Описание работы

Владимир Димитров
email: v.dimitrov@g.nsu.ru

14 октября 2022 г.

Содержание

1 Данные

Набор данных ESC-50 представляет собой размеченную выборку из 2000 записей окружающей среды. Набор данных состоит из 50 классов, которые можно распределить по 5 главным категориям:

1. Животные
2. Звуки природы
3. Звуки человека
4. Бытовые звуки
5. Звуки города

Для каждого из класса доступно по 50 наблюдений. Давайте обучать наш классификатор предсказывать один из 50 классов. Для этого сначала необходимо удобно представить данные, потом найти подходящую архитектуру, используя статьи или посты в интернете.

2 Классы

2.1 TaskConfig

Для удобной работы с кодом создадим класс TaskConfig, в котором есть возможность редактирования параметров как модели, так и данных. Он умеет создавать вавку более длинной (за это отвечает переменная n_mels), можно поиграться с длиной батча и шагом обучения (learning_rate, batch_size). 'Ужать' нашу модель с помощью параметра регуляризации (weight_decay), выбрать размер тренировочной выборки и многое другое.

```python
@dataclasses.dataclass
class TaskConfig:
    keyword: str = '?-*-*-*.wav'  # Как сохранена вавка
    batch_size: int = 128
    learning_rate: float = 3e-4
    weight_decay: float = 1e-5
    num_epochs: int = 20
    n_mels: int = 64
    cnn_out_channels: int = 8
    kernel_size: Tuple[int, int] = (5, 20)
    stride: Tuple[int, int] = (2, 8)
    hidden_size: int = 64
    gru_num_layers: int = 2
    bidirectional: bool = False
    num_classes: int = 50
    sample_rate: int = 44_100
    n_fft = 1024
    win_length = 1024
    hop_length = 256
    device: torch.device = torch.device(
        'cuda:0' if torch.cuda.is_available() else 'cpu')
    path2noise = r'C:\VS_code\Deep Learning\Audio\audio\Noise'
    train_ratio = 0.9
    length_wav = 220_500
    length = 862
```

Рис. 1: Реализация класса TaskConfig

Более подробное описание представлено в главе 5.1.

2.2 Dataset

```python
class ESC_50(Dataset):
    SR = TaskConfig.sample_rate
    """
    Каждый формат имеет следующую структуру:
        {FOLD}-{CLIP_ID}-{TAKE}-{TARGET}.wav

        * {FOLD} - index of the cross-validation fold,
        * {CLIP_ID} - ID of the original Freesound clip,
        * {TAKE} - letter disambiguating between different fragments from the same Freesound clip,
        * {TARGET} - class in numeric format [0, 49].

    Здесь нас интересует только target)
    """
    def __init__(self, path2dir=None,transform=None,csv=None):
        self.transform=transform
        if csv is None:
            path2dir = pathlib.Path(path2dir)
            self.paths = list(path2dir.rglob(TaskConfig.keyword))
            twins = []
            for j in range(len(self.paths)):
                path_to_wav = self.paths[j].as_posix()
                label = int(path_to_wav.split('/')[-1].split('-')[-1].split('.')[0])
                twins.append((path_to_wav, label))
            self.csv = pd.DataFrame(
                twins,
                columns = ['path', 'label']
            )
        else:
            self.csv = csv

    def __getitem__(self, index):
        instance = self.csv.iloc[index]

        path2wav = instance['path']
        wav, sr = torchaudio.load(path2wav)
        wav = wav.sum(dim=0) # убираем размерность

        if self.transform:
            wav = self.transform(wav)

        return {
            'wav':wav,
            'label':instance['label']
        }

    def __len__(self):
        return len(self.csv)
```

Рис. 2: Реализация класса ESC_50

Данный класс имеет типичную структуру: метод get и метод len. Get вызывает элемент соответствующего индекса, а метод len возвращает длину. Из интересного - это инициализатор, в который подается путь до наших файлов, а он находит метки класса (необходимо чтобы они были в имени файла) и создает csv файл с двумя колонками: путь до файла и метка класса.

Более подробное описание представлено в главе 5.2.

2.3 Augmentation

```python
class AugsCreation:
    def __init__(self):
        self.background_noises = ['/content/drive/MyDrive/Noise/dude_miaowing.wav',
                                  '/content/drive/MyDrive/Noise/running_tap.wav',
                                  '/content/drive/MyDrive/Noise/pink_noise.wav',
                                  '/content/drive/MyDrive/Noise/exercise_bike.wav',
                                  '/content/drive/MyDrive/Noise/white_noise.wav']

        self.noises = [
            torchaudio.load(p)[0].squeeze()
            for p in self.background_noises
        ]

    def add_rand_noise(self, audio):

        # randomly choose noise
        noise_num = torch.randint(low=0, high=len(
            self.background_noises), size=(1,)).item()
        noise = self.noises[noise_num]
        noise_level = torch.Tensor([1])  # [0, 40]
        noise_energy = torch.norm(noise)
        audio_energy = torch.norm(audio)
        alpha = (audio_energy / noise_energy) * \
            torch.pow(10, -noise_level / 20)
        start = torch.randint(
            low=0,
            high=max(int(noise.size(0) - audio.size(0) - 1), 1),
            size=(1,)
        ).item()
        noise_sample = noise[start: start + audio.size(0)]
        audio_new = audio + alpha * noise_sample
        audio_new.clamp_(-1, 1)
        return audio_new

    def __call__(self, wav):
        aug_num = torch.randint(low=0, high=5, size=(1,)).item()  # choose 1 random aug from augs
        augs = [
            lambda x: x,
            lambda x: (x + distributions.Normal(0, 0.01).sample(x.size())).clamp_(-1, 1),
            lambda x: librosa.effects.time_stretch(x.numpy().squeeze(), 2.0)
            lambda x: librosa.effects.pitch_shift(x.numpy().squeeze(), sr, -5)
            lambda x: torchaudio.transforms.Vol(.25)(x),
            lambda x: self.add_rand_noise(x)

        ]

        return augs[aug_num](wav)
```

Рис. 3: Реализация класса Augmentation

Класс Augmentaion реализован на примере статьи. В данном классе есть 5 различных видов аугментации:

- Gaussian Noise - довольно простой метод аугментации, связан с добавлением случайной величины распределенной нормально.
- Time Stretching - ускорение вавки. Можно использовать наивный подход и брать каждый второй элемент нашей вавки, а можно использовать библиотеку librosa (Для более подробного ознакомления читать здесь [2]).
- Pitch Shifting - эта аугментация изменяет высоту голоса. Например, трансформация, которая представлена в классе делает голос более "тяжелым".

- Volume - увеличивает уровень громкости.
- Noise - добавляет случайные звуки

Еще существуют аугментации, которые накладываются на MEL-спектограммы, в статье [1] подробно рассказываются как их можно реализовать. Также есть интересный модуль.

2.4 Collator

```
class Collator:
    def __call__(self, data):
        lengths = []
        wavs = []
        labels = []

        for el in data:
            wavs.append(el['wav'])
            labels.append(el['label'])
            lengths.append(len(el['wav']))

        wavs = pad_sequence(wavs, batch_first=True)
        labels = torch.Tensor(labels).long()
        length = torch.Tensor(lengths).long()
        return {
            'wav':wavs,
            'label': labels,
            'length': length
        }
```

Рис. 4: Реализация класса Collator

Данный класс необходим для удобного обращения с данными. Через него можно указывать модели по какой переменной обучаться, по какой считать лосс, а по какой нормализовать (если это требуется).

2.5 Формирование данных

```
train_dataloader = DataLoader(
    train_set, batch_size=TaskConfig.batch_size,
    shuffle=False, collate_fn=Collator(),
    sampler=train_sampler,
    num_workers=2, pin_memory=True
)
val_loader = DataLoader(
    val_set, batch_size=TaskConfig.batch_size, shuffle=False,
    collate_fn=Collator(),
    num_workers=2,
    pin_memory=True
)
```

Рис. 5: Создание датасетов

В DataLoader подаем наши данные, которые мы сначала обработали с помощью класса ESC-50. Sampler необходим для несбалансированной выборки, он пересчитывает веса в зависимости от количества наблюдений одной метки, которые попали в train (Реализация данной функции представлена в главе 5.3). Шафл не производится, так как индексы по которым мы доставали наши данные - случайны. Переменная pin_memory оптимизирует работу нашей модели путем резервирования памяти (Более подробно можно почитать здесь).

2.6 Featurizer

```
class Featurizer(nn.Module):

    def __init__(self):
        super(Featurizer, self).__init__()

        self.featurizer = torchaudio.transforms.MelSpectrogram(
            sample_rate=TaskConfig.sample_rate,
            n_fft=1024,
            win_length=1024,
            hop_length=256,
            n_mels=64,
        )

    def forward(self, wav, length=None):
        mel_spectrogram = self.featurizer(wav)
        mel_spectrogram = mel_spectrogram.clamp(min=1e-5).log()

        if length is not None:
            length = (length - self.featurizer.win_length) // self.featurizer.hop_length
            # We add `4` because in MelSpectrogram center==True
            length += 1 + 4

            return mel_spectrogram, length

        return mel_spectrogram
```

Рис. 6: Реализация класса Featurizer

Данный класс трансформирует нашу вавку и переводит ее в Mel-спектрограмму (Более подробно о Mel-спектограмме здесь). Сам алгоритм будет на вход

получать вавку, которую мы пропустили через класс Featurizer. Из интересного, это нормализация вавки в зависимости от ее длины. Так как в реальной жизни, скорее всего, данные имеют разную длину, то конкретно данная фитча явялется достаточно полезным дополнением. Почитать более подробно про класс можно в главе 5.4

3 Model

Замечание: В дальнейшем все модели будут обучаться на 10 эпохах. Это связано с вычислительными мощностями, которые в Google Colab строго регламентируются. Обучение будет происходить на основе CrossEntropy loss. Оптимизатор будет - Adam. Другие параметры модели можно найти в классе TaskConfig (5.1).

3.1 Baseline

```
class SimpleModel(nn.Module):
  def __init__(self,config: TaskConfig):
    super().__init__()
    self.config = config
    self.conv1 = nn.Conv2d(in_channels=1, out_channels=config.cnn_out_channels,kernel_size=config.kernel_size)
    self.conv2 = nn.Conv2d(in_channels = config.cnn_out_channels, out_channels =config.cnn_out_channels, kernel_size = config.kernel_size)
    self.conv3 = nn.Conv2d(in_channels = config.cnn_out_channels, out_channels =config.cnn_out_channels, kernel_size = config.kernel_size)
    self.number = 20904
    self.pool = nn.MaxPool2d(4)
    self.flatten = nn.Flatten()
    self.linear1 = nn.Linear(self.number, int(self.number / 4))
    self.linear2 = nn.Linear(int(self.number / 4), int(self.number / 4 / 16))
    self.linear3 = nn.Linear(int(self.number / 4 / 16), config.num_classes)

  def forward(self, batch):
    batch = F.relu(self.conv1(batch.unsqueeze(dim=1)))
    batch = F.relu(self.conv2(batch))
    batch = F.relu(self.conv3(batch))
    batch = self.pool(batch)
    batch = self.flatten(batch)
    batch = F.relu(self.linear1(batch))
    batch = F.relu(self.linear2(batch))
    batch = F.relu(self.linear3(batch))
    return batch
```

Рис. 7: Baseline model

Давайте рассмотрим достаточно простую архитектуру, которую будем считать в качестве безлайна.

1. VGG блок с 8 каналами. В него входят 3 сверточных слоя с ядром 3*3 и нелинейностью ReLU. Перед входом в линейный слой применяется Max pooling с ядром 4*4

2. 3 полносвязных слоя между которыми применяется нелинейность ReLU.

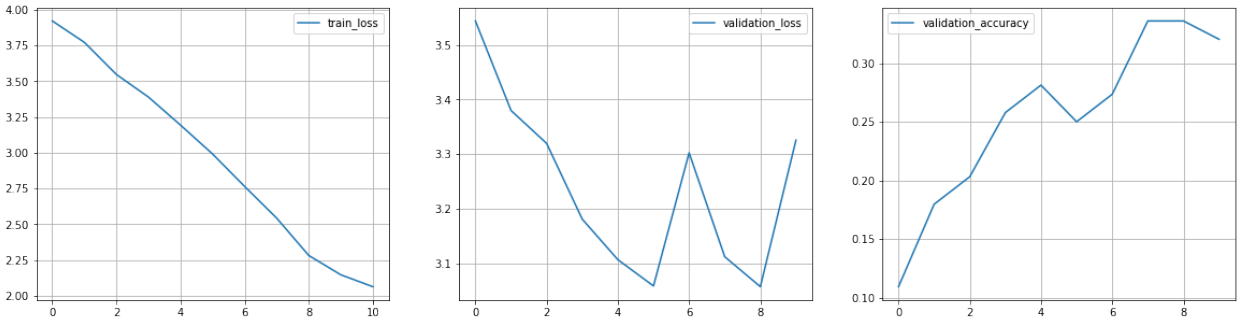

Рис. 8: Результаты базовой модели

Как видим потенциал у модели достаточно неплохой, на валидации точность составила порядка 33 процента.

3.2 AdvancedModel

Особенность VGG блока заключается в том, что повышение качества работы сети достигается за счет увеличения числа последовательных блоков. При этом число фильтров в каждом новом блоке в 2 раза больше, чем в предыдущем. Усложним нашу модель, используя также **Dropout** и **Batch Normalize**.

Правда сравнивать такие архитектуры достаточно не корректно, ведь для более глубоких сетей необходимо большее число эпох. Так как мощностей все также мало, мы докрутим шаг.

```python
class AdvancedModel(nn.Module):
    def __init__(self, config:TaskConfig):
        super().__init__()
        self.config = config
        # block 1
        self.conv1_1 = nn.Conv2d(in_channels=1, out_channels=config.cnn_out_channels,kernel_size=config.kernel_size, padding=1) # Conv2d(1, 8, kernel_size=(5, 20), stride=(1, 1), padding=(1, 1))
        self.conv1_2 = nn.Conv2d(in_channels = config.cnn_out_channels, out_channels =config.cnn_out_channels, kernel_size = config.kernel_size, padding=1)
        self.norm1 = nn.BatchNorm2d(config.cnn_out_channels)
        self.drop1 = nn.Dropout(0.2)

        # block 2
        self.conv2_1 = nn.Conv2d(in_channels=config.cnn_out_channels, out_channels=config.cnn_out_channels * 2,kernel_size=config.kernel_size, padding=1)
        self.conv2_2 = nn.Conv2d(in_channels = config.cnn_out_channels * 2, out_channels =config.cnn_out_channels * 2, kernel_size = config.kernel_size, padding=1)
        self.norm2 = nn.BatchNorm2d(config.cnn_out_channels * 2)
        self.drop2 = nn.Dropout(0.2)
        # block 3
        self.conv3_1 = nn.Conv2d(in_channels=config.cnn_out_channels * 2,out_channels=config.cnn_out_channels * 4,kernel_size=config.kernel_size, padding=1)
        self.conv3_2 = nn.Conv2d(in_channels = config.cnn_out_channels * 4, out_channels =config.cnn_out_channels * 4, kernel_size = config.kernel_size, padding=1)
        self.norm3 = nn.BatchNorm2d(config.cnn_out_channels * 4)
        self.drop3 = nn.Dropout(0.2)
        # block linear
        self.number = 9984
        self.pool = nn.MaxPool2d(2,2)
        self.flatten = nn.flatten()
        self.linear1 = nn.Linear(self.number, int(self.number / 4))
        self.norm4 = nn.BatchNorm1d(int(self.number / 4))
        self.drop4 = nn.Dropout(0.2)
        self.linear2 = nn.Linear(int(self.number / 4), int(self.number / 4 / 16))
        self.norm5 = nn.BatchNorm1d(int(self.number / 4 / 16))
        self.drop5 = nn.Dropout(0.5)
        self.linear3 = nn.Linear(int(self.number / 4 / 16), config.num_classes)

    def forward(self, batch):
        # first block
        batch = F.relu(self.norm1(self.conv1_1(batch.unsqueeze(dim=1))))
        batch = F.relu(self.norm1(self.conv1_2(batch)))
        batch = self.pool(batch)
        batch = self.drop1(batch)
        # second block
        batch = F.relu(self.norm2(self.conv2_1(batch)))
        batch = F.relu(self.norm2(self.conv2_2(batch)))
        batch = self.pool(batch)
        batch = self.drop2(batch)
        # third block
        batch = F.relu(self.norm3(self.conv3_1(batch)))
        batch = F.relu(self.norm3(self.conv3_2(batch)))
        batch = self.pool(batch)
        batch = self.drop3(batch)
        # flatten block
        batch = self.flatten(batch)
        # linear first block
        batch = self.norm4((self.linear1(batch)))
        batch = self.drop4(F.relu(batch))
        # linear second block
        batch = self.norm5((self.linear2(batch)))
        batch = self.drop5(F.relu(batch))
        # linear third block
        batch = self.linear3(batch)
        return batch
```

Рис. 9: Реализация улучшенной модели

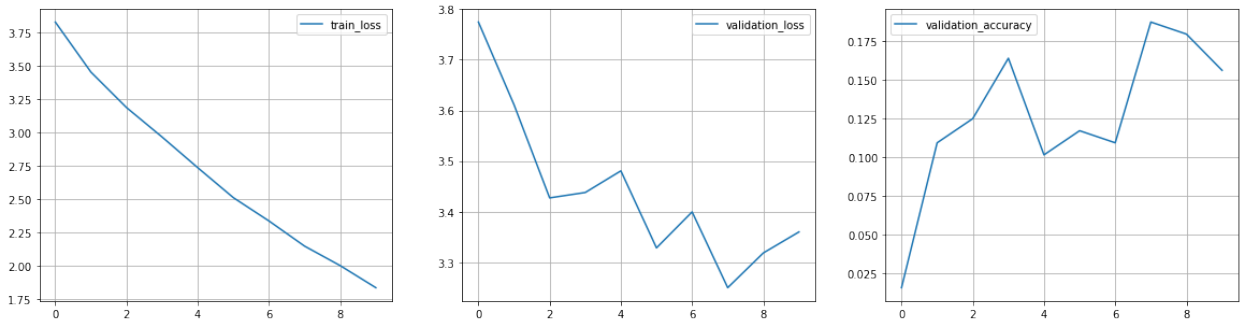

Рис. 10: Результаты улучшенной модели

Как мы видим, увеличение шага в два раза не спасло и модель недообучилась, поэтому необходимо в дальнейшем увеличить шаг еще больше или брать большее количество эпох.

3.3 LSTM

Давайте рассмотрим рекуррентные нейронные сети с долгосрочной памятью, возьмем однослойнную LSTM (Идейный вдохновитель). Преимущество данной сети в том, что она может аккумулировать прошлую информацию, которая находится на достаточно дальнем расстоянии.

```python
class Model(nn.Module):

    def __init__(self, input_dim, hidden_size):
        super().__init__()
        self.input_dim = input_dim
        self.hidden_size = hidden_size

        self.rnn = nn.LSTM(input_size=input_dim, hidden_size=hidden_size, batch_first=True)
        self.clf = nn.Linear(hidden_size, 50)

    def forward(self, input, length=None):
        output, _ = self.rnn(input.transpose(-1, -2))
        last_hidden = torch.gather(
            output,
            dim=1,
            # substract 1 because index start from 0 (not 1 as length)
            index=length.sub(1).view(-1, 1, 1).expand(-1, -1, self.hidden_size)
        )

        logits = self.clf(last_hidden.squeeze(dim=1))

        return logits
```

Рис. 11: Реализация LSTM

Из интересных особенностей, здесь был использован torch.gather, который позволяет быстрее доставать индексы из тензора, нежели, чем это делать через обычные команды pythona (Почитать более подобно про torch.gather можно здесь).

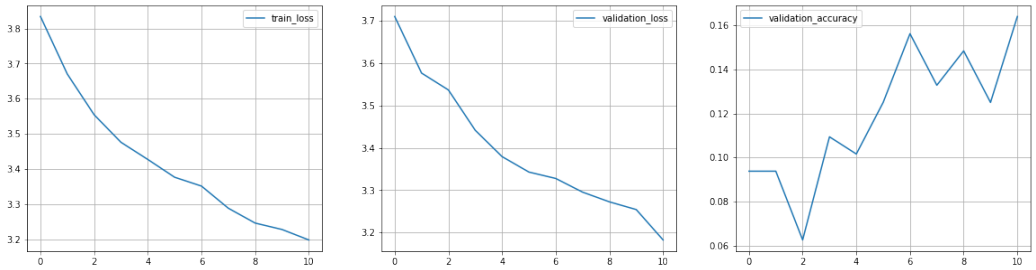

Рис. 12: Результаты LSTM

Данная модель хоть и не является глубокой (поэтому сравнение будет корректно), но она показала более худший результат на тех же 10 эпоха обучения, по сравнению с базовой моделью. Возможно информация, которая получает сеть из более ранних слоев, не содержит никаких полезных свойств, а возможно необходимо добавить сверточные слои,

которые хорошо себя показали у базовой модели. Это имплементация будет реализована дальше.

3.4 CRNN-A

Данная архитектура объединяет несколько моделей в себе. Она использует:

- рекуррентные нейронные сети, которые позволяют агрегировать информацию в различные периоды времени.
- сверточные сети, которые позволяют улавливать сложные закономерности.
- механизм внимания, который позволяет сконцентрироваться на более значимых признаках.

```python
class Attention(nn.Module):

    def __init__(self, hidden_size: int):
        super().__init__()

        self.energy = nn.Sequential(
            nn.Linear(hidden_size, hidden_size),
            nn.Tanh(),
            nn.Linear(hidden_size, 1)
        )

    def forward(self, input):
        energy = self.energy(input)
        alpha = torch.softmax(energy, dim=-2)
        return (input * alpha).sum(dim=-2)

class CRNN(nn.Module):

    def __init__(self, config: TaskConfig):
        super().__init__()
        self.config = config

        self.conv = nn.Sequential(
            nn.Conv2d(
                in_channels=1, out_channels=config.cnn_out_channels,
                kernel_size=config.kernel_size, stride=config.stride
            ),
            nn.Flatten(start_dim=1, end_dim=2),
        ) # kernel_size - отвечает за размер свертки, out_channels - количество выходных каналов

        self.conv_out_frequency = (config.n_mels - config.kernel_size[0]) // \
            config.stride[0] + 1

        self.gru = nn.GRU(
            input_size=self.conv_out_frequency * config.cnn_out_channels,
            hidden_size=config.hidden_size,
            num_layers=config.gru_num_layers,
            dropout=0.1,
            bidirectional=config.bidirectional,
            batch_first=True
        )

        self.attention = Attention(config.hidden_size)
        self.classifier = nn.Linear(config.hidden_size, config.num_classes)

    def forward(self, input):
        input = input.unsqueeze(dim=1)
        conv_output = self.conv(input).transpose(-1, -2)
        gru_output, _ = self.gru(conv_output)
        contex_vector = self.attention(gru_output)
        output = self.classifier(contex_vector)
        return output
```

Рис. 13: CRNN-A модель

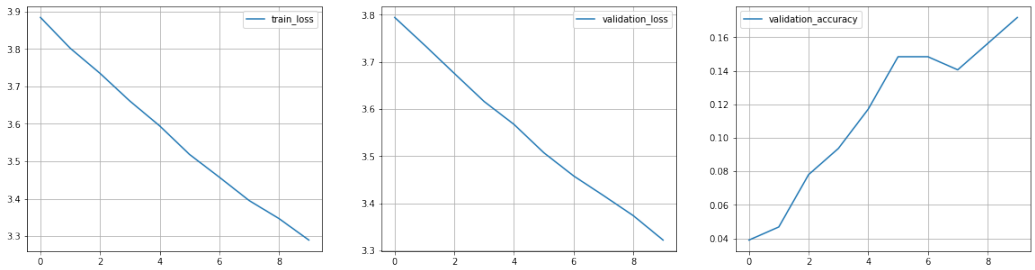

Рис. 14: Результаты CRNN модели

Все результаты необходимо воспринимать достаточно условно, ведь модели не успевают научиться обобщать наши данные. Поэтому стоит ее обучать чуть дольше, но так как вычислительных мощностей мало, давайте обратимся уже к предобученной модели.

3.5 Transfer learning: VGG19

В библиотеке torchvision имплементировано не только большое множество моделей (всевозможные ResNet'ы, Inception, VGG, AlexNet, DenseNet, ResNext, WideResNet, MobileNet...), но и загружены чекпоинты обучения этих моделей. Однако, из всей этой роскоши, нам понадобится только одно: модель VGG. Заменим 1 слой и последний для соответствия размерности.

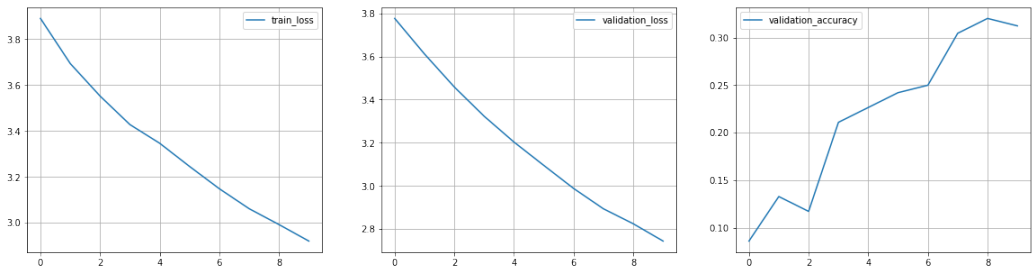

Рис. 15: Результаты VGG19

Обученная модель показала производительность на уровне baseline модели, возможно, это связано с тем, что модель изначально обучалась на ImageNet данных, поэтому ей необходимо чуть больше времени на донастройку параметров.

3.6 Github model

Код был взят отсюда. Код написан по статье [5]. Структура достаточно тривиальна, она вбирает в себя достаточно много сверток, которые на каждом шагу пулятся и от них берется нелинейность.

```
[ ] # Create by: https://github.com/anuragkr90/weak_feature_extractor

    class Githubmodel(nn.Module):

        def __init__(self, config:TaskConfig):
            super(Githubmodel,self).__init__()
            self.config = config
            self.layer1 = nn.Sequential(nn.Conv2d(1,16,kernel_size=3,padding=1),nn.BatchNorm2d(16),nn.ReLU())
            self.layer2 = nn.Sequential(nn.Conv2d(16,16,kernel_size=3,padding=1),nn.BatchNorm2d(16),nn.ReLU())
            self.layer3 = nn.MaxPool2d(2)

            self.layer4 = nn.Sequential(nn.Conv2d(16,32,kernel_size=3,padding=1),nn.BatchNorm2d(32),nn.ReLU())
            self.layer5 = nn.Sequential(nn.Conv2d(32,32,kernel_size=3,padding=1),nn.BatchNorm2d(32),nn.ReLU())
            self.layer6 = nn.MaxPool2d(2)

            self.layer7 = nn.Sequential(nn.Conv2d(32,64,kernel_size=3,padding=1),nn.BatchNorm2d(64),nn.ReLU())
            self.layer8 = nn.Sequential(nn.Conv2d(64,64,kernel_size=3,padding=1),nn.BatchNorm2d(64),nn.ReLU())
            self.layer9 = nn.MaxPool2d(2)

            self.layer10 = nn.Sequential(nn.Conv2d(64,128,kernel_size=3,padding=1),nn.BatchNorm2d(128),nn.ReLU())
            self.layer11 = nn.Sequential(nn.Conv2d(128,128,kernel_size=3,padding=1),nn.BatchNorm2d(128),nn.ReLU())
            self.layer12 = nn.MaxPool2d(2)

            self.layer13 = nn.Sequential(nn.Conv2d(128,256,kernel_size=3,padding=1),nn.BatchNorm2d(256),nn.ReLU())
            self.layer14 = nn.Sequential(nn.Conv2d(256,256,kernel_size=3,padding=1),nn.BatchNorm2d(256),nn.ReLU())
            self.layer15 = nn.MaxPool2d(2) #

            self.layer16 = nn.Sequential(nn.Conv2d(256,512,kernel_size=3,padding=1),nn.BatchNorm2d(512),nn.ReLU())
            # self.layer17 = nn.MaxPool2d(2) #

            self.layer18 = nn.Sequential(nn.Conv2d(512,1024,kernel_size=2),nn.BatchNorm2d(1024),nn.ReLU())
            self.layer19 = nn.Sequential(nn.Conv2d(1024,config.num_classes,kernel_size=1),nn.Sigmoid())

            self.flatten = nn.Flatten()
            self.layer20 = nn.Linear(1250, config.num_classes)

        def forward(self,x):
            out = self.layer1(x.unsqueeze(dim=1))
            out = self.layer2(out)
            out = self.layer3(out)
            out = self.layer4(out)
            out = self.layer5(out)
            out = self.layer6(out)
            out = self.layer7(out)
            out = self.layer8(out)
            out = self.layer9(out)
            out = self.layer10(out)
            out = self.layer11(out)
            out = self.layer12(out)
            out = self.layer13(out)
            out = self.layer14(out)
            out = self.layer15(out)
            out = self.layer16(out)
            #out = self.layer17(out)
            out = self.layer18(out)
            out1 = self.layer19(out)
            out1 = self.layer20(self.flatten(out1))
            #out = self.globalpool(out1,kernel_size=out1.size()[2:])
            #out = out.view(out.size(0),-1)
            return out1
```

Рис. 16: Github model

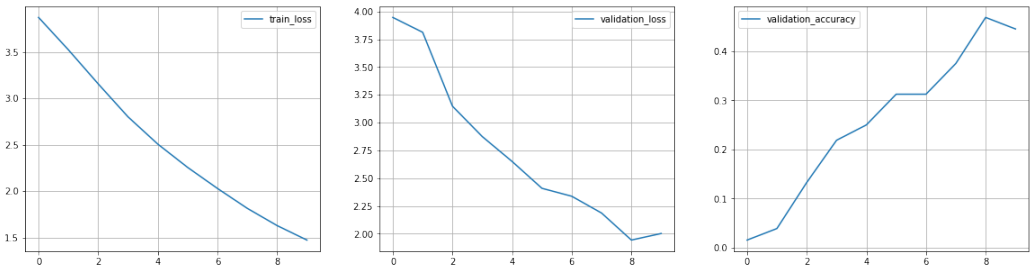

Рис. 17: Результаты Github model

Данная структура показала лучшую производительность среди необученных архитектур. На последней эпохе точность составила 42 процента. Автор статьи утверждает, что лучший результат составил 81 процент!

4 Вывод

Лучше всего из рассмотренных архитектур показали себя сверточные нейронные сети. Возможно, это связано с тем, что мы по сути предсказываем метку по Mel-спектрограмме, а последняя в свою очередь является ничем иным, как картинкой. А сверточные сети как раз хорошо работают именно с картинками. Также достаточно полезной была техника аугментации, кодирования весов в зависимости от количества классов, попадающих в тренировочную выборку, применения dropout и нормализации. В дальнейшем хотелось бы реализовать архитектуры, связанные с трансформерами, попробовать ML модели и поработать с более 'живыми' данными.

5 Приложение

5.1 Описание класса TaskConfig

- *keyword* - способ сохранения аудио, например: 'ABC-31-31-32.wav' в переменную надо подать '?-*-*-*.wav'
- *batch_size* - сколько данных попадет в батч
- *learning_rate* - длина шага обучения
- *weight_decay* - параметр регуляризации
- *num_epochs* - количество эпох обучения
- *n_mels* - количество мелов, на которые будет разбита вавка (влияет на размер входного тензора)
- *cnn_out_channels* - количество каналов выходов для сверточного слоя
- *stride* - длина шага
- *hidden_size* - размер скрытого слоя
- *gru_num_layers* - число рекуррентных слоев
- *bidirectional* - создание двунаправленной рекуррентной сети
- *num_classes* - число классов для предсказания
- *sampel_rate* - частота дискретизации
- *device* - девайс на котором будут производиться расчеты

5.2 Описание класса ESC_50

- *__init__* - подается путь до файла, в данном методе создается файл csv, для более удобного представления данных. Также возможна применение аугментации (в переменной transform)
- *__getitem__* - возвращает объект из csv файла по указанному индексу и применяет (если указано) аугментацию. На выходе возвращает словарь состоящий из тензорного представления вавки и метки класса
- *__len__* - возвращает длину нашего csv файла

5.3 Get_sampler

```python
# Чтобы взвешивать таргет
def get_sampler(target):
    class_sample_count = np.array(
        [len(np.where(target==t)[0]) for t in np.unique(target)])

    weight = 1. / class_sample_count
    samples_weight = np.array([weight[t] for t in target])
    samples_weight = torch.from_numpy(samples_weight)
    samples_weight = samples_weight.float()
    sampler = WeightedRandomSampler(samples_weight, len(samples_weight))
    return sampler

train_sampler = get_sampler(train_set.csv['label'].values)
```

Рис. 18: Реализация функции get_sampler

5.4 Описание класса Featurizer

- n_fft - количество амплитуд, на которые мы раскладываем вавку
- win_length - окно, с которым мы вырезаем вавку
- hop_length - длина шага, на которую мы сдвигаемся
- center - переменная, которая отвечает за паддинг тишины.В статьях написано, что введение данной переменной необходимо для того, чтобы не потерять начало сигнала из-за окна
- onesided - одно из свойств спектрограммы ее симметричность, данная функция возвращает только одну сторону (отрезает половину амплитуд).
- n_mels - параметр сжатия, отвечает за конечную размерность данных
- window_fn - окно, которое сглаживает нашу спектрограмму.

5.5 Github

Code

Список литературы

[1] Park D. S. et al. Specaugment: A simple data augmentation method for automatic speech recognition //arXiv preprint arXiv:1904.08779. – 2019.

[2] De Götzen A., Bernardini N., Arfib D. Traditional implementations of a phase-vocoder: The tricks of the trade //Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00), Verona, Italy. – 2000.

[3] Tian C., Ji W. Auxiliary multimodal LSTM for audio-visual speech recognition and lipreading //arXiv preprint arXiv:1701.04224. – 2017.

[4] Shan C. et al. Attention-based end-to-end models for small-footprint keyword spotting //arXiv preprint arXiv:1803.10916. – 2018.

[5] Kumar A., Khadkevich M., Fügen C. Knowledge transfer from weakly labeled audio using convolutional neural network for sound events and scenes //2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). – IEEE, 2018. – С. 326-330.