



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ_____

КАФЕДРА _____СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ (ИУ5)_____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Распознавания голосовых команд

Студент ИУ5-61Б
(Группа)

(Подпись, дата) **Рысьева Е. А.**
(И.О.Фамилия)

Руководитель

(Подпись, дата) **Канев А. И.**
(И.О.Фамилия)

Москва, 2023 г.



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени
Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ5
(Индекс)

« _____ » _____ 20 ____ г.
(И.О.Фамилия)

ЗАДАНИЕ
на выполнение научно-исследовательской работы

по теме Распознавание голосовых команд

Студент группы ИУ5-61Б

Рысьева Елизавета Антоновна
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

Исследовательская

Источник тематики (кафедра, предприятие, НИР) НИР

График выполнения НИР: 25% к ____ нед., 50% к ____ нед., 75% к ____ нед., 100% к ____ нед.

Техническое задание: Исследовать использование методов машинного обучения для решения задачи распознавания простых голосовых команд

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на 18 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

Дата выдачи задания «07» февраля 2023 г.

Руководитель НИР

(Подпись, дата)

Канев А.И.

(И.О.Фамилия)

Студент

(Подпись, дата)

Рысьева Е. А.

(И.О.Фамилия)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1 Постановка задачи	5
2 Описание данных, используемых методов и метрик	8
3 Выполнение примера по варианту	11
ЗАКЛЮЧЕНИЕ	23
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	24

ВВЕДЕНИЕ

Распознавание человеческой речи является одной из наиболее актуальных задач в современном мире. Ежедневно миллиарды людей по всему миру используют распознавание речи для голосового поиска в Интернете, общения с голосовыми ассистентами, управления умным домом и валидации пользователей для доступа к их личной информации, а также для облегчения жизни людей с ограниченными возможностями. С каждым годом новые подходы к решению этой задачи все более точны и приближаются к человеческому уровню. Лучшим результатом в данной области на данный момент является работа

Нам предстоит использовать набор данных Speech Commands Dataset для создания алгоритма, который понимает простые голосовые команды. Повышая точность распознавания инструментов голосового интерфейса с открытым исходным кодом, мы можем повысить эффективность продуктов и их доступность.

Цель данной работы заключается в создании алгоритма для распознавания речи. Задачами работы являются изучение современных подходов к решению проблемы, использование различных инструментов и построение оригинальной архитектуры, объединяющей преимущества существующих исследований. Основная цель - достижение желаемого результата, учитывая ограниченность вычислительных мощностей. Это является нетипичной задачей, так как распознавание речи является самой по себе вычислительно трудной проблемой.

1. Постановка задачи

Распознавание речи является одной из наиболее актуальных задач в области искусственного интеллекта и обработки естественного языка. С момента появления первых систем распознавания речи в 1950-х годах, технологии в этой области продолжают развиваться и улучшаться.

Одной из основных причин актуальности задачи распознавания речи является ее потенциальная применимость в различных сферах жизни. Например, в медицине распознавание речи может использоваться для диагностики заболеваний, а в образовании - для создания систем обучения, которые могут оценивать произношение студентов.

Кроме того, распознавание речи может иметь большое значение для людей с ограниченными возможностями, таких как глухие или глухонемые. Технологии распознавания речи могут помочь им в общении и повышении качества жизни.

Наконец, распознавание речи может быть полезно в бизнесе и маркетинге. Например, системы распознавания речи могут использоваться для анализа звонков в call-центрах и определения эффективности работы операторов, а также для анализа отзывов клиентов и определения их настроения.

Задача распознавания речи является важной и актуальной, и ее решение может принести множество практических польз и улучшить качество жизни людей в различных сферах.

За последние несколько лет голосовые помощники стали повсеместными благодаря популярности Google Home, Amazon Echo, Siri, Cortana и других. Это самые известные примеры автоматического распознавания речи (ASR). Этот класс приложений начинается с фрагмента произнесенной аудиозаписи на каком-либо языке и извлекает произнесенные слова в виде текста. По этой причине они также известны как алгоритмы преобразования речи в текст.

В русском сегменте лидирует компания Яндекс с новой разработкой Yandex.SpeechKit которая представляет собой мультиплатформенную

библиотеку, предоставляющая разработчикам мобильных приложений доступ к технологии распознавания речи Яндекс [1]. Тем не менее, проблема речевого ввода информации осложняется рядом факторов: различием структуры языков, спецификой произношения, шумами и помехами, акцентами, ударениями и т.п.

Существующие на сегодняшний день системы распознавания речи основываются на сборе всей доступной и даже избыточной информации, необходимой для распознавания лексических элементов. Системы подобные распознавания речи крупных компаний, таких как Google Inc. Используют широкую базу сэмплов речевых паттернов сотен и даже тысяч дикторов, что позволяет им добиваться уверенного распознавания многих слов неизвестных дикторов. Некоторые исследователи считают [2,3], что таким образом задача распознавания образца речи, основанная на качестве сигнала, подверженного изменениям, будет достаточной для распознавания, однако в настоящее время даже при распознавании небольших сообщений нормальной речи, пока невозможно после получения разнообразных реальных сигналов осуществить прямую трансформацию в лингвистические символы, что является желаемым результатом.

Для распознавания речи акустический сигнал при помощи детектирующих и оцифровывающих устройств и машинной обработки фиксируется и преобразуется в цифровую форму [5]. В результате дискретизации непрерывный (аналоговый) сигнал переводится в последовательность чисел. Наиболее популярные методы цифровой обработки речевых сигналов: частотный анализ в базисе Фурье, вейвлет анализ, кепстральный анализ, субполосный анализ.

В последнее время набирают популярность системы распознавания речи, использующие неакустические параметры такие как: движения губ, языка, мышц лица (фиксируемые камерой), ультразвук, колебания в костях черепа, а также электромиографию фиксирующую активность голосовых связок и гортани [2,3,4]. Основной причиной появления таких методов является желание повысить количество информации, пригодной для распознавания. Подобного

рода ларингофоны впервые использовались в системах связи танковых войск, где уровень акустических шумов внутри машин был довольно высок и требовались дополнительные источники информации, кроме непосредственной фиксации звуковых колебаний. Системы распознавания речи, использующие такие подходы позволяют добиваться более высокой точности, а также снимать многие ограничения, накладываемые на акустический тракт в виду воздействия на него помех и шумов.

2. Описание данных, используемых методов и метрик

Набор звуковых данных произносимых слов, предназначенный для обучения и оценки систем определения ключевых слов. Его основная цель — предоставить способ создания и тестирования небольших моделей, которые определяют, когда произносится одно слово из набора из десяти целевых слов, с минимальным количеством ложных срабатываний из-за фонового шума или несвязанной речи.

Для обучения модели, которая будет распознавать простые голосовые команды, необходимо решить следующие задачи:

1. Сбор и подготовка данных: необходимо собрать достаточное количество аудиозаписей голосовых команд для обучения модели. Данные должны быть разнообразными и представлять различные голосовые характеристики, такие как пол, возраст, акцент и т.д. После сбора данных необходимо провести их предварительную обработку, включающую в себя нормализацию амплитуды, фильтрацию шума и т.д. Многие наборы данных распознавания голоса требуют предварительной обработки, прежде чем на них можно будет построить модель. Чтобы помочь в этом, TensorFlow недавно выпустил наборы данных Speech Commands. Он включает в себя 65 000 односекундных произнесений 30 коротких слов тысячами разных людей

2. Выбор архитектуры модели: для распознавания речи можно использовать различные архитектуры:

- 2.1. Сверточные нейронные сети (CNN)
- 2.2. преобразование Фурье (STFT)
- 2.3. спектрограмма Мела (MFCC)

3. Обучение модели: после выбора архитектуры модели необходимо провести ее обучение на подготовленных данных

4. Оценка качества модели: после обучения модели необходимо оценить ее качество на тестовых данных. Для этого используются:

- 4.1. Метрики:
 - 4.1.1. Метрика accuracy

Метрика accuracy в распознавании речи показывает долю правильно распознанных голосовых команд от общего количества распознанных команд. Эта метрика является одной из основных метрик для оценки качества модели в задачах классификации, включая распознавание речи

4.1.2. Метрика качества классификации F1 – мера.

F1-мера является метрикой качества классификации, которая учитывает как точность (precision), так и полноту (recall) модели

4.1.3. Метрика precision

Метрика Precision в задаче распознавания речи используется для оценки точности модели в предсказании правильного класса

4.1.4. График потерь

График потерь на обучающем и валидационном наборе данных в задаче распознавания речи используется для оценки качества модели и определения того, как изменяется потеря (ошибка) модели в процессе обучения. График позволяет определить, насколько хорошо модель обучается и как быстро она сходится к оптимальным значениям параметров

4.1.5. Матрица путаницы

Матрица путаницы используется для оценки производительности модели машинного обучения. Она представляет собой таблицу, которая показывает, какие объекты были верно и неверно классифицированы моделью

4.2. Методы:

4.2.1. Сверточные нейронные сети:

Сверточные нейронные сети используются для извлечения признаков из аудио-сигналов, которые затем могут быть использованы для идентификации. Эти нейронные сети могут обрабатывать данные в режиме реального времени. Кроме того, сверточные нейронные сети могут обучаться на больших объемах данных, что позволяет им достичь высокой точности в распознавании речи

4.3. Языки программирования:

4.3.1. Python

Для реализации работы был выбран язык программирования Python, так как данный язык имеет большое количество сторонних библиотек для разработки нейронных сетей и анализа данных. Он является самым передовым языком в области машинного обучения, при этом предоставляет простой синтаксис для работы

4.4. Библиотеки:

4.4.1. Pandas

Pandas – программная библиотека на языке Python для обработки и анализа данных. С ее помощью производилась обработка данных о звуковых файлах и производился анализ информации о наборе данных

4.4.2. NumPy

NumPy – библиотека с открытым исходным кодом для языка программирования Python

Данная библиотека применялась для предобработки звуковых файлов, в частности для аугментации, а также был использован контейнер `pru`, который позволил загружать и обрабатывать данные в несколько раз быстрее, чем стандартные `cvs` и `mp3`, за счет чего было ускорено обучение модели

4.4.3. LibROSA

LibROSA – библиотека для языка программирования Python, предназначенная для обработки и анализа аудиозаписей. Главным отличием является то, что есть возможность работать с форматом `mp3`, что было необходимо для данной работы

4.4.4. TensorFlow

TensorFlow – открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия.

3. Выполнение работы

3.1. Разведочный анализ

Импортируем необходимые модули и зависимости. (рисунок 3.1), (рисунок 3.2).

```
[ ] import os
    for dirname, _, filenames in os.walk('/kaggle/input'):
        for filename in filenames:
            print(os.path.join(dirname, filename))

[ ] import pathlib

    import tensorflow as tf

    from os.path import isdir, join
    from pathlib import Path
    import pandas as pd

    import numpy as np
    from scipy.fftpack import fft
    from scipy import signal
    from scipy.io import wavfile
    import librosa
    import librosa.display

    from sklearn.decomposition import PCA

    import matplotlib.pyplot as plt
    import seaborn as sns
    import IPython.display as ipd
    import librosa.display
```

Рисунок 3.1 – импорт библиотек

```
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import pandas as pd

%matplotlib inline

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

    # Set the seed value for experiment reproducibility.
    seed = 42
    tf.random.set_seed(seed)
    np.random.seed(seed)
```

Рисунок 3.2 – импорт библиотек

Чтобы сэкономить время при загрузке данных, мы будем работать с уменьшенной версией набора данных Speech Commands. Аудиоклипы набора данных хранятся в восьми папках, соответствующих каждой голосовой команде: no , yes , down , go , left , up , right и stop. Выгрузим наши данные (рисунок 3.3 – выгрузка данных).

```
[ ] DATASET_PATH = 'data/mini_speech_commands'

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

Downloading data from http://storage.googleapis.com/download.tensorflow.org/data/mini_speech_commands.zip
182082353/182082353 [=====] - 1s 0us/step

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

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

[ ] filenames = tf.io.gfile.glob(str(data_dir) + '/*/*')
filenames = tf.random.shuffle(filenames)
num_samples = len(filenames)
print('Number of total examples:', num_samples)
print('Number of examples per label:',
      len(tf.io.gfile.listdir(str(data_dir/commands[0]))))
print('Example file tensor:', filenames[0])

Number of total examples: 8000
Number of examples per label: 1000
Example file tensor: tf.Tensor(b'data/mini_speech_commands/no/4f2ab70c_nohash_1.wav', shape=(), dtype=bytes)
```

Рисунок 3.3 – выгрузка данных.

Обработка аудио файлов перед обучением модели по распознаванию речи может помочь улучшить качество модели и ее точность. Это может включать в себя фильтрацию шума, нормализацию громкости, извлечение признаков из звуковых волн и т.д. Обработка данных также может помочь улучшить скорость обучения и уменьшить время, необходимое для обучения модели. Будем работать с аудио файлом `/yes/0ab3b47d_nohash_0.wav`. Проанализируем графически аудиодорожку до фильтрации (рисунок 3.4).

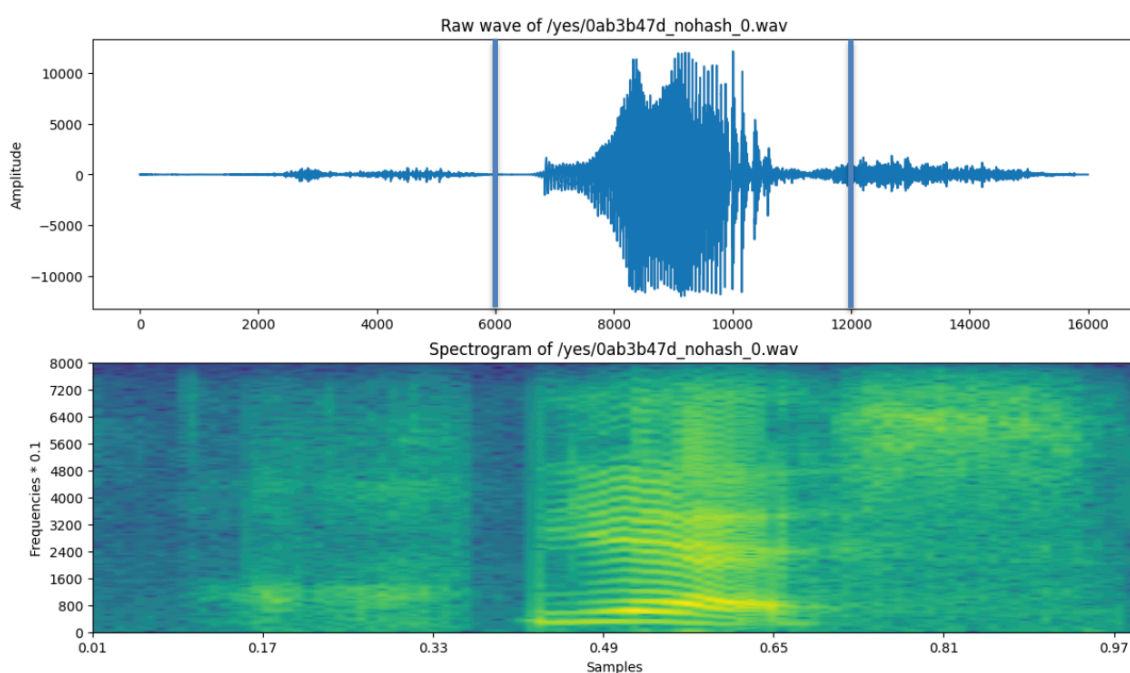


Рисунок 3.4 – спектрограмма аудиодорожки

Как мы видим, что на аудиодорожке и спектрограмма есть шумы – это участки до 6500 и после 13000. Посмотрим, как будут выглядеть графики после обработки (рисунок 3.5).

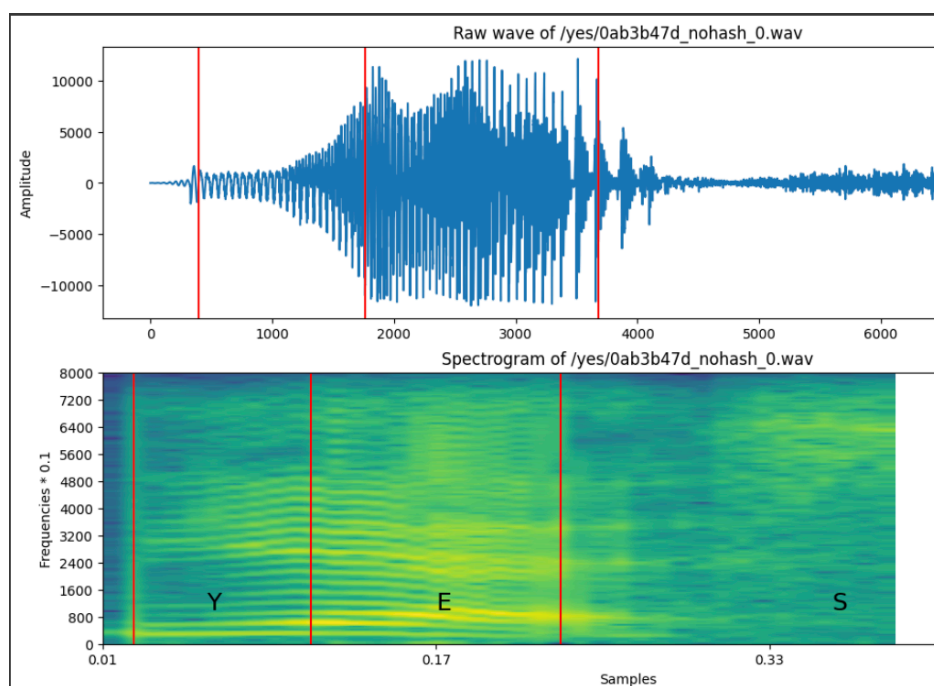


Рисунок 3.4 – спектрограмма аудиодорожки

По теореме Найквиста—Шеннона—Котельникова можно посчитать частоту вхождения: частоты находятся в диапазоне – 0,8

$$s(t) = \sum_{k=-\infty}^x s(kT) \frac{\sin 2\pi f_B(t - kT)}{2\pi f_B(t - kT)}$$

$s(kT)$ – дискретные значения функции, f_B – верхняя частота спектра.

Поскольку сигнал производит разные звуки по мере его изменения во времени, составляющие его частоты также меняются со временем. Другими словами, его спектр меняется со временем. Спектрограммы генерируются из звуковых сигналов с использованием преобразований Фурье. (рисунок 3.7)

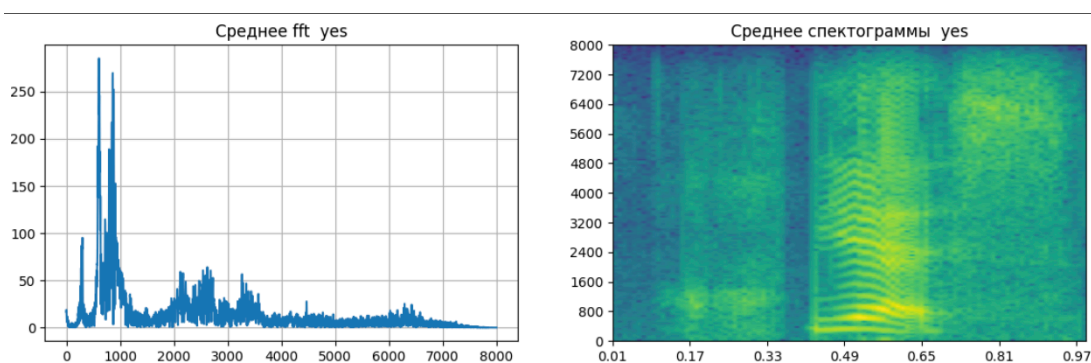


Рисунок 3.7 – преобразования Фурье

Спектрограмма сигнала отображает его спектр с течением времени и похожа на «фотографию» сигнала. Он отображает время по оси x и частоту по оси y.

Он использует разные цвета для обозначения амплитуды или силы каждой частоты. Чем ярче цвет, тем выше энергия сигнала. Каждый вертикальный «срез» спектрограммы, по сути, представляет собой спектр сигнала в данный момент времени и показывает, как мощность сигнала распределяется по каждой частоте, обнаруженной в сигнале в этот момент.

Люди не воспринимают частоты линейно. Мы более чувствительны к различиям между более низкими частотами, чем между более высокими частотами. Мы слышим их в логарифмической шкале, а не в линейной. Чтобы учесть это, была разработана шкала Мела.

Переведем нашу аудиодорожку в спектрограмму со шкалой Мела (рисунок 3.8)

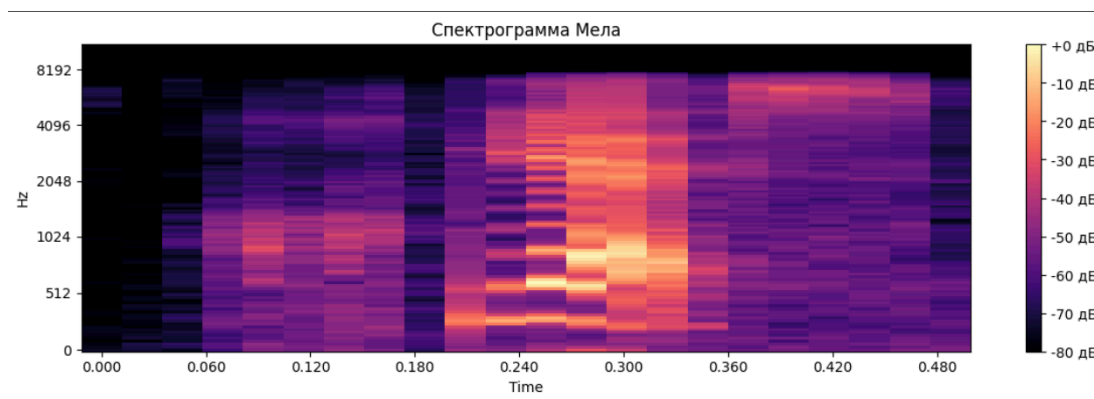


Рисунок 3.8 – спектрограмма со шкалой Мела

Извлекаем признаки из звуковых волн, используя метод MFCC (количество коэффициентов равно 13), а затем вычислим второй порядок дельта-коэффициентов. (рисунок 3.9)

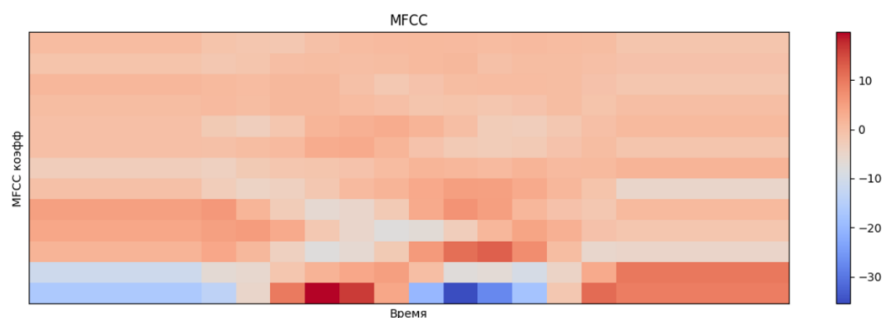


Рисунок 3.9 – метод MFCC

3.2. Обучение

Цель нашей работы заключается в создании алгоритма, который сможет распознавать простые голосовые команды. Разделим `filenames` на обучающие, проверочные и тестовые наборы, используя соотношение 80:10:10 соответственно. (рисунок 3.10)

```
[41] train_files = filenames[:6400]
     val_files = filenames[6400: 6400 + 800]
     test_files = filenames[-800:]

     print('Размер тренировочного набора', len(train_files))
     print('Размер набора для проверки', len(val_files))
     print('Размер тестового набора', len(test_files))

Размер тренировочного набора 6400
Размер набора для проверки 800
Размер тестового набора 800
```

Рисунок 3.10 – разделение набора

Создадим модель для классификации звуковых команд. (рисунок 3.11) Он будет использовать спектрограммы звуковых файлов в качестве входных данных, нормализовать их и пропускать через несколько сверточных слоев и полно связанных слоев, чтобы получить выходные данные, которые представляют вероятности принадлежности к каждому из заданных классов. Функция `model.summary()` выводит сводку модели, включая количество параметров и форму каждого слоя

```
for spectrogram, _ in spectrogram_ds.take(1):
    input_shape = spectrogram.shape
    print('Input shape:', input_shape)
    num_labels = len(commands)

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

    model = models.Sequential([
        layers.Input(shape=input_shape),
        # Downsample the input.
        layers.Resizing(32, 32),
        # Normalize.
        norm_layer,
        layers.Conv2D(32, 3, activation='relu'),
        layers.Conv2D(64, 3, activation='relu'),
        layers.MaxPooling2D(),
        layers.Dropout(0.25),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(num_labels),
    ])

    model.summary()
```

Рисунок 3.11 – модель для классификации звуковых команд

На данном этапе создается модель нейронной сети для классификации звуковых команд на основе их спектрограмм. Она определяет форму входных данных, количество классов, использует слой нормализации для подготовки данных к обучению, а затем создает последовательную модель с несколькими сверточными слоями, пулингом¹, слоями Dropout² и полносвязными слоями³. Код выводит сводку модели.

Будем совершать 8 входов в нейронную сеть, это позволит использовать информацию о различных характеристиках аудиофайла. Каждый вход представляет собой спектрограмму, полученную с помощью различных параметров. Использование нескольких спектрограмм позволяет моделировать зависимости между различными характеристиками аудиофайла и его классом, что может улучшить точность классификации.

Входной тензор является трехмерным массивом (tensor) с размерами 124 по первому измерению, 129 по второму измерению и 1 по третьему измерению. Каждое измерение представляет собой количество элементов в соответствующей оси тензора. В данном случае, 124 означает количество пикселей изображения по ширине, 129 – по высоте и 1 (один канал) обозначает, что изображение черно – белое. Таким образом, входной тензор содержит $124 \times 129 \times 1 = 15996$ элементов.

Модель содержит последовательность слоев (рисунок 3.12), начиная с Resizing и Normalization. Затем идут два сверточных слоя Conv2D с ReLU и фильтрами размера 3×3 ⁴. Слой MaxPooling2D уменьшает размерность, а Dropout используется для регуляризации. Следующие слои – Flatten и полносвязный Dense с ReLU и 128 нейронами. Снова используется Dropout. Последний полносвязный слой Dense с одним нейроном и sigmoid⁵ для

¹ Пулинг (pooling) в сверточных нейронных сетях – это операция уменьшения размерности изображения путем выбора наиболее значимых значений в определенном окне.

² Dropout – это метод регуляризации в нейронных сетях, который заключается в случайном отключении (обнулении) некоторых нейронов во время обучения.

³ Полносвязные слои используются в нейронных сетях для выполнения операций линейной алгебры, таких как умножение матриц и сложение векторов

⁴ Фильтр 3×3 – стандартный размер фильтра для обработки изображения.

⁵ Sigmoid – функция активации

бинарной классификации. Общее количество параметров – 1,625,611, все обучаемые.

```
Input shape: (124, 129, 1)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
resizing (Resizing)	(None, 32, 32, 1)	0
normalization (Normalization)	(None, 32, 32, 1)	3
conv2d (Conv2D)	(None, 30, 30, 32)	320
conv2d_1 (Conv2D)	(None, 28, 28, 64)	18496
max_pooling2d (MaxPooling2D)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 128)	1605760
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 8)	1032

```
=====
Total params: 1,625,611
Trainable params: 1,625,608
Non-trainable params: 3
```

Рисунок 3.12 – результат работы модели

Обучим модель (рисунок 3.13) на обучающем наборе данных (train_ds) и оценим ее производительность на валидационном наборе данных (val_ds) в течение a-эпох. Будем использовать обратный вызов EarlyStopping, который прерывает обучение, если модель перестает улучшаться на валидационном наборе данных после двух эпох. Результаты обучения сохраняются в объекте history.

Рекомендуется начинать с небольшого количества эпох (например, 10) и постепенно увеличивать количество эпох до тех пор, пока модель не перестанет улучшаться на валидационных данных.

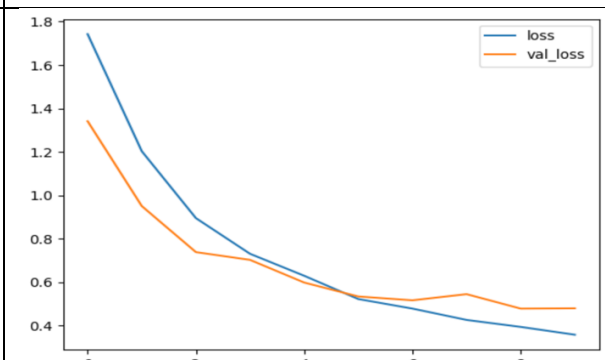
```
[ ] a = 10
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=a,
    callbacks=tf.keras.callbacks.EarlyStopping(verbose=1, patience=2),
)

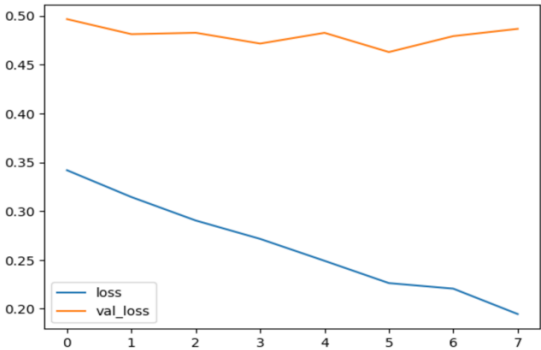
Epoch 1/10
100/100 [=====] - 39s 372ms/step - loss: 1.7424 - accuracy: 0.3717 - val_loss: 1.3415 - val_accuracy: 0.5600
Epoch 2/10
100/100 [=====] - 36s 359ms/step - loss: 1.2035 - accuracy: 0.5831 - val_loss: 0.9510 - val_accuracy: 0.6913
Epoch 3/10
100/100 [=====] - 36s 356ms/step - loss: 0.8955 - accuracy: 0.6816 - val_loss: 0.7390 - val_accuracy: 0.7625
Epoch 4/10
100/100 [=====] - 36s 357ms/step - loss: 0.7313 - accuracy: 0.7370 - val_loss: 0.7033 - val_accuracy: 0.7650
Epoch 5/10
100/100 [=====] - 42s 421ms/step - loss: 0.6302 - accuracy: 0.7763 - val_loss: 0.5990 - val_accuracy: 0.8050
Epoch 6/10
100/100 [=====] - 35s 346ms/step - loss: 0.5230 - accuracy: 0.8136 - val_loss: 0.5348 - val_accuracy: 0.8150
Epoch 7/10
100/100 [=====] - 35s 346ms/step - loss: 0.4791 - accuracy: 0.8309 - val_loss: 0.5173 - val_accuracy: 0.8238
Epoch 8/10
100/100 [=====] - 37s 372ms/step - loss: 0.4269 - accuracy: 0.8467 - val_loss: 0.5456 - val_accuracy: 0.8062
Epoch 9/10
100/100 [=====] - 38s 373ms/step - loss: 0.3948 - accuracy: 0.8609 - val_loss: 0.4791 - val_accuracy: 0.8475
Epoch 10/10
100/100 [=====] - 37s 373ms/step - loss: 0.3588 - accuracy: 0.8716 - val_loss: 0.4804 - val_accuracy: 0.8512
```

Рисунок 3.13 – обучение модели

В результате обучения точность на обучающем наборе постепенно увеличивается. Ниже представлена сравнительная таблица точностей.

Таблица 1 – Сравнительная таблица точности на обучающем наборе данных

а - эпох	Обучающий набор	Валидационный набор	График потерь
10	0,8589	0,8363	 <p>Функция потерь на обучающем и валидационном наборах данных уменьшается, это означает, что модель продолжает улучшаться и лучше подстраивается под данные.</p>

12	0,9292	0,8413	 <p>Функция потерь на обучающем наборе данных уменьшается, а на валидационном увеличивается. Это означает, что модель находится в состоянии переобучения</p>
----	--------	--------	--

Оптимальное количество эпох – 10.

Выполним тестирование модели на тестовом наборе данных (рисунок 3.15). Будем проходить через каждый элемент тестового набора данных и добавляет аудио и метки в соответствующие списки. Затем использовать модель для предсказания меток для каждого аудиофайла в тестовом наборе и сравнивает предсказанные метки с реальными метками из тестового набора.

```
[ ] test_audio = []
    test_labels = []

    for audio, label in test_ds:
        test_audio.append(audio.numpy())
        test_labels.append(label.numpy())

    test_audio = np.array(test_audio)
    test_labels = np.array(test_labels)

    y_pred = np.argmax(model.predict(test_audio), axis=1)
    y_true = test_labels

    test_acc = sum(y_pred == y_true) / len(y_true)
    print(f'Test set accuracy: {test_acc:.0%}')

25/25 [=====] - 1s 28ms/step
Test set accuracy: 84%
```

Рисунок 3.15 – пример тестирование модели с помощью метрики ассурасу

Таблица 1 – Сравнительная таблица точности с использованием разных метрик

Метрика	Точность	learning rate	Оптимизатор
Accuracy	84%	Без настройки	Adam
	85%	0,01	SGD
	85%		RMSporo
F1 – мера	86%	Без настройки	Adam
	86%	0,01	RMSporo
	86%		SGD

Видим, что результат точности модели не сильно отличается, можно считать, что точность модели с метрикой ассурасу или F1 – мера – $85\% \pm 1\%$

Построим матрицу путаницы. (рисунок 3.16) Код загружает звуковой файл, преобразует его в спектрограмму и использует обученную модель для предсказания класса звука. Затем он отображает матрицу с вероятностями принадлежности к каждому классу и названием класса, который был предсказан для данного звукового файла.

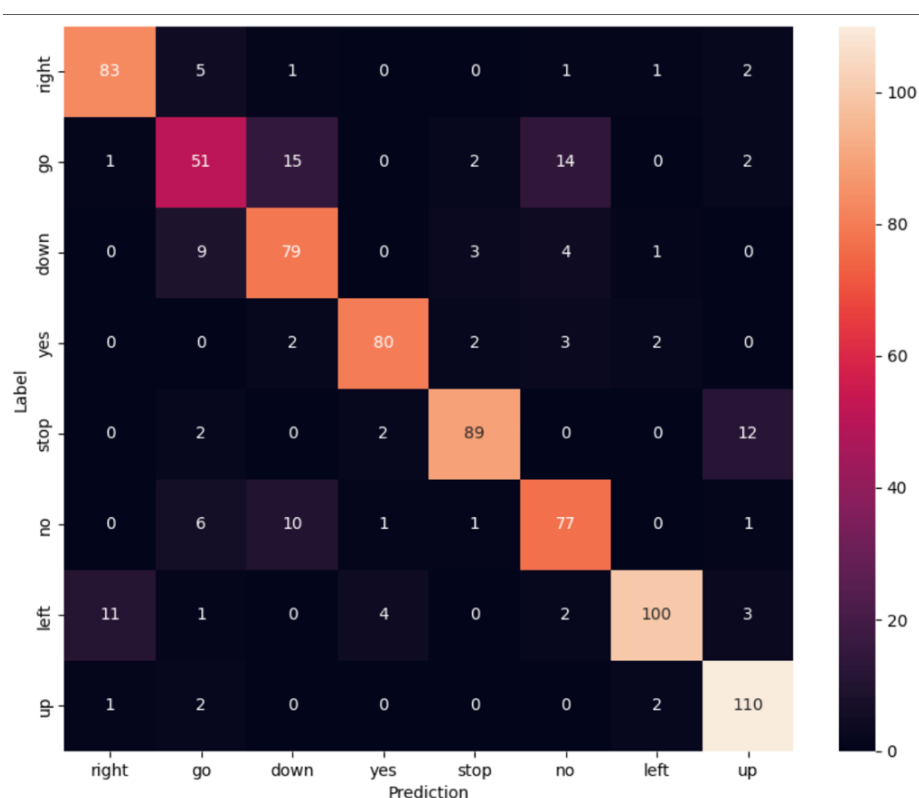


Рисунок 3.16 – матрица путаницы

Как мы видим, что больше всего неточностей было у команды go.

Выполним логические выводы по аудиофайлам:

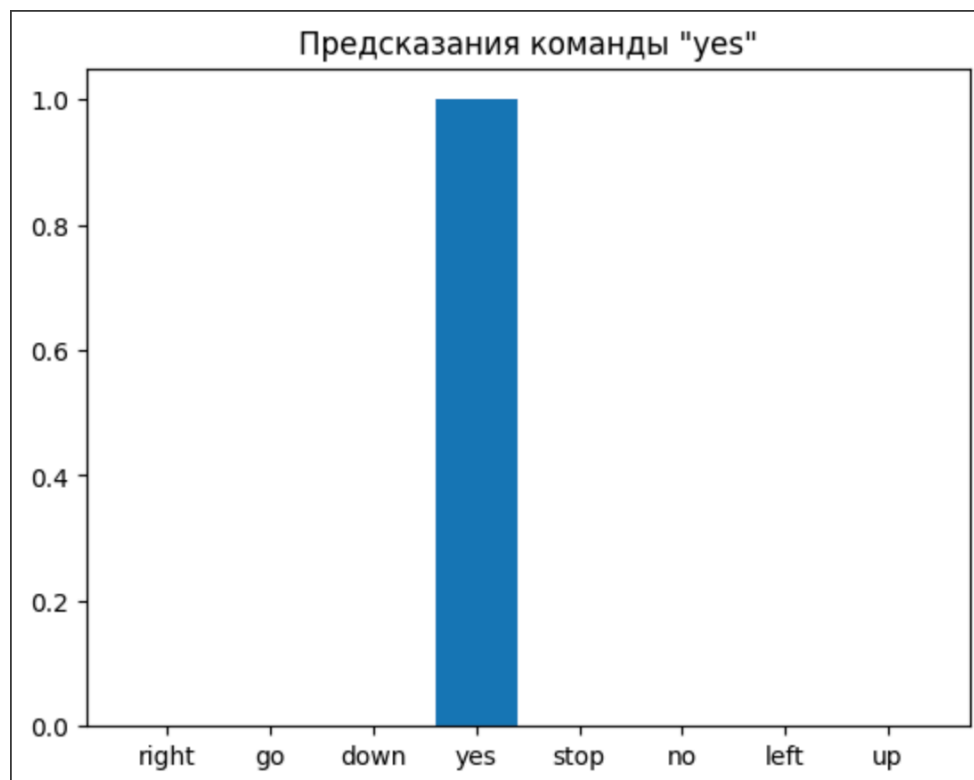


Рисунок 3.17 – гистограмма предсказания команды “yes”

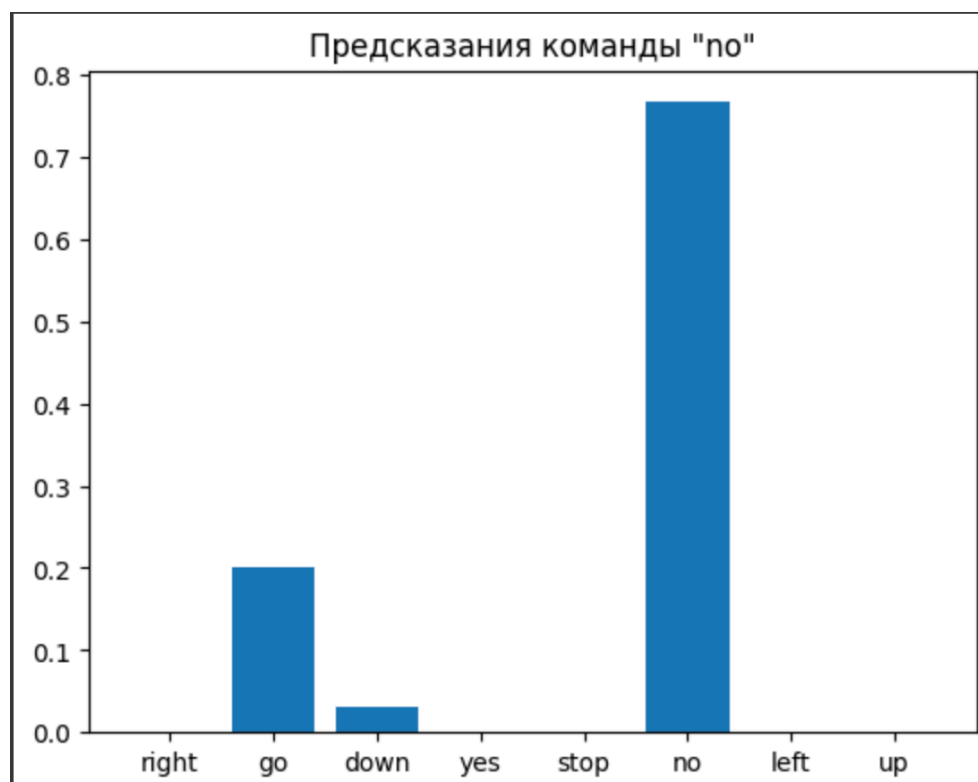


Рисунок 3.18 – гистограмма предсказания команды “no”

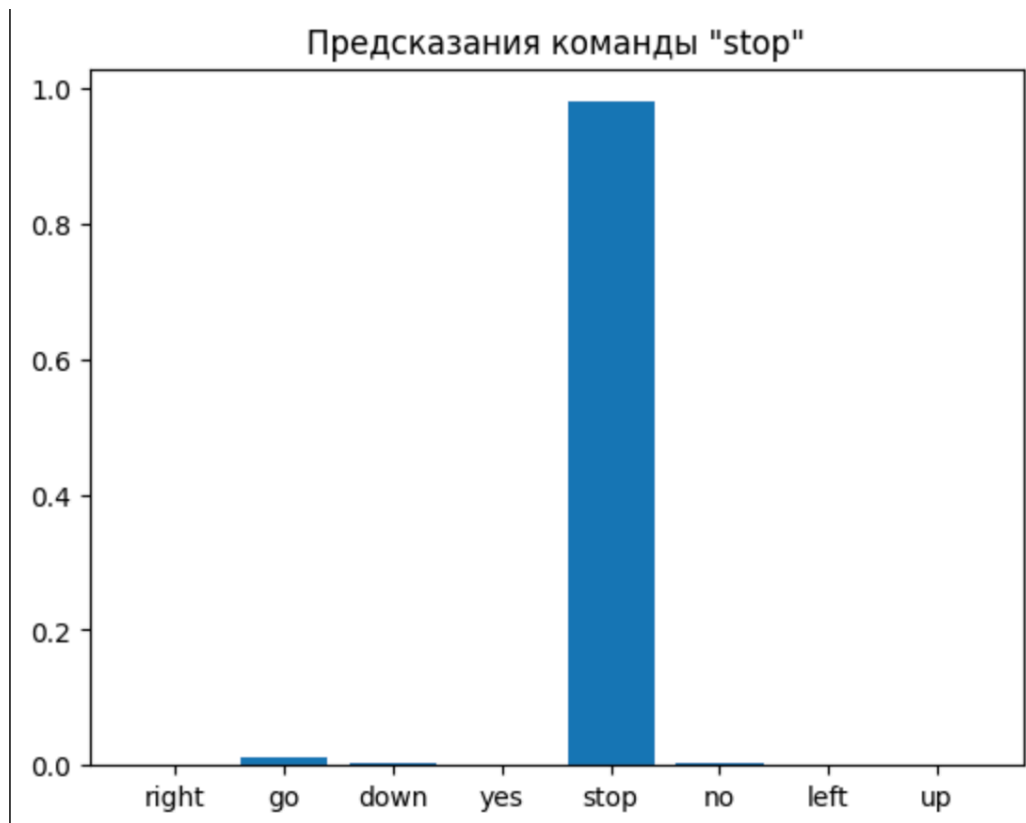


Рисунок 3.19 – гистограмма предсказания команды “stop”

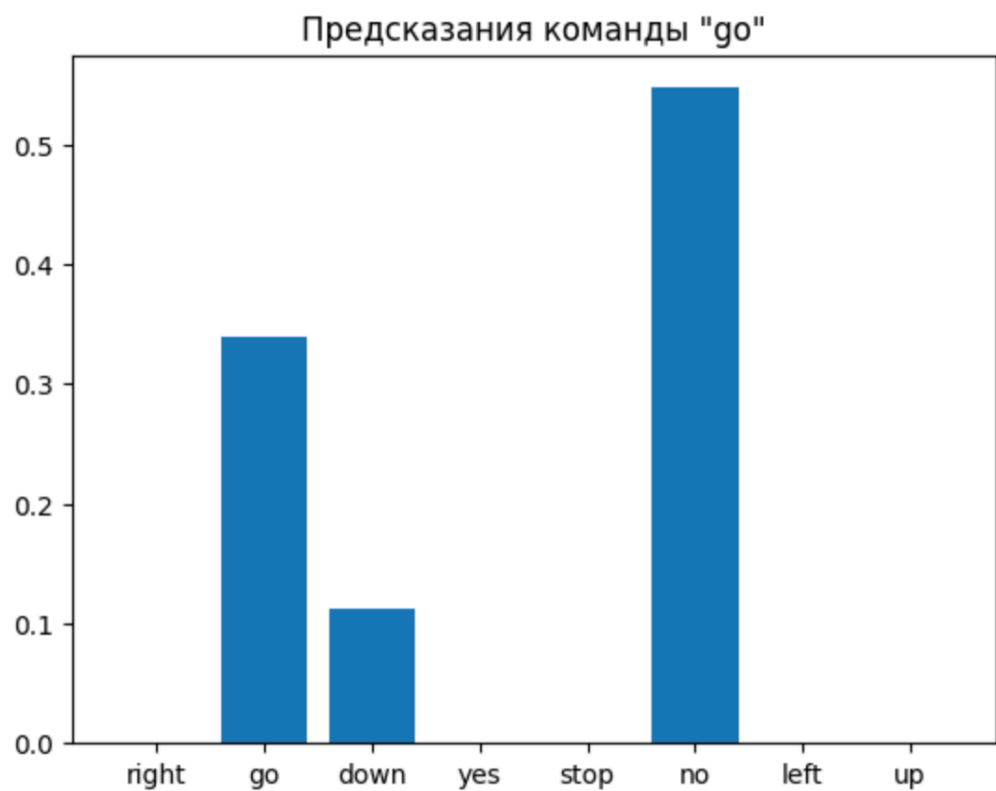


Рисунок 3.20 – гистограмма предсказания команды “go”

ЗАКЛЮЧЕНИЕ

1. В результате выполнения научно-исследовательской работы была решена задача классификации аудиофайлов на два класса: "сигнал" и "шум".

2. С использованием метода машинного обучения была обучена модель и протестирована с помощью различных метрик. Точность составила $85\% \pm 1\%$

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Яндекс. SpeechKit API [Электронный ресурс] – Дата обновления 24 окт. 2013, Режим доступа: <http://api.yandex.ru/speechkit/> (Дата обращения: 29.03.2023).
2. Brumberg J.S., Nieto-Castanon A, Kennedy P.R., Guenther F.H. (2010). Brain–computer interfaces for speech communication. *Speech Communication* 52:367–379. 2010 (Дата обращения: 29.03.2023).
3. Jorgensen C, Dusan S. (2010). Speech interfaces based upon surface electromyography. *Speech Communication*, 52: 354–366 (Дата обращения: 29.03.2023).
4. Жилияков Е.Г., Бабаринов С.Л., Чадюк П.В. Исследование сервиса компании Google Inc. по распознаванию русской речи Научные ведомости Белгородского Государственного Университета, №15 (158) 2013 г., выпуск 27/1 (Дата обращения: 29.03.2023).
5. Audio Deep Learning Made Simple: Automatic Speech Recognition (ASR), How it Works[Электронный ресурс] – Дата обновления 25 марта 2021, Режим доступа: <https://towardsdatascience.com/> (Дата обращения: 29.03.2023).