МИНОБРНАУКИ РОССИИ САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра САПР

**КУРСОВАЯ РАБОТА**

дисциплина «Алгоритмы и структуры данных»

Тема: «Создание голосового помощника на базе микроконтроллера»

Выполнили Студенты гр. 3351

\_\_\_\_\_\_\_\_\_\_\_\_Морозов А.А

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Фабер К.А.

Преподаватель

\_\_\_\_\_\_\_\_\_\_\_ Пестрев Д.О.

Санкт-Петербург

2025

**ЗАДАНИЕ**

**на курсовую работу**

1. Собрать схему из микроконтроллера, микрофона, дисплея и динамика.
2. Разработать скрипт для ESP32, который:

* Будет считывать сигнал с микрофона
* Отправлять записанный сигнал на сервер при помощи API
* Принимать ответ от сервера и исполнять команду на основе этого ответа

1. Разработать ПО для сервера:

* Разработать API для взаимодействия МК с сервером
* Разработать программу для обработки сигнала (на основе FFT или подобного)
* Разработать нейросеть для распознавания команд
* Реализовать хранение обработанных данных для дальнейшего обучения нейросети
* Обучить нейросеть

Основные команды:

1. «Дата»
2. «Время»
3. «Алгоритмы» (тестовая команда)

|  |
| --- |
| Содержание пояснительной записки:  «Содержание», «Введение», «Численный анализ цепи методом переменных состояния при действии одиночного импульса на входе», «Анализ цепи операторным методом при действии одиночного импульса на входе», «Анализ цепи спектральным методом при действии одиночного импульса на входе», «Анализ цепи частотным методом при периодическом воздействии», «Заключение», «Список использованных источников» |
| Предполагаемый объем пояснительной записки:  Не менее 25 страниц. |
| Дата выдачи задания: |
| Дата сдачи реферата: |
| Дата защиты реферата: |

Выполнили Студенты гр. 3351

\_\_\_\_\_\_\_\_\_\_\_\_Морозов А.А

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Фабер К.А.

Преподаватель

\_\_\_\_\_\_\_\_\_\_\_ Пестрев Д.О.

**Содержание.**

[Введение. 5](#_Toc199159702)

[1. Численный анализ цепи методом переменных состояния при действии одиночного импульса на входе. 8](#_Toc199159703)

[1.1. Составление системы уравнений состояния цепи и уравнения связи реакции цепи с переменными состояния. 8](#_Toc199159704)

[1.2. Нахождение корней характеристического полинома. 10](#_Toc199159705)

[1.3. Получение аналитического выражения для входного сигнала. 11](#_Toc199159706)

[1.4. Определение выходного сигнала с помощью численного метода решения системы уравнений состояния. 11](#_Toc199159707)

[2. Анализ цепи операторным методом при действии одиночного импульса на входе. 12](#_Toc199159708)

[2.1. Определение функции передачи. 12](#_Toc199159709)

[2.2. Нахождение нулей и полюсов функции передачи 15](#_Toc199159710)

[2.3. Определение изображения по Лапласу входного одиночного импульса 16](#_Toc199159711)

[2.4. Определение изображения выходного сигнала. 16](#_Toc199159712)

[3. Анализ цепи спектральным методом при действии одиночного импульса на входе. 18](#_Toc199159713)

[3.1. Нахождение АЧХ, ФЧХ, АФЧХ 18](#_Toc199159714)

[3.2. Определение полосы пропускания цепи. 20](#_Toc199159715)

[3.3. Определение амплитудного и фазового спектра входного одиночного импульса. 21](#_Toc199159716)

[3.4. Сопоставление спектров входного импульса с частотными характеристиками цепи. 23](#_Toc199159717)

[3.5. Определение амплитудного и фазового спектра выходного сигнала. 23](#_Toc199159718)

[4. Анализ цепи частотным методом при периодическом воздействии. 25](#_Toc199159719)

[4.1. Разложение в ряд Фурье заданного входного периодического сигнала. 25](#_Toc199159720)

[4.2. Построение на одном графике заданного входного периодического сигнала и его аппроксимации отрезком ряда Фурье. 27](#_Toc199159721)

[4.3. Определение и построение амплитудного и фазового дискретных спектров выходного периодического сигнала. 27](#_Toc199159722)

[4.4. Определение периодического выходного сигнала цепи в виде отрезка ряда Фурье. 28](#_Toc199159723)

[4.5. Заключение об искажении периодического сигнала на выходе цепи. 29](#_Toc199159724)

[Заключение. 30](#_Toc199159725)

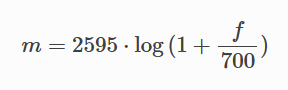
[Список использованных источников. 30](#_Toc199159726)

Введение.

1. Теоретическая часть
   1. Способы представления звука.

Любой звук, с физической точки зрения, является волной. Эти колебания характеризуются амплитудой, определяющей громкость, и частотой, отвечающей за высоту тона. Микрофон преобразует звуковые волны в электрические и тем самым позволяет оцифровать сигнал. После оцифровки сигнал можно преобразовывать в другие форматы, в зависимости от задачи. В случае классификации команд, более удобной формой представления звука является спектрограмма – она показывает какие частоты преобладают и как меняются с течением времени.

Однако, человеческий слух воспринимает различные частоты не линейно: мы лучше различаем низкие частоты и хуже верхние, поэтому вместо частот можно использовать мелы – психофизическая единица высоты звука. Мел-шкала и формула перехода от Герц к мелам были выведены эмпирически, посредством проведения большого количества психоакустических тестов:



Таким образом, вместо обычной спектрограммы, можно использовать mel-спектрограмму, которая более точно описывает человеческое восприятие звука.

Самым эффективным алгоритмом разложения сырого сигнала на частотные спектры является «быстрое преобразование Фурье», который имеет сложность .

* 1. Сверточная нейронная сеть ResNet18.

ResNet18 – сверточная нейросеть, предназначенная для классификации изображений. Её главным преимуществом является использование остаточных связей, которые позволяют решить проблему затухания градиента на начальных слоях. Архитектура ResNet18:

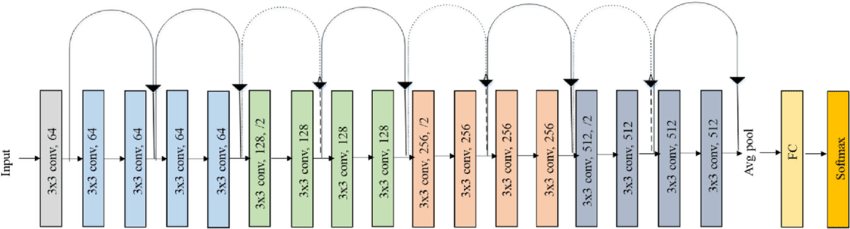


Рисунок – Архитектура ResNet18

ResNet18 состоит из 18 слоев, включая сверточные слои, max/avg poolling, активационные слои и полносвязный слой.

Сверточные слои получаются путем фильтрации входных данных определенным фильтром – он проходит по всей исходной матрице с определенным шагом и выстраивает новый слой, который содержит информацию об определенном признаке входных данных. Для изображений это могут быть такие признаки, как цвет, вертикальные линии, горизонтальные линии, резкие/плавные переходы и тд. Пример свертки:

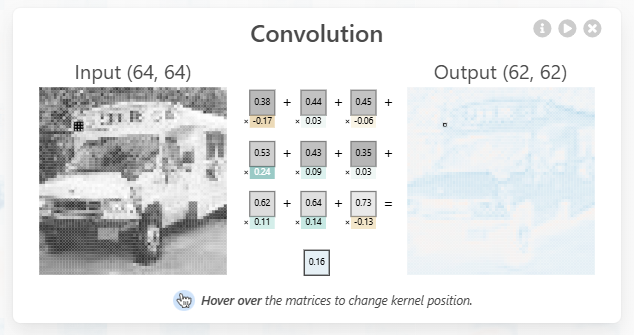
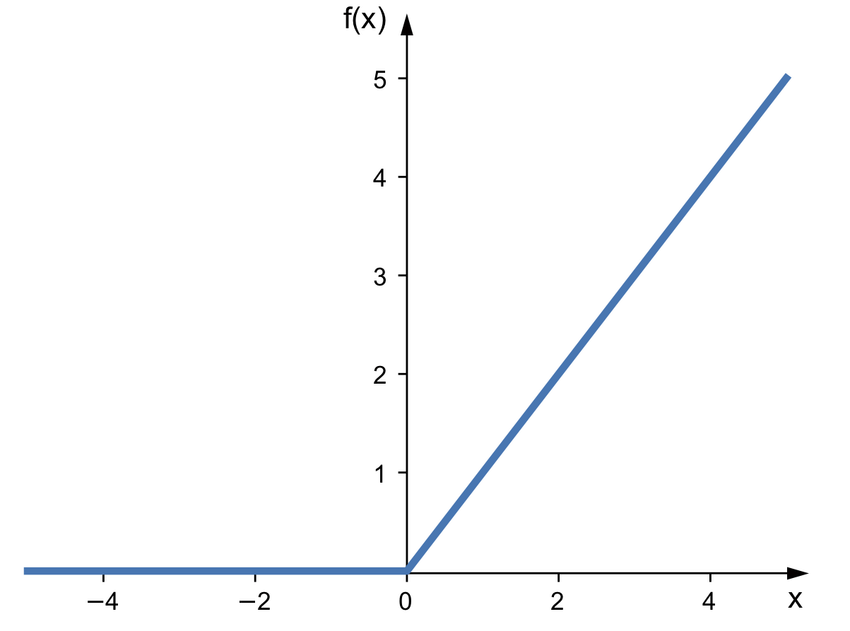


Рисунок – Пример свертки

В ResNet18 в качестве функции активации после свертки используется ReLU, которая возвращает исходное число, если оно положительное и 0, если число отрицательное:



Риснуок – ReLU

Слои max/avg pool используются для даунсемплинга исходной матрицы и помогают выделить общие детали:

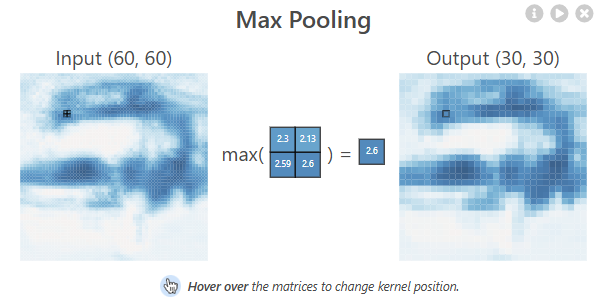


Рисунок – Пример Max pooling

ResNet18 обрабатывает входные данные следующим образом: сначала на вход подаются 3 канала, которые проходят через свертку, номрализацию по батчам, ReLU и MaxPooling. Далее идут остаточные блоки (4 шт.), каждый из которых содержит по 2 сверточных слоя с нормализацией и ReLU. В финальных слоях используется глобальный avg pooling и полносвязный слой для классификации.

Использование ResNet18 для классификации голосовых команд обуславливается возможностью представлять звуковой сигнал в виде mel-спектрограммы – то есть, в виде матрицы, что по сути, является одноканальным изображением. Так как ResNet18 работает только с трехканальными изображениями, необходимо модифицировать структуру, для поддержки одноканальных изображений: как минимум, нужно изменить количество входных данных для первого сверточного слоя. В дальнейшем, если это потребуется, можно изменить и остальные компоненты: например, уменьшить количество признаков при свертке или наоборот увеличить, т.к. признаки спектрограммы отличаются от признаков изображений.

1. Практическая часть

Начать обзор практической части курсовой работы следует с описания того, что у нас есть – компонентов. В качестве мозга для проекта был взят микроконтроллер (далее может называться просто МК) ESP32 Wroom32 38pin. В нём есть встроенный WiFi и Bluetooth модули, что значительно облегчает задачу. Кроме этого был использован микрофон на модуле MAX9814, матрица 8 на 8 пикселей на модуле MAX7219 и динамик 2 ватта.

* 1. Реализация прошивки на МК

В первую очередь была реализована прошивка для микроконтроллера ESP32 Wroom32, которая позволяет записывать аудио с микрофона, отправлять его на сервер для обработки и отображать результат на светодиодной матрице. Далее рассмотрю работу алгоритмов и прочие штуки по коду.

Опишу кратко функции в коде. Основная функция setup() инициализирует систему, настраивает подключение к WiFi, настраивает аналого-цифровой преобразователь (АЦП) для работы с микрофоном, устанавливает параметры светодиодной матрицы и подготавливает прерывание для кнопки. Ну это самая базовая функция для прошивки микроконтроллера с использованием <Arduino.h>. Функция loop() является основным циклом программы, который проверяет состояние кнопки и запускает процессы записи и отправки данных при нажатии. Функция buttonISR() обрабатывает прерывание от кнопки с учетом дребезга контактов. То есть когда я случайно буду нажимать много раз подряд кнопку, а она будет считать все нажатия ошибочно. Вот это я предусмотрел. Функция connectWiFi() устанавливает и поддерживает соединение с WiFi сетью. Функция recordAudio() отвечает за запись аудиоданных с микрофона в буфер с заданной частотой дискретизации. Функция prepareAudioChunkJson() формирует JSON-структуру для отправки части аудиоданных на сервер. Функция sendAudioToServer() осуществляет передачу данных на сервер частями (чанками) и обрабатывает ответ. Функция handleCommand() анализирует команду, полученную от сервера, и отображает соответствующую реакцию на матрице.

Частота дискретизации 16000 Гц была выбрана потому что это стандартная частота для речевых приложений, которая обеспечивает достаточное качество распознавания речи. Человеческая речь обычно лежит в диапазоне от 300 до 3400 Гц, поэтому 16 кГц более чем достаточно. Такая частота позволяет снизить нагрузку на микроконтроллер и уменьшить объем передаваемых данных по сравнению с более высокими частотами дискретизации. 8000 в пример не беру, по моему мнению этого мало.

Далее я реализовал отправку аудио данных на сервер. Они разбиваются на чанки, иначе говоря блоки, размером 2048 сэмплов. Для каждого чанка формируется JSON-структура, содержащая сами аудиоданные, информацию о частоте дискретизации, номере чанка и общем количестве чанков. Затем с помощью HTTP-клиента эти данные отправляются POST-запросом на указанный сервер. Используется протокол HTTP, а не HTTPS, так как в данном случае сервер находится в локальной сети. Курсовая работа подразумевает относительно простой перенос сервера на хостинг. Достаточно лишь загрузить файлы на хостинг, а также заменить в коде прошивки ссылку, по которой отправляются аудио данные. Хостинг уже может поддерживать защищенное соединение. Таким образом я могу относительно просто менять сервера для обработки аудио данных.

Получение ответа от сервера происходит после отправки последнего чанка данных. Сервер обрабатывает весь аудиопоток и отправляет ответ, который содержит команду для выполнения. Этот ответ парсится из JSON-формата, и если в нем присутствует поле "command", вызывается функция handleCommand() для выполнения соответствующего действия. Ответ от сервера приходит в виде текстовой строки, которая обрабатывается далее для отображения на светодиодной матрице. В целом можно что угодно там реализовать, хоть синтез речи через динамик.

Светодиодная матрица подключена через интерфейс SPI с использованием библиотеки GyverMAX7219. Матрица управляется через три пина: CS, DIN и CLK. Спасибо Alex Gyver за реально удобные библиотеки. У него много всего интересного на сайте, я посмотрел. В коде реализовано несколько предопределенных изображений, которые хранятся в памяти программы PROGMEM для экономии оперативной памяти. Для чего используется матрица я уже описал.

Микрофон подключен к аналоговому входу микроконтроллера через АЦП. В данном случае используется канал ADC1\_CHANNEL\_0, который соответствует GPIO36 (это пин SP на моем МК). АЦП микроконтроллера ESP32 настроен на 12-битное разрешение, то есть его диапазон значений от 0 до 4096. В коде после чтения значения из АЦП вычитается 2048 для центрирования сигнала относительно нуля. Я получается преобразовал из беззнакового в знаковый тип данных. Это облегчает обработку.

Если вдруг кто не знает… АЦП (аналого-цифровой преобразователь другими словами) – это устройство, преобразующее непрерывный аналоговый сигнал в цифровой код. В микроконтроллере ESP32 встроен 12-битный АЦП. То есть я могу представить входное напряжения одним из 4096 дискретных значений. Микрофон подключен к пину SP (GPIO36, он же SP, он же VP иногда подписывается), потому что этот пин специально предназначен для аналогового ввода и подключен к первому АЦП (ADC1). Это стандартный пин для аналогового ввода в ESP32, который обеспечивает стабильное чтение аналоговых сигналов.

Что касается типа данных, то я использовал int16\_t (16-битное знаковое целое) для хранения аудиосэмплов. Во-первых, это стандартный формат для представления аудиоданных, который поддерживается большинством аудиобиблиотек и серверов обработки звука. Во-вторых, 16 бит обеспечивают достаточную динамику для представления аудиосигнала (96 дБ), что важно для качественной записи звука. В-третьих, этот тип данных занимает меньше места в памяти по сравнению с 32-битными форматами, но при этом обеспечивает лучшее качество по сравнению с 8-битными форматами. Кроме того, использование знакового типа позволяет корректно представлять аудиосигнал, который имеет положительные и отрицательные значения относительно нуля.

Использование сервера для обработки данных вместо выполнения всех вычислений на микроконтроллере было выбрано потому что сервер обладает значительно большей вычислительной мощностью. Это позволяет использовать сложные алгоритмы обработки звука и распознавания речи. Во-вторых, на сервере проще обновлять и модифицировать алгоритмы обработки без необходимости перепрошивки микроконтроллера. Ну и также сервер может использовать предобученные модели машинного обучения, которые требуют значительных ресурсов и не могут быть запущены на микроконтроллере.

* 1. Сбор и обработка данных

Для входных данных было решено использовать следующие параметры:

N\_FFT = 512

N\_MELS = 64

HOP\_LEN = 256

SAMPLE\_RATE = 16000

DURATION = 1.5

Таким образом, получаются спектрограммы размерностью (64, 94)

Данные были разделены на 4 класса:

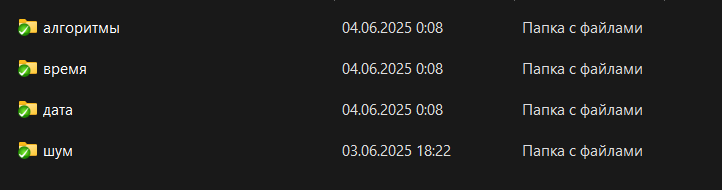


Рисунок – классы команд

Для заполнения датасета был написан скрипт на Python, который считывает сигнал с микрофона ПК, строит mel-спектрограмму, производит нормализацию и сохраняет данные в нужный класс. Пример входных данных:

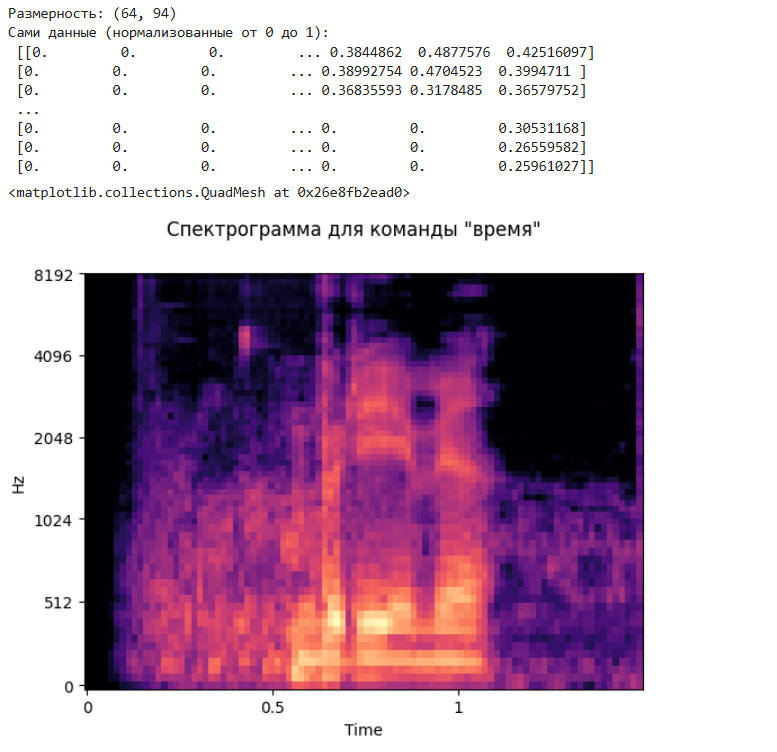
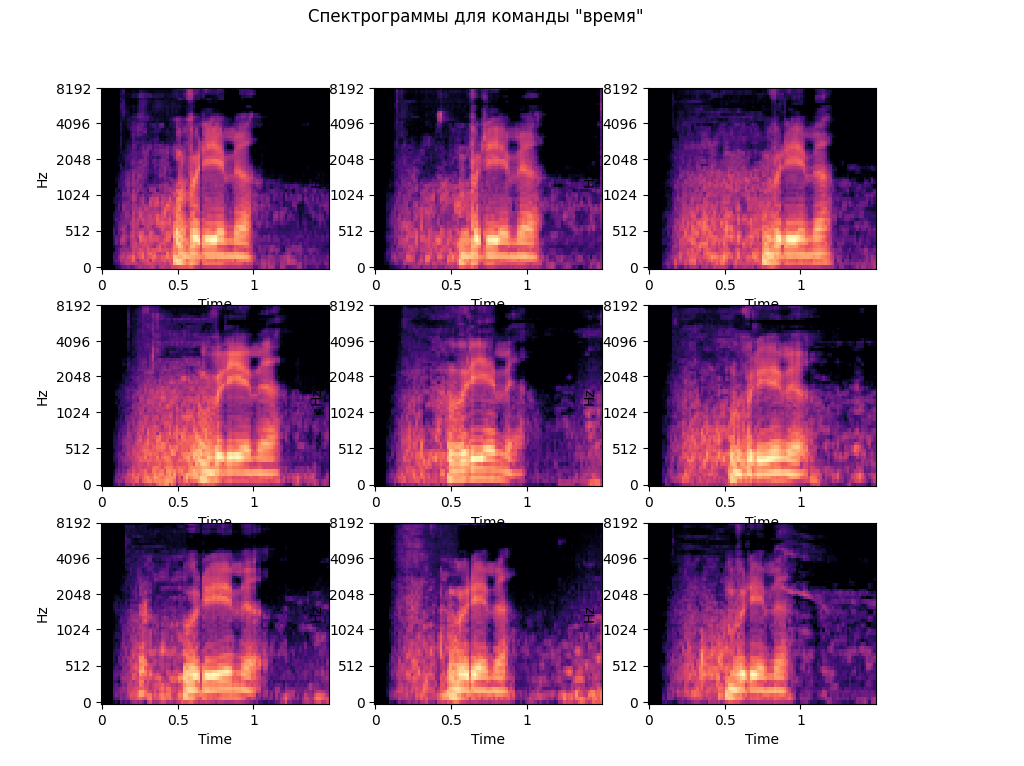
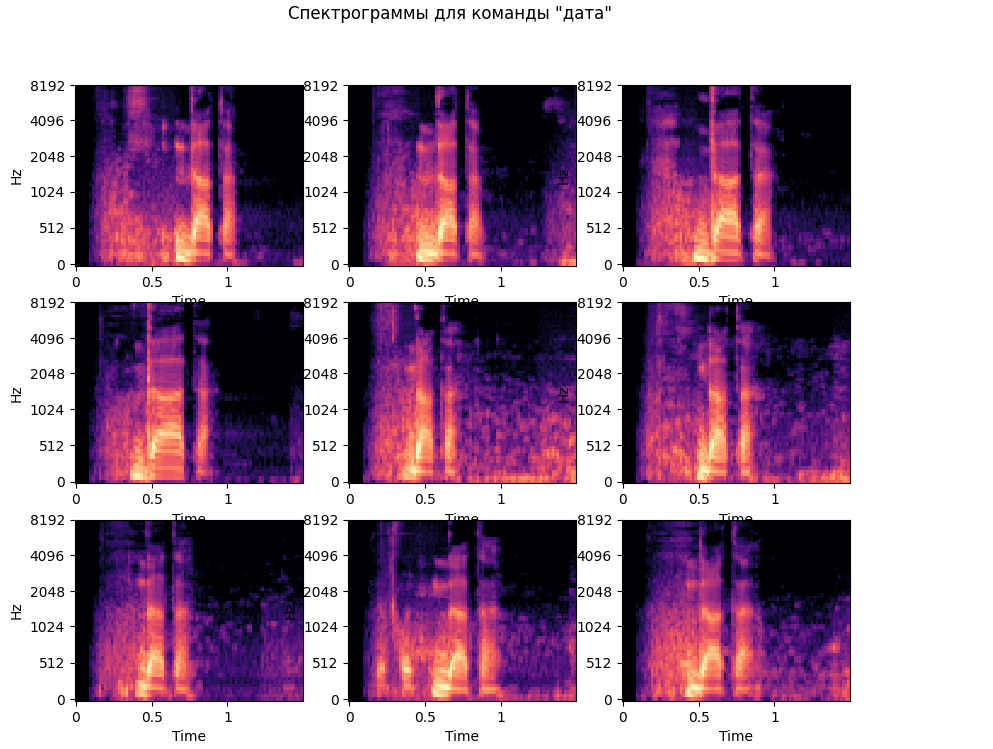
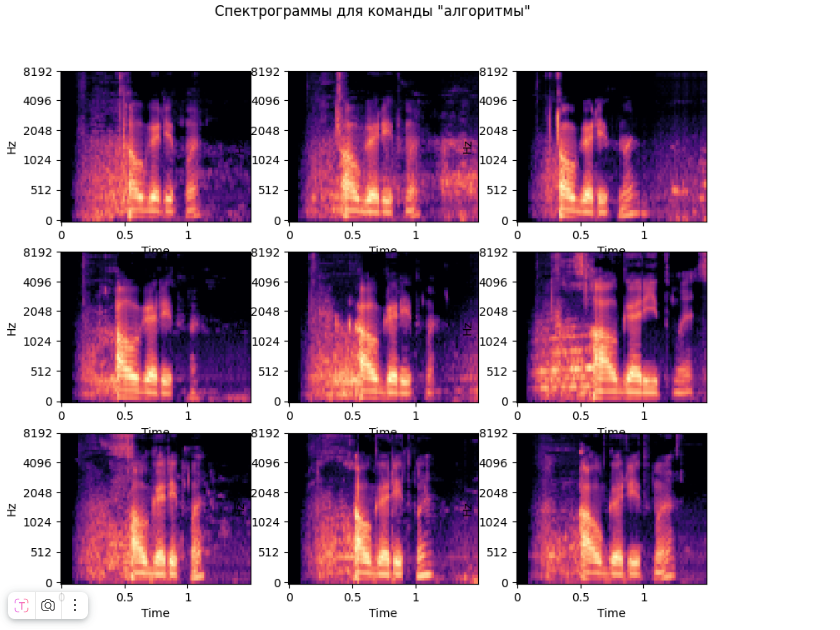


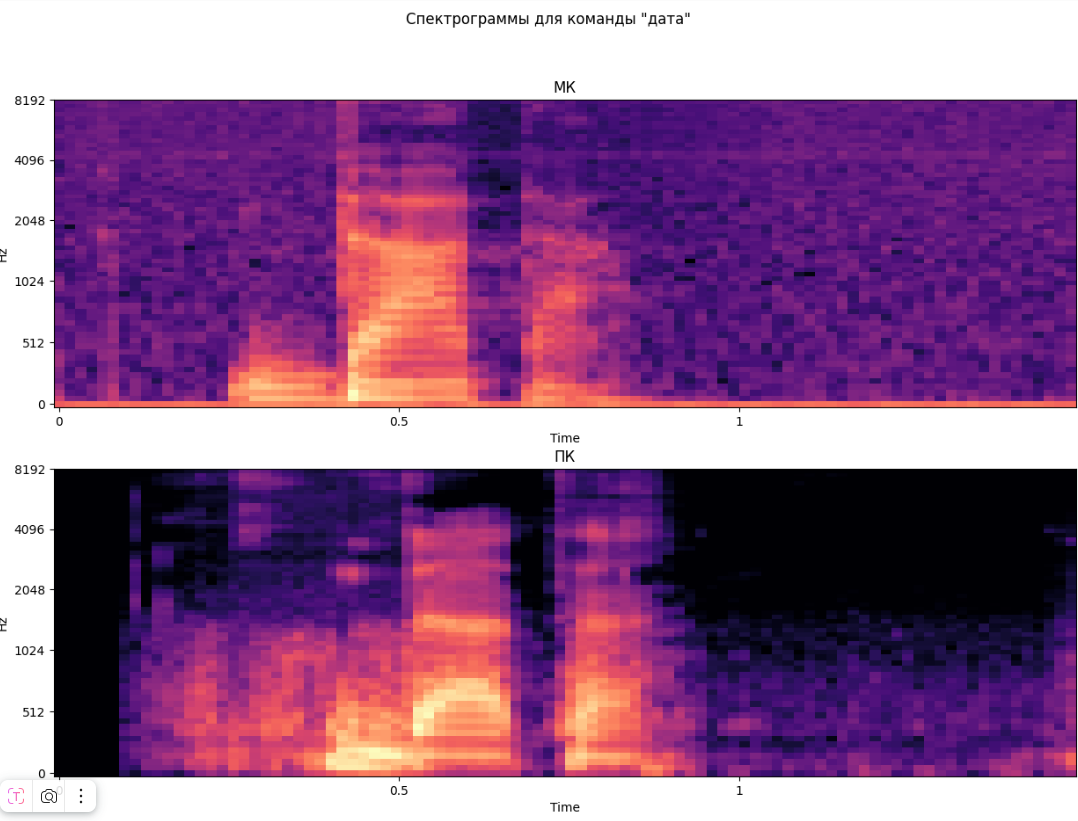
Рисунок – Пример входных данных

Еще несколько примеров спектрограмм:  


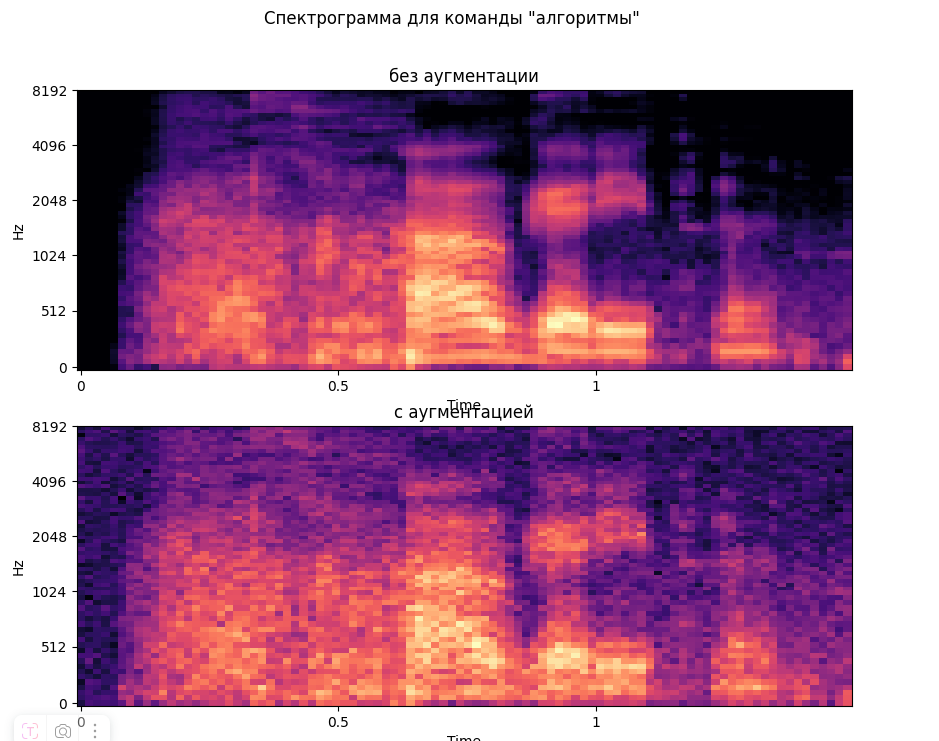




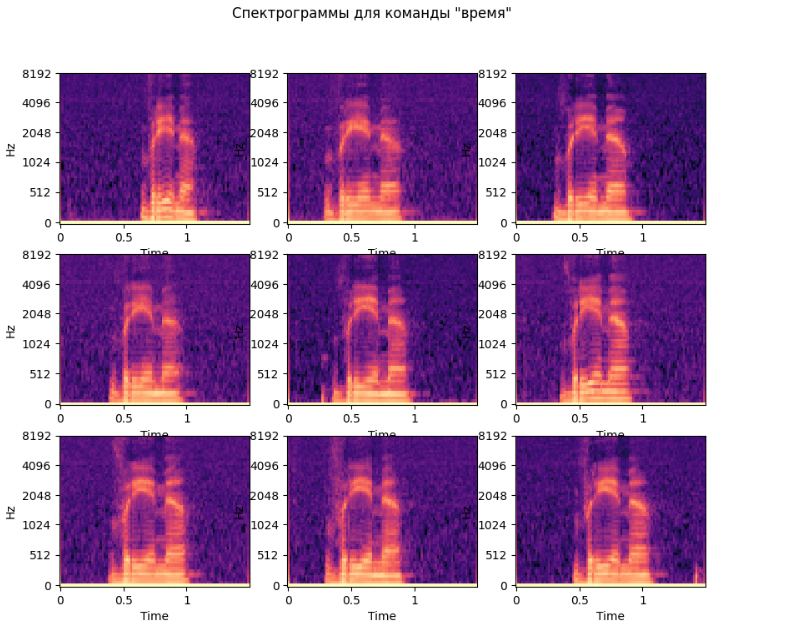
Сравнение спектрограммы из датасета с спектрограммой с микрофона микроконтроллера:

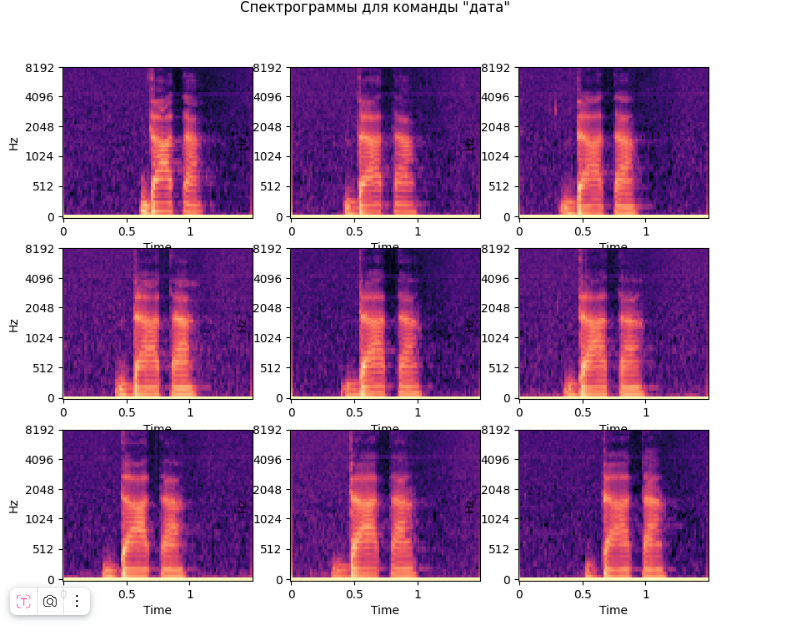


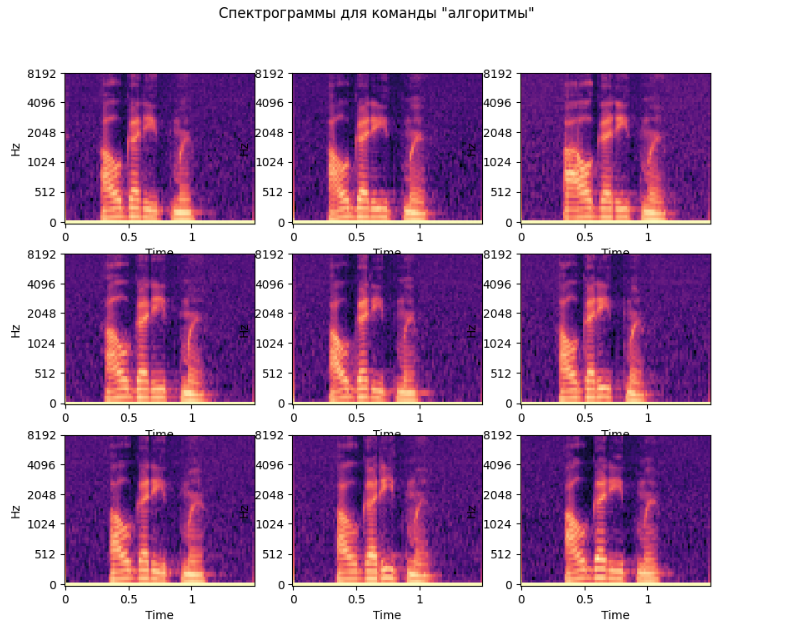
Ввиду наличия большого количества шумов при записи на микрофон микроконтроллера была проведена аугментация данных:



Также, датасет был дополнен некоторыми спектрограммами с микрофона микроконтроллера для более точных предсказаний.

Спектрограммы с МК:  






* 1. Разработка нейросети

Нейросеть разрабатывалась на основе структуры ResNet18 с измененным первым сверточным слоем:

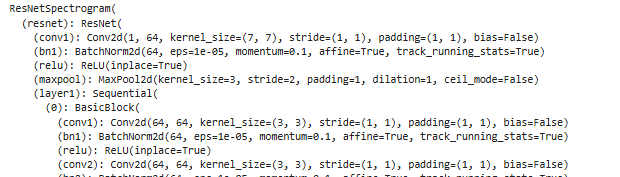


Рисунок – обновленная структура ResNet18

Теперь сверточный слой принимает 1 канал с паддингом 1 и проходит по спектрограмме ядром размера 7 и шагом 1, при такой конфигурации модель уже имеет точность около 99%, поэтому дальнейшие модификации не производились.

Тренировочная выборка представляет собой 80% датасета, а валидационная, соответственно, 20%.

Процесс обучения нейронной сети:

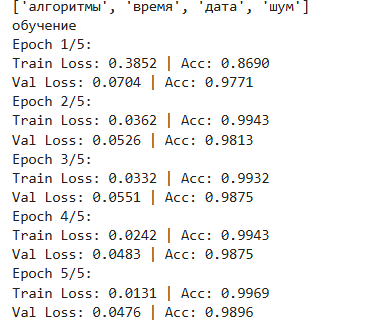


Рисунок – Точность модели по эпохам

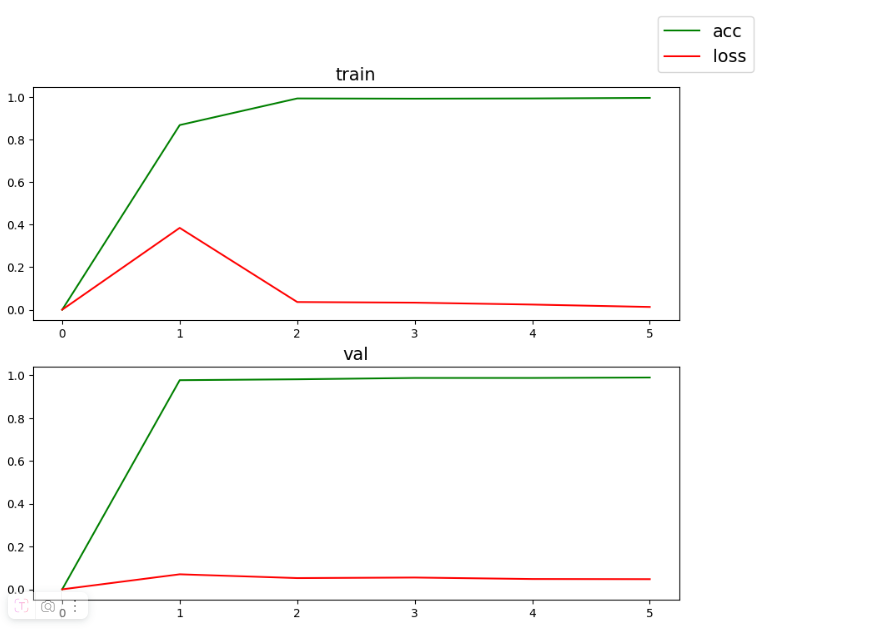


Рисунок – Визуализация обучения

Резкое возрастание точности может быть обусловлено наличием однообразных данных, и возможно, модель будет показывать плохие результаты в нестандартных условиях (шум, разный голос, разная скорость речи и т.д.), однако при обычных обстоятельствах модель показывает себя хорошо.

1. Ссылки:

Репозиторий GitHub:

1. Код
   1. Скрипт для ESP32
   2. Скрипт на Python для обработки сигнала
   3. Сбор данных и функции для обработки

**fill\_data.py**

import librosa

import matplotlib.pyplot as plt

import numpy as np

import sounddevice as sd

import pickle

import os

N\_FFT = 512

N\_MELS = 64

HOP\_LEN = 256

SAMPLE\_RATE = 16000

DURATION = 1.5

def get\_signal():

    print("Запись началась...")

    signal = sd.rec(int(DURATION \* SAMPLE\_RATE),

                   samplerate=SAMPLE\_RATE,

                   channels=1,

                   dtype='float32')

    sd.wait()

    print("Запись завершена!")

    return signal.flatten()

def create\_mel\_spec(signal):

    fft = librosa.stft(y = signal, n\_fft=N\_FFT, hop\_length=HOP\_LEN)

    power\_spec = np.abs(fft) \*\* 2

    mel\_spec = librosa.feature.melspectrogram(S=power\_spec, sr =SAMPLE\_RATE, n\_fft=N\_FFT, hop\_length=HOP\_LEN, n\_mels = N\_MELS)

    mel\_spec\_db = librosa.power\_to\_db(mel\_spec, ref=np.max)

    return mel\_spec\_db

def normalize(mel\_spec\_db):

    return (mel\_spec\_db - np.min(mel\_spec\_db)) / (np.max(mel\_spec\_db) - np.min(mel\_spec\_db))

def save\_spec(mel\_spec, filename):

    with open(filename, 'wb') as f:

        pickle.dump(mel\_spec, f)

def fill(FROM = 0, N = 100, debug = False, classes = ['алгоритмы', 'дата', 'время']):

    norm\_path = './data/norm/'

    wo\_norm\_path = './data/wo\_normalize/'

    raw\_path = './data/raw/'

    for cls in classes:

        for i in range(FROM, FROM + N):

            print(f'запись {i} для класса: {cls}')

            input('enter for record')

            signal = get\_signal()

            if not debug or bool(int(input("Пишем? 1- ДА, 0 - НЕТ:\n"))):

                mel\_spec = create\_mel\_spec(signal)

                norm\_mel\_spec = normalize(mel\_spec)

                save\_spec(mel\_spec, wo\_norm\_path + cls + f'/{i}.pkl')

                save\_spec(norm\_mel\_spec, norm\_path + cls + f'/{i}.pkl')

                save\_spec(signal, raw\_path + cls + f'/{i}.pkl')

            else:

                i -= 1

def fill\_noise(FROM = 0, N = 100):

    norm\_path = './data/norm/'

    wo\_norm\_path = './data/wo\_normalize/'

    raw\_path = './data/raw/'

    print('запись для класса:\t шум')

    for i in range(FROM, FROM + N):

        #input('enter for record')

        signal = get\_signal()

        mel\_spec = create\_mel\_spec(signal)

        norm\_mel\_spec = normalize(mel\_spec)

        save\_spec(mel\_spec, wo\_norm\_path + 'шум' + f'/{i}.pkl')

        save\_spec(norm\_mel\_spec, norm\_path + "шум" + f'/{i}.pkl')

        save\_spec(signal, raw\_path + "шум" + f'/{i}.pkl')

def augmentation\_data():

    norm\_path = './data/norm/'

    wo\_norm\_path = './data/wo\_normalize/'

    raw\_path = './data/raw/'

    paths = [norm\_path, wo\_norm\_path]

    classes = ['алгоритмы', 'дата', 'время']

    f = open(norm\_path + 'алгоритмы/1.pkl', 'rb')

    audio = pickle.load(f)

    f.close()

    noise = np.random.normal(0, 0.06, audio.shape)

    for path in paths:

        for c in classes:

            for filename in os.listdir(path + c):

                filepath = os.path.join(path + c, filename)

                f = open(filepath,  'rb')

                audio = pickle.load(f)

                f.close()

                audio\_noisy = normalize(audio + noise)

                f = open(filepath[:-4] + 'A.pkl',  'wb')

                pickle.dump(audio\_noisy, f)

                f.close()

if \_\_name\_\_ ==  '\_\_main\_\_':

    #print(normalize(create\_mel\_spec(get\_signal())).shape)

    #fill(FROM=400, N = 100)

    #fill\_noise(FROM=300)

    #augmentation\_data()

* 1. Нейросеть (Структура + обучение)

import torch

import torch.nn as nn

import torch.optim as optim

from torchvision.models import resnet18

from torch.utils.data import DataLoader, TensorDataset

import numpy as np

import pickle

import os

from sklearn.model\_selection import train\_test\_split

def load\_data(data\_dir):

spectrograms = []

labels = []

class\_names = sorted(os.listdir(data\_dir))

for class\_idx, class\_name in enumerate(class\_names):

class\_dir = os.path.join(data\_dir, class\_name)

for pkl\_file in os.listdir(class\_dir):

if pkl\_file.endswith('.pkl'):

with open(os.path.join(class\_dir, pkl\_file), 'rb') as f:

spec = pickle.load(f)

spectrograms.append(spec)

labels.append(class\_idx)

return np.array(spectrograms), np.array(labels), class\_names

class ResNetSpectrogram(nn.Module):

def \_\_init\_\_(self, num\_classes):

super().\_\_init\_\_()

self.resnet = resnet18(pretrained=False)

self.resnet.conv1 = nn.Conv2d(1, 64, kernel\_size=7, stride=1, padding=0, bias=False)

self.resnet.avgpool = nn.Identity()

self.resnet.fc = nn.Identity()

self.adaptive\_pool = nn.AdaptiveAvgPool2d((1, 1))

self.fc = nn.Linear(512, num\_classes)

def forward(self, x):

x = self.resnet.conv1(x)

x = self.resnet.bn1(x)

x = self.resnet.relu(x)

x = self.resnet.maxpool(x)

#print(x.shape)

x = self.resnet.layer1(x)

x = self.resnet.layer2(x)

x = self.resnet.layer3(x)

x = self.resnet.layer4(x)

#print(x.shape)

x = self.adaptive\_pool(x)

#print(x.shape)

x = torch.flatten(x, 1)

#print(x.shape)

x = self.fc(x)

#print(x.shape)

return x

def prepare\_loaders(X, y, test\_size=0.2, batch\_size=32):

X = torch.FloatTensor(X).unsqueeze(1)

y = torch.LongTensor(y)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(

X, y, test\_size=test\_size, random\_state=42, stratify=y

)

train\_loader = DataLoader(

TensorDataset(X\_train, y\_train),

batch\_size=batch\_size,

shuffle=True

)

val\_loader = DataLoader(

TensorDataset(X\_val, y\_val),

batch\_size=batch\_size

)

return train\_loader, val\_loader

def train\_model(model, train\_loader, val\_loader, classes, epochs=15, lr=0.0001):

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

model = model.to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=lr)

scheduler = optim.lr\_scheduler.ReduceLROnPlateau(optimizer, 'max', patience=2)

train\_loss\_data = [0]

val\_loss\_data = [0]

train\_acc\_data = [0]

val\_acc\_data = [0]

best\_acc = 0

for epoch in range(epochs):

model.train()

train\_loss, correct = 0, 0

for inputs, labels in train\_loader:

inputs, labels = inputs.to(device), labels.to(device)

optimizer.zero\_grad()

outputs = model(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

train\_loss += loss.item()

\_, predicted = torch.max(outputs, 1)

correct += (predicted == labels).sum().item()

model.eval()

val\_loss, val\_correct = 0, 0

all\_preds, all\_labels = [], []

with torch.no\_grad():

for inputs, labels in val\_loader:

inputs, labels = inputs.to(device), labels.to(device)

outputs = model(inputs)

val\_loss += criterion(outputs, labels).item()

\_, predicted = torch.max(outputs, 1)

val\_correct += (predicted == labels).sum().item()

all\_preds.extend(predicted.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

# Метрики

train\_loss /= len(train\_loader)

train\_acc = correct / len(train\_loader.dataset)

val\_loss /= len(val\_loader)

val\_acc = val\_correct / len(val\_loader.dataset)

train\_loss\_data.append(train\_loss)

train\_acc\_data.append(train\_acc)

val\_loss\_data.append(val\_loss)

val\_acc\_data.append(val\_acc)

print(f"Epoch {epoch+1}/{epochs}:")

print(f"Train Loss: {train\_loss:.4f} | Acc: {train\_acc:.4f}")

print(f"Val Loss: {val\_loss:.4f} | Acc: {val\_acc:.4f}")

if val\_acc > best\_acc:

best\_acc = val\_acc

torch.save(model.state\_dict(), 'best\_model.pth')

scheduler.step(val\_acc)

return train\_loss\_data, train\_acc\_data, val\_loss\_data, val\_acc\_data

if \_\_name\_\_ == "\_\_main\_\_":

DATA\_DIR = "./data/norm/"

BATCH\_SIZE = 20

EPOCHS = 5

X, y, classes = load\_data(DATA\_DIR)

print(f'размер датасета {X.shape[0]}')

train\_loader, val\_loader = prepare\_loaders(X, y, batch\_size=BATCH\_SIZE)

model = ResNetSpectrogram(num\_classes=len(classes))

print(classes)

print('обучение')

train\_loss, train\_acc, val\_loss, val\_acc = train\_model(model, train\_loader, val\_loader, classes, epochs=EPOCHS)