音视频场景识别实践报告

— 饶翔云,520030910366

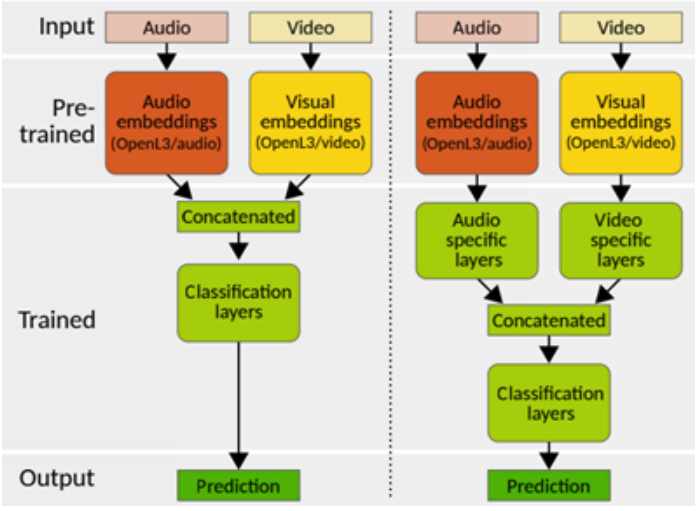
项目地址： `/lustre/home/acct-stu/stu143/projects/av_scene_classify/av_scene_classify`

0. 原理分析

多模态表征学习是一种新颖的学习方式。他旨在通过整合两种以上的识别技术，利用多重生物识别技术耦合起来的独特优势，并结合数据融合技术，让学习得到的表征更加可靠，让以此为基础的识别、认证过程更加精确安全。与传统单模态的学习方式不同的是，多模态表征学习通过采集不同种类的特征，建立多模态融合的表征方式，并通过分析、判断多种识别方式的特征进行识别和认证。

本次实践项目让我们完成一个多模态识别的任务。根据文档描述，该任务选取整合了不同场景下音频和视频的数据的TAU Audio-Visual Urban Scenes 2021数据集，要求我们完成一种通过分析音频和视频信息，提取特征，并最终用于训练网络，完成对不同场景的分类任务。本任务的意义在于，如果用单一的音频特征去完成场景识别工作，得到的结果具有混淆性、比如当输入汽车、电车和地铁上录制得到的音频时，网络很难判断出其属于那种场景下的声音。而通过增加了视频这一视觉特征，所得到的表征维度进一步提高，有利于网络去识别区分不同场景。

多模态识别任务中特征融合方式十分重要，做得好坏对结果的影响完全是1+1=2和1+1=100之间的区别。在本次任务中，我们通过将视频特征和音频特征融合后进行编码，再将编码通过分类器得到该场景所属类别。基于实践项目文档，本次任务我将会尝试三种不同的特征融合方式：



早期特征融合与中期特征融合

1. 早期特征融合

即将预编码好的音频特征和视频特征直接拼接，传入分类器中。

2. 中期特征融合

即将预编码好的音频特征和视频特征传入子编码网络，再将输出拼接，传入分类器中。

3. 晚期特征融合（基于之前的创新）

和中期类似，但是对从子编码网络中的输出额外进行分类，最终两个分类器进行投票得到最后结果。

三者各有优劣，比如早期特征融合简单，但是无法充分利用多模态特征的互补性；而中晚期实现复杂，但是由集成学习得到的结果错误率低，通常可靠。

实验中选取Acc和LogLoss作为评分指标，其中LogLoss公式为：

$$L_{log}(y,p) = -(y\log p + (1-y)\log(1-p))$$

1. 代码分析

老师给出的代码框架主要分为特征提取、训练和测试三部分，其中特征提取已经完成并保存，训练测试部分和寻常的分类神经网络类似。下面主要介绍不同融合方式的实现。

Baseline给出的是中期特征融合，其forward函数结构如下：

```
def forward(self, audio_feat, video_feat):
    # audio_feat: [batch_size, time_steps, feat_dim]
    # video_feat: [batch_size, time_steps, feat_dim]
    audio_emb = audio_feat.mean(1)
    audio_emb = self.audio_embed(audio_emb)

    video_emb = video_feat.mean(1)
    video_emb = self.video_embed(video_emb)

    embed = torch.cat((audio_emb, video_emb), 1)
    output = self.outputlayer(embed)
    return output
```

首先对于输入的音频、视频三维张量，首先先降维到二维（通过在第一维度取平均），然后通过不同的编码层得到长度为128的张量，并将他们的第一维拼接，得到融合特征，输入分类层，最终得到象征着各类别概率的张量。编码层和分类层如下：

```
self.audio_embed = nn.Sequential(
    nn.Linear(audio_emb_dim, 512),
    nn.BatchNorm1d(512),
    nn.ReLU(),
    nn.Dropout(p=0.2),
    nn.Linear(512, 128)
)
self.video_embed = nn.Sequential(
    nn.Linear(video_emb_dim, 512),
    nn.BatchNorm1d(512),
    nn.ReLU(),
    nn.Dropout(p=0.2),
    nn.Linear(512, 128)
)
self.outputlayer = nn.Sequential(
    nn.Linear(256, 128),
    nn.Linear(128, self.num_classes),
)
```

以上是中期特征融合。下面首先介绍早期特征融合的具体实现。我去除了两种特征各自的embed层，然后作为补偿构建了深度为4的总embed层。

```
self.early_output = nn.Sequential(
    nn.Linear(audio_emb_dim + video_emb_dim, 1024),
    nn.BatchNorm1d(1024),
    nn.ReLU(),
    nn.Dropout(p=0.2),
    nn.Linear(1024, 512),
    nn.BatchNorm1d(512),
    nn.ReLU(),
    nn.Dropout(p=0.2),
    nn.Linear(512, 256),
    nn.BatchNorm1d(256),
    nn.ReLU(),
    nn.Dropout(p=0.2),
    nn.Linear(256, 128),
    nn.Linear(128, self.num_classes)
)

def forward(self, audio_feat, video_feat)->None:

    audio_feat = audio_feat.mean(1)
    video_feat = video_feat.mean(1)

    feat = torch.cat((audio_feat video_feat), 1)
    output = self.early_output(feat)
    return output
```

然后是晚期特征融合。我构建了两个分类器，并最终通过取平均的方式投票。

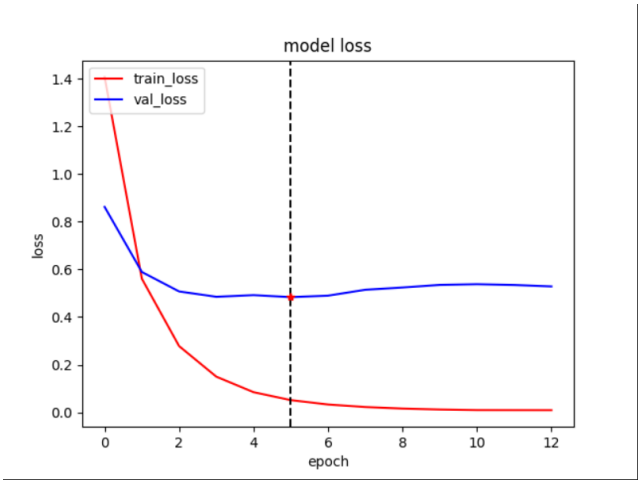
```
self.audio_output = nn.Sequential(
    nn.Linear(128, 128),
    nn.Linear(128, self.num_classes),
)
self.video_output = nn.Sequential(
    nn.Linear(128, 128),
    nn.Linear(128, self.num_classes),
)
def forward(self, audio_feat, video_feat):
    # audio_feat: [batch_size, time_steps, feat_dim]
    # video_feat: [batch_size, time_steps, feat_dim]
    audio_emb = audio_feat.mean(1)
    audio_emb = self.audio_embed(audio_emb)

    video_emb = video_feat.mean(1)
    video_emb = self.video_embed(video_emb)

    audio_out, video_out = self.audio_output(audio_emb),
    self.video_output(video_emb)
    output = (audio_out + video_out) / 2
    return output
```

2. 运行结果：

Baseline运行结果如下：

	precision	recall	f1-score	support
airport	0.78	0.71	0.74	281
bus	0.90	0.89	0.90	327
metro	0.86	0.84	0.85	360
metro_station	0.87	0.89	0.88	386
park	0.96	0.86	0.91	386
public_square	0.72	0.70	0.71	387
shopping_mall	0.76	0.73	0.74	387
street_pedestrian	0.71	0.72	0.71	421
street_traffic	0.85	0.94	0.89	402
tram	0.70	0.79	0.74	308
accuracy			0.81	3645
macro avg	0.81	0.81	0.81	3645
weighted avg	0.81	0.81	0.81	3645

accuracy: 0.809
overall log loss: 0.535



三者比较如下

	Early	Middle(Baseline)	Late
Acc	0.802	0.809	**0.828**
LogLoss	0.581	0.535	**0.495**

3. 模型修改以及参数调整

a. 对晚期融合的调参

batch_size	16	32	64	128	256
Acc	0.819	0.820	0.822	**0.828**	0.820
LogLoss	**0.481**	0.505	0.491	0.495	0.525

lr	1e-3	1e-4	1e-5
Acc	0.779	**0.828**	0.825
LogLoss	0.591	**0.495**	0.495

最后选择进行100个epoch、early-stop=7、lr=1e-4、batch_size=128的训练。

b. 对模型的修改

将晚期模型进一步改进，改成三个分类器按各自权重投票的方式。

```
def forward(self, audio_feat, video_feat):
    # audio_feat: [batch_size, time_steps, feat_dim]
    # video_feat: [batch_size, time_steps, feat_dim]
    audio_emb = audio_feat.mean(1)
    audio_emb = self.audio_embed(audio_emb)

    video_emb = video_feat.mean(1)
    video_emb = self.video_embed(video_emb)

    audio_out, video_out = self.audio_output(audio_emb),
    self.video_output(video_emb)
    mix_embed = torch.cat((audio_emb, video_emb), 1)
    mix_out = self.outputlayer(mix_embed)
    output = ((audio_out + video_out) / 2) * 0.8 + mix_out * 0.2
    return output
```

选用之前的超参数进行训练，最后得到Acc=82.2%，LogLoss=0.498的比baseline更好的结果

4. 实验总结

本实践项目与之前的纯语音或纯视觉的项目不同，通过多个模态的特征融合可以更好地对不同场景下的声源信息、视频信息进行分类，这也是我第一次接触多模态的项目，无论从原理上还是实践的结果上来看多模态的引入都极大地优化了场景分类任务下模型的性能。而不同的特征融合方式也对模型最终的性能有着或多或少的影响，其应用的领域也有所不同。本次实践的高阶要求部分我通过利用不同的特征融合方式进行投票的新模型，让最终结果较baseline提高，但是提高不多。