

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:

Евдаков Евгений Владимирович

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.

Ход работы:

Практика 1. Обработка аудиосигналов с помощью нейронных сетей (полное выполнение данной практики можно посмотреть в репозитории по ссылке в конце работы).

В данной практике рассматривается правильность пользования библиотекой LibROSA и также рассматриваются параметры которые можно получить с ее помощью.

В первой части практики рассматривается обработка аудиосигналов, а именно параметризация аудио. Для этого вначале происходит импорт библиотек, включая LibROSA. Далее происходит загрузка датасета и его распаковка. Далее рассмотрены примеры представлений аудиосигналов, для была написана функция загрузки аудио и вывода текстовой информации (рис. 1).

```
[6] # Функция загрузки аудио и вывода текстовой информации
    # о сигнале, а также проигрывателя аудио

def load_audio(audio_path,          # путь к файлу с аудио
               show_text=True,     # показывать ли текстовую сводку по аудио
               show_player=True):  # выводить ли проигрыватель в ячейку
    """
    # Загрузка аудиофайла, на выходе:
    # x - массив данных временного ряда аудио
    # sr - частота дискретизации временного ряда
    x, sr = librosa.load(audio_path)

    if show_text:
        # Вывод текстовых данных о сигнале
        print(f'Типы данных x и sr: {type(x)}, {type(sr)}')
        print(f'Форма данных x: {x.shape}, sr = {sr}')
        print(f'Продолжительность сигнала: {round(x.shape[0]/sr)}, 'c.\n')

    if show_player:
        # Вывод проигрывателя в ячейку colab
        ipd.display(ipd.Audio(audio_path))

    # Возврат загруженных данных для дальнейшего использования
    return x, sr
```

Рисунок 1. Функция загрузки аудио и вывода текстовой информации

Были рассмотрены амплитудно-временные (волновые) представления. Пример одно из них показан на рисунке 2, а именно вывод сигнала на экран.

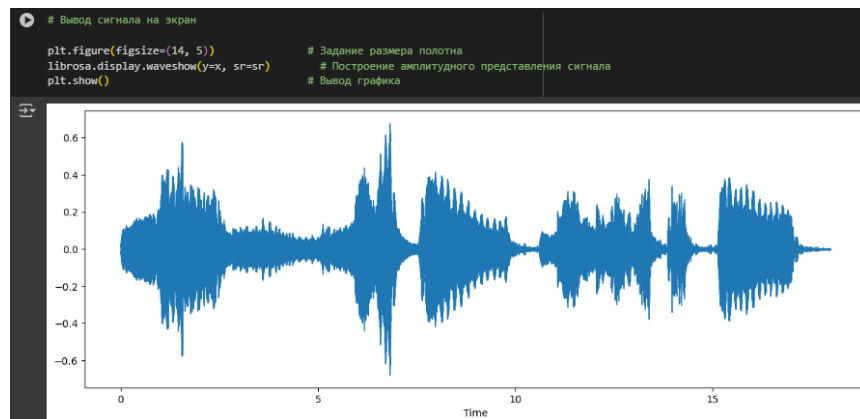


Рисунок 2. Вывод сигнала на экран

Далее рассматриваются спектрально-временные представления аудиосигнала (сонограмма). Также прописывается функция для вывода сонограммы сигнала. После рассмотрим один из примеров сонограммы музыкальных звуков, а именно исследуем ноту Ля (рис. 3 - 4).

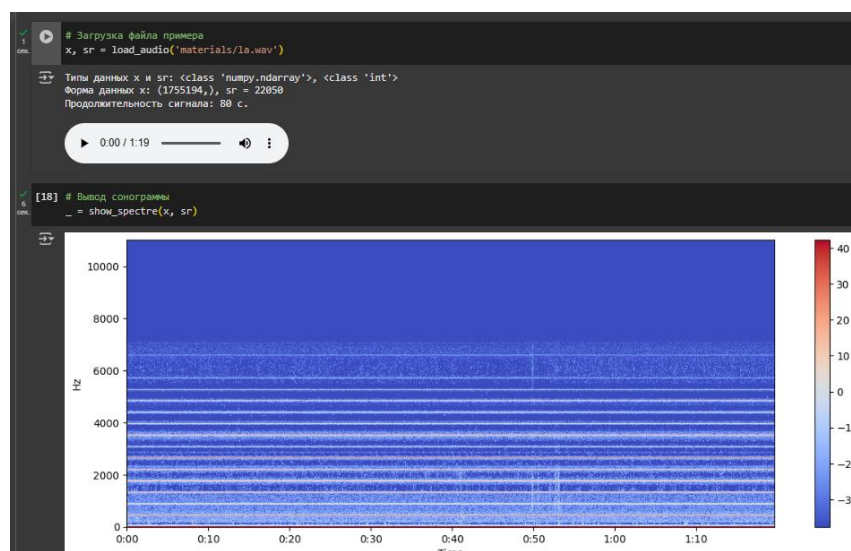


Рисунок 3. Загрузка файла

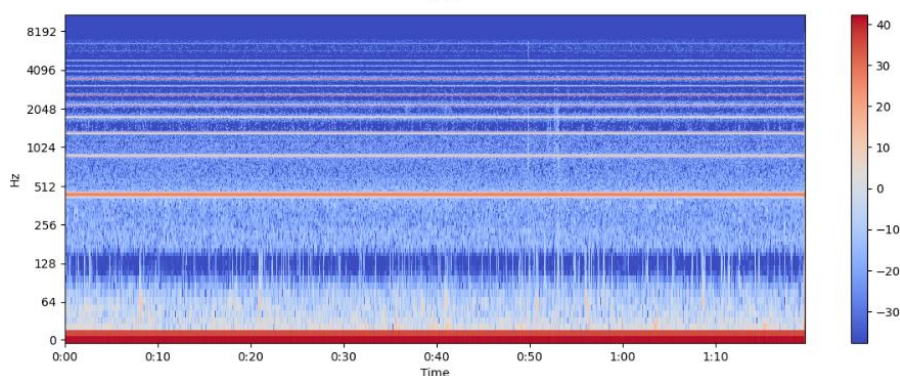


Рисунок 4. Вывод сонограммы

Затем рассматривается извлечение признаков из аудиосигнала, а так же формируется спектральный центроид (рис. 5).

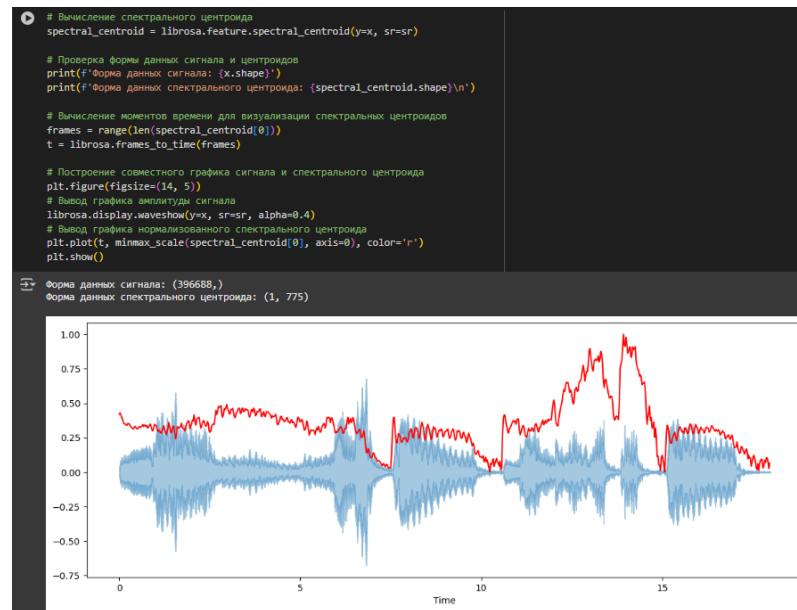


Рисунок 5. Спектральный центроид

Далее рассматриваются спектральный спад ("завал" частоты) и мел-частотные кепстральные коэффициенты, а также хромограммы музыкальных отрывков, рассмотрим пример одного из них (рис. 6 - 7).

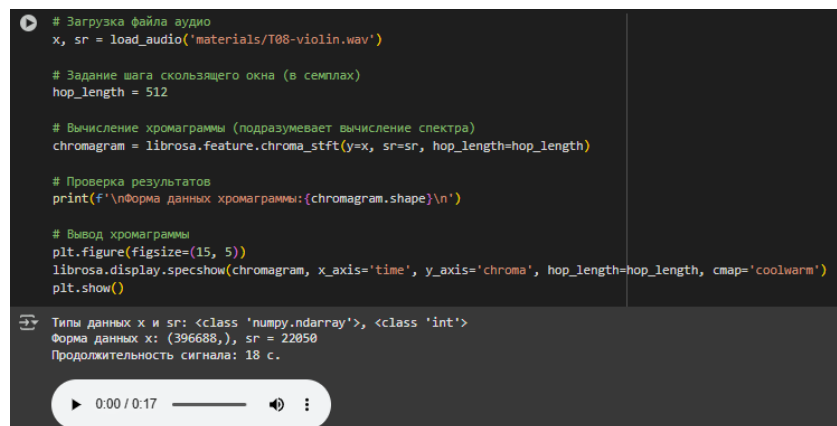


Рисунок 6. Загрузка файла аудио

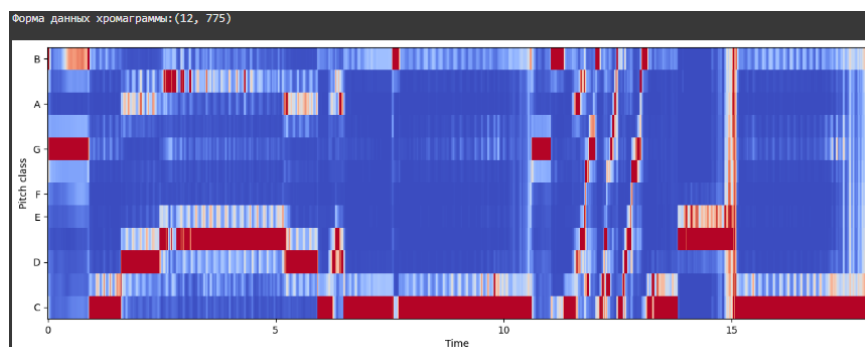


Рисунок 7. Хромограмма музыкального отрывков

Практика 2. Обработка аудиосигналов с помощью нейронных сетей (полное выполнение данной практики можно посмотреть в репозитории по ссылке в конце работы).

В данной практике рассматривается решение задачи - классификации музыкальных жанров на основании усреднения набора признаков.

Для выполнения данной практики необходимо импортировать нужные библиотеки. Далее выполняется обработка аудиосигналов, а именно классификация музыкальных жанров, для этого выполним загрузку датасета и подготовку данных. Напишем функцию для аудио (рис. 8).

```
[6] # Функция параметризации аудио
def get_features(y, sr, n_fft=N_FFT, hop_length=HOP_LENGTH):
    # Вычисление различных параметров (признаков) аудио
    # (среднеквадратическая амплитуда)
    rms = librosa.feature.rms(y=y, hop_length=hop_length)
    # Спектральный центр
    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 сдвиг частоты
    rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr, n_fft=n_fft, hop_length=hop_length)
    # Пересечение нуля
    zcr = librosa.feature.zero_crossing_rate(y, hop_length=hop_length)
    # Мел-кепстральные коэффициенты
    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_fft=n_fft, hop_length=hop_length)
    # Хромограмма
    chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr, n_fft=n_fft, hop_length=hop_length)

    # Сборка параметров в общий список
    # На один файл один усредненный вектор признаков
    features = {'rms': rms.mean(axis=1, keepdims=True),
               'spec': spec_cent.mean(axis=1, keepdims=True),
               'specbw': spec_bw.mean(axis=1, keepdims=True),
               'rolloff': 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 8. Функция параметризации аудио

Далее выполняется создание нейронной сети для выполнения задания (рис. 9).

```
# Функция сборки и обучения классификатора на полносвязных слоях
def create_train_classifier(in_shape, epochs=200, batch_size=20):
    # форма входных данных модели
    # количество эпох обучения
    # размер батча

    model = Sequential()
    model.add(Dense(256, activation='relu', input_shape=in_shape))
    model.add(Dropout(0.3))
    model.add(BatchNormalization())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.3))
    model.add(BatchNormalization())
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.3))
    model.add(BatchNormalization())
    model.add(Dense(CLASS_COUNT, activation='softmax'))

    # Компиляция модели
    model.compile(optimizer=Adam(lr=1e-4),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    model.summary()

    # Обучение модели
    history = model.fit(x_train,
                        y_train,
                        epochs=epochs,
                        batch_size=batch_size,
                        validation_data=(x_val, y_val))

    # Вывод графика точности распознавания на обучающей и проверочной выборках
    plt.figure(figsize=(12, 5))
    plt.plot(history.history['accuracy'], label='Точность на обучающем наборе')
    plt.plot(history.history['val_accuracy'], label='Точность на проверочном наборе')
    plt.xticks(range(0, epochs, 10))
    plt.xlabel('Эпохи')
    plt.ylabel('Точность')
    plt.legend()
    plt.show()

    return model
```

Рисунок 9. Создание модели

Затем выполняется обучение созданной модели и выводится график процесса обучения (рис. 10).

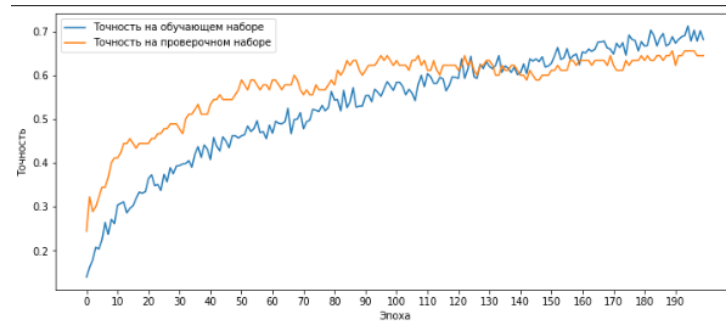


Рисунок 10. График процесса обучения

Отсюда видно, что что точность после 200 эпох достигает 65%. Далее сохраняется данная выборка для следующих экспериментах. Затем выполняется оценка точности сети на проверочной и тестовой выборках. Рассмотрим матрицу нормализации на тестовой выборке (рис. 11).

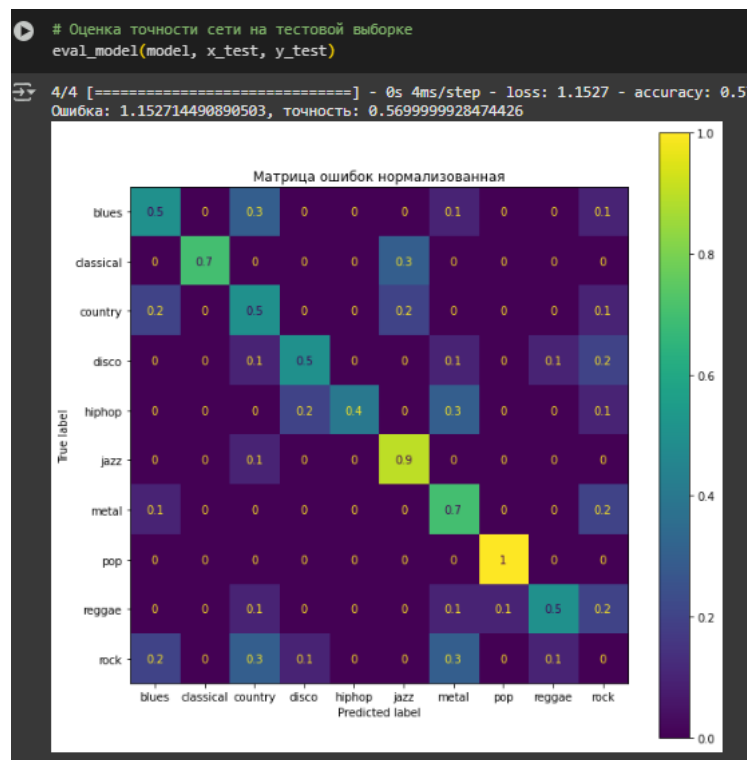


Рисунок 11. Оценка точности сети на тестовой выборке

Затем выполняется классификация файла и визуализация предсказания модели для него, и классификация и визуализация нескольких файлов каждого класса. После посмотрим на визуализацию классификации файлов из тренировочного набора (рис. 12).



Рисунок 12. Визуализация классификации файлов из тренировочного набора

Далее рассматривается методика подбора значимых параметров для обучения модели, для этого пишется функция создания и обучения упрощенной архитектуры классификатора. Затем происходит испытание малой модели на различных подмножествах входных признаков, указывая их индексы явно или диапазоном. Рассмотрим один из примеров обучения малой модели (рис. 13).

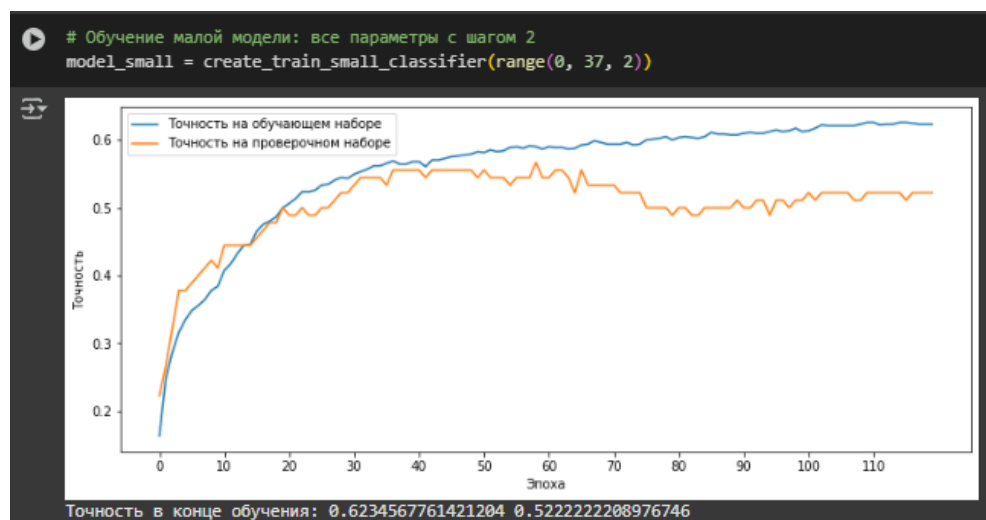


Рисунок 13. Обучение малой модели, все параметры с шагом 2

Затем прodelьваются все те же действия только для новой архитектуры с softmax (процесс выполнения можно посмотреть в репозитории).

Практика 3. Обработка аудиосигналов с помощью нейронных сетей.

В данной практике код повторяет только что изученную вторую практику, за исключением того, что используется значительно более объемный набор входных признаков за счет другого способа параметризации аудио (без усреднения по всему файлу).

Рассматривается обработка аудиосигналов, а именно классификация музыкальных жанров (полный набор признаков). Для этого выполняется импорт библиотек и загрузка датасета, а затем подготовка данных. Далее код выполнения совпадает с практикой 2. Рассмотрим функцию параметризации аудио (рис. 14).

```
# 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-кепстральные 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 признаков в общий список:
    # На один файл несколько векторов признаков, количество определяется
    # продолжительностью аудио и параметром hop_length в функциях расчета признаков
    features = {'rmse': rmse,
               'spct': spec_cent,
               'spbw': spec_bw,
               'roff': rolloff,
               'zcr': zcr,
               'mfcc': mfcc,
               'stft': chroma_stft}

    return features
```

Рисунок 14. Функция параметризации аудио

Далее после написания всех функций и задания параметров строится функция сборки и обучения модели классификатора на полносвязных слоях (рис. 15).


```

# Функция сборки и обучения классификатора на полносвязных слоях

def create_train_classifier(in_shape,      # форма входных данных модели
                           epochs=50,    # количество эпох обучения
                           batch_size=512 # размер батча
                           ):
    model = Sequential()
    model.add(Dense(256, activation='relu', input_shape=in_shape))
    model.add(Dropout(0.3))
    model.add(BatchNormalization())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.3))
    model.add(BatchNormalization())
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.3))
    model.add(BatchNormalization())
    model.add(Dense(CLASS_COUNT, activation='softmax'))

    # Компиляция модели
    model.compile(optimizer=Adam(lr=1e-4),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    model.summary()

    # Обучение модели
    history = model.fit(x_train,
                        y_train,
                        epochs=epochs,
                        batch_size=batch_size,
                        validation_data=(x_val, y_val))

    # Вывод графика точности распознавания на обучающей и проверочной выборках
    plt.figure(figsize=(12, 5))
    plt.plot(history.history['accuracy'], label='Точность на обучающем наборе')
    plt.plot(history.history['val_accuracy'], label='Точность на проверочном наборе')
    plt.xticks(range(epochs))
    plt.xlabel('Эпоха')
    plt.ylabel('Точность')
    plt.legend()
    plt.show()

    return model

```

Рисунок 15. Функция сборки и обучения классификатора

Далее посмотрим на график процесса обучения (рис. 16).

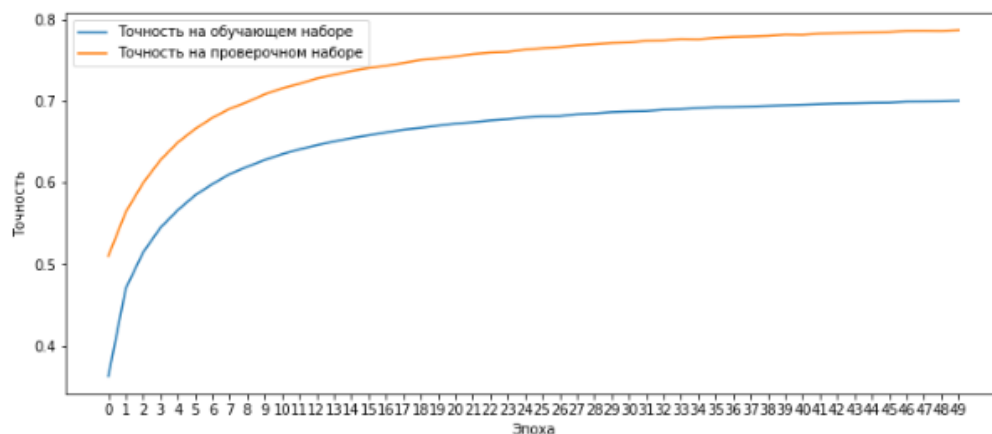


Рисунок 16. График процесса обучения

Затем выполняется сохранение модели выборки, а потом проводится проверка точности предсказаний модели. Для этого пишется функция оценки точности модели на заданной выборке, а потом выполняется вывод оценки точности сети на проверочной и тестовой выборках. Рассмотрим матрицу ошибок нормализации на проверочной выборке (рис. 17).

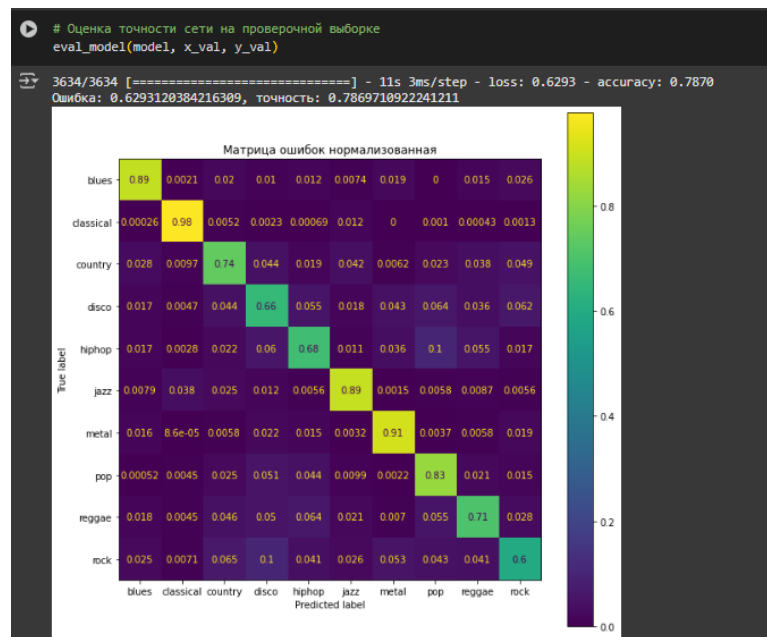


Рисунок 17. Оценка точности сети на проверочной выборке

Затем выполняется классификация файла и визуализация предсказания модели для него, и классификация и визуализация нескольких файлов каждого класса. После посмотрим на визуализацию классификации файлов из тренировочного набора (рис. 18).

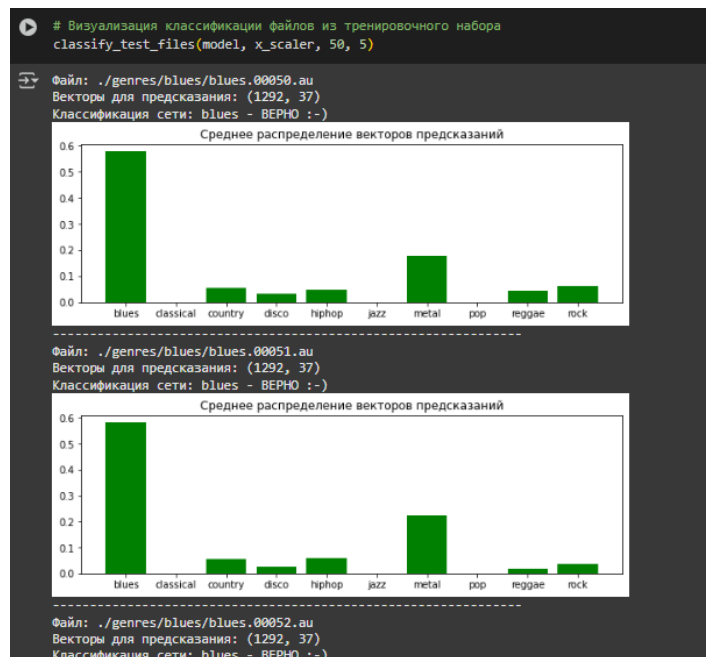


Рисунок 18. Визуализация классификации файлов из тренировочного набора

Далее рассматривается методика подбора значимых параметров для обучения модели, для этого пишется функция создания и обучения упрощенной архитектуры классификатора (рис. 19).

```

# Функция создания и обучения упрощенной архитектуры классификатора
def create_train_small_classifier(index_list,
                                x_train=x_train,
                                y_train=y_train,
                                x_val=x_val,
                                y_val=y_val,
                                epochs=120,
                                batch_size=2048,
                                verbose=0
                                ):
    # Список номеров признаков в списке
    # обучающая выборка - вход
    # обучающая выборка - выход
    # проверочная выборка - вход
    # проверочная выборка - выход
    # количество эпох обучения
    # размер батча
    # подробность вывода при обучении

    # Сборка модели
    model = Sequential()
    model.add(Dense(200, activation='elu', input_shape=(len(index_list),)))
    model.add(Dense(10, activation='softmax'))

    model.compile(optimizer=Adam(lr=1e-4),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    # Обучение модели на подмножестве признаков
    history = model.fit(x_train[:, index_list],
                        y_train,
                        epochs=epochs,
                        batch_size=batch_size,
                        verbose=verbose,
                        validation_data=(x_val[:, index_list], y_val))

    # Вывод графика точности распознавания на обучающей и проверочной выборках
    plt.figure(figsize=(12,5))
    plt.plot(history.history['accuracy'], label='Точность на обучающем наборе')
    plt.plot(history.history['val_accuracy'], label='Точность на проверочном наборе')
    plt.xticks(range(0, epochs, 10))
    plt.xlabel('Эпоха')
    plt.ylabel('Точность')
    plt.legend()
    plt.show()
    print('Точность в конце обучения:',
          history.history['accuracy'][-1],
          history.history['val_accuracy'][-1])

    return model

```

Рисунок 19. Функция создания и обучения упрощенной архитектуры классификатора

Затем происходит испытание малой модели с использованием функции `get_features()` признаки добавляются в следующем порядке индексов:

- 0: 'rmse' - Среднеквадратическая амплитуда;
- 1: 'spct' - Спектральный центроид;
- 2: 'spbw' - Ширина полосы частот;
- 3: 'roff' - Спектральный спад частоты;
- 4: 'zcr' - Пересечения нуля;
- 5-24: 'mfcc' - Мел-кепстральные коэффициенты (20 признаков);
- 25-36: 'stft' - Хромограмма (12 признаков).

Рассмотрим один из примеров обучения малой модели (рис. 20).

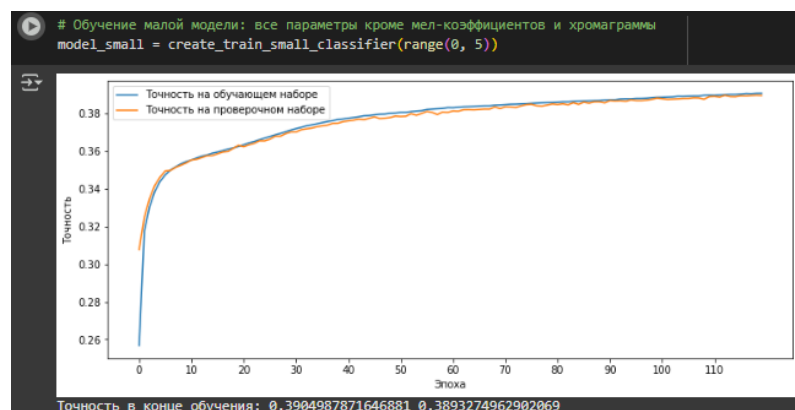


Рисунок 20. Обучение малой модели

Затем проделываются все те же действия только для новой архитектуры с softmax (процесс выполнения можно посмотреть в репозитории).

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 (рис. 21).

```
[26] # Импорт библиотек
import numpy as np

# Отрисовка графиков
import matplotlib.pyplot as plt

# Загрузка из google облака
import gdown

# Преобразование категориальных данных в one hot encoding
from tensorflow.keras.utils import to_categorical

# Работа с папками и файлами
import os

# Утилиты работы со временем
import time

# Работа со случайными числами
import random

# Математические функции
import math

# Сохранение и загрузка структур данных Python
import pickle

# Параметризация аудио
import librosa

# Оптимизаторы для обучения моделей
from tensorflow.keras.optimizers import Adam, RMSprop

# Конструирование и загрузка моделей нейронных сетей
from tensorflow.keras.models import Sequential, Model, load_model

# Основные слои
from tensorflow.keras.layers import concatenate, Input, Dense, Dropout, BatchNormalization, Flatten, Conv1D, Conv2D, LSTM

# Разделение на обучающую и проверочную выборку
from sklearn.model_selection import train_test_split

# Кодирование категориальных меток, нормирование числовых данных
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Матрица ошибок классификатора
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

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

%matplotlib inline
```

Рисунок 21. Импорт библиотек

Далее выполним загрузку датасета и подготовленных данных (рис. 22).

```
[27] # Загрузка датасета из облака
gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/112/genres.zip', None, quiet=False)

# Загрузка подготовленных данных
gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/112/audio_data_mean.pickle', None, quiet=True)

Downloading...
From: https://storage.yandexcloud.net/aiueducation/Content/base/112/genres.zip
To: /content/genres.zip
100%[██████████] 1.23G/1.23G [01:50<00:00, 11.1MB/s]
'audio_data_mean.pickle'
```

Рисунок 22. Загрузка данных

Затем выполняем распаковку архива на локальный диск, а также просмотр выгруженных папок и содержимое одной из них (рис. 23).

```
[28] # Распаковка архива на локальный диск colab
!unzip -qo genres.zip

# Проверка выгруженных папок
!ls genres

# Проверка содержимого одной папки
!ls genres/blues
```

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

Рисунок 23. Проверка содержимого одной из папок

Далее выполняется установка констант и вывод списка классов (рис. 24).

```
[29] # Установка констант

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

[30] # Проверка списка классов
print(CLASS_LIST)
```

```
['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'pop', 'reggae', 'rock']
```

Рисунок 24. Установка констант

Затем выполняется функция параметризации аудио (рис. 25).

```
[31] # 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-кепстральные 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),
               'sptc': 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 25. Функция параметризации аудио

Далее напишем функцию объединения признаков в набор векторов (рис.

26).

```

[32] # 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 26. Функция объединения признаков в набор векторов

Потом выполним функцию формирования набора признаков и метки класса для аудиофайла (рис. 27).

```

[33] # Функция 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 27. Функция формирования набора признаков для аудиофайла

Затем напишем функцию формирования подвыборки признаков и меток класса для одного файла (рис. 28).

```

[34] # 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 28. Функция формирования подвыборки признаков для одного файла

Далее выполним последнюю функцию формирования набора данных из файлов всех классов по диапазону номеров файлов (рис. 29).

```

[35] # 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 29. Функция формирования набора данных по диапазону номеров файлов

Затем выполняется восстановление датасета аудио (рис. 30).

```

[36] # 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 30. Восстановление датасета аудио

Далее реализуется нормирование признаков в соответствии со стандартным нормальным распределением и разделение набора данных на обучающую и проверочную выборки (рис. 31).

```

[37] # Нормирование признаков в соответствии со стандартным нормальным распределением

x_scaler = StandardScaler()
x_train_data_scaled = x_scaler.fit_transform(x_train_data)

[38] # Разделение набора данных на обучающую и проверочную выборки
# Параметр stratify указывает метки классов, по которым происходит балансировка разделения

x_train, x_val, y_train, y_val = train_test_split(x_train_data_scaled,
                                                  y_train_data,
                                                  stratify=y_train_data,
                                                  test_size=VALIDATION_SPLIT)

```

Рисунок 31. Разделение набора данных на обучающую и проверочную выборки

Затем выполним функцию вывода графиков точности и ошибки распознавания на обучающей и проверочной выборках (рис. 32).

```

[39] # Вывод графиков точности и ошибки распознавания на обучающей и проверочной выборках

def show_history(history # объект-результат метода обучения .fit()
):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 5))
    fig.suptitle('График процесса обучения модели')
    ax1.plot(history.history['accuracy'],
              label='Доля верных ответов на обучающем наборе')
    ax1.plot(history.history['val_accuracy'],
              label='Доля верных ответов на проверочном наборе')
    ax1.xaxis.get_major_locator().set_params(integer=True)
    ax1.set_xlabel('Эпоха обучения')
    ax1.set_ylabel('Доля верных ответов')
    ax1.legend()

    ax2.plot(history.history['loss'],
              label='Ошибка на обучающем наборе')
    ax2.plot(history.history['val_loss'],
              label='Ошибка на проверочном наборе')
    ax2.xaxis.get_major_locator().set_params(integer=True)
    ax2.set_xlabel('Эпоха обучения')
    ax2.set_ylabel('Ошибка')
    ax2.legend()
    plt.show()

```

Рисунок 32. Функция вывода графиков точности и ошибки

Далее напишем функцию для классификации звукового файла и визуализации предсказания модели для него (рис. 33).

```

[40] # 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], '-', 'БЕРНО :-)' if predict_good else 'НЕБЕРНО.')
    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 33. Функция классификации файла и визуализация предсказания модели

Также напишем функцию классификации и визуализации для нескольких файлов каждого класса (рис. 34).


```

[41] # 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 штуки
    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 34. 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 (рис. 35).

```

[42] 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 35. Вывод форм обучающей и проверочной выборок

Задание 2. Необходимо составить модель классификатора на полносвязных слоях и сохранить ее в переменной model (рис. 36). 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'.

```
[43] from tensorflow.keras.models import Sequential
      from tensorflow.keras.layers import Dense, Dropout, BatchNormalization

      model = Sequential([
          Dense(64, activation='relu', input_shape=(x_train.shape[1,])),
          Dropout(0.3),
          Dense(32, activation='relu'),
          Dropout(0.3),
          Dense(16, activation='relu'),
          Dropout(0.2),
          BatchNormalization(),
          Dense(CLASS_COUNT, activation='softmax')
      ])
```

Рисунок 36. 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() (рис. 37).

```
[44] model.compile(optimizer=Adam(learning_rate=0.0001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
      model.summary()
```

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 37. 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` (рис. 38).

- 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.

```
[45] history = model.fit(x_train, y_train,
                        validation_data=(x_val, y_val),
                        batch_size=32,
                        epochs=1000,
                        verbose=1)

show_history(history)
Epoch 964/1000
```

Рисунок 38. Обучение модели

Далее посмотрим на результат обучения и на построенные графики процесса обучения (рис. 39).

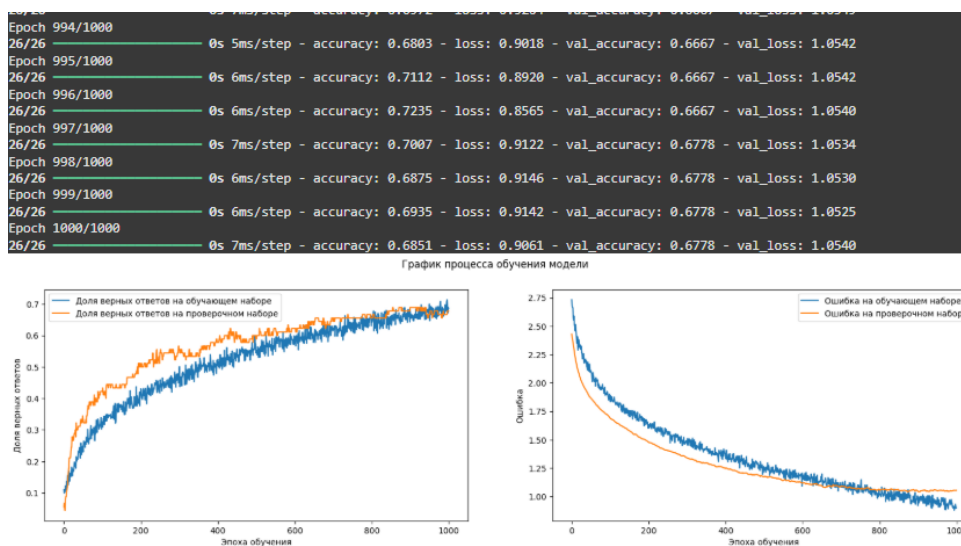


Рисунок 39. Графики процесса обучения модели

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 (рис. 40), 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 40. Команда для вывода визуализации работы классификатора

После посмотрим на графики среднего распределения векторов предсказания (рис. 41), все полученные графики можно посмотреть в репозитории.

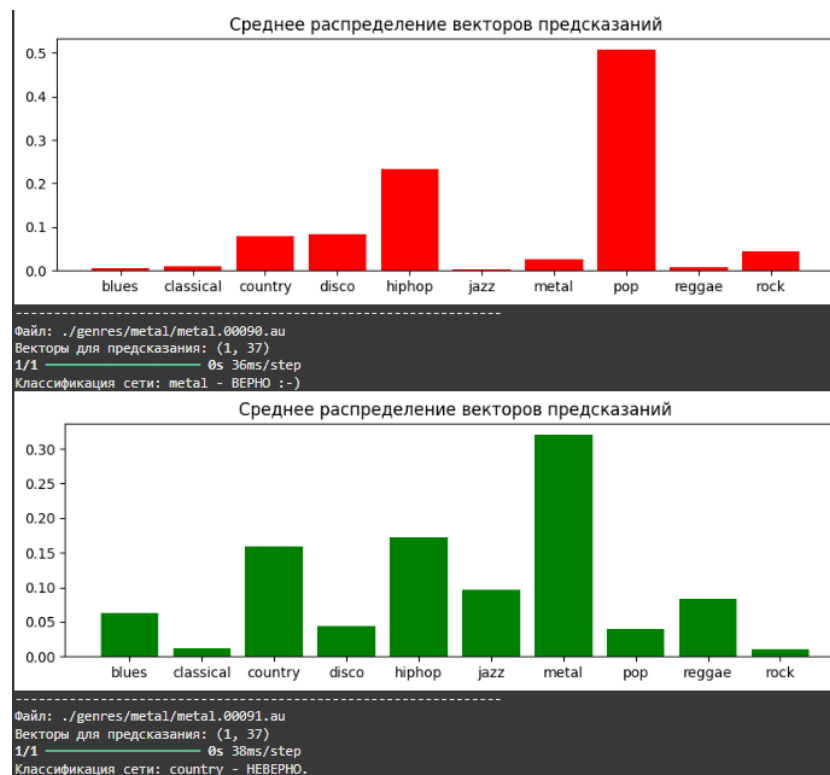


Рисунок 41. Среднее распределение векторов предсказаний

Далее посмотрим на полученную матрицу ошибок по файлам аудио (рис. 42).

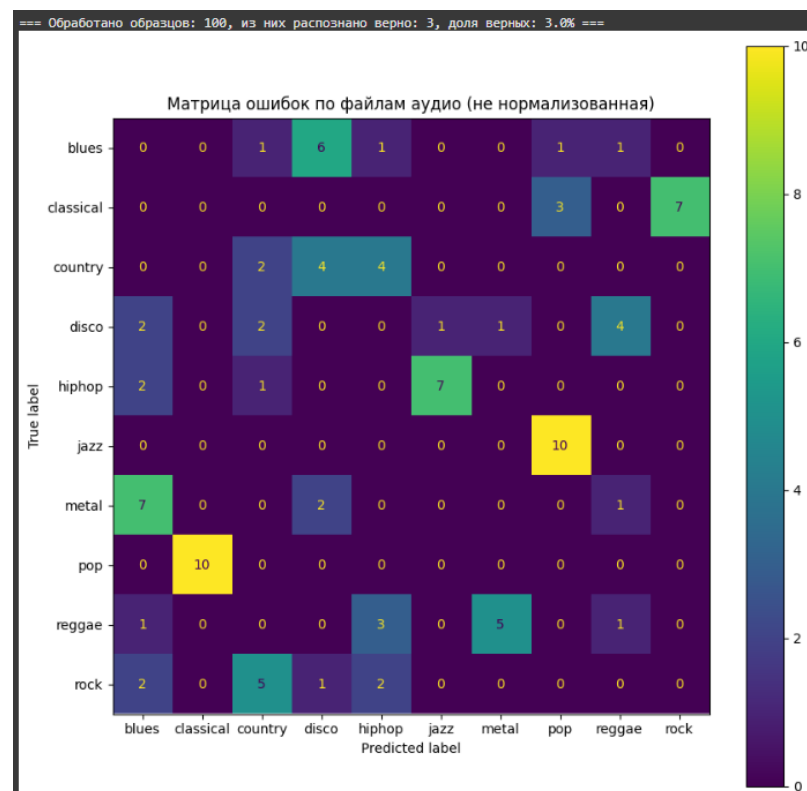


Рисунок 42. Матрица ошибок по файлам аудио

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 (рис. 43).

```
[1] # Массивы
import numpy as np

# Отрисовка графиков
import matplotlib.pyplot as plt

# Загрузка из google облака
import gdown

# Преобразование категориальных данных в one hot encoding
from tensorflow.keras.utils import to_categorical

# Работа с папками и файлами
import os

# Утилиты работы со временем
import time

# Работа со случайными числами
import random

# Математические функции
import math

# Сохранение и загрузка структур данных Python
import pickle

# Параметризация аудио
import librosa

# Оптимизаторы для обучения моделей
from tensorflow.keras.optimizers import Adam, RMSprop

# Конструирование и загрузка моделей нейронных сетей
from tensorflow.keras.models import Sequential, Model, load_model

# Основные слои
from tensorflow.keras.layers import concatenate, Input, Dense, Dropout, BatchNormalization, Flatten, Conv1D, Conv2D, LSTM
from tensorflow.keras.layers import MaxPooling1D, AveragePooling1D, SpatialDropout1D

# Разделение на обучающую и проверочную выборку
from sklearn.model_selection import train_test_split

# Кодирование категориальных меток, нормирование числовых данных
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

Рисунок 43. Импорт библиотек

Далее выполним загрузку датасета из облака и его распаковку, а также выполним проверку выгруженных папок и содержимого одной папки (рис. 45).

```
[2] # Загрузка датасета из облака
gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/112/genres.zip', None, quiet=True)

'genres.zip'

[3] # Распаковка архива на локальный диск colab
!unzip -qo genres.zip

# Проверка выгруженных папок
!ls genres

# Проверка содержимого одной папки
!ls genres/blues

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

Рисунок 44. Загрузка датасета и распаковка

Затем зададим параметры, как сказано по заданию (рис. 45) (Берем отрезок фиксированной длины (например, 5с) и получаем набор признаков для этого отрезка. Смещаемся на шаг (например, 1с).

```
[4] # 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 45. 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 (рис. 46).

```
[5] # 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 46. Вывод размеров массивов признаков и меток

Далее выполним кодировку меток жанров и масштабирование признаков, а также разделение данных на обучающую и тестовую выборки (рис. 47).

```
[6] # 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)

[7] # 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 47. Разделение данных на обучающую и тестовую выборки

Затем создадим модель нейронной сети для выполнения задания, то есть разработаем классификатор на одномерных сверточных слоях Conv1D (рис. 48).

```
[18] # Создаем последовательную модель
model = Sequential([
    Conv1D(64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])),
    BatchNormalization(),
    MaxPooling1D(pool_size=2),
    Dropout(0.2),

    Conv1D(128, kernel_size=3, activation='relu'),
    BatchNormalization(),
    MaxPooling1D(pool_size=2),
    Dropout(0.2),

    # Полносвязный слой: 128 нейронов
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.2),
    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, 216, 64)	2,560
batch_normalization_4 (BatchNormalization)	(None, 216, 64)	256
max_pooling1d_4 (MaxPooling1D)	(None, 108, 64)	0
dropout_6 (Dropout)	(None, 108, 64)	0
conv1d_5 (Conv1D)	(None, 108, 128)	36,704
batch_normalization_5 (BatchNormalization)	(None, 108, 128)	512
max_pooling1d_5 (MaxPooling1D)	(None, 54, 128)	0
dropout_7 (Dropout)	(None, 54, 128)	0
flatten_2 (Flatten)	(None, 6630)	0
dense_4 (Dense)	(None, 128)	852,800
dropout_8 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 10)	1,300

Total params: 891,414 (3.36 MB)
Trainable params: 891,414 (3.36 MB)
Non-trainable params: 0 (1.50 KB)

Рисунок 48. Архитектура модели

Далее выполним обучение созданной модели (рис. 49).

```
# Обучение модели
history = model.fit(X_train,
                    y_train,
                    validation_data=(X_test, y_test),
                    epochs=200,
                    batch_size=32,
                    verbose=1)
```

Epoch 172/200

65s/step - accuracy: 0.9873 - loss: 0.0435 - val_accuracy: 0.9288 - val_loss: 0.3145

Epoch 173/200

65s/step - accuracy: 0.9889 - loss: 0.0372 - val_accuracy: 0.9286 - val_loss: 0.2686

Epoch 174/200

65s/step - accuracy: 0.9894 - loss: 0.0345 - val_accuracy: 0.9300 - val_loss: 0.2754

Epoch 175/200

65s/step - accuracy: 0.9891 - loss: 0.0346 - val_accuracy: 0.9248 - val_loss: 0.3118

Epoch 176/200

65s/step - accuracy: 0.9892 - loss: 0.0363 - val_accuracy: 0.9321 - val_loss: 0.2825

Epoch 177/200

65s/step - accuracy: 0.9898 - loss: 0.0355 - val_accuracy: 0.9294 - val_loss: 0.2862

Epoch 178/200

65s/step - accuracy: 0.9882 - loss: 0.0396 - val_accuracy: 0.9340 - val_loss: 0.2925

Epoch 179/200

65s/step - accuracy: 0.9896 - loss: 0.0314 - val_accuracy: 0.9334 - val_loss: 0.2736

Epoch 180/200

65s/step - accuracy: 0.9909 - loss: 0.0265 - val_accuracy: 0.9304 - val_loss: 0.2922

Epoch 181/200

65s/step - accuracy: 0.9904 - loss: 0.0315 - val_accuracy: 0.9350 - val_loss: 0.2878

Epoch 182/200

65s/step - accuracy: 0.9916 - loss: 0.0278 - val_accuracy: 0.9263 - val_loss: 0.3198

Epoch 183/200

65s/step - accuracy: 0.9916 - loss: 0.0278 - val_accuracy: 0.9263 - val_loss: 0.3198

Epoch 184/200

65s/step - accuracy: 0.9893 - loss: 0.0411 - val_accuracy: 0.9248 - val_loss: 0.3200

Epoch 185/200

65s/step - accuracy: 0.9875 - loss: 0.0392 - val_accuracy: 0.9329 - val_loss: 0.3012

Epoch 186/200

65s/step - accuracy: 0.9900 - loss: 0.0349 - val_accuracy: 0.9280 - val_loss: 0.3280

Epoch 187/200

65s/step - accuracy: 0.9909 - loss: 0.0312 - val_accuracy: 0.9352 - val_loss: 0.2820

Epoch 188/200

65s/step - accuracy: 0.9908 - loss: 0.0313 - val_accuracy: 0.9327 - val_loss: 0.3466

Epoch 189/200

65s/step - accuracy: 0.9898 - loss: 0.0359 - val_accuracy: 0.9213 - val_loss: 0.3340

Epoch 190/200

65s/step - accuracy: 0.9908 - loss: 0.0266 - val_accuracy: 0.9279 - val_loss: 0.3552

Epoch 191/200

65s/step - accuracy: 0.9898 - loss: 0.0408 - val_accuracy: 0.9315 - val_loss: 0.2885

Epoch 192/200

65s/step - accuracy: 0.9868 - loss: 0.0452 - val_accuracy: 0.9321 - val_loss: 0.2895

Epoch 193/200

65s/step - accuracy: 0.9899 - loss: 0.0360 - val_accuracy: 0.9352 - val_loss: 0.2704

Epoch 194/200

65s/step - accuracy: 0.9883 - loss: 0.0398 - val_accuracy: 0.9294 - val_loss: 0.3211

Epoch 195/200

65s/step - accuracy: 0.9893 - loss: 0.0369 - val_accuracy: 0.9336 - val_loss: 0.3024

Epoch 196/200

65s/step - accuracy: 0.9899 - loss: 0.0320 - val_accuracy: 0.9313 - val_loss: 0.2861

Epoch 197/200

65s/step - accuracy: 0.9908 - loss: 0.0346 - val_accuracy: 0.9271 - val_loss: 0.3587

Epoch 198/200

65s/step - accuracy: 0.9914 - loss: 0.0253 - val_accuracy: 0.9306 - val_loss: 0.3465

Epoch 199/200

65s/step - accuracy: 0.9870 - loss: 0.0450 - val_accuracy: 0.9316 - val_loss: 0.3583

Epoch 200/200

65s/step - accuracy: 0.9902 - loss: 0.0323 - val_accuracy: 0.9238 - val_loss: 0.3792

Рисунок 49. Обучение модели

Построим график процесса обучения модели (рис. 50).

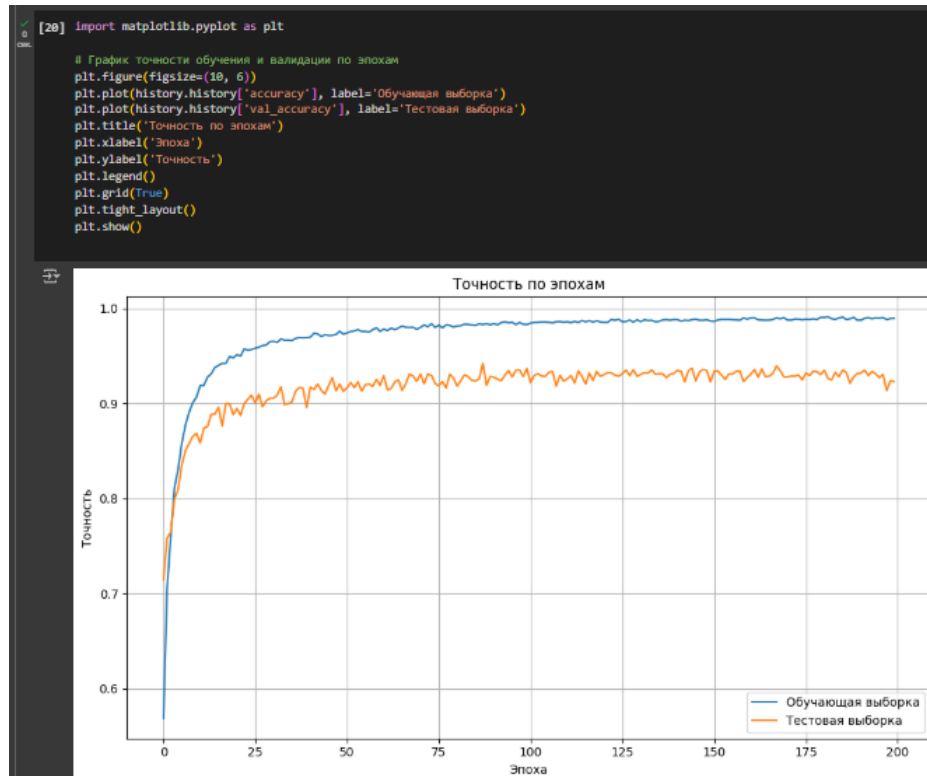


Рисунок 50. График процесса обучения

Затем выполним проверку точности на обучающей и тестовой выборках (рис. 51)

```
[21] # 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%
```

Рисунок 51. 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 (рис. 52).

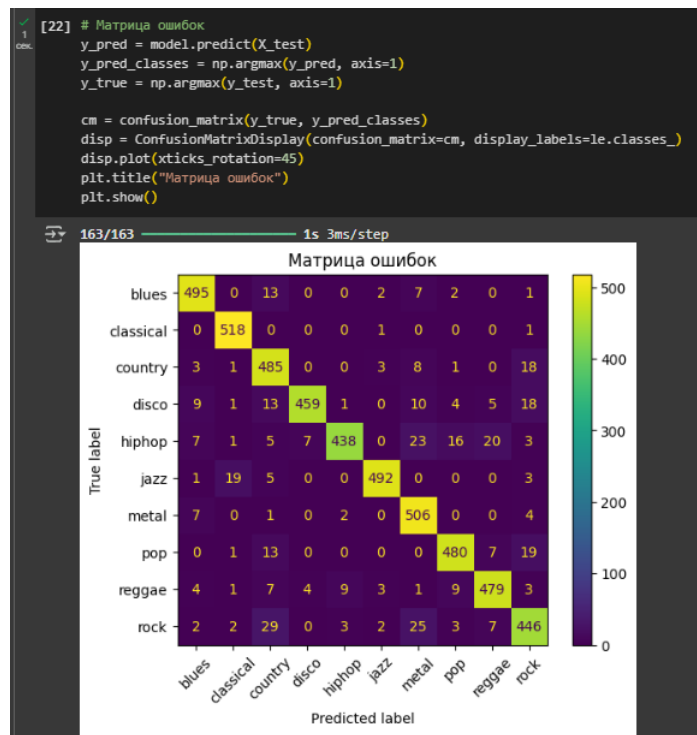


Рисунок 52. Матрица ошибок

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

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/112/archive.zip>

6. Прогнать обученный классификатор на файлах из датасета SAVEE.

7. Сделать выводы.

Для выполнения данного задания выполним импорт необходимых библиотек (рис. 53).

```
[28] import gdown
import os
import librosa
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import random
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report, accuracy_score
```

Рисунок 53. Импорт библиотек

Далее выполним загрузку датасета образцов эмоциональной речи Toronto emotional speech set (TESS) (рис. 54).

```
[29] # Скачиваем файл dataverse_files.zip по ссылке с помощью gdown
!gdown https://storage.yandexcloud.net/aiueducation/Content/base/112/dataverse_files.zip

!unzip -qo dataverse_files.zip

Downloading...
From: https://storage.yandexcloud.net/aiueducation/Content/base/112/dataverse_files.zip
To: /content/dataverse_files.zip
100% 224M/224M [00:19<00:00, 11.4MB/s]
```

Рисунок 54. Загрузка датасета

Затем разберем датасет, зададим словарь соответствия эмоциям и выполним преобразование списков в массивы numpy (рис. 55).

```
[30] 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')]

# Обработаем 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 55. Преобразование списков в массивы numpy

Далее разделим данные на обучающие и тестовые выборки (рис. 56).

```
[31] # 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
      )
```

Рисунок 56. Разбиение данных на обучающую и тестовую выборки

Выполним масштабирование признаков и One-hot кодирование меток (рис. 57).

```
[32] # 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 57. Масштабирование признаков

Далее выполним создание модели нейронной сети и компиляцию модели (рис. 58).

```
[33] # Создаем последовательную нейронную сеть
      model = Sequential([
          Dense(256, activation='relu', input_shape=(X_train_scaled.shape[1],)),
          Dropout(0.3),
          Dense(128, activation='relu'),
          Dropout(0.3),
          Dense(64, activation='relu'),
          Dense(7, activation='softmax')
      ])

      # Компилируем модель
      model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
      model.summary()
```

/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass `super().__init__(activity_regularizer=activity_regularizer, **kwargs)` to the constructor of the subclass. Instead, use `super().__init__(**kwargs)` and set `activity_regularizer` as an attribute of the layer.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 256)	18,496
dropout_2 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 128)	32,896
dropout_3 (Dropout)	(None, 128)	0
dense_6 (Dense)	(None, 64)	8,256
dense_7 (Dense)	(None, 7)	495

Total params: 52,103 (203.53 KB)
Trainable params: 52,103 (203.53 KB)
Non-trainable params: 0 (0.00 B)

Рисунок 58. Архитектура модели

После выполним обучение созданной модели (рис. 59).

```
8 в Обсуждение модели
9 history = model.fit(
10     X_train_scaled, y_train_cat,
11     epochs=50,
12     batch_size=32,
13     validation_data=(X_test_scaled, y_test_cat),
14     verbose=1
15 )
```



```
EPOCH 22/50      0s 3ms/step - accuracy: 0.9939 - loss: 0.0118 - val_accuracy: 1.0000 - val_loss: 6.9253e-04
EPOCH 23/50      0s 4ms/step - accuracy: 0.9999 - loss: 0.0017 - val_accuracy: 1.0000 - val_loss: 9.3891e-04
EPOCH 24/50      0s 3ms/step - accuracy: 0.9992 - loss: 0.0034 - val_accuracy: 1.0000 - val_loss: 4.6074e-04
EPOCH 25/50      0s 3ms/step - accuracy: 0.9995 - loss: 0.0017 - val_accuracy: 1.0000 - val_loss: 6.8855e-04
EPOCH 26/50      0s 3ms/step - accuracy: 1.0000 - loss: 0.0013 - val_accuracy: 1.0000 - val_loss: 9.8769e-04
EPOCH 27/50      0s 4ms/step - accuracy: 0.9997 - loss: 0.0021 - val_accuracy: 0.9982 - val_loss: 0.0024
EPOCH 28/50      0s 3ms/step - accuracy: 0.9996 - loss: 0.0014 - val_accuracy: 1.0000 - val_loss: 0.0014
EPOCH 29/50      0s 3ms/step - accuracy: 0.9983 - loss: 0.0027 - val_accuracy: 1.0000 - val_loss: 0.0569e-04
EPOCH 30/50      0s 4ms/step - accuracy: 0.9996 - loss: 0.0027 - val_accuracy: 1.0000 - val_loss: 0.0014
EPOCH 31/50      0s 4ms/step - accuracy: 0.9987 - loss: 0.0039 - val_accuracy: 1.0000 - val_loss: 0.0010
EPOCH 32/50      0s 3ms/step - accuracy: 1.0000 - loss: 8.7088e-04 - val_accuracy: 1.0000 - val_loss: 0.0026
EPOCH 33/50      0s 3ms/step - accuracy: 0.9993 - loss: 0.0015 - val_accuracy: 1.0000 - val_loss: 7.3136e-04
EPOCH 34/50      0s 5ms/step - accuracy: 0.9996 - loss: 0.0016 - val_accuracy: 0.9964 - val_loss: 0.0053
EPOCH 35/50      0s 4ms/step - accuracy: 1.0000 - loss: 9.2673e-04 - val_accuracy: 1.0000 - val_loss: 8.7886e-04
EPOCH 36/50      0s 4ms/step - accuracy: 1.0000 - loss: 7.7289e-04 - val_accuracy: 1.0000 - val_loss: 7.9599e-04
EPOCH 37/50      0s 4ms/step - accuracy: 0.9984 - loss: 0.0020 - val_accuracy: 0.9982 - val_loss: 0.0044
EPOCH 38/50      0s 4ms/step - accuracy: 0.9980 - loss: 0.0063 - val_accuracy: 1.0000 - val_loss: 6.4309e-04
EPOCH 39/50      0s 3ms/step - accuracy: 0.9994 - loss: 0.0017 - val_accuracy: 1.0000 - val_loss: 3.1343e-04
EPOCH 40/50      0s 3ms/step - accuracy: 1.0000 - loss: 4.7493e-04 - val_accuracy: 1.0000 - val_loss: 2.8087e-04
EPOCH 41/50      0s 5ms/step - accuracy: 0.9998 - loss: 6.7083e-04 - val_accuracy: 1.0000 - val_loss: 6.3327e-04
EPOCH 42/50      1s 6ms/step - accuracy: 0.9992 - loss: 0.0078 - val_accuracy: 1.0000 - val_loss: 2.2724e-04
EPOCH 43/50      1s 6ms/step - accuracy: 1.0000 - loss: 7.8954e-04 - val_accuracy: 1.0000 - val_loss: 8.1897e-04
EPOCH 44/50      0s 4ms/step - accuracy: 0.9970 - loss: 0.0103 - val_accuracy: 1.0000 - val_loss: 0.0012
EPOCH 45/50      1s 4ms/step - accuracy: 1.0000 - loss: 0.0012 - val_accuracy: 1.0000 - val_loss: 7.1927e-04
EPOCH 46/50      0s 3ms/step - accuracy: 1.0000 - loss: 2.2647e-04 - val_accuracy: 1.0000 - val_loss: 0.0018
EPOCH 47/50      0s 3ms/step - accuracy: 1.0000 - loss: 8.4416e-04 - val_accuracy: 0.9982 - val_loss: 0.0017
EPOCH 48/50      0s 4ms/step - accuracy: 1.0000 - loss: 3.4383e-04 - val_accuracy: 1.0000 - val_loss: 0.0013
EPOCH 49/50      0s 3ms/step - accuracy: 0.9987 - loss: 0.0050 - val_accuracy: 0.9982 - val_loss: 0.0044
EPOCH 50/50      0s 3ms/step - accuracy: 0.9983 - loss: 0.0077 - val_accuracy: 0.9929 - val_loss: 0.0268
```

Рисунок 59. Обучение модели

Затем определим точность на тесте и оценку качества (рис. 60).

```
[38] from sklearn.metrics import confusion_matrix
import seaborn as sns

# 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

	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 60. 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%.

Далее построим график точности на обучающей выборке по эпохам и график для отображения функции потерь (рис. 61).

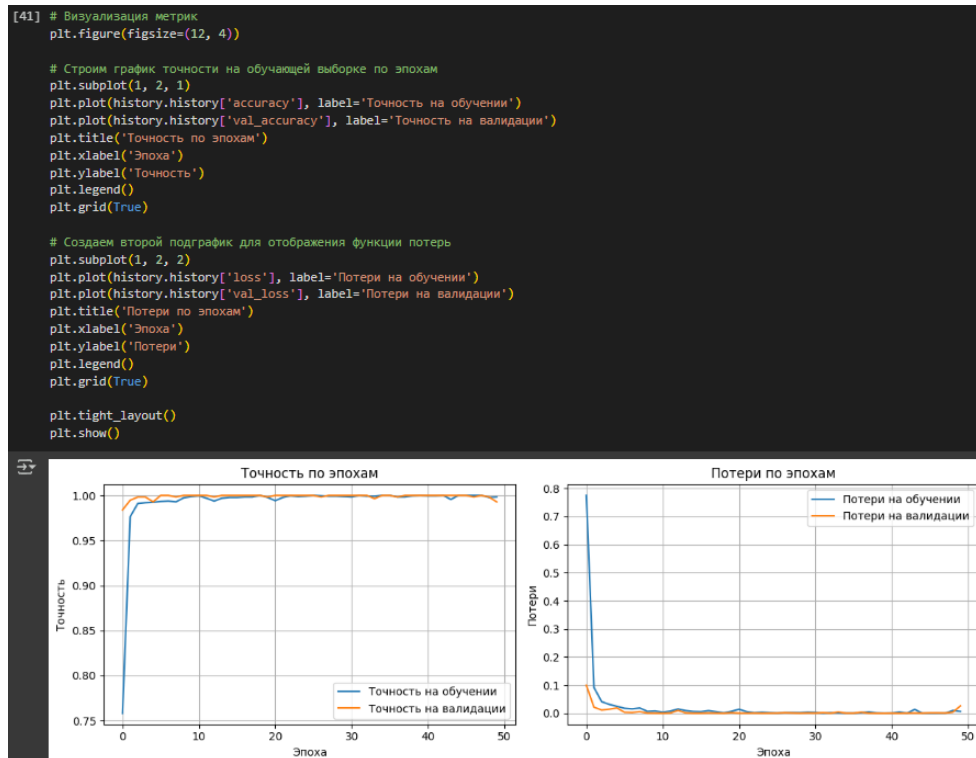


Рисунок 61. График точности на обучающей выборке

Также построим матрицу ошибок (рис. 62).

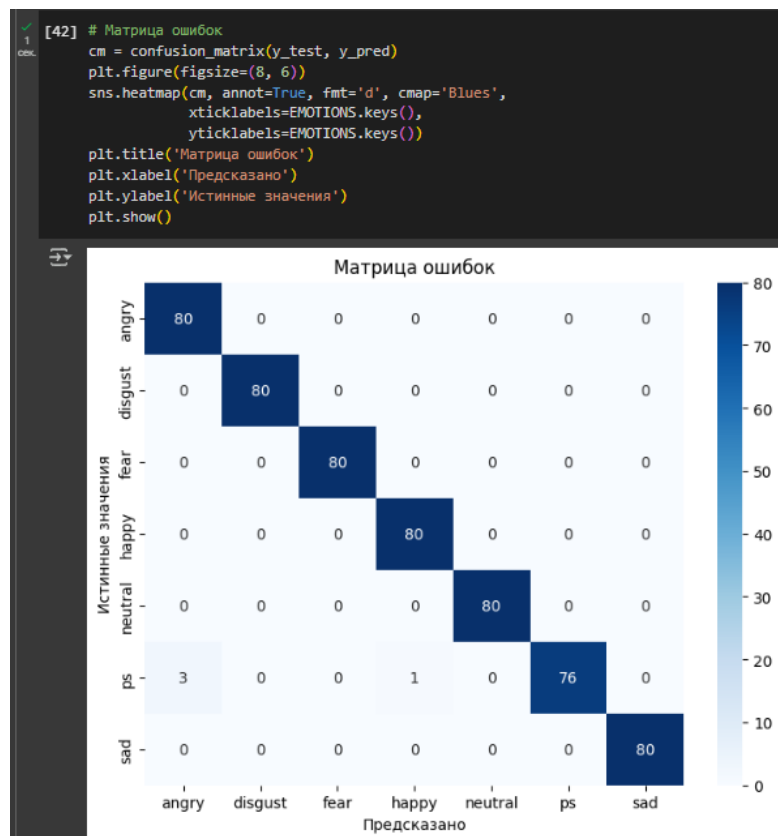


Рисунок 62. Матрица ошибок

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 (рис. 63).

```
import zipfile

# Распаковка архива
!wget https://storage.yandexcloud.net/aiueducation/Content/base/112/archive.zip
!unzip -qo 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 63. Загрузка датасета и распаковка архива

Далее выполним загрузку и обработку файлов из SAVEE (рис. 64).

```
[45] import re

X_savee = [] # Список для признаков (MFCC) аудиофайлов
y_savee = [] # Список для меток (эмоций)

# Проходим по всем файлам в директории с аудиоданными SAVEE
for fname in os.listdir(savee_dir):
    if not fname.endswith('.wav'):
        continue

    # Поиск кода эмоции между "_" и числом
    match = re.search(r'_([a-z]+)\d+', fname.lower())
    if not match:
        continue

    emo_code = match.group(1)

    if emo_code not in EMO_MAP:
        continue

    label = EMO_MAP[emo_code]
    path = os.path.join(savee_dir, fname)

    try:
        audio, sr = librosa.load(path, sr=22050)
        mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=40)
        mfcc_mean = np.mean(mfcc.T, axis=0)

        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")
```

Рисунок 64. Обработка данных

Будем использовать тот же scaler, что обучался на TESS и выполним предсказание (рис. 65).

```
[46] # 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)
```

Рисунок 65. Выполнение предсказания

Далее вычислим точность предсказаний модели на данных SAVEE и выведем отчет по классификации модели на данных SAVEE (рис. 66). А также построим матрицу ошибок (рис. 67).

```
[48] # 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 66. Вычисление точности предсказания

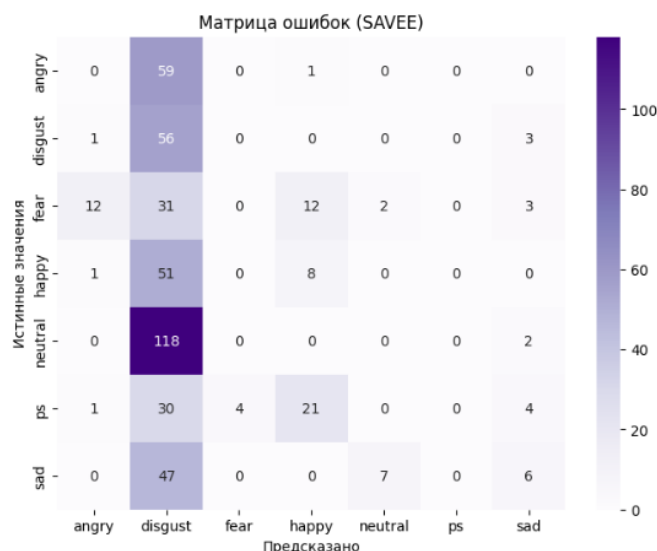


Рисунок 67. Матрица ошибок

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.

Ссылка на гитхаб с файлами: <https://github.com/EvgenyEvdakov/NS-8>

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