Санкт-Петербургский государственный университет
Факультет Прикладной математики и процессов управления

Распознавание эмоций по голосу говорящего

Выполнили студенты СПбГУ
ПМ-ПУ группы 21.Б03-ПУ:
Ваганов Иван Станиславович
st102632@student.spbu.ru
Сыдыгалиева Бегаим Нурбековна
st075536@student.spbu.ru

Санкт-Петербург
2024 г.

Тема: Распознавание эмоциональной окраски голоса человека

Описание Задачи:

Анализ и применение различных подходов классификации эмоционального окраса по аудио данным:

Первый подход:

Одномерная сверточная нейронная сеть, с использованием библиотеки keras

Библиотеки:

1. Dusha:

Набор данных состоит из более 300 000 аудиозаписей с расшифровками и эмоциональными метками. Длительность составляет около 350 часов аудио.

Четыре основных эмоции: радость, грусть, злость и нейтральная эмоция.

2. CREMA-D

CREMA-D - это набор данных из 7442 оригинальных клипов, снятых 91 актером. В этих роликах приняли участие 48 актеров мужского и 43 женского пола в возрасте от 20 до 74 лет, представляющих различные расы и этнические группы

В работе рассмотрены три основные эмоции: радость, нейтральность и грусть

Четыре различных уровня эмоций: низкий, средний, высокий и неопределенный.

В качестве признаков были извлечены числовые характеристики звука:

Характеристики:

- Мел-частотные кепстральные коэффициенты

Представляют собой набор признаков , которые описывают общую форму спектральной огибающей. Они моделируют характеристики человеческого голоса.

- Спектральный контраст

Высокие значения контрастности обычно соответствуют четким узкополосным сигналам, а низкие значения контрастности соответствуют широкополосным шумам

- Спектральный центроид

Указывает, на какой частоте сосредоточена энергия спектра или, другими словами, указывает, где расположен “центр масс” для звука.

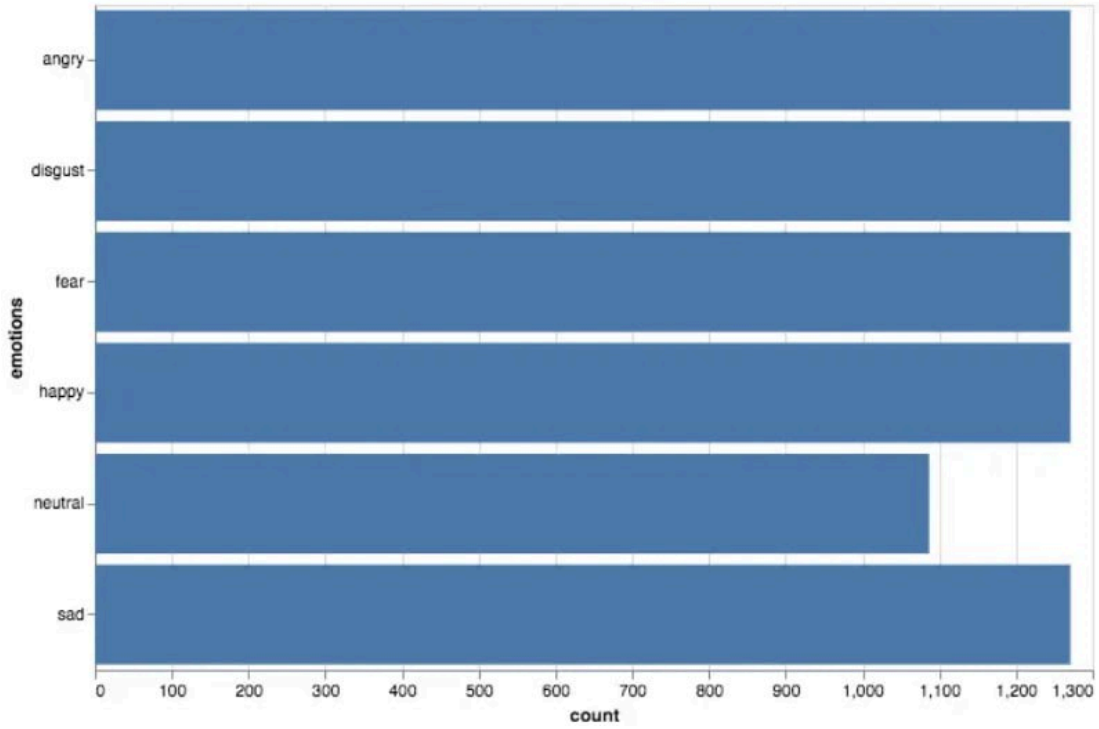
Для работы с аудиоданными были использованы библиотеки: `librosa, IPython`

```
podcast_train_data = pd.DataFrame(columns=['annotator_emo', 'duration', 'mfccs_feature', 'cent_mean', 'contrast'])
i=0
for index,audio_path in enumerate(podcast_train_.audio_path):
    path = path_podc_train + podcast_train_['audio_path'][i]
    print(i, path)
    X, sample_rate = librosa.load(path)
    sample_rate = np.array(sample_rate)

    mfccs = [np.mean(e) for e in librosa.feature.mfcc(y=X,
                                        sr=sample_rate,
                                        n_mfcc=20)]

    cent_mean = np.mean(librosa.feature.spectral_centroid(y=X, sr=sample_rate).T,
                        axis = 0)[0]
    contrast = np.mean(librosa.feature.spectral_contrast(y=X, sr=sample_rate).T,
                        axis = 0)[0]
    #tonnetz = np.mean(librosa.feature.tonnetz(y=X, sr=sample_rate),axis = 0)
    #np.mean(librosa.feature.spectral_rolloff(y=y, sr=sr).T, axis = 0)[0])      # rolloff_mean

    podcast_train_data.loc[i] = [podcast_train_['annotator_emo'][i]] + [podcast_train_['duration'][i]] + [mfccs]+ [np.array(cent_mean)] + [np.array(contrast)]

    i=i+1
```

В качестве предобработки данных были удалены значения с пустыми ячейками(объем таковых не превосходил 5 процентов от всего датасета),

В датасете Душа преобладал класс neutral, для того, чтобы сбалансировать классы мы рассмотрели количество самого малочисленного и отобрали аналогичное количество из остальных классов.
Так же удалены чересчур малочисленные классы

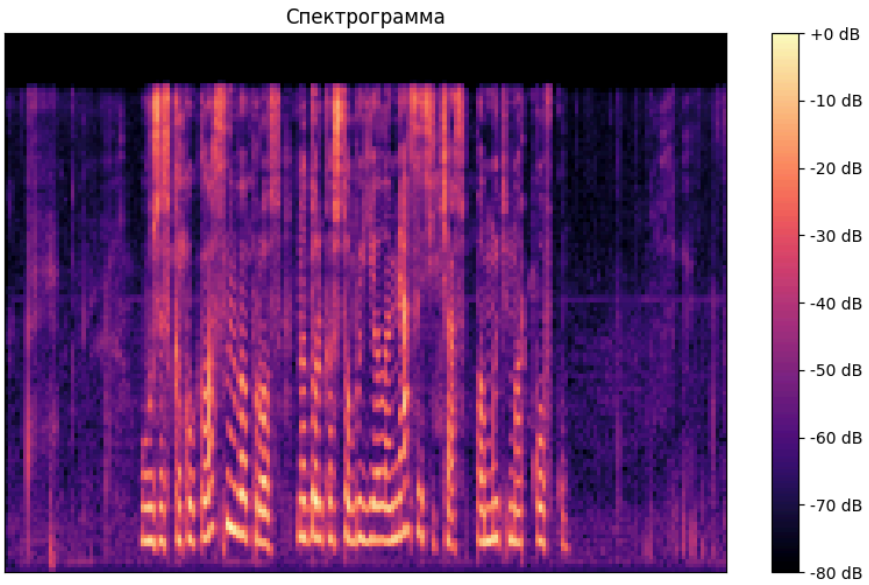
При работе со вторым датасетом проблем со сбалансированностью классов не наблюдалось



Второй подход:

Построение двумерной сверточной нейронной сети, с помощью библиотеки torch, для классификации мел-спектограммы

В качестве признаков были рассмотрены Мел-спектрограммы - это графическое представление спектра звука на основе мелового анализа.




```
path_data_crowd_test_image = "data_test/"

image_crowd_test_data = pd.DataFrame(columns=['annotator_emo', 'duration', 'img_name'])
i=0

for index,audio_path in enumerate(crowd_test_image.audio_path):
    path = path_crowd_test + crowd_test_image['audio_path'][i]
    print(i)
    signal, sr = librosa.load(path)
    mel_spect = librosa.feature.melspectrogram(y=signal, sr=sr, n_mels=128,  n_fft=2048, hop_length=1024)
    mel_spect = librosa.power_to_db(mel_spect, ref=np.max)
    # Визуализация спектрограммы
    plt.figure(figsize=(10, 6))
    librosa.display.specshow(mel_spect, fmax=8000)
    plt.savefig(path_data_crowd_test_image + str(i) + '.png',  bbox_inches='tight', pad_inches=0)
    image_crowd_test_data.loc[i] = [crowd_test_image['annotator_emo'][i]] + [crowd_test_image['duration'][i]] + [str(i) + '.png']
    plt.clf()
    plt.close()
    i=i+1
```

Архитектуры нейронных сетей

Для решения задачи нами были выбраны два подхода:

- Построение двумерной сверточной нейронной сети, с помощью библиотеки torch, для классификации мел-спектограммы
- Построение одномерной сверточной нейронной сети, с помощью библиотеки keras, для классификации эмоций, используя мел-кепстральные коэффициенты

Архитектура первой нейронной сети содержит в себе 5 сверточных слоев с функцией активации relu и один линейный слой выход, предсказывающий принадлежность к классам:

```
class SimpleCnn(nn.Module):

    def __init__(self, n_classes):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=8, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)
        )
        self.conv4 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)
        )
        self.conv5 = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=96, kernel_size=3),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)
        )

        self.out = nn.Linear(96 * 5 * 5, n_classes)
```

Архитектура второй нейронной сети содержала в себе два сверточных слоя, после которых шел слой dropout и maxpooling1d, три сверточных слоя, после которых шел слой dropout и maxpooling1d, сверточный слой, за ним последний линейный

слой, который на выходе возвращал вероятности принадлежности к искомым классам:

```
model_keras_final = keras.models.Sequential([
    keras.layers.Conv1D(64, kernel_size = 3, strides = 1, input_shape = (52, 1)), keras.layers.Activation('relu'),
    keras.layers.Conv1D(64, kernel_size = 3, strides = 1), keras.layers.Activation('relu'),
    keras.layers.Dropout(0.25),
    keras.layers.MaxPooling1D(pool_size = 2, strides = 2),

    keras.layers.Conv1D(128, kernel_size = 3), keras.layers.Activation('relu'),
    keras.layers.Conv1D(128, kernel_size = 3), keras.layers.Activation('relu'),
    keras.layers.Conv1D(128, kernel_size = 3), keras.layers.Activation('relu'),
    keras.layers.Dropout(0.25),
    keras.layers.MaxPooling1D(pool_size = 2, strides = 2),

    keras.layers.Conv1D(256, kernel_size = 3), keras.layers.Activation('relu'),

    keras.layers.Flatten(),
    keras.layers.Dense(3), keras.layers.Activation('softmax')
])
```

Двумерная сверточная нейронная сеть, библиотека torch

Для двумерной сверточной сети было необходимо создать датасет, который будет подгружать изображения мел-спектограммы, а также их разметки. Данная задача осуществлялась с Dataset и Dataloader из библиотеки torch.

Был реализован класс Emotions_dataset, который наследовал класс Dataset.

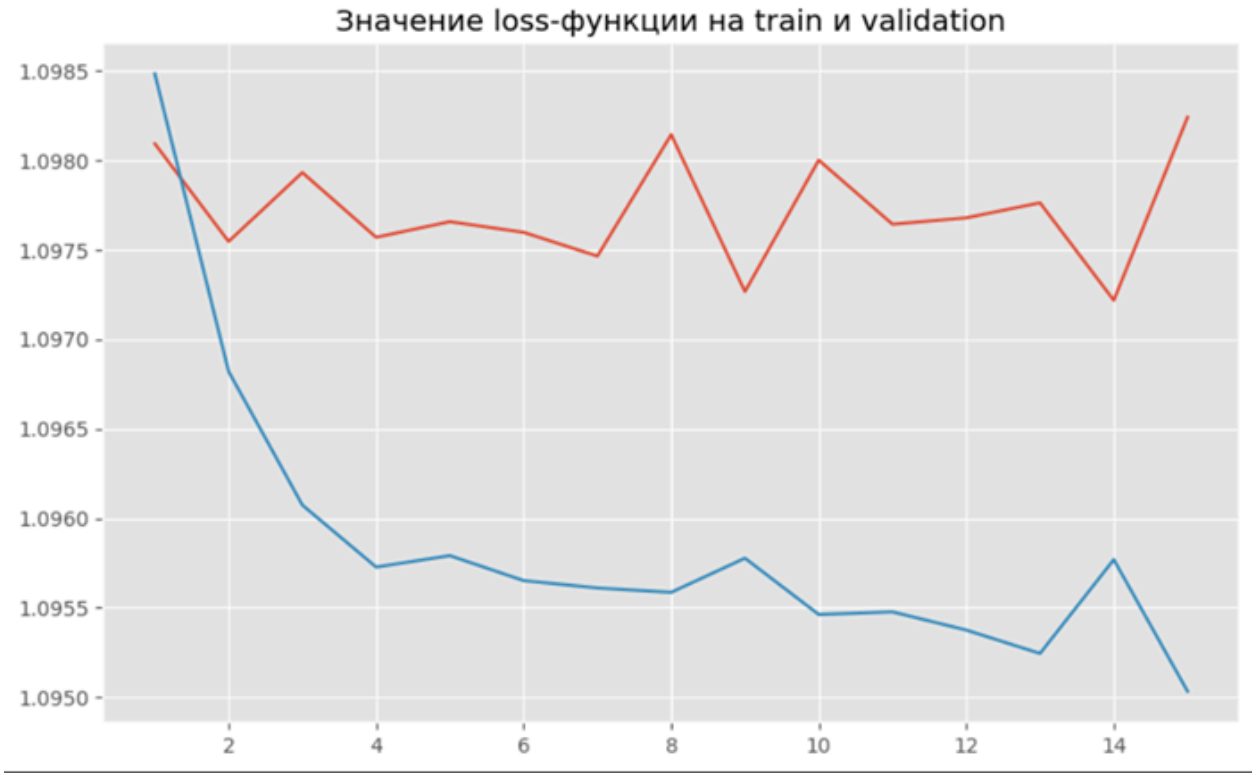
Также для процесса обучения были реализованы функции train, fit_epoch, eval_epoch:

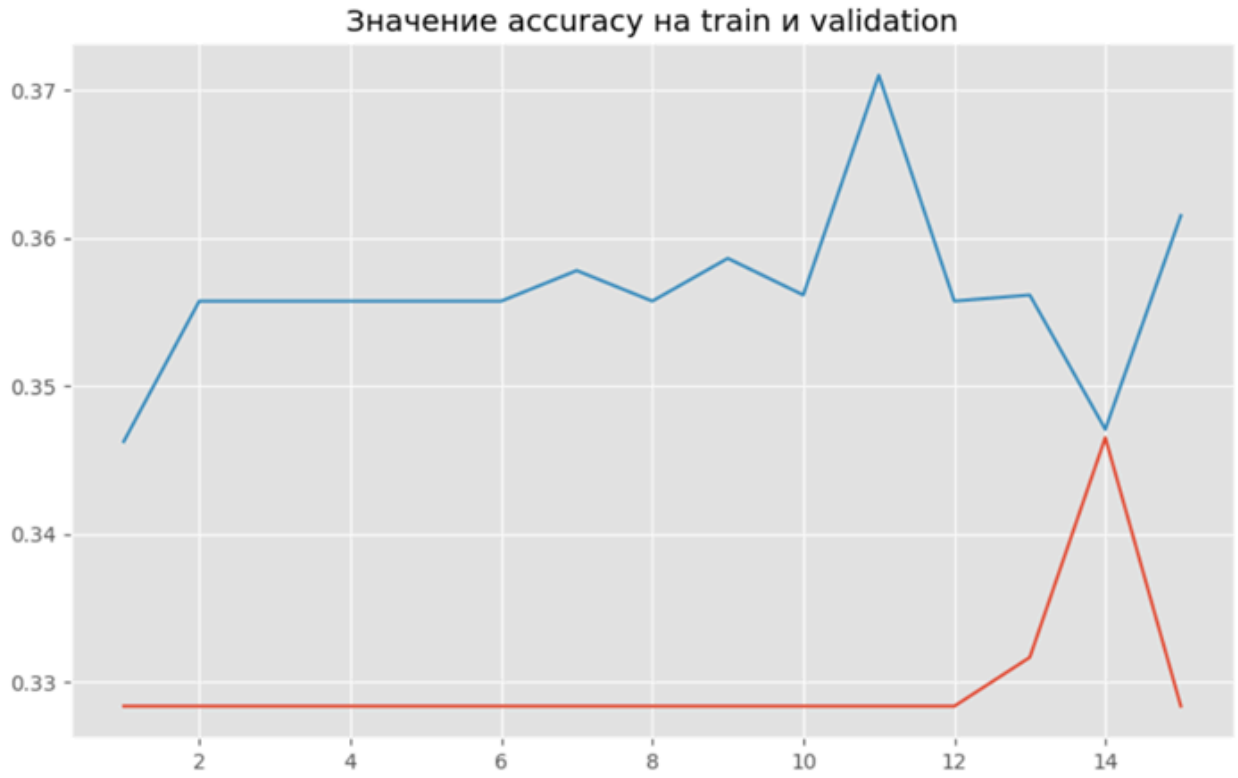
```
def train(model, train_files, val_files, epochs, batch_size, lr):
  train_loader = DataLoader(train_files, batch_size = batch_size, shuffle = True)
  val_loader = DataLoader(val_files, batch_size = batch_size, shuffle = False)

  history = []
  log_template = "\nEpoch {ep:03d} train_loss: {t_loss:0.4f} \
  val_loss {v_loss:0.4f} train_acc {t_acc:0.4f} val_acc {v_acc:0.4f}"

  with tqdm(desc = 'epoch', total = epochs) as pbar_outer:
    optim = torch.optim.Adam(model.parameters(), lr)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(epochs):
      train_loss, train_acc = fit_epoch(model, train_loader, criterion, optim)
      print('loss: ', train_loss)

      val_loss, val_acc = eval_epoch(model, val_loader, criterion)
      history.append((train_loss, train_acc, val_loss, val_acc))

      pbar_outer.update(1)
      tqdm.write(log_template.format(ep = epoch + 1, t_loss = train_loss, v_loss = val_loss, t_acc = train_acc, v_acc = val_acc))

  return history
```

Гиперпараметры:

- Размер батча: 128
- Количество эпох: 15
- Learning rate: 1e-4
- Оптимайзер: adam
- Функция ошибки: CrossEntropyLoss
- Метрика: accuracy

Итоговое время обучения: 9 минут





Одномерная сверточная нейронная сеть, библиотека keras

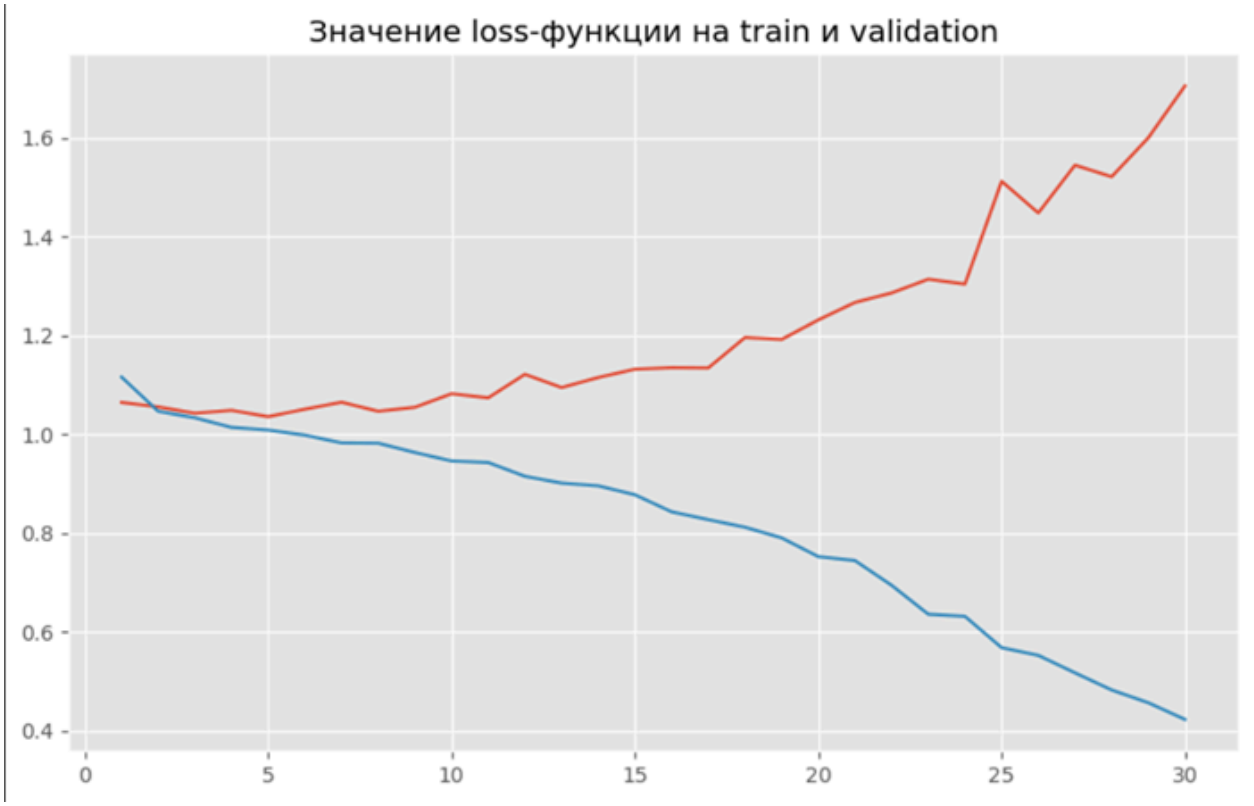
Для обучения одномерной сверточной сети были извлечены мел-кепстральные коэффициенты.

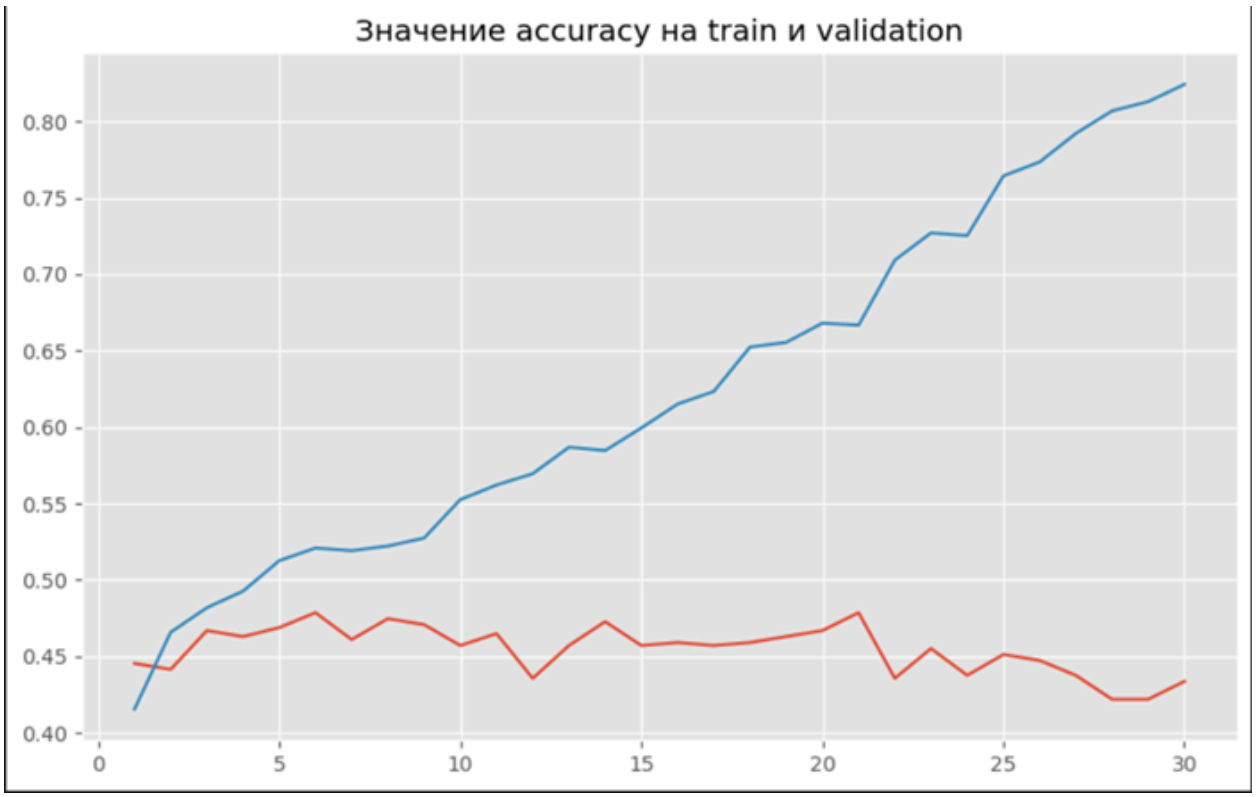
Гиперпараметры:

- Размер батча: 128
- Количество эпох: на датасете crema – 30, на датасете ‘душа’ – 100
- Оптимайзер: adam
- Функция ошибки: categorical crossentropy
- Метрика: accuracy
- Learning rate: 1e-4

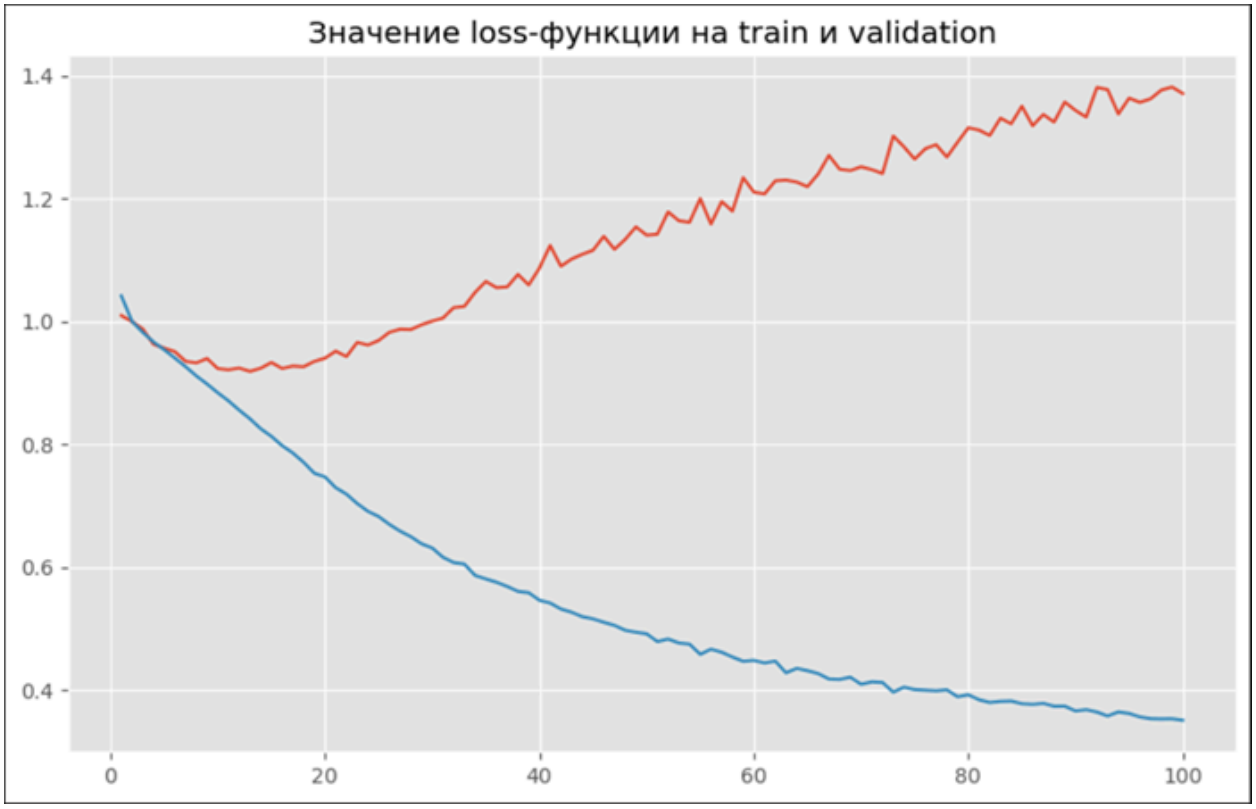
Итоговое время обучения: на датасете crema – минута, на датасете ‘душа’ – 35 минут

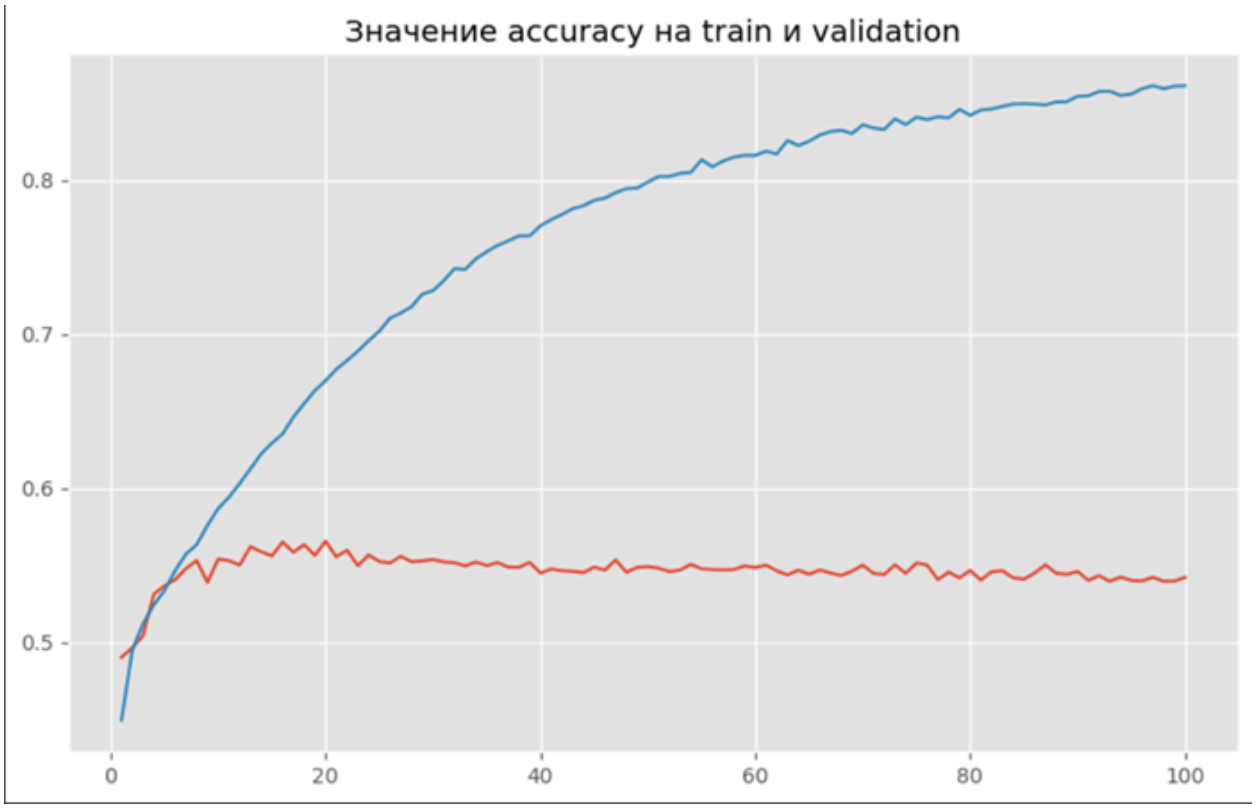
Графики датасета crema:





Графики датасета ‘душа’:


Значение loss-функции на train и validation
1.4
1.2
1.0
0.8
0.6
0.4
0
20
40
60
80
100


Значение accuracy на train и validation
0.8
0.7
0.6
0.5
0
20
40
60
80
100

Итоги

Подход с применением двумерной сверточной нейронной сети не оправдал себя, точность модели не превышала одной трети, что в нашем случае с 3 классами равносильно случайному попаданию.

Одномерная сверточная нейронная сеть, как видно на графиках, успешно обучалась только первые несколько эпох, затем значение функции ошибок на валидации переставало уменьшаться, а в последствии начинало увеличиваться, а значение метрик accuracy выходило на плато. Но этот подход показал себя намного лучше и может быть многообещающим в случае глубокой работы с данными, в частности аугментация с применением зашумленности может сильно повысить обобщающую способность модели и повысить точность предсказаний.