北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

机 器 学 习 基 础

专 题 作 业

学　　院　自动化学院

班　　级　06111704

姓　　名　危昊成

学　　号　1120171140

2020 年 12 月 27 日

题目：基于多分类的睡眠自动分期

1. 任务描述

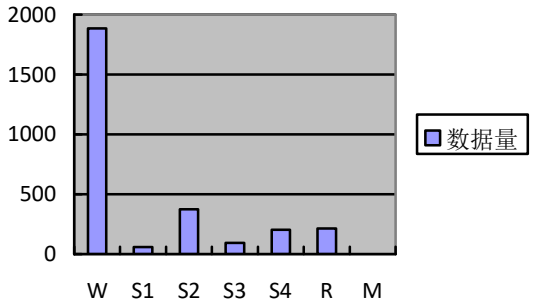
本次任务为 1 维时序信号的 7 目标的多分类问题。该问题的研究已经非常成熟，除通过基本的机器学习模型如 SVM、决策树进行分类之外，还可以通过简单的深度神经网络模型实现该问题的求解。在之前的模式识别课程中，我们通过一维卷积神经网络（1D-CNN）对轴承故障信号的多分类问题搭建了类似的模型[1][2]。该问题同样为一维信号，且对该信号进行 10 目标分类问题。与本次唯一不同的是该信号为 3000 个采样值的时序序列，较本次信号的序列更长。据此推论：在使用卷积神经网络时，除了全连接层的输入节点不同外，在结构上并没有过多差别。

故本次任务中拟采用一维卷积神经网络对数据进行分类，网络结构共包含 8 个卷积层，每个卷积层分别以一个对应大小的批标准化层，最大池化层和激活层连接，最后使用全连接层合并预测结果。

2. 数据描述，原始数据可视化（分类别可视化），每类数据样本数量（是否非平衡数据），预处理说明（噪声/缺值/滤波），训练集与测试集划分说明；

本次任务中，各分类的数据量如下表所示：

表 1 数据集各分类数据分布



W	S1	S2	S3	S4	R	M
1885	59	373	94	203	215	1

由表格可以看出，数据集并非平衡数据集，因此对模型的要求比较高，更可能会出现过拟合的情况。为适当减小模型的过拟合，在预处理时，给分段后的时

序信号添加少量的高斯噪声，其概率密度服从高斯分布，公式为：

$$f(x)=\frac{1}{\sqrt{2\pi\sigma}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

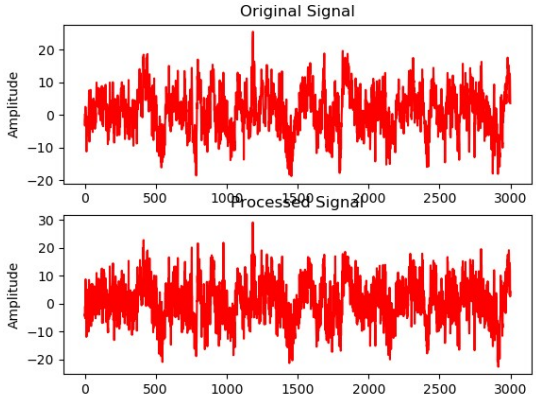


图 1 原信号切片(上图)和加了 σ =3.0 的高斯噪声的原信号切片(下图)

对于过拟合，在预处理时进行数据增广也可以适当减小过拟合现象，但这里没有对样本进行线性变换以扩充数量较少的分类，主要是考虑到数据集为睡眠波的脑电信号，该脑电信号在进行线性变换后，标签值是否会因此发生改变，题目要求中并未给出明确的证明。为防止幅值和正反序、正反相与标签值的耦合性较大的情况，在这里并没有使用数据增广的方法，相反我认为，盲目的数据增广是极其危险的。

在预处理时，对所给的时序信号进行切片，将原采样信号变为(2830，3000)大小的二维矩阵，即：切成 2830 个信号段，每段含 3000 个采样值。将切片后的信号和对应标签值存入 csv 文件，位于工程目录的./processed 文件夹中。

3. 特征提取

根据往次经验，对于该时序信号，在数据集足够丰富时并不需要通过统计方法添加特征，且通过统计方法添加的特征未必合理，特征之间很可能存在非常大的耦合性。因此在特征提取上，我提出了使用快速傅里叶变换(FFT)和不使用忍和特征处理直接将原信号带入网络的两种方法。与以往不同的是，对于本次的数据，在实际训练时，使用快速傅里叶变换将信号转化为频谱形式进行特征提取的方法的训练结果并不理想，将其带入 1D-CNN 模型时，训练迭代的准确率和损失函数值如图所示。

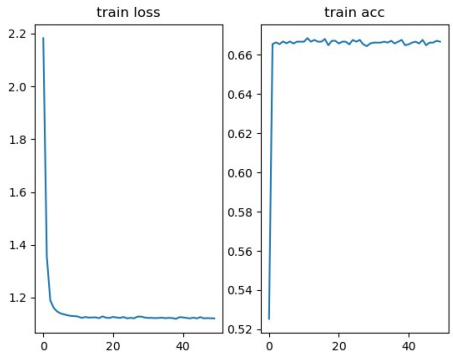


图 2 使用 FFT 进行特征提取后的训练结果

可以看出，训练的准确率在到达 0.66 之后就不再增加了，而是在附近不断摆荡。对于该问题我也非常费解，因此打印了部分信号段的频谱图，并查看了其类内和类间差距如下图所示。通过对比发现，相比于类内差距，类间差距并不明显。在快速傅里叶变换后由于只取了幅频特性而忽略了相频特性，因此很可能丢失了不少分类信息，导致分类失败。

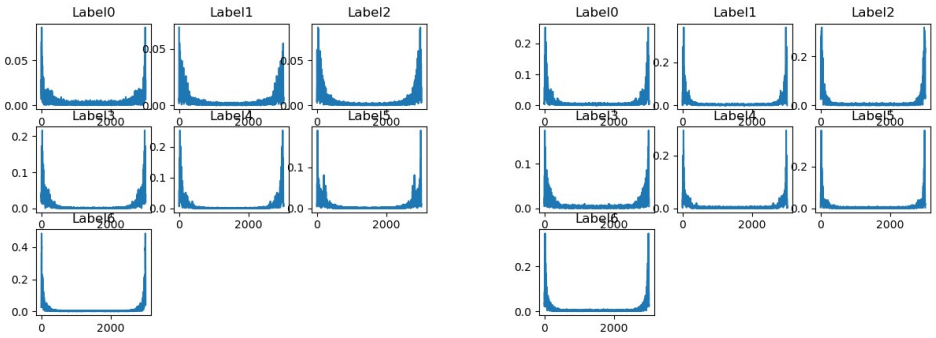


图 3 （左）类间差距（右）类内差距

因此最后进行分类时，对于原信号，我使用了不进行任何特征工程的策略，并充分发挥卷积神经网络的本身实力。

本问题的模型采用一维卷积神经网络，其网络结构较大，故可视化结构图可能不清晰，可在工程目录./eegnet_model.pdf 文件中查看。网络总共分为八层，每层由以下结构串联而成：

- 一维卷积层：Conv1d()
- 批标准化层：BatchNorm1d()
- 最大池化层：MaxPooling1D()
- Relu 层：ReLU()

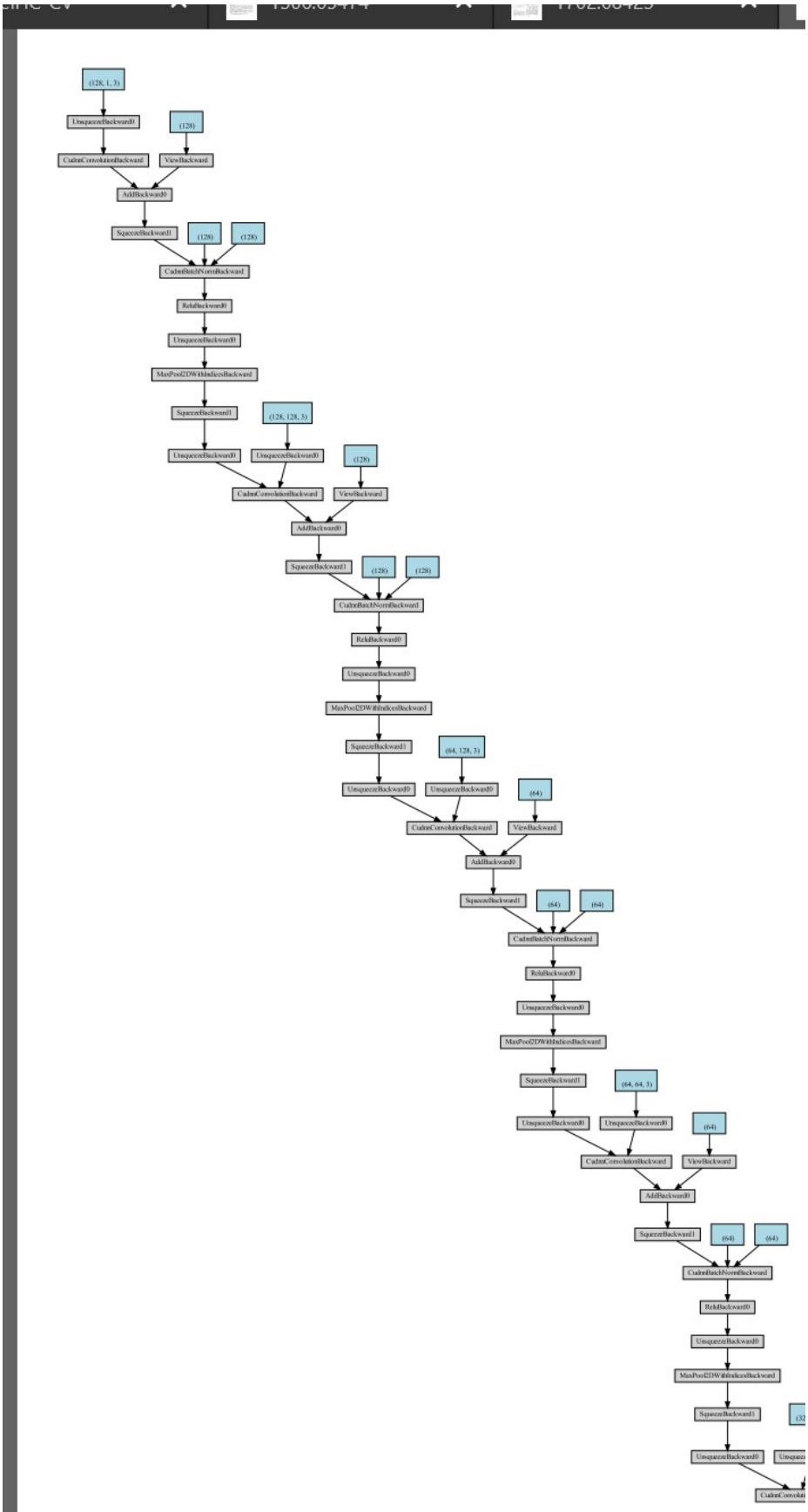


图 4 一维卷积神经网络结构示意图

4. 全部原代码

preprocess.py 提供了预处理相关的函数

```
import numpy as np
from scipy import signal
from scipy import fft
import os


def bw_filter(data, fs=3000, lowcut=40, highcut=260, order=2):
    '''
    对 list 型输入 data 进行 butterworth 滤波，返回滤波后的数据
    '''
    nyq = 0.5*fs  # nyquist frequency of sample
    low = lowcut/nyq
    high = highcut/nyq
    b,a = signal.butter(order, [low,high], btype='bandpass')
    data = signal.lfilter(b, a, data)
    return data


def preprocess(X_noise=0.0):
    '''
    数据加载和预处理，X_noise 为高斯噪声参数，默认为 0（不加噪声）
    '''
    with open("./sc4002e0/sc4002e0_data.txt") as f:
        data = f.readlines()
    data = [float(s) for s in data]
    data = np.array(data)
    # data = bw_filter(data)
    data = data.reshape(2830, 3000)

    with open("./sc4002e0/sc4002e0_label.txt") as f:
        label = f.readlines()
    label = [float(s) for s in label]
    label = np.array(label)[0:2830]

    if not X_noise == 0.0:
        print("adding noise, sigma is {}".format(X_noise))
        noise = np.random.normal(0, X_noise, size=data.shape)
        data = data + noise

    return data, label
```

```
def split_dataset(data, index, percentile=0.8):
    """
    划分数据集，data 为数据集，index 为外部生成的随机序列索引，percentile 为划分比例
    """
    rand_data = []
    for i in range(len(data)):
        rand_data.append(data[index[i]])
    rand_data = np.array(rand_data)
    pi = int(data.shape[0] * percentile)
    stro = "pi=" + str(pi)
    train_set = rand_data[:pi]
    test_set = rand_data[pi:-1]
    return train_set, test_set


def fastft(data):
    """
    根据 FFT 处理特征值
    """
    fjw = np.array([fft(data[i, :]) for i in range(len(data))])
    gjw = np.array([np.abs(fjw[i, :]) for i in range(len(data))])
    gjw = gjw / (gjw.max() - gjw.min())
    phijw = np.array([np.angle(fjw[i, :]) for i in range(len(fjw))])

    return gjw, phijw


if __name__ == "__main__":
    if not os.path.exists(os.path.dirname("./processed")):
        os.mkdir("./processed")
    data, label = preprocess()
    np.savetxt("./processed/data.csv", data, delimiter=",")
    np.savetxt("./processed/label.csv", label, delimiter=",")
```

showsignal.py 提供了信号和其频谱的可视化功能

```
import numpy as np
import matplotlib.pyplot as plt
from preprocess import preprocess, fastft
```

```
def draw(s1, s1f, figdir):
    '''
    对信号 s1 和信号 s1f 进行对比绘图，保存在 figdir 路径
    '''
    fig = plt.figure()

    ax1 = fig.add_subplot(211)
    plt.plot(s1,color='r')
    ax1.set_title('Original Signal')
    plt.ylabel('Amplitude')

    ax2 = fig.add_subplot(212)
    plt.plot(s1f,color='r')
    ax2.set_title('Processed Signal')
    plt.ylabel('Amplitude')

    plt.savefig(figdir)
    plt.close('all')


if __name__ == "__main__":
    data, label = preprocess()
    data_noised, _ = preprocess(X_noise=3)
    f_data, _ = fastft(data)
    f_noised, _ = fastft(data_noised)
    # 画图
    draw(data[0], data_noised[0], "./figures/preprocess_gn_slice.png")
    vis = [898,945,974,1048,1300,1477,1688]
    fig = plt.figure()
    x = []
    for i in range(7):
        x.append(fig.add_subplot(331 + i))
        # freq = np.abs(np.array(fft(train[rep[i], :]), dtype=complex))
        x[i].plot(f_noised[vis[i]])
        title = "Label" + str(i)
        x[i].set_title(title)
    # fig.subplots_adjust(left=0, top=20, right=5, bottom=5, wspace=0.0
1, hspace=0.01)
    fig.savefig("./figures/fft-inclass.png")
```

eegcnn.py 定义了 1D-CNN 的结构。

```
import torch
from torch import nn


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = torch.nn.Sequential(
            nn.Conv1d(1, 128, 3, 2, 1),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            nn.MaxPool1d(2, 2)
        )
        self.conv2 = torch.nn.Sequential(
            nn.Conv1d(128, 128, 3, 2, 1),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            nn.MaxPool1d(2, 2)
        )
        self.conv3 = torch.nn.Sequential(
            nn.Conv1d(128, 64, 3, 2, 1),
            nn.BatchNorm1d(64),
            nn.ReLU(),
            nn.MaxPool1d(2, 2)
        )
        self.conv4 = torch.nn.Sequential(
            nn.Conv1d(64, 64, 3, 2, 1),
            nn.BatchNorm1d(64),
            nn.ReLU(),
            nn.MaxPool1d(2, 2)
        )
        self.conv5 = torch.nn.Sequential(
            nn.Conv1d(64, 32, 3, 2, 1),
            nn.BatchNorm1d(32),
            nn.ReLU(),
            nn.MaxPool1d(2, 2)
        )
        self.conv6 = torch.nn.Sequential(
            nn.Conv1d(32, 32, 3, 2, 1),
            nn.BatchNorm1d(32),
            nn.ReLU(),
            nn.MaxPool1d(2, 2)
        )
```

```
        self.conv7 = torch.nn.Sequential(
            nn.Conv1d(32, 16, 3, 2, 1),
            nn.BatchNorm1d(16),
            nn.ReLU(),
            nn.MaxPool1d(2, 2)
        )
        self.conv8 = torch.nn.Sequential(
            nn.Conv1d(16, 16, 3, 2, 1),
            nn.BatchNorm1d(16),
            nn.ReLU(),
            nn.MaxPool1d(2, 2)
        )
        self.mlp = torch.nn.Linear(3,1)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)
        x = self.mlp(x)
        return x
```

train.py 提供了主要执行的训练和测试部分。

```
import numpy as np
import torch
from torch import nn
from torch.autograd import Variable
from torch.utils.data import DataLoader, TensorDataset
import matplotlib.pyplot as plt
import eegcnn # import my cnn model
from preprocess import preprocess, split_dataset, fastft # import my pr
eprocess funcs
from sklearn.metrics import confusion_matrix, f1_score
import pickle
from torchviz import make_dot
import os
os.environ["PATH"] += os.pathsep + 'D:/Program Files/Graphviz/bin'

if __name__ == "__main__":
    net = eegcnn.Net()  # 加载训练网络
    print(net)  # 输出网络结构
    net = net.cuda()  # 网络使用 GPU
```

```
    # 定义超参数
    batch_size = 16  # 批大小
    learning_rate = 0.02  # 学习率
    num_epoches = 20  # 迭代次数
    percentile = 0.7  # 数据集分割比例
    noise = 1.2  # 高斯噪声
    # 定义优化器、loss
    optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate)  # 定义随机梯度下降优化器
    loss_func = torch.nn.CrossEntropyLoss()  # 定义交叉熵
    losses, acces, eval_losses, eval_acces = [], [], [], []  # 实例化 loss 和 acc 序列
    best_acc = 0
    best_epoch = 0
    # 载入数据集
    data, label = preprocess(X_noise=noise)  # 载入数据集，噪声为设定值
    index = np.arange(len(data))
    np.random.shuffle(index)  # 随机打乱数据 index
    x0, x1 = split_dataset(data, index, percentile=percentile)  # 随机划分数据
    y0, y1 = split_dataset(label, index, percentile=percentile)  # 随机划分标签
    x0 = torch.from_numpy(x0).long().cuda()  # GPU 化
    y0 = torch.from_numpy(y0).long().cuda()
    deal_set = TensorDataset(x0, y0)  # 训练集装载到 tensor
    train_data = DataLoader(
        dataset=deal_set,
        batch_size=batch_size,
        shuffle=True,
        num_workers=0)  # 定义训练集加载器
    # 开始迭代
    net.train()  # 网络调整为训练模式
    for e in range(num_epoches):
        train_loss, train_acc = 0, 0  # 每次迭代清零 loss 和 acc
        for im, label in train_data:  # 分批载入数据和标签
            im = Variable(torch.unsqueeze(im, dim=1).float())  # 数据矩阵降维
            label = Variable(torch.unsqueeze(label, dim=1).long())  # 标签矩阵降维
            out = net(im).cuda()  # 喂数据给网络
            # 迭代
            loss = loss_func(out, label)
            optimizer.zero_grad()
            loss.backward()
```

```
            optimizer.step()
            # 统计结果
            train_loss += loss.item()
            _, pred = out.max(1)  # 得到预测值
            num_correct = (pred == label).sum().item()  # 计算准确个数
            acc = num_correct / im.shape[0]  # 计算准确率
            # acc = f1_score(label.cpu(), pred.cpu(), average="macro")
            train_acc += acc  # 更新总准确率
        # 打印本轮迭代结果
        print("epoch %d finished: loss=%6f,  acc=%6f"
            % (e, train_loss / len(train_data), train_acc / len(train_d
ata)))
        losses.append(train_loss / len(train_data))  # 更新 loss
        acces.append(train_acc / len(train_data))  # 更新 acc
        # 保存当前训练的最佳模型
        if train_acc > best_acc:
            best_acc = train_acc
            best_epoch = e
            if best_acc > 0.95:
                best_model = net
                best_out = out.cpu()
    # 画图
    fig = plt.figure()
    ax = fig.add_subplot(121)
    bx = fig.add_subplot(122)
    ax.set_title('train loss')
    ax.plot(np.arange(len(losses)), losses)
    bx.set_title('train acc')
    bx.plot(np.arange(len(acces)), acces)
    acc_path = ("./figures/Acc_lr="
        + str(learning_rate) + "_bs=" + str(batch_size) + "_percentile=
" + str(percentile)
        + "_epoch" + str(best_epoch) + "_noise=" + str(noise) + ".png")
    plt.savefig(acc_path)
    # 可视化模型
    # 这一步如果报错是因为第 13 行 graphviz 的路径不对，需要安装 graphviz
    g = make_dot(best_out)
    g.render('eegnet_model', view=False)
    # 保存模型
    model_path = ("./pickles/conv1d_lr="
        + str(learning_rate) + "_bs=" + str(batch_size) + "_percentile=
" + str(percentile)
        + "_epoch" + str(best_epoch) + "_noise=" + str(noise))
    with open(model_path, 'wb') as f:
```

```
            pickle.dump(best_model, f)  # 保存训练好的模型
        # 载入测试集
        x1 = torch.from_numpy(x1).float().cuda()  # 载入测试数据 GPU 化
        deal_set = TensorDataset(x1)  # 装载测试集
        test_data = DataLoader(
            dataset=deal_set, batch_size=64,
            shuffle=False, num_workers=0)  # 定义测试集载入器
        # 进行测试
        net.eval()  # 网络调整为测试模式
        res = []  # 预测结果
        for im in test_data:
            im = Variable(torch.unsqueeze(im[0], dim=1))  # 数据矩阵降维
            out = net(im).cuda()  # 预测
            _, pred = out.max(1)  # 得到预测值
            pred.unsqueeze_(1)
            res.append(pred)
        # 整理测试结果
        res = torch.cat(res, dim=0)
        res = res.cpu().numpy().squeeze()  # 整理结果
        num_correct = 0
        for i in range(len(y1)):
            if res[i] == y1[i]:
                num_correct += 1
        acc = num_correct / len(y1)  # 统计准确率
        print(acc, f1_score(y1, res, average="macro"))  # 得到预测结果
accuracy 和 F1-macro 分数
        # 画混淆矩阵
        confusion = confusion_matrix(y1, res)
        figure, ax = plt.subplots()
        plt.imshow(confusion, cmap=plt.cm.Blues)
        # 画色卡
        plt.colorbar()
        # 画坐标轴刻度
        y_tick = ['W', 'S1', 'S2', 'S3', 'S4', 'R', 'M']
        classes = list(y_tick)
        indices = range(len(confusion))
        plt.tick_params(labelsize=11)
        plt.xticks(indices, classes)
        plt.yticks(indices, classes, rotation=90)
        # 画坐标轴标题
        font1 = {
            'family' : 'Times New Roman',
            'weight' : 'normal',
            'size'   : 15,}
```

```
    plt.xlabel('prediction', font1)
    plt.ylabel('real label', font1)
    # 画图的标题
    font2 = {
        'family' : 'Times New Roman',
        'weight' : 'normal',
        'size'   : 23,}
    plt.title('Confusion Matrix', font2)
    # 画混淆矩阵上的文字标注
    # for first_index in range(len(confusion)):
    #     for second_index in range(len(confusion[first_index])):
    #         plt.text(
    #             first_index,
    #             second_index,
    #             confusion[first_index][second_index])
    # 保存
    cfsmat_path = ("./figures/ConfusionMatrix_lr="
        + str(learning_rate) + "_bs=" + str(batch_size) + "_percentile=
" + str(percentile)
        + "_epoch" + str(best_epoch) + "_noise=" + str(noise) + ".png")
    plt.savefig(cfsmat_path)
```

5. 调试分析

运行时可能存在以下问题：

graphviz.backend.ExecutableNotFound: failed to execute ['dot', '-Tpdf', '-O', 'espnet_model'], make sure the Graphviz executables are on your systems' PATH

该问题的原因是未正确配置 graphviz 的路径，解决方法为下载 graphviz 并将其 bin 文件夹路径添加到系统环境变量中。[3]

6. 模型评价

模型在训练集上的训练效果如图 5 所示，测试集上的混淆矩阵如图 6 所示。测试采用 F1-macro 分数和准确率分数共同评价测试结果，其中准确率分数为 0.89，F1 分数为 0.49。造成 F1 分数较低的主要原因是测试集中存在部分类别样本数为 0 的情况，因此在计算召回率时该类别的召回率被认为是 0，从而影响了整体的 F1 分数。通过准确率分数可以得知，该模型可以在很大程度上帮助进行脑电睡

眠波信号的分类，且在较低质量的数据集上也能有较好的表现。

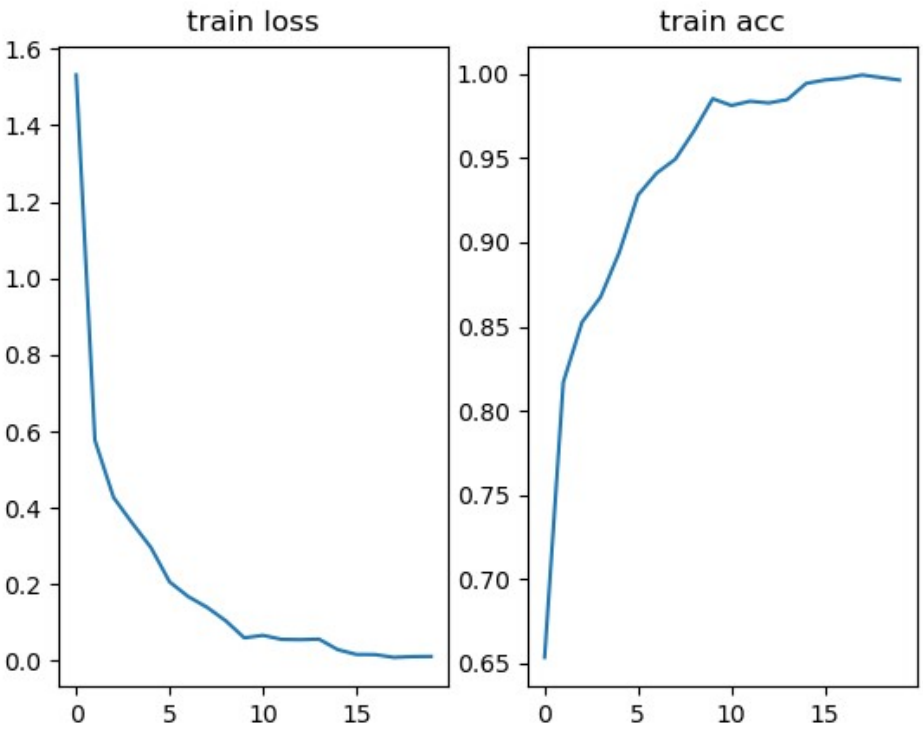


图 5 模型训练的准确率和损失函数

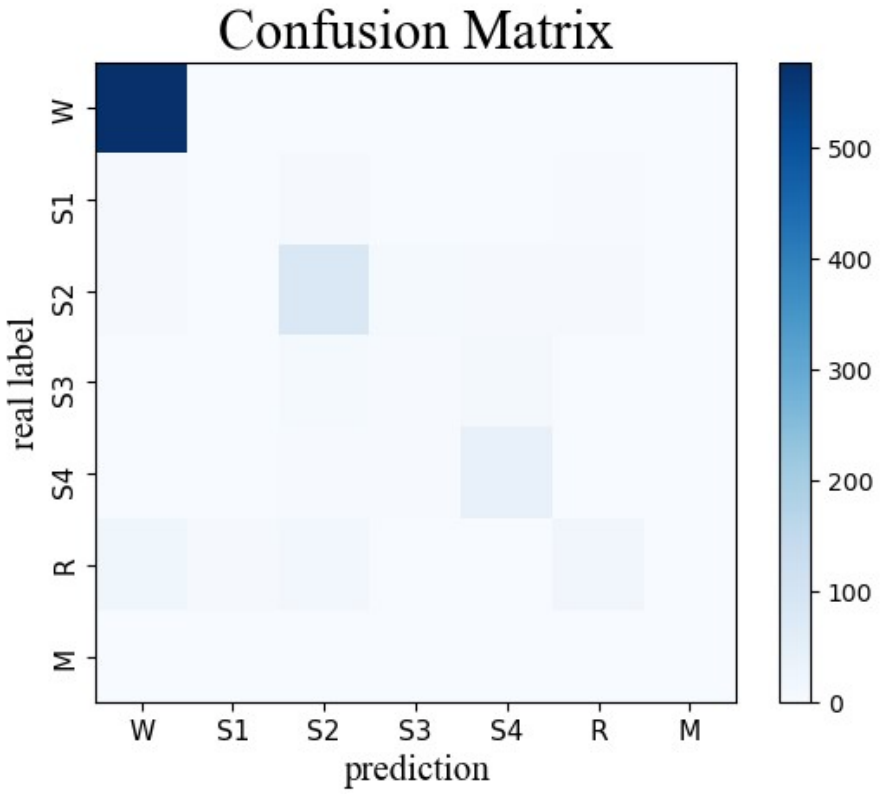


图 6 测试集的混淆矩阵

7. 运行结果截图

```
PS D:\Program Data\Github\repos\eeg-semantic> & C:/Users/ephemera/anaconda3/envs/pytorch-CycleGAN-and-pix2pix/python.exe "d:/Program Data/Github/repos/eeg-semantic/train.py"
Net(
  (conv1): Sequential(
    (0): Conv1d(1, 128, kernel_size=(3,), stride=(2,), padding=(1,))
    (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv1d(128, 128, kernel_size=(3,), stride=(2,), padding=(1,))
    (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv3): Sequential(
    (0): Conv1d(128, 64, kernel_size=(3,), stride=(2,), padding=(1,))
    (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv4): Sequential(
    (0): Conv1d(64, 64, kernel_size=(3,), stride=(2,), padding=(1,))
    (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv5): Sequential(
    (0): Conv1d(64, 32, kernel_size=(3,), stride=(2,), padding=(1,))
    (1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
```

图 7 执行结果：导入网络，打印网络结构

```
adding noise, sigma is 1.2
epoch 0 finished: loss=2.232560,  acc=0.623320
epoch 1 finished: loss=0.870205,  acc=0.781418
epoch 2 finished: loss=0.548842,  acc=0.796875
epoch 3 finished: loss=0.459340,  acc=0.827621
epoch 4 finished: loss=0.366079,  acc=0.862231
epoch 5 finished: loss=0.294338,  acc=0.900370
epoch 6 finished: loss=0.220827,  acc=0.928427
epoch 7 finished: loss=0.195445,  acc=0.936492
epoch 8 finished: loss=0.150922,  acc=0.948757
epoch 9 finished: loss=0.116153,  acc=0.968750
epoch 10 finished: loss=0.106830,  acc=0.968750
epoch 11 finished: loss=0.090719,  acc=0.974126
epoch 12 finished: loss=0.074177,  acc=0.982359
epoch 13 finished: loss=0.067509,  acc=0.983871
epoch 14 finished: loss=0.037966,  acc=0.994792
epoch 15 finished: loss=0.033598,  acc=0.992944
epoch 16 finished: loss=0.023205,  acc=0.996976
epoch 17 finished: loss=0.039819,  acc=0.993448
epoch 18 finished: loss=0.026828,  acc=0.995968
epoch 19 finished: loss=0.023187,  acc=0.995968

0.8869257950530035 0.4921971954540108
```

图 8 执行结果：训练及测试

8. 参考文献

[1] 危昊成,王沛展,张云鑫.模式识别大作业报告-基于一维卷积神经网络的轴承故障检测[R].北京理工大学:北京理工大学,2019.

[2] 裴君楠,陈一萱.一维卷积神经网络在轴承故障检测中的应用[J].现代矿

业,2019,35(10):190-193.
[3]Rosa b..“RuntimeError: Make sure the Graphviz executables are on your system's path” after installing Graphviz
2.38[EB/OL].https://stackoverflow.com/questions/35064304/runtimeerror-make-sure-the-graphviz-executables-are-on-your-systems-path-aft,.