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

ОТЧЕТ

по индивидуальному заданию
по дисциплине «Искусственные нейронные сети»
Тема: Определение цифры по аудио.

Студент гр. 7383 _____ Александров Р.А.

Студент гр. 7383 _____ Зуев Д.В.

Студент гр. 7383 _____ Рудоман В.А.

Преподаватель _____ Жукова Н. А.

Санкт-Петербург
2020

Цель работы.

Реализовать предсказание цифры, которая была сказана по аудиофайлу.

Постановка задачи.

В данном индивидуальном задании необходимо реализовать нейросеть, которая по аудиофайлу, содержащему произношение цифры, определяет, что за цифра была произнесена.

Выполнение работы.

1. Представление данных.

В ходе выполнения работы был использован датасет из 2000 аудиофайлов, в которых записаны произношения цифр от 0 до 9. Цифры произносятся четырьмя дикторами на английском языке по 50 на каждого диктора.

Для решения поставленной задачи следовало преобразовать аудиофайлы в числовые данные, чтобы сеть могла с ними работать. Для этого было решено перевести аудиофайлы в мел-частотные кепстральные коэффициенты или MFCC.

Мел-частотный кепструм (MFC) является представлением кратковременного спектра мощности звука, на основе линейного косинусного преобразования из спектра мощности журнала на нелинейной Мел шкале частоты.

MFCC – это коэффициенты, которые в совокупности составляют MFC.

Вычисление этих векторов производится с помощью функций, представленных в библиотеке `librosa`. Перевод из .wav в MFCC выполняется в функции `wav2mfcc`. Исходный код программы представлен в приложении А.

Функция `draw_spectrogram` демонстрирует спектрограмму, полученную в функции `wav2mfcc`. Пример спектрограмм представлен на рис. 1-4.

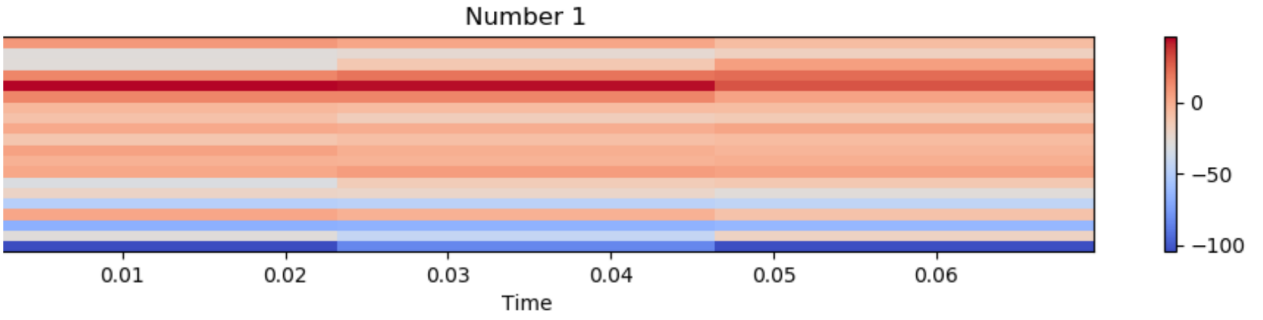

Рисунок 1 - Спектрограмма числа 1

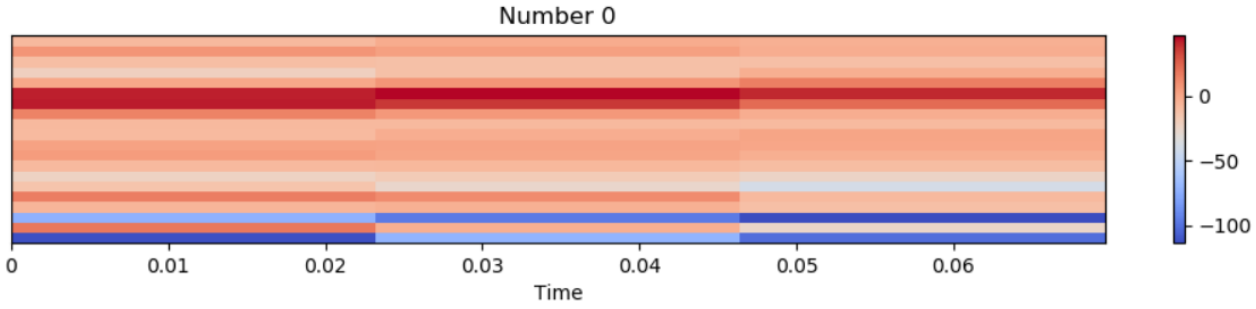

Рисунок 2 - Спектрограмма числа 0

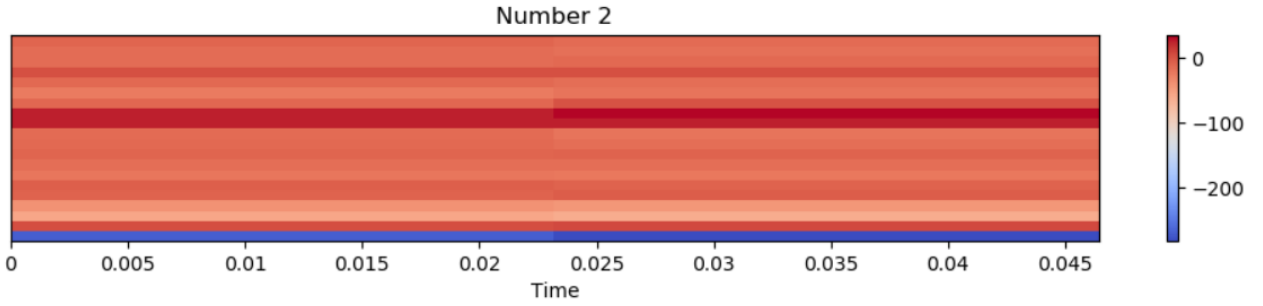

Рисунок 3 - Спектрограмма числа 2

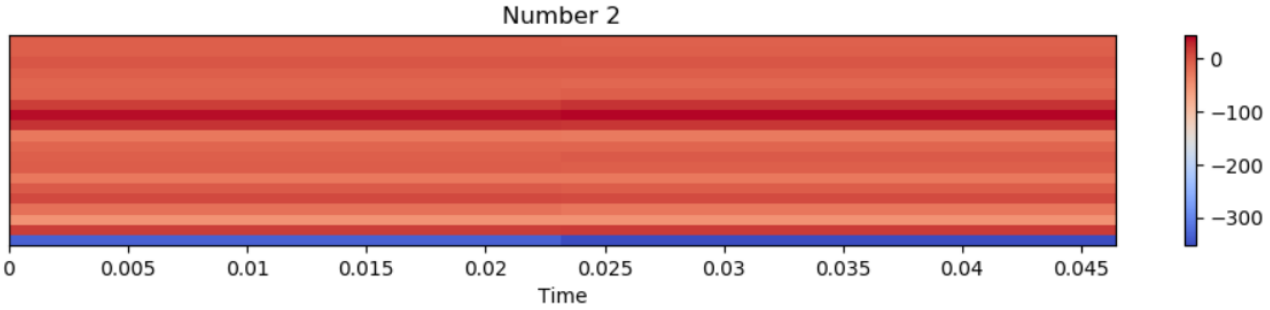

Рисунок 4 - Другая спектрограмма числа 2

По спектрограммам можно определить, что распределение данных не одинаковое на них, поэтому целесообразно использовать пакетную нормализацию, нежели обычную.

С помощью функции `reshape` тензоры были преобразованы в 4-х мерные (т.к. изображение представляется 3-х мерным тензором, а первое число – это количество образцов).

Данные разделялись на 3 набора: 2 набора по 1620 и 180 образцов использовались при обучении как тренировочный и тестовый наборы. Данные из третьего набора размером 200 образцов использовались при тестировании обученной сети. Данная мера может помочь в борьбе с переобучением сети [2].

2. Подключение GPU.

Для ускорения процесса обучения нейросетей была использована видеокарта.

Так как TensorFlow поддерживает выполнение вычислений на различных типах устройств, включая CPU и GPU, то было принято решение ради ускорения обучения нейросетей перенести вычисления на графический процессор. Подключение проводилось при использовании технологии CUDA.

3. Выбор архитектуры.

Для обработки аудиофайлов была выбрана сверточная нейросеть, так данные представлены в виде двумерных спектрограмм, что можно интерпретировать как изображения.

Первоначально была выбрана архитектура сети, представленная на рис. 2.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(2, 2), activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=categorical_crossentropy, optimizer=SGD(lr=0.01), metrics=['accuracy'])
```

Рисунок 5 - Архитектура 1

Определение цифры по аудиофайлу является задачей многоклассовой, многозначной (определение вероятности принадлежности к тому или иному классу) классификации, то в качестве функции потерь была выбрана функция категориальной кроссэнтропии, а в качестве функции активации выходного слоя была выбрана функция `softmax`.

В качестве оптимизатора был выбран `SGD`, согласно [1] он показывает лучшие результаты на сверточных нейронных сетях с обратным распространением ошибки.

Точность модели представлена на рис. 3.

Рисунок 6 - Точность первой модели

Добавим слои разреживания для уменьшения переобучения нормализации пакетов и слои пакетной нормализации, так как распределение признаков в спектрограммах может отличаться, и это помогает увеличить производительность [3]. Полученная сеть представлена на рис. 4.

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(2, 2), activation='relu', input_shape=input_shape))
model.add(BatchNormalization())

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(num_classes, activation='softmax'))
opt = SGD(lr=0.01)
model.compile(loss=categorical_crossentropy, optimizer=opt, metrics=['accuracy'])
```

Рисунок 7 - Архитектура 2

Точность модели представлена на рис. 5.

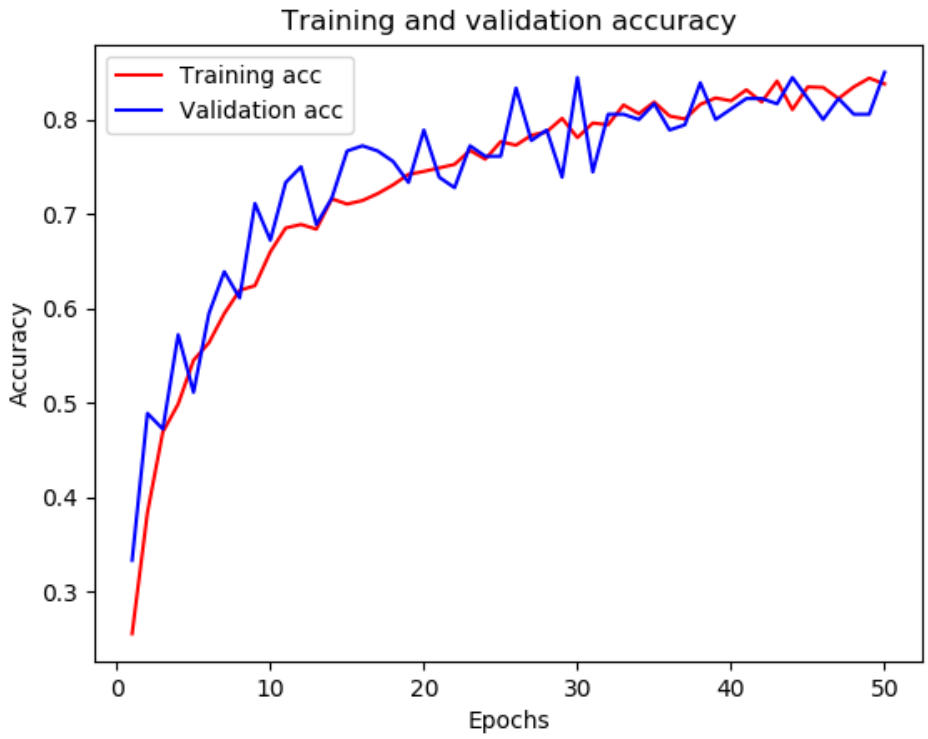

Рисунок 8 - Точность второй сети

Добавим еще один слой свертки. Получим сеть, представленную на рис. 6.

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(2, 2), activation='relu', input_shape=input_shape))
model.add(BatchNormalization())

model.add(Conv2D(32, kernel_size=(2, 2), activation='relu'))
model.add(BatchNormalization())

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(num_classes, activation='softmax'))
opt = SGD(lr=0.01)
model.compile(loss=categorical_crossentropy, optimizer=opt, metrics=['accuracy'])
```

Рисунок 9 - Архитектура 3

Точность сети представлена на рис. 7.

Рисунок 10 - Точность третьей архитектуры

Как видно, точность значительно повысилась по сравнению с предыдущими моделями.

Попробуем еще увеличить число слоев свертки, а также число полносвязных слоев. Полученная сеть представлена на рис. 7.

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(2, 2), activation='relu', input_shape=input_shape))
model.add(BatchNormalization())

model.add(Conv2D(48, kernel_size=(2, 2), activation='relu'))
model.add(BatchNormalization())

model.add(Conv2D(120, kernel_size=(2, 2), activation='relu'))
model.add(BatchNormalization())

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())

model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())

model.add(Dropout(0.4))
model.add(Dense(num_classes, activation='softmax'))
opt = SGD(lr=0.01)
model.compile(loss=categorical_crossentropy, optimizer=opt, metrics=['accuracy'])
```

Рисунок 11 - Архитектура 4

Точность модели представлена на рис. 8.

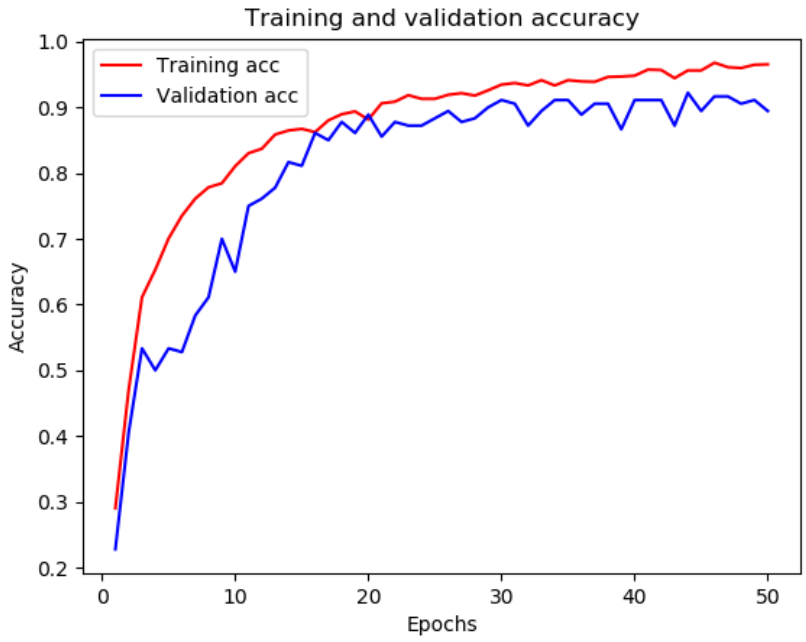

Рисунок 12 - Точность последней модели

4. Использование коллбэков.

Для отслеживания окончания обучения нейросетей был написан коллбэк `EmailNotificationCallback`, отправляющий на почту уведомление о завершении обучения и точность обучения. Пример сообщения представлен на рис. 10.

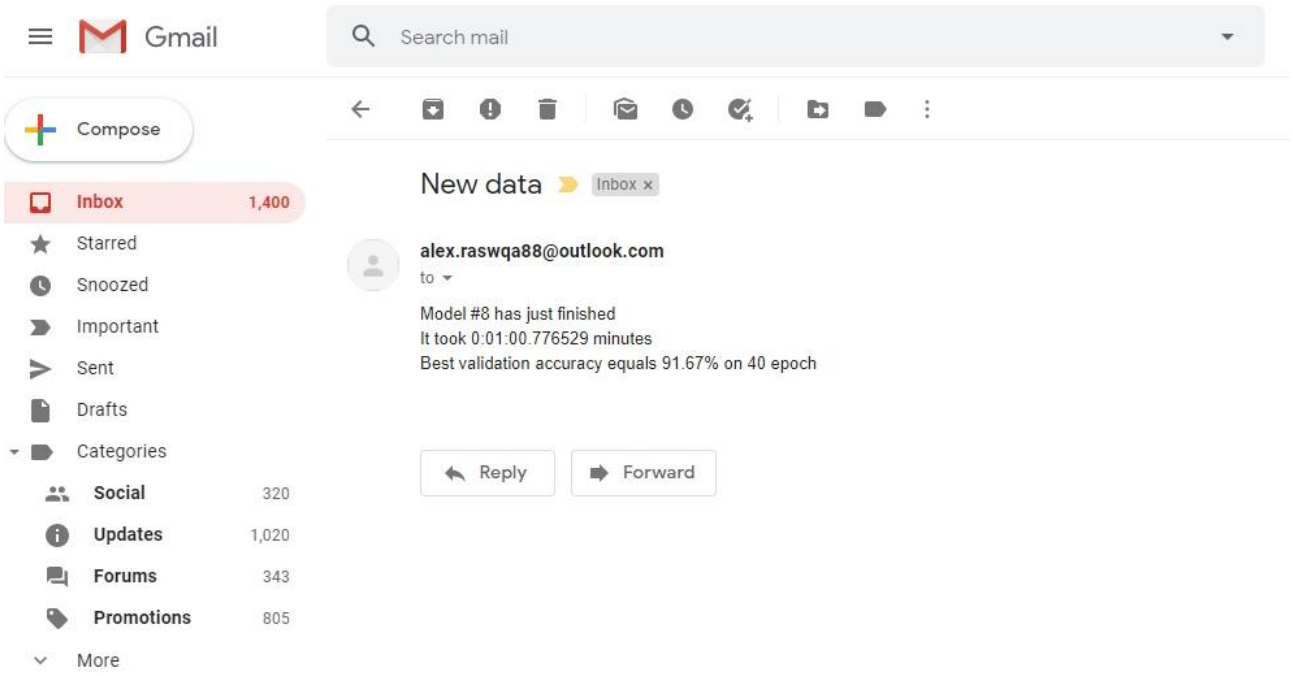

Рисунок 13 - Оповещение о завершении обучения

По графикам точности видно, что при в процессе обучения модели точность на тестовых данных осциллирует, поэтому для сохранения лучшей модели при обучении был использован коллбэк `ModelCheckpoint` с параметром `save_best_only=True`.

Для более наглядного сравнения моделей сетей с разными архитектурами был использован коллбэк `TensorBoard`, с помощью которого выведены точности всех моделей. Полученный график представлен на рис. 11.

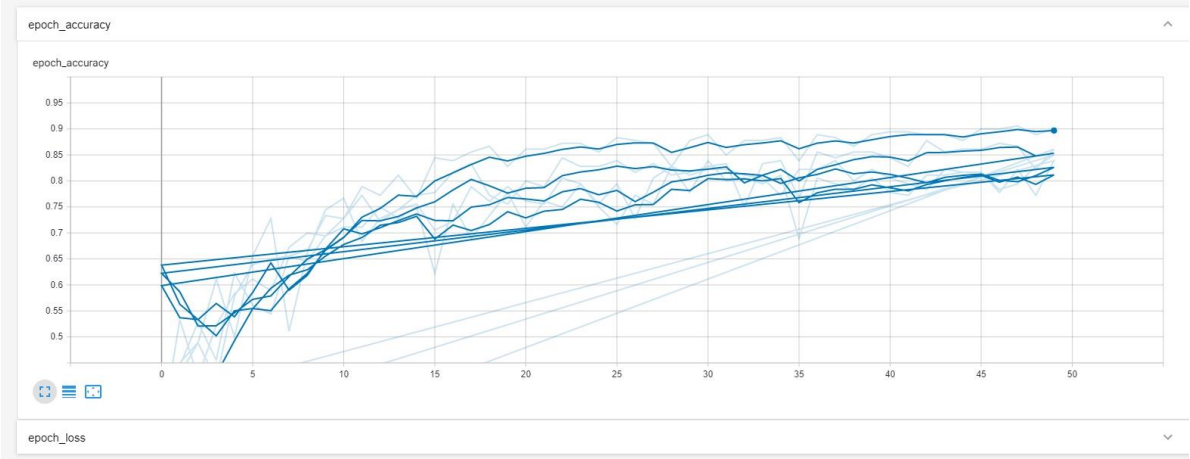

Рисунок 14 - Точности моделей

Так как значения точностей при обучении различных моделей записываются в массив последовательно, то по графику можно увидеть, что точность каждой следующей модели лучше точности предыдущей.

5. Тестирование лучшей сети.

Лучшая сеть была получена после обучения модели с последней архитектурой. Для этого лучшая модель с лучшей архитектурой была обучена на тысяче эпох. Коллбэки позволили выявить модель с наивысшей точностью. Точность при обучении представлена на рис. 12.

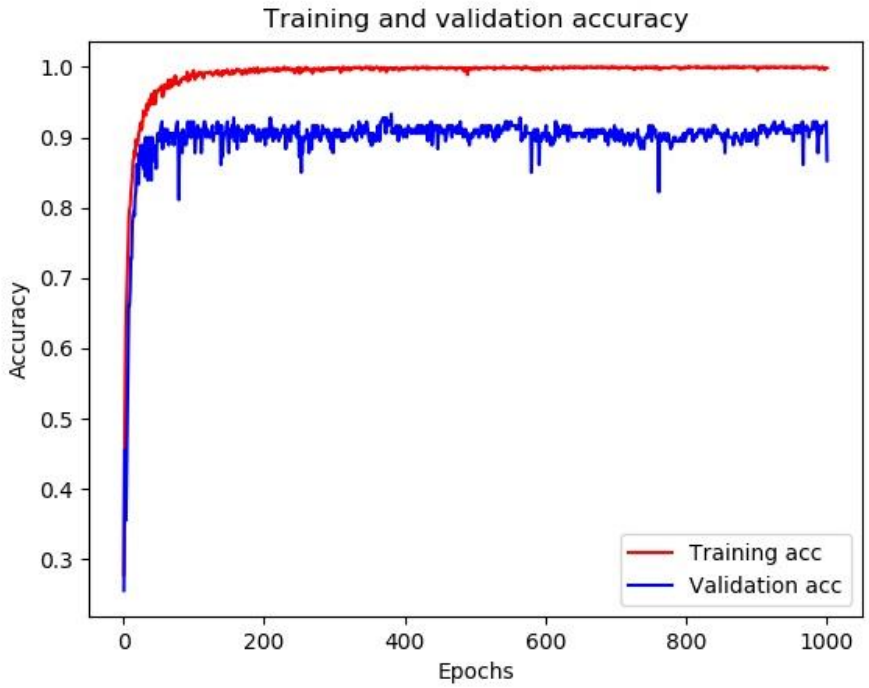

Рисунок 15 - Обучение лучшей сети

Лучшие результаты показывает модель на 381 эпохе. Её лучшая точность равна 93.33%.

Тестирование проходило с помощью файла runner_on_best_model.py. Результаты тестирования представлены на рис. 13.

	precision	recall	f1-score	support
0	0.95	1.00	0.97	19
1	0.82	0.82	0.82	17
2	0.94	1.00	0.97	17
3	1.00	0.92	0.96	25
4	1.00	1.00	1.00	20
5	1.00	0.90	0.95	21
6	0.95	0.95	0.95	22
7	0.95	1.00	0.98	21
8	0.94	0.89	0.92	19
9	0.86	0.95	0.90	19
micro avg	0.94	0.94	0.94	200
macro avg	0.94	0.94	0.94	200
weighted avg	0.95	0.94	0.95	200
samples avg	0.94	0.94	0.94	200

Рисунок 16 - Тестирование сети

Разделение ролей.

- Александров Руслан – разработка и улучшение архитектуры, разработка коллбэков;
- Зуев Даниил – разработка и улучшение архитектуры, преобразование данных;
- Рудоман Вадим – разработка и улучшение архитектуры, тестирование архитектуры.

Выводы.

В ходе выполнения данного индивидуального задания была решена задача предсказания цифры по аудиофайлу. Реализовано представление аудиофайла в формате, пригодном для обучения нейросети. Была найдена оптимальная

архитектура сети, позволяющая достаточно точно определять цифру по аудиофайлу. Были разработаны собственные коллбэки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. C. Ye, Y. Yang, C. Fermuller, and Y. Aloimonos On the Importance of Consistency in Training Deep Neural Networks // arXiv:1708.00631, 2017.
2. Preventing Deep Neural Network from Overfitting // Towards Data Science. URL: https://towardsdatascience.com/preventing-deep-neural-network-from-overfitting-953458db800a (Дата обращения: 25.05.2020)
3. Шолле Франсуа Глубокое обучение на Python. СПб.: Питер, 2018. 400 с.

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

main.py

```
from src import converter_audio
from src.email_sender import sent_email_notification
from tensorflow.keras import callbacks
import tensorflow as tf
import os
import matplotlib.pyplot as plt
import datetime
from sklearn.metrics import classification_report
from tensorflow.keras.utils import to_categorical

# set gpu visible to tf
os.environ['CUDA_VISIBLE_DEVICES'] = "0"

model_number = 1
epochs = 50


class EmailNotificationCallback(callbacks.Callback):
    def __init__(self, logs=None):
        self.time_now = 0
        self.epoch_counter = 0
        self.epoch_accuracy = 0
        self.best_epoch_accuracy = 0
        self.best_epoch = 0

    def on_train_begin(self, logs=None):
        self.time_now = datetime.datetime.now()

    def on_epoch_begin(self, epoch, logs=None):
        self.epoch_counter += 1

    def on_epoch_end(self, epoch, logs=None):
        self.epoch_accuracy = logs['val_accuracy']
        if self.best_epoch_accuracy == 0:
            self.best_epoch_accuracy = self.epoch_accuracy

        if self.epoch_accuracy > self.best_epoch_accuracy:
            self.best_epoch_accuracy = self.epoch_accuracy
            self.best_epoch = self.epoch_counter
```

```
    # after model training set best accuracy with epoch to email
    def on_train_end(self, logs=None):
        try:
            to = 'alex.raswqa@gmail.com'
            val_accuracy = round(self.best_epoch_accuracy * 100, 2)
            time_to_train = datetime.datetime.now() - self.time_now
            message = 'Model #' + str(model_number) + ' has just finished
\nIt took ' + str(
                time_to_train) + ' minutes' + '\nBest validation accuracy
equals ' + str(val_accuracy) + '% on ' + str(
                self.best_epoch) + ' epoch'
            print(message)
            sent_email_notification(to, message)
        except Exception as inst:
            print(type(inst))
            print(inst.args)
            print(inst)


# gpu limit
def set_gpu_limit_memory():
    gpus = tf.config.experimental.list_physical_devices('GPU')
    if gpus:
        try:
            for gpu in gpus:
                tf.config.experimental.set_memory_growth(gpu, True)
        except RuntimeError as e:
            print(e)


def create_plot(history_dict):
    loss_values = history_dict['loss']
    val_loss_values = history_dict['val_loss']
    acc = history_dict['accuracy']
    val_acc = history_dict['val_accuracy']

    plt.plot(epochs, loss_values, 'r', label='Training loss')
    plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.savefig("loss" + str(model_number) + ".png")
    plt.clf()
```

```
    plt.plot(epochs, acc, 'r', label='Training acc')
    plt.plot(epochs, val_acc, 'b', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.savefig("accuracy" + str(model_number) + ".png")


def get_callbacks():
    # tensorboard view
    tensorboard         =         callbacks.TensorBoard(log_dir='./logs',
histogram_freq=1,
                                          write_graph=True,
write_images=True, profile_batch=100000000)

    # save best model by accuracy
    checkpoint  =  callbacks.ModelCheckpoint('best_model.h5',  verbose=1,
monitor='val_accuracy',
                                             save_best_only=True,
mode='auto')

    return [tensorboard, EmailNotificationCallback(), checkpoint]


def test_this_model(model, x_test, y_test):
    predictions = model.predict_classes(x_test)
    print(classification_report(y_test, to_categorical(predictions)))


try:
    set_gpu_limit_memory()

    # use gpu
    with tf.device('/gpu:0'):
        train_callbacks = get_callbacks()

        X_train,     X_test,     y_train,     y_test,     models     =
converter_audio.prepared_data_and_get_models()

        for model in models:
            try:
                print('try ' + str(model_number))
```

```
                history = model.fit(X_train, y_train, batch_size=64,
epochs=epochs, verbose=1, validation_split=0.1,
                                    callbacks=train_callbacks)
                create_plot(history.history)
                test_this_model(model, X_test, y_test)
                model_number += 1
            except Exception as inst:
                print(type(inst))
                print(inst.args)
                print(inst)
except RuntimeError as e:
    print(e)
```

converter_audio.py

```
import numpy as np
import librosa
from librosa import display
import os
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
from src import converter_audio, model_creator
from sklearn.model_selection import train_test_split

def draw_spectrogram(m_slaney, label):
    plt.figure(figsize=(10, 4))
    plt.subplot(2, 1, 1)
    display.specshow(m_slaney, x_axis='time')
    plt.colorbar()
    plt.title('Number ' + label)
    plt.tight_layout()
    plt.savefig(label + ".png")
    plt.show()

def wav2mfcc(file_path, label, max_pad_len=20):
    wave, _ = librosa.load(file_path, mono=True, sr=None)
    wave = wave[::3]
    # Compute MFCC features from the raw signal
    # sr = 8000 ~ 8 kHz as default for this dataset
    mfcc = librosa.feature.mfcc(wave, sr=8000)

    # draw spectrogram
    #     showPictures(mfcc, label)
    pad_width = max_pad_len - mfcc.shape[1]
    mfcc = np.pad(mfcc, pad_width=((0, 0), (0, pad_width)),
mode='constant')
```

```
    return mfcc

# get all files from "recordings" folder
def get_sound_tensors_with_labels():
    labels = []
    mfccs = []

    for f in os.listdir('./recordings'):
        if f.endswith('.wav'):
            # parse label
            label = f.split('_')[0]

            mfccs.append(wav2mfcc('./recordings/' + f, label))

            labels.append(label)

    return np.asarray(mfccs), to_categorical(labels)

# get prepared data and 8 models to train
def prepared_data_and_get_models():
    mfccs, labels = get_sound_tensors_with_labels()
    dim_1 = mfccs.shape[1]
    dim_2 = mfccs.shape[2]
    channels = 1

    mfccs_copy = mfccs
    X = mfccs_copy.reshape((mfccs.shape[0], dim_1, dim_2, channels))
    y = labels

    input_shape = (dim_1, dim_2, channels)

    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1, random_state=1)
    models = [
        model_creator.get_cnn_model_1(input_shape),
        model_creator.get_cnn_model_2(input_shape),
        model_creator.get_cnn_model_3(input_shape),
        model_creator.get_cnn_model_4(input_shape),
        model_creator.get_cnn_model_5(input_shape),
        model_creator.get_cnn_model_6(input_shape),
        model_creator.get_cnn_model_7(input_shape),
        model_creator.get_cnn_model_8(input_shape)
              ]
```

```
    return X_train, X_test, y_train, y_test, models
```

email_sender.py

```
import os
import smtplib

def sent_email_notification(to, message):
    login = 'alex.raswqa88@outlook.com'
    password = os.environ['Password']

    server = smtplib.SMTP('smtp.office365.com:587')
    server.ehlo()
    server.starttls()
    server.login(login, password)

    message = 'Subject: {}\n\n{}'.format('New data', message)
    server.sendmail(login, to, message)
    server.close()
```

model_creator.py

```
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D,
MaxPooling2D, BatchNormalization
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import SGD

# 10 digits
num_classes = 10


def get_cnn_model_1(input_shape):
    model = Sequential()
    model.add(Conv2D(32,    kernel_size=(2,    2),    activation='relu',
input_shape=input_shape))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())

    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(num_classes, activation='softmax'))
    model.compile(loss=categorical_crossentropy,  optimizer=SGD(lr=0.01),
metrics=['accuracy'])
```

```
    return model


def get_cnn_model_2(input_shape):
    model = Sequential()

    model.add(Conv2D(32,    kernel_size=(2,    2),    activation='relu',
input_shape=input_shape))

    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())

    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss=categorical_crossentropy,  optimizer=SGD(lr=0.01),
metrics=['accuracy'])

    return model


def get_cnn_model_3(input_shape):
    model = Sequential()

    model.add(Conv2D(32,    kernel_size=(2,    2),    activation='relu',
input_shape=input_shape))
    model.add(BatchNormalization())

    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())

    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss=categorical_crossentropy,  optimizer=SGD(lr=0.01),
metrics=['accuracy'])

    return model
```

```
def get_cnn_model_4(input_shape):
    model = Sequential()

    model.add(Conv2D(32,    kernel_size=(2,    2),    activation='relu',
input_shape=input_shape))
    model.add(BatchNormalization())

    model.add(Conv2D(32, kernel_size=(2, 2), activation='relu'))
    model.add(BatchNormalization())

    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())

    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss=categorical_crossentropy,  optimizer=SGD(lr=0.01),
metrics=['accuracy'])

    return model


def get_cnn_model_5(input_shape):
    model = Sequential()

    model.add(Conv2D(32,    kernel_size=(2,    2),    activation='relu',
input_shape=input_shape))
    model.add(BatchNormalization())

    model.add(Conv2D(48, kernel_size=(2, 2), activation='relu'))
    model.add(BatchNormalization())

    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())

    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.4))
```

```
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss=categorical_crossentropy,  optimizer=SGD(lr=0.01),
metrics=['accuracy'])

    return model


def get_cnn_model_6(input_shape):
    model = Sequential()

    model.add(Conv2D(32,    kernel_size=(2,    2),    activation='relu',
input_shape=input_shape))
    model.add(BatchNormalization())

    model.add(Conv2D(48, kernel_size=(2, 2), activation='relu'))
    model.add(BatchNormalization())

    model.add(Conv2D(64, kernel_size=(2, 2), activation='relu'))
    model.add(BatchNormalization())

    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())

    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss=categorical_crossentropy,  optimizer=SGD(lr=0.01),
metrics=['accuracy'])

    return model


def get_cnn_model_7(input_shape):
    model = Sequential()

    model.add(Conv2D(32,    kernel_size=(2,    2),    activation='relu',
input_shape=input_shape))
    model.add(BatchNormalization())

    model.add(Conv2D(48, kernel_size=(2, 2), activation='relu'))
```

```
    model.add(BatchNormalization())

    model.add(Conv2D(120, kernel_size=(2, 2), activation='relu'))
    model.add(BatchNormalization())

    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())

    model.add(Dense(128, activation='relu'))
    model.add(BatchNormalization())

    model.add(Dense(64, activation='relu'))
    model.add(BatchNormalization())

    model.add(Dropout(0.4))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss=categorical_crossentropy,  optimizer=SGD(lr=0.01),
metrics=['accuracy'])

    return model


def get_cnn_model_8(input_shape):
    model = Sequential()

    model.add(Conv2D(32,    kernel_size=(2,    2),    activation='relu',
input_shape=input_shape))
    model.add(BatchNormalization())

    model.add(Conv2D(48, kernel_size=(2, 2), activation='relu'))
    model.add(BatchNormalization())

    model.add(Conv2D(120, kernel_size=(2, 2), activation='relu'))
    model.add(BatchNormalization())

    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.3))

    model.add(Flatten())

    model.add(Dense(128, activation='relu'))
```

```
    model.add(BatchNormalization())
    model.add(Dropout(0.35))
    model.add(Dense(64, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.4))
    model.add(Dense(num_classes, activation='softmax'))
    model.compile(loss=categorical_crossentropy,  optimizer=SGD(lr=0.01),
metrics=['accuracy'])

    return model
```

runner_on_best_model.py

```
from tensorflow import keras
from sklearn.metrics import classification_report
from tensorflow.keras.utils import to_categorical
from src import converter_audio

_, X_test, _, y_test, _ = converter_audio.prepared_data_and_get_models()

trained_model = keras.models.load_model('./best_model.h5')
predictions = trained_model.predict_classes(X_test)

print(classification_report(y_test, to_categorical(predictions)))
```