华东师范大学数据科学与工程学院实验报告

课程名称： 当代人工智能　　年级： 大三　　上机作业：实验五 多模态情感分析

姓名： 许瑞琪　　学号： 10205501450　　上机实践时间：2023.6.9 - 2023.7.14

实验任务

设计一个多模态融合模型。给定配对的文本和图像，预测对应的情感标签，为一个三分类任务，类别有 positive, neutral, negative。任务示例如下：

'??? #stunned #sunglasses #gafas #gafasdesol \n'

Positive

实验数据集（本人将其更名为 origin_data，放在 data 文件夹下）中包含三个部分：data 文件夹包括所有的训练文本和图片各 4869 条数据，每个文件按照唯一的 guid 命名。train.txt 文件为数据的 guid 和对应的情感标签，一共 4000 条训练及验证数据。test_without_label.txt 文件为数据的 guid 和空的情感标签，共有 511 条测试数据。

实验要求自行从训练集中划分验证集，调整超参数。预测测试集（test_without_label.txt）上的情感标签。

额外要求把自己的代码上传到 github。本人代码上传到的github 地址在此。

实验环境

实验环境是 Visual Studio Code，实验所用编程语言为 python。

在此基础上，我额外在本次实验的根目录下上传了在 Imagenet 等数据集上训练得很好的公开的预训练模型和模型权重的包，放在 model 文件夹中，当我们使用相同的网络结构时可以直接拿来调用 transformers 预训练模型库。

存放预训练模型和模型权重的文件夹 model/pretrain/bert-base-uncased 里面包含基于 pytorch 的 bert 模型，共四个文件：tokenizer.json 为词汇表（bert 的分词器），config.json 为本人为了提高准确率得分而修改的配置文件，bert_config.json 为该中文 bert 模型的原始配置文件（代码运行实际不会再用到它），pytorch_model.bin 为网上下载的已训练好的 bert 模型权重文件（怕文件太大，bert 预训练模型并没有随代码提交，如果有什么问题的话再联系我吧）。

以上是我用到的特殊 python 代码库并附加的该代码库作用介绍。加入该预训练模型库后，我们便可以开始构建多模态融合模型。

实验过程

每一种信息的来源或者形式，都可以称为一种模态。例如，人有触觉，听觉，视觉，嗅觉；信息的媒介，有语音、视频、文字等。多模态机器学习旨在建立能够处理和关联来自多种模态信息的模型。

图像和文本配对有助于多语种检索任务，并且更好地理解如何配对文本和图像输入可以提升图像描述任务。视觉和文本数据上的协同训练有助于提升视觉分类任务的准确率和稳健性，同时图像、视频和语音任务上的联合训练能够提升所有模态的泛化性能。只要能够转换成数字形式并且保留语义信息的数据，都能被机器加以利用。

为此，本次实验将融合文本和图像数据信息来构建多模态融合模型，并对成对的文本和图像数据做情感分析，预测对应情感标签。

数据预处理

数据读取

首先进行数据预处理。刚开始我用常规方法读入文本和图像数据时，图像读入无误，但文本读入用 utf-8 通用的解码方式会报错。进入报错的待读取文件进行查看发现其编码方式为相对过时的 ANSI，也即文本不都是以 utf-8 方式编码的，ANSI 方式编码的文本用 utf-8 方式解码会报错。

为解决文本读入问题，我使用了 BS 的内置库 UnicodeDammit, 主要用来文档编码自动检测。检测某段未知编码时，可以使用 UnicodeDammit() 方法先检测编码方式，再用 .unicode_markup 得到正确的解码结果。即不同编码方式的文本都可以用该方法被程序正确的读入。

运用该方法读入文本数据时会遇到该提示：`Some characters could not be decoded, and were replaced with REPLACEMENT CHARACTER.` 问题不大，大部分数据都正确读取出来可以直接用，据说出现提示是因为 Twitter 的一些表情符号没法被解码，自动换成一些替代符号了。

在 data_deal.py 文件中我对数据进行方便读取的处理，还写了个划分数据集以及读取文本数据的函数。数据处理代码如下：

```
def train_split(file,split=True):
    train = pd.read_csv(file)
    img_path = []
    text_content = []
    label = []
    label_to_id ={"negative":0,"neutral":1,"positive":2}
    for index , row in train.iterrows():
        id = str(int(row ["guid"]))
        img = os.path.join("data/origin_data/data",id+".jpg")
        text = os.path.join("data/origin_data/data",id+".txt")
        img_path.append(img)
        try :
            label.append(label_to_id[row["tag"]])
        except:
            label.append(-1)
        with open(text,"rb") as f:
            dammit = UnicodeDammit(f.read())
            text_content.append(dammit.unicode_markup)
    train["image"]=img_path
```

```
    train["text"] =text_content
    train["label"] =label
    if split:
        train_data, test_data = train_test_split(train, train_size=0.8,
random_state=50, stratify=train['label'])
        # 保存训练集和验证集为csv文件
        train_data.to_csv("data/train.csv",columns=
["image","text","label"],sep=",",index=False)
        test_data.to_csv("data/valid.csv",columns=
["image","text","label"],sep=",",index=False)
    else:
        train.to_csv("data/test.csv", columns=["image", "text", "label"], sep=",",
index=False)
```

对 data/origin_data/ 目录下的两个文本（训练集或测试集）数据每行循环，找到其 guid 对应的图像和文本数据所在路径。若文本为训练集文本，将每行对应的标签转换为数字 0、1 或 2，方便后续网络中的分类。若遇到标签为 null、在 label_to_id 找不到对应转换规则的数据（测试集数据）时，则先把标签记为 -1。

然后根据前面找到的对应文本所在路径读取文本数据后进行数据文件重写，格式为：图像数据所在路径,"文本数据",类别标签（数字），并存为 csv 格式。如果是 train.txt 文本则需要划分训练集和验证集，在还没写入为新文件的训练集上根据 4:1 的大小划分验证集，以学号为随机种子，以不同标签的比例划分，让训练集与验证集的不同标签比例大致相同。

至此，训练集、验证集、测试集相关信息被存入 data 文件夹下的 train.csv、valid.csv、test.csv 文件中。

数据特征处理

接下来是要进入多模态模型训练及预测的数据处理，每对数据包含大小不一的图片，以及单词数目不同的文本。需要进行成对图像与文本信息的融合，解决不同模态数据的向量空间不一致问题，增强模型针对评论的情感分析的能力。

批处理样本都有一个固定的长度。在本次学习任务中，每个样本都是由成对的图像与文本特征信息融合而成的，其中每条信息数据列可能具有不同的长度。

为了提高计算效率，读入数据集后一次处理一个 batch 的数据序列，同一个 batch 中的每条数据都应该具有相同的长度，因此我们需要对各个不同大小的图像和文本提取相同长度的特征并拼接，便于后续训练。

- 图像数据的处理过程：先根据图片地址找到并读取图像数据，不进行填充。我先把每个图像转化为 3 * 224 * 224 的 3 通道灰度图像，把每条数据转化为 tensor 类型并用均值和标准差对张量图像进行归一化后，提取大小为 1000 的特征。
- 文本数据的处理过程：数据集文件存的已经是文本数据本身，因此直接对文本进行编码，当文本序列的词元数目不等于 256 时，在其末尾进行填充或者截断，并返回张量。随后对文本进行编码，得到输出特征并取第一个隐藏层作为文本特征，大小为 768。

这里解释一下特征提取所用模型。图像向量用 torchvision.models 库里预训练的 resnet18 模型提取特征，并将最后一层替换为一个线性层，用于将图片向量变换为 1000 维图片特征。文本使用 AutoTokenizer 和 AutoModel 自动加载预训练的 BERT 模型作为特征提取器处理文本，用 encode_plus 方法对文本进行编码、填充和截断后自动分词，再对文本进行编码，得到文本特征。

图像和文本数据提取特征后，图像特征之间、文本特征之间将具有相同的长度，将两种特征拼接，依旧能以相同形状的 batch 进行加载。对训练集、验证集和测试集做同样的特征序列长度处理，才能用于后续任务。返回图像和文本数据特征及其对应标签并包装成数据集的特征提取方法代码如下：

```
class MultiModalDataSet(Dataset):
    def __init__(self, csv_file, img_transform=None, text_transform=None,
need_text=True, need_image=True):
        self.data = pd.read_csv(csv_file)
        self.need_text = need_text
        self.need_image = need_image
        self.img_transform = img_transform or torchvision.transforms.Compose([
            torchvision.transforms.Resize((224, 224)),
            torchvision.transforms.Grayscale(num_output_channels=3),
            torchvision.transforms.ToTensor(),
            torchvision.transforms.Normalize(mean=[0.485, 0.456, 0.406], std=
[0.229, 0.224, 0.225])
        ])
        self.text_transform = text_transform or
transformers.AutoTokenizer.from_pretrained("model\\pretrain\\bert-base-uncased",
local_files_only=True)
        self.text_model =
transformers.AutoModel.from_pretrained("model\\pretrain\\bert-base-uncased",
local_files_only=True)
        self.image_model = torchvision.models.resnet18(pretrained=True)
        self.image_model.eval()

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        image_path = self.data.iloc[idx, 0]
        if image_path and self.need_image:
            image = Image.open(image_path)
            image = self.img_transform(image)
            image = self.image_model(image.unsqueeze(0))
            image = image[0]
        else:
            image = torch.zeros(1000)
        text = self.data.iloc[idx, 1]
        if text and self.need_text:
            text =
self.text_transform.encode_plus(text,return_tensors="pt",padding="max_length",trun
cation=True,max_length=256)
            text_output = self.text_model(**text)
            text_feature = text_output[0][:,0,:]
            text_feature= text_feature.squeeze(0)
        else:
            text_feature = torch.zeros(768)
        label = int(self.data.iloc[idx, 2])
        feature = torch.cat([text_feature, image], dim=0)
        sample = {"feature":feature, "label": label}
        return sample
```

数据批处理

定义批处理数据。用上一小节构建数据集的 MultiModalDataSet 方法对数据进行打包后，将数据集用 DataLoader 方法拆为 batch_size 大小的 batch，对训练集进行乱序降低过拟合可能。批处理后，后续训练时调用 tqdm 方法显示进度条可以更好地进度查询。

至此，适用于多模态融合模型的数据预处理完毕。

模型框架

从技术上讲，由于在前面数据处理过程中已经提取了数据特征，输入模型的数据已经是经过信息提取得到的特征向量，因此可以设置一个简单的网络，直接将特征丢进全连接层进行分类，模型架构较为简单，模型代码如下：

```
class MultiModalClassify(nn.Module):
    def __init__(self):
        super(MultiModalClassify, self).__init__()
        self.fc1 = nn.Linear(1768,512)
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(512, 3)

    def forward(self,x):
        hidden = self.fc1(x)
        hidden = F.relu(hidden)
        hidden = self.dropout(hidden)
        output = self.fc2(hidden)
        return output
```

模型的前向传播对输入的图片和文本特征数据提取特征，得到一个 512 维的向量，接着通过激活函数增加非线性，通过 dropout 层防止过拟合，再通过一个全连接层把处理过的特征分为三类，分别对每个类别进行评分，输出分类结果为一个三维的向量，三维中最大值即图像最可能类别。

至此，多模态融合模型框架构建完毕。

训练、验证与测试

开始模型训练之前，要先创建模型实例，定义超参数、优化器和损失函数。batch_size 设为 32，从进度条反映出的分母可以推出当前数据集的总数据量。我电脑用 CPU 跑速度很慢，训练加验证一轮大概要两个多小时，因此只设 epoch 为 5，学习率为 0.01。

定义优化方式和损失函数，我选用的优化方式为公认“好用”的自适应算法 Adam 优化器，自适应调整学习率；损失函数采用通常用于多分类问题上的交叉熵损失函数 CrossEntropyLoss。

接着写模型训练框架。在使用 pytorch 构建神经网络的时候，训练过程中会在程序上方添加一句 model.train()，作用是启用 batch normalization 和 dropout。

开始在 EPOCH 个轮次中训练模型。定义一个训练函数，输入模型、数据加载器、损失函数和优化器，返回训练损失、准确率和预测概率。

遍历训练数据加载器获取每个批次的数据，获取图片、文本和标签，前向传播得到模型输出后对展开成同维度的输出结果和真实标签对比计算损失，再反向传播根据梯度计算下一次的权重值，即正向传播后进行反向传播更新网络参数。因为可能出现梯度爆炸的情况，注意每个 batch 前梯度要清零。 累加训练损失和准确率，出循环后计算训练损失和准确率的平均值；将模型输出转换为概率，并添加到列表中，最后三者均作为返回值输出。进度条会根据批处理数据量显示训练进度。训练函数代码如下：

```
def train(model, loader, criterion, optimizer):
    train_loss = 0.0
    train_acc = 0.0
    train_proba = []
    model.train()
    for batch in tqdm(loader):
        feature = batch["feature"].to(device)
        label = batch["label"].to(device)
        optimizer.zero_grad()
        output = model(feature)
        loss = criterion(output, label)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
        train_acc += (output.argmax(dim=-1) == label).sum().item()
        proba = F.softmax(output, dim=-1).detach().cpu().numpy()
        train_proba.extend(proba)
    train_loss /= len(loader)
    train_acc /= len(loader.dataset)
    return train_loss, train_acc, train_proba
```

在每个 EPOCH 中模型训练后，生成的模型 model 要用来测试验证集样本了。在测试之前，需要加上 model.eval()，否则的话，有输入数据，即使不训练它也会改变权值。而用 eval() 后，pytorch 会自动把 BN 和 DropOut 固定住，不会取平均，而是用训练好的值。

用验证集数据对训练成的模型做评估，观察训练好的模型在验证集上表现效果如何。定义一个验证函数，输入模型、数据加载器和损失函数，返回测试损失、准确率和预测概率。

遍历测试数据加载器，获取每个批次的图片、文本和标签，前向传播得到模型输出后与真实标签比较计算损失，累加测试损失和准确率，计算测试损失和准确率的平均值；将模型输出转换为概率，并添加到列表中，最后三者均作为返回值输出。函数返回前保存每轮训练到的模型，省得预测时还要重新训练。进度条会根据批处理数据量显示验证进度。验证函数代码如下：

```
def valid(model, loader, criterion):
    valid_loss = 0.0
    valid_acc = 0.0
    valid_proba = []
    model.eval()
    for batch in tqdm(loader):
        feature = batch["feature"].to(device)
        label = batch["label"].to(device)
        output = model(feature)
        loss = criterion(output, label)
```

```
            valid_loss += loss.item()
            valid_acc += (output.argmax(dim=-1) == label).sum().item()
            proba = F.softmax(output, dim=-1).detach().cpu().numpy()
            valid_proba.extend(proba)
        print('Saving model......')
        torch.save(model.state_dict(), 'model/net_%02d.pth' % (epoch + 1))
        valid_loss /= len(loader)
        valid_acc /= len(loader.dataset)
        return valid_loss, valid_acc, valid_proba
```

测试函数和验证函数结构大致相同，输入测试集数据经测试后只返回预测结果。用最后一次训练得到的模型进行测试，模型测试函数代码如下：

```
def predict_func(model, loader):
    test_proba = []
    model.eval()
    for batch in tqdm(loader):
        feature = batch["feature"].to(device)
        label = batch["label"].to(device)
        output = model(feature)
        proba = F.softmax(output, dim=-1).detach().cpu().numpy()
        test_proba.extend(proba)
    return  test_proba
```

最后将测试返回的分类概率最大结果的数字标签映射为对应的文本标签，将预测结果写入 predict_result.txt 文件中，预测文件格式与测试集文件格式一致。

消融实验结果

分别只输入文本或图像数据，分析多模态融合模型在验证集会获得怎样的表现。

创建数据集时，分别设置参数 need_text 和 need_image 为 FALSE，则将不读入文本或图像的特征，将原本文本或图像的特征向量置为等长零向量，生成数据集就不包含这部分特征，达到只输入图像或只输入文本数据的效果。

数据处理完毕后同样用最后一次训练得到的模型数据，调用 predict.py 里的 predict_func() 方法进行验证。得到最大分类概率转化为文本标签后与真实标签对比分别得到只输入文本或图像数据的准确率后输出。

至此，实验代码部分介绍完毕，详细代码解读已随代码注释给出。

代码上传 GitHub

首先新建一个仓库 repository，为仓库取名 MultiModal，通过本地上传文件到 GitHub 上。

接着在本地新建一个文件夹用来存放项目文件，在本地文件夹中打开 git bash。使用`git clone https://github.com/<user_name>/<repo_name>.git` 克隆到本地。

然后将我的项目文件复制或移动到克隆下来的仓库同名文件夹中，然后输入 `git add .` 命令，将所有文件添加到暂存区。输入 `git commit -m "<information>"` 命令，将暂存区的文件提交到本地仓库，并添加一个提交

信息。输入 `git push` 命令，将本地仓库的内容推送到 GitHub 上的仓库，到 GitHub 上刷新出现最新一次的提交信息，则代码上传成功。

```
Xu Dashuai@LAPTOP-6KL6BD2E MINGW64 ~/Desktop/大学/编程/当代人工智能/GitHub5/Mult
iModal (main)
$ git add .
warning: LF will be replaced by CRLF in model/MultiModal.py.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in model/pretrain/bert-base-uncased/bert_co
nfig.json.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in model/pretrain/bert-base-uncased/config.
json.
The file will have its original line endings in your working directory
```

```
Xu Dashuai@LAPTOP-6KL6BD2E MINGW64 ~/Desktop/大学/编程/当代人工智能/GitHub5/Mult
iModal (main)
$ git commit -m "model withoutbin"
[main 79057de] model withoutbin
 14 files changed, 59 insertions(+)
 create mode 100644 model/MultiModal.py
 create mode 100644 model/__init__.py
 create mode 100644 model/__pycache__/MultiModal.cpython-37.pyc
 create mode 100644 model/__pycache__/MultiModal.cpython-38.pyc
 create mode 100644 model/__pycache__/__init__.cpython-37.pyc
 create mode 100644 model/__pycache__/__init__.cpython-38.pyc
 create mode 100644 model/net_01.pth
 create mode 100644 model/net_02.pth
 create mode 100644 model/net_03.pth
 create mode 100644 model/net_04.pth
 create mode 100644 model/net_05.pth
 create mode 100644 model/pretrain/bert-base-uncased/bert_config.json
 create mode 100644 model/pretrain/bert-base-uncased/config.json
 create mode 100644 model/pretrain/bert-base-uncased/tokenizer.json
```

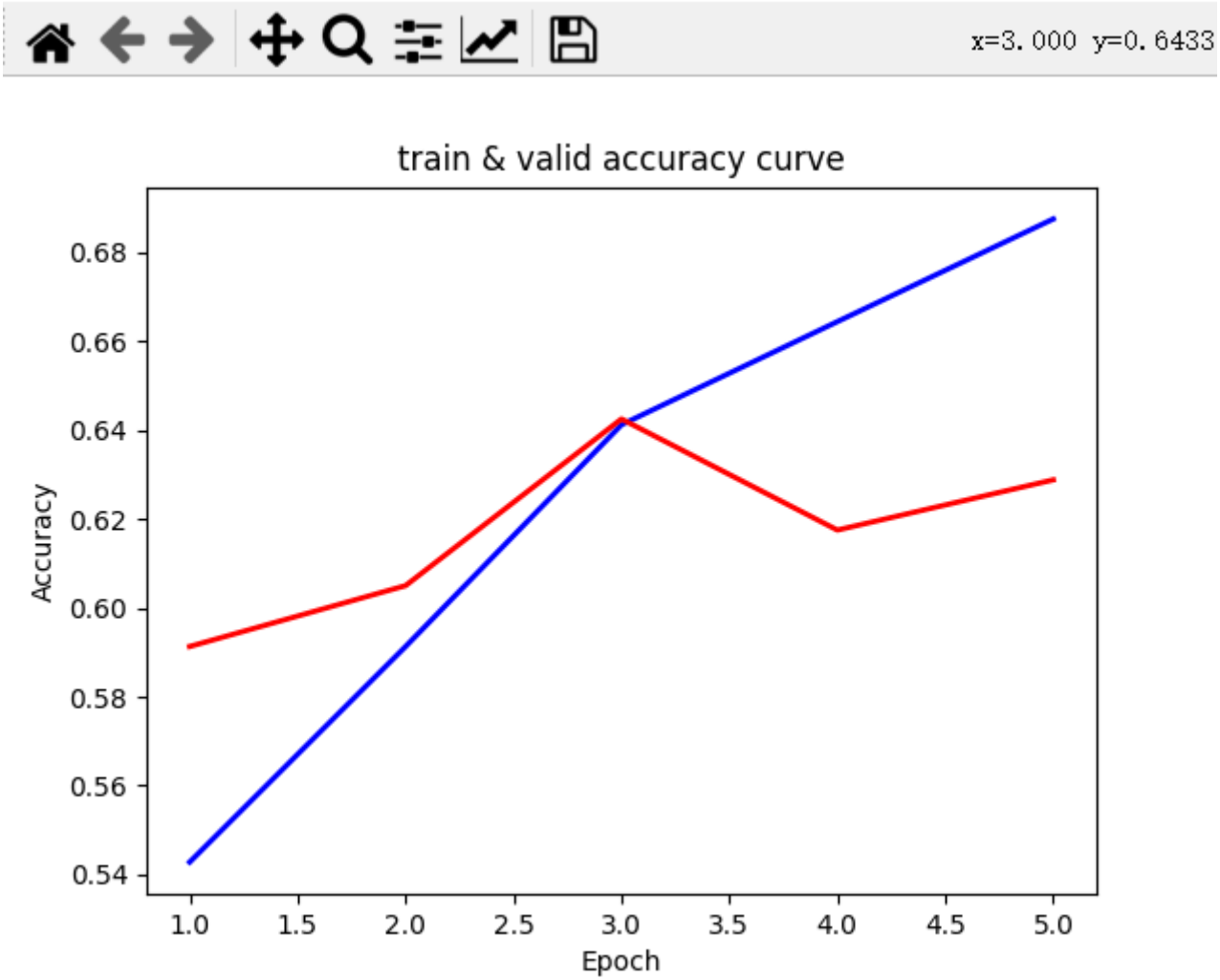
```
Xu Dashuai@LAPTOP-6KL6BD2E MINGW64 ~/Desktop/大学/编程/当代人工智能/GitHub5/Mult
iModal (main)
$ git push
Enumerating objects: 21, done.
Counting objects: 100% (21/21), done.
Delta compression using up to 8 threads
Compressing objects: 100% (18/18), done.
Writing objects: 100% (20/20), 12.59 MiB | 13.01 MiB/s, done.
Total 20 (delta 3), reused 1 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/XuDashuai0827/MultiModal.git
   302508e..79057de  main -> main
```

实验结果分析

以下是本次实验超参数设置 EPOCH 为 5、学习率为 0.001 的训练及验证过程和对应准确率结果：

```
start MultiModal training
==========
Epoch [1/5] training start:
100%|| 100/100 [1:59:49<00:00, 71.90s/it]
Epoch [1/5] validating start:
100%|| 25/25 [12:25<00:00, 29.82s/it]
Saving model......
Epoch [1/5], train loss: 3.3584, train acc: 0.5428
Epoch [1/5], valid loss: 1.4667, valid acc: 0.5913
Epoch [2/5] training start:
100%|| 100/100 [2:15:36<00:00, 81.36s/it]
Epoch [2/5] validating start:
100%|| 25/25 [14:06<00:00, 33.84s/it]
Saving model......
Epoch [2/5], train loss: 1.0806, train acc: 0.5913
Epoch [2/5], valid loss: 0.9646, valid acc: 0.6050
Epoch [3/5] training start:
100%|| 100/100 [2:18:00<00:00, 82.80s/it]
Epoch [3/5] validating start:
100%|| 25/25 [13:25<00:00, 32.23s/it]
Saving model......
Epoch [3/5], train loss: 0.7984, train acc: 0.6412
Epoch [3/5], valid loss: 0.8721, valid acc: 0.6425
Epoch [4/5] training start:
100%|| 100/100 [2:20:51<00:00, 84.51s/it]
Epoch [4/5] validating start:
100%|| 25/25 [13:54<00:00, 33.40s/it]
Saving model......
Epoch [4/5], train loss: 0.7455, train acc: 0.6644
Epoch [4/5], valid loss: 0.8937, valid acc: 0.6175
Epoch [5/5] training start:
100%|| 100/100 [2:19:19<00:00, 83.60s/it]
Epoch [5/5] validating start:
100%|| 25/25 [12:23<00:00, 29.73s/it]
Saving model......
Epoch [5/5], train loss: 0.7240, train acc: 0.6875
Epoch [5/5], valid loss: 0.9656, valid acc: 0.6288
```

生成的（其中一次）准确率曲线图如下所示：



从实验结果趋势图可以看到，训练集准确率（蓝色曲线）是不断提升的，然而在测试集表现好的模型在验证集上准确率不一定能每轮都提升，图中三、四轮 epoch 时验证集准确率下降（红色曲线），证明模型出现了过拟合倾向。

测试集预测过程：

```
Some weights of the model checkpoint at model\pretrain\bert-base-uncased were not used when initializing BertModel: ['cls.seq_relationship.bias', 'cls.predictions.transform
.LayerNorm.bias', 'cls.predictions.transform.dense.weight', 'cls.seq_relationship.weight', 'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.dense.bi
as', 'cls.predictions.bias', 'cls.predictions.decoder.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenc
eClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassificatio
n model from a BertForSequenceClassification model).
100%|                                                                                                                                                  | 16/16 [06:22<00:00, 23.89s/it]
```

根据进度条可以看到 511 条测试集，当每个 batch_size 为 32 时，确实会被分为 16 个 batch。

消融验证集数据实验结果：

```
text only
100%|                                                                                                                                                  | 25/25 [09:57<00:00, 23.90s/it]
image only
100%|                                                                                                                                                  | 25/25 [00:49<00:00,  2.00s/it]
with out image auc is 0.741047906842886
with out text auc is 0.6490608573514286
```

消融实验不管是只输入文本或图像准确率都比最后一次验证集输出得分高，可能是由于多模态融合模型架构仅对提取出的特征做了简单变换就得出分类结果导致。而只输入文本数据准确率比只输入图片数据准确率高，可能跟二者用的模型中，用于处理文本的 bert 预训练模型要高级一些有关。

实验问题

本次实验中遇到的数据维度与网络不相适配的情况相比上次作业改善了许多，其中有深度学习期末大作业让我们实现中文拼写检测的功劳，我从中举一反三了不少东西到这次 AI 实验来，学以致用。总之，有数据问题就一个个输出调试即可，总能凑出正确的来。

还有就是不知道为什么，消融代码中明明只引用 predict.py 里的 predict_func() 函数，但还是会连对测试集预测的过程再重复进行一遍。不过不是很占时间就放任它去吧... 消融实验时忽视输出的第一个 batch 数为 16 的进度条即可。

在部署 GitHub 从本地上传文件时，文件夹只能从本地上传，因此多遇到很多问题。网上方法五花八门容易学杂了，下面总结几个我遇到的问题及解决方案。

- 仓库的 clone 和 push 过程中遇到 "Connection was aborted" 或者超时的报错为网络问题，解决方法参考此处，当然重试几次也行。
- 上传大文件时遇到 "fatal: the remote end hung up unexpectedly" 问题即文件太大，运行 `git config http.postBuffer 524288000` 增加缓存空间到 500M。但由于我的文件太大超过 GitHub 允许范围了，因此修改了 postBuffer 也不起作用。
- 回溯到历史状态运行 `git reset --hard HEAD~x`，这句命令的意思是回退到 x 个 commit 之前，这 x 次提交所改动的代码都没有了，用于操作失误时退回之前的正确状态。

其他就没遇到什么问题了。

实验总结

本次实验我对多模态融合模型的应用及实现做了一定了解与学习，学习途径包括但不限于看视频、找论文、查 CSDN。综合所学内容按我的个人理解，我认为多模态融合模型和前几次实验学习的神经网络模型最大的不同在于，多模态融合模型先把多种不同类型的数据分别进行数据处理，利用不同的神经网络分别进行特征提取，将得到的特征向量拼接融合成完整特征，最后放进一个处理总特征的模型统一分类。

也得益于前几次以及其他课程的学习内容，帮助我更高效掌握了本次实验相关知识；且额外要求的 GitHub 代码传输也让我对 GitHub 使用方法温故知新，操作愈发熟练，就是我家这个网速慢得离谱，传个文件得对网页反复刷新，人麻了。

总的来说，这次实验让我收获颇丰，获得感满满。

参考文献

1. https://blog.csdn.net/weixin_30254435/article/details/96843211
2. https://blog.csdn.net/weixin_45658131/article/details/109243548
3. https://blog.csdn.net/weixin_45658131/article/details/112863150