復旦大學
FUDAN UNIVERSITY

大数据中的案例实务

期末项目报告：语种识别模型

学院 大数据学院

专业 应用统计

学号 23210980102

姓名 周宇轩

2023 年 12 月 20 日

目录

1 项目概览

本项目实现了一个基于梅尔频谱语音数据集的中英文语音分类模型，完整包括了数据读取和处理、模型构建、模型训练和模型测试的全流程。

本项目的文件结构如下：

```
./
|   main.py
|   test.py
|   test.csv
|
└────src
       dataloader.py
       model.py
       train.py
```

图 1: 项目文件树

如图1所示，在项目目录中，/src是各模块的代码文件存放目录，其中dataloader.py包含了数据加载、数据处理和数据集分割方法，model.py包含了定义模型的类，train.py包含了训练模型的方法；main.py是整合各模块进行数据读取和处理、模型构建和模型训练的代码；test.py是读取本地保存的模型文件进行测试集预测的代码；test.csv是模型在测试集上的结果。

2 问题分析

2.1 数据集分析

在读取数据集后，观察其中若干数据，可知这些梅尔频谱语音数据的维度为 (x, y)，其中 x 为时间维度，y 为特征维度（频率为度），不同数据的时间维度是不一致的（图2，但是特征维度均为 80。

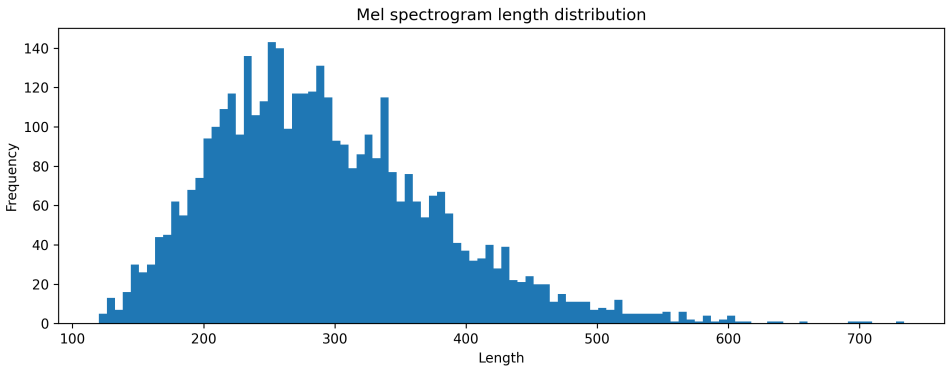

图 2: 训练集数据时间维度分布图

在了解了数据集整体的情况后，选择其中一条数据，并尝试绘制出一个梅尔频谱图（图3），可知梅尔频谱实际上可以用图像表示，这为我们后面的模型构建奠定了基础。

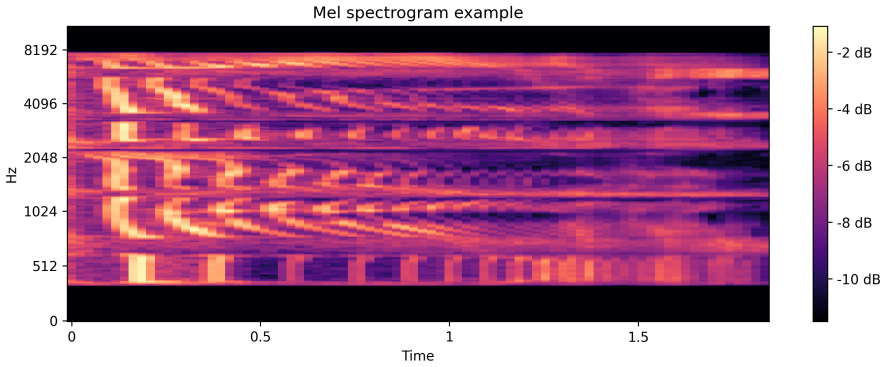

图 3: 梅尔频谱图像可视化

2.2 模型构建分析

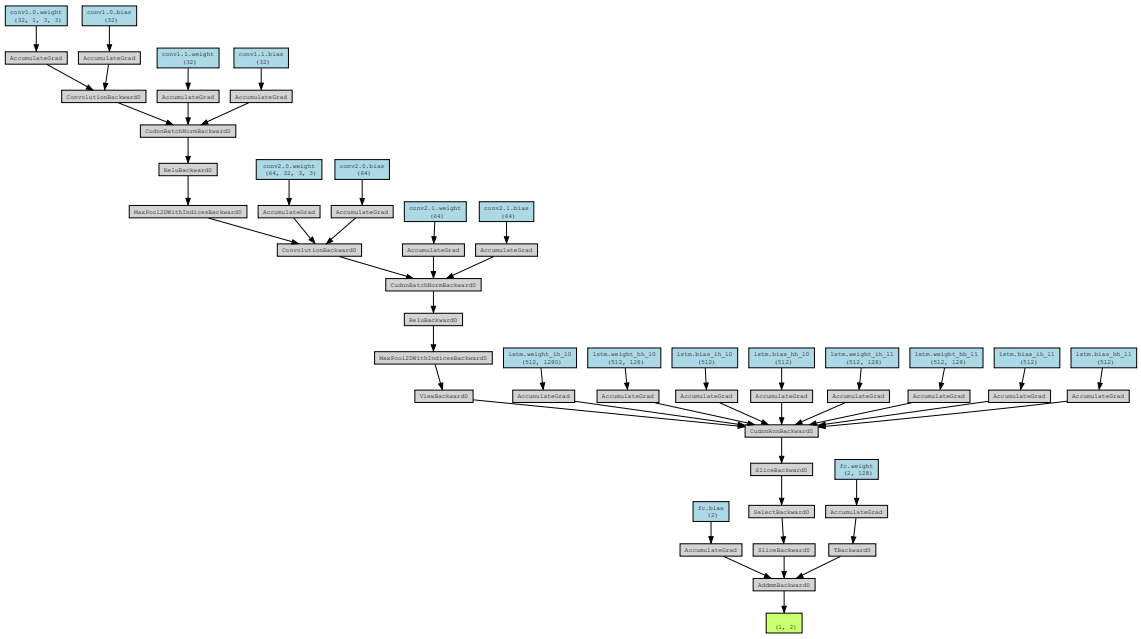

图 4: 模型结构图

由前面对数据集的分析可知梅尔频谱实际上可以绘制成图片，也就是说可以把梅尔频谱数据作为图像数据处理，我们可以将对图像分类的思想应用于梅尔频谱语音数据语种分类任务上。

卷积神经网络（CNN）普遍应用于图像分类任务上，它具有很强的特征提取能力，

而中英文语音的梅尔频谱图像势必有各自的显著特征，将卷积神经网络应用于语音语种分类任务是合理且有效的。

而语音数据相较于传统的图像数据的不同点在于它是一种序列数据，而循环神经网络（RNN）在处理序列数据上有显著的优势，它能够学习和记住长期的时间依赖性，这对于理解语音非常重要。

因此，本项目结合了卷积神经网络和循环神经网络，构建了包含两个卷积块、一个 LSTM 层和一个全连接层的神经网络模型。

图4是我使用 torchviz 库生成的一张模型结构图。

3 模型训练与预测

3.1 数据读取和处理

首先，需要读取训练集，并处理成模型可以读取的格式。

上文提及，不同数据的时间维度是不一致的，因此我们需要统一数据的维度，否则模型将无法处理不同长度的序列输入。这里采取的处理方法是，设定一个统一的目标值，时间维度小于目标值的数据，通过填充（padding）空数据（这里体现为在数据结尾补 0 向量）来延长时间维度；时间维度大于目标值的数据，通过截断（truncate）目标值之后的数据来缩短时间维度。

项目中设定的目标值为 800，因此统一处理后的梅尔频谱数据的维度均为 (800, 80)。

为了了解模型训练过程中的性能变化，我们需要对数据集做分割，分为训练集和验证集。我们先将中英文的数据全部混合并打乱，再从中抽取一定比例的验证集（本项目中的抽取比例设定为 0.2）。

3.2 模型训练

```
yxzhou
def __init__(self):
    # 继承父类的初始化方法
    super(LanguageIdentificationModel, self).__init__()

    # 定义神经网络结构
    # 第一个卷积层，输入为(1, 800, 80)，输出为(32, 400, 40)
    self.conv1 = nn.Sequential(
        nn.Conv2d(in_channels=1, out_channels=32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
        nn.BatchNorm2d(32),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2))
    )
    # 第二个卷积层，输入为(32, 400, 40)，输出为(64, 200, 20)
    self.conv2 = nn.Sequential(
        nn.Conv2d(in_channels=32, out_channels=64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
        nn.BatchNorm2d(64),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2))
    )
    # LSTM层，输入为(200, 64 * 20)，输出为(200, 128)
    self.lstm = nn.LSTM(input_size=64 * 20, hidden_size=128, num_layers=2, batch_first=True)
    # 全连接层，输入为(200, 128)，输出为(2)
    self.fc = nn.Linear( in_features: 128,  out_features: 2)
```

图 5: 神经网络结构定义代码

本项目基于 PyTorch 框架构建神经网络模型，模型的结构细节不在此过多赘述，仅展示定义模型结构部分的代码（图5）。

本项目采取了小批量多轮次的训练方法，定义交叉熵损失作为损失函数，优化器使用的是 Adam 算法，并在训练开始前设定了批量大小（batch_size）、训练轮次（epochs）和学习率（learning_rate）等参数。

用于训练的设备为一张 NVIDIA GeForce RTX 4060。

模型训练过程中，每训练一轮次，控制台输出一行信息，包含当前轮次模型分别在训练集和验证集上的 loss、分别在训练集和验证集上的 accuracy 和该轮训练耗费的时长。我选取了某一次训练过程的控制台输出作为样例（图6），该样例不代表最终提交结果所使用的模型的效果。

```
Epoch: 1        Training Loss: 0.004725         Validation Loss: 0.005356       Training Acc: 0.666562  Validation Acc: 0.666250        Time: 13.814997s
Epoch: 2        Training Loss: 0.002384         Validation Loss: 0.002691       Training Acc: 0.883125  Validation Acc: 0.896250        Time: 12.527012s
Epoch: 3        Training Loss: 0.001548         Validation Loss: 0.001994       Training Acc: 0.927500  Validation Acc: 0.926250        Time: 12.599627s
Epoch: 4        Training Loss: 0.001427         Validation Loss: 0.001919       Training Acc: 0.931875  Validation Acc: 0.925000        Time: 12.546350s
Epoch: 5        Training Loss: 0.001391         Validation Loss: 0.001977       Training Acc: 0.933750  Validation Acc: 0.912500        Time: 11.667345s
Epoch: 6        Training Loss: 0.001322         Validation Loss: 0.002000       Training Acc: 0.932812  Validation Acc: 0.918750        Time: 11.698172s
Epoch: 7        Training Loss: 0.001523         Validation Loss: 0.002493       Training Acc: 0.920000  Validation Acc: 0.887500        Time: 11.857666s
Epoch: 8        Training Loss: 0.000898         Validation Loss: 0.001867       Training Acc: 0.962187  Validation Acc: 0.933750        Time: 11.705188s
Epoch: 9        Training Loss: 0.001137         Validation Loss: 0.002024       Training Acc: 0.936250  Validation Acc: 0.902500        Time: 11.666724s
Epoch: 10       Training Loss: 0.000763         Validation Loss: 0.002213       Training Acc: 0.961250  Validation Acc: 0.920000        Time: 11.808676s
Epoch: 11       Training Loss: 0.000427         Validation Loss: 0.001822       Training Acc: 0.984062  Validation Acc: 0.933750        Time: 11.737773s
Epoch: 12       Training Loss: 0.000260         Validation Loss: 0.002430       Training Acc: 0.989375  Validation Acc: 0.927500        Time: 11.897796s
Epoch: 13       Training Loss: 0.000130         Validation Loss: 0.002547       Training Acc: 0.994062  Validation Acc: 0.937500        Time: 12.050992s
Epoch: 14       Training Loss: 0.000266         Validation Loss: 0.002868       Training Acc: 0.987500  Validation Acc: 0.928750        Time: 12.068385s
Epoch: 15       Training Loss: 0.000375         Validation Loss: 0.002258       Training Acc: 0.982187  Validation Acc: 0.922500        Time: 11.604305s
Total Time: 181.257526s
```

图 6: 训练过程控制台输出

训练结束后，将模型文件保存到本地。

3.3 模型预测

为了完成项目任务，我们需要利用训练得到的模型对测试集中的数据进行语种分类。

首先，按照处理训练集的方法处理测试集，这其中包括数据读取、格式转换和维度统一等工作。然后从本地读取保存的模型权重文件，加载为自定义的模型类，然后逐条为测试集数据打标签即可。

测试集的输出标签保存为`test.csv`文件。

4 项目代码

https://github.com/Benson114/DATA620001_Final