音视频场景识别实践报告

519030910383 张哲昊

1. 摘要

本次实践项目尝试对音视频场景检测进行了相应的实验与探索，包括基础部分：理解多模态融合工作不同表征获取的方式、模态融合的方式，分析有冲突的模态结果，替换为 early 特征融合和 late 决策融合的两类融合方式。以及高阶要求：修改模型、和超参数进行模型调优，改进模型性能。

2. 任务简介

音视频场景识别任务的输入为场景的声音和视频，需要通过神经网络输出该场景的标签。任务难点是如何将音频和视频的特征进行融合，使得模型性能比单模态输入更好。

3. 多模态融合工作不同表征获取的方式

本次实践项目使用 OpenL3 进行视频和音频的特征提取。L3 网络的网络结构如下：

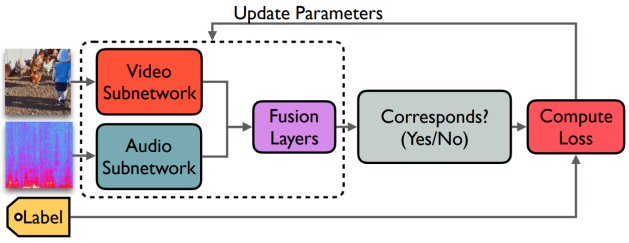

图 1: *L3 网络的结构*

其中音频特征为 Mel-frequency log-magnitude spectrograms。网络进行预训练时，抽取了一些带有声音的视频，不需要手动标注标签，预训练任务为判断视频与声音是否匹配。上述结构图中，视觉子网络部分和音频子网络都用了 4 层的卷积神经网络和最大池化层。两个子网络的输出拉伸之后再进行拼接，然后经过输出层得到两者是否匹配的概率。我们使用 oplenL3 库使用预训练模型来达到视觉和音频方向的特征。

4. 有冲突的模态结果

本次项目中将 baselien 中的音视频融合部分去掉，然后单独使用视频或音频子网络的输出输入到输出网络，得到以下的两个单模态的输出结果。每个类的结果如下表所示，以及分类的 heatmap 如下图所示。

表 1: 单独使用视频模态的分类结果

类别	precision	recall	f-1
airport	0.48	0.28	0.35
bus	0.64	0.82	0.72
metro	0.82	0.69	0.75
metro_station	0.75	0.85	0.80
park	0.85	0.82	0.83
public_square	0.53	0.58	0.56
shopping_mall	0.55	0.69	0.61
street_pedestrian	0.58	0.49	0.53
street_traffic	0.74	0.70	0.72
tram	0.50	0.48	0.49

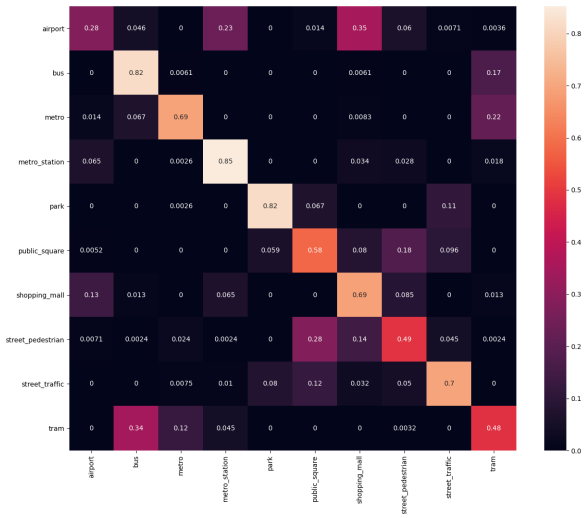

图 2: 单独使用视频模态的分类结果 *heatmap*

从上述实验结果中发现，airport，bus，shop-

表 2: 单独使用声音模态的分类结果

类别	precision	recall	f-1
airport	0.68	0.77	0.72
bus	0.78	0.72	0.75
metro	0.73	0.64	0.68
metro_station	0.66	0.70	0.68
park	0.92	0.77	0.84
public_square	0.57	0.66	0.61
shopping_mall	0.70	0.62	0.66
street_pedestrian	0.65	0.62	0.63
street_traffic	0.82	0.82	0.82
tram	0.62	0.76	0.69

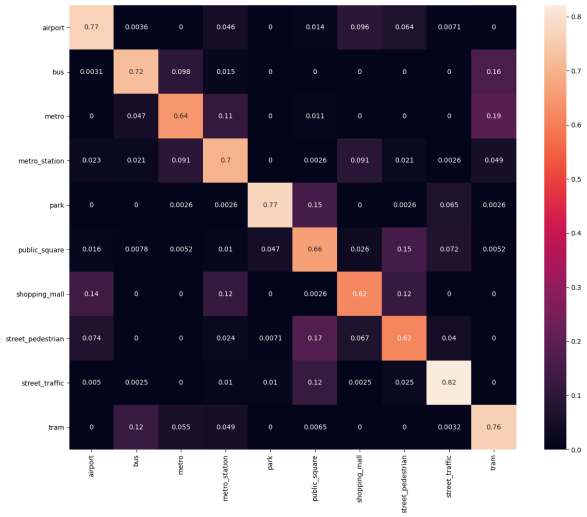

图 3: 单独使用声音模态的分类结果 *heatmap*

ping_mall 这三类场景，对于单独使用视频模态和声音模态的结果差距较大，有明显的冲突。

对于 airport 类，单用视频模态进行分类的 recall 极低，本次项目将 groud truth 中 label 为 airport 的场景分错成哪些类别输出得到表3的实验结果。从表3中的实验结果可以看出，单独使用视频特征的情况下，模型将大部分的 airport 类识别为了 shopping_mall 和 metro_station。其原因首先可能是 shopping_mall 与 airport 两类可能在视频场景中都有人来人往，同时人们手中都有袋子，箱子，背包等。metro_station 与 airport 类在视频层面容易混淆可能的原因是都有站台，旅客等相似场景。

对于 bus 类，同样进行了相似的结果分析，得到如表4所示的实验结果。从实验结果可以看出，模型的绝大多数错误判断都将 bus 类识别成了 tram。其原因可能是由于这两类场景都有类似的交通工具出现，同时区别十分微小，故使得模型在只有视觉特征时很难区分。

5. early 特征融合和 late 决策融合的两类融合方式

Baseline 中提供的模型分别将图像特征，音频特征都平均到一个维度，然后经过两个单独的子网络，然后将两个子网络的输出进行拼接，之后再经过一个输出层输出每个类别的概率。Baseline 的实验结果见表5。

5.1. early 特征融合

本次项目中进行了 early 特征融合，做法为首先将视频和音频的特征平均，然后通过全连接神经网络的输出层。模型代码如下：

```python
class Early_fusion(nn.Module):
    def __init__(self, audio_emb_dim, video_emb_dim, num_classes) -> None:
        super().__init__()
        self.num_classes = num_classes
        self.all_embed = nn.Sequential(
            nn.Linear(video_emb_dim+audio_emb_dim, 1024),
            nn.BatchNorm1d(1024),
            nn.ReLU(),
            nn.Dropout(p=0.2),
            nn.Linear(1024, 10)
        )
    def forward(self, audio_feat, video_feat):
        audio_emb = audio_feat.mean(1)
        video_emb = video_feat.mean(1)
        embed = torch.cat((audio_emb, video_emb), 1) #(128,1024)
        output = self.all_embed(embed)
        return output
```

使用 early 特征融合之后的实验结果见表6，图。

表 3: 单独使用视频模态在 *airport* 场景下的错误分类结果

类别	bus	metro	metro_station	park	public_square	shopping_mall	street_pedestrian	street_traffic
比例	0.06	0.0	0.3267	0.0	0.0198	0.49	0.001	0.005

表 4: 单独使用视频模态在 *airport* 场景下的错误分类结果

类别	airport	metro	metro_station	park	public_square	shopping_mall	street_pedestrian	street_traff
比例	0	0.03	0.32673	0.0	0.0198	0.03	0.001	0.933

表 5: *baseline* 的实验结果

模型	准确率	log loss
baseline	0.809	0.535

表 6: *early fusion* 和 *late fusion* 的实验结果

模型	准确率	log loss
early fusion	0.812	0.529
late fusion	0.816	0.497

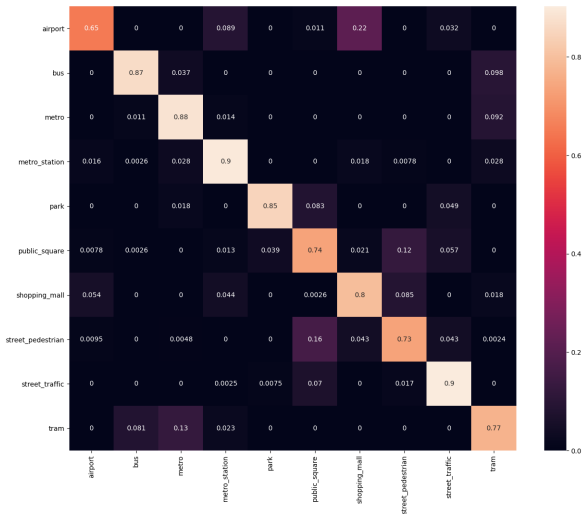

图 4: *early fusion* 分类结果 *heatmap*

5.2. late 决策融合

本次项目中将音频和视觉的子网络分别修改为输出为类别总数的全连接神经网络，然后将两个输出以各 0.5 的权重比例相加作为最终的输出概率。模型代码如下代码如下：

```
class late_fusion(nn.Module):
    def __init__(self, audio_emb_dim,
     video_emb_dim, num_classes,alpha = 0.5) ->
     None:
        super().__init__()
        self.num_classes = num_classes
        self.alpha = alpha
        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_video = nn.Sequential(
            nn.Linear(128, 256),
            nn.Linear(256, self.num_classes),
        )
        self.outputlayer_audio = nn.Sequential(
            nn.Linear(128, 256),
            nn.Linear(256, self.num_classes),
        )

    def forward(self, audio_feat, video_feat):
        audio_emb = audio_feat.mean(1)
        audio_emb = self.audio_embed(audio_emb)
        output_audio = self.outputlayer_audio(
    audio_emb)
        video_emb = video_feat.mean(1)
        video_emb = self.video_embed(video_emb)
        output_video = self.outputlayer_video(
    video_emb)
        output = (1-self.alpha)*output_audio +
    self.alpha*output_video
        return output
```

实验结果见表6，heatmap 见图5。实验结果表明这两种方式均相对于 baseline 有所提升。

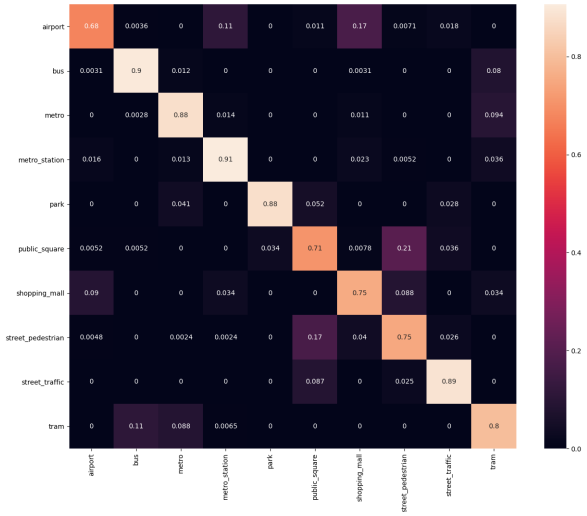

图 5: *late fusion* 分类结果 *heatmap*

6. 超参数与模型改进提高性能

6.1. 超参数调整

本次项目对于 late fusion 两种模态的输出的加权的权值 α。调整 α 在 [0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65,0.7,0.75,0.8,0.85,0.9] 的范围内。得到 α 与准确率的关系图如图6所示。

最优的 α 设定之下的实验结果如表7所示，

表 7: 最优 $\alpha = 0.25$ 的实验结果

模型	准确率	log loss
baseline	0.824	0.480

heatmap 如图7所示，训练和验证的损失函数曲线如图8所示。

6.2. 模型改进

由于 baseline 中对于视觉和音频上的特征都对于时间维度进行了平均，会产生一定的信息损失，因此本次项目尝试了更加复杂的网络信息来更好地利用特征。

6.2.1. 全连接神经网络

本次项目尝试了将每一个 batch 的特征进行展开而不是对时间维度取平均。代码实现如下：

```
audio_emb = torch.flatten(audio_feat,1)
video_emb = torch.flatten(video_feat,1)
```

然后经过维度较大的全连接层。

实验结果如表8所示：实验结果并不理想，可能的

表 8: 特征先展开然后通过全连接神经网络的结果

模型	准确率	log loss
baseline	0.798	0.585

原因是数据量太少，收敛效果不好。

6.2.2. LSTM

由于视频特征和音频特征都是时序特征，本次项目尝试使用 LSTM 对其进行建模并尝试提取出不同时间的特征之间的联系，以提高模型的性能。代码实现如下：

```
class Mynet(nn.Module):
    def __init__(self, audio_emb_dim, video_emb_dim, num_classes) -> None:
        super().__init__()
        self.num_classes = num_classes
        self.rnn1 =  nn.LSTM(512,256,num_layers=2, bidirectional=True,batch_first=True)
        self.rnn2 =  nn.LSTM(512,256,num_layers=2, bidirectional=True,batch_first=True)
        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),
        )

    def forward(self, audio_feat, video_feat):
```

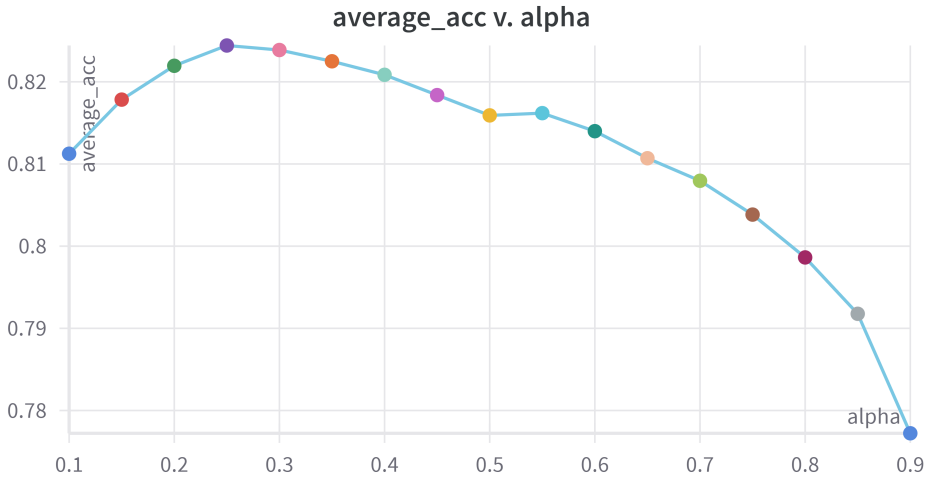

图 6: *alpha* 与准确率之间的关系

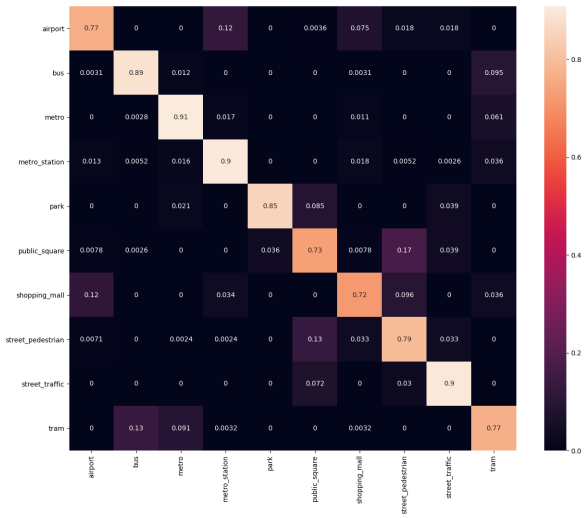

图 7: 最佳 $\alpha = 0.25$ 时分类结果 *heatmap*

```
        audio_emb,_ = self.rnn1(audio_feat)
        audio_emb = torch.flatten(audio_emb,1)
        audio_emb = self.audio_embed(audio_emb)
        video_emb,_ = self.rnn2(video_feat)
        video_emb = torch.flatten(video_emb,1)
        video_emb = self.video_embed(video_emb)
        embed = torch.cat((audio_emb, video_emb),
    1)
        output = self.outputlayer(embed)
        return output
```

实验结果见表 实验结果发现经过 LSTM 处理之后的实验效果并不尽如人意，可能的原因是 LSTM

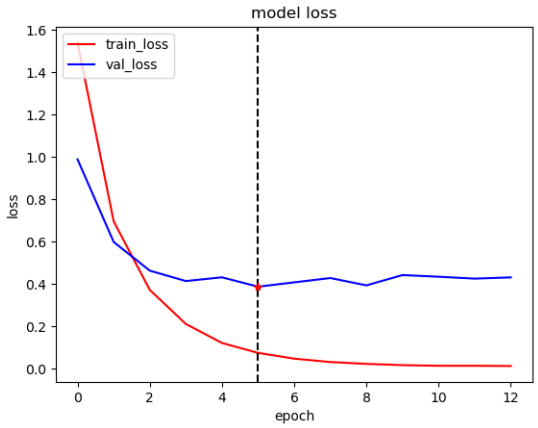

图 8: 最佳 $\alpha = 0.25$ 时训练集和验证集上的 *loss* 曲线

表 9: *LSTM* 对特征进行处理之后的实验结果

模型	准确率	log loss
baseline	0.777	0.645

处理之后的特征对于两个模态而言分布差距较大，简单的拼接之后通过全连接神经网络后无法有效提取两个维度的有效信息。