САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО
Институт Компьютерных наук и кибербезопасности
Высшая школа искусственного интеллекта
Направление 02.04.01 Математика и компьютерные науки

Отчёт по по дисциплине «Промышленное программирование на языке python»

Курсовая работа

Группа: 5140201/40302

Студент: _____ Губарев Артём Владимирович

Преподаватель: _____ Вениаминов Николай Андреевич

«____»_____ 20__г.

Санкт-Петербург – 2025

Содержание

1 Постановка задачи

Целью курсового проекта является разработка приложения для классификации музыкальных треков по жанрам с использованием методов машинного обучения. Пользователь предоставляет аудиофайл, система обрабатывает его, извлекает звуковые признаки и определяет жанр композиции.

Для выполнения поставленной задачи необходимо выполнить следующие шаги:

1. Выбрать метод машинного обучения, наиболее подходящий для классификации музыкальных треков по жанрам;
2. Подготовить обучающую выборку и провести обучение модели на основе выбранного метода;
3. Разработать пользовательский интерфейс, обеспечивающий удобную подачу аудиофайлов и отображение результатов классификации;
4. Собрать приложение и развернуть его на хостинге для обеспечения доступа конечным пользователям.

2 Метод и обучение модели

2.1 Выбор метода

Для решения задачи классификации музыкальных жанров была выбрана сверточная нейронная сеть (CNN). Это обосновано тем, что аудиофайлы предварительно преобразуются в двумерные матрицы признаков — мел-частотные кепстральные коэффициенты (MFCC), которые по своей структуре аналогичны изображениям. CNN эффективно выявляют локальные паттерны и зависимости в таких данных, что делает их особенно подходящими для аудиоклассификации.

Кроме того, по сравнению с полносвязными архитектурами, сверточные сети обладают меньшим числом обучаемых параметров, что снижает риск переобучения и ускоряет обучение модели.

2.2 Подготовка обучающей выборки

В качестве обучающего датасета использовалась стандартная коллекция GTZAN, включающая 100 аудиофайлов продолжительностью 30 секунд для каждого из 10 жанров (всего 1000 треков). Для увеличения объема выборки и повышения устойчивости модели, каждый трек разбивался на 10 равных сегментов по 3 секунды.

Из каждого сегмента извлекались MFCC-признаки с использованием библиотеки `librosa` при следующих параметрах:

- количество коэффициентов: 13,
- длина окна: 2048 отсчётов,
- шаг окна: 512 отсчётов.

В результате каждый сегмент представлялся в виде матрицы размера $(130, 13)$, где 130 — количество временных кадров. Затем данные были нормализованы и преобразованы к виду $(\text{samples}, 130, 13, 1)$ для подачи на вход CNN.

2.3 Архитектура модели

Архитектура модели представлена в таблице:

#	Слой	Тип	Параметры	Выходная форма
1	Входной	—	input_shape = (130, 13, 1)	(130, 13, 1)
2	Conv2D	Сверточный	32 фильтра, ядро 3×3, activation='relu'	(128, 11, 32)
3	MaxPooling2D	Подвыборка	окно 3×3, шаг 2×2, padding='same'	(64, 6, 32)
4	BatchNormalization	Нормализация	—	(64, 6, 32)
5	Conv2D	Сверточный	32 фильтра, ядро 3×3, activation='relu'	(62, 4, 32)
6	MaxPooling2D	Подвыборка	окно 3×3, шаг 2×2, padding='same'	(31, 2, 32)
7	BatchNormalization	Нормализация	—	(31, 2, 32)
8	Conv2D	Сверточный	32 фильтра, ядро 2×2, activation='relu'	(30, 1, 32)
9	MaxPooling2D	Подвыборка	окно 2×2, шаг 2×2, padding='same'	(15, 1, 32)
10	BatchNormalization	Нормализация	—	(15, 1, 32)
11	Flatten	Векторизация	—	(480,)
12	Dense	Полносвязный	64 нейрона, activation='relu'	(64,)
13	Dropout	Отсев	dropout rate = 0.1	(64,)
14	Dense	Выходной	10 нейронов, activation='softmax'	(10,)

Таблица 1: Архитектура сверточной нейронной сети для классификации музыкальных жанров

2.4 Обучение

Модель компилировалась с использованием оптимизатора `Adam` с параметром learning rate = 0.0001. В качестве функции потерь использовалась `sparse_categorical_crossentropy`, поскольку метки классов представлены в виде целых чисел (0–9).

Обучение проводилось на протяжении 100 эпох с размером батча 32, с валидацией на отдельной выборке.

Итоговая точность на валидационной выборке составила 0.7276. Такая точность считается приемлемой, так как предсказание будет проводиться не по одному сегменту, а с агрегацией результатов по нескольким сегментам аудиофайла.

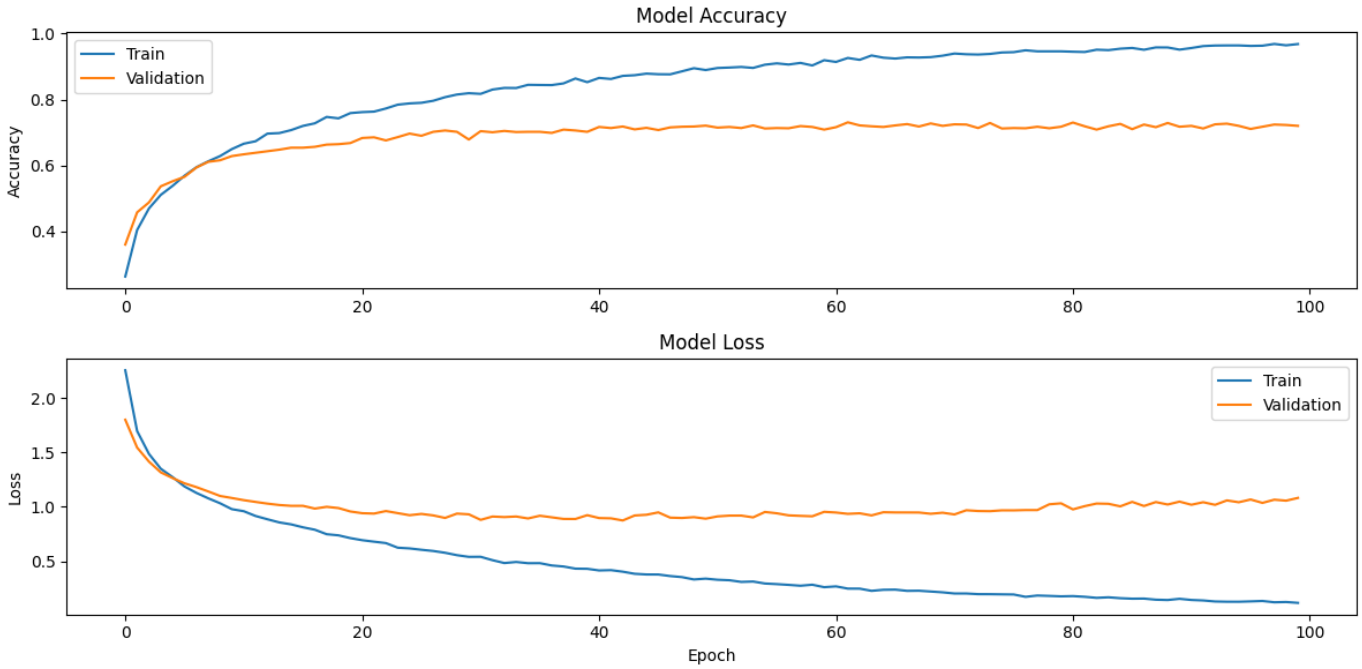

Рис. 1: Графики точности и потерь модели на этапах обучения и валидации

3 Алгоритм предсказания жанра

Процесс предсказания жанра для нового аудиофайла включает следующие этапы:

1. Пользователь передаёт аудиофайл.
2. Аудио преобразуется в формат моно WAV с частотой дискретизации 22050 Гц с помощью `FFmpeg` или `librosa`.
3. Полученный файл разбивается на 10 равных сегментов.
4. К каждому сегменту применяется функция извлечения признаков `extract_features()`, аналогичная используемой при обучении.
5. Каждый сегмент подаётся на вход обученной CNN-модели.
6. Модель возвращает распределение вероятностей по жанрам (softmax) для каждого сегмента.
7. Итоговый жанр определяется как мода — наиболее часто встречающийся жанр среди всех сегментов.

4 Сборка приложения

Приложение для классификации музыкальных жанров было реализовано в виде Telegram-бота. Такой подход позволил обеспечить удобный и быстрый доступ к функциональности без необходимости развёртывания полноценного веб-интерфейса или мобильного клиента. Пользователь может взаимодействовать с системой напрямую через Telegram, отправляя аудиофайлы и получая жанровую классификацию в ответ.

Для реализации бота использовалась библиотека `pyTelegramBotAPI` (обёртка над Telegram Bot API). Бот принимает аудиофайлы различных популярных форматов, проверяет их размер и длительность, а затем передаёт на вход обученной модели для классификации. После предсказания жанра результат отправляется обратно пользователю в виде текстового сообщения.

Сборка и запуск приложения включает в себя следующие этапы:

1. Подготовка обученной модели нейронной сети в формате HDF5 (`.h5`);
2. Разработка вспомогательных функций для обработки аудио (конвертация, извлечение признаков, предсказание жанра);
3. Интеграция модели в Telegram-бота с учётом ограничений Telegram API (например, ограничение на размер файла);
4. Реализация логики обработки различных типов сообщений, в том числе валидации и фильтрации неподдерживаемого контента;
5. Развёртывание бота на хостинге или локальной машине с запуском через бесконечный цикл опроса.

Такой формат позволил упростить пользовательский путь: пользователю не требуется установка дополнительного ПО — достаточно отправить трек боту в Telegram. Решение оказалось удобным, мобильным и легко масштабируемым для дальнейшего расширения функциональности.

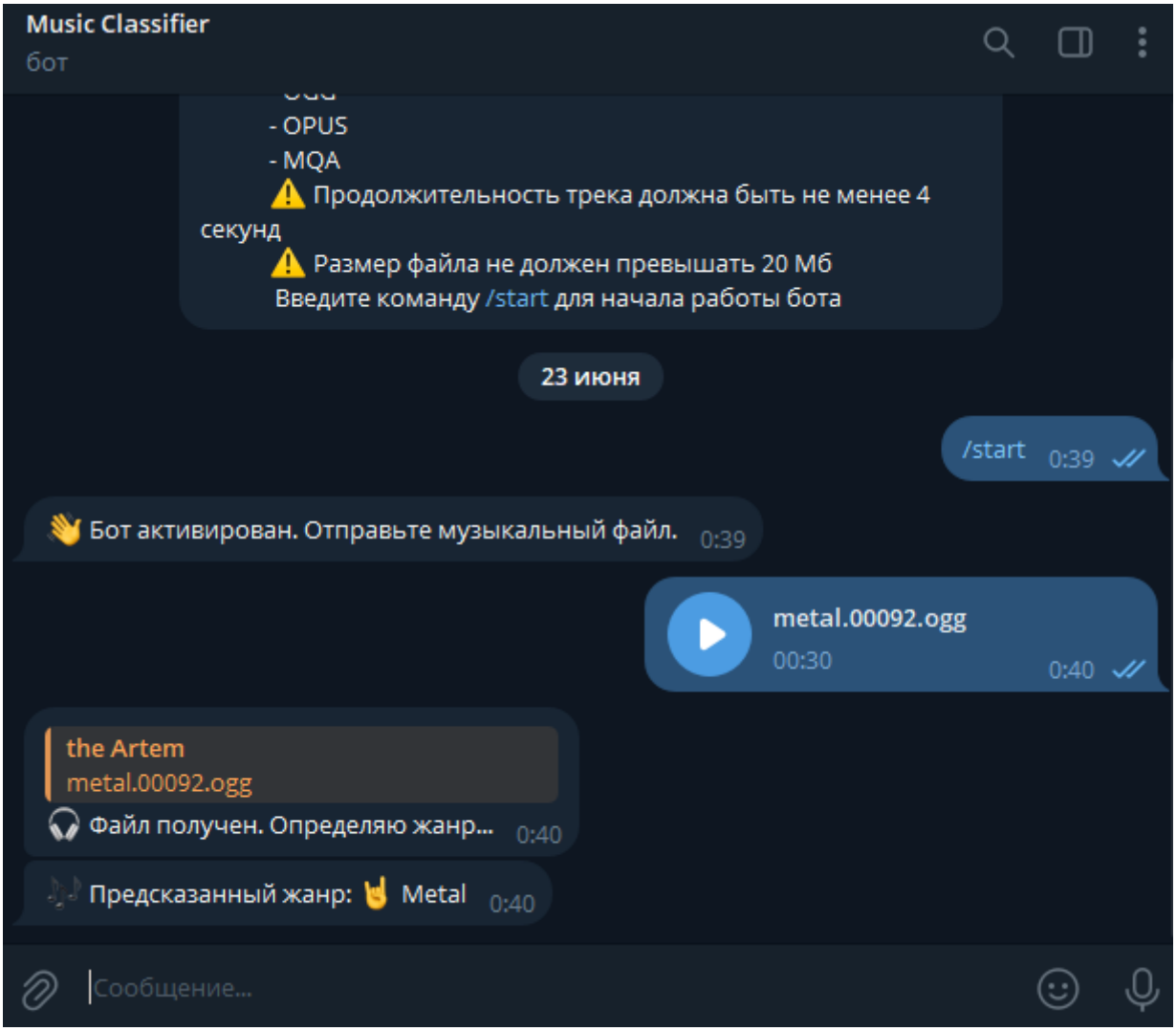

Рис. 2: Интерфейс работы Telegram-бота

5 Исходный код приложения

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import librosa
import os
from tqdm import tqdm
from keras.models import Sequential, load_model
from keras.layers import Dense, Flatten, Dropout, Conv2D, MaxPooling2D,
    BatchNormalization
from sklearn.model_selection import train_test_split

genres = {
    0: 'blues', 1: 'classical', 2: 'country', 3: 'disco', 4: 'hiphop',
    5: 'jazz', 6: 'metal', 7: 'pop', 8: 'reggae', 9: 'rock'
}

def extract_features(audio_path, n_mfcc=13, sr=22050, n_fft=2048,
                     hop_length=512, duration=30, num_segments=10):
    return np.array(segments_mfcc) if segments_mfcc else None

X, y = [], []
BASE_PATH = "gtzan_data/Data/genres_original"
for label, genre in genres.items():
    for i in tqdm(range(100), desc=f"Processing {genre}"):
        if features is not None:
            X.extend(features)
            y.extend([label] * len(features))

X = np.array(X)[..., np.newaxis]
y = np.array(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
    random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size
    =0.2, random_state=42)

def build_model(input_shape):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D((3, 3), strides=(2, 2), padding='same'),
        BatchNormalization(),
        Conv2D(32, (3, 3), activation='relu'),
        MaxPooling2D((3, 3), strides=(2, 2), padding='same'),
        BatchNormalization(),
        Conv2D(32, (2, 2), activation='relu'),
        MaxPooling2D((2, 2), strides=(2, 2), padding='same'),
        BatchNormalization(),
        Flatten(),
        Dense(64, activation='relu'),
        Dropout(0.1),
        Dense(10, activation='softmax')
    ])
    return model

model = build_model(input_shape=(X.shape[1], X.shape[2], 1))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=[
    'accuracy'])
history = model.fit(X_train, y_train, validation_data=(X_val, y_val), batch_size
    =32, epochs=100)
```

```
def plot_history(history):
    plt.show()

plot_history(history)

test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print(f'\nTest accuracy: {test_acc:.4f}')
model.save("genre_classification_cnn.h5")
```

Листинг 1: Исходный код обучения модели CNN

```
import os
import math
import librosa
import numpy as np
import tempfile
import subprocess
import imageio_ffmpeg
from scipy.stats import mode

genres = {
    0: 'Blues', 1: 'Classical', 2: 'Country', 3: 'Disco',
    4: 'Hip-Hop', 5: 'Jazz', 6: 'Metal', 7: 'Pop',
    8: 'Reggae', 9: 'Rock'
}

def convert_to_wav_ffmpeg(input_path):
    ffmpeg_path = imageio_ffmpeg.get_ffmpeg_exe()
    temp_wav = tempfile.NamedTemporaryFile(suffix=".wav", delete=False)
    temp_wav.close()
    try:
        subprocess.run([
            ffmpeg_path, "-y", "-i", input_path,
            "-ar", "44100", "-ac", "1", temp_wav.name
        ], check=True, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)
        return temp_wav.name
    except subprocess.CalledProcessError:
        os.unlink(temp_wav.name)
        raise RuntimeError(f"FFmpeg failed to convert file: {input_path}")

def extract_features(audio_path, n_mfcc=13, sr_target=22050, n_fft=2048,
                     hop_length=512, duration=30, num_segments=10):
    temp_file = None
    try:
        try:
            signal, sr = librosa.load(audio_path, sr=sr_target)
        except Exception:
            temp_file = convert_to_wav_ffmpeg(audio_path)
            signal, sr = librosa.load(temp_file, sr=sr_target)

        samples_per_track = duration * sr_target
        samples_per_segment = int(samples_per_track / num_segments)
        expected_frames = math.ceil(samples_per_segment / hop_length)

        if len(signal) < samples_per_segment:
            return None

        segments_mfcc = []
        for s in range(num_segments):
            start = samples_per_segment * s
```

```
            finish = start + samples_per_segment
            if len(signal[start:finish]) < samples_per_segment:
                continue
            mfcc = librosa.feature.mfcc(
                y=signal[start:finish], sr=sr, n_fft=n_fft,
                n_mfcc=n_mfcc, hop_length=hop_length
            ).T
            if len(mfcc) == expected_frames:
                segments_mfcc.append(mfcc)
        return np.array(segments_mfcc) if segments_mfcc else None
    finally:
        if temp_file is not None and os.path.exists(temp_file):
            os.remove(temp_file)

def predict_genre(model, audio_path):
    features = extract_features(audio_path)
    if features is None or features.shape[0] == 0:
        return "Audio is too short or corrupted."
    features = features[..., np.newaxis]
    predictions = model.predict(features)
    predicted_indices = np.argmax(predictions, axis=1)
    final_prediction = mode(predicted_indices, keepdims=True).mode[0]
    return f"Predicted genre: {genres[final_prediction]}"

def get_audio_duration(audio_path):
    temp_file = None
    try:
        try:
            y, sr = librosa.load(audio_path, sr=None)
        except Exception:
            temp_file = convert_to_wav_ffmpeg(audio_path)
            y, sr = librosa.load(temp_file, sr=None)
        return librosa.get_duration(y=y, sr=sr)
    finally:
        if temp_file is not None and os.path.exists(temp_file):
            os.remove(temp_file)
```

Листинг 2: Модуль предсказания жанра на основе аудиофайла

```
import telebot
import os
import tempfile
import threading
from predictor import predict_genre, get_audio_duration
from keras.models import load_model

telebot.apihelper.READ_TIMEOUT = 60
telebot.apihelper.CONNECT_TIMEOUT = 10

MAX_FILE_SIZE_BYTES = 20 * 1024 * 1024
bot = telebot.TeleBot('YOUR_BOT_TOKEN')

SUPPORTED_AUDIO_MIME_TYPES = {
    'audio/mpeg', 'audio/wav', 'audio/x-wav', 'audio/vnd.wave', 'audio/x-aiff',
    'audio/flac', 'audio/mp4', 'audio/aac', 'audio/ogg', 'audio/x-ms-wma',
    'audio/midi', 'audio/x-midi', 'audio/amr', 'audio/ac3', 'audio/x-dsf',
    'audio/opus', 'audio/x-opus', 'application/x-mqa',
}

MIN_DURATION_SECONDS = 4
model = load_model("genre_classification_cnn.h5")
```

```
user_states = set()

def process_file(message, file_info):
    try:
        if file_info.file_size > MAX_FILE_SIZE_BYTES:
            bot.send_message(message.chat.id, "File too large (> 20MB). Please
   send a smaller one.")
            return

        file = bot.get_file(file_info.file_id)
        downloaded = bot.download_file(file.file_path)

        ext = os.path.splitext(file.file_path)[1] or ".mp3"
        with tempfile.NamedTemporaryFile(delete=False, suffix=ext) as tmp:
            tmp.write(downloaded)
            tmp_path = tmp.name

        duration = get_audio_duration(tmp_path)
        if duration < MIN_DURATION_SECONDS:
            bot.send_message(message.chat.id, f"Track too short ({duration:.2f}
   sec). Minimum is 4 seconds.")
            return

        result = predict_genre(model, tmp_path)
        bot.send_message(message.chat.id, result)

    except Exception as e:
        bot.send_message(message.chat.id, f"Error while processing: {str(e)}")

    finally:
        if 'tmp_path' in locals() and os.path.exists(tmp_path):
            os.remove(tmp_path)

@bot.message_handler(commands=['start'])
def start_handler(message):
    user_states.add(message.chat.id)
    bot.send_message(message.chat.id, "Bot activated. Please send an audio file."
   )

@bot.message_handler(content_types=['audio'])
def handle_audio(message):
    if message.chat.id not in user_states:
        bot.reply_to(message, "Please start with the /start command.")
        return
    bot.reply_to(message, "Audio received. Predicting genre...")
    threading.Thread(target=process_file, args=(message, message.audio)).start()

@bot.message_handler(content_types=['document'])
def handle_document(message):
    if message.chat.id not in user_states:
        bot.reply_to(message, "Please start with the /start command.")
        return
    mime_type = message.document.mime_type
    if mime_type in SUPPORTED_AUDIO_MIME_TYPES:
        bot.reply_to(message, "Audio received. Predicting genre...")
        threading.Thread(target=process_file, args=(message, message.document)).
   start()
    else:
        bot.reply_to(message, f"File type `{mime_type}` is not supported.")
```

```
@bot.message_handler(func=lambda m: True, content_types=[
    'text', 'photo', 'video', 'voice', 'sticker', 'location', 'contact',
    'animation', 'video_note', 'poll', 'dice', 'venue'
])
def block_everything_else(message):
    if message.chat.id not in user_states:
        bot.reply_to(message, "Please start with the /start command.")
    else:
        bot.reply_to(message, "Only audio files are supported.")

bot.polling()
```

Листинг 3: Telegram-бот для классификации музыкальных жанров