

Классификация звуков с помощью библиотек TensorFlow и Librosa в Python

Эрдман Александр Алексеевич

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье рассмотрен процесс создания программы на языке программирования Python. Данная программа является алгоритмом машинного обучения и использует ключевые библиотеки TensorFlow и Librosa. Алгоритм обрабатывает звуковой файл, производит сравнение и определение звука на основе тестовых данных.

Ключевые слова: Python, TensorFlow, машинное обучение, Librosa

Classification of sounds using TensorFlow and Librosa libraries in Python

Erdman Alexander Alekseevich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article describes the process of creating a program in the Python programming language. This program is a machine learning algorithm and uses the TensorFlow and Librosa libraries. The algorithm processes the audio file, compares and determines the sound based on the test data.

Keywords: python, TensorFlow, machine learning, Librosa

1 Введение

1.1 Актуальность

В настоящее время машинное обучение и обработка данных являются существенным инструментом во многих сферах деятельности людей, начиная от научных исследований и заканчивая анализом статистики бизнес-проектов. Практически все сферы, где задействованы данные, используются машинное обучение и анализ на основе искусственного интеллекта. Часто ярким результатом активного использования машинного обучения можно считать предсказание цен на биржах или изменения спроса потребителей на основе конкретных признаков в бизнесе. В научных же целях машинное обучение используется для предсказания тех или иных погодных явлений, результатов экспериментов или предсказания срока эксплуатации того или иного материала. Помимо масштабных проектов в науке и бизнесе, машинное обучение активно используют энтузиасты в создании, например, изображений, что на сегодняшний день считается крайне популярным

занятием. Другим примером можно считать анализ и обработку звуковых файлов, выявление их явных признаков, на основе которых можно отнести звуковой файл к конкретному классу звука. Определение типа звука будет рассмотрено в данной статье, в качестве возможности машинного обучения обрабатывать и классифицировать звуковые данные.

1.2 Обзор исследований

Р.Л. Астапов и Р.М. Мухамадеева рассмотрели вопросы машинного обучения, а именно задача автоматизации подбора параметров машинного обучения и обучение модели с помощью кросс-валидации [1]. П.М. Горбунов, Ю.А. Мацкевич и А.В. Чубарь привели объяснение различных видов функционирования машинного обучения [2]. С.Ж. Дуалет реализовал разделения информации на части в социальной сети Twitter в среде Python [3]. В.А. Мельникова и Д.А. Медведев провели сравнительный анализ библиотек языка программирования Python для решения задач анализа данных [4]. Е.А. Григорьев и Н.С. Климонов рассмотрели и описали разведочный анализ данных с помощью Python [5]. О.А. Ермаков и Н.П. Брозгунова рассмотрели язык программирования возможности языка Python для анализа данных [6].

1.3 Цель исследования

Целью исследования является создание программы для определения звуков и его классификацию на основе dataset.

2 Материалы и методы

Для создания программы используется IDE Google Colab. Подключённая библиотека TensorFlow и Librosa. DataSet с классификацией звуков.

3 Результаты и обсуждения

Перед тем, как приступить к реализации классификации и определения звука, нужно звук сначала преобразовать. Программа не может напрямую обрабатывать звук и тем более его определять. Звук нужно представить в другом виде. Звук – это колебания, которые называются акустической волной. Волну можно измерить. Для этого используется частота, скорость, амплитуда и направление волны. Конкретно в задаче машинного обучения, существенную роль играет только частота и амплитуда. В создаваемой программе будут использована библиотека Librosa для анализа музыки. Благодаря данной библиотеки можно получить данные музыкального файла и его частоту дискретизации, на основе которой и будет происходить машинное обучение.

Запускаем web-сервис Google Colab, в данном сервисе будет происходить разработка программы. Далее необходимо импортировать библиотеку Librosa, а также дополнительные необходимые модули для работы:

```
import pandas as pd
import os
import librosa
import librosa.display
import numpy as np
import IPython.display as ipd
import matplotlib.pyplot as plt
%matplotlib inline
```

Данные берутся с открытых источников интернета. Далее файл загружается в папку проекта Google Colab. В конечном итоге получены два файла: первый содержит только аудиоданные, которые представлены звуками города, а второй представлен .csv и содержит необходимые данные о каждом звуке. После загрузки файлов и их подключения, можно взять произвольный звуковой файл. Выберем аудиофайл с играющими детьми. С помощью Librosa извлечём данные `audio_data` и `sampling_rate` (рис. 1).

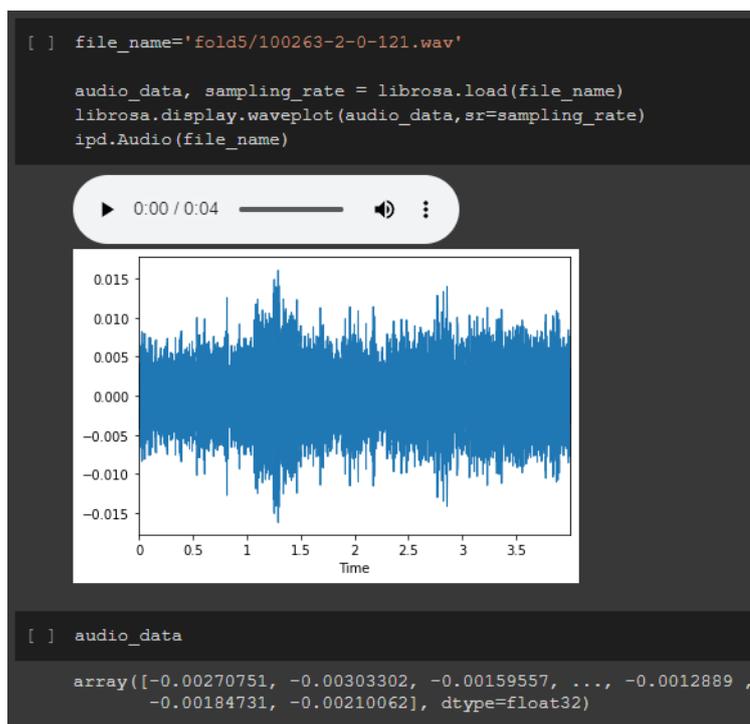
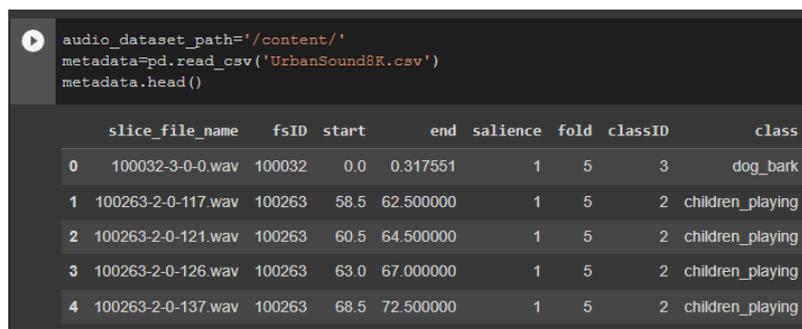


Рисунок 1. Вывод данных о звуке с помощью библиотеки Librosa

Далее выведем данные о аудиофайле. Можно видеть, что Librosa преобразовала файл в моносигнал. Всё верно, так как звуки из dataset являются монозвуками. Моно звук используется Librosa для упрощения дальнейшей обработки. С помощью команды `sampling_rate` можно узнать частоту дискретизации. По умолчанию библиотека использует дискретизацию 22050. С помощью библиотеки pandas открываем для чтения .csv файл:

```
audio_dataset_path='/content/'  
metadata=pd.read_csv('UrbanSound8K.csv')  
metadata.head()
```

После открытия dataset загружаем UrbanSound8K.csv файл, доступный в нашей папке загруженного набора данных. Затем идёт сохранение его в переменной, известной как metadata. Далее можно использовать метод head() для просмотра таблицы данных (рис. 2).



```
audio_dataset_path='/content/'  
metadata=pd.read_csv('UrbanSound8K.csv')  
metadata.head()
```

	slice_file_name	fsID	start	end	salience	fold	classID	class
0	100032-3-0-0.wav	100032	0.0	0.317551	1	5	3	dog_bark
1	100263-2-0-117.wav	100263	58.5	62.500000	1	5	2	children_playing
2	100263-2-0-121.wav	100263	60.5	64.500000	1	5	2	children_playing
3	100263-2-0-126.wav	100263	63.0	67.000000	1	5	2	children_playing
4	100263-2-0-137.wav	100263	68.5	72.500000	1	5	2	children_playing

Рисунок 2. Открытие и считывание данных с dataset

Данные также организованы в соответствующие классы файлов. Набор данных не должен быть несбалансированным. Для просмотра сбалансированности данных используем следующую команду:

```
metadata['class'].value_counts()
```

после выполнения программы можно увидеть результат (рис. 3).



```
[ ] metadata['class'].value_counts()  
  
street_music      1000  
jackhammer        1000  
air_conditioner   1000  
engine_idling     1000  
children_playing  1000  
dog_bark          1000  
drilling          1000  
siren             929  
car_horn          429  
gun_shot         374  
Name: class, dtype: int64
```

Рисунок 3. Проверка сбалансированности данных

Результаты показывают, что большинство классов в наборе данных сбалансированы. Таким образом, это будет хороший набор данных для анализа. Данные предварительно обрабатываются, чтобы извлечь значимые функции. Затем извлеченные функции используются для обучения вместо использования данных в необработанном виде.

Чтобы извлечь функции, будет использоваться алгоритм кепстральных коэффициентов Mel-Frequency (MFCC). Алгоритм MFCC суммирует частотное распределение по размеру окна. Это позволяет анализировать как частотные, так и временные характеристики предоставленного звука. Это позволит определить признаки для классификации:

```
mfccs = librosa.feature.mfcc(y=audio_data, sr=sampling_rate, n_mfcc=40)
```

После выполнения команды `mfccs` будут получены шаблоны из одного аудиофайла на основе частотных и временных характеристик (рис. 4).

```
[ ] mfccs
array([[ -4.6613168e+02, -4.6417816e+02, -4.7455182e+02, ...,
        -4.4540848e+02, -4.5221939e+02, -4.5637799e+02],
       [ 1.0846554e+02,  1.1128984e+02,  1.0955853e+02, ...,
        1.1160173e+02,  1.1063791e+02,  1.1319142e+02],
       [-2.5252140e+01, -2.7399439e+01, -3.2546665e+01, ...,
        -3.8440331e+01, -3.4312595e+01, -3.5521683e+01],
       ...,
       [ 2.3573508e+00,  1.6371250e+00,  3.2692363e+00, ...,
        7.8856702e+00,  1.0755114e+01,  1.1197763e+01],
       [-3.2311397e+00, -2.6380532e+00,  4.6177328e-01, ...,
        1.0223865e+01,  1.1984882e+01,  1.3385002e+01],
       [-1.3852274e+01, -1.0576165e+01, -2.1510942e+00, ...,
        2.9695926e+00,  2.1894133e+00,  6.6635776e-01]], dtype=float32)
```

Рисунок 5. Шаблоны, которые были извлечены из аудио на основе частотных и временных характеристик.

Чтобы извлечь функции из всех аудиофайлов в наборе данных, создаем список для хранения всех извлеченных функций (рис. 6).

```
[ ] def features_extractor(file):
    audio, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
    mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
    mfccs_scaled_features = np.mean(mfccs_features.T, axis=0)

    return mfccs_scaled_features
```

Рисунок 6. Создание списка для хранения функций

Затем перебирается каждый аудиофайл и извлекается функция, используя кепстральные коэффициенты Mel-Frequency (рис. 7).

```
[ ] extracted_features=[]
for index_num, row in tqdm(metadata.iterrows()):
    file_name = os.path.join(os.path.abspath(audio_dataset_path), 'fold'+str(row["fold"])+"/'+str(row["slice_file_name"]))
    final_class_labels=row["class"]
    data=features_extractor(file_name)
    extracted_features.append([data, final_class_labels])
```

Рисунок 7. Извлечение функций

Далее следует преобразовать весь список во фрейм данных с помощью библиотеки Pandas. Данный процесс представляет результаты в таблицы для более простого анализа (рис. 8).

```
[ ] extracted_features_df=pd.DataFrame(extracted_features,columns=['feature','class'])
extracted_features_df.head(10)
```

	feature	class
0	[-215.79301, 71.66612, -131.81377, -52.09133, ...]	dog_bark
1	[-424.68677, 110.56227, -54.148235, 62.01074, ...]	children_playing
2	[-459.56467, 122.800354, -47.92471, 53.265705, ...]	children_playing
3	[-414.55377, 102.896904, -36.66495, 54.18041, ...]	children_playing
4	[-447.397, 115.0954, -53.809113, 61.60859, 1.6...]	children_playing
5	[-447.70856, 118.409454, -35.24866, 56.73993, ...]	children_playing
6	[-477.1972, 120.63773, -29.692501, 57.051914, ...]	children_playing
7	[-464.84656, 117.71454, -30.163269, 50.72254, ...]	children_playing
8	[-472.1215, 126.76601, -38.36653, 58.748646, ...]	children_playing
9	[-196.18527, 114.94506, -14.661183, 1.2298629, ...]	car_horn

Рисунок 8. Извлечённые функции и их соответствующие классы

Следующая команда (функция `extract()` возвращает элементы массива) разбивает набор данных на независимые и зависимые наборы данных, X и y :

```
X=np.array(extracted_features_df['feature'].tolist())
y=np.array(extracted_features_df['class'].tolist())
```

X используется заглавная, так как принято обозначать двумерные массивы с заглавной буквы, а одномерные с маленьких, как пример y .

Затем идёт импорт оба метода из TensorFlow `to_categorical` и LabelEncoder Sklearn:

```
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
y=to_categorical(labelencoder.fit_transform(y))
```

LabelEncoder — служебный класс, помогающий нормализовать метки таким образом, чтобы они содержали только значения от 0 до $n_classes-1$. Используется для преобразования классов в числовые значения.

Этот шаг включает в себя использование метода `sklearn_train_test_split` для разделения набора данных на обучающие и тестовые наборы:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
```

После предварительных установок можно приступить к созданию модели (рис. 9). Слои модели будут последовательно сложены. Последний уровень будет иметь уровень активации `softmax`:

```
model=Sequential()
###первый слой
model.add(Dense(100,input_shape=(40,)))
```

```
model.add(Activation('relu'))
model.add(Dropout(0.5))
###второй слой
model.add(Dense(200))
model.add(Activation('relu'))
model.add(Dropout(0.5))
###третий слой
model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dropout(0.5))
###заключительный слой
model.add(Dense(num_labels))
model.add(Activation('softmax'))
```

Softmax функция активации используется последней для нормализации выходных данных. Использование других функций нормализации с Softmax выдаст выходные данные в вероятностной форме.

Функция активации ReLU используется для всех слоев, кроме выходного.

```
model.add(Dense(100, input_shape=(40,)))
```

теперь модель будет принимать в качестве входных массивов фигуры (*, 40)# и выходных массивов фигуры (*, 100)

Слой Dense - это слой, выходные нейроны которого связаны со всеми входными нейронами, в котором каждый нейрон предыдущего слоя связан с каждым нейроном данного. Параметры dense:

Первый параметр – выходной массив нейронов (100, чем больше нейронов, тем больше чувствительность сети к изменению), второй параметр (input_shape) – входной массив нейронов (40, так как у нас такое количество параметров, полученных с помощью librosa).

Параметр Activation отвечает за активацию функций нормализации.

Параметр Dropout (метод регуляризации переобучения сети) выключает нейроны с вероятностью p и оставляет их включенными с вероятностью q=1-p.

Теперь можно приступить к обучению модели. Чем больше число эпох, тем больше увеличивается точность. Рассмотрим модель с 200 эпохами (рис. 10).

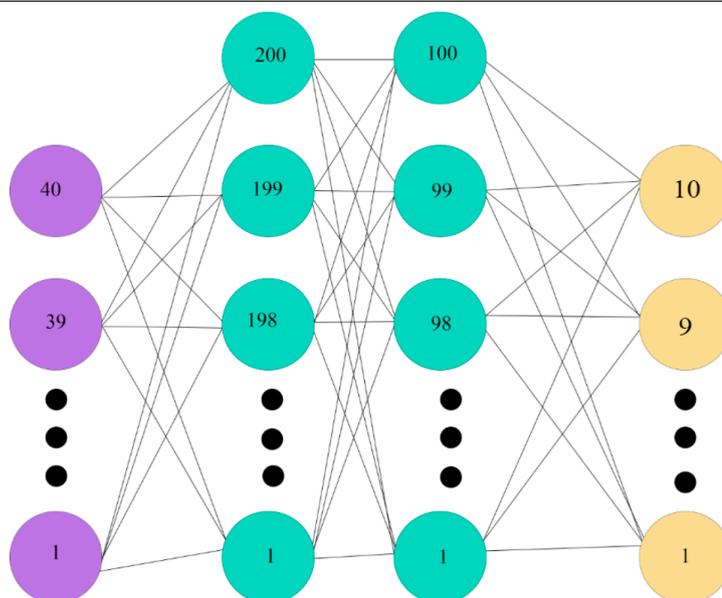


Рисунок 9. Модель нейросети

```

from tensorflow.keras.callbacks import ModelCheckpoint
from datetime import datetime

num_epochs = 200
num_batch_size = 32

checkpointer = ModelCheckpoint(filepath='saved_models/audio_classification.hdf5',
                               verbose=1, save_best_only=True)
start = datetime.now()

model.fit(X_train, y_train, batch_size=num_batch_size, epochs=num_epochs, validation_data=(X_test, y_test), callbacks=[checkpointer])

duration = datetime.now() - start
print("Training completed in time: ", duration)

```

Рисунок 10. Обучение модели

Можно посмотреть точность модели. Делается это с помощью следующей команды:

```

test_accuracy=model.evaluate(X_test,y_test,verbose=0)
print(test_accuracy[1])

```

Результат проверки точности имеет значение 0.7870635390281677. если переводить в проценты, то модель имеет точность в 78,71%. Данное значение удовлетворительно. Тестирование модели будет осуществляться с помощью аудиофайла с собачим лаем из набора данных. С помощью библиотеки Librosa извлекаем необходимые данные о файле. Данные нужны, для работы модели, так как на входе она принимает не аудиофайл, а данные о дискретизации звука. Использую `inverse_transform` метод из `scikitlearn`, для получения предсказания - к какому классу относится аудиофайл (рис. 10). В завершении работы модели, получится результат, что файл принадлежит к классу «dog_bark», что в свою очередь является верным.

```
[ ] filename="fold8/103076-3-0-0.wav"
audio, sample_rate = librosa.load(filename, res_type='kaiser_fast')
mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)

print(mfccs_scaled_features)
mfccs_scaled_features=mfccs_scaled_features.reshape(1,-1)
print(mfccs_scaled_features)
print(mfccs_scaled_features.shape)
predicted_label=model.predict(mfccs_scaled_features)
print(predicted_label)
classes_x=np.argmax(predicted_label,axis=1)
prediction_class = labelencoder.inverse_transform(classes_x)
prediction_class

[-4.3655270e+02  5.9230797e+01 -4.3656607e+00 -1.8696339e+01
-2.1808062e+01 -2.1441746e+01 -2.0656868e+01 -1.0533067e+01
-1.0726240e+01  1.1780758e+00  2.9764354e+00  1.8751724e+00
 1.1146365e+00  3.6338196e+00  5.0674267e+00  1.7557142e+00
-3.5938418e+00 -3.6886477e+00 -6.2886648e+00 -8.8013878e+00
-8.1570635e+00 -4.9429231e+00 -1.7873219e+00  2.0938160e+00
 2.1526849e+00 -4.7784606e-01 -2.7652733e+00 -3.8324444e+00
-3.7962842e+00 -4.3711638e+00 -2.7436407e+00  1.5701178e-01
-2.4533394e-01  7.0105374e-01  1.4864315e+00  1.7523667e+00
 1.7043139e+00  4.5822546e-01  5.2978408e-01 -1.0451510e+00]
[[-4.3655270e+02  5.9230797e+01 -4.3656607e+00 -1.8696339e+01
-2.1808062e+01 -2.1441746e+01 -2.0656868e+01 -1.0533067e+01
-1.0726240e+01  1.1780758e+00  2.9764354e+00  1.8751724e+00
 1.1146365e+00  3.6338196e+00  5.0674267e+00  1.7557142e+00
-3.5938418e+00 -3.6886477e+00 -6.2886648e+00 -8.8013878e+00
-8.1570635e+00 -4.9429231e+00 -1.7873219e+00  2.0938160e+00
 2.1526849e+00 -4.7784606e-01 -2.7652733e+00 -3.8324444e+00
-3.7962842e+00 -4.3711638e+00 -2.7436407e+00  1.5701178e-01
-2.4533394e-01  7.0105374e-01  1.4864315e+00  1.7523667e+00
 1.7043139e+00  4.5822546e-01  5.2978408e-01 -1.0451510e+00]]
(1, 40)
[[1.1630526e-21  5.3596851e-09  2.6831966e-09  9.9984801e-01  4.6294679e-09
 9.3139047e-12  1.5179862e-04  4.0151480e-34  1.1097348e-07  4.4551591e-08]]
array(['dog_bark'], dtype='<U16')
```

Рисунок 11. Результат работы модели.

Для оценки точности для текущих параметров сети, используются дополнительные два звуковых файла не из загруженного dataset, то есть взятые извне. Взятые звуки: звук музыки на улице, два аудиофайла лаю собак, два аудиофайла гудка автомобиля. В результате, нейронная сеть точно определила аудиофайлы с лаем собак и звуком музыки на улице. При этом один из аудиофайлов гудка автомобиля был отнесен к неверному классу – к лаю собак. Результаты (рис. 12,13 и 14).

<pre>filename="out_sound_dog.wav" audio, sample_rate = librosa.load(filename, res_type='kaiser_fast') mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40) mfccs_scaled_features = np.mean(mfccs_features.T,axis=0) print(mfccs_scaled_features) mfccs_scaled_features=mfccs_scaled_features.reshape(1,-1) print(mfccs_scaled_features) print(mfccs_scaled_features.shape) predicted_label=model.predict(mfccs_scaled_features) print(predicted_label) classes_x=np.argmax(predicted_label,axis=1) prediction_class = labelencoder.inverse_transform(classes_x) prediction_class [[-1.8236664e+02 1.3183081e+02 -9.8588486e+01 -2.1093405e+01 -4.6659416e+01 -9.7709694e+00 -5.9313278e+01 -3.0955156e+01 -7.5421147e+00 -2.8683283e+01 -7.2482958e+00 -1.0420638e+01 6.1837566e-01 -2.0641050e-01 -1.4241486e+01 4.0081148e+00 6.1063957e+00 7.5485930e+00 -9.6257770e-01 1.6003011e+00 -1.0698760e+01 -1.0022529e+00 3.1427519e+00 -1.2578575e+01 -8.8509876e-01 -3.4090400e+00 -4.0870514e+00 5.7263875e+00 -5.7918894e-01 -2.6811323e+00 -6.2039274e-01 4.0337639e+00 3.5109911e+00 1.1389457e+00 -2.3520644e+00 7.9514273e-02 3.5263414e+00 -4.6442709e+00 -4.2763463e-01 1.6682541e+00] (1, 40) [=====] - 0s 27ms/step [[0.01294169 0.01368043 0.272765 0.3132455 0.08068335 0.02814014 0.08031598 0.00111667 0.10605847 0.09105281]] array(['dog_bark'], dtype='<U16')</pre>	<pre>filename="out_sound_dog2.wav" audio, sample_rate = librosa.load(filename, res_type='kaiser_fast') mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40) mfccs_scaled_features = np.mean(mfccs_features.T,axis=0) print(mfccs_scaled_features) mfccs_scaled_features=mfccs_scaled_features.reshape(1,-1) print(mfccs_scaled_features) print(mfccs_scaled_features.shape) predicted_label=model.predict(mfccs_scaled_features) print(predicted_label) classes_x=np.argmax(predicted_label,axis=1) prediction_class = labelencoder.inverse_transform(classes_x) prediction_class [[-1.4078561e+02 1.8173198e+02 -7.0200615e+01 -1.0300575e+01 -4.9674370e+01 1.3458523e+01 -2.7396250e+01 -4.4098110e+00 -8.7458115e+00 -6.9304929e+00 -3.9779851e+00 -1.3994117e+00 8.4119010e-01 -6.9902415e+00 -4.6730294e+00 -5.2945619e+00 8.0926018e+00 1.0921448e+01 1.4664588e+01 7.5979381e+00 1.1070359e+01 5.4111099e+00 7.9251552e+00 3.7951047e+00 -8.2319403e-01 -3.4129705e+00 1.6432283e+00 6.7218691e-01 -5.7664981e+00 2.3629658e+00 2.5142777e+00 7.7215009e+00 1.5526319e+00 1.4523986e+01 1.3229584e+00 4.6799230e-04 -4.7848778e+00 -2.1940784e+00 2.6135480e+00 3.9355737e-01] [[-1.4078561e+02 1.8173198e+02 -7.0200615e+01 -1.0300575e+01 -4.9674370e+01 1.3458523e+01 -2.7396250e+01 -4.4098110e+00 -8.7458115e+00 -6.9304929e+00 -3.9779851e+00 -1.3994117e+00 8.4119010e-01 -6.9902415e+00 -4.6730294e+00 -5.2945619e+00 8.0926018e+00 1.0921448e+01 1.4664588e+01 7.5979381e+00 1.1070359e+01 5.4111099e+00 7.9251552e+00 3.7951047e+00 -8.2319403e-01 -3.4129705e+00 1.6432283e+00 6.7218691e-01 -5.7664981e+00 2.3629658e+00 2.5142777e+00 7.7215009e+00 1.5526319e+00 1.4523986e+00 2.6135480e+00 4.6799230e-04 -4.7848778e+00 -2.1940784e+00 2.1325480e+00 3.9355737e-01]] (1, 40) [=====] - 0s 30ms/step [[3.3640224e-09 1.2820633e-04 1.1594877e-04 9.6925688e-01 2.3084491e-02 7.9239425e-07 4.6159057e-03 8.4258867e-12 2.4083490e-03 3.8948414e-04]] array(['dog_bark'], dtype='<U16')</pre>
--	--

Рисунок 12. Тестирование нейронной сети на внешних аудиофайлах со лаем собак

```

filename="car_sound1.wav"
audio, sample_rate = librosa.load(filename, res_type='kaiser_fast')
mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)

print(mfccs_scaled_features)
mfccs_scaled_features=mfccs_scaled_features.reshape(1,-1)
print(mfccs_scaled_features)
print(mfccs_scaled_features.shape)
predicted_label=model.predict(mfccs_scaled_features)
print(predicted_label)
classes_x=np.argmax(predicted_label,axis=1)
prediction_class = labelencoder.inverse_transform(classes_x)
prediction_class

[-5.7462189e+02  3.3808994e+01 -5.6201100e+00  3.9300971e+00
 -2.2574799e+00 -7.7198071e+00  1.5800316e+00  3.9706926e+00
 -2.0904150e+00 -6.2591368e-01 -3.2974979e+00  1.8491436e+00
 -3.2435424e+00  4.2411475e+00  4.0831432e+00  6.5412908e+00
 3.2112709e+00  4.2118416e+00  1.4302151e+00  5.6432569e+00
 4.6867991e+00  4.0760684e+00 -7.8926790e-01 -2.2131443e+00
 -1.4251387e+00  3.4480147e+00  3.2554731e+00  2.7947581e-01
 -8.5148937e-01  6.2161422e+00  7.2225089e+00  3.4155545e-01
 -1.9953592e+00 -7.6961380e-01 -3.3882323e-01 -6.9021964e-01
 1.8432094e+00  2.2144227e+00  1.8588995e+00  1.3880973e+00]
[[[-5.7462189e+02  3.3808994e+01 -5.6201100e+00  3.9300971e+00
 -2.2574799e+00 -7.7198071e+00  1.5800316e+00  3.9706926e+00
 -2.0904150e+00 -6.2591368e-01 -3.2974979e+00  1.8491436e+00
 -3.2435424e+00  4.2411475e+00  4.0831432e+00  6.5412908e+00
 3.2112709e+00  4.2118416e+00  1.4302151e+00  5.6432569e+00
 4.6867991e+00  4.0760684e+00 -7.8926790e-01 -2.2131443e+00
 -1.4251387e+00  3.4480147e+00  3.2554731e+00  2.7947581e-01
 -8.5148937e-01  6.2161422e+00  7.2225089e+00  3.4155545e-01
 -1.9953592e+00 -7.6961380e-01 -3.3882323e-01 -6.9021964e-01
 1.8432094e+00  2.2144227e+00  1.8588995e+00  1.3880973e+00]]]
(1, 40)
1/1 [=====] - 0s 30ms/step
[[6.0326151e-08 4.0734917e-04 7.9587934e-04 9.3632458e-01 2.4268923e-02
 6.960806e-06 3.6100805e-02 1.3265615e-10 8.4894238e-04 4.4655620e-04]]
array(['dog_bark'], dtype='<U16')

filename="car_sound2.wav"
audio, sample_rate = librosa.load(filename, res_type='kaiser_fast')
mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)

print(mfccs_scaled_features)
mfccs_scaled_features=mfccs_scaled_features.reshape(1,-1)
print(mfccs_scaled_features)
print(mfccs_scaled_features.shape)
predicted_label=model.predict(mfccs_scaled_features)
print(predicted_label)
classes_x=np.argmax(predicted_label,axis=1)
prediction_class = labelencoder.inverse_transform(classes_x)
prediction_class

[-253.5821  109.29108 -44.41666  8.280988 -25.104841
  6.571808  -6.0477734  4.425404  -9.715536  -3.6272295
 -2.3679652  -3.981425  -3.4095535  1.0707242  11.0597105
 12.013969  14.005458  7.4469337  12.171533  7.489733
 0.9137353  -13.967778  -10.721594  -2.8176706  5.5276604
 3.997398  -2.4007936  6.6634207  16.81286  10.892124
 -12.504068  -15.02983  -1.7986317  14.982389  14.863367
 9.794505  3.0410419  -8.367804  -12.226035  -2.417758 ]
[[-253.5821  109.29108 -44.41666  8.280988 -25.104841
  6.571808  -6.0477734  4.425404  -9.715536  -3.6272295
 -2.3679652  -3.981425  -3.4095535  1.0707242  11.0597105
 12.013969  14.005458  7.4469337  12.171533  7.489733
 0.9137353  -13.967778  -10.721594  -2.8176706  5.5276604
 3.997398  -2.4007936  6.6634207  16.81286  10.892124
 -12.504068  -15.02983  -1.7986317  14.982389  14.863367
 9.794505  3.0410419  -8.367804  -12.226035  -2.417758 ]]
(1, 40)
1/1 [=====] - 0s 34ms/step
[[1.2664827e-04 4.7270608e-01 4.0016552e-03 5.2544188e-02 2.2002526e-03
 8.2689198e-03 2.1083422e-04 3.4798672e-06 4.4651376e-04 4.5949137e-01]]
array(['car_horn'], dtype='<U16')
    
```

Рисунок 13. Тестирование нейронной сети на внешних аудиофайлах со гудком автомобиля

```

filename="sound_music_street1.wav"
audio, sample_rate = librosa.load(filename, res_type='kaiser_fast')
mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)

print(mfccs_scaled_features)
mfccs_scaled_features=mfccs_scaled_features.reshape(1,-1)
print(mfccs_scaled_features)
print(mfccs_scaled_features.shape)
predicted_label=model.predict(mfccs_scaled_features)
print(predicted_label)
classes_x=np.argmax(predicted_label,axis=1)
prediction_class = labelencoder.inverse_transform(classes_x)
prediction_class

[-2.4404710e+02  1.1938208e+02 -1.1296660e+01  3.5044010e+01
 1.0013022e+01  1.1802057e+01 -8.5653362e+00  2.0679411e+01
 6.3036098e+00  3.8377669e+00 -7.7066207e+00  2.2258438e+01
 -1.1481325e+01 -2.6872604e+00 -3.3246212e+00  8.2322950e+00
 -7.5488505e+00 -1.3672558e+00  9.6457720e-01  9.3113441e+00
 -1.6073011e+01  7.0302777e+00  3.2503090e+00  3.7664301e+00
 -5.0800734e+00  7.0393124e+00 -1.2052009e-01 -2.9346967e+00
 -5.0865850e+00  6.2249932e+00  2.2597167e+00 -5.2914991e+00
 1.3862771e+00  1.5837450e+00 -1.3335245e+00 -1.8011390e+00
 -3.9155236e-01  1.6463252e+00 -4.8073783e+00 -1.8071476e+00]]
[[-2.4404710e+02  1.1938208e+02 -1.1296660e+01  3.5044010e+01
 1.0013022e+01  1.1802057e+01 -8.5653362e+00  2.0679411e+01
 6.3036098e+00  3.8377669e+00 -7.7066207e+00  2.2258438e+01
 -1.1481325e+01 -2.6872604e+00 -3.3246212e+00  8.2322950e+00
 -7.5488505e+00 -1.3672558e+00  9.6457720e-01  9.3113441e+00
 -1.6073011e+01  7.0302777e+00  3.2503090e+00  3.7664301e+00
 -5.0800734e+00  7.0393124e+00 -1.2052009e-01 -2.9346967e+00
 -5.0865850e+00  6.2249932e+00  2.2597167e+00 -5.2914991e+00
 1.3862771e+00  1.5837450e+00 -1.3335245e+00 -1.8011390e+00
 -3.9155236e-01  1.6463252e+00 -4.8073783e+00 -1.8071476e+00]]]
(1, 40)
1/1 [=====] - 0s 41ms/step
[[0.03875499 0.01044103 0.08913295 0.04834225 0.03842955 0.02879944
 0.00530116 0.02326409 0.00801405 0.7095506 ]]
array(['street_music'], dtype='<U16')

filename="sound_music_street2.wav"
audio, sample_rate = librosa.load(filename, res_type='kaiser_fast')
mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)

print(mfccs_scaled_features)
mfccs_scaled_features=mfccs_scaled_features.reshape(1,-1)
print(mfccs_scaled_features)
print(mfccs_scaled_features.shape)
predicted_label=model.predict(mfccs_scaled_features)
print(predicted_label)
classes_x=np.argmax(predicted_label,axis=1)
prediction_class = labelencoder.inverse_transform(classes_x)
prediction_class

[-2.1725014e+02  1.4075960e+02 -3.8480167e+01  2.3919878e+01
 -1.0248403e+01 -8.1848135e+00 -1.6991596e+01  1.1453503e+01
 -2.1640848e+01  4.7526040e+00 -1.1588472e+01  1.2221844e+01
 -1.5336685e+01  7.6295462e+00 -6.5573945e+00  5.5851278e+00
 -9.4296589e+00  6.5117507e+00 -5.6672606e+00  3.0875707e+00
 -4.2792702e+00  3.4702168e+00 -3.9924436e+00  1.8086906e+00
 -2.1477988e+00  1.2286112e+00 -1.8102539e+00  1.8557609e+00
 -2.3083696e+00  1.9204533e-01 -1.0716035e-01  8.4439123e-01
 -6.4679140e-01 -4.5215431e-01 -8.3006692e-01 -8.4185261e-01
 -8.5995960e-01 -6.1489034e-01 -3.1687611e-01 -1.3732024e-01]]
[[-2.1725014e+02  1.4075960e+02 -3.8480167e+01  2.3919878e+01
 -1.0248403e+01 -8.1848135e+00 -1.6991596e+01  1.1453503e+01
 -2.1640848e+01  4.7526040e+00 -1.1588472e+01  1.2221844e+01
 -1.5336685e+01  7.6295462e+00 -6.5573945e+00  5.5851278e+00
 -9.4296589e+00  6.5117507e+00 -5.6672606e+00  3.0875707e+00
 -4.2792702e+00  3.4702168e+00 -3.9924436e+00  1.8086906e+00
 -2.1477988e+00  1.2286112e+00 -1.8102539e+00  1.8557609e+00
 -2.3083696e+00  1.9204533e-01 -1.0716035e-01  8.4439123e-01
 -6.4679140e-01 -4.5215431e-01 -8.3006692e-01 -8.4185261e-01
 -8.5995960e-01 -6.1489034e-01 -3.1687611e-01 -1.3732024e-01]]]
(1, 40)
1/1 [=====] - 0s 33ms/step
[[0.0141067 0.04420345 0.08188788 0.09214321 0.25824043 0.02476823
 0.02382949 0.05937076 0.0090454 0.3924044 ]]
array(['street_music'], dtype='<U16')
    
```

Рисунок 14. Тестирование нейронной сети на внешних аудиофайлах со звуком музыки на улице

Изменим параметры модели. Поменяем выходные нейроны у параметра Dense со 100 до 50. Количество эпох (сколько раз алгоритм обучения будет работать со всем набором обучающих данных) останется неизменным - 200. Тест нейронной сети пройдет с теми же данными. В результате подстановки точность модели упала с 78,71% до 67,88%. Можно предположить, что нейронная сеть допустит больше ошибок в этот раз. Проверка работы. Лай собак распознал верно в обоих случаях, в то время как один из файлов с гудком автомобиля был определен верно, а другой не

верно, а оставшиеся аудиофайлы со звуком музыки на улице были определены полностью неправильно (рис. 15,16).

```

filename="sound_music_street1.wav"
audio, sample_rate = librosa.load(filename, res_type='kaiser_fast')
mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)

print(mfccs_scaled_features)
mfccs_scaled_features=mfccs_scaled_features.reshape(1,-1)
print(mfccs_scaled_features)
print(mfccs_scaled_features.shape)
predicted_label=model.predict(mfccs_scaled_features)
print(predicted_label)
classes_x=np.argmax(predicted_label,axis=1)
prediction_class = labelencoder.inverse_transform(classes_x)
prediction_class

[-2.4404710e+02  1.1938208e+02 -1.1296660e+01  3.5044010e+01
 1.0013022e+01  1.1802057e+01 -8.5653362e+00  2.0679411e+01
 6.3036098e+00  3.8377669e+00 -7.7066207e+00  2.2258438e+01
 -1.1481325e+01 -2.6872604e+00 -3.3246212e+00  8.2322950e+00
 -7.5488505e+00 -1.3672588e+00  9.6457720e-01  9.3113441e+00
 -1.6073011e+01  7.0302777e+00  3.2503090e+00  3.7664301e+00
 -5.0800734e+00  7.0393124e+00 -1.2052009e-01 -2.9346967e+00
 -5.0865850e+00  6.2249932e+00  2.2597167e+00 -5.2914991e+00
 1.3862771e+00  1.5837450e+00 -1.3335245e+00 -1.8011390e+00
 -3.9155236e-01  1.6463252e+00 -4.8073783e+00 -1.8071476e+00]
[[[-2.4404710e+02  1.1938208e+02 -1.1296660e+01  3.5044010e+01
 1.0013022e+01  1.1802057e+01 -8.5653362e+00  2.0679411e+01
 6.3036098e+00  3.8377669e+00 -7.7066207e+00  2.2258438e+01
 -1.1481325e+01 -2.6872604e+00 -3.3246212e+00  8.2322950e+00
 -7.5488505e+00 -1.3672588e+00  9.6457720e-01  9.3113441e+00
 -1.6073011e+01  7.0302777e+00  3.2503090e+00  3.7664301e+00
 -5.0800734e+00  7.0393124e+00 -1.2052009e-01 -2.9346967e+00
 -5.0865850e+00  6.2249932e+00  2.2597167e+00 -5.2914991e+00
 1.3862771e+00  1.5837450e+00 -1.3335245e+00 -1.8011390e+00
 -3.9155236e-01  1.6463252e+00 -4.8073783e+00 -1.8071476e+00]]]
(1, 40)
1/1 [=====] - 0s 26ms/step
[[0.23468757 0.0168385 0.11098956 0.03198819 0.04574782 0.12142473
 0.01577493 0.13592051 0.04882997 0.12235235]]
array(['air_conditioner'], dtype='<U16')

filename="sound_music_street2.wav"
audio, sample_rate = librosa.load(filename, res_type='kaiser_fast')
mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)

print(mfccs_scaled_features)
mfccs_scaled_features=mfccs_scaled_features.reshape(1,-1)
print(mfccs_scaled_features)
print(mfccs_scaled_features.shape)
predicted_label=model.predict(mfccs_scaled_features)
print(predicted_label)
classes_x=np.argmax(predicted_label,axis=1)
prediction_class = labelencoder.inverse_transform(classes_x)
prediction_class

[-2.1725014e+02  1.4075960e+02 -3.8480167e+01  2.3919878e+01
 -1.0248403e+01 -8.1848135e+00 -1.6991596e+01  1.1453503e+01
 -2.1640848e+01  4.7526040e+00 -1.1588472e+01  1.2221844e+00
 -1.5336685e+01  7.6295462e+00 -6.5573945e+00  5.9851278e+00
 -9.4296589e+00  6.5117507e+00 -5.6672606e+00  3.0875070e+00
 -4.2792702e+00  3.4702168e+00 -3.9924365e+00  1.8086900e+00
 -2.1477988e+00  1.2286112e+00 -1.8102539e+00  1.8557609e+00
 -2.3083696e+00  1.9204533e-01 -1.0716035e-01  8.4493123e-01
 -6.4679140e-01 -4.5215431e-01 -8.3006692e-01 -1.4185261e-01
 -8.5995960e-01 -6.1489034e-01 -3.168711e-01 -1.3732024e-01]
[[[-2.1725014e+02  1.4075960e+02 -3.8480167e+01  2.3919878e+01
 -1.0248403e+01 -8.1848135e+00 -1.6991596e+01  1.1453503e+01
 -2.1640848e+01  4.7526040e+00 -1.1588472e+01  1.2221844e+01
 -1.5336685e+01  7.6295462e+00 -6.5573945e+00  5.9851278e+00
 -9.4296589e+00  6.5117507e+00 -5.6672606e+00  3.0875070e+00
 -4.2792702e+00  3.4702168e+00 -3.9924365e+00  1.8086900e+00
 -2.1477988e+00  1.2286112e+00 -1.8102539e+00  1.8557609e+00
 -2.3083696e+00  1.9204533e-01 -1.0716035e-01 -8.4493123e-01
 -6.4679140e-01 -4.5215431e-01 -8.3006692e-01 -1.4185261e-01
 -8.5995960e-01 -6.1489034e-01 -3.168711e-01 -1.3732024e-01]]]
(1, 40)
1/1 [=====] - 0s 33ms/step
[[0.11228879 0.01930218 0.30590713 0.09370744 0.06778705 0.05020875
 0.03898185 0.00667817 0.04051195 0.2646267 ]]
array(['children_playing'], dtype='<U16')
    
```

Рисунок 15. Тестирование нейронной сети с альтернативными параметрами на внешних аудиофайлах со звуком музыки на улице

```

filename="car_sound1.wav"
audio, sample_rate = librosa.load(filename, res_type='kaiser_fast')
mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)

print(mfccs_scaled_features)
mfccs_scaled_features=mfccs_scaled_features.reshape(1,-1)
print(mfccs_scaled_features)
print(mfccs_scaled_features.shape)
predicted_label=model.predict(mfccs_scaled_features)
print(predicted_label)
classes_x=np.argmax(predicted_label,axis=1)
prediction_class = labelencoder.inverse_transform(classes_x)
prediction_class

[-5.7462189e+02  3.3808994e+01 -5.6201100e+00  3.9300971e+00
 -2.2574799e+00 -7.7198071e+00  1.5800316e+00  3.9706926e+00
 -2.0904150e+00 -6.2591368e-01 -3.2574379e+00  1.8491436e+00
 -3.2435424e+00  4.2411475e+00  4.0831432e+00  6.5412908e+00
 3.2112708e+00  4.2118416e+00  1.4302151e+00  3.6432569e+00
 4.6867981e+00  4.0760684e+00 -7.8926790e-01 -2.2131443e+00
 -1.4251387e+00  3.4480147e+00  3.2554731e+00  2.7947581e-01
 -8.5148937e-01  6.2161422e+00  7.2225089e+00  3.4155545e-01
 -1.9953592e+00 -7.6961380e-01 -3.3882323e-01 -6.9021964e-01
 1.8432094e+00  2.2144227e+00  1.8588995e+00  1.3880973e+00]
[[[-5.7462189e+02  3.3808994e+01 -5.6201100e+00  3.9300971e+00
 -2.2574799e+00 -7.7198071e+00  1.5800316e+00  3.9706926e+00
 -2.0904150e+00 -6.2591368e-01 -3.2574379e+00  1.8491436e+00
 -3.2435424e+00  4.2411475e+00  4.0831432e+00  6.5412908e+00
 3.2112708e+00  4.2118416e+00  1.4302151e+00  3.6432569e+00
 4.6867981e+00  4.0760684e+00 -7.8926790e-01 -2.2131443e+00
 -1.4251387e+00  3.4480147e+00  3.2554731e+00  2.7947581e-01
 -8.5148937e-01  6.2161422e+00  7.2225089e+00  3.4155545e-01
 -1.9953592e+00 -7.6961380e-01 -3.3882323e-01 -6.9021964e-01
 1.8432094e+00  2.2144227e+00  1.8588995e+00  1.3880973e+00]]]
(1, 40)
1/1 [=====] - 0s 43ms/step
[[9.4688912e-06 2.3942158e-04 2.0393388e-01 4.9560394e-11 7.0414762e-03
 2.5119467e-04 2.7016991e-01 5.6397417e-11 2.0833366e-02 1.9178576e-03]]
array(['dog_bark'], dtype='<U16')

filename="car_sound2.wav"
audio, sample_rate = librosa.load(filename, res_type='kaiser_fast')
mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
mfccs_scaled_features = np.mean(mfccs_features.T,axis=0)

print(mfccs_scaled_features)
mfccs_scaled_features=mfccs_scaled_features.reshape(1,-1)
print(mfccs_scaled_features)
print(mfccs_scaled_features.shape)
predicted_label=model.predict(mfccs_scaled_features)
print(predicted_label)
classes_x=np.argmax(predicted_label,axis=1)
prediction_class = labelencoder.inverse_transform(classes_x)
prediction_class

[-253.5821  109.29108 -44.41666  8.280988 -25.104841
 6.571808 -6.0477734  4.425404 -9.715536 -3.6272295
 -2.3679652 -3.981425 -3.4095535  1.0707242  11.0597105
 12.013969  14.005458  7.4469387  12.171533  7.489793
 0.137353 -13.97779 -10.721594 -2.8176706  5.5276604
 3.997398 -2.4007936  6.6634207  16.81286  10.892124
 -12.504068 -15.02983 -1.7986317  14.982389  14.863367
 9.794505  3.0410419 -8.367804 -12.226035 -2.417758 ]
[[[-253.5821  109.29108 -44.41666  8.280988 -25.104841
 6.571808 -6.0477734  4.425404 -9.715536 -3.6272295
 -2.3679652 -3.981425 -3.4095535  1.0707242  11.0597105
 12.013969  14.005458  7.4469387  12.171533  7.489793
 0.137353 -13.97779 -10.721594 -2.8176706  5.5276604
 3.997398 -2.4007936  6.6634207  16.81286  10.892124
 -12.504068 -15.02983 -1.7986317  14.982389  14.863367
 9.794505  3.0410419 -8.367804 -12.226035 -2.417758 ]]]]
(1, 40)
1/1 [=====] - 0s 24ms/step
[[13.3635700e-10 6.7379731e-01 1.3064747e-05 2.2438372e-02 5.1548828e-05
 5.0068661e-03 5.7084144e-06 4.8493435e-11 7.2179845e-04 2.9796544e-01]]]
array(['car_horn'], dtype='<U16')
    
```

Рисунок 16. Тестирование нейронной сети с альтернативными параметрами на внешних аудиофайлах со звуком гудка автомобиля

При изменении параметров Dense в большую (со 100 нейронов до 500), увеличилась точность нейронной сети в процентах (88,49%), но также увеличилось время обучения нейронной сети. Тесты нейронных сетей одинаковы с первым тестом – наблюдается тот же самый результат, где 5 из 6 файлов распознаны правильно, а один распознан неверно, причём это тот же самый файл с гудком автомобиля. Исходя из данных тестов, можно сказать, что для решение поставленной задачи достаточно 78,71% точности нейронной сети. Увеличение нейронов в параметре Dense ведёт к

увеличению времени обучения нейронной сети, при неизменном результате относительно первого экспериментального тестирования.

Выводы

Таким образом была разработана программа с алгоритмом машинного обучения, при использовании библиотек TensorFlow и Librosa. Несмотря на точность в 78,71%, модель успешно определила к какому классу относится входящий аудиофайл из тестирующего набора, а в эксперименте с другими внешними, то есть не входящими в dataset файлами, нейронная сеть повела себя также удовлетворительно – 5 из 6 аудиофайлов были отнесены к верным классам. Уменьшение параметра Dense привело к существенным ошибкам классификации файлов, в то время как существенное увеличение параметров Dense привело к увеличению времени обучения нейронной сети, но не к улучшению нейронной сети в тестах.

Библиографический список

1. Астапов Р.Л., Мухамадеева Р.М. Автоматизация подбора параметров машинного обучения и обучение модели машинного обучения // Актуальные научные исследования в современном мире. 2021. № 5-2 (73). С. 34-37.
2. Горбунов П.М., Мацкевич Ю.А., Чубарь А.В. Машинное обучение. Автоматизация подбора модели машинного обучения // В сборнике: Робототехника и искусственный интеллект. материалы XIII Всероссийской научно-технической конференции с международным участием. Министерство науки и высшего образования Российской Федерации, Сибирский федеральный университет, Межинститутская базовая кафедра «Прикладная физика и космические технологии». 2021. С. 155-160.
3. Даулет С.Ж. Извлечение данных в twitter, предварительная обработка и анализ настроений с использованием python 3.0 // Молодой ученый. 2020. № 13 (303). С. 260-264.
4. Мельникова В.А., Медведев Д.А. Анализ больших данных с использованием python // Труды Братского государственного университета. Серия: Естественные и инженерные науки. 2019. Т. 1. С. 46-49.
5. Григорьев Е.А., Климов Н.С. Разведочный анализ данных с помощью python // E-Scio. 2020. № 2 (41). С. 165-176.
6. Ермаков О.А., Брозгунова Н.П. Python - как инструмент для анализа данных // Наука и Образование. 2020. Т. 3. № 4. С. 26.