声音事件检测实践报告

— 饶翔云,520030910366

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

0. 原理分析

声音事件检测，即通过识别在各种场景下录制的不同声源的音频来判断音频中出现的声源或发生的事件，一般会通过输入给好标记的音频进行神经网络训练，从而对未知的某一场景下的音频进行预测。传统的声音事件检测采用强标签，即给定该音频中声源及发生事件的类别，同时会给定声音事件发生的起止时间戳，因此大体来说可以看做是分类的任务，需要得到每个音频中声源及事件的类别，同时预测出声音事件的起始位置。而本次实践项目采用弱标签，即只给出了声音事件类别，并没有给出起止时间戳。这种弱监督任务相较强监督任务更难，因为不能直接提取声音事件片段内的信息，因此需要其他方法来获取声音事件片段。任务提供的的数据集（即DESED 2020数据集）中有十类子事件，由在家庭环境中录制或合成以模拟家庭环境的10秒音频剪辑得到。

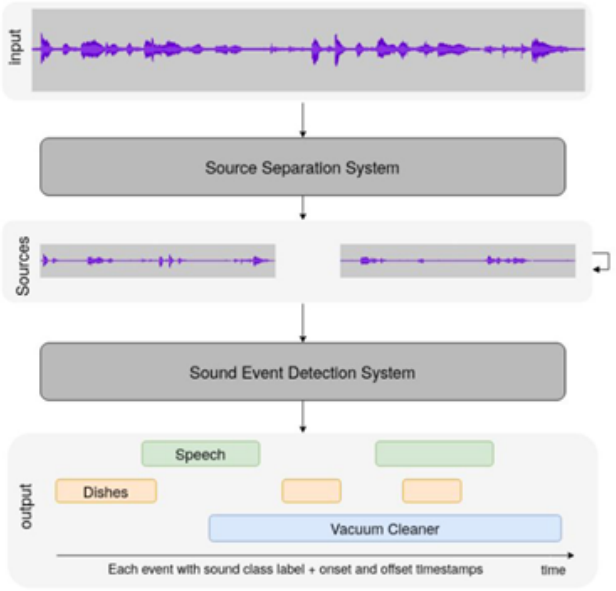
强弱标签对比

训练及测试部分与传统机器学习任务的步骤类似，baseline采用三层卷积模块的CRNN架构，即对输入的三维张量（三个维度分别代表batch size、时间步长和频数）进行多次卷积和maxpooling操作，并且通过BiGRU层后，用全连接层将其改变为batch size、时间步长和声音事件类别数对应的三维张量，以分数表示声音时间出现的概率。由于本任务是分类任务，因此在训练时计算损失的损失函数采用BCEloss。

$$Loss(x,y) = -x\log y + (1-x)\log(1-y)$$

基线模型是一种针对弱监督的声音事件检测系统。据文档中描述，此模型将声音分离与声音事件检测结合使用的系统，即使用模型分离重叠的声音事件，并从背景声音中提取前景声音事件。这里可以使用用于预处理的基线声音分离模型作为特征提取模块，即先将原音频经过基线模型后分离出类似声音事件的部分和背景音部分，然后再将其送入事件检测模型中。提取特征模块，即获取弱标签事件对应时间步长的方法已经给出。代码的基本思路是通过阈值区分背景音和声音事件，从而得到声音事件的起止时间戳，用于训练检测模型。

值得一提的是，在调用的脚本获取数据时，所需要的环境比较特殊，要求python环境安装有旧版本的librosa，否则会报错。



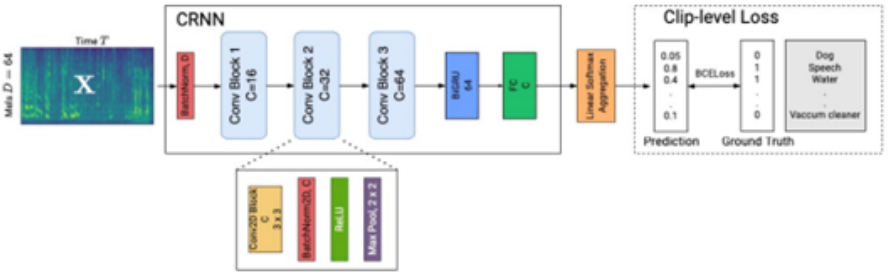
强弱标签对比

1. 代码实现

本次任务要求我们实现如文档中所示的模型结构。对输入三维张量数据（三个维度分别代表batch size、时间步长和频数）进行unsqueeze操作变换为四维，首先通过一个BatchNorm2d层做batch normalization操作，接下来分别进入三个卷积模块，每个卷积模块有一个二维卷积层，并继续进行batch normalization、relu和max pooling操作得到模块的输出，三个卷积模块的输入维度和输出维度分别为1->16,16->32,32->64。接下来将卷积模块的输出reshape为三维张量后通过一个单层的BiGRU层，最后用全连接层将其第三个维度对应到到声音事件的总类别数（10），最后从网络中输出，后续还需要通过线性的softmax回归层（在这里由于使用BCELoss即sigmoid层），得到预测的该段音频出现每一个声音事件的概率，并将其与ground truth对比得到loss。Baseline代码如下：

```
def __init__(self, inputdim, outputdim):
    ##############################
    # YOUR IMPLEMENTATION
    # Args:
    #     num_freq: int, mel frequency bins
    #     class_num: int, the number of output classes
    ##############################
    super(Crnn, self).__init__()
    self.num_freq = inputdim
    self.class_num = outputdim


    self.bn0 = nn.BatchNorm2d(64)
    self.bn1 = nn.BatchNorm2d(16)
    self.bn2 = nn.BatchNorm2d(32)
    self.bn3 = nn.BatchNorm2d(64)


    self.conv1 = nn.Conv2d(in_channels = 1, out_channels = 16,
                           kernel_size=(3, 3), stride=(1, 1),
                           padding = (1, 1), bias=False)

    self.conv2 = nn.Conv2d(in_channels = 16, out_channels = 32,
                           kernel_size=(3, 3), stride=(1, 1),
                           padding = (1, 1), bias=False)

    self.conv3 = nn.Conv2d(in_channels = 32, out_channels = 64,
                           kernel_size=(3, 3), stride=(1, 1),
                           padding = (1, 1), bias=False)

    self.bigru = nn.GRU(input_size = 64 * 8, hidden_size = 128,
    num_layers = 1, batch_first = True, bidirectional = True)
    self.fc = nn.Linear(256, outputdim, bias=True)

    self.relu = nn.ReLU()

def detection(self, x):
    ##############################
    # YOUR IMPLEMENTATION
    # Args:
    #     x: [batch_size, time_steps, num_freq]
    # Return:
    #     frame_prob: [batch_size, time_steps, num_class]
    ##############################
    batch_size, time_steps, num_freq = x.shape[0], x.shape[1], x.shape[2]
    x = x.unsqueeze(1)
    x = x.transpose(1, 3)
    x = self.bn0(x)
    x = x.transpose(1, 3)

    x = self.relu(self.bn1(self.conv1(x)))
    x = nn.functional.max_pool2d(x, kernel_size = (1, 2), stride = (1, 2))

    x = self.relu(self.bn2(self.conv2(x)))
    x = nn.functional.max_pool2d(x, kernel_size = (1, 2), stride = (1, 2))

    x = self.relu(self.bn3(self.conv3(x)))
    x = nn.functional.max_pool2d(x, kernel_size = (1, 2), stride = (1, 2))
```

```
59
60          x = x.transpose(1, 2)
61          x = x.reshape(batch_size, time_steps, -1)
62          x, _ = self.bigru(x)
63          x = torch.sigmoid(self.fc(x))
64
65          return x
```



模型架构

值得一提的是，由于2x2的池化层表现不好，我将2x2的池化层换成1x2的池化层，让特征向量只在特征维度上发生变化。

2. 实验结果

无初始化的baseline：

	f_measure	precision	recall
event_based	0.103453	0.0821127	0.168975
segment_based	0.605021	0.633168	0.584185
tagging_based	0.652854	0.704907	0.618575

mAP: 0.6921633860963843

有初始化的baseline:

	f_measure	precision	recall
event_based	0.124536	0.100468	0.199054
segment_based	0.59984	0.677639	0.551891
tagging_based	0.64412	0.712186	0.606888

mAP: 0.6970813445074393

3. 模型改进

卷积层数

首先尝试使用更多的卷积层。即在原来64维输出后再加一个128维的卷积块，以此类推。通过实验发现，使用baseline的效果最好，体现在event-f1 score最高。

添加了一层卷积层：

	f_measure	precision	recall
event_based	0.0810878	0.0609902	0.14534
segment_based	0.603121	0.635136	0.587384
tagging_based	0.651892	0.703494	0.620342

mAP: 0.7089784001261551

添加了两层卷积层：

	f_measure	precision	recall
event_based	0.104801	0.0847685	0.164822
segment_based	0.599651	0.698627	0.542786
tagging_based	0.653265	0.70865	0.617254

mAP: 0.7090126292548391

添加了三层卷积层：

	f_measure	precision	recall
event_based	0.0990031	0.0895508	0.156797
segment_based	0.550772	0.672999	0.496926
tagging_based	0.585278	0.702029	0.532202

mAP: 0.671683126660398

减掉一层卷积层：

	f_measure	precision	recall
event_based	0.0938055	0.0742739	0.149826
segment_based	0.565591	0.648304	0.513777
tagging_based	0.618738	0.703474	0.563749

mAP: 0.665923251512383

GRU层数

分别选用GRU层数为2,3，4：

层数为2：

	f_measure	precision	recall
event_based	0.156315	0.143853	0.18916
segment_based	0.585994	0.669415	0.530632
tagging_based	0.656607	0.705061	0.627788

mAP: 0.704467109277257

层数为3：

	f_measure	precision	recall
event_based	0.180065	0.164479	0.230796
segment_based	0.585314	0.63454	0.561748
tagging_based	0.642138	0.672748	0.622871

mAP: 0.6813355437005297

层数为4：

	f_measure	precision	recall
event_based	0.164828	0.15356	0.207774
segment_based	0.576002	0.628543	0.559364
tagging_based	0.626728	0.668175	0.60378

mAP: 0.6627647447584915

池化方式

avgpooling：

	f_measure	precision	recall
event_based	0.192007	0.176872	0.246644
segment_based	0.598698	0.661375	0.566875
tagging_based	0.662041	0.708126	0.634091

mAP: 0.6903039328558717

总结

最终选用启用初始化，三层卷积层，层数为2的GRU，avgpooling组成的模型作为最优模型。得到的最好性能指标为event-f1 score=0.192007，相比baseline提高了近两倍。

4. 音频增强方法

1. 向音频中插入随机噪声。
2. 转移时间，即随即将音频向前、向后移动（若向前移动x帧，则音频的后x帧标记为0，即静音）。
3. 改变音频的音高。
4. 使音频变速。
5. 时间扭曲，在音频中选择一个随机点并以距离w向左或向右扭曲，该距离从0到沿该线的时间扭曲参数W的均匀分布中选择。
6. 时域掩盖，将t个连续的时间步长[t_0,t_0+t]屏蔽，t选自从0到时间掩码参数T的均匀分布，且t_0选自[0,T-t]，T为音频的总长度。
7. 频域掩盖，将频率域的[f_0,f_0+f]部分屏蔽，f选自从0到频率掩码参数F的均匀分布，且f_0选自[0,v-f]，其中v为频率通道的数量（如80维的FBank）。

5. 实验总结

本次实验选取了声音事件检测领域中较困难但是较为实用的弱标签训练架构，让我们学习从数据预处理、特征提取到进行训练、测试的全过程，旨在让我们了解声音事件检测任务的大体流程，以及如何评测弱监督下的评测结果。通过调整模型的各级参数以及网络模型结构，实现了对event-f1 score的优化，让我对模型结构对模型性能的影响有了进一步的认识。同时这也是我的最后一个实践项目。感谢老师的无私帮助和徐徐教诲！