**《统计学习与数据科学》课 程 论 文**

**(2023-2024学年第二学期)**

**基于卷积神经网络的ECG分类**

**学生姓名： 陶俊**

**提交日期： 2024 年 8 月 18 日 学生签名： 陶俊**

|  |  |  |  |
| --- | --- | --- | --- |
| **学 号** | **202066138066** | **座位编号** |  |
| **学 院** | **数学学院** | **专业班级** | **数学与应用数学（统计学）** |
| **课程名称** | **统计学习与数据科学课程设计** | **任课教师** | **夏立** |
| **教师评语：** | | | |
| **本论文成绩评定：** **分** | | | |

**基于卷积神经网络的ECG分类**

**陶俊**

**摘要：** 在数据信息急剧膨胀的今天，数据挖掘与数据分析的能力已经不再是一个专业人员所具备的特有技能，而是应该成为现代人所掌握的基本技能。本课程设计使用 PTB-XL数据集完成了一项多标签心电图分类任务，开发了一个机器学习模型，该模型可以预测心电图记录是否属于这些类别中的一个或多个。通过PTB-XL数据集的训练，可得到其在测试集上的表现良好，在大多数类别上都取得了较高的 AUROC 和 F1-score。

**关键词：ECG，CNN，PTB-XL**

1. **问题概述**

心电图 （ECG） 记录心脏的电活动，通过测量心脏不同位置产生的电信号来反映其功能状态。心电图在临床医学中具有重要意义，对于诊断心脏病、评估心脏功能和监测心脏健康至关重要。随着机器学习和统计学习技术的进步，越来越多的人探索将这些发展应用于心电图的分类和分析，以实现更准确、更及时的心脏病诊断和监测。分析心电图的方法大致可分为传统的机器学习方法和深度学习方法。近年来，深度学习方法在该领域的普及程度激增。卷积神经网络 （CNN） 和递归神经网络 （RNN） 等深度学习技术在自动从 ECG 信号中提取复杂特征方面显示出有希望的结果，从而提高了 ECG 分类任务的准确性。

在本课程项目中，作者使用 PTB-XL 数据集完成一项多标签心电图分类任务，该数据集包含 {“NORM”、“CD”、“HYP”、“MI”、“STTC”} 5 个诊断等级的心电图数据。该任务是一个多标签分类任务，这意味着一个心电记录可以对应多个标签。作者开发了一个机器学习模型，该模型可以预测心电图记录是否属于这些类别中的一个或多个。本文训练的目标即如图1所示：

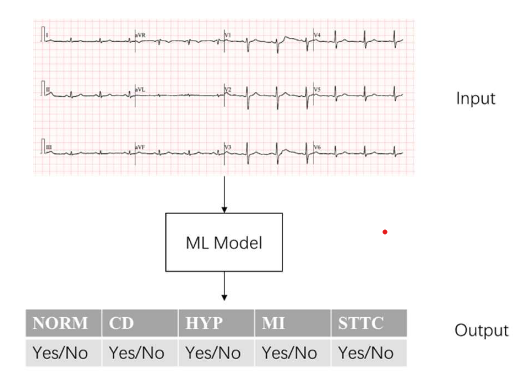


图1.算法预期目标

1. **算法原理分析**

2.1 卷积神经网络（CNN）

传统神经网络对图像的识别分类具有局限性，无法将所处位置不同的同一物体准确识别出来。而卷积神经网络可以有效克服传统神经网络在图像识别的局限性，并在目标检测、图像生成、视频分析、医学成像、自动驾驶、卫星图像处理、语音识别和自然语言处理等领域有着重大优势和广泛应用。卷积神经网络是一类特殊类型的深度前馈型神经网络，它们在处理具有网格结构数据的时候表现出色。卷积神经网络通过局部连接、权重共享和池化等技术，有效地减少了网络的参数数量，同时保留了图像的空间层次结构。卷积神经网络的基本结构由输入层、卷积层、池化层、全连接层及输出层构成，卷积层和池化层一般会取若干个，采用卷积层和池化层交替设置。卷积层中输出特征面的每个神经元与其输入进行局部连接，并通过对应的连接权值与局部输入进行加权求和再加上偏置值，得到神经元的输入值，该过程为卷积。卷积神经网络通常由以下几个部分组成：

1）输入层：

输入层接收原始图像的数据。图像通常由红、绿、蓝三个颜色通道的数据组成，每个通道的数据由一个矩阵表示，矩阵的每一个元素对应一个像素，代表颜色的深浅，三个矩阵排列起来构成一个张量。

2）卷积层：

卷积神经网络的核心层，用于执行卷积操作，其主要特点是局部连接和参数共享。卷积操作通过滑动一个小的窗口（卷积核、滤波器）在输入图像上，并计算窗口内像素与卷积核的乘积之和，加上偏置值，然后将结果输出为新的特征图。不同的卷积核可以提取输入图像的不同特征，如边缘、角点、纹理等。

输入层前向传播到卷积层是卷积神经网络前向传播算法的第一步，其过程为：

其中，输入层是个矩阵，矩阵的值是图片各像素的值。如果样本为RGB彩色图片，则输入层为三个矩阵；若样本为灰度图片，则输入层为一个矩阵。卷积核也是个矩阵，一般为方阵，且卷积层的深度必须与输入图像的深度相同。公式上标表示层数，星号表示卷积，表示偏置，𝜎为激活函数。公式（1）表示输入层为一个矩阵，卷积核深度为32，有32个偏置，激活函数采用函数。

3）池化层：

模拟人眼对图像的模糊处理。池化层也称为下采样层，用于降低特征图的维度，提取主要特征，这有助于减少计算量和参数数量，控制过拟合，并使特征具有一定的平移不变性。常见的池化操作包括最大池化和平均池化：平均池化能够提取出图像局部特征区域的整体特征，提炼局部特征区域的背景；最大池化可以提取出图像的边缘纹路，放大不同特征区域的内容差异，但可能会引起图像的过度失真。

4）全连接层：

在卷积神经网络的末端，通常有一个或多个全连接层，用于整合前面各层提取的特征，并输出最终的结果。在全连接层中，每个神经元都与前一层的所有神经元相连。

5）输出层：

根据任务的不同，输出层可以设计为分类任务中的层，用于输出类别的概率分布，或回归任务中的线性层，用于直接输出预测值。

下面说明卷积神经网络的前向传播和反向传播的算法思想。前向传播是指神经元的输出是上一层神经元输出的加权和，加上一个偏置项，然后通过激活函数进行非线性变换传递到下一层中的过程。 步骤如下：

1. 若第层是卷积层，则输出为：

其中为输入的张量，为卷积核的参数，为偏置，为状态值，作为激活函数的一个输入。

1. 若第层是池化层，则输出为：

这里的是指按池化区域的大小和池化标准将输入张量缩小的过程。

1. 若第层是全连接层，则输出为：

其中为输入的张量，为每个神经元的连接参数，为偏置，为激活函数。

1. 若第层是输出层，则输出为：

其中为输入的张量，为每个神经元的连接参数，为偏置，为激活函数。

卷积神经网络的反向传播过程是基于链式法则，计算损失函数对模型参数的梯度，从而更新权重和偏差，减少预测误差的过程。主要步骤如下：

1）若当前为全连接层：

其中为当前第层的梯度误差，表示点乘运算，是神经元的权重参数，是激活函数的导数，为当前第层的状态值。

更新权重为：

更新偏置为：

其中为学习率，为第-1层的输出张量，为第层的梯度误差。

1. 若当前为卷积层：

其中表示卷积核被旋转了度。

更新权重为：

更新偏置为：

1. 若当前是池化层：

其中指在反向传播时把的所有子矩阵矩阵大小还原成池化之前大小的过程。如果是最大池化，则把所有子矩阵的各个池化局域的值放在之前做前向传播算法得到最大值的位置。如果是平均池化，则把所有子矩阵的池化局域的值取平均值后放在还原后的子矩阵位置。完成了池化误差矩阵放大与误差重新分配的逻辑。

1. **问题的求解**

**3.1数据预处理**

题目要求使用PTB-XL心电图数据集。PTB-XL ECG数据集是一个大型数据集，包含10秒长的18869名患者的21799个临床12导心电图。原始波形数据由多达两名心脏病专家注释，他们将潜在的多个心电图语句分配给每个记录。总共有71个不同的心电图语句符合SCP-ECG标准，涵盖了诊断、形式和节奏语句。这个数据集其中 52% 为男性，48% 为女性，年龄范围从 0 到 95 岁（中位数 62 岁，分位数范围为 22）。

预处理过程主要是以下几个部分：首先，我们需要处理特征数据，即“X”：患者的 ECG 数据。我们需要从 DAT 文件中读取数据并将其存储在张量中。这是一个相对复杂的过程。为了完成此任务，我们定义了一个名为 PTBXLDataset 的类，并将此过程封装在 PTBXLDataset 类的初始化函数中，将文件夹地址作为参数。在初始化函数中，我们处理文件夹地址。首先，我们定位地址指定的文件夹中的“ptbxl\_database.csv”文件。该文件作为目录文件，其中“filename\_hr”和“filename\_lr”列包含特定数据的文件名，采样率分别为 500 和 100。我们从这两个列中提取元素，并将文件夹地址添加一个级别，以获得特定文件地址。通过使用 for 循环遍历 filename 列中的所有元素，我们可以使用 wfdb 包中的 rdsamp 方法读取所有数据；其次，为了处理标签，我们做的过程与特征数据类似，但添加了两个步骤将其转换为one hot标签。读取标签数据并将其存储到列表（这是一个多标签列表，因为一个病人可能有多个标签）之后，我们需要将该列表转换为可以在pytorch中处理的一个热标签。它包含两个步骤:首先，使用Label mapping列表，我们可以将每个标签编码为表示标签的数字；接着，质量检验过程包括缺失值分析和异常值分析。通过对数据集的检查，我们没有发现缺失值，并且根据PTBXL数据集的描述，数据采集过程是在严格的管理和检查下进行的，所以我们可以假设特征数据中没有缺失值和异常值。但对于标签来说，由于多标签样品，可能存在某种反逻辑问题。我们假设样本没有逻辑错误，并平等地处理它们。而对于特征提取过程，我们跳过这个过程，因为深度学习模型的能力可以让我们在不需要人工特征提取的情况下进行端到端的学习。我们只需将序列数据输入到模型中，卷积层就可以自动完成特征提取过程；最后，该项目使用“strat\_fold”的值对训练和测试数据集进行拆分，对于每个样本，如果“strat\_fold“的值等于“test\_fold”，则它属于测试数据集。否则它属于训练数据集。此时，训练集与测试集样本比例大约在9:1。

**3.2模型的建立**

本课程设计建立的模型是5层经典的卷积神经网络。该模型包含5层和1个fc层。每层包含一个批量归一化层、一个dropout层（dropout rate=0.2）和一个Avg pooling层。对于第一层，它有12个输入通道和16个输出通道，最后4层有越来越多的输出通道，以增强模型的高级特征提取能力。该模型具有更浅的结构和更小的尺寸，实现了更高的准确性，因为池化层有效地减小了模型的大小，并且这些过程增加了模型的泛化能力和防止模型过度拟合。基本结构如下：

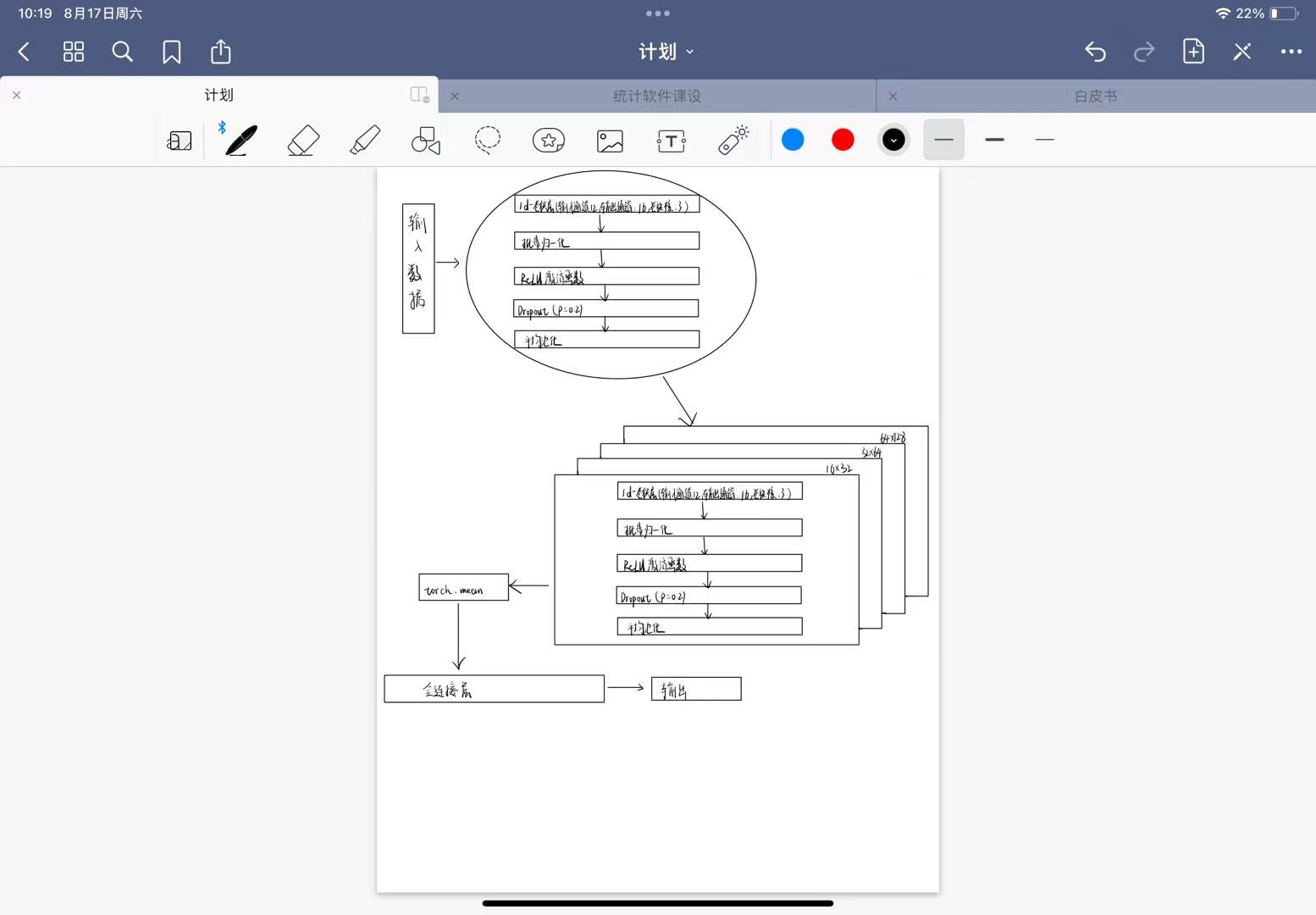


图2.6层卷积神经网络的基本架构

**3.3结果展示与分析**

代码跑出的结果如下：

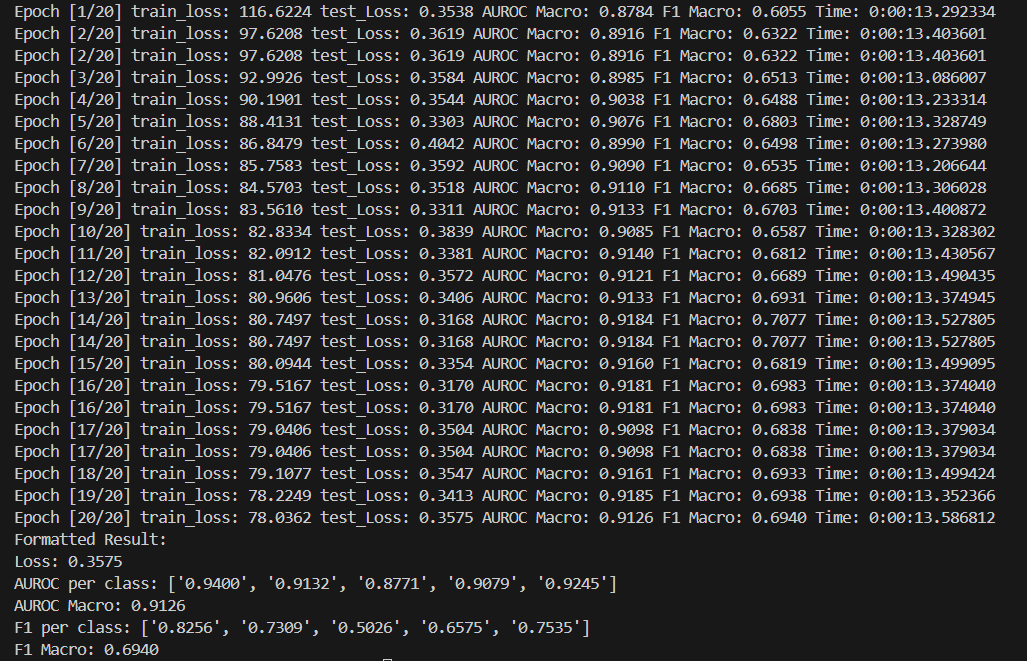


图3.运行结果

这里需要对Formatted Result中几个参数进行说明：

（1）Loss：MultiLabelSoftMarginLoss是PyTorch中用于多标签分类任务的损失函数。它适用于每个样本可能属于多个类的情况。该损失函数将每个样本的每个类视为一个独立的二进制分类问题，并计算每个类别的二进制交叉熵损失。对于每个样本， MultiLabelSoftMarginLoss 计算预测概率与真实标签之间的交叉熵损失。且对于每个类别，损失函数平均所有样本的交叉熵损失，以获得该类别的损失。最后，再次对所有类别的损失进行平均，得到最终损失值；

（2）AUROC是一种常用的二分类评价指标，表示 ROC 曲线下面积。ROC 曲线展示了分类模型的真正率（True Positive Rate, TPR）与假正率（False Positive Rate, FPR）之间的关系，AUROC 值越接近 1，说明模型的分类性能越好。**AUROC per class** 是指针对多分类问题中每个类别分别计算的 AUROC（ROC 曲线下面积）。对于多分类问题，可以将每个类别单独视为一个二分类问题（该类别为“正类”，其他类别为“负类”），然后计算该类别的 AUROC；Macro AUROC 是在多分类任务中使用的一种方法，主要通过以下方式计算：针对多分类问题的每个类别，将该类别视为“正类”，其余类别视为“负类”，然后计算出每个类别的 AUROC(其值越接近 1，模型的区分能力越强。0.5 表示模型没有分类能力，相当于随机猜测。低于 0.5 表示模型在反向分类，即把负样本预测为正样本，把正样本预测为负样本)；取这些 AUROC 的平均值：将每个类别的 AUROC 取平均值，得到宏观意义上的 AUROC，即 Macro AUROC；

（3）F1 per class 和 F1 Macro 是用于评估多分类模型性能的两种方式，尤其是在类别不平衡情况下非常有用。它们都基于 F1-score，这是精确率（Precision）和召回率（Recall）的调和平均数。F1 per class 是指针对每个类别分别计算的 F1-score。具体计算步骤如下：针对每个类别计算 Precision 和 Recall（其中Precision（精确率）：正确预测为该类别的样本数量占所有被预测为该类别的样本总数的比例，Recall（召回率）：正确预测为该类别的样本数量占该类别的真实样本总数的比例）；计算 F1-score：

F1-score 是精确率（Precision）和召回率（Recall）的调和平均数，因此越接近 1，模型的表现越好。F1-score 为 0 表示模型预测完全错误，为 1 表示预测完全正确。

F1 Macro 是对所有类别的 F1 per class 的平均值，具体步骤如下：计算每个类别的 F1-score：按照 F1 per class 的方式计算;取平均值：将所有类别的 F1-score 相加，再除以类别数，得到宏观 F1-score，即 F1 Macro。F1 Macro 是所有类别 F1-score 的平均值，越接近 1，表示模型在所有类别上的整体表现越好。与 F1 per class 类似，接近 1 表示模型的表现较优。F1 Macro 是评估模型整体性能的一个简便指标，而 F1 per class 则提供了更细粒度的模型分析。

实验结果表明，模型在大多数类别上都取得了较高的 AUROC 和 F1-score，证明了其有效性。然而，模型在某些类别上的表现还有改进空间，特别是在数据较为稀少的类别上。针对这些问题，我计划在后续的工作中进一步优化模型结构和训练策略。

**3.4模型的评价**

**3.4.1 模型优点**

1）卷积神经网络具有强大的特征学习能力，能够自动学习图像中的特征，‌无需手动设计特征提取器，‌大大简化了特征工程的过程，‌提高了模型的泛化性能。‌

2）卷积神经网络‌通过分块操作，能够捕捉到图像中的空间信息，‌如位置、‌方向等，‌有助于区分不同类别的图像。‌

3）‌卷积神经网络的参数可以根据任务需求进行调整和扩展，‌以适应不同规模和复杂度的数据集。‌

**3.4.2 模型缺点**

1）模型迭代次数较少，可能权重正则化对模型参数的调整不明显。

2）训练卷积神经网络需要大量的计算资源和时间，‌需要高性能的硬件设备支持。‌

3）卷积神经网络的结构选择和参数调优是一项复杂的工作，‌需要大量的经验和实验。‌不同参数的选择和调整对模型性能有很大影响。

1. **总结**

本课程项目展示了如何使用 PTB-XL 数据集进行多标签心电图分类任务。通过这一项目，我不仅掌握了多标签分类任务的基本方法，还积累了处理医学信号数据的经验。这些经验对未来从事相关领域的研究和应用具有重要意义。

其中遭遇的最大的一个难题是Pytorch安装好几次，都无法在vscode中导入。经过一系列的环境配置和Pytorch的多次下载，还是无法解决问题。经过请教同学以及一系列的尝试，我们才发现可能是因为下载的Pytorch是最新版本，自身存在了bug。后面我下载了2.2.0版本，就完美地解决了问题。

另外，这次的课设我做的整体还是显得较为粗糙。究其原因，还是自己投入的时间不够，再加之，自己的代码能力以及对机器学习的相关领域了解的非常浅显。大三升大四这个暑假确实较为繁忙，首先是参加各大高效的夏令营一直到7月底，接着八月初开始准备两个从零开始的课程设计，与此同时还准备着开学的缓考科目和预推免相关的考试资料。但好在有着同学，学长学姐和老师们的帮助，让我能够艰难地完成此次课程设计，真的让我受益匪浅。在今后，还是得加强相关能力的训练。

附录：

import torch

import numpy as np

from scipy import io

import pandas as pd

from torch.utils.data import Dataset, DataLoader

from sklearn import preprocessing

from sklearn.preprocessing import MultiLabelBinarizer

import wfdb

import ast

import matplotlib.pyplot as plt

class PTBXLDataset(Dataset):

    def \_\_init\_\_(self, path,sampling\_rate = 500, train\_set =True, transform=None):

        def load\_raw\_data(df, sampling\_rate, path):

            if sampling\_rate == 100:

                data = [wfdb.rdsamp(path+'/'+f) for f in df.filename\_lr]

            else:

                data = [wfdb.rdsamp(path+'/'+f) for f in df.filename\_hr]

            data = np.array([signal for signal, meta in data])

            return data

        Y = pd.read\_csv(path+'\ptbxl\_database.csv', index\_col='ecg\_id')

        Y.scp\_codes = Y.scp\_codes.apply(lambda x: ast.literal\_eval(x))

        # 导入原始数据

        X = load\_raw\_data(Y, sampling\_rate, path)

        agg\_df = pd.read\_csv(path+'\scp\_statements.csv', index\_col=0)

        agg\_df = agg\_df[agg\_df.diagnostic == 1]

        def aggregate\_diagnostic(y\_dic):

            tmp = []

            for key in y\_dic.keys():

                if key in agg\_df.index:

                    tmp.append(agg\_df.loc[key].diagnostic\_class)

            return list(set(tmp))

        # 应用诊断超类

        Y['diagnostic\_superclass'] = Y.scp\_codes.apply(aggregate\_diagnostic)

        # 拆分成训练集与测试集

        test\_fold = 10

        # 训练

        X\_train = X[np.where(Y.strat\_fold != test\_fold)]

        y\_train = Y[(Y.strat\_fold != test\_fold)].diagnostic\_superclass

        # 测试

        X\_test = X[np.where(Y.strat\_fold == test\_fold)]

        y\_test = Y[Y.strat\_fold == test\_fold].diagnostic\_superclass

        if train\_set:

            self.data = np.array(X\_train)

            labels = list(y\_train)

        else:

            self.data = np.array(X\_test)

            labels = list(y\_test)

        label\_mapping = {"NORM": 0, "CD": 1, "HYP": 2, "MI": 3, "STTC": 4}

        self.label = [[label\_mapping[label] for label in sample] for sample in labels]

        mlb = MultiLabelBinarizer()

        self.onehot\_label = mlb.fit\_transform(self.label)

        self.onehot\_label = torch.tensor(self.onehot\_label, dtype=torch.float32)

        self.data = torch.tensor(self.data, dtype=torch.float32).permute(0,2,1)

        self.data = torch.nan\_to\_num(self.data, nan=0.0)

        self.X=X

    def \_\_len\_\_(self):

        return len(self.data)

    def \_\_getitem\_\_(self, idx):

        data = self.data[idx]

        onehot\_label = self.onehot\_label[idx]

        return data,onehot\_label

#读取数据，

path = r"c:\Users\TJ\Desktop\统计软件课程代码\ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.3"

sampling\_rate = 100

train\_dataset = PTBXLDataset(path,sampling\_rate=100,train\_set=True)

test\_dataset = PTBXLDataset(path,sampling\_rate=100,train\_set=False)

fine\_tune\_loader = DataLoader(train\_dataset, batch\_size=64, shuffle=True)

test\_loader = DataLoader(test\_dataset, batch\_size=64, shuffle=False)

train\_dataset.X

train\_dataset.label[0:10]

train\_dataset.onehot\_label[0:10]

train\_dataset.data.shape

test\_dataset.data.shape

import torch.nn as nn

# 五层经典卷积网络

class CNN\_1D\_5L(nn.Module):

    def \_\_init\_\_(self, n\_in):

        super().\_\_init\_\_()

        self.n\_in = n\_in

        self.layer1 = nn.Sequential(

            nn.Conv1d(12, 16, (3,), stride=1, padding=4),

            nn.BatchNorm1d(16),

            nn.ReLU(),

            nn.Dropout(p=0.2),

            nn.AvgPool1d(2,stride=2)

        )

        self.layer2 = nn.Sequential(

            nn.Conv1d(16, 16, (5,), stride=1, padding=2),

            nn.BatchNorm1d(16),

            nn.ReLU(),

            nn.Dropout(p=0.2),

            nn.AvgPool1d(2,stride=2)

        )

        self.layer3 = nn.Sequential(

            nn.Conv1d(16, 32, (5,), stride=1, padding=2),

            nn.BatchNorm1d(32),

            nn.ReLU(),

            nn.Dropout(p=0.2),

            nn.AvgPool1d(2,stride=2)

        )

        self.layer4 = nn.Sequential(

            nn.Conv1d(32, 64, (5,), stride=1, padding=2),

            nn.BatchNorm1d(64),

            nn.ReLU(),

            nn.Dropout(p=0.2),

            nn.AvgPool1d(2,stride=2)

        )

        self.layer5 = nn.Sequential(

            nn.Conv1d(64, 128, (5,), stride=1, padding=2),

            nn.BatchNorm1d(128),

            nn.ReLU(),

            nn.Dropout(p=0.2),

            nn.AvgPool1d(2,stride=2)

        )

        self.linear1 = nn.Linear(128, 5)

    def forward(self, x):

        x = x.view(-1, 12, self.n\_in)

        #print('===debug===',x.shape)

        x = self.layer1(x)

        #print('===debug===',x.shape)

        x = self.layer2(x)

        #print('===debug===',x.shape)

        x = self.layer3(x)

        #print('===debug===',x.shape)

        x = self.layer4(x)

        #print('===debug===',x.shape)

        x = self.layer5(x)

        #print('===debug===',x.shape)

        ##x = x.view(-1, 128\*31)

        x=torch.mean(x,dim=-1)

        return self.linear1(x)

#评价函数和训练函数

from torch import nn as nn

import torch

from sklearn.metrics import confusion\_matrix, f1\_score, accuracy\_score

from sklearn.metrics import roc\_auc\_score

import datetime

import torch.nn.functional as F

import torch

#评价函数

def evaluate\_multilabel\_classification(model, dataloader, device, criterion):

    model.eval()

    all\_labels = []

    all\_predictions = []

    total\_loss = 0.0

    total\_samples = 0

    with torch.no\_grad():

        for inputs, labels in dataloader:

            inputs, labels = inputs.to(device), labels.to(device)

            outputs = model(inputs)

            predictions = torch.sigmoid(outputs)

            loss = criterion(outputs, labels.float())

            total\_loss += loss.item() \* inputs.size(0)

            total\_samples += inputs.size(0)

            all\_labels.append(labels.cpu().numpy())

            all\_predictions.append(predictions.cpu().numpy())

    average\_loss = total\_loss / total\_samples

    all\_labels = np.vstack(all\_labels)

    all\_predictions = np.vstack(all\_predictions)

    # Calculate AUROC for each label

    auroc\_list = []

    for i in range(all\_labels.shape[1]):

        auroc = roc\_auc\_score(all\_labels[:, i], all\_predictions[:, i])

        auroc\_list.append(auroc)

    # Calculate average AUROC

    auroc\_macro = np.mean(auroc\_list)

    # Calculate F1 score for each label

    f1\_list = []

    for i in range(all\_labels.shape[1]):

        f1 = f1\_score(all\_labels[:, i], (all\_predictions[:, i] > 0.5).astype(int))

        f1\_list.append(f1)

    # Calculate average F1 score

    f1\_macro = np.mean(f1\_list)

    return {"Loss": average\_loss, "AUROC per class": auroc\_list, "AUROC Macro": auroc\_macro, "F1 per class": f1\_list, "F1 Macro": f1\_macro}

#训练函数

def train\_model\_ptbxl(train\_loader, test\_loader, model, criterion, optimizer, num\_epochs, device='cpu',evaluate=True):

    model.to(device)

    train\_loss = []  # List to store the loss values

    test\_loss=[]

    macro\_auc=[]

    f1\_auc=[]

    for epoch in range(num\_epochs):

        model.train()

        total\_loss = 0.0

        start\_time = datetime.datetime.now()

        for inputs, labels in train\_loader:

            optimizer.zero\_grad()

            outputs = model(inputs.to(device))

            loss = criterion(outputs, labels.to(device))

            loss.backward()

            optimizer.step()

            total\_loss += loss.item()

        end\_time = datetime.datetime.now()

        if evaluate:#if evaluate and epoch % 5 == 4:

            model.eval()

            result = evaluate\_multilabel\_classification(model, test\_loader, device, criterion)

            formatted\_result = {

                "Loss": "{:.4f}".format(result["Loss"]),

                "AUROC per class": ["{:.4f}".format(auroc) for auroc in result["AUROC per class"]],

                "AUROC Macro": "{:.4f}".format(result["AUROC Macro"]),

                "F1 per class": ["{:.4f}".format(f1) for f1 in result["F1 per class"]],

                "F1 Macro": "{:.4f}".format(result["F1 Macro"])

            }

            train\_loss.append("{:.4f}".format(total\_loss))

            test\_loss.append(formatted\_result["Loss"])

            macro\_auc.append(formatted\_result["AUROC Macro"])

            f1\_auc.append(formatted\_result["F1 Macro"])

            print(f"Epoch [{epoch + 1}/{num\_epochs}]","train\_loss:", "{:.4f}".format(total\_loss),"test\_Loss:", formatted\_result["Loss"],

                "AUROC Macro:", formatted\_result["AUROC Macro"],

                "F1 Macro:", formatted\_result["F1 Macro"], "Time:", end\_time - start\_time)

    return train\_loss,test\_loss,macro\_auc,f1\_auc

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

device

#加载模型以及训练参数

model = CNN\_1D\_5L(1000)

model.to(device)

lr = 0.001

optimizer = torch.optim.AdamW(model.parameters(), lr = lr,weight\_decay = 1e-2)

criterion = nn.MultiLabelSoftMarginLoss()

epochs = 20

#训练和评价模型

train\_loss,test\_loss,macro\_auc,f1\_auc=train\_model\_ptbxl(fine\_tune\_loader, test\_loader, model, criterion, optimizer,epochs, device=device)

model.eval()

torch.save(model.state\_dict(), 'model\_weights\_ptbxl.pth')

result = evaluate\_multilabel\_classification(model, test\_loader, device, criterion)

formatted\_result = {

    "Loss": "{:.4f}".format(result["Loss"]),

    "AUROC per class": ["{:.4f}".format(auroc) for auroc in result["AUROC per class"]],

    "AUROC Macro": "{:.4f}".format(result["AUROC Macro"]),

    "F1 per class": ["{:.4f}".format(f1) for f1 in result["F1 per class"]],

    "F1 Macro": "{:.4f}".format(result["F1 Macro"])

}

# 输出结果

print("Formatted Result:")

print("Loss:", formatted\_result["Loss"])

print("AUROC per class:", formatted\_result["AUROC per class"])

print("AUROC Macro:", formatted\_result["AUROC Macro"])

print("F1 per class:", formatted\_result["F1 per class"])

print("F1 Macro:", formatted\_result["F1 Macro"])