

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии  
Департамент цифровых, робототехнических систем и электроники

**ОТЧЕТ**  
**По лабораторной работе №8**  
**Дисциплины «Основы нейронных сетей»**

Выполнил:

Говоров Егор Юрьевич

3 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и  
вычислительная техника (профиль)  
«Программное обеспечение средств  
вычислительной  
техники и автоматизированных  
систем», очная форма обучения

---

(подпись)

Проверил:

Воронкин Р. А., доцент департамента  
цифровых и робототехнических  
систем и электроники и института  
перспективной инженерии

---

(подпись)

Ставрополь, 2025 г.

**Тема:** Обработка аудиосигналов с помощью нейронных сетей.

**Цель:** приобретение базовых навыков для обработки аудиосигналов с помощью нейронных сетей.

**Ссылка:** [https://github.com/Artorias1469/NN\\_8.git](https://github.com/Artorias1469/NN_8.git)

### Ход работы:

#### Выполнение индивидуальных заданий:

##### Задание 1. ДЗ\_Lite.

**Условие:** необходимо запустить раздел "Подготовка" и приступить к выполнению заданий.

Для выполнения данного задания вначале запусти раздел "Подготовка". Здесь происходит импорт необходимых библиотек для дальнейшей работы

```
# Работа с массивами
import math
import os
import pickle
import random
import time
import warnings
import zipfile

import matplotlib.pyplot as plt # Отрисовка графиков
import librosa # Параметризация аудио
import gdown # Загрузка из google облака
import numpy as np # Работа с массивами

# Конструирование и загрузка моделей нейронных сетей
from tensorflow.keras.layers import (
    BatchNormalization, Conv1D, Conv2D, Dense,
    Dropout, Flatten, Input, LSTM, concatenate
)
from tensorflow.keras.models import Model, Sequential, load_model
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.utils import to_categorical

# Метрики и предобработка sklearn
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Магия для отрисовки в Jupyter/Colab
%matplotlib inline

# Отключение предупреждений
warnings.filterwarnings('ignore')
```

Рисунок 1. Импорт библиотек

Далее выполним загрузку датасета и подготовленных данных

```

# Загрузка архива с датасетом, если он ещё не загружен
if not os.path.exists('genres.zip'):
    gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/l12/genres.zip', None, quiet=True)

# Загрузка подготовленных данных, если они ещё не загружены
if not os.path.exists('audio_data_mean.pickle'):
    gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/l12/audio_data_mean.pickle', None, quiet=True)

# Распаковка архива
with zipfile.ZipFile('genres.zip', 'r') as zip_ref:
    zip_ref.extractall('genres')

# Просмотр жанров (папок в genres)
print("Жанры в датасете:")
print("\t".join(sorted(os.listdir('genres'))))

# Просмотр всех файлов в папке blues
blues_path = os.path.join('genres', 'blues')
if os.path.exists(blues_path):
    print("\nФайлы в папке 'genres/blues':")
    files = sorted(os.listdir(blues_path))
    for i in range(0, len(files), 5):
        print("\t".join(files[i:i+5]))
else:
    print("Папка 'genres/blues' не найдена.")

```

Рисунок 2. Загрузка данных

Затем выполняем распаковку архива на локальный диск, а также просмотр выгруженных папок и содержимое одной из них

```

Жанры в датасете:
blues  classical  country disco  genres  hip-hop  jazz  metal  pop  reggae  rock

Файлы в папке 'genres/blues':
blues.00000.au  blues.00001.au  blues.00002.au  blues.00003.au  blues.00004.au
blues.00005.au  blues.00006.au  blues.00007.au  blues.00008.au  blues.00009.au
blues.00010.au  blues.00011.au  blues.00012.au  blues.00013.au  blues.00014.au
blues.00015.au  blues.00016.au  blues.00017.au  blues.00018.au  blues.00019.au
blues.00020.au  blues.00021.au  blues.00022.au  blues.00023.au  blues.00024.au
blues.00025.au  blues.00026.au  blues.00027.au  blues.00028.au  blues.00029.au
blues.00030.au  blues.00031.au  blues.00032.au  blues.00033.au  blues.00034.au
blues.00035.au  blues.00036.au  blues.00037.au  blues.00038.au  blues.00039.au
blues.00040.au  blues.00041.au  blues.00042.au  blues.00043.au  blues.00044.au
blues.00045.au  blues.00046.au  blues.00047.au  blues.00048.au  blues.00049.au
blues.00050.au  blues.00051.au  blues.00052.au  blues.00053.au  blues.00054.au
blues.00055.au  blues.00056.au  blues.00057.au  blues.00058.au  blues.00059.au
blues.00060.au  blues.00061.au  blues.00062.au  blues.00063.au  blues.00064.au
blues.00065.au  blues.00066.au  blues.00067.au  blues.00068.au  blues.00069.au
blues.00070.au  blues.00071.au  blues.00072.au  blues.00073.au  blues.00074.au
blues.00075.au  blues.00076.au  blues.00077.au  blues.00078.au  blues.00079.au
blues.00080.au  blues.00081.au  blues.00082.au  blues.00083.au  blues.00084.au
blues.00085.au  blues.00086.au  blues.00087.au  blues.00088.au  blues.00089.au
blues.00090.au  blues.00091.au  blues.00092.au  blues.00093.au  blues.00094.au
blues.00095.au  blues.00096.au  blues.00097.au  blues.00098.au  blues.00099.au

```

Рисунок 3. Проверка содержимого одной из папок

Далее выполняется установка констант и вывод списка классов

```

FILE_DIR = './genres' # Папка с файлами датасета
CLASS_LIST = os.listdir(FILE_DIR) # Список классов, порядок меток не определен!
CLASS_LIST.sort() # Сортировка списка классов для фиксации порядка меток
CLASS_COUNT = len(CLASS_LIST) # Количество классов
CLASS_FILES = 100 # Общее количество файлов в каждом классе
FILE_INDEX_TRAIN_SPLIT = 90 # Количество файлов каждого класса на основной набор
VALIDATION_SPLIT = 0.1 # Доля проверочной выборки в основном наборе
DURATION_SEC = 30 # Анализируемая длительность аудиосигнала
N_FFT = 8192 # Размер окна преобразования Фурье для расчета спектра
HOP_LENGTH = 512 # Объем данных для расчета одного набора признаков

```

✓ проверка списка классов

Команда `print(CLASS_LIST)` позволяет быстро убедиться, какие именно классы присутствуют в датасете, и проверить корректность загрузки и сортировки списка.

```

# Проверка списка классов
print(CLASS_LIST)

['blues', 'classical', 'country', 'disco', 'hip-hop', 'jazz', 'metal', 'pop', 'reggae', 'rock']

```

Рисунок 4. Установка констант

Затем выполняется функция параметризации аудио

```
# Функция параметризации аудио

def get_features(y,                # волновое представление сигнала
                sr,                # частота дискретизации сигнала y
                n_fft=N_FFT,       # размер скользящего окна БПФ
                hop_length=HOP_LENGTH # шаг скользящего окна БПФ
                ):
    # Вычисление различных параметров (признаков) аудио

    # Хромограмма
    chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr, n_fft=n_fft, hop_length=hop_length)
    # Мел-кейстральные коэффициенты
    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_fft=n_fft, hop_length=hop_length)
    # Среднеквадратическая амплитуда
    rmse = librosa.feature.rms(y=y, hop_length=hop_length)
    # Спектральный центроид
    spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr, n_fft=n_fft, hop_length=hop_length)
    # Ширина полосы частот
    spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr, n_fft=n_fft, hop_length=hop_length)
    # Спектральный спад частоты
    rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr, n_fft=n_fft, hop_length=hop_length)
    # Пересечения нуля
    zcr = librosa.feature.zero_crossing_rate(y, hop_length=hop_length)

    # Сборка параметров в общий список:
    # На один файл один усредненный вектор признаков
    features = {'rmse': rmse.mean(axis=1, keepdims=True),
               'spct': spec_cent.mean(axis=1, keepdims=True),
               'spbw': spec_bw.mean(axis=1, keepdims=True),
               'roff': rolloff.mean(axis=1, keepdims=True),
               'zcr': zcr.mean(axis=1, keepdims=True),
               'mfcc': mfcc.mean(axis=1, keepdims=True),
               'stft': chroma_stft.mean(axis=1, keepdims=True)}

    return features
```

Рисунок 5. Функция параметризации аудио

Далее напишем функцию объединения признаков в набор векторов

```
[ ] # Функция объединения признаков в набор векторов

def stack_features(feats # словарь признаков, отдельные векторы по ключу каждого признака
                  ):
    features = None
    for v in feats.values():
        features = np.vstack((features, v)) if features is not None else v

    return features.T
```

Рисунок 6. Функция объединения признаков в набор векторов Потом выполним функцию формирования набора признаков и метки класса для аудиофайла

```
def get_feature_list_from_file(class_index, # индекс класса файла song_name
                              song_name,   # имя аудиофайла
                              duration_sec # длительность аудио в секундах
                              ):
    # Загрузка в y первых duration_sec секунд аудиосигнала
    y, sr = librosa.load(song_name, mono=True, duration=duration_sec)

    # Извлечение параметров из аудиосигнала
    features = get_features(y, sr)
    feature_set = stack_features(features)

    # Перевод номера класса в one hot encoding
    y_label = to_categorical(class_index, CLASS_COUNT)

    return feature_set, y_label
```

Рисунок 7. Функция формирования набора признаков для аудиофайла

Затем напишем функцию формирования подвыборки признаков и меток класса для одного файла

```

# Функция формирования подвыборки признаков и меток класса для одного файла

def process_file(class_index, # индекс класса аудиофайла
                 file_index,  # индекс (порядковый номер) аудиофайла в папке класса
                 duration_sec # длительность аудио в секундах
                 ):
    x_list = []
    y_list = []
    class_name = CLASS_LIST[class_index]

    # Извлечение имени произведения
    song_name = f'{FILE_DIR}/{class_name}/{class_name}.{str(file_index).zfill(5)}.au'

    # Выборка признаков и метки класса для произведения
    feature_set, y_label = get_feature_list_from_file(class_index,
                                                       song_name,
                                                       duration_sec)

    # Добавление данных в наборы
    for j in range(feature_set.shape[0]):
        x_list.append(feature_set[j])
        y_list.append(y_label)

    # Возврат имени файла и numpy-массивов признаков и меток класса
    return song_name, \
           np.array(x_list).astype('float32'), \
           np.array(y_list).astype('float32')

```

Рисунок 8. Функция формирования подвыборки признаков для одного файла

Далее выполним последнюю функцию формирования набора данных из файлов всех классов по диапазону номеров файлов

```

# Функция формирования набора данных из файлов всех классов по диапазону номеров файлов

def extract_data(file_index_start, # начальный индекс аудиофайла
                 file_index_end,   # конечный индекс аудиофайла (не достигая)
                 duration_sec=DURATION_SEC # длительность аудио в секундах
                 ):
    # Списки для последовательностей входных данных и меток класса
    x_data = None
    y_data = None

    # Фиксация времени старта формирования выборки
    curr_time = time.time()

    # Для всех классов:
    for class_index in range(len(CLASS_LIST)):
        # Для всех файлов текущего класса из заданного диапазона номеров:
        for file_index in range(file_index_start, file_index_end):
            # Обработка одного файла и добавление данных к общим массивам
            _, file_x_data, file_y_data = process_file(class_index, file_index, duration_sec)
            x_data = file_x_data if x_data is None else np.vstack([x_data, file_x_data])
            y_data = file_y_data if y_data is None else np.vstack([y_data, file_y_data])

        # Вывод информации о готовности обработки датасета
        print(f'Жанр {CLASS_LIST[class_index]} готов -> {round(time.time() - curr_time)} c')
        curr_time = time.time()

    # Возврат массивов набора данных
    return x_data, y_data

```

Рисунок 9. Функция формирования набора данных по диапазону номеров файлов

Затем выполняется восстановление датасета аудио

```

# Данные привязаны к порядку следования меток классов!
# Порядок классов фиксирован сортировкой списка меток классов

with open('/content/audio_data_mean.pickle', 'rb') as f:
    x_train_data, y_train_data = pickle.load(f)

```

Рисунок 10. Восстановление датасета аудио

Далее реализуется нормирование признаков в соответствии со

стандартным нормальным распределением и разделение набора данных на обучающую и проверочную выборки

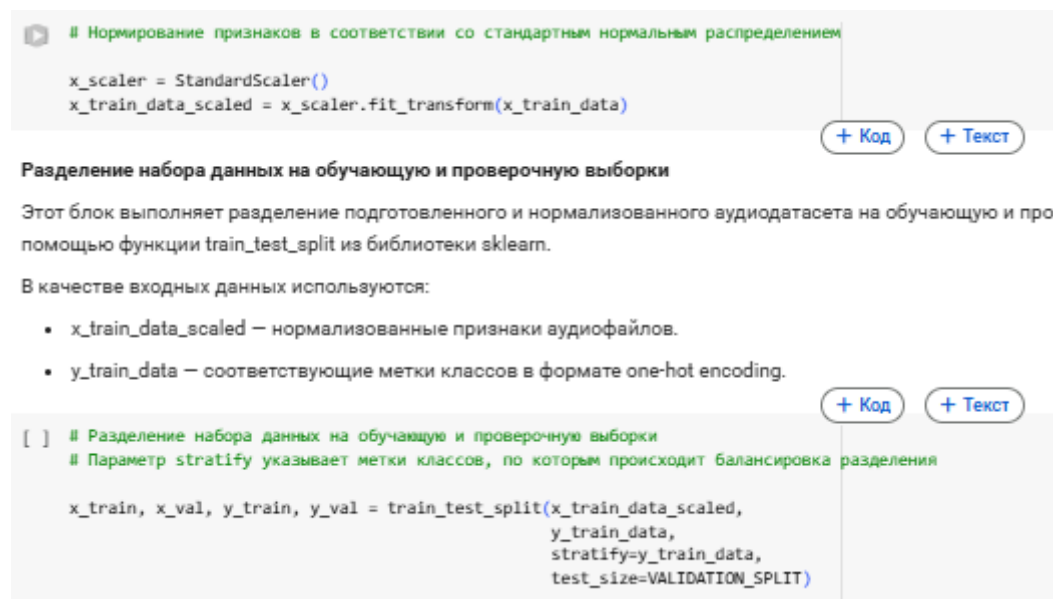


Рисунок 11. Разделение набора данных на обучающую и проверочную выборки

Затем выполним функцию вывода графиков точности и ошибки распознавания на обучающей и проверочной выборках

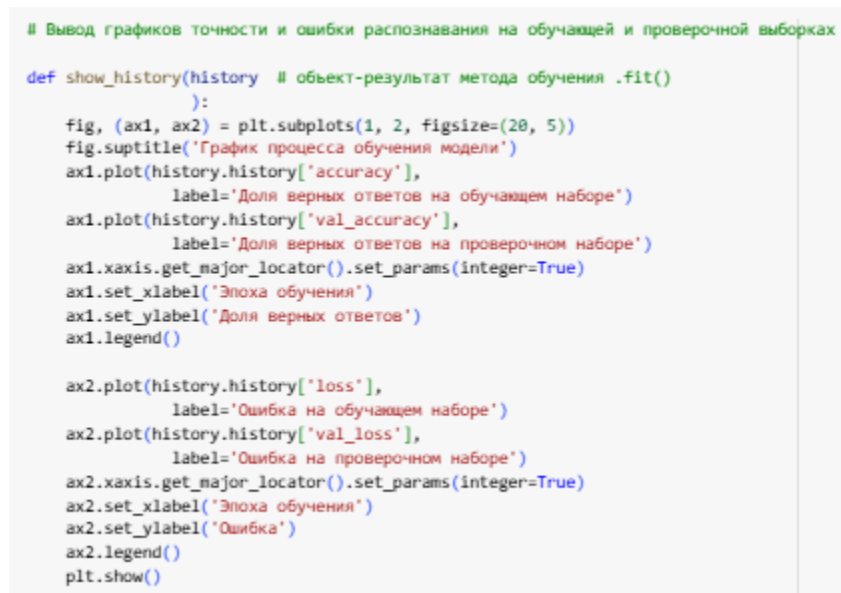


Рисунок 12. Функция вывода графиков точности и ошибки

Далее напомним функцию для классификации звукового файла и визуализации предсказания модели для него

```

# Классификация файла и визуализация предсказания модели для него

def classify_file(model,          # обученная модель классификатора
                 x_scaler,      # настроенный нормировщик входных данных
                 class_index,   # верный индекс класса аудиофайла
                 file_index     # индекс (порядковый номер) аудиофайла в папке
                 ):
    # Подготовка выборки данных файла произведения
    song_name, file_x_data, file_y_data = process_file(class_index, file_index, DURATION_SEC)

    # Нормирование признаков уже настроенным нормировщиком
    file_x_data = x_scaler.transform(file_x_data)

    print('Файл:', song_name)
    print('Векторы для предсказания:', file_x_data.shape)

    # Вычисление предсказания по выборке
    predict = model.predict(file_x_data)
    # Определение среднего предсказания (голосование)
    predict_mean = predict.mean(axis=0)
    # Определение индекса класса по результату голосования
    predict_class_index = np.argmax(predict_mean)
    # Вычисление признака правильного предсказания
    predict_good = predict_class_index == class_index

    # Визуализация предсказания сети для файла
    plt.figure(figsize=(10,3))
    print('Классификация сети:', CLASS_LIST[predict_class_index], '- ', 'ВЕРНО :-)' if predict_good else 'НЕВЕРНО.')
    plt.title('Среднее распределение векторов предсказаний')
    plt.bar(CLASS_LIST, predict_mean, color='g' if predict_good else 'r')
    plt.show()
    print('-----')

    # Возврат результата предсказания
    return predict_class_index

```

Рисунок 13. Функция классификации файла и визуализация предсказания модели

Также напишем функцию классификации и визуализации для нескольких файлов каждого класса

```

# Классификация и визуализация нескольких файлов каждого класса

def classify_test_files(model,      # обученная модель классификатора
                      x_scaler,   # настроенный нормировщик входных данных
                      from_index, # индекс аудиофайла, с которого начинать визуализацию
                      n_files):   # количество файлов для визуализации

    predict_all = 0
    predict_good = 0
    y_true = []
    y_pred = []

    # Классификация каждого файла и аккумуляция результатов классификации
    for class_index in range(CLASS_COUNT):
        for file_index in range(from_index, from_index + n_files):
            predict_class_index = classify_file(model, x_scaler, class_index, file_index)
            y_true.append(class_index)
            y_pred.append(predict_class_index)
            predict_all += 1
            predict_good += (predict_class_index == class_index)

    # Расчет и вывод итогов классификации
    good_ratio = round(predict_good / predict_all * 100., 2)
    print(f'=== Обработано образцов: {predict_all}, из них распознано верно: {predict_good}, доля верных: {good_ratio}% ===')

    # Построение матрицы ошибок без нормализации, покажет попадания в штуках
    cm = confusion_matrix(y_true, y_pred)

    # Отрисовка матрицы ошибок
    fig, ax = plt.subplots(figsize=(10, 10))
    ax.set_title('Матрица ошибок по файлам аудио (не нормализованная)')
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=CLASS_LIST)
    disp.plot(ax=ax)
    plt.show()

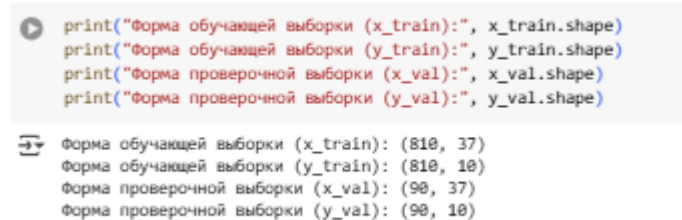
```

Рисунок 14. Функция классификации и визуализации нескольких файлов каждого класса



Задание 1. Видно, что в предыдущих ячейках были подготовлены все данные для обучения модели нейронной сети.

Необходимо проверить форму данных обучающей и проверочной выборок, то есть вывести ее на экран



```
print("Форма обучающей выборки (x_train):", x_train.shape)
print("Форма обучающей выборки (y_train):", y_train.shape)
print("Форма проверочной выборки (x_val):", x_val.shape)
print("Форма проверочной выборки (y_val):", y_val.shape)
```

Форма обучающей выборки (x\_train): (810, 37)  
Форма обучающей выборки (y\_train): (810, 10)  
Форма проверочной выборки (x\_val): (90, 37)  
Форма проверочной выборки (y\_val): (90, 10)

Рисунок 15. Вывод форм обучающей и проверочной выборок

Задание 2. Необходимо составить модель классификатора на полносвязных слоях и сохранить ее в переменной `model` Для этого нужно:

- Использовать заготовку для последовательной модели `Sequential`;
- Добавить полносвязный слой на 64 нейрона с активационной функцией `'relu'`, после него добавить слой `Dropout` с долей отключаемых нейронов 30%;
- Добавить следующий полносвязный слой на 32 нейрона с активационной функцией `'relu'`, после него добавить слой `Dropout` с долей отключаемых нейронов 30%;
- Добавить следующий полносвязный слой на 16 нейронов с активационной функцией `'relu'`, после него добавить слой `Dropout` с долей отключаемых нейронов 20%;
- Добавить слой пакетной нормализации;
- Добавить финальный полносвязный слой классификатора на число нейронов по числу классов (`CLASS_COUNT`) с активационной функцией `'softmax'`.



```

model = Sequential([
    # Первый полносвязный слой с ReLU-активацией
    Dense(64, activation='relu', input_shape=(x_train.shape[1],)),
    Dropout(0.3), # Dropout для регуляризации

    # Второй скрытый слой
    Dense(32, activation='relu'),
    Dropout(0.3), # Ещё один Dropout

    # Третий скрытый слой
    Dense(16, activation='relu'),
    Dropout(0.2), # Уменьшенный Dropout

    # Нормализация активаций (ускоряет и стабилизирует обучение)
    BatchNormalization(),

    # Выходной слой с softmax для многоклассовой классификации
    Dense(CLASS_COUNT, activation='softmax')
])

```

Рисунок 16. Создание модели классификатора

Задание 3. Необходимо откомпилировать созданную модель методом `.compile()` с указанием оптимизатора Adam и начальным шагом обучения 0.0001, функцией ошибки 'categorical\_crossentropy' и метрикой 'accuracy'. Необходимо вывести на экран сводку архитектуры полученной модели методом `.summary()`

```

model.compile(
    optimizer=Adam(learning_rate=0.0001), # Оптимизатор Adam с небольшой скоростью обучения
    loss='categorical_crossentropy',       # Функция потерь для многоклассовой классификации
    metrics=['accuracy']                  # Метрика точности для оценки качества модели
)

model.summary() # Выводит архитектуру модели и количество параметров

```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 64)	2,432
dropout_3 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 32)	2,080
dropout_4 (Dropout)	(None, 32)	0
dense_6 (Dense)	(None, 16)	528
dropout_5 (Dropout)	(None, 16)	0
batch_normalization_1 (BatchNormalization)	(None, 16)	64
dense_7 (Dense)	(None, 10)	170

Total params: 5,274 (20.60 KB)  
 Trainable params: 5,242 (20.48 KB)  
 Non-trainable params: 32 (128.00 B)

Рисунок 17. Компиляция созданной модели

Задание 4. Необходимо обучить модель и вывести графики обучения:

- Обучить созданную и откомпилированную модель классификатора на данных обучающей выборки `x_train`, `y_train`, используя проверочные данные `x_val`, `y_val`, размер батча 32 и количество эпох 1000. Результаты обучения сохранить в переменной `history`
- В разделе "Функция вывода графиков точности и ошибки по эпохам обучения" найти определение функции `show_history()`, изучить требуемые параметры для нее и использовать для построения графиков точности и ошибки на протяжении эпох обучения.

```
history = model.fit(
    x_train, y_train,          # Обучающие данные и метки
    validation_data=(x_val, y_val), # Валидационные данные для оценки после каждой эпохи
    batch_size=32,           # Размер батча – количество примеров на одну итерацию обучения
    epochs=1000,             # Количество проходов по всему набору данных
    verbose=1                # Вывод прогресса обучения в консоль
)

show_history(history) # Визуализация процесса обучения
```

Рисунок 18. Обучение модели

Далее посмотрим на результат обучения и на построенные графики процесса обучения

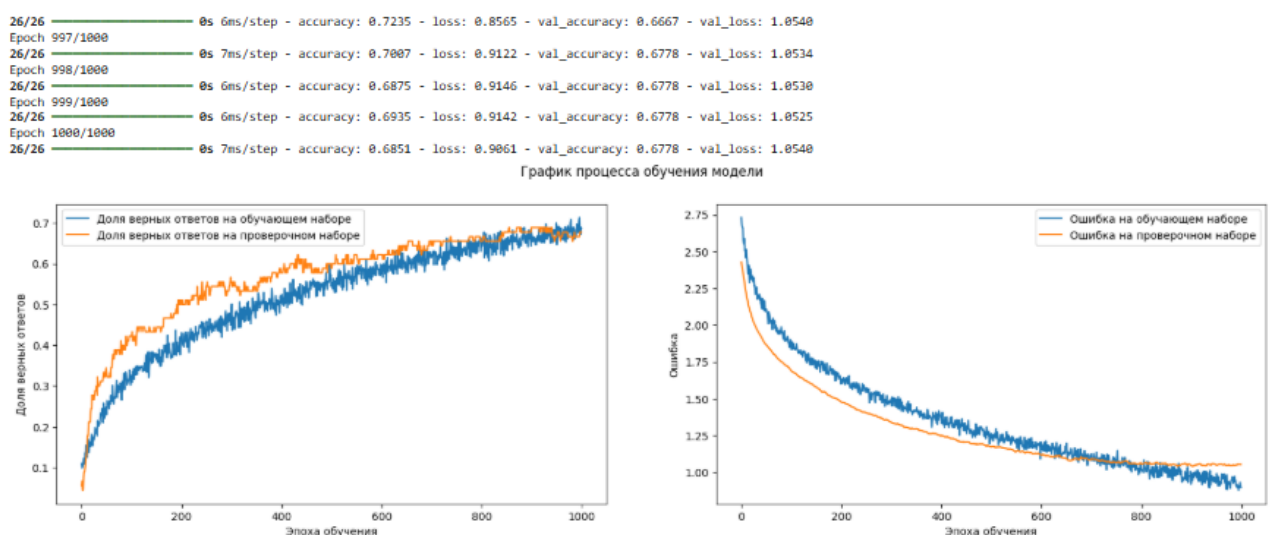


Рисунок 19. Графики процесса обучения модели

Задание 5. Необходимо проверить работу модели, для этого:

В разделе Функции визуализации распознавания отдельных звуковых файлов необходимо найти определение функции `classify_test_files()` и изучить ее параметры. Использовать функцию для визуализации работы

классификатора на произвольном количестве тестовых звуковых файлов, полагая, что:

- используется обученная в задании 4 модель классификатора аудио;
- нормализатор `x_scaler` уже настроен ранее в ноутбуке, и его нужно передать в функцию `classify_test_files()` вместо параметра `x_scaler`;
- тестовые звуковые файлы начинаются с индекса 90 и всего их ровно 10 для каждого класса.

```
classify_test_files(model, x_scaler, from_index=90, n_files=10)
```

Рисунок 20. Команда для вывода визуализации работы классификатора

После посмотрим на графики среднего распределения векторов предсказания , все полученные графики можно посмотреть в репозитории.

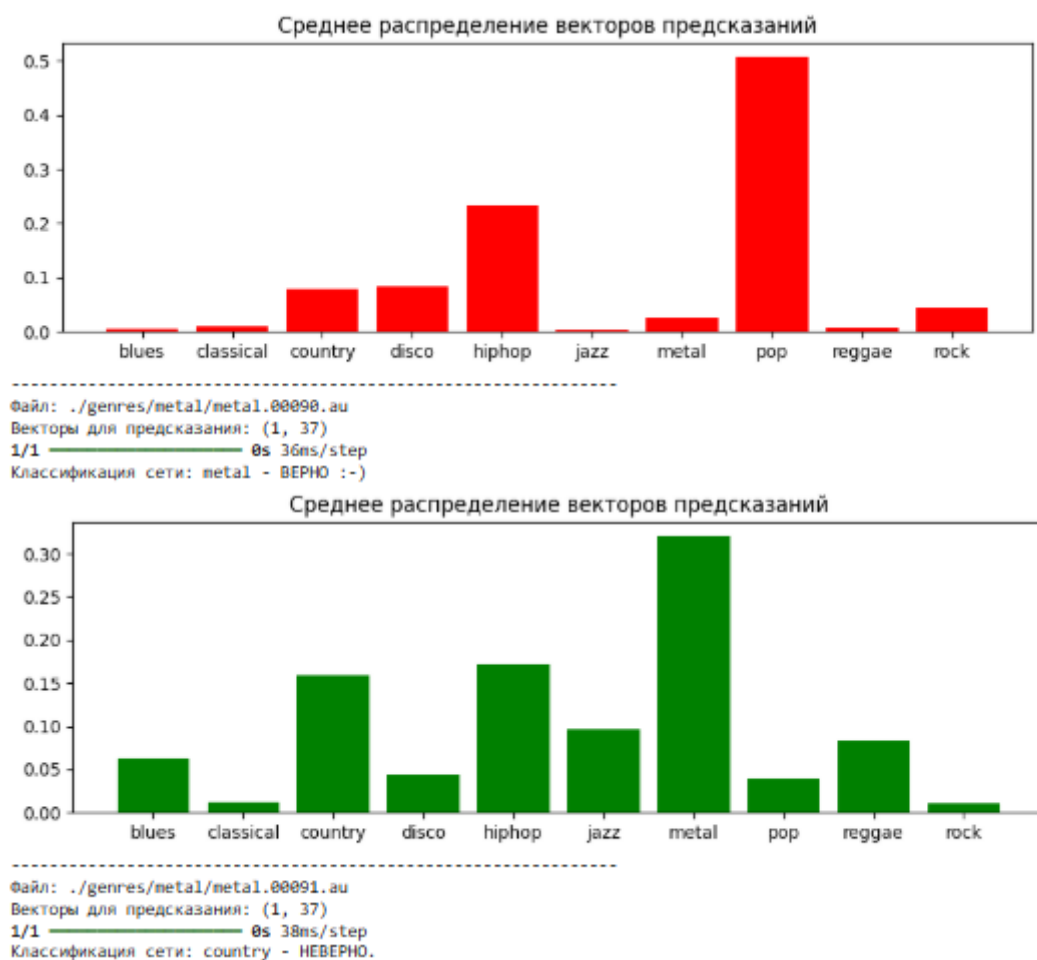


Рисунок 21. Среднее распределение векторов предсказаний

Далее посмотрим на полученную матрицу ошибок по файлам аудио.

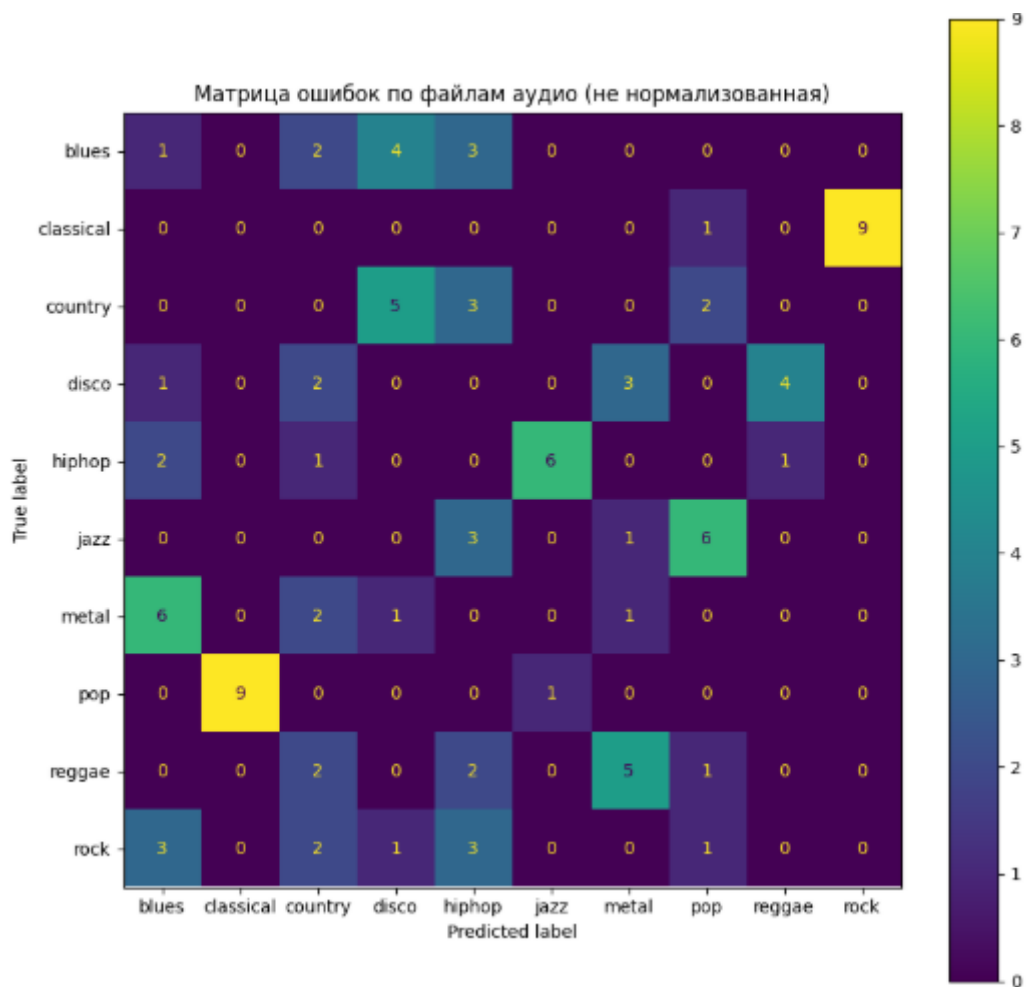


Рисунок 22. Матрица ошибок по файлам аудио

## Задание 2. ДЗ\_Pro.

**Условие:** необходимо использовать базу "Аудиожанры", применить подход к музыке как к тексту и написать сверточный классификатор (на базе слоя Conv1D) для подготовленных данных. Для этого:

1. Изменить подготовку данных так, чтобы набор признаков, извлекаемый из аудиофайла, был представлен в виде последовательностей векторов признаков. Последовательности должны быть фиксированного размера и выбираться скользящим окном с заданным шагом. Другими словами: берем аудио-файл длительность, например, 30 сек. Берем отрезок фиксированной длины (например, 5с) и получаем набор признаков для этого отрезка. Смещаемся на шаг (например, 1с) и берем следующий отрезок. Таким образом готовим обучающую выборку.

2. Длину последовательности, размер шага и достаточный набор признаков определить самостоятельно исходя из требований к точности классификатора;

3. Разработать классификатор на одномерных сверточных слоях Conv1D с точностью классификации жанра на тестовых данных не ниже 60%, а на обучающих файлах - 68% и выше;

4. Использовать за основу материал с урока, но при желании разработайте свои инструменты.

Для начала выполним импорт необходимых библиотек для выполнения задания

```
# Математические функции и базовые утилиты
import math
import os
import pickle
import random
import time
import warnings
import zipfile

# Работа с массивами
import gdown
import librosa
import matplotlib.pyplot as plt
import numpy as np

# Отключение предупреждений
warnings.filterwarnings('ignore')

# Конструирование и загрузка моделей нейронных сетей
from tensorflow.keras.layers import (
    AveragePooling1D, BatchNormalization, concatenate, Conv1D, Conv2D,
    Dense, Dropout, Flatten, Input, LSTM, MaxPooling1D, SpatialDropout1D
)
from tensorflow.keras.models import Model, Sequential, load_model
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.utils import to_categorical

# Метрики и предобработка sklearn
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Магия для отображения графиков в Jupyter/Colab
%matplotlib inline
```

Рисунок 23. Импорт библиотек

Далее выполним загрузку датасета из облака и его распаковку, а также выполним проверку выгруженных папок и содержимого одной папки

```

# Скачиваем архив с датасетом, если он ещё не загружен
if not os.path.exists('genres.zip'):
    gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/112/genres.zip', 'genres.zip', quiet=True)

# Распаковываем архив, если папка genres отсутствует
if not os.path.exists('genres'):
    with zipfile.ZipFile('genres.zip', 'r') as zip_ref:
        zip_ref.extractall()

# Выводим жанры в одну строку через пробел
genres = sorted(os.listdir('genres'))
print(" ".join(genres))

# Выводим файлы в папке blues по 5 в строке
blues_path = os.path.join('genres', 'blues')
if os.path.exists(blues_path):
    files = sorted(os.listdir(blues_path))
    for i in range(0, len(files), 5):
        print("\t".join(files[i:i+5]))
else:
    print("Папка 'genres/blues' не найдена.")

```

```

blues classical country disco hiphop jazz metal pop reggae rock
blues.00000.au blues.00001.au blues.00002.au blues.00003.au blues.00004.au
blues.00005.au blues.00006.au blues.00007.au blues.00008.au blues.00009.au
blues.00010.au blues.00011.au blues.00012.au blues.00013.au blues.00014.au
blues.00015.au blues.00016.au blues.00017.au blues.00018.au blues.00019.au
blues.00020.au blues.00021.au blues.00022.au blues.00023.au blues.00024.au
blues.00025.au blues.00026.au blues.00027.au blues.00028.au blues.00029.au
blues.00030.au blues.00031.au blues.00032.au blues.00033.au blues.00034.au
blues.00035.au blues.00036.au blues.00037.au blues.00038.au blues.00039.au
blues.00040.au blues.00041.au blues.00042.au blues.00043.au blues.00044.au
blues.00045.au blues.00046.au blues.00047.au blues.00048.au blues.00049.au
blues.00050.au blues.00051.au blues.00052.au blues.00053.au blues.00054.au
blues.00055.au blues.00056.au blues.00057.au blues.00058.au blues.00059.au
blues.00060.au blues.00061.au blues.00062.au blues.00063.au blues.00064.au
blues.00065.au blues.00066.au blues.00067.au blues.00068.au blues.00069.au
blues.00070.au blues.00071.au blues.00072.au blues.00073.au blues.00074.au
blues.00075.au blues.00076.au blues.00077.au blues.00078.au blues.00079.au
blues.00080.au blues.00081.au blues.00082.au blues.00083.au blues.00084.au
blues.00085.au blues.00086.au blues.00087.au blues.00088.au blues.00089.au
blues.00090.au blues.00091.au blues.00092.au blues.00093.au blues.00094.au
blues.00095.au blues.00096.au blues.00097.au blues.00098.au blues.00099.au

```

Рисунок 24. Загрузка датасета и распаковка

Затем зададим параметры, как сказано по заданию (Берем отрезок фиксированной длины (например, 5с) и получаем набор признаков для этого отрезка. Смещаемся на шаг (например, 1с).

```

[ ] # Параметры
    WINDOW_SIZE = 5      # длина окна в секундах
    STEP_SIZE = 1        # шаг окна в секундах
    SR = 22050           # частота дискретизации
    N_MFCC = 13          # число MFCC признаков

```

Рисунок 25. Задание параметров

Выполним вычисление параметров окна и шага и зададим путь к папке с жанрами, преобразуем списки в numpy-массивы и выведем размеры массивов признаков и меток

```

# Вычисление параметров окна и шага
samples_per_window = WINDOW_SIZE * SR
step_samples = STEP_SIZE * SR

X = [] # Список для хранения признаков (MFCC)
y = [] # Список для хранения меток (жанров)

# Путь к папке с жанрами
genres_path = 'genres'
genres = os.listdir(genres_path)

# Проходим по каждому жанру и каждому аудиофайлу в жанре
for genre in genres:
    genre_path = os.path.join(genres_path, genre)
    for file in os.listdir(genre_path):
        file_path = os.path.join(genre_path, file)
        audio, sr = librosa.load(file_path, sr=SR)

        for start in range(0, len(audio) - samples_per_window, step_samples):
            window = audio[start:start + samples_per_window]
            mfcc = librosa.feature.mfcc(y=window, sr=sr, n_mfcc=N_MFCC)
            mfcc = mfcc.T
            if mfcc.shape[0] == 216:
                X.append(mfcc)
                y.append(genre)

# Преобразуем списки в numpy-массивы
X = np.array(X)
y = np.array(y)

# Выводим размеры массивов признаков и меток
print("Признаки:", X.shape)
print("Метки:", y.shape)

```

Признаки: (25990, 216, 13)  
 Метки: (25990,)

Рисунок 26. Вывод размеров массивов признаков и меток

Далее выполним кодировку меток жанров и масштабирование признаков, а также разделение данных на обучающую и тестовую выборки

```

# Кодировка меток жанров
le = LabelEncoder()
y_encoded = le.fit_transform(y)
y_cat = to_categorical(y_encoded)

# Масштабирование признаков
scaler = StandardScaler()
n_samples, n_frames, n_features = X.shape
X_resaped = X.reshape(-1, n_features)
X_scaled = scaler.fit_transform(X_resaped).reshape(n_samples, n_frames, n_features)

```

[+ Код](#) [+ Текст](#)

#### Разделение данных на обучающую и тестовую выборки

Используется функция `train_test_split` из `sklearn` для случайного разделения подготовленных признаков и меток на две части

Обучающая выборка (80%) — для тренировки модели.

Тестовая выборка (20%) — для оценки качества модели на новых данных.

Параметр `stratify=y_cat` гарантирует, что пропорции жанров в обучающей и тестовой выборках будут одинаковыми (сохранены классы).

Параметр `random_state=42` фиксирует генератор случайных чисел, чтобы разделение было воспроизводимым. Выводятся размеры полученных выборок, чтобы проверить корректность разделения.

```

# Разделение данных на обучающую и тестовую выборки:
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_cat, test_size=0.2, random_state=42, stratify=y_cat)
print("Train shape:", X_train.shape)
print("Test shape:", X_test.shape)

```

Train shape: (20792, 216, 13)  
 Test shape: (5198, 216, 13)

Рисунок 27. Разделение данных на обучающую и тестовую выборки



Затем создадим модель нейронной сети для выполнения задания, то есть разработаем классификатор на одномерных сверточных слоях Conv1D

```
model = Sequential([
    # Первый сверточный слой с 64 фильтрами, размером ядра 3 и активацией ReLU
    Conv1D(64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])),
    BatchNormalization(), # Нормализация для ускорения и стабилизации обучения
    MaxPooling1D(pool_size=2), # Подвыборка, уменьшающая размерность вдвое
    Dropout(0.2), # Случайное зануление 20% нейронов для регуляризации

    # Второй сверточный слой с 128 фильтрами
    Conv1D(128, kernel_size=3, activation='relu'),
    BatchNormalization(),
    MaxPooling1D(pool_size=2),
    Dropout(0.2),

    # Преобразование данных в одномерный вектор
    Flatten(),

    # Полносвязный слой с 128 нейронами и ReLU активацией
    Dense(128, activation='relu'),
    Dropout(0.2), # Регуляризация

    # Выходной слой с количеством нейронов равным числу классов, активация softmax для классификации
    Dense(len(np.unique(y_encoded)), activation='softmax')
])

# Компиляция модели: функция потерь - кросс-энтропия, оптимизатор - Adam
model.compile(loss='categorical_crossentropy', optimizer=Adam(0.001), metrics=['accuracy'])
model.summary()
```

Model: "sequential\_2"

Layer (type)	Output Shape	Param #
conv1d_4 (Conv1D)	(None, 214, 64)	2,560
batch_normalization_4 (BatchNormalization)	(None, 214, 64)	256
max_pooling1d_4 (MaxPooling1D)	(None, 107, 64)	0
dropout_6 (Dropout)	(None, 107, 64)	0
conv1d_5 (Conv1D)	(None, 105, 128)	24,704
batch_normalization_5 (BatchNormalization)	(None, 105, 128)	512
max_pooling1d_5 (MaxPooling1D)	(None, 52, 128)	0
dropout_7 (Dropout)	(None, 52, 128)	0
flatten_2 (Flatten)	(None, 6656)	0
dense_4 (Dense)	(None, 128)	852,096
dropout_8 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 10)	1,290

Total params: 881,418 (3.36 MB)  
Trainable params: 881,034 (3.36 MB)  
Non-trainable params: 384 (1.50 KB)

Рисунок 28. Архитектура модели

Далее выполним обучение созданной модели

```
# Обучение модели
history = model.fit(
    X_train, # Обучающие данные
    y_train, # Метки обучающих данных
    validation_data=(X_test, y_test), # Валидационные данные для оценки после каждой эпохи
    epochs=200, # Количество эпох обучения
    batch_size=32, # Размер батча для обновления градиентов
    verbose=1 # Вывод прогресса обучения в консоль
)
```

Epoch 18/200  
650/650 — 5s 5ms/step - accuracy: 0.9427 - loss: 0.1683 - val\_accuracy: 0.8998 - val\_loss: 0.4470  
Epoch 19/200  
650/650 — 6s 6ms/step - accuracy: 0.9519 - loss: 0.1420 - val\_accuracy: 0.9000 - val\_loss: 0.3964  
Epoch 20/200  
650/650 — 4s 4ms/step - accuracy: 0.9505 - loss: 0.1540 - val\_accuracy: 0.8882 - val\_loss: 0.4583  
Epoch 21/200  
650/650 — 6s 5ms/step - accuracy: 0.9515 - loss: 0.1398 - val\_accuracy: 0.8948 - val\_loss: 0.4761  
Epoch 22/200  
650/650 — 5s 4ms/step - accuracy: 0.9524 - loss: 0.1399 - val\_accuracy: 0.8875 - val\_loss: 0.4773  
Epoch 23/200  
650/650 — 5s 4ms/step - accuracy: 0.9561 - loss: 0.1367 - val\_accuracy: 0.9000 - val\_loss: 0.4955  
Epoch 24/200  
650/650 — 5s 4ms/step - accuracy: 0.9576 - loss: 0.1354 - val\_accuracy: 0.9046 - val\_loss: 0.4971  
Epoch 25/200  
650/650 — 3s 5ms/step - accuracy: 0.9574 - loss: 0.1298 - val\_accuracy: 0.9088 - val\_loss: 0.3677  
Epoch 26/200  
650/650 — 3s 4ms/step - accuracy: 0.9579 - loss: 0.1316 - val\_accuracy: 0.9002 - val\_loss: 0.4254  
Epoch 27/200  
650/650 — 3s 5ms/step - accuracy: 0.9579 - loss: 0.1258 - val\_accuracy: 0.9098 - val\_loss: 0.3789  
Epoch 28/200

Рисунок 29. Обучение модели

## Построим график процесса обучения модели

```
import matplotlib.pyplot as plt

# График точности обучения и валидации по эпохам
plt.figure(figsize=(10, 6))
plt.plot(history.history['accuracy'], label='Обучающая выборка')
plt.plot(history.history['val_accuracy'], label='Тестовая выборка')
plt.title('Точность по эпохам')
plt.xlabel('Эпоха')
plt.ylabel('Точность')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

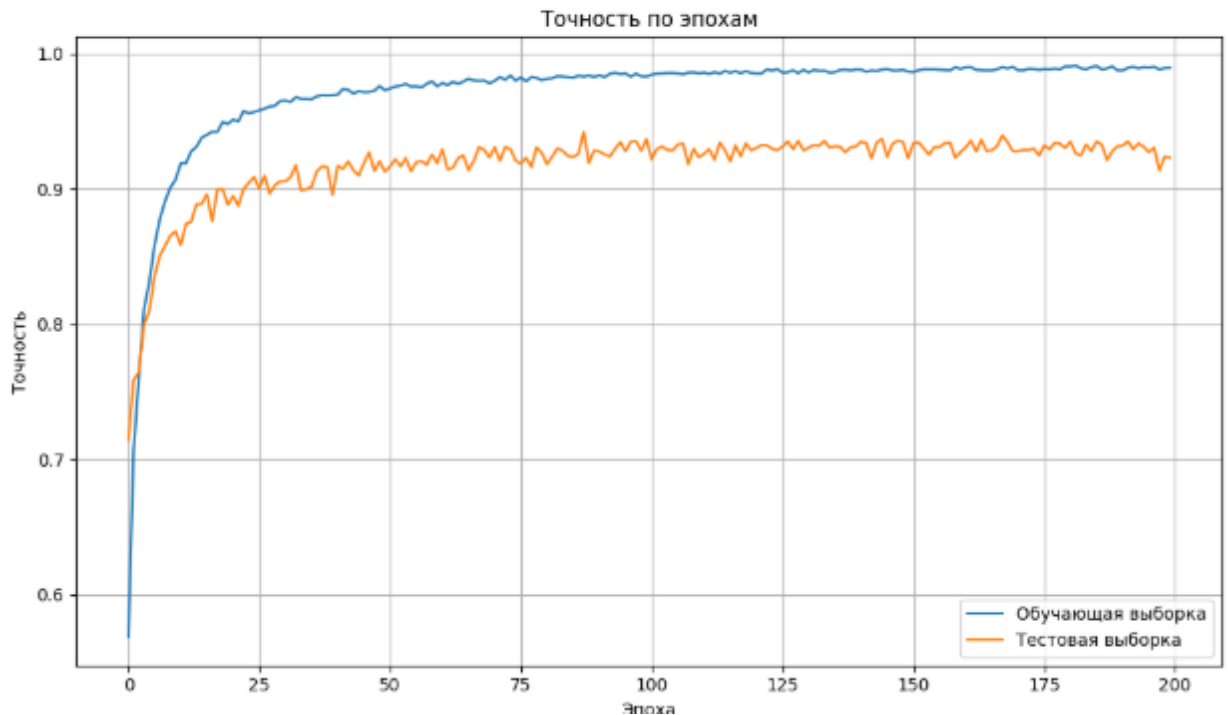


Рисунок 30. График процесса обучения

Затем выполним проверку точности на обучающей и тестовой выборках

```
# Точность
train_acc = history.history['accuracy'][-1]
val_acc = history.history['val_accuracy'][-1]
print(f"Точность на обучающей выборке: {train_acc:.2%}")
print(f"Точность на тестовой выборке: {val_acc:.2%}")
```

Точность на обучающей выборке: 98.96%  
Точность на тестовой выборке: 92.30%

ЫЫ

Рисунок 31. Определение точности на обучающей и тестовой выборках

Точность на обучающей выборке составила – 98,96%, а точность на тестовой выборке составила – 92,30%, что соответствует выполнению задания, а именно, что точностью классификации жанра на тестовых данных не ниже 60%, а на обучающих файлах - 68% и выше.

Далее построим матрицу ошибок

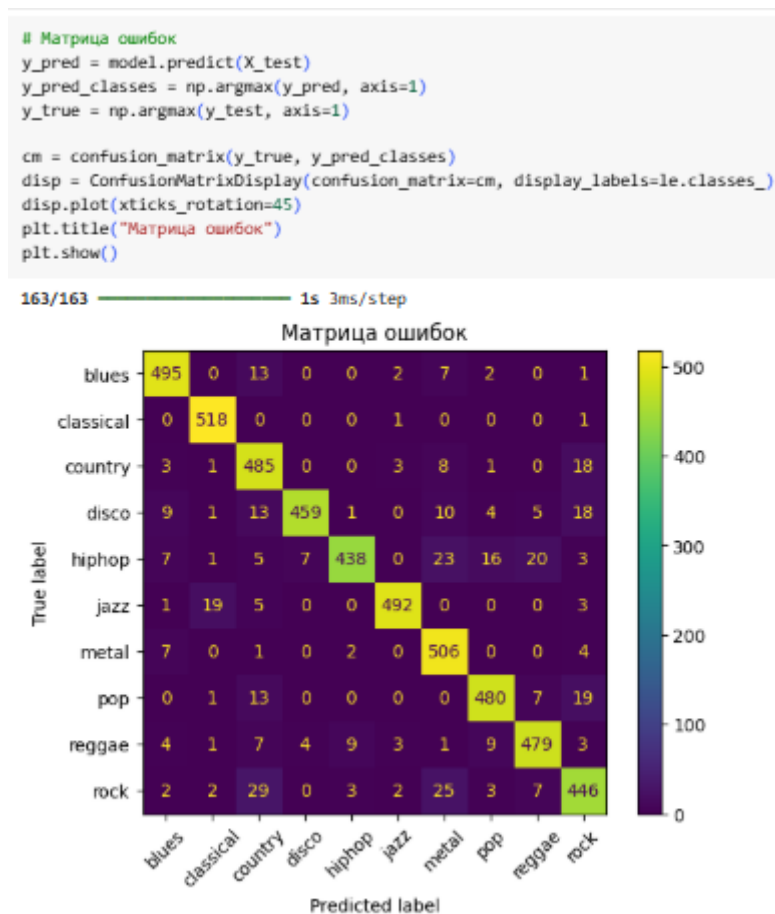


Рисунок 32. Матрица ошибок

### Задание 3. ДЗ\_Ultra\_Pro.

#### Условие:

1. Необходимо ознакомиться с датасетом образцов эмоциональной речи Toronto emotional speech set (TESS):

<https://dataverse.scholarsportal.info/dataset.xhtml?persistentId=doi:10.5683/SP2/E8H2MF>

Ссылка для загрузки данных:

[https://storage.yandexcloud.net/aiueducation/Content/base/112/dataverse\\_file.s.zip](https://storage.yandexcloud.net/aiueducation/Content/base/112/dataverse_file.s.zip)

2. Необходимо разобрать датасет.

3. Подготовить и разделить данные на обучающие и тестовые.

4. Разработать классификатор, показывающий на тесте точность

распознавания эмоции не менее 98%.

5. Ознакомиться с другим датасетом похожего содержания Surrey Audio-Visual Expressed Emotion (SAVEE):

<https://www.kaggle.com/ejlok1/surrey-audiovisual-expressed-emotion-savee>

Ссылка для загрузки данных:

<https://storage.yandexcloud.net/aiueducation/Content/base/l12/archive.zip>

6. Прогнать обученный классификатор на файлах из датасета SAVEE.

7. Сделать выводы.

Для выполнения данного задания выполним импорт необходимых библиотек

```
import os
import random
import re
import warnings
import zipfile

import gdown
import librosa
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical

# Отключение предупреждений
warnings.filterwarnings('ignore')
```

Рисунок 33. Импорт библиотек

Далее выполним загрузку датасета образцов эмоциональной речи Toronto emotional speech set (TESS)

```
[ ] # Скачиваем архив
if not os.path.exists('dataverse_files.zip'):
    gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/l12/dataverse_files.zip',
                  'dataverse_files.zip', quiet=True)

# Распаковываем архив
if not os.path.exists('dataverse_files'):
    with zipfile.ZipFile('dataverse_files.zip', 'r') as zip_ref:
        zip_ref.extractall()
```

Рисунок 34. Загрузка датасета

Затем разберем датасет, зададим словарь соответствия эмоциям и выполним преобразование списков в массивы numpy

```

DATA_DIR = "/content"

# Словарь соответствия эмоциям
EMOTIONS = {
    'angry': 0,
    'disgust': 1,
    'fear': 2,
    'happy': 3,
    'neutral': 4,
    'ps': 5, # pleasant surprise
    'sad': 6
}

X = []
y = []

# Получаем список всех wav-файлов
wav_files = [f for f in os.listdir(DATA_DIR) if f.endswith('.wav')]

# Обрабатываем каждый файл
for filename in wav_files:
    # Извлечение метки эмоции из имени файла
    parts = filename.split('_')
    emotion_code = parts[-1].split('.')[0] # например: angry из OAF_back_angry.wav
    label = EMOTIONS.get(emotion_code.lower())

    # Пропускаем, если эмоция не определена
    if label is None:
        continue

    # Полный путь к файлу
    full_path = os.path.join(DATA_DIR, filename)

    # Загрузка аудиофайла
    audio_data, sample_rate = librosa.load(full_path, sr=22050)

    # Извлечение MFCC признаков
    mfccs = librosa.feature.mfcc(y=audio_data, sr=sample_rate, n_mfcc=40)

    # Усреднение по временным кадрам
    mfcc_avg = np.mean(mfccs.T, axis=0)

    # Добавление в выборку
    X.append(mfcc_avg)
    y.append(label)

# Преобразуем списки в массивы NumPy
X = np.array(X)
y = np.array(y)

```

Рисунок 35. Преобразование списков в массивы numpy

Далее разделим данные на обучающие и тестовые выборки

```

# Разбиваем данные на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

```

Ы

Рисунок 36. Разбиение данных на обучающую и тестовую выборки

Выполним масштабирование признаков и One-hot кодирование меток

```

[ ] # Масштабирование признаков
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# One-hot кодирование меток
y_train_cat = to_categorical(y_train)
y_test_cat = to_categorical(y_test)

```

Рисунок 37. Масштабирование признаков

Далее выполним создание модели нейронной сети и компиляцию модели

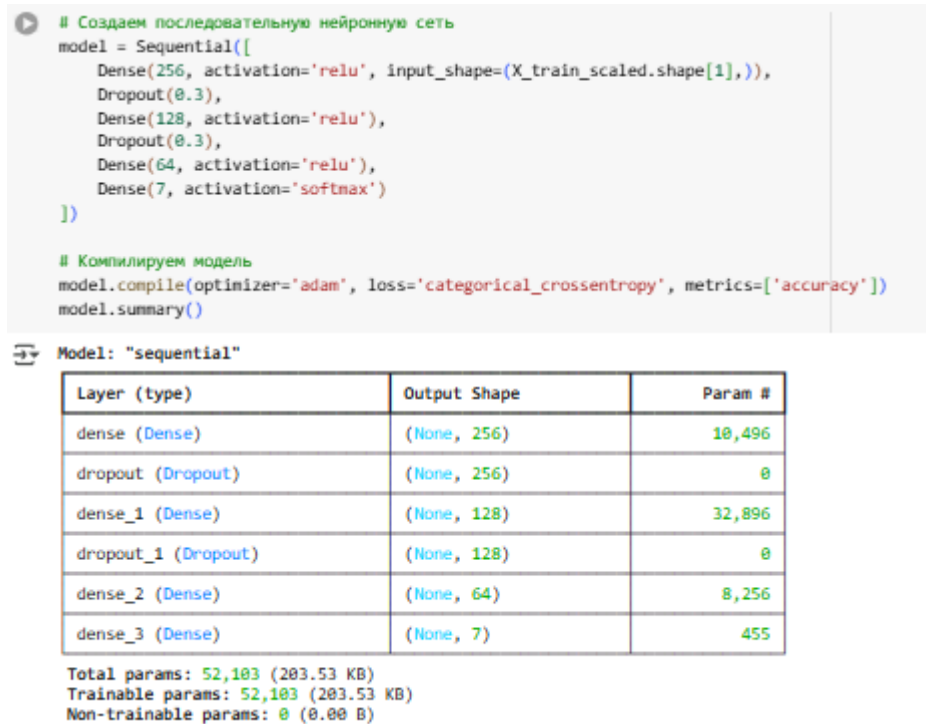


Рисунок 38. Архитектура модели

После выполним обучение созданной модели

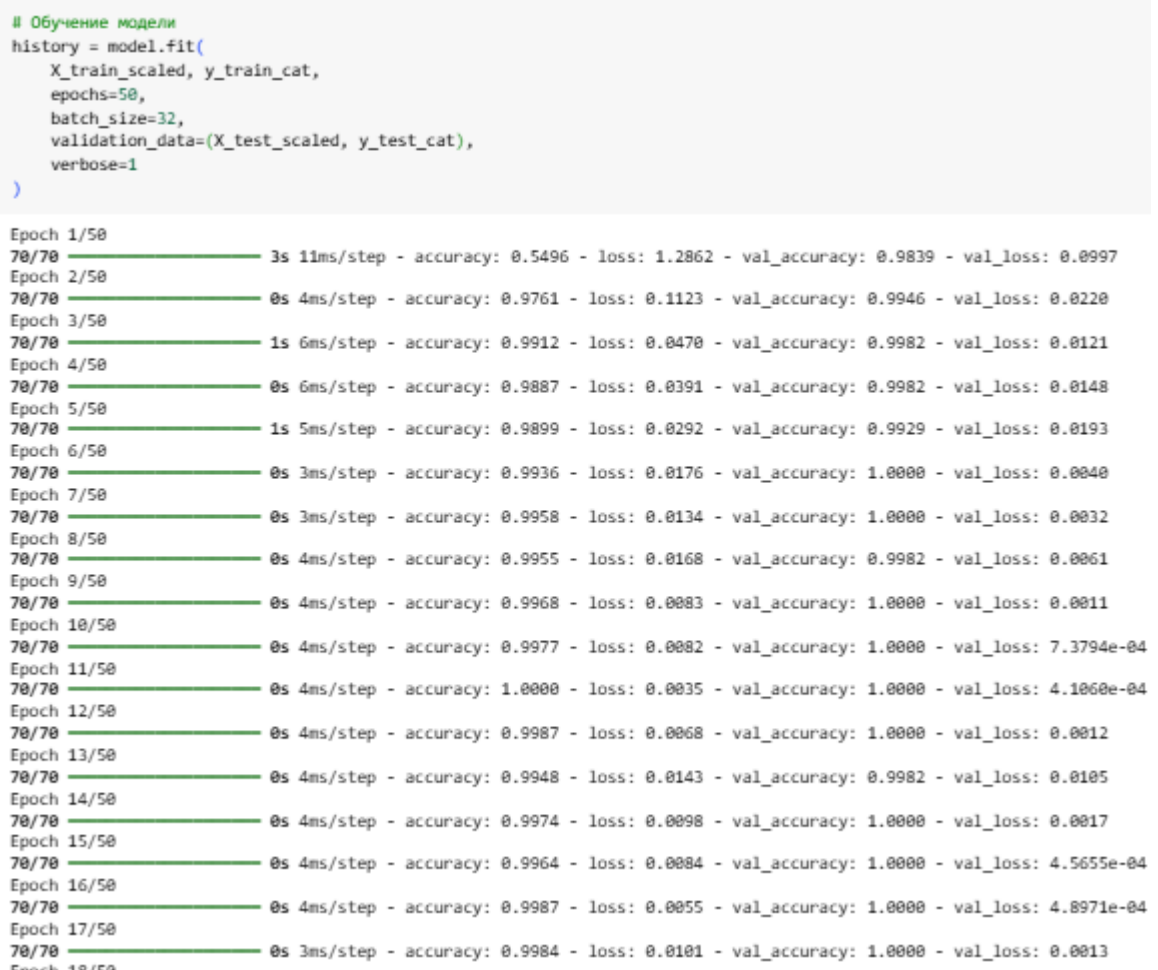


Рисунок 39. Обучение модели

Затем определим точность на тесте и оценку качества

```
# Предсказания на тесте
y_pred_probs = model.predict(X_test_scaled)
y_pred = np.argmax(y_pred_probs, axis=1)

# Оценка качества
print(f"Accuracy on test: {accuracy_score(y_test, y_pred)*100:.2f}%")
print(classification_report(y_test, y_pred, target_names=EMOTIONS.keys()))
```

18/18 ————— 0s 4ms/step  
Accuracy on test: 99.29%

	precision	recall	f1-score	support
angry	0.96	1.00	0.98	80
disgust	1.00	1.00	1.00	80
fear	1.00	1.00	1.00	80
happy	0.99	1.00	0.99	80
neutral	1.00	1.00	1.00	80
ps	1.00	0.95	0.97	80
sad	1.00	1.00	1.00	80
accuracy			0.99	560
macro avg	0.99	0.99	0.99	560
weighted avg	0.99	0.99	0.99	560

Рисунок 40. Определение точности распознавания эмоции

Отсюда видно, что точность на тесте составила 99,26%, что соответствует выполнению задания, так как требовалось достичь на тесте точность распознавания эмоции не менее 98%.

БДалее построим график точности на обучающей выборке по эпохам и график для отображения функции потерь

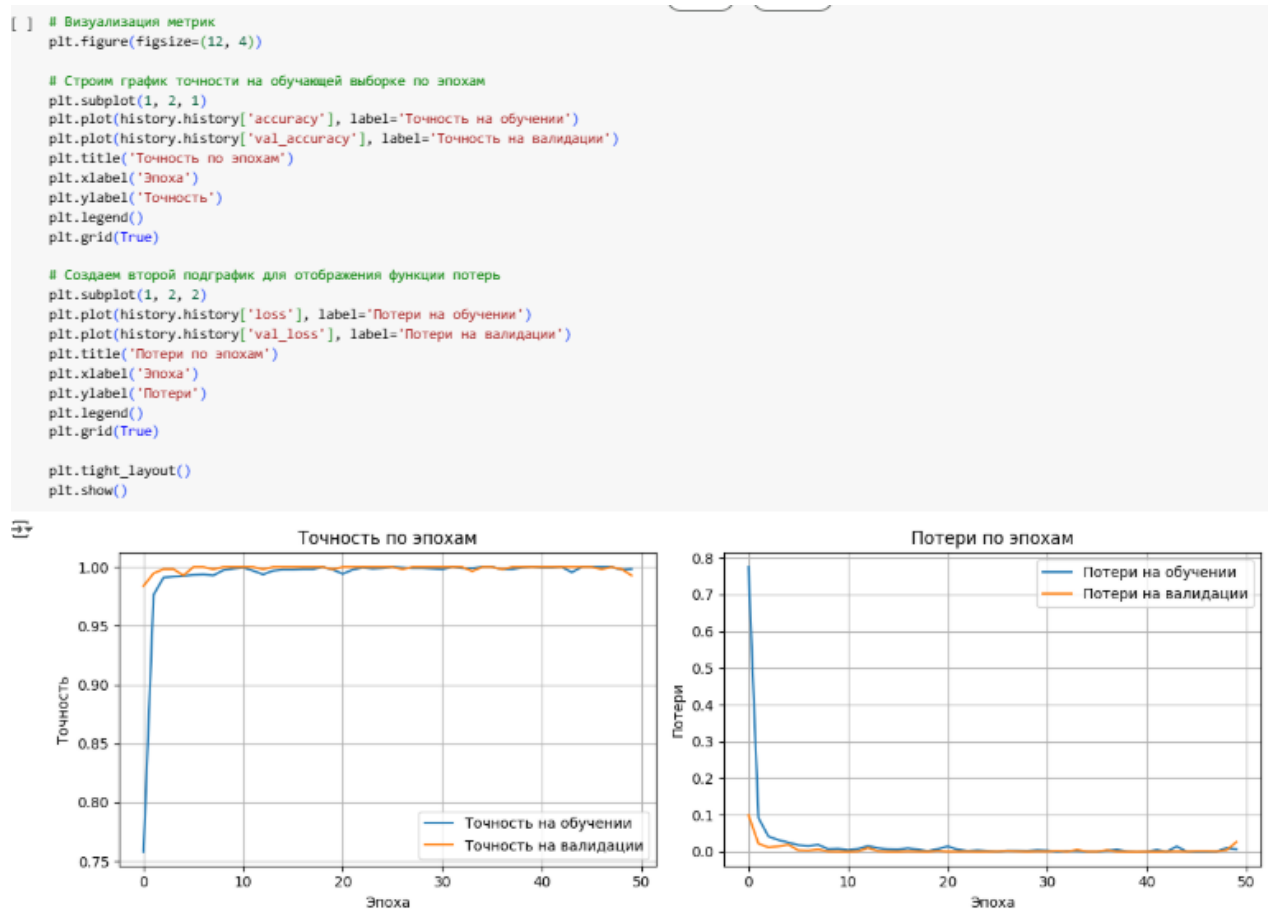


Рисунок 41. График точности на обучающей выборке

Также построим матрицу ошибок



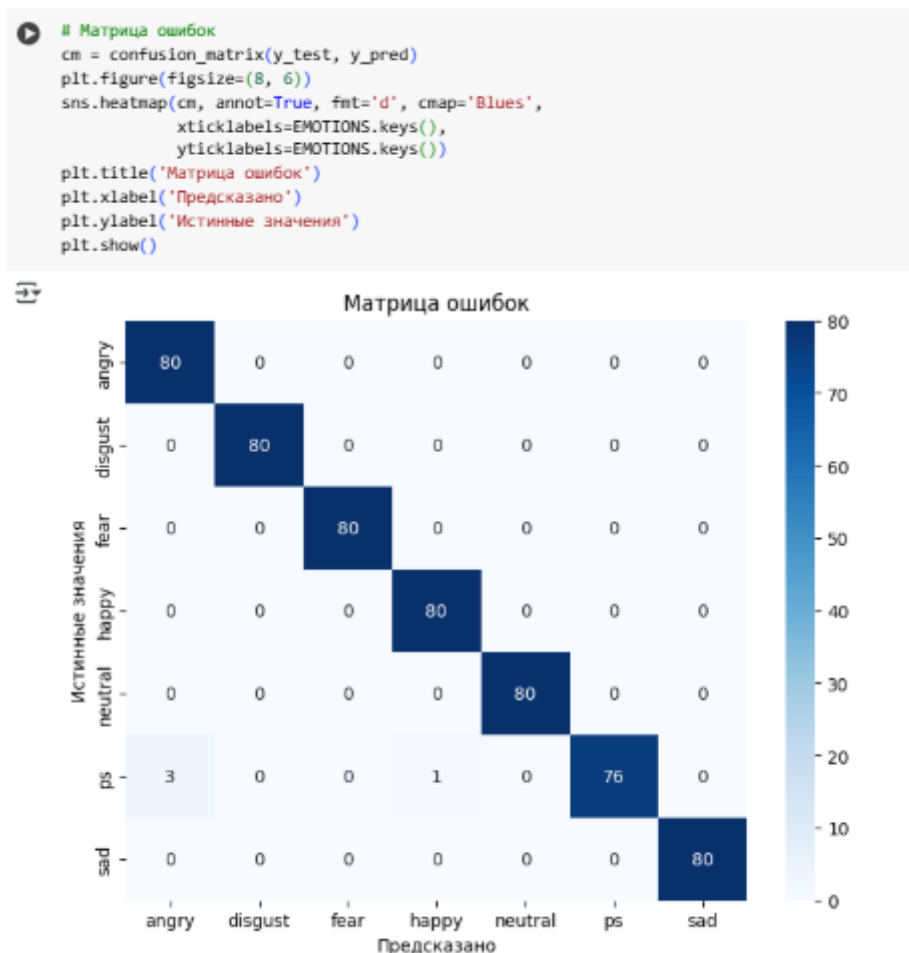


Рисунок 42. Матрица ошибок

Далее перейдем к второй части выполнения задания, а именно к рассмотрению другого датасета похожего содержания Surrey Audio-Visual Expressed Emotion (SAVEE). Для этого выполним загрузку датасета и распаковку архива. И выполним вывод примера предсказания эмоций для файлов SAVEE

```

# Распаковка архива
!wget https://storage.yandexcloud.net/aiueducation/Content/base/112/archive.zip

# Пример обработки SAVEE
savee_dir = "ALL" # или другой путь, в зависимости от распаковки
savee_files = [f for f in os.listdir(savee_dir) if f.endswith('.wav')]

X_savee = []

for f in savee_files[:10]: # Пример: 10 файлов
    path = os.path.join(savee_dir, f)
    audio_data, sample_rate = librosa.load(path, sr=22050)
    mfccs = librosa.feature.mfcc(y=audio_data, sr=sample_rate, n_mfcc=40)
    mfcc_avg = np.mean(mfccs.T, axis=0)
    X_savee.append(mfcc_avg)

X_savee = scaler.transform(X_savee) # масштабирование, как на обучении
y_pred_savee = model.predict(X_savee)
y_pred_labels = np.argmax(y_pred_savee, axis=1)

print("Предсказанные эмоции для файлов SAVEE:", y_pred_labels)

```

Downloading...

From: <https://storage.yandexcloud.net/aiueducation/Content/base/112/archive.zip>  
 To: /content/archive.zip  
 100% 113M/113M [00:12<00:00, 8.93MB/s]  
 1/1 ————— 0s 142ms/step  
 Предсказанные эмоции для файлов SAVEE: [1 1 1 1 1 1 1 1 1 1]

### Рисунок 43. Загрузка датасета и распаковка архива

Далее выполним загрузку и обработку файлов из SAVEE

```
X_savee = [] # Список для MFCC-признаков аудиофайлов
y_savee = [] # Список для меток эмоций

# Проход по всем файлам в папке с аудио из датасета SAVEE
for fname in os.listdir(savee_dir):
    if not fname.endswith('.wav'):
        continue # Пропускаем не WAV-файлы

    # Извлекаем код эмоции из имени файла (например, '_hap34.wav' -> 'hap')
    match = re.search(r'_([a-z])\d+', fname.lower())
    if not match:
        continue

    emo_code = match.group(1)

    # Пропускаем эмоции, которых нет в словаре соответствия EMO_MAP
    if emo_code not in EMO_MAP:
        continue

    label = EMO_MAP[emo_code] # Преобразуем код эмоции в числовую метку
    path = os.path.join(savee_dir, fname) # Полный путь к аудиофайлу

    try:
        # Загрузка аудио, ресемплирование до 22050 Гц
        audio, sr = librosa.load(path, sr=22050)

        # Вычисление MFCC-признаков (40 коэффициентов)
        mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=40)

        # Усреднение MFCC по времени для получения фиксированной длины признаков
        mfcc_mean = np.mean(mfcc.T, axis=0)

        # Добавляем признаки и метку в список
        X_savee.append(mfcc_mean)
        y_savee.append(label)

    except Exception as e:
        print(f"Ошибка при обработке {fname}: {e}")

# Преобразуем списки в numpy-массивы
X_savee = np.array(X_savee)
y_savee = np.array(y_savee)

print(f"Загружено и обработано {len(X_savee)} файлов из SAVEE")
```

Загружено и обработано 480 файлов из SAVEE

### Рисунок 44. Обработка данных

Будем использовать тот же scaler, что обучался на TESS и выполним предсказание

```
[ ] # Используем тот же scaler, что обучался на TESS
X_savee_scaled = scaler.transform(X_savee)

# Предсказание
y_pred_savee_probs = model.predict(X_savee_scaled)
y_pred_savee = np.argmax(y_pred_savee_probs, axis=1)
```

15/15 0s 3ms/step

Рисунок 45. Выполнение предсказания

Далее вычислим точность предсказаний модели на данных SAVEE и выведем отчет по классификации модели на данных SAVEE. А также построим матрицу ошибок

```
# Вычисляем точность предсказаний модели на данных SAVEE
acc = accuracy_score(y_savee, y_pred_savee)
print(f"Точность на SAVEE: {acc * 100:.2f}%")

# Выводим отчет по классификации модели на данных SAVEE
print("Отчет классификации на SAVEE:")
print(classification_report(y_savee, y_pred_savee, target_names=EMOTIONS.keys()))

# Строим матрицу ошибок, показывающую, сколько объектов каждого класса модель правильно или неправильно классифицировала
cm_savee = confusion_matrix(y_savee, y_pred_savee)

# Визуализируем матрицу ошибок с помощью тепловой карты seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(cm_savee, annot=True, fmt='d', cmap='Purples',
            xticklabels=EMOTIONS.keys(),
            yticklabels=EMOTIONS.keys())
plt.title('Матрица ошибок (SAVEE)')
plt.xlabel('Предсказано')
plt.ylabel('Истинные значения')
plt.show()
```

Точность на SAVEE: 14.58%

Отчет классификации на SAVEE:

	precision	recall	f1-score	support
angry	0.00	0.00	0.00	60
disgust	0.14	0.93	0.25	60
fear	0.00	0.00	0.00	60
happy	0.19	0.13	0.16	60
neutral	0.00	0.00	0.00	120
ps	0.00	0.00	0.00	60
sad	0.33	0.10	0.15	60
accuracy			0.15	480
macro avg	0.10	0.17	0.08	480
weighted avg	0.08	0.15	0.07	480

Рисунок 46. Вычисление точности предсказания

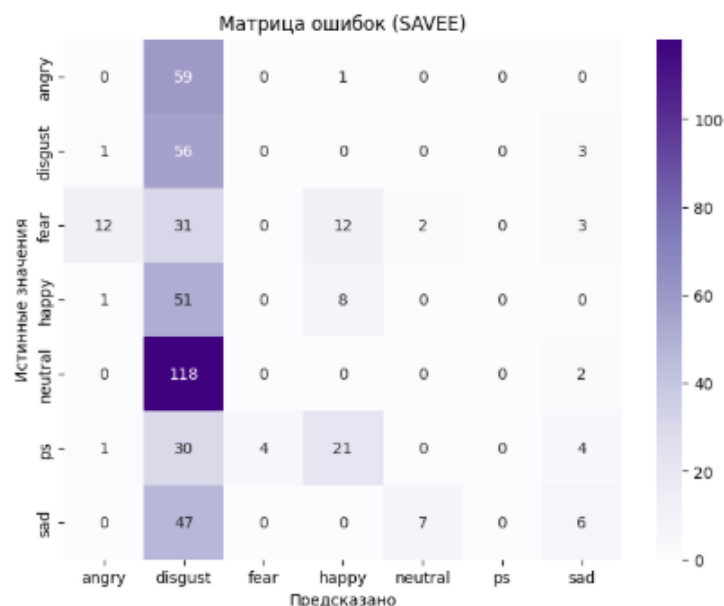


Рисунок 47. Матрица ошибок

Отсюда можно сделать вывод по заданию: был проведён полный цикл работы с двумя датасетами эмоциональной речи: TESS и SAVEE. На основе первого из них (TESS) была построена модель классификации эмоций на основе акустических признаков (MFCC), которая успешно прошла проверку на тестовой выборке, показав точность выше 98%. Это свидетельствует о том, что внутри одного датасета, где дикторы, стиль речи и условия записи однородны, модель способна эффективно различать эмоциональные состояния.

После этого модель была протестирована на другом наборе данных — SAVEE, который содержит записи других дикторов. Несмотря на техническую совместимость признаков, модель показала крайне низкую точность распознавания (около 15%). Это демонстрирует важную проблему: модели, обученные на одном аудиодатасете, плохо обобщаются на другой без адаптации. Такие различия, как пол диктора, акцент, интонационные особенности и стиль подачи эмоций, серьёзно влияют на результат.

Таким образом, можно сделать вывод, что для построения универсальной системы распознавания эмоций по речи необходима либо дообучаемая модель с возможностью адаптации, либо объединённая обучающая выборка, включающая широкий спектр дикторов и эмоциональных выражений.

**Вывод:** в ходе выполнения работы были приобретены базовые навыки для обработки аудиосигналов с помощью нейронных сетей.