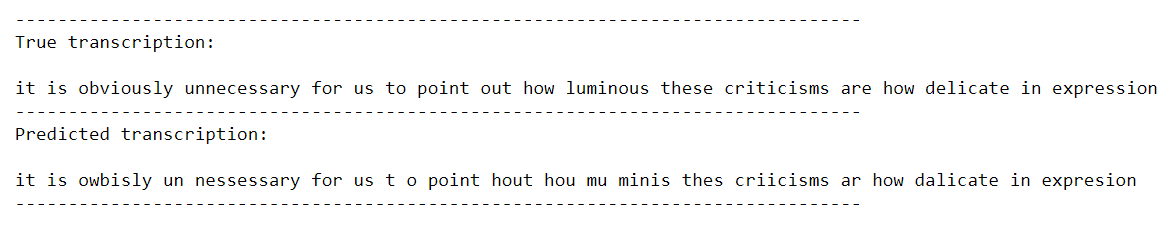


Рисунок 42 - Пример распознавания фразы акустической моделью

При этом коэффициенты точности распознавания WER и CER составляют около WER = 40 процентов, CER = 34 процента для валидационных данных.

Одним из стандартных способов улучшения результатов декодера является включение языковой модели. Добавим языковую модель для повышения точности распознавания речи. Для этого сравним два варианта ее построения.

Будем разбивать каждое предложение на слова и пытаться оптимизировать каждое. Для этого будем либо минимизировать расстояние Левенштейна, либо максимизировать коэффициент Жаккара.

На практике использование разных методов не дает больших различий. Первый метод дает немного более точный результат, поэтому было принято решение использовать его, хотя разница в качестве работы незначительная.

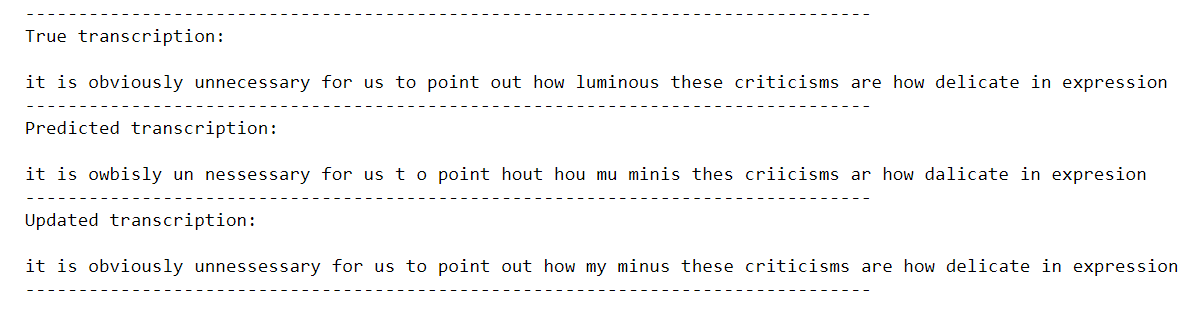


Рисунок 43 - Пример распознавания фразы всей системой

На рисунке 43 представлен результат распознавания фразы отдельно акустической моделью и полностью всей системой, включающей как акустическую, так и языковую модель. Видно, что качество распознавания улучшается, исправляются мелкие опечатки и ошибки. Однако некоторые ошибки, созданные акустической моделью, могут быть ухудшены еще сильнее языковой моделью и сильно отдалить финальный результат от верного ответа (“luminous” “my minus”).

Для полной системы коэффициенты точности составляют по валидационным данным WER = 33 процента, CER = 29 процентов, что является достаточно неплохим результатом.

**3.4 Экспериментальное исследование влияния архитектуры модели на точность идентификации по голосу**

Для определения оптимальных параметров модели UBM было построено 27 моделей различной архитектуры, для каждой модели проведено тестирование работы при 40 добавленных дикторах. Результаты приведены в таблице 6. Значения Accuracy, Precision, Recall вычислены как среднее арифметическое для каждого класса (диктора). Также для сравнения было измерено среднее время принятия решения классификатором в зависимости от длительности аудиофрагмента речи, поданного на вход для идентификации (столбцы «Время на 40 с» и «Время на 5 с»).

Таблица 6 - Качество работы построенных моделей

| Компоненты | Ковариация | MFCC | Accuracy | Precision | Recall | Время (на 40 с) | Время (на 5 с) |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | diag | 13 | 0.9946 | 0.8125 | 0.8916 | 2.2191 | 1.6222 |
| 8 | diag | 20 | 0.9951 | 0.8125 | 0.9 | 2.2509 | 1.7025 |
| 8 | diag | 8 | 0.9942 | 0.8203 | 0.8833 | 2.1254 | 1.4941 |
| 8 | full | 13 | 0.9945 | 0.8375 | 0.8916 | 4.71203 | 3.1141 |
| 8 | full | 20 | 0.9945 | 0.8511 | 0.8916 | 5.2238 | 3.4823 |
| 8 | full | 8 | 0.9945 | 0.8125 | 0.898 | 3.3213 | 2.1422 |
| 8 | tied | 13 | 0.9945 | 0.8502 | 0.898 | 4.5222 | 3.0525 |
| 8 | tied | 20 | 0.9954 | 0.8875 | 0.9083 | 5.3009 | 3.5039 |
| 8 | tied | 8 | 0.9952 | 0.8527 | 0.878 | 3.2472 | 2.0821 |
| 16 | diag | 13 | 0.9937 | 0.8751 | 0.925 | 2.45507 | 1.67226 |
| 16 | diag | 20 | 0.9954 | 0.8875 | 0.9 | 2.5688 | 1.7152 |
| 16 | diag | 8 | 0.9937 | 0.8125 | 0.875 | 2.3292 | 1.6191 |
| 16 | full | 13 | 0.9951 | 0.8501 | 0.9 | 6.2387 | 4.1502 |
| 16 | full | 20 | 0.9951 | 0.8512 | 0.9 | 8.86304 | 5.6113 |
| 16 | full | 8 | 0.9951 | 0.8524 | 0.9 | 4.6579 | 2.96404 |
| 16 | tied | 13 | 0.9963 | 0.8503 | 0.925 | 6.2921 | 4.0401 |
| 16 | tied | 20 | 0.9963 | 0.8125 | 0.925 | 8.6095 | 5.5437 |
| 16 | tied | 8 | 0.9946 | 0.8208 | 0.875 | 4.6362 | 2.8073 |
| 32 | diag | 13 | 0.9954 | 0.8125 | 0.875 | 2.82457 | 2.05204 |
| 32 | diag | 20 | 0.9958 | 0.8875 | 0.925 | 2.8914 | 2.0058 |
| 32 | diag | 8 | 0.9951 | 0.8513 | 0.925 | 2.7116 | 1.9027 |
| 32 | full | 13 | 0.9954 | 0.8629 | 0.9083 | 12.5033 | 8.06306 |
| 32 | full | 20 | 0.9954 | 0.8629 | 0.9083 | 15.4039 | 9.91009 |
| 32 | full | 8 | 0.9954 | 0.8629 | 0.9083 | 7.4432 | 4.3401 |
| 32 | tied | 13 | 0.9951 | 0.8125 | 0.886 | 12.8066 | 8.0172 |
| 32 | tied | 20 | 0.9954 | 0.8875 | 0.9083 | 15.3268 | 9.8261 |
| 32 | tied | 8 | 0.9951 | 0.8208 | 0.965 | 7.3689 | 4.3601 |

Таким образом, описанный алгоритм обучения и построения модели GMM-UBM был опробован с различными параметрами:

1. Количество MFCC: 8, 13, 20;
2. Тип ковариации в GMM: full, diag, tied;
3. Количество компонент: 8, 16, 32;
4. С разной длительностью тестовых данных: 5 сек, 20 сек, 40 сек.

Для обучения GMM брались аудиофрагменты речи длительностью 20 секунд.

**3.5 Выводы по результатам экспериментальных исследований**

В процессе экспериментальных исследований влияния архитектуры модели на точность распознавания речи было проведено экспериментальное сравнение качества работы отдельных компонентой системы распознавания речи в зависимости от выбранного метода их реализации. Была определена оптимальная нейронная сеть среди всех предложенных выше вариантов нейронных моделей для построения акустической модели на основе различия их архитектур и формы представления их обучающих данных. Также сравнивалось качество работы методов, использующихся внутри языковой модели.

Экспериментально было установлено, что нейронная сеть с архитектурой 3 дает наиболее точные предсказания и наименьшую функцию ошибки. Для улучшения точности ответа для языковой модели будем минимизировать расстояние Левенштейна. Такая система дает показатели точности WER = 33 процента, CER = 29 процентов, что является неплохим результатом.

После проведения экспериментальных исследований влияния архитектуры модели на качество идентификации по голосу была сформирована таблица с показателями качества. Как видно из таблицы, все построенные модели дают хороший результат работы. Было замечено, что модели с типом ковариации full требуют намного больший объем памяти по сравнению с моделями других параметров (например, для 16 компонент и MFCC = 20 объем памяти для моделей с типами diag, tied, full равен соответственно 32, 93, 1359 КБ). Таким образом, по объему памяти больше всех требует модель full, меньше всех – diag. Причиной этому может выступать большее число настраиваемых параметров модели. Также модель с типом ковариации full дольше всех обучается на выборке (соответственно, модель с типом ковариации diag обучается быстрее всех).

Время работы процесса идентификации значительно увеличивается при увеличении количества компонент смеси, увеличении длительности входного аудиофрагмента для идентификации, увеличении количества MFCC (размерности вектора признаков) и выборе типа ковариации full.

Как итог, все спроектированные модели дают примерно одинаковый результат. Оптимальной моделью может выступать любая модель с относительно небольшим временем работы. Как пример, оптимальной может быть принята модель с параметрами:

1. Количество компонент UBM: 16,
2. Тип ковариации: diag,
3. Количество MFCC: 13,
4. Длительность обучения GMM: 20 сек,
5. Длительность тестового фрагмента: 20 сек.

Важно заметить, что длительность тестового фрагмента может отличаться от длительности фрагмента, поданного на вход модели. В процессе предобработки после удаления тишины и пауз фрагмент может значительно сократиться в размере, поэтому оптимальная длительность выбрана «с запасом».

**3.6 Программная реализация API**

**3.6.1 Спецификация требований**

Опишем подробнее требования в системе.

* Для решения задачи необходимо реализовать сервис, который принимает аудиозапись с речью пользователя и возвращает строку-ответ для полученного аудиофайла;
* Критическими параметрами системы является правильность её работы и время её работы;
* Сервис должен выдерживать большое количество обращений (быть высоконагруженным);
* Сервис должен быть легко масштабируемым (возможность запустить в нескольких параллельных воркерах);
* Сервис должен иметь возможность менять свою конфигурацию без остановки работы;
* Сервис может не иметь авторизации при запросе к нему, так как предполагается к использованию внутри защищенной системы;
* Основные ресурсы при работе сервиса должны тратиться на работы системы распознавания речи или идентификации по голосу. Время работы самого сервиса не должно вносить значительный вклад в общее время обработки запроса;
* Система должна осуществлять аутентификацию как по двум факторам, так и по какому-то одному (при отсутствии возможности аутентификации по обоим).

**3.6.2 Выбор фреймворка**

Для решения поставленной задачи было принято решение разрабатывать асинхронный API. Асинхронность позволит оптимально распределять время для обработки запросов в условиях существования частей системы, которые требуют значительных временных затрат на работу.

Для более понятного описания преимуществ асинхронности обычно приводится пример с игрой в шахматы одновременно с несколькими соперниками. Пусть имеет гроссмейстера, который играет одновременно с 24 соперниками. На ход он должен тратить по 5 секунд в то время как соперники – по 55 секунд. Рассмотрим два варианта игры – синхронный и асинхронный. В случае синхронной игры гроссмейстер играет одну игру до завершения партии. Пусть игра требует 30 ходов. Она займет (5+55)\*30 == 1800 секунд или 30 минут. Всё мероприятие займет 24\*30 == 720 минут или 12 часов. В случае асинхронной игры гроссмейстер получает возможность перемещаться от одного стола к другому и играть несколько партий параллельно. Пока соперники размышляют над своим ходом, гроссмейстер перемещается к другим столам и делает ходы в других играх. Тогда 1 ход в каждой игре займет 24\*5 == 120 секунд или 2 минуты. Всё мероприятие уместится в 120\*30 == 3600 секунд или один час.

Рассмотрим возможные варианты фреймворков на Python для реализации сервиса.

1. Tornado

Торнадо является одним из самых популярных фреймворков для реализации асинхронных приложений. Он способен поддерживать много открытых пользовательских соединений, что является несомненным плюсом. Прекрасно подходит в задачах, когда необходимо подолгу держать открытое соединение с каждым пользователем. Он был создан достаточно давно (считается первым асинхронным фреймворком на Python. Был написан в 2009 году). Несмотря на большую популярность имеет конкурентов на рынке фреймворков.

1. Aiohttp

Это асинхронный фреймворк, базирующийся на asyncio. Aiohttp был создан русским программистом Андреем Светловым, который работал над библиотекой asyncio. Фреймворк был признан лидером по производительности в начале 2019 года. В исследовании [47] и [48] при сравнении производительности aiohttp выиграл во всех тестах при сравнении с tornado. для реализации высоконагруженного сервиса аутентификации был выбран именно он.

Также aiohttp позиционирует себя как микрофреймворк Python, что позволит упростить архитектуру сервиса и облегчить написание кода.

**3.6.3 Архитектура системы**

Система представляет собой приложение, которое запускается на определенном порте или сокете и входит в бесконечный цикл событий - event loop (Приложение Е). Сервис имеет свой конфигурационный файл, в котором задаются необходимые параметры, константные значения, ссылки или пути до файлов. Конфигурационный файл приложения содержит следующие параметры:

1. app:
2. name**:** auth-api
3. test**:** true
5. languageSystem:
6. dir**:** 'C:/Users/super/auth\_api/data/models'
8. voiceSystem:
9. dir**:** 'C:/Users/super/auth\_api/data/models/ubm\_16\_diag\_13mfcc.pkl'
10. border**:** 80
12. inputData:
13. dinamicDir**:** 'C:/Users/super/auth\_api/data/load\_data'
15. inventory:
16. dir: 'C:/Users/super/auth\_api/data/users\_data'
17. voiceDirName**:** voice
18. pswdDirName**:** pswd

В первых строках указывается название приложения, режим запуска (test: true/false). В зависимости от режима запуска будет выводиться разная дополнительная информация о проведении аутентификации – при тестовом режиме информация более подробная и помогает отслеживать корректность работы приложения. Именно в этом режиме проводится тестирование системы, описанное в работе ниже. Например, в тестовом режиме можно отследить, какие пользователи внесли в систему полную информацию о себе, а какие нет. В случае полной информации система будет выполнять двухфакторную аутентификацию, иначе только на основе какого-то одного фактора, а второй будет автоматически считаться пройденным.

Параметр border как подпараметр voiceSystem задает границу идентификации по голосу. Если степень уверенности системы выше. Остальные параметры в конфигурационном файле позволяют задать директории для хранения информации о пользователях, переданные ими аудиофайлы и построенные модели.

Все параметры могут быть изменены в процессе работы приложения. После изменения конфигурационного файла необходимо послать сигнал SIGHUP процессу, на котором работает приложение. Поймав этот сигнал, приложение заново перечитает конфигурационный файл и сохранит обновленные настройки.

Для запуска приложения на определенном порте или сокете необходимо передать соответствующие параметры в командной строке при запуске приложения. Также можно передать параметр, указывающий приложению, с каким конфигурационном файлом запускаться (также через командную строку).

При запуске приложение проверяет, что конфигурационный файл корректен (директории, которые были указаны в конфигурационном файле, существуют), иначе завершает работу.

**3.6.4 Описание запросов**

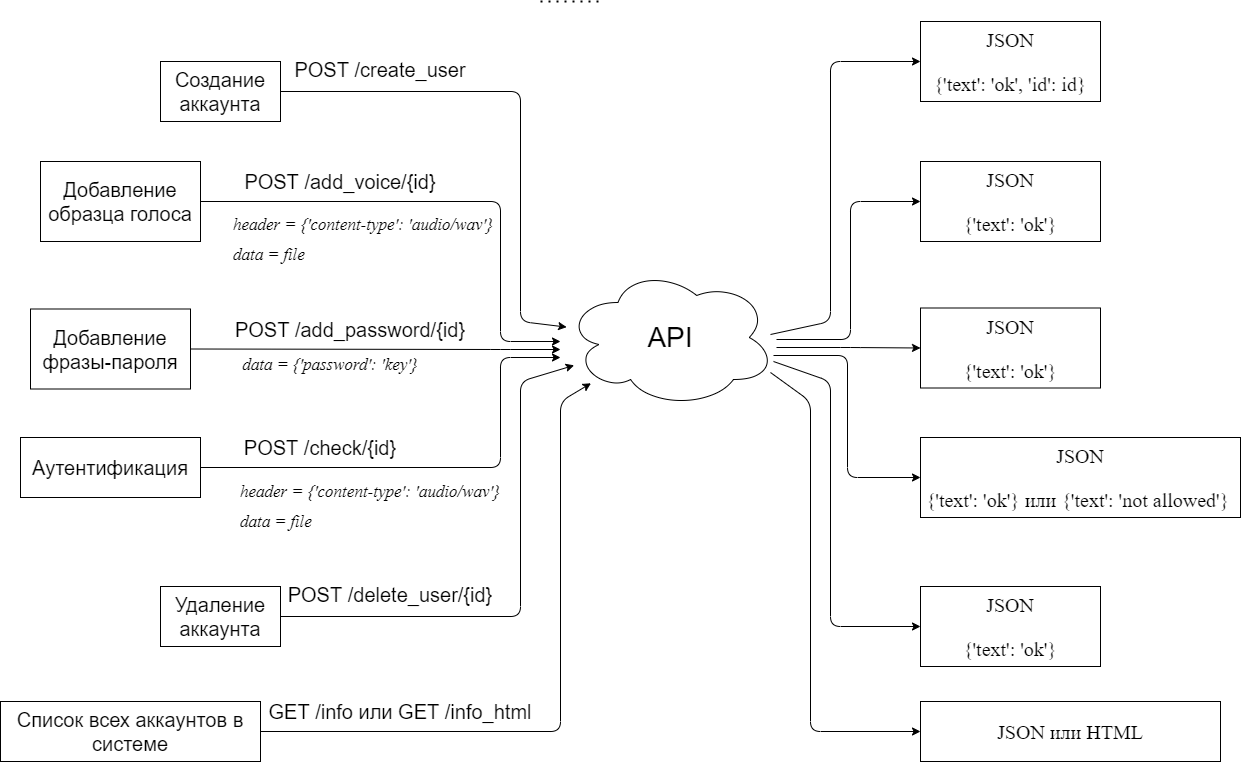
****

Рисунок 44 - Схема запросов к API

Схема запросов к сервису представлена на рисунке 44. Сам сервис в процессе работы итерирует такими сущностями как аккаунт пользователя. Аккаунт может быть создан, удален и дополнен информацией. При создании аккаунта происходит поиск уникального ID, который будет возвращен клиенту и будет представлять из себя его персональный идентификатор. Это что-то вида пароля – указатель на личность, которую в процессе аутентификации и нужно подтверждать. Получив запрос на создание нового аккаунта, система ищет новый ID, создает новую директорию для аккаунта (в котором будет храниться вся информация о нем) и завершает работу, возвращая клиенту статус работы и его новый идентификатор. Создание аккаунта пользователя происходит с помощью POST запроса на */create\_user*.

Затем в аккаунт пользователя нужно добавить информацию о нем. Это может быть добавление образца его голоса или добавление фразы-пароля. Соответственно, существует два отдельных вида запроса – на добавление образца голоса POST запрос на */add\_voice/{id}* и на добавление ключевой фразы POST запрос на */add\_password/{id}*. Здесь в запросе вместо {id} необходимо подставить идентификатор ID того пользователя, информацию к которому нужно добавить. В случае добавления голоса к запросу необходимо добавить заголовок *{'content-type': 'audio/wav'}*, а в качестве данных передать файл с аудиозаписью голоса формата .wav. В случае добавления фразы-пароля нужно передать JSON с полем значением фразы-пароля по ключу ‘password’. С помощью этих запросов можно также перезаписать уже существующую информацию. В этом случае старая информация будет удалена и сохранена новая.

Для удаления аккаунта существует POST запрос /*delete\_user/{id}*. В этом случае будет удалена и вся информация о пользователе, и его аккаунт целиком, и его ID будет считаться свободным для дальнейшей работы.

Есть возможность посмотреть имеющуюся базу данных пользователей системы. Для этого необходимо отправить GET запрос по пути */info*. В ответе клиент получит JSON файл со списком всех аккаунтов пользователей, которые зарегистрированы в системе в данный момент (пример ответа на рисунке 45). Будут учитываться как аккаунты с полной информацией, так и с частичной или даже без добавленной информации.



Рисунок 45 - Пример ответа от сервера с информацией о аккаунтах

Во всех описанных вызовах идентификатор, который должен быть указан в пути запроса, представляет из себя какое-то числовое значение. Если передавать в качестве ID строку символов алфавита, то такой запрос будет считаться ошибочным и код ответа будет 405.

Также реализована возможность получить информацию по всем пользователям в более удобном для чтения формате. Для этого необходимо из браузера перейти по пути */info\_html*. Примерответа изображен на рисунке 46.

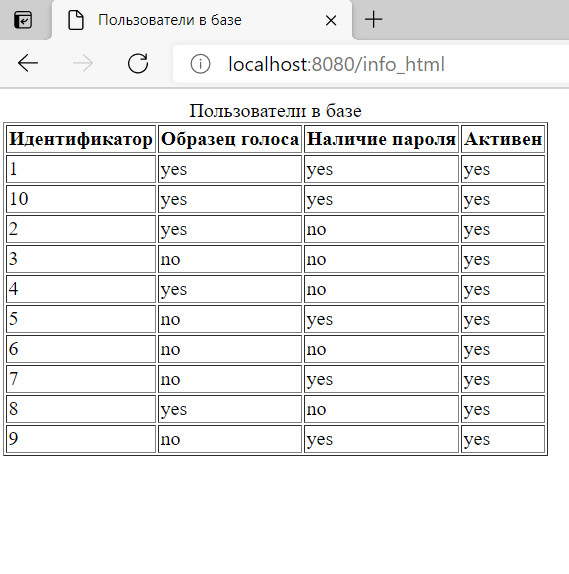


Рисунок 46 - Пример ответа от сервера в HTML формате

К каждому аккаунту добавлено поле активности. Это позволит в будущем отключать какие-то аккаунты из системы аутентификации без удаления имеющейся информации.

Для провидения аутентификации реализован POST запрос */check/{id}*. Также, как и в случае добавления образца голоса, в запросе необходимо передать аудиозапись тем же образом.

Таблица 7 - Формат запросов к сервису

| Метод запроса | URL ресурса | Передача данных | Код ответа | Тело ответа |
| --- | --- | --- | --- | --- |
| POST | /create\_user | - | 200 | JSON  {'text': 'ok', 'id': id} |
| POST | /delete\_user/{id} | - | 200 | JSON  {'text': 'ok'} |
| 400 | JSON  {'text': 'user does not exist'} |
| POST | /add\_voice/{id} | headers = {'content-type': 'audio/wav'}  data = file | 200 | JSON  {'text': 'ok'} |
| 400 | JSON  {'text': 'cant download'} |
| 400 | JSON  {'text': 'user do not exist'} |

Продолжение таблицы 7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод запроса | URL ресурса | Передача данных | Код ответа | Тело ответа |
| POST | /add\_password/{id} | json.dumps({'password': 'key'})  content\_type="application/json" | 200 | JSON  {'text': 'ok'} |
| 400 | JSON  {'text': 'wrong params'} |
| 400 | JSON  {'text': 'user do not exist'} |
| POST | /check/{id} | headers = {'content-type': 'audio/wav'}  data=file | 200 | JSON  {'text': 'ok'} |
| 400 | JSON  {'text': 'cant download file'} |
| 401 | JSON  {'text': 'not allowed'} |
| GET | /info | - | 200 | JSON |
| GET | /info\_html | - | 200 | HTML |

Формат запросов, которые поддерживает приложение, представлен в таблице 7, на рисунке 44 и в Приложении Ж. Обработка запросов описана в Приложении И. Все запросы, которые не соответствуют описанному формату, будут возвращать ответ со статусом 405. В случае, если произошла какая-то крупная ошибка при обработке запроса, сервис вернет код ответа 500.

**3.7 Тестирование системы**

Тестирование работы системы будет состоять из двух больших этапов:

1. Тестирование качества аутентификации,
2. Тестирование качества работы системы.

В процессе первого этапа внимание будет уделяться качеству аутентификации – настолько точные ответы дает система, как часто ошибается. Второй этап необходим для оценки качества реализации, поиска слабых мест и уязвимостей, контроля обработки исключений и частных случаев – из-за чего система может падать, на какие запросы дает неожиданный ответ, как быстро работает.

**3.7.1 Тестирование качества аутентификации**

Достигнута степень успешного распознавания речи составляет около WER = 33 процента, CER = 29 процентов на тестовой выборке. Ошибки в ответах возникают из-за малого объема данных для обучения нейронной сети. Для улучшения работы и повышения точности в дальнейшем следует увеличить количество обучающих данных для качественного обучения акустической модели.

Достигнутая степень успешной идентификации по голосу составляет более 99 процентов на тестовых данных при точности и полноте более 87 и 90 процентов соответственно.

При прохождении аутентификации всей системой каждый пользователь должен проходить аутентификацию на основе обоих факторов. Каждый тестовый пользователь должен обладать как минимум двумя записями речи длительностью в сумме больше 40 секунд (20 секунд на обучение, 20 секунд на тестирование). При этом нарезка на фрагменты должна происходить с учетом недопустимости наличия отдельных неполных произнесенных фрагментов слов в фрагментах, иначе распознавание речи будет давать неправильный ответ при распознавании обрезанного слова. Для тестирования было проведено 1094 тестовых аутентификационных запросов среди созданных 233 аккаунтов пользователей. Результаты работы указаны в таблице 8.

Таблица 8 - Матрица ошибок аутентификации реализованной системы

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | True Positive (187) | False Positive (0) |
|  | False Negative (46) | True Negative (861) |

Полученные метрики качества:



Можно заметить, что точность (precision) составляет 100 процентов. Это достигается именно за счет двухфакторности системы. По сравнению с аутентификацией на основе только голоса точность выросла от 87 до 100 процентов. Однако полнота уменьшилась – выросла доля ответов с вердиктом False Negative. Ошибки системы усиливают друг друга, это отображается именно в этом показателе качества. Аутентификация на основе голоса давала полнота была на 10 процентов выше. Для повышения качества аутентификации рекомендуется увеличивать качество распознавания речи. Как было указано выше, наибольшую долю ошибок в распознавание вносит акустическая система.

**3.7.2 Тестирование качества работы системы**

Тестирование качества реализации системы проводилось с помощью Юнит-тестов (Приложение К). Юнит-тестирование представляет из себя модульное тестирование и позволяет проверить корректность работы отдельных модулей исходного кода программы. «Цель модульного тестирования — изолировать отдельные части программы и показать, что по отдельности эти части работоспособны.» [49]

Написанные тесты проверяют случаи, описанные в таблицах 9-14.

Таблица 9 - Тестовые случаи для пути /create\_user

| Случай | Ожидаемый результат |
| --- | --- |
| Пользователей нет, запросы на создание новых пользователей | Код возврата 200, ID уникальны |
| Пользователи есть, запросы на создание новых пользователей | Код возврата 200, ID уникальны |

Таблица 10 - Тестовые случаи для пути /delete\_user

| Случай | Ожидаемый результат |
| --- | --- |
| Удаление пользователей по тем ID, к которым не привязан ни один аккаунт | Код возврата 400, сообщение об ошибке 'user does not exist' |
| Удаление пользователей по тем ID, к которым привязан аккаунт | Код возврата 200, сообщение 'ok' |
| Проверка, что удаление было произведено – повторный запрос по одному ID | Код возврата 400, сообщение об ошибке 'user does not exist' |
| Удаление аккаунта пользователя с сохраненной информацией | Код возврата 200, сообщение 'ok' |
| Удаление аккаунта пользователя без сохраненной информации | Код возврата 200, сообщение 'ok' |

Таблица 11 - Тестовые случаи для пути /add\_voice

| Случай | Ожидаемый результат |
| --- | --- |
| У существующего пользователя не было образца голоса. Добавление образца голоса к пользователю | Код возврата 200, сообщение 'ok' |
| Добавление образца голоса к несуществующему пользователю | Код возврата 400, сообщение об ошибке 'user does not exist' |
| У существующего пользователя уже был образец голоса. Обновление образца голоса | Код возврата 200, сообщение 'ok' |
| Отправка запроса без вложенного файла | Код возврата 400, сообщение об ошибке 'cant download' |

Таблица 12 - Тестовые случаи для пути /add\_password

| Случай | Ожидаемый результат |
| --- | --- |
| У существующего пользователя не было фразы-пароля. Добавление пароля к пользователю | Код возврата 200, сообщение 'ok' |
| Добавление фразы-пароля к несуществующему пользователю | Код возврата 400, сообщение об ошибке 'user does not exist' |
| У существующего пользователя уже была фраза-пароль. Обновление пароля | Код возврата 200, сообщение 'ok' |
| Отправка запроса без пароля или с пустым паролем | Код возврата 400, сообщение об ошибке 'wrong params' |

Таблица 13 - Тестовые случаи для пути /check

| Случай | Ожидаемый результат |
| --- | --- |
| Аутентификация пользователя, аккаунт которого не содержит ни образца голоса, ни фразы-пароля | Код возврата 200, сообщение {‘text’: ‘ok’, ‘voice’: ‘not done’, ‘speech’: ‘not done’} |
| Аутентификация пользователя, аккаунт которого не содержит образца голоса, но содержит фразу-пароль (1 фактор авторизации) | Код возврата 200, сообщение {‘text’: ‘ok’, ‘voice’: ‘not done’} |
| Аутентификация пользователя, аккаунт которого не содержит фразы-пароля, но содержит образец голоса (1 фактор авторизации) | Код возврата 200, сообщение {‘text’: ‘ok’, ‘speech’: ‘not done’} |
| Аутентификация пользователя, аккаунт которого содержит и образец голоса, и фразу-пароль | Код возврата 200, сообщение {‘text’: ‘ok’} |
| Аутентификация не существующего пользователя | Код возврата 401, сообщение {‘text’: ‘not allowed’} |

Таблица 14 - Тестовые случаи для пути /info

| Случай | Ожидаемый результат |
| --- | --- |
| Аккаунт пользователя, у которого не сохранен ни образец голоса, ни фраза-пароль | Статус 200, сообщение {‘voice\_exist’: ‘no’, ‘password\_exist’: ‘no’, ‘active’: ‘yes'} |
| Аккаунт пользователя, у которого не сохранен образец голоса, сохранена фраза-пароль | Статус 200, сообщение {‘voice\_exist’: ‘no’, ‘password\_exist’: ‘yes’, ‘active’: ‘yes'} |
| Аккаунт пользователя, у которого не сохранена фраза-пароль, сохранен образец голоса | Статус 200, сообщение {‘voice\_exist’: ‘yes’, ‘password\_exist’: ‘no’, ‘active’: ‘yes'} |
| Аккаунт пользователя, у которого сохранен образец голоса, сохранена фраза-пароль | Статус 200, сообщение {‘voice\_exist’: ‘yes’, ‘password\_exist’: ‘yes’, ‘active’: ‘yes'} |
| Аккаунт не существующего пользователя | Не отображается в ответе |

Также в процессе тестирования были измерены показатели работы системы. Они указаны в таблице 15. Сервис был поднят на том же хосте, с которого к нему приходили клиентские запросы. Хост имеет следующую конфигурацию:

* Система: Windows 10 Pro,
* Процессор: Intel Core i5-8250u 1.60ghz,
* ОЗУ: 8 ГБ.

Таблица 15 - Измеренные показатели системы

| Показатель | Значение |
| --- | --- |
| Среднее время обработки запроса | 2651 мс |
| Среднее время передачи аудиофайла (длительность 20 сек) | 123 мс |
| Среднее время обработки и распознавания | 2520 мс |

Наибольшая часть времени при обработке запроса на авторизацию уходит на обработку аудиофайла – на предобработку и распознавание. Таким образом, поставленное требование распределения временных ресурсов было достигнуто. Основные ресурсы при работе сервиса тратятся на работу системы распознавания речи и на работу системы идентификации по голосу. Время передачи аудиофайла и работы системы не вносит значительный вклад в общее время обработки запроса. Другие требования к приложению также были выполнены.

**3.8 Выводы**

В процессе разработки и экспериментальных исследований была получена оптимальная архитектура модели для распознавания речи и оптимальная модель для идентификации говорящего по голосу. Из полученных моделей затем была сформирована полноценная система аутентификации и реализована в виде API. Система обладает хорошими показателями точности и удовлетворяет всем сформулированным требованиям.

**ЗАКЛЮЧЕНИЕ**

В результате выполнения выпускной квалификационной работы была достигнута главная цель – создана система, повышающая защищенность систем от несанкционированного доступа за счет проведения двухфакторной аутентификации зарегистрированных в ней пользователей, с использованием разработанного компонента распознавания речи из аудиосигнала и компонента идентификации личности диктора по его голосу в полученной аудиозаписи. Система реализована в виде API. В ходе работы были выполнены следующие задачи:

1. выполнен обзор существующих методов распознавания речи;
2. выполнен обзор существующих методов идентификации по голосу;
3. получены размеченные обучающие данные;
4. разработан метод распознавания речи;
5. экспериментально определена оптимальная реализация метода распознавания речи;
6. разработан метод идентификации по голосу;
7. экспериментально определена оптимальная реализация метода идентификации по голосу;
8. реализована система аутентификации в виде API;
9. проведено тестирование системы и определена степень надежности.

Достигнутые показатели качества работы разработанной системы:

* точность (precision) составляет 100 процентов;
* полнота (recall) составляет 80 процентов;
* точность (accuracy) составляет 95,8 процентов.

Достигнутая степень успешной идентификации по голосу составляет более 99 процентов (accuracy) при точности (precision) и полноте (recall) более 87 процентов и 90 процентов соответственно. Показатели качества распознавания речи составляют WER = 33 процента, CER = 29 процентов. Ошибки при аутентификации возникают преимущественно из-за ошибок при распознавании речи. В дальнейшем для улучшения работы и повышения точности распознавания речи следует увеличить объем обучающих данных для качественного обучения акустической модели. Областью применения системы является любая задача аутентификации. При необходимости, можно добавить компонент фильтрации аудиосигнала перед распознаванием, что позволит давать качественный результат в большем количестве практических задач.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. M.A.Anusuya, S.K.Katti Speech Recognition by Machine: A Review, 2009 URL:https://arxiv.org/pdf/1001.2267.pdf
2. Кодзасов С. В. Артикуляция // Лингвистический энциклопедический словарь / Главный редактор В. Н. Ярцева. — М.: Советская энциклопедия, 1990 https://www.fon.hum.uva.nl/praat/manual/Rabiner\_\_1989\_.html
3. L.R. Rabiner: "A tutorial on Hidden Markov Models and selected applications in speech recognition." Proceedings of the IEEE 77: 257–286. 1989
4. Juri Lember, Alexey A. Koloydenko A generalized risk-based approach to segmentation based on hidden Markov models, URL:https://arxiv.org/pdf/1007.3622v1.pdf
5. Pablo Gimeno Jordán A Voice Activity Detection Tool Based on Recurrent Neural Networks, 2019 URL:https://medium.com/vivolab/vivovad-a-voice-activity-detection-tool-based-on-recurrent-neural-networks-32356526321c
6. Открытые лекции компании Яндекс // Распознавание речи, 2013. URL: https://habr.com/ru/company/yandex/blog/198556.
7. Алборова Ж. В., Рубцов В.И. Алгоритм и методы распознавания речи // МГТУ им. Н.Э. Баумана, 2016. URL: http://ainsnt.ru/file/out/841241.
8. Екатерина Черняк. Глубинное обучение в обработке и анализе текстов, 2018. URL: https://postnauka.ru/longreads/85951
9. Гребнов С.В. Аналитический обзор методов распознавания речи в системах голосового управления, 2009. URL: http://vestnik.ispu.ru/sites/vestnik.ispu.ru/files/publications/83-85\_0.pdf.
10. Wikipedia, URL: https://ru.wikipedia.org/wiki/N-грамма (дата обращения 20.10.2020).
11. Andrew Gibiansky. Speech Recognition with Neural Networks, 2014. URL: https://andrew.gibiansky.com/blog/machine-learning/speech-recognition-neural-networks/.
12. Периодические нейронные сети и LSTM, Никлас Донгес, TowardsDataScience.com,https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5 (дата обращения 30.09.2020)
13. Stefan Leijnen The Neural Network Zoo - The Asimov Institute, 2016. URL: https://www.asimovinstitute.org/author/stefanleijnen/
14. Alex Graves, Abdel-rahman Mohamed, Geoffrey Hinton. Speech recognition with deep recurrent neural networks, University of Toronto. URL: https://arxiv.org/pdf/1303.5778.pdf
15. Pablo Gimeno Jordán. A Voice Activity Detection Tool Based on Recurrent Neural Networks, 2019. URL: https://medium.com/vivolab/vivovad-a-voice-activity-detection-tool-based-on-recurrent-neural-networks-32356526321c.
16. Andrey Sovse. Rus speech recognition. URL: https://github.com/sovse/Rus-SpeechRecognition-LSTM-CTC-VoxForge.
17. Глубокое обучение с использованием Keras: (V) объяснение и практика RNN и двухстороннего RNN. URL:https://blog.csdn.net/fendouaini/article/details/80096717
18. Mike Schuster, Kuldip K. Paliwal. Bidirectional recurrent neural networks, 1997. URL: https://www.researchgate.net/publication/3316656\_Bidirectional\_recurrent\_neural\_networks
19. Ozhiganov I. How Neural Networks Recognize Speech-to-Text, 2019. URL: https://dzone.com/articles/how-to-train-a-neural-network-to-recognize-speech.
20. Xuejiao Li, Zixuan Zhou. Speech Command Recognition with Convolutional Neural Network. URL: http://cs229.stanford.edu/proj2017/final-reports/5244201.pdf.
21. Dimitri Palaz, Mathew Magimai-Doss, Ronan Collobert. Convolutional neural networks-based continuous speech recognition using raw speech signal. URL: https://ronan.collobert.com/pub/matos/2015\_rawspeech\_icassp.pdf.
22. Глубокое обучение для новичков: распознаем изображения с помощью сверточных сетей, 2016. URL: https://www.pvsm.ru/algoritmy/210060.
23. CS231n: Convolutional Neural Networks for Visual Recognition, Stanford, Spring 2021. URL: https://cs231n.github.io/convolutional-networks/#pool
24. Henrik I. Christensen. CSE 291 - Pattern Recognition – Introduction. URL: http://www.hichristensen.net/CSE291/pdf/lecture1.pdf
25. Фролов А.В., Фролов Г.В. Синтез и распознавание речи // Современные решения. URL: https://frolov-lib.ru/books/hi/ch05.html.
26. Ле, Нгуен Виен. Распознавание речи на основе искусственных нейронных сетей / Нгуен Виен Ле, Д. П. Панченко. URL: https://moluch.ru/conf/tech/archive/3/712/ (дата обращения: 20.10.2020).
27. Девятов М. Программы распознавания речи // Что понимает искусственный интеллект, 2017. URL: https://www.primavista.ru/blog/2017/08/03/programmyi-raspoznavaniya-rechi
28. Докуметация к библиотеке librosa. URL: https://librosa.org/doc/main/generated/librosa.feature.delta.html
29. Bimbot F. et al. A tutorial on text-independent speaker verification, 2004. Стр. 430-451. URL: https://studylib.net/doc/12607069/a-tutorial-on-text-independent-speaker-verification-pleas...
30. Reynolds D., Rose R. Robust text-independent speaker identification using Gaussian mixture speaker models, 1995. Стр. 72–83
31. Reynolds D. Experimental evaluation of features for robust speaker identification, 1994. Стр. 639–643
32. Бучнева Т.И., Кудряшов М.Ю. Нейронные сети в задаче идентификации диктора по голосу, 2015. URL: http://pmk-vestnik.tversu.ru/issues/2015-2/vestnik-pmk-2015-2-buchneva.pdf.
33. Рахманенко И. А., Шелупанов А. А., Костюченко Е. Ю. Автоматическая верификация диктора по произвольной фразе с применением свёрточных глубоких сетей доверия. URL: https://cyberleninka.ru/article/n/avtomaticheskaya-verifikatsiya-diktora-po-proizvolnoy-fraze-s-primeneniem-svyortochnyh-glubokih-setey-doveriya.
34. Сорокин В.Н., Вьюгин В.В., Тананыкин А.А. Распознавание личности по голосу: аналитический обзор. URL: http://www.jip.ru/2012/1-30-2012.pdf.
35. Единая биометрическая система. URL: https://bio.rt.ru/citizens.
36. TAdviser. Сбербанк вводит возможность голосового подтверждения операций. URL: https://www.tadviser.ru/a/37158.
37. Российская газета RG // Биометрия, 2021. URL:https://rg.ru/2021/03/10/mincifry-biometriia-dlia-polucheniia-gosuslug-ne-budet-obiazatelnoj.html.
38. Алексей Лукацкий. Голосовая биометрия // Краткий обзор технологии, 2015. URL: https://www.securitylab.ru/blog/personal/Business\_without\_danger/147943.php.
39. Oscar Contreras Carrasco. Gaussian Mixture Models Explained // From intuition to implementation, 2019. URL: https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95
40. Jake VanderPlas. In Depth: Gaussian Mixture Models. URL: https://jakevdp.github.io/PythonDataScienceHandbook/05.12-gaussian-mixtures.html
41. Stephen M Chu, Daniel Povey. Universal background model based speech recognition, 2008. URL: https://www.researchgate.net/publication/224762192\_Universal\_background\_model\_based\_speech\_recognition
42. Matsui T., Furui S. Likelihood normalization for speaker verification using a phoneme-and speaker-independent model, 1995. Стр. 109-116.
43. Reynolds D. A. Comparison of background normalization methods for text-independent speaker verification, 1997. Стр. 963-966.
44. Hermansky H., Malayath N. Speaker verification using speaker-specific mappings, 1998. Стр.111-114.
45. Quatieri T. F. et al. Speaker and language recognition using speech codec parameters, 1999. Стр. 787-790.
46. Abid Mohsin. Podcast Transcription Benchmark, 2019. URL: https://www.rev.ai/blog/podcast-transcription-benchmark-part-1/
47. Кленов К. Измерение производительности популярных python web-фреймворков, 2015. URL: https://klen.github.io/python-web-benchmarks.html
48. Joan Miquel Web Framework Benchmarks, 2019. URL: https://www.techempower.com/benchmarks/#section=data-r18&hw=ph&test=fortune&l=zijzen-1
49. Модульное тестирование URL:https://ru.wikipedia.org/wiki/Модульное\_тестирование

**ПРИЛОЖЕНИЕ А**

**Код файла AudioGenerator**

1. **import** json
2. **import** numpy **as** np
3. **import** random
4. **from** python\_speech\_features **import** mfcc
5. **import librosa**
6. **import** scipy.io.wavfile **as** wav
7. **import** matplotlib.pyplot **as** plt
8. **from** mpl\_toolkits.axes\_grid1 **import** make\_axes\_locatable
9. **from** data\_utils **import** calc\_feat\_dim, spectrogram\_from\_file, text\_to\_int\_sequence
10. **from data\_utils import conv\_output\_length**
12. RNG\_SEED = 327
14. **class** AudioGenerator():
15. **def \_\_init\_\_(self, step=10, window=20, max\_freq=8000, mfcc\_dim=13,minibatch\_size=20, desc\_file=None, spectrogram=True, max\_duration=10.0,**
16. sort\_by\_duration=False):
17. self.feat\_dim = calc\_feat\_dim(window, max\_freq)
18. self.mfcc\_dim = mfcc\_dim
19. self.feats\_mean = np.zeros((self.feat\_dim,))
20. **self.feats\_std = np.ones((self.feat\_dim,))**
21. self.rng = random.Random(RNG\_SEED)
22. **if** desc\_file **is** **not** None:
23. self.load\_metadata\_from\_desc\_file(desc\_file)
24. self.step = step
25. **self.window = window**
26. self.max\_duration = max\_duration
27. self.minibatch\_size = minibatch\_size
28. self.spectrogram = spectrogram
29. self.sort\_by\_duration = sort\_by\_duration
30. **self.max\_freq = max\_freq**
31. self.cur\_train\_index = 0
32. self.cur\_valid\_index = 0
33. self.cur\_test\_index = 0
35. **def get\_batch(self, partition):**
36. **if** partition == 'train':
37. audio\_paths = self.train\_audio\_paths
38. cur\_index = self.cur\_train\_index
39. texts = self.train\_texts
40. **elif partition == 'valid':**
41. audio\_paths = self.valid\_audio\_paths
42. cur\_index = self.cur\_valid\_index
43. texts = self.valid\_texts
44. **elif** partition == 'test':
45. **audio\_paths = self.test\_audio\_paths**
46. cur\_index = self.test\_valid\_index
47. texts = self.test\_texts
48. **else**:
49. **raise** Exception("Invalid partition. Must be train/validation")
50. **features = [self.normalize(self.featurize(a)) for a in**
51. audio\_paths[cur\_index:cur\_index+self.minibatch\_size]]
52. max\_length = max([features[i].shape[0]
53. **for** i **in** range(0, self.minibatch\_size)])
54. max\_string\_length = max([len(texts[cur\_index+i])
56. **for** i **in** range(0, self.minibatch\_size)])
57. X\_data = np.zeros([self.minibatch\_size, max\_length,
58. self.feat\_dim\*self.spectrogram + self.mfcc\_dim\*(**not** self.spectrogram)])
59. labels = np.ones([self.minibatch\_size, max\_string\_length]) \* 28
60. **input\_length = np.zeros([self.minibatch\_size, 1])**
61. label\_length = np.zeros([self.minibatch\_size, 1])
63. **for** i **in** range(0, self.minibatch\_size):
64. feat = features[i]
65. **input\_length[i] = feat.shape[0]**
66. X\_data[i, :feat.shape[0], :] = feat
67. label = np.array(text\_to\_int\_sequence(texts[cur\_index+i]))
68. labels[i, :len(label)] = label
69. label\_length[i] = len(label)
71. outputs = {'ctc': np.zeros([self.minibatch\_size])}
72. inputs = {'the\_input': X\_data,
73. 'the\_labels': labels,
74. 'input\_length': input\_length,
75. **'label\_length': label\_length**
76. }
77. **return** (inputs, outputs)
79. **def** shuffle\_data\_by\_partition(self, partition):
80. **if partition == 'train':**
81. self.train\_audio\_paths, self.train\_durations, self.train\_texts = shuffle\_data(
82. self.train\_audio\_paths, self.train\_durations, self.train\_texts)
83. **elif** partition == 'valid':
84. self.valid\_audio\_paths, self.valid\_durations, self.valid\_texts = shuffle\_data(
85. **self.valid\_audio\_paths, self.valid\_durations, self.valid\_texts)**
86. **else**:
87. **raise** Exception("Invalid partition. Must be train/validation")
89. **def** sort\_data\_by\_duration(self, partition):
90. **if partition == 'train':**
91. self.train\_audio\_paths, self.train\_durations, self.train\_texts = sort\_data(
92. self.train\_audio\_paths, self.train\_durations, self.train\_texts)
93. **elif** partition == 'valid':
94. self.valid\_audio\_paths, self.valid\_durations, self.valid\_texts = sort\_data(
95. **self.valid\_audio\_paths, self.valid\_durations, self.valid\_texts)**
96. **else**:
97. **raise** Exception("Invalid partition. "
98. "Must be train/validation")
100. **def next\_train(self):**
101. **while** True:
102. ret = self.get\_batch('train')
103. self.cur\_train\_index += self.minibatch\_size
104. **if** self.cur\_train\_index >= len(self.train\_texts) - self.minibatch\_size:
105. **self.cur\_train\_index = 0**
106. self.shuffle\_data\_by\_partition('train')
107. **yield** ret
109. **def** next\_valid(self):
110. **while True:**
111. ret = self.get\_batch('valid')
112. self.cur\_valid\_index += self.minibatch\_size
113. **if** self.cur\_valid\_index >= len(self.valid\_texts) - self.minibatch\_size:
114. self.cur\_valid\_index = 0
115. **self.shuffle\_data\_by\_partition('valid')**
116. **yield** ret
118. **def** next\_test(self):
119. **while** True:
120. **ret = self.get\_batch('test')**
121. self.cur\_test\_index += self.minibatch\_size
122. **if** self.cur\_test\_index >= len(self.test\_texts) - self.minibatch\_size:
123. self.cur\_test\_index = 0
124. **yield** ret
126. **def** load\_train\_data(self, desc\_file='train\_corpus.json'):
127. self.load\_metadata\_from\_desc\_file(desc\_file, 'train')
128. self.fit\_train()
129. **if** self.sort\_by\_duration:
130. **self.sort\_data\_by\_duration('train')**
132. **def** load\_validation\_data(self, desc\_file='valid\_corpus.json'):
133. self.load\_metadata\_from\_desc\_file(desc\_file, 'validation')
134. **if** self.sort\_by\_duration:
135. **self.sort\_data\_by\_duration('valid')**
137. **def** load\_test\_data(self, desc\_file='test\_corpus.json'):
138. self.load\_metadata\_from\_desc\_file(desc\_file, 'test')
140. **def load\_metadata\_from\_desc\_file(self, desc\_file, partition):**
141. audio\_paths, durations, texts = [], [], []
142. **with** open(desc\_file) **as** json\_line\_file:
143. **for** line\_num, json\_line **in** enumerate(json\_line\_file):
144. **try**:
145. **spec = json.loads(json\_line)**
146. **if** float(spec['duration']) > self.max\_duration:
147. **continue**
148. audio\_paths.append(spec['key'])
149. durations.append(float(spec['duration']))
150. **texts.append(spec['text'])**
151. **except** Exception **as** e:
152. **print**('Error reading line #{}: {}'
153. .format(line\_num, json\_line))
154. **if** partition == 'train':
155. **self.train\_audio\_paths = audio\_paths**
156. self.train\_durations = durations
157. self.train\_texts = texts
158. **elif** partition == 'validation':
159. self.valid\_audio\_paths = audio\_paths
160. **self.valid\_durations = durations**
161. self.valid\_texts = texts
162. **elif** partition == 'test':
163. self.test\_audio\_paths = audio\_paths
164. self.test\_durations = durations
165. **self.test\_texts = texts**
166. **else**:
167. **raise** Exception("Invalid partition to load metadata. Must be train/validation/test")
169. **def** fit\_train(self, k\_samples=100):
170. **k\_samples = min(k\_samples, len(self.train\_audio\_paths))**
171. samples = self.rng.sample(self.train\_audio\_paths, k\_samples)
172. feats = [self.featurize(s) **for** s **in** samples]
173. feats = np.vstack(feats)
174. self.feats\_mean = np.mean(feats, axis=0)
175. **self.feats\_std = np.std(feats, axis=0)**
177. **def** featurize(self, audio\_clip):
178. **if** self.spectrogram:
179. **return** spectrogram\_from\_file(
180. **audio\_clip, step=self.step, window=self.window,**
181. max\_freq=self.max\_freq)
182. **else**:
183. (rate, sig) = wav.read(audio\_clip)
184. **return** mfcc(sig, rate, numcep=self.mfcc\_dim)
186. **def** normalize(self, feature, eps=1e-14):
187. **return** (feature - self.feats\_mean) / (self.feats\_std + eps)
189. **def** shuffle\_data(audio\_paths, durations, texts):
190. **p = np.random.permutation(len(audio\_paths))**
191. audio\_paths = [audio\_paths[i] **for** i **in** p]
192. durations = [durations[i] **for** i **in** p]
193. texts = [texts[i] **for** i **in** p]
194. **return** audio\_paths, durations, texts
196. **def** sort\_data(audio\_paths, durations, texts):
197. p = np.argsort(durations).tolist()
198. audio\_paths = [audio\_paths[i] **for** i **in** p]
199. durations = [durations[i] **for** i **in** p]
200. **texts = [texts[i] for i in p]**
201. **return** audio\_paths, durations, texts
203. **def** vis\_train\_features(index=0):
204. audio\_gen = AudioGenerator(spectrogram=True)
205. **audio\_gen.load\_train\_data()**
206. vis\_audio\_path = audio\_gen.train\_audio\_paths[index]
207. vis\_spectrogram\_feature = audio\_gen.normalize(audio\_gen.featurize(vis\_audio\_path))
208. audio\_gen = AudioGenerator(spectrogram=False)
209. audio\_gen.load\_train\_data()
210. **vis\_mfcc\_feature = audio\_gen.normalize(audio\_gen.featurize(vis\_audio\_path))**
211. vis\_text = audio\_gen.train\_texts[index]
212. vis\_raw\_audio, \_ = librosa.load(vis\_audio\_path)
213. **return** vis\_text, vis\_raw\_audio, vis\_mfcc\_feature, vis\_spectrogram\_feature, vis\_audio\_path

216. **def** plot\_raw\_audio(vis\_raw\_audio):
217. *# plot the raw audio signal*
218. fig = plt.figure(figsize=(12,3))
219. ax = fig.add\_subplot(111)
220. **steps = len(vis\_raw\_audio)**
221. ax.plot(np.linspace(1, steps, steps), vis\_raw\_audio)
222. plt.title('Audio Signal')
223. plt.xlabel('Time')
224. plt.ylabel('Amplitude')
225. **plt.show()**
227. **def** plot\_mfcc\_feature(vis\_mfcc\_feature):
228. fig = plt.figure(figsize=(12,5))
229. ax = fig.add\_subplot(111)
230. **im = ax.imshow(vis\_mfcc\_feature, cmap=plt.cm.jet, aspect='auto')**
231. plt.title('Normalized MFCC')
232. plt.ylabel('Time')
233. plt.xlabel('MFCC Coefficient')
234. divider = make\_axes\_locatable(ax)
235. **cax = divider.append\_axes("right", size="5%", pad=0.05)**
236. plt.colorbar(im, cax=cax)
237. ax.set\_xticks(np.arange(0, 13, 2), minor=False);
238. plt.show()
240. **def plot\_spectrogram\_feature(vis\_spectrogram\_feature):**
241. fig = plt.figure(figsize=(12,5))
242. ax = fig.add\_subplot(111)
243. im = ax.imshow(vis\_spectrogram\_feature, cmap=plt.cm.jet, aspect='auto')
244. plt.title('Normalized Spectrogram')
245. **plt.ylabel('Time')**
246. plt.xlabel('Frequency')
247. divider = make\_axes\_locatable(ax)
248. cax = divider.append\_axes("right", size="5%", pad=0.05)
249. plt.colorbar(im, cax=cax)
250. **plt.show()**

**ПРИЛОЖЕНИЕ Б**

**Код Models**

1. **from** keras **import** backend **as** K
2. **from** keras.models **import** Model
3. **from** keras.layers **import** (BatchNormalization, Conv1D, Dense, Input,
4. TimeDistributed, Activation, Bidirectional, SimpleRNN, GRU, LSTM)
6. **class** Models():
7. **def** simple\_rnn\_model(input\_dim, output\_dim=29):
8. input\_data = Input(name='the\_input', shape=(None, input\_dim))
9. simp\_rnn = GRU(output\_dim, return\_sequences=True,
10. **implementation=2, name='rnn')(input\_data)**
11. y\_pred = Activation('softmax', name='softmax')(simp\_rnn)
12. model = Model(inputs=input\_data, outputs=y\_pred)
13. model.output\_length = **lambda** x: x
14. **print**(model.summary())
15. **return model**
17. **def** rnn\_model(input\_dim, units, activation, output\_dim=29):
18. input\_data = Input(name='the\_input', shape=(None, input\_dim))
19. simp\_rnn = LSTM(units, activation=activation,
20. **return\_sequences=True, implementation=2, name='rnn')(input\_data)**
21. bn\_rnn = BatchNormalization(name='bn\_rnn\_1d')(simp\_rnn)
22. time\_dense = TimeDistributed(Dense(output\_dim))(bn\_rnn)
23. y\_pred = Activation('softmax', name='softmax')(time\_dense)
24. model = Model(inputs=input\_data, outputs=y\_pred)
25. **model.output\_length = lambda x: x**
26. **print**(model.summary())
27. **return** model

30. **def cnn\_rnn\_model(input\_dim, filters, kernel\_size, conv\_stride,**
31. conv\_border\_mode, units, output\_dim=29):
32. input\_data = Input(name='the\_input', shape=(None, input\_dim))
33. conv\_1d = Conv1D(filters, kernel\_size,
34. strides=conv\_stride,
35. **padding=conv\_border\_mode,**
36. activation='relu',
37. name='conv1d')(input\_data)
38. bn\_cnn = BatchNormalization(name='bn\_conv\_1d')(conv\_1d)
39. simp\_rnn = GRU(units, activation='relu',
40. **return\_sequences=True, implementation=2, name='rnn')(bn\_cnn)**
41. bn\_rnn = BatchNormalization(name='bn\_rnn\_1d')(simp\_rnn)
42. time\_dense = TimeDistributed(Dense(output\_dim))(bn\_rnn)
43. y\_pred = Activation('softmax', name='softmax')(time\_dense)
44. model = Model(inputs=input\_data, outputs=y\_pred)
45. **model.output\_length = lambda x: cnn\_output\_length(**
46. x, kernel\_size, conv\_border\_mode, conv\_stride)
47. **print**(model.summary())
48. **return** model
50. **def cnn\_output\_length(input\_length, filter\_size, border\_mode, stride, dilation=1):**
51. **if** input\_length **is** None:
52. **return** None
53. **assert** border\_mode **in** {'same', 'valid'}
54. dilated\_filter\_size = filter\_size + (filter\_size - 1) \* (dilation - 1)
55. **if border\_mode == 'same':**
56. output\_length = input\_length
57. **elif** border\_mode == 'valid':
58. output\_length = input\_length - dilated\_filter\_size + 1
59. **return** (output\_length + stride - 1) // stride
61. **def** deep\_rnn\_model(input\_dim, units, recur\_layers, output\_dim = 29):
62. input\_data = Input(name = 'the\_input', shape=(None, input\_dim))
63. **if** recur\_layers == 1:
64. layer = LSTM(units, return\_sequences=True, activation='relu')(input\_data)
65. **layer = BatchNormalization(name='bt\_rnn\_1')(layer)**
66. **else**:
67. layer = LSTM(units, return\_sequences=True, activation='relu')(input\_data)
68. layer = BatchNormalization(name='bt\_rnn\_1')(layer)
70. **for i in range(recur\_layers - 2):**
71. layer = LSTM(units, return\_sequences=True, activation='relu')(layer)
72. layer = BatchNormalization(name='bt\_rnn\_{}'.format(2+i))(layer)
74. layer = LSTM(units, return\_sequences=True, activation='relu')(layer)
75. **layer = BatchNormalization(name='bt\_rnn\_last\_rnn')(layer)**
77. time\_dense = TimeDistributed(Dense(output\_dim))(layer)
78. y\_pred = Activation('softmax', name='softmax')(time\_dense)
79. model = Model(inputs=input\_data, outputs=y\_pred)
80. **model.output\_length = lambda x: x**
81. **print**(model.summary())
82. **return** model
84. **def** bidirectional\_rnn\_model(input\_dim, units, output\_dim=29):
85. **input\_data = Input(name='the\_input', shape=(None, input\_dim))**
86. bidir\_rnn = Bidirectional(LSTM(units, return\_sequences=True, activation='relu'), merge\_mode='concat')(input\_data)
87. time\_dense = TimeDistributed(Dense(output\_dim))(bidir\_rnn)
88. y\_pred = Activation('softmax', name='softmax')(time\_dense)
89. model = Model(inputs=input\_data, outputs=y\_pred)
90. **model.output\_length = lambda x: x**
91. **print**(model.summary())
92. **return** model
94. **def** final\_model(input\_dim, filters, kernel\_size, conv\_stride,
95. **conv\_border\_mode, units, output\_dim=29, dropout\_rate=0.5, number\_of\_layers=2,**
96. cell=GRU, activation='tanh'):
97. input\_data = Input(name='the\_input', shape=(None, input\_dim))
98. conv\_1d = Conv1D(filters, kernel\_size,
99. strides=conv\_stride,
100. **padding=conv\_border\_mode,**
101. activation='relu',
102. name='layer\_1\_conv',
103. dilation\_rate=1)(input\_data)
104. conv\_bn = BatchNormalization(name='conv\_batch\_norm')(conv\_1d)
105. **if number\_of\_layers == 1:**
106. layer = cell(units, activation=activation,
107. return\_sequences=True, implementation=2, name='rnn\_1', dropout=dropout\_rate)(conv\_bn)
108. layer = BatchNormalization(name='bt\_rnn\_1')(layer)
109. **else**:
110. **layer = cell(units, activation=activation,**
111. return\_sequences=True, implementation=2, name='rnn\_1', dropout=dropout\_rate)(conv\_bn)
112. layer = BatchNormalization(name='bt\_rnn\_1')(layer)
114. **for** i **in** range(number\_of\_layers - 2):
115. **layer = cell(units, activation=activation,**
116. return\_sequences=True, implementation=2, name='rnn\_{}'.format(i+2), dropout=dropout\_rate)(layer)
117. layer = BatchNormalization(name='bt\_rnn\_{}'.format(i+2))(layer)
119. layer = cell(units, activation=activation,
120. **return\_sequences=True, implementation=2, name='final\_layer\_of\_rnn')(layer)**
121. layer = BatchNormalization(name='bt\_rnn\_final')(layer)
122. time\_dense = TimeDistributed(Dense(output\_dim))(layer)
123. y\_pred = Activation('softmax', name='softmax')(time\_dense)
124. model = Model(inputs=input\_data, outputs=y\_pred)
125. **model.output\_length = lambda x: cnn\_output\_length(**
126. x, kernel\_size, conv\_border\_mode, conv\_stride)
127. **print**(model.summary())
128. **return** model

**ПРИЛОЖЕНИЕ В**

**Код LanguageModel**

1. **from** \_\_future\_\_ **import** division
2. **import** numpy **as** np
3. **import** nltk
4. **from** nltk.metrics **import** \*
5. **from nltk.util import ngrams**
6. **import** enchant
7. **from** enchant.checker **import** SpellChecker
8. **from** nltk.stem **import** PorterStemmer
9. **from** nltk.corpus **import** words
11. spell\_dictionary = enchant.Dict('en')
12. **class** LanguageModel:
13. **def** \_\_init\_\_(self,dictionary):
14. self.dictionary = dictionary
15. **self.check = SpellChecker("en\_US")**
16. self.stemmer = PorterStemmer()
18. **def** levenshtein\_distance(self,s1,s2):
19. distance\_btw\_strings = edit\_distance(s1,s2)
20. **return distance\_btw\_strings**
22. **def** ngram(self,word,n):
23. grams = list(ngrams(word,n))
24. **return** grams
26. **def** check\_mistakes\_in\_sentence(self,sentence):
27. misspelled\_words = []
28. self.check.set\_text(sentence)
29. **for** err **in** self.check:
30. **misspelled\_words.append(err.word)**
32. **if** len(misspelled\_words) == 0:
33. **print** " No mistakes found"
34. **return** misspelled\_words
36. **def** jaccard(self,a,b):
37. union = list(set(a+b))
38. intersection = list(set(a) - (set(a)-set(b)))
39. jaccard\_coeff = float(len(intersection))/len(union)
40. **return jaccard\_coeff**
42. **def** minimumEditDistance\_spell\_corrector(self,word):
43. max\_distance = 2
44. **if** (self.dictionary.check(word)):
45. **return word**
46. suggested\_words = self.suggest\_words(word)
47. num\_modified\_characters = []
48. **if** suggested\_words != 0:
49. **for** sug\_words **in** suggested\_words:
50. **num\_modified\_characters.append(self.levenshtein\_distance(word,sug\_words))**
51. minimum\_edit\_distance = min(num\_modified\_characters)
52. best\_arg = num\_modified\_characters.index(minimum\_edit\_distance)
53. **if** max\_distance > minimum\_edit\_distance:
54. best\_suggestion = suggested\_words[best\_arg]
55. **return best\_suggestion**
56. **else**:
57. **return** word
58. **else**:
59. **return** word

62. **def** ngram\_spell\_corrector(self,word):
63. max\_distance = 2
64. **if** (self.dictionary.check(word)):
65. **return word**
66. suggested\_words = self.suggest\_words(word)
67. num\_modified\_characters = []
68. max\_jaccard = []
69. list\_of\_sug\_words = []
70. **if suggested\_words != 0:**
71. word\_ngrams = self.ngram(word,2)
72. **for** sug\_words **in** suggested\_words:
73. **if** (self.levenshtein\_distance(word,sug\_words)) < 3 :
74. sug\_ngrams = self.ngram(sug\_words,2)
75. **jac = self.jaccard(word\_ngrams,sug\_ngrams)**
76. max\_jaccard.append(jac)
77. list\_of\_sug\_words.append(sug\_words)
78. highest\_jaccard = max(max\_jaccard)
79. best\_arg = max\_jaccard.index(highest\_jaccard)
80. **word = list\_of\_sug\_words[best\_arg]**
81. **return** word
82. **else**:
83. **return** word

**ПРИЛОЖЕНИЕ Г**

**Код скрипта предобработки данных для голосовой идентификации**

1. import collections
2. import contextlib
3. import librosa
4. import numpy as np
5. **import python\_speech\_features**
6. import wave

9. def extract\_features(y, sr, n\_mfcc):
10. **mfcc = python\_speech\_features.mfcc(signal=y, samplerate=sr, numcep=n\_mfcc)**
11. mfcc = mfcc.T
12. mfcc\_delta = librosa.feature.delta(mfcc)
13. mfcc\_delta2 = librosa.feature.delta(mfcc, order=2)
14. stacked = np.vstack((mfcc, mfcc\_delta, mfcc\_delta2))
15. **return stacked.T**

18. def normalize(feature, eps=1e-14):
19. feats\_mean = np.mean(feature, axis=0)
20. **feats\_std = np.std(feature, axis=0)**
21. return (feature - feats\_mean) / (feats\_std + eps)


25. **def write\_wave(path, audio, sample\_rate):**
26. with contextlib.closing(wave.open(path, 'wb')) as wf:
27. wf.setnchannels(1)
28. wf.setsampwidth(2)
29. wf.setframerate(sample\_rate)
30. **wf.writeframes(audio)**

33. class Frame(object):
34. def \_\_init\_\_(self, bytes, timestamp, duration):
35. **self.bytes = bytes**
36. self.timestamp = timestamp
37. self.duration = duration

40. **def frame\_generator(frame\_duration\_ms, audio, sample\_rate):**
41. n = int(sample\_rate \* (frame\_duration\_ms / 1000.0) \* 2)
42. offset = 0
43. timestamp = 0.0
44. duration = (float(n) / sample\_rate) / 2.0
45. **while offset + n < len(audio):**
46. yield Frame(audio[offset:offset + n], timestamp, duration)
47. timestamp += duration
48. offset += n

51. def vad\_collector(sample\_rate, frame\_duration\_ms, padding\_duration\_ms, vad, frames):
52. num\_padding\_frames = int(padding\_duration\_ms / frame\_duration\_ms)
53. ring\_buffer = collections.deque(maxlen=num\_padding\_frames)
54. triggered = False
56. voiced\_frames = []
57. for frame in frames:
58. is\_speech = vad.is\_speech(frame.bytes, sample\_rate)
60. **if not triggered:**
61. ring\_buffer.append((frame, is\_speech))
62. num\_voiced = len([f for f, speech in ring\_buffer if speech])
63. if num\_voiced > 0.9 \* ring\_buffer.maxlen:
64. triggered = True
65. **for f, s in ring\_buffer:**
66. voiced\_frames.append(f)
67. ring\_buffer.clear()
68. else:
69. voiced\_frames.append(frame)
70. **ring\_buffer.append((frame, is\_speech))**
71. num\_unvoiced = len([f for f, speech in ring\_buffer if not speech])
72. if num\_unvoiced > 0.9 \* ring\_buffer.maxlen:
73. triggered = False
74. yield b''.join([f.bytes for f in voiced\_frames])
75. **ring\_buffer.clear()**
76. voiced\_frames = []
77. if voiced\_frames:
78. yield b''.join([f.bytes for f in voiced\_frames])

**ПРИЛОЖЕНИЕ Д**

**Код скрипта создания модели UBM, GMM**

1. import numpy as np

4. def map\_adaptation(gmm, data, max\_iterations=300, likelihood\_threshold=1e-20, relevance\_factor=16):
5. **N = data.shape[0]**
6. D = data.shape[1]
7. K = gmm.n\_components
9. mu\_new = np.zeros((K, D))
10. **n\_k = np.zeros((K, 1))**
12. mu\_k = gmm.means\_
13. cov\_k = gmm.covariances\_
14. pi\_k = gmm.weights\_
16. old\_likelihood = gmm.score(data)
17. new\_likelihood = 0
18. iterations = 0
19. while abs(old\_likelihood - new\_likelihood) > likelihood\_threshold and iterations < max\_iterations:
20. **iterations += 1**
21. old\_likelihood = new\_likelihood
22. z\_n\_k = gmm.predict\_proba(data)
23. n\_k = np.sum(z\_n\_k, axis=0)
24. for i in range(K):
25. **temp = np.zeros((1, D))**
26. for n in range(N):
27. temp += z\_n\_k[n][i] \* data[n, :]
28. mu\_new[i] = (1 / max(n\_k[i], 1e-20)) \* temp
30. **adaptation\_coefficient = n\_k / (n\_k + relevance\_factor)**
31. for k in range(K):
32. mu\_k[k] = (adaptation\_coefficient[k] \* mu\_new[k]) + ((1 - adaptation\_coefficient[k]) \* mu\_k[k])
33. gmm.means\_ = mu\_k
35. **log\_likelihood = gmm.score(data)**
36. new\_likelihood = log\_likelihood
37. return gmm

**ПРИЛОЖЕНИЕ Е**

**Код файла main.py**

1. **from** aiohttp **import** web
2. **import** argparse
3. **import** asyncio
4. **import** os
5. **import signal**
6. **import** traceback
8. **from** backends.backends **import** setup\_backends
9. **from** logger **import** setup\_logger
10. **from routes import setup\_routes**
11. **from** settings **import** CONFIG\_PATH, get\_config, get\_secure\_config, test\_config

14. **def** setup\_config(app, config\_path=None):
15. **try:**
16. **if** config\_path **is** None:
17. app['config'] = get\_config(CONFIG\_PATH)
18. test\_config(app['config'])
19. **else**:
20. **app['config'] = get\_config(config\_path)**
21. test\_config(app['config'])
22. **except** Exception **as** e:
23. app.logger.exception("config is broken.")
24. **return** 1
25. **setup\_logger(app)**
26. secure\_config = get\_secure\_config(app['config'])
27. app.logger.info(f"config: {secure\_config}")
28. **return** 0

31. **def** setup\_app(config\_path=None):
32. *#app\_middlewares = middlewares()*
33. app = web.Application()
34. status = setup\_config(app, config\_path)
35. **if status == 1:**
36. app.logger.error("Cant start app")
37. exit(1)
38. setup\_backends(app)
39. setup\_routes(app)
41. **return** app

44. @asyncio.coroutine
45. **async def reload\_config(app, config\_path=None):**
46. app.logger.info(f"handling SIGHUP - reload\_config and logger")
47. secure\_config = get\_secure\_config(app['config'])
48. status = setup\_config(app, config\_path)
49. **if** status == 1:
50. **app['config'] = secure\_config**
51. app.logger.info(f"config was not reloaded")
52. **else**:
53. app.logger.info(f"config was reloaded")

56. **if** \_\_name\_\_ == "\_\_main\_\_":
57. PARSER = argparse.ArgumentParser()
58. PARSER.add\_argument('--path')
59. PARSER.add\_argument('--port')
60. **PARSER.add\_argument('--config')**
61. ARGS = PARSER.parse\_args()
63. OLD\_UMASK = os.umask(0o022)
64. os.umask(0)
66. **try**:
67. loop = asyncio.get\_event\_loop()
68. app = setup\_app(config\_path=ARGS.config)
69. loop.add\_signal\_handler(signal.SIGHUP, **lambda**: asyncio.create\_task(
70. **reload\_config(app=app, config\_path=ARGS.config)))**
71. web.run\_app(app=app,
72. path=ARGS.path,
73. port=ARGS.port)
74. **except**:
75. **print(traceback.format\_exc())**
76. **finally**:
77. os.umask(OLD\_UMASK)

**ПРИЛОЖЕНИЕ Ж**

**Код файла routes.py**

1. **from** aiohttp **import** web
2. **from** functions.handler **import** create\_user, add\_voice, add\_password, check, info, delete\_user, info\_html

5. **def setup\_routes(app):**
6. app.router.add\_route('POST',
7. '/create\_user',
8. create\_user,
9. name='create\_user')
11. app.router.add\_route('POST',
12. r'/delete\_user/{id:**\d**+}',
13. delete\_user,
14. name='delete\_user')
16. app.router.add\_route('POST',
17. r'/add\_voice/{id:**\d**+}',
18. add\_voice,
19. name='add\_voice')
21. app.router.add\_route('POST',
22. r'/add\_password/{id:**\d**+}',
23. add\_password,
24. name='add\_password')
26. app.router.add\_route('POST',
27. r'/check/{id:**\d**+}',
28. check,
29. name='check')
31. app.router.add\_route('GET',
32. '/info',
33. info,
34. name='info')
36. app.router.add\_route('GET',
37. '/info\_html',
38. info\_html,
39. name='info\_html')

**ПРИЛОЖЕНИЕ И**

**Код файла handler.py**

1. **from** aiohttp **import** web
2. **import** json
3. **import** os
4. **from** .speech\_recognition **import** predict
5. **from .voice\_identification import create\_gmm, predict\_voice**

8. async **def** recognize\_voice(app, id\_, file):
9. **print**('recognize\_voice - done')
11. **if** **not** os.path.exists(app['config']['inventory']['dir'] + '/' + str(id\_) +
12. '/' + app['config']['inventory']['voiceDirName']):
13. **return** 2
14. features\_path = app['config']['inventory']['dir'] + '/' + str(id\_) + \
15. **'/' + app['config']['inventory']['voiceDirName'] + '/features'**
16. **if** **not** os.path.exists(features\_path):
17. os.makedirs(features\_path)
18. id2 = predict\_voice(app, file, features\_path)
19. await delete\_path(features\_path)
20. **if id\_ == id2:**
21. **return** 0
22. **return** 1

25. **async def recognize\_speech(app, id\_, file):**
26. **print**('recognize\_speech - done')
27. **if** **not** os.path.exists(app['config']['inventory']['dir'] + '/' + str(id\_) +
28. '/' + app['config']['inventory']['pswdDirName']):
29. **return** 2
30. **answer1 = answer2 = ''**
31. answer1 = predict(app, file)
32. **with** open(app['config']['inventory']['dir'] + '/' + str(id\_) +
33. '/' + app['config']['inventory']['pswdDirName'] + '/1.txt', "r") **as** f:
34. answer2 = f.read()
35. **print('recognize\_speech -', answer1)**
36. **if** answer1 == answer2:
37. **return** 0
38. **return** 1

41. async **def** train\_GMM(app, id\_):
42. create\_gmm(app, app['config']['voiceSystem']['dir'], id\_)
43. **print**('train\_GMM - done')
44. **return**

47. async **def** download\_file(request, path):
48. **try**:
49. **with** open(path, 'wb') **as** fd:
50. **while True:**
51. chunk = await request.content.read(10)
52. **if** **not** chunk:
53. **break**
54. fd.write(chunk)
55. **except Exception as e:**
56. **print**('download\_file - ', e)
57. **return** False
58. **return** True

61. async **def** get\_static(file):
62. file\_path = os.path.join(os.getcwd(), 'static', file)
63. **if** os.path.isfile(file\_path):
64. **with** open(file\_path, 'rb') **as** static\_file:
65. **static\_content = static\_file.read()**
66. **return** web.Response(body=static\_content, status=200, content\_type='text/html')
67. **else**:
68. **raise** web.HTTPNotFound()

71. async **def** create\_user(request):
72. ids = [int(i) **for** i **in** os.listdir(request.app['config']['inventory']['dir'])]
73. id = (max(ids) **if** len(ids) > 0 **else** 0) + 1
74. **if** **not** os.path.exists(request.app['config']['inventory']['dir'] + '/' + str(id)):
75. **os.makedirs(request.app['config']['inventory']['dir'] + '/' + str(id))**
76. **else**:
77. **return** web.Response(status=400,
78. text=json.dumps({'text': 'cant create user {0}'.format(id)}),
79. content\_type="application/json")
80. **return web.Response(status=200,**
81. text=json.dumps({'text': 'ok', 'id': id}),
82. content\_type="application/json")

85. **async def add\_voice(request):**
86. id = request.match\_info['id']
87. **if** **not** os.path.exists(request.app['config']['inventory']['dir'] + '/' + str(id)):
88. **return** web.Response(status=400,
89. text=json.dumps({'text': 'user do not exist'}),
90. **content\_type="application/json")**
91. filename = request.app['config']['inputData']['dinamicDir'] + '/{0}.wav'.format(id)
92. **if** await download\_file(request, filename):
93. old\_filename = request.app['config']['inventory']['dir'] + '/' + str(id) + \
94. '/' + request.app['config']['inventory']['voiceDirName']
95. **await delete\_path(old\_filename)**
96. os.mkdir(old\_filename)
97. os.replace(filename, old\_filename + '/1.wav')
98. **else**:
99. **return** web.Response(status=400,
100. **text=json.dumps({'text': 'cant download'}),**
101. content\_type="application/json")
103. await train\_GMM(request.app, id) *# train GMM*
104. **return** web.Response(status=200,
105. **text=json.dumps({'text': 'ok'}),**
106. content\_type="application/json")

109. async **def** get\_data\_from\_request(request):
110. **try:**
111. data = await request.text()
112. data = json.loads(data)
113. **except** Exception:
114. data = await request.post()
115. **if data.get('password') is None:**
116. data = data.get('content')
117. **else**:
118. data = data.get('password')
119. **return** data

122. async **def** add\_password(request):
123. id = request.match\_info['id']
124. **if** **not** os.path.exists(request.app['config']['inventory']['dir'] + '/' + str(id)):
125. **return web.Response(status=400,**
126. text=json.dumps({'text': 'user do not exist'}),
127. content\_type="application/json")
128. *# data = request.query.get('password')*
129. data = await get\_data\_from\_request(request)
130. **print('add\_password - ', data)**
131. **if** data **and** len(data) > 0:
132. old\_filename = request.app['config']['inventory']['dir'] + '/' + str(id) + \
133. '/' + request.app['config']['inventory']['pswdDirName']
134. await delete\_path(old\_filename)
135. **os.mkdir(old\_filename)**
136. **with** open(old\_filename + '/1.txt', 'w+') **as** f:
137. f.write(data)
138. **else**:
139. **return** web.Response(status=400,
140. **text=json.dumps({'text': 'wrong params'}),**
141. content\_type="application/json")
142. **return** web.Response(status=200,
143. text=json.dumps({'text': 'ok'}),
144. content\_type="application/json")

147. async **def** check(request):
148. id = request.match\_info['id']
149. **if** **not** os.path.exists(request.app['config']['inventory']['dir'] + '/' + str(id)):
150. **return web.Response(status=401, text=json.dumps({'text': 'not allowed'}), content\_type="application/json")**
152. filename = request.app['config']['inputData']['dinamicDir'] + '/check\_{0}.wav'.format(id)
153. **if** await download\_file(request, filename):
154. flag1 = await recognize\_voice(request.app, id, filename)
155. **flag2 = await recognize\_speech(request.app, id, filename)**
157. await delete\_path(filename)
158. **if** flag1 == 0 **or** flag1 == 2 **and** flag2 == 0 **or** flag2 == 2:
159. text = {'text': 'ok'}
160. **if flag1 == 2:**
161. text['voice'] = 'not done'
162. **if** flag2 == 2:
163. text['speech'] = 'not done'
164. **return** web.Response(text=json.dumps(text), content\_type="application/json")
165. **else:**
166. text = {'text': 'not allowed'}
167. **if** request.app['config']['app']['test']:
168. **if** flag1 == 1:
169. text['voice'] = 'not allowed'
170. **if flag2 == 1:**
171. text['speech'] = 'not allowed'
172. **return** web.Response(status=401, text=json.dumps(text), content\_type="application/json")
173. **return** web.Response(status=400, text=json.dumps({'text': 'cant download file'}), content\_type="application/json")

176. async **def** delete\_path(path):
177. **try**:
178. **if** os.path.exists(path):
179. **for** root, dirs, files **in** os.walk(path, topdown=False):
180. **for name in files:**
181. os.remove(os.path.join(root, name))
182. **for** name **in** dirs:
183. os.rmdir(os.path.join(root, name))
184. os.rmdir(path)
185. **return True**
186. **except** Exception **as** e:
187. *# print('delete\_path 1 - ', e)*
188. **try**:
189. os.remove(path)
190. **return True**
191. **except** Exception **as** er:
192. **print**('delete\_path 2 - ', e)
193. **return** False

196. async **def** delete\_user(request):
197. id = request.match\_info['id']
198. path = request.app['config']['inventory']['dir'] + '/' + str(id)
199. **if** **not** await delete\_path(path):
200. **return web.Response(status=400,**
201. text=json.dumps({'text': 'user does not exist'}),
202. content\_type="application/json")
203. **else**:
204. **print**('user {0} deleted'.format(id))
205. **return web.Response(status=200,**
206. text=json.dumps({'text': 'ok'}),
207. content\_type="application/json")

210. **async def get\_users(app):**
211. users = os.listdir(app['config']['inventory']['dir'])
212. answer = []
213. **for** i **in** users:
214. voice = pswd = 'no'
215. **factors = os.listdir(app['config']['inventory']['dir'] + '/' + str(i))**
216. **if** app['config']['inventory']['voiceDirName'] **in** factors:
217. voice = 'yes'
218. **if** app['config']['inventory']['pswdDirName'] **in** factors:
219. pswd = 'yes'
220. **answer.append((i, voice, pswd, 'yes'))**
221. **return** answer

224. async **def** info(request):
225. **users = await get\_users(request.app)**
226. data = []
227. **for** (id\_, voice, pswd, active) **in** users:
228. data.append({'id': id\_,
229. 'voice\_exist': voice,
230. **'password\_exist': pswd,**
231. 'active': active})
232. **return** web.Response(status=200,
233. text=json.dumps(data, ensure\_ascii=False),
234. content\_type="application/json")

237. async **def** info\_html(request):
238. users = await get\_users(request.app)
239. data = ''
240. **users = ['<tr><td>{0}</td><td>{1}</td><td>{2}</td><td>{3}</td></tr>'.format(id\_, voice, pswd, active) for**
241. (id\_, voice, pswd, active) **in** users]
242. **for** i **in** users:
243. data += i
244. **print**(data)
245. **content = '''<!DOCTYPE HTML><html><head><meta charset="utf-8"><title>Пользователи в базе</title></head><body><table border="1">**
246. <caption>Пользователи в базе</caption><tr><th>Идентификатор</th><th>Образец голоса</th><th>Наличие пароля</th><th>Активен</th>
247. </tr>''' + data + '''</table></body></html>'''
248. **return** web.Response(body=content, status=200, content\_type='text/html')

**ПРИЛОЖЕНИЕ К**

**Код файла test.py**

1. **import** json
2. **import** requests
3. **import** unittest
4. **import** random
5. **import time**
7. data\_path = 'C:/Users/super/PycharmProjects/auth\_api/data/test\_data/1.wav'

10. **class TestAPI(unittest.TestCase):**
12. **def** create\_user(self):
13. r = requests.post('http://localhost:8080/create\_user')
14. self.assertEqual(r.status\_code, 200)
15. **answer = json.loads(r.text)**
16. self.assertEqual(answer['text'], 'ok')
17. **return** answer['id']
19. **def** create\_users(self):
20. **for id\_ in range(100):**
21. r = requests.post('http://localhost:8080/delete\_user/{0}'.format(id\_))
22. users\_list = []
23. **for** i **in** range(100):
24. users\_list.append(self.create\_user())
25. **for i in range(500):**
26. id\_ = random.randint(max(users\_list) + 1, max(users\_list) + 501)
27. **if** id\_ **in** users\_list:
28. **print**(id\_)
29. **continue**
30. **r = requests.post('http://localhost:8080/delete\_user/{0}'.format(id\_))**
31. self.assertEqual(r.status\_code, 400)
32. answer = json.loads(r.text)
33. self.assertEqual(answer['text'], 'user does not exist')
35. **for id\_ in users\_list:**
36. r = requests.post('http://localhost:8080/delete\_user/{0}'.format(id\_))
37. self.assertEqual(r.status\_code, 200)
38. answer = json.loads(r.text)
39. self.assertEqual(answer['text'], 'ok')
41. **for** id\_ **in** range(max(users\_list) + 1):
42. r = requests.post('http://localhost:8080/delete\_user/{0}'.format(id\_))
43. self.assertEqual(r.status\_code, 400)
44. answer = json.loads(r.text)
45. **self.assertEqual(answer['text'], 'user does not exist')**
47. **def** test\_create\_users(self):
48. self.create\_users()
50. **def test\_add\_delete\_data(self):**
51. users\_list = set()
52. **for** j **in** range(3):
53. id\_ = self.create\_user()
54. users\_list.add(id\_)
55. **for \_ in range(3):**
56. self.add\_voice(id\_)
57. r = requests.post('http://localhost:8080/delete\_user/{0}'.format(id\_))
58. self.assertEqual(r.status\_code, 200)
59. answer = json.loads(r.text)
60. **self.assertEqual(answer['text'], 'ok')**
61. **for** id\_ **in** users\_list:
62. r = requests.post('http://localhost:8080/delete\_user/{0}'.format(id\_))
63. self.assertEqual(r.status\_code, 400)
64. answer = json.loads(r.text)
65. **self.assertEqual(answer['text'], 'user does not exist')**
67. **def** test\_add\_data(self):
68. *# user one*
69. id\_ = self.create\_user()
70. **self.add\_voice(id\_)**
71. r = requests.post('http://localhost:8080/check/{0}'.format(id\_))
72. self.assertEqual(r.status\_code, 200)
73. answer = json.loads(r.text)
74. self.assertEqual(answer['text'], 'ok') *# {'text': 'not allowed'} ['speech'] = 'not done'*
75. **self.assertEqual(answer['speech'], 'not done')**
76. *#self.assertEqual(answer.get('speech'), None)*
77. self.assertEqual(answer.get('voice'), None)
79. self.add\_speech(id\_)
80. **r = requests.post('http://localhost:8080/check/{0}'.format(id\_))**
81. answer = json.loads(r.text)
82. **print**(answer)
83. self.assertEqual(r.status\_code, 200)
84. self.assertEqual(answer['text'], 'ok')
85. **self.assertEqual(answer.get('speech'), None)**
86. self.assertEqual(answer.get('voice'), None)
88. r = requests.post('http://localhost:8080/delete\_user/{0}'.format(id\_))
89. self.assertEqual(r.status\_code, 200)
90. **answer = json.loads(r.text)**
91. self.assertEqual(answer['text'], 'ok')
93. *# user two*
94. id\_ = self.create\_user()
95. **self.add\_speech(id\_)**
96. r = requests.post('http://localhost:8080/check/{0}'.format(id\_))
97. self.assertEqual(r.status\_code, 200)
98. answer = json.loads(r.text)
99. self.assertEqual(answer['text'], 'ok')
100. ***#self.assertEqual(answer['voice'], 'not done')***
101. self.assertEqual(answer.get('speech'), None)
102. self.assertEqual(answer.get('voice'), 'not done')
104. self.add\_voice(id\_)
105. **r = requests.post('http://localhost:8080/check/{0}'.format(id\_))**
106. self.assertEqual(r.status\_code, 200)
107. answer = json.loads(r.text)
108. self.assertEqual(answer['text'], 'ok')
109. self.assertEqual(answer.get('speech'), None)
110. **self.assertEqual(answer.get('voice'), None)**
112. r = requests.post('http://localhost:8080/delete\_user/{0}'.format(id\_))
113. self.assertEqual(r.status\_code, 200)
114. answer = json.loads(r.text)
115. **self.assertEqual(answer['text'], 'ok')**
117. *# user three*
118. id\_ = self.create\_user()
119. r = requests.post('http://localhost:8080/check/{0}'.format(id\_))
120. **self.assertEqual(r.status\_code, 200)**
121. answer = json.loads(r.text)
122. self.assertEqual(answer['text'], 'ok') *# {'text': 'not allowed'} ['speech'] = 'not done'*
123. self.assertEqual(answer['voice'], 'not done')
124. self.assertEqual(answer['speech'], 'not done')
126. self.add\_speech(id\_)
127. self.add\_voice(id\_)
128. r = requests.post('http://localhost:8080/check/{0}'.format(id\_))
129. self.assertEqual(r.status\_code, 200)
130. **answer = json.loads(r.text)**
131. self.assertEqual(answer['text'], 'ok')
132. self.assertEqual(answer.get('speech'), None)
133. self.assertEqual(answer.get('voice'), None)
135. **r = requests.post('http://localhost:8080/delete\_user/{0}'.format(id\_))**
136. self.assertEqual(r.status\_code, 200)
137. answer = json.loads(r.text)
138. self.assertEqual(answer['text'], 'ok')
140. **def test\_info\_valid(self):**
141. users\_list = set()
142. **for** i **in** range(100):
143. users\_list.add(self.create\_user())
144. users\_list\_voice = random.sample(users\_list, 20)
145. **users\_list\_speech = random.sample(users\_list, 20)**
146. **for** i **in** users\_list\_voice:
147. self.add\_voice(i)
148. **for** i **in** users\_list\_speech:
149. self.add\_speech(i)
151. users\_list\_voice = set(users\_list\_voice)
152. users\_list\_speech = set(users\_list\_speech)
154. r = requests.get('http://localhost:8080/info')
155. **self.assertEqual(r.status\_code, 200)**
156. answer = json.loads(r.text)
157. **for** one\_answer **in** answer:
158. id\_ = int(one\_answer['id'])
159. voice = one\_answer['voice\_exist']
160. **pswd = one\_answer['password\_exist']**
161. active = one\_answer['active']
163. self.assertTrue(id\_ **in** users\_list)
164. self.assertEqual(active, 'yes')
165. **if id\_ in users\_list\_speech:**
166. self.assertEqual(pswd, 'yes')
167. **else**:
168. self.assertEqual(pswd, 'no')
169. **if** id\_ **in** users\_list\_voice:
170. **self.assertEqual(voice, 'yes')**
171. **else**:
172. self.assertEqual(voice, 'no')
174. **for** id\_ **in** users\_list:
175. **r = requests.post('http://localhost:8080/delete\_user/{0}'.format(id\_))**
176. self.assertEqual(r.status\_code, 200)
177. answer = json.loads(r.text)
178. self.assertEqual(answer['text'], 'ok')
180. **def add\_voice(self, id\_):**
182. **with** open(data\_path, 'rb') **as** f:
183. data = f
184. headers = {'content-type': 'audio/wav'}
185. **start = time.time()**
186. r = requests.post('http://localhost:8080/add\_voice/{0}'.format(id\_), data=data, headers=headers)
187. **print**(time.time() - start)
188. self.assertEqual(r.status\_code, 200)
189. answer = json.loads(r.text)
190. **self.assertEqual(answer['text'], 'ok')**
192. **def** add\_speech(self, id\_):
193. pswd = 'key phrase for auth api. user number is {0}'.format(id\_)
194. r = requests.post('http://localhost:8080/add\_password/{0}'.format(id\_),
195. **data=json.dumps({'password': pswd}))**
196. self.assertEqual(r.status\_code, 200)
197. answer = json.loads(r.text)
198. self.assertEqual(answer['text'], 'ok')

201. **if** \_\_name\_\_ == '\_\_main\_\_':
202. unittest.main()