**语音多分类实验报告**

1. **实验简介**

文本相关的语音识别任务，是指从音频文件的声纹信息中提取数据特征，然后通过统计学或者机器学习算法，对该音频可能属于的小组成员进行判断的任务。

1. **实验环境**

pytorch+t4

1. **实验内容**
2. **数据集介绍**

本实验所用的数据集为小组成员对统一文本的朗读音频文件，每位成员的文件包含两个训练集，一个测试集，文本内容为“百舸争流”，音频时长为3秒，小组成员共6位。

1. **模型介绍**

本实验自定义Dataset类，使用torchaudio读取数据，并在可视化的基础上设置标签，转化为数。由于单通道只有序列数据，所以只有3维，因此进行一维卷积。

1. **实验目的**

了解如何通过音频信息的数据特征，并在特定环境下进行声纹识别的要求。

1. **实验过程**

###### 步骤一：自定义数据集用于测试，并进行数据处理、参数处理

#截取自定义数据集部分的代码#

class WavDataset(Dataset):

def \_\_init\_\_(self, data\_folder, length=300000, transform=None):

self.data\_folder = data\_folder

self.dim = length

self.wav\_list = []

self.transform = transform

formats = [".wav", ".WAV"]

for root, dirnames, filenames in os.walk(data\_folder):

for filename in filenames:

if os.path.splitext(filename)[1] in formats:

label = str(root).split("/")[-1]

self.wav\_list.append([os.path.join(root, filename), label])

def \_\_getitem\_\_(self, item):

filename, label = self.wav\_list[item]

wb\_wav, sr = load(filename)

wb\_wav = wb\_wav[[0], :] # 单声道

length = wb\_wav.shape[1]

if length >= self.dim:

max\_audio\_start = length - self.dim

audio\_start = np.random.randint(0, max\_audio\_start)

wb\_wav = wb\_wav[audio\_start: audio\_start + self.dim]

else:

wb\_wav = F.pad(wb\_wav, (0, self.dim - length), "constant")

if self.transform is not None:

wb\_wav = normalize(self.transform(wb\_wav))

return wb\_wav, sr, filename, label

def \_\_len\_\_(self):

return len(self.wav\_list)

###### 步骤二：建立模型

class M5(nn.Module):

def \_\_init\_\_(self, n\_input=1, n\_output=35, stride=16, n\_channel=32):

super().\_\_init\_\_()

self.conv1 = nn.Conv1d(n\_input, n\_channel, kernel\_size=80, stride=stride)

self.bn1 = nn.BatchNorm1d(n\_channel)

self.pool1 = nn.MaxPool1d(4)

self.conv2 = nn.Conv1d(n\_channel, n\_channel, kernel\_size=3)

self.bn2 = nn.BatchNorm1d(n\_channel)

self.pool2 = nn.MaxPool1d(4)

self.conv3 = nn.Conv1d(n\_channel, 2 \* n\_channel, kernel\_size=3)

self.bn3 = nn.BatchNorm1d(2 \* n\_channel)

self.pool3 = nn.MaxPool1d(4)

self.conv4 = nn.Conv1d(2 \* n\_channel, 2 \* n\_channel, kernel\_size=3)

self.bn4 = nn.BatchNorm1d(2 \* n\_channel)

self.pool4 = nn.MaxPool1d(4)

self.fc1 = nn.Linear(2 \* n\_channel, n\_output)

def forward(self, x):

x = self.conv1(x)

x = F.relu(self.bn1(x))

x = self.pool1(x)

x = self.conv2(x)

x = F.relu(self.bn2(x))

x = self.pool2(x)

x = self.conv3(x)

x = F.relu(self.bn3(x))

x = self.pool3(x)

x = self.conv4(x)

x = F.relu(self.bn4(x))

x = self.pool4(x)

x = F.avg\_pool1d(x, x.shape[-1])

x = x.permute(0, 2, 1)

x = self.fc1(x)

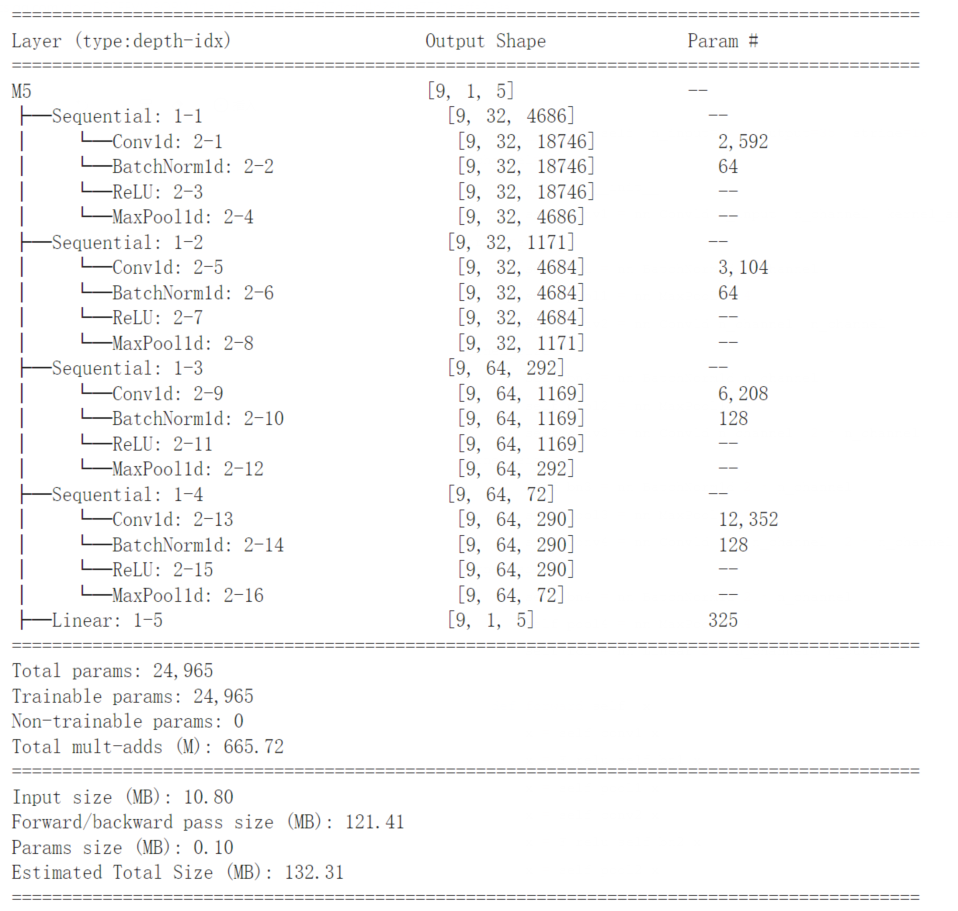
return F.log\_softmax(x, dim=2)

model = M5(n\_input=transformed.shape[0], n\_output=len(labels))

model.to(device)

print(model)

模型如图所示：



###### 步骤三：进行训练完成任务

1. **部分代码介绍**

###### 6.1 设置device与超参数 全部代码如下：

import torch

import torch.nn as nn

import torch.nn.functional as F

import torch.optim as optim

from torchaudio import load, transforms

import librosa

import matplotlib.pyplot as plt

import IPython.display as ipd

from tqdm import tqdm

from torch.utils.data import Dataset, DataLoader

import os

import glob

import numpy as np

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

print(device)

tf = transforms.MFCC(sample\_rate=8000)

def normalize(tensor):

tensor\_minusmean = tensor - tensor.mean()

return tensor\_minusmean / tensor\_minusmean.max()

###### 6.2 自定义数据集 全部代码如下：

class WavDataset(Dataset):

def \_\_init\_\_(self, data\_folder, length=300000, transform=None):

self.data\_folder = data\_folder

self.dim = length

self.wav\_list = []

self.transform = transform

formats = [".wav", ".WAV"]

for root, dirnames, filenames in os.walk(data\_folder):

for filename in filenames:

if os.path.splitext(filename)[1] in formats:

label = str(root).split("/")[-1]

self.wav\_list.append([os.path.join(root, filename), label])

def \_\_getitem\_\_(self, item):

filename, label = self.wav\_list[item]

wb\_wav, sr = load(filename)

wb\_wav = wb\_wav[[0], :] # 单声道

length = wb\_wav.shape[1]

if length >= self.dim:

max\_audio\_start = length - self.dim

audio\_start = np.random.randint(0, max\_audio\_start)

wb\_wav = wb\_wav[audio\_start: audio\_start + self.dim]

else:

wb\_wav = F.pad(wb\_wav, (0, self.dim - length), "constant")

if self.transform is not None:

wb\_wav = normalize(self.transform(wb\_wav))

return wb\_wav, sr, filename, label

def \_\_len\_\_(self):

return len(self.wav\_list)

###### 6.3 读取数据并进行可视化：

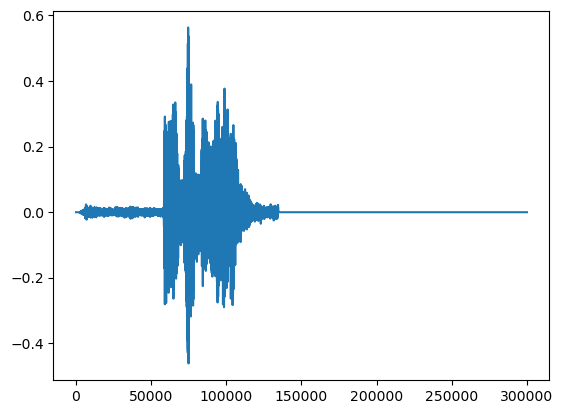
train\_set= WavDataset("datasets/train/")

test\_set = WavDataset("datasets/test/")

wavform, sr, name, label = train\_set[0]

plt.plot(wavform.t().numpy());

结果如图：



###### 6.4 transform并设置sample\_rate，将标签转化为数，并以“WX”测试函数

new\_sample\_rate = 8000

transform = transforms.Resample(orig\_freq=sr, new\_freq=new\_sample\_rate)

transformed = transform(wavform)

ipd.Audio(transformed, rate=new\_sample\_rate)

def label\_to\_index(word):

return torch.tensor(labels.index(word))

def index\_to\_label(index):

return labels[index]

word\_start = "WX"

index = label\_to\_index(word\_start)

word\_recovered = index\_to\_label(index)

print(word\_start, "-->", index, "-->", word\_recovered)

###### 6.5 定义collate\_fn整理数据，便于整理成NCHW格式

def collate\_fn(batch):

# A data tuple has the form:

# wb\_wav, sr, filename, label

tensors, targets = [], []

# Gather in lists, and encode labels as indices

for waveform, \*\_, label in batch:

tensors += [waveform]

targets += [label\_to\_index(label)]

# Group the list of tensors into a batched tensor

tensors = pad\_sequence(tensors)

targets = torch.stack(targets)

return tensors, targets

###### 6.6 模型建立

class M5(nn.Module):

def \_\_init\_\_(self, n\_input=1, n\_output=35, stride=16, n\_channel=32):

super().\_\_init\_\_()

self.conv1 = nn.Conv1d(n\_input, n\_channel, kernel\_size=80, stride=stride)

self.bn1 = nn.BatchNorm1d(n\_channel)

self.pool1 = nn.MaxPool1d(4)

self.conv2 = nn.Conv1d(n\_channel, n\_channel, kernel\_size=3)

self.bn2 = nn.BatchNorm1d(n\_channel)

self.pool2 = nn.MaxPool1d(4)

self.conv3 = nn.Conv1d(n\_channel, 2 \* n\_channel, kernel\_size=3)

self.bn3 = nn.BatchNorm1d(2 \* n\_channel)

self.pool3 = nn.MaxPool1d(4)

self.conv4 = nn.Conv1d(2 \* n\_channel, 2 \* n\_channel, kernel\_size=3)

self.bn4 = nn.BatchNorm1d(2 \* n\_channel)

self.pool4 = nn.MaxPool1d(4)

self.fc1 = nn.Linear(2 \* n\_channel, n\_output)

def forward(self, x):

x = self.conv1(x)

x = F.relu(self.bn1(x))

x = self.pool1(x)

x = self.conv2(x)

x = F.relu(self.bn2(x))

x = self.pool2(x)

x = self.conv3(x)

x = F.relu(self.bn3(x))

x = self.pool3(x)

x = self.conv4(x)

x = F.relu(self.bn4(x))

x = self.pool4(x)

x = F.avg\_pool1d(x, x.shape[-1])

x = x.permute(0, 2, 1)

x = self.fc1(x)

return F.log\_softmax(x, dim=2)

model = M5(n\_input=transformed.shape[0], n\_output=len(labels))

model.to(device)

print(model)

###### 6.7 设置优化器

optimizer = optim.Adam(model.parameters(), lr=learning\_rate, weight\_decay=0.0001)

scheduler = optim.lr\_scheduler.StepLR(optimizer, step\_size=20, gamma=0.1) # reduce the learning after 20 epochs by a factor of 10

###### 6.8 运行

losses = []

def train(model, epoch, log\_interval):

model.train()

for batch\_idx, (data, target) in enumerate(train\_loader):

data = data.to(device)

target = target.to(device)

# apply transform and model on whole batch directly on device

data = transform(data)

output = model(data)

# 计算 loss

loss = F.nll\_loss(output.squeeze(), target)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

# 打印训练进度

if batch\_idx % log\_interval == 0:

print(f"Train Epoch: {epoch} Loss: {loss.item():.6f}")

# 记录 loss

losses.append(loss.item())

# 计算预测正确的数目

def number\_of\_correct(pred, target):

return pred.squeeze().eq(target).sum().item()

# 找到最有可能的标签

def get\_likely\_index(tensor):

return tensor.argmax(dim=-1)

def test(model, epoch):

model.eval()

correct = 0

for data, target in test\_loader:

data = data.to(device)

target = target.to(device)

# apply transform and model on whole batch directly on device

data = transform(data)

output = model(data)

pred = get\_likely\_index(output)

correct += number\_of\_correct(pred, target)

print(f"Test Epoch: {epoch} Accuracy: {correct}/{len(test\_loader.dataset)} ({100. \* correct / len(test\_loader.dataset):.0f}%)\n")

log\_interval = 10 # 每10个batch打印一次训练结果

# The transform needs to live on the same device as the model and the data.

transform = transform.to(device)

for epoch in range(1, epochs + 1):

train(model, epoch, log\_interval)

test(model, epoch)

scheduler.step()

###### 6.9 可视化损失计算函数

plt.plot(losses)

plt.xlabel("Step", fontsize=12)

plt.ylabel("Loss", fontsize=12)

plt.title("Training Loss")

结果为：

Train Epoch: 1 Loss: 1.669137

Test Epoch: 1 Accuracy: 2/9 (22%)

Train Epoch: 2 Loss: 1.328404

Test Epoch: 2 Accuracy: 2/9 (22%)

Train Epoch: 3 Loss: 1.162512

Test Epoch: 3 Accuracy: 3/9 (33%)

……

Train Epoch: 98 Loss: 0.002460

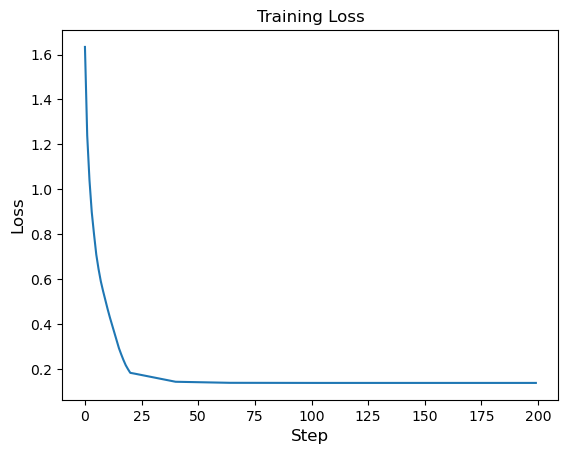
Test Epoch: 98 Accuracy: 7/9 (78%)

Train Epoch: 99 Loss: 0.002460

Test Epoch: 99 Accuracy: 7/9 (78%)

Train Epoch: 100 Loss: 0.002460

Test Epoch: 100 Accuracy: 7/9 (78%)



1. **实验总结**

本次实验对语音文件数据进行处理分类识别，自定义了数据集，自定义了模型，并设置了优化器进行训练，采取用单声道进行输入，直接对序列进行一维卷积，最后得出结果后将结果进行可视化。了解了对语音文件识别、分类的相关操作。

1. **实验完成时间和提交方式**

完成时间：2023/4/19

提交方式：word文档