第五次实验实验报告

姓名：张昊天 **学号**：10215304453
仓库地址：
https://github.com/5048429/AI_5th.git

一、**实验名称**：多模态情感分析

二、实验目的：结合前几次实验与课上学到的知识与实现的模型，实际实现多模态情感分析的分类问题，加深对各类模型的理解。

三、实验任务：三分类任务：positive, neutral, negative。对给定配对的文本和图像，预测对应的情感标签。其中给定配对的文本和图像在 data 目录下，train.txt 中包含训练数据的序号和对应标签，需要根据训练数据训练模型预测 test_without_labels.txt 中序号对应的情感标签。

四、实验环境：Visual Studio Code python 3.11.5　Pytorch

五、实验过程：

1.首先进行**数据处理**，

(1) .train.txt 与 test_without_label.txt 文件中包含了用到的标签数据，需要将唯一标识符“guid”和对应的情感标签“tag”获取并存储，便于后续使用。

```python
def load_labels(filename):
    labels = {}
    with open(filename, mode='r', encoding='utf-8') as file:
        reader = csv.reader(file, delimiter=',')
        next(reader)  # 跳过标题行
        for row in reader:
            guid, tag = row
            labels[guid] = tag
    return labels
```

(2) .使用之前读到的的标签数据，加载 data 目录下对应的文本和图像数据，对于文本数据，使用 open 函数读取文件内容，使用 errors='replace'来处理编码错误，以保证即使文本中有些字符因为编码问题无法正确解读，代码也不会因此抛出异常，从而允许程序继续运行。对于图像数据，使用 PIL 库的 Image.open 函数加载图像文件。

```python
def load_data(data_dir, labels):
    data = []
    for guid, tag in labels.items():
        text_file = os.path.join(data_dir, f"{guid}.txt")
        image_file = os.path.join(data_dir, f"{guid}.jpg")

        # 读取文本数据
        with open(text_file, 'r', encoding='utf-8', errors='replace') as file:
            text_data = file.read()

        # 读取图像数据
        image_data = Image.open(image_file)

        data.append((guid, text_data, image_data, tag))
    return data
```

(3) .之后记载数据后使用 train_test_split 函数将加载的数据划分为训练集和验证集

```python
# 使用数据加载函数
labels = load_labels("train.txt")
data_dir = 'data'
data = load_data(data_dir, labels)

# 划分数据集
train_data, val_data = train_test_split(data, test_size=0.2, random_state=42)
```

(4) .定义一个转换流水线 image_transform 来处理图像数据,因为之后使用了预训练模型 ResNet50，需要图形具有一致的尺寸将图像的大小统一调整为 224x224 像素。并且将图像数据从 PIL 图像转换为 PyTorch 的 Tensor 格式，并将图像的像素值从 0-255 范围重新缩放到 0-1 范围。最后对图像进行标准化，使用 ImageNet 数据集上的均值和标准差，调整每个颜色通道，便于模型更快更好地收敛。

```
# 图像转换函数
image_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

(5) .对文本数据进行预处理，将文本数据转为模型可接受的格式。

```
# 加载tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

(6) .继承 PyTorch 的 Dataset 类定义一个数据集类，在 __getitem__方法中，对每个样本的图像数据通过之前定义的 image_transform 进行处理，文本数据通过 BERT 分词器处理，包括填充或阶段到最大长度、转换为 Tensor。并将文本标签转换为 0、1、2 的数字形式。

```python
# 自定义数据集类
class MultimodalDataset(Dataset):
    def __init__(self, data, tokenizer, max_text_len=512):
        self.data = data
        self.tokenizer = tokenizer
        self.max_text_len = max_text_len
        self.label_mapping = {'negative': 0, 'neutral': 1, 'positive': 2}

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

    def __getitem__(self, idx):
        _, text, image, tag = self.data[idx]

        # 处理图像
        image = image_transform(image)

        # 处理文本
        text = self.tokenizer(text, padding='max_length', max_length=self.max_text_len, truncation=True, return_tensors="pt")

        # 获取标签
        label = self.label_mapping[tag]

        return image, text, label
```

2.定义一个多模态类，

(1) .网络结构的初始化

首先使用 models.resnet50(pretrained=True)来初始化图像模型。加载了预训练的 ResNet50 模型，用于提取图像特征。预训练模型携带了在大规模数据集（如 ImageNet）上学习到的丰富特征，有助于提高模型在特定任务上的性能。

使用 BertModel.from_pretrained('bert-base-uncased') 来初始化文本模型。这个组件加载了预训练的 BERT 模型，专门用于处理文本数据，提取文本特征。

之后通过遍历 self.image_model.parameters() 并设置 param.requires_grad = False 来冻结 ResNet50 模型的参数。使得在训练过程中，图像模型的权重不会被更新，仅仅使用它作为一个特征提取器。

最后将 ResNet50 的最后一个全连接层替换为 nn.Identity()。使模型只输出特征向量而不执行分类任务。

(2) .前向传播过程

首先图像通过修改后的 ResNet50 模型进行前向传播，提取图像特征。

文本数据通过 BERT 模型进行前向传播，提取文本特征。使用的是 BERT 模型的池化输出（pooler_output），它是对最后一个隐藏层的输出的一个固定大小的表示。

之后图像特征和文本特征通过 torch.cat 函数在特征维度上合并。这样，每个样本的图像和文本特征都被组合成一个单一的特征向量。

最后组合的特征向量被传递给一个线性层（self.classifier），线性层的作用是将融合后的特征映射到最终的分类结果上。分类层的输出维度是 3，对应于三种情感类别（负面、中性、正面）。

```python
# 自定义多模态网络类
class MultimodalNetwork(nn.Module):
    def __init__(self):
        super(MultimodalNetwork, self).__init__()
        self.image_model = models.resnet50(pretrained=True)
        self.text_model = BertModel.from_pretrained('bert-base-uncased')

        # 冻结ResNet50的参数
        for param in self.image_model.parameters():
            param.requires_grad = False

        # 修改ResNet50的最后一层以提取特征，而不是进行分类
        self.image_model.fc = nn.Identity()

        # 定义组合特征后的分类层
        self.classifier = nn.Linear(2048 + 768, 3)

    def forward(self, images, input_ids, attention_mask):
        # 提取图像特征
        img_features = self.image_model(images)

        # 提取文本特征
        text_outputs = self.text_model(input_ids=input_ids, attention_mask=attention_mask)
        text_features = text_outputs.pooler_output

        # 合并特征
        combined_features = torch.cat((img_features, text_features), dim=1)

        # 分类
        logits = self.classifier(combined_features)

        return logits
```

(3) .**设计动机与亮点，**

1.强大的特征提取能力：

图像模型：采用了预训练的 ResNet50 模型作为图像处理部分，这是一个在图像识别任务中表现卓越的深度学习模型。

2. 参数冻结策略：通过冻结 ResNet50 模型的参数，将其作为固定的特征提取器，这样可以减少训练时间和避免过拟合。

3. 特征融合：设计了一个融合图像和文本特征的机制。这种融合方式使得模型能够同时考虑到视觉和文本信息，为情感分析提供更全面的视角。

4. 定制的分类层：使用自定义的线性层（nn.Linear(2048 + 768, 3)）对合并的特征进行分类。这个层结合了来自 ResNet50 和 BERT 模型的特征，有效地映射到最终的情感类别上。

5. 灵活性与效率：模型在设计上既考虑了灵活性，也注重了计算效率，使其适用于不同规模的数据集和各种计算资源。

3.定义一个多模态类，

(1) .实例化训练和验证数据集：使用 MultimodalDataset 类，将处理过的训练数据和验证数据转换为适合模型训练和验证的格式。

(2) .创建 DataLoader：为训练集和验证集分别创建 DataLoader，从而以批处理的方式提供数据，并且支持数据的随机洗牌和并行加载。

```
# 为训练集和验证集创建 DataLoader
train_dataset = MultimodalDataset(train_data, tokenizer)
train_dataloader = DataLoader(train_dataset, batch_size=16, shuffle=True)

val_dataset = MultimodalDataset(val_data, tokenizer)
val_dataloader = DataLoader(val_dataset, batch_size=16, shuffle=False)
```

4.模型训练，

（1）. 设置设备：选择使用 GPU 进行训练。

（2）. 初始化模型、损失函数和优化器：将多模态网络模型移至选定的设备，使用交叉熵损失函数和 Adam 优化器。

（3）. 训练循环：对于设定的迭代次数（epoch），在训练数据上迭代，计算每个批次的损失，执行反向传播，并更新模型的权

重。

```
# 设置设备
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# 模型、损失函数和优化器
model = MultimodalNetwork().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr=1e-4)

# 训练循环
num_epochs = 10

for epoch in range(num_epochs):
    model.train()
    for i, (images, texts, labels) in enumerate(train_dataloader):
        images = images.to(device)
        input_ids = texts['input_ids'].squeeze(1).to(device)
        attention_mask = texts['attention_mask'].squeeze(1).to(device)
        labels = labels.to(device)

        optimizer.zero_grad()
        outputs = model(images, input_ids, attention_mask)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        if (i + 1) % 100 == 0:
            print(f'Epoch [{epoch + 1}/{num_epochs}], Step [{i + 1}/{len(train_dataloader)}], Loss: {loss.item()}')

    print(f'Epoch [{epoch + 1}/{num_epochs}] Finished Training')
```

5.模型评估，

定义评估函数 evaluate_model，在验证集上评估模型的性能，计算分类的准确率，**并记录预测错误的样本。**这有助于理解模型在哪些方面可能需要改进。

```
def evaluate_model(model, dataloader, file_name="error_analysis.txt"):
    model.eval()  # 设置模型为评估模式
    correct = 0
    total = 0

    with torch.no_grad(), open(file_name, "w") as file:
        for images, texts, labels in dataloader:
            images = images.to(device)
            input_ids = texts['input_ids'].squeeze(1).to(device)
            attention_mask = texts['attention_mask'].squeeze(1).to(device)
            labels = labels.to(device)

            outputs = model(images, input_ids, attention_mask)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            # 收集并记录错误预测的样本
            mismatches = (predicted != labels).nonzero(as_tuple=False)
            for idx in mismatches:
                actual_label = labels[idx].item()
                predicted_label = predicted[idx].item()
                file.write(f"Actual Label: {actual_label}, Predicted Label: {predicted_label}\n")

    accuracy = 100 * correct / total
    print(f'Accuracy of the model on the validation set: {accuracy} %')
```

6.预测结果和输出，

(1) .加载测试数据：使用类似的方法加载没有标签的测试数据。

```
# 加载测试数据
def load_test_data(test_file, data_dir):
    test_data = []
    with open(test_file, mode='r', encoding='utf-8') as file:
        reader = csv.reader(file, delimiter=',')
        next(reader)  # 跳过标题行
        for row in reader:
            guid = row[0]
            text_file = os.path.join(data_dir, f"{guid}.txt")
            image_file = os.path.join(data_dir, f"{guid}.jpg")

            # 读取文本数据
            with open(text_file, 'r', encoding='utf-8', errors='replace') as file:
                text_data = file.read()

            # 读取图像数据
            image_data = Image.open(image_file)

            test_data.append((guid, text_data, image_data))
    return test_data

test_data = load_test_data('test_without_label.txt', data_dir)

# 创建测试数据集
class TestDataset(Dataset):
    def __init__(self, data, tokenizer, max_text_len=512):
        self.data = data
        self.tokenizer = tokenizer
        self.max_text_len = max_text_len

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

    def __getitem__(self, idx):
        guid, text, image = self.data[idx]

        # 处理图像
        image = image_transform(image)

        # 处理文本
        text = self.tokenizer(text, padding='max_length', max_length=self.max_text_len, truncation=True, return_tensors="pt")

        return guid, image, text

test_dataset = TestDataset(test_data, tokenizer)
test_dataloader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

(2) .预测函数（predict）：在测试数据上运行模型，收集预测结果。

(3) .结果转换和保存：将模型的预测转换为相应的标签并将结果保存到文件中。

```python
# 预测函数
def predict(model, dataloader):
    model.eval()
    predictions = {}
    with torch.no_grad():
        for guid, images, texts in dataloader:
            images = images.to(device)
            input_ids = texts['input_ids'].squeeze(1).to(device)
            attention_mask = texts['attention_mask'].squeeze(1).to(device)

            outputs = model(images, input_ids, attention_mask)
            _, predicted = torch.max(outputs.data, 1)

            for g, p in zip(guid, predicted):
                predictions[g] = p.item()

    return predictions

# 进行预测
model_predictions = predict(model, test_dataloader)

# 转换预测结果为标签
predicted_labels = {guid: ['negative', 'neutral', 'positive'][label] for guid, label in model_predictions.items()}

print(predicted_labels)

# 将预测结果写入新文件
output_file = 'predicted_labels_0128.txt'

with open(output_file, 'w', encoding='utf-8') as file:
    file.write('guid,tag\n')

    # 写入预测的标签
    for guid, label in predicted_labels.items():
        file.write(f'{guid},{label}\n')

print(f'预测结果已保存到文件：{output_file}')
```

模型在验证集上的结果如下，最高达到 **70.5%**的准确度

```
Epoch [3/10], Step [100/200], Loss: 0.7071638107299805
Epoch [3/10], Step [200/200], Loss: 0.7235562801361084
Epoch [3/10] Finished Training
Accuracy of the model on the validation set: 70.5 %
```

```
Epoch [1/10], Step [100/200], Loss: 0.7988011240959167
Epoch [1/10], Step [200/200], Loss: 0.5584688186645508
Epoch [1/10] Finished Training
Accuracy of the model on the validation set: 67.125 %
Epoch [2/10], Step [100/200], Loss: 0.6507971286773682
Epoch [2/10], Step [200/200], Loss: 0.6159613132476807
Epoch [2/10] Finished Training
Accuracy of the model on the validation set: 63.75 %
Epoch [3/10], Step [100/200], Loss: 0.7071638107299805
Epoch [3/10], Step [200/200], Loss: 0.7235562801361084
Epoch [3/10] Finished Training
Accuracy of the model on the validation set: 70.5 %
Epoch [4/10], Step [100/200], Loss: 0.3674232065677643
Epoch [4/10], Step [200/200], Loss: 0.20071054995059967
Epoch [4/10] Finished Training
Accuracy of the model on the validation set: 65.25 %
Epoch [5/10], Step [100/200], Loss: 0.9101855158805847
Epoch [5/10], Step [200/200], Loss: 1.0329341888427734
Epoch [5/10] Finished Training
Accuracy of the model on the validation set: 61.625 %
Epoch [6/10], Step [100/200], Loss: 0.6803607940673828
Epoch [6/10], Step [200/200], Loss: 0.6207922101020813
Epoch [6/10] Finished Training
Accuracy of the model on the validation set: 63.125 %
Epoch [7/10], Step [100/200], Loss: 0.9624748826026917
Epoch [7/10], Step [200/200], Loss: 0.874904453754425
Epoch [7/10] Finished Training
Accuracy of the model on the validation set: 65.25 %
Epoch [8/10], Step [100/200], Loss: 0.8199388980865479
Epoch [8/10], Step [200/200], Loss: 0.8202042579650879
Epoch [8/10] Finished Training
Accuracy of the model on the validation set: 65.125 %
Epoch [9/10], Step [100/200], Loss: 0.8819740414619446
Epoch [9/10], Step [200/200], Loss: 0.7638258337974548
Epoch [9/10] Finished Training
Accuracy of the model on the validation set: 64.25 %
Epoch [10/10], Step [100/200], Loss: 0.5789544582366943
Epoch [10/10], Step [200/200], Loss: 0.7478683590888977
Epoch [10/10] Finished Training
Accuracy of the model on the validation set: 61.25 %
Accuracy of the model on the validation set: 61.25 %
```

6.消融实验结果，

分别只输入文本或图像数据，得到在验证集上的结果

```python
# 仅文本数据的数据集类
class TextOnlyDataset(Dataset):
    def __init__(self, data, tokenizer, max_text_len=512):
        self.data = data
        self.tokenizer = tokenizer
        self.max_text_len = max_text_len
        self.label_mapping = {'negative': 0, 'neutral': 1, 'positive': 2}

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

    def __getitem__(self, idx):
        _, text, _, tag = self.data[idx]
        text = self.tokenizer(text, padding='max_length', max_length=self.max_text_len, truncation=True, return_tensors="pt")
        label = self.label_mapping[tag]
        return text, label

class ImageOnlyDataset(Dataset):
    def __init__(self, data):
        self.data = data
        self.label_mapping = {'negative': 0, 'neutral': 1, 'positive': 2}

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

    def __getitem__(self, idx):
        _, _, image, tag = self.data[idx]
        image = image_transform(image)
        label = self.label_mapping[tag]
        return image, label
```

```python
# 仅文本的模型
class TextOnlyModel(nn.Module):
    def __init__(self):
        super(TextOnlyModel, self).__init__()
        self.text_model = BertModel.from_pretrained('bert-base-uncased')
        self.classifier = nn.Linear(768, 3)

    def forward(self, input_ids, attention_mask):
        text_outputs = self.text_model(input_ids=input_ids, attention_mask=attention_mask)
        text_features = text_outputs.pooler_output
        logits = self.classifier(text_features)
        return logits

# 仅图像的模型
class ImageOnlyModel(nn.Module):
    def __init__(self):
        super(ImageOnlyModel, self).__init__()
        self.image_model = resnet50(weights=ResNet50_Weights.IMAGENET1K_V1)
        # 冻结ResNet50的参数
        for param in self.image_model.parameters():
            param.requires_grad = False
        self.image_model.fc = nn.Linear(self.image_model.fc.in_features, 3)  # 修改为分类层

    def forward(self, images):
        logits = self.image_model(images)
        return logits
```

```python
# 训练和评估函数
def train_model(model, dataloader, criterion, optimizer, num_epochs=1, is_text_model=False):
    model.train()  # 将模型设置为训练模式
    for epoch in range(num_epochs):
        for i, batch in enumerate(dataloader):
            # 如果是文本模型
            if is_text_model:
                texts, labels = batch
                input_ids = texts['input_ids'].squeeze(1).to(device)
                attention_mask = texts['attention_mask'].squeeze(1).to(device)
                labels = labels.to(device)
                outputs = model(input_ids, attention_mask)  # 传递正确的参数
            else:
                # 如果是图像模型
                images, labels = batch
                images = images.to(device)
                labels = labels.to(device)
                outputs = model(images)

            # 重置梯度
            optimizer.zero_grad()

            # 前向传播
            loss = criterion(outputs, labels)

            # 反向传播和优化
            loss.backward()
            optimizer.step()

            if (i + 1) % 100 == 0:
                print(f'Epoch [{epoch + 1}/{num_epochs}], Step [{i + 1}/{len(dataloader)}], Loss: {loss.item()}')

    print(f'Epoch [{epoch + 1}/{num_epochs}] Finished Training')
```

一个 epoch 在验证集上的结果如下

```
Epoch [1/1], Step [100/200], Loss: 1.0511832237243652
Epoch [1/1], Step [200/200], Loss: 0.6059260368347168
Epoch [1/1] Finished Training
Accuracy of the Text Model on the validation set: 76.84375 %
Epoch [1/1], Step [100/200], Loss: 0.9519345760345459
Epoch [1/1], Step [200/200], Loss: 0.8102669715881348
Epoch [1/1] Finished Training
Accuracy of the Image Model on the validation set: 61.71875 %
```

为了得到与原多模态模型同样条件下的输出结果，在保证其他超参数相同的情况下，同时进行 10epoch，并在验证集上得到的结果如下

```
cuda
Epoch [1/10], Step [100/200], Loss: 1.0498110055923462
Epoch [1/10], Step [200/200], Loss: 0.7482496500015259
Epoch [2/10], Step [100/200], Loss: 0.7246271967887878
Epoch [2/10], Step [200/200], Loss: 0.7929370403289795
Epoch [3/10], Step [100/200], Loss: 0.9160317182540894
Epoch [3/10], Step [200/200], Loss: 1.274042129516016
Epoch [4/10], Step [100/200], Loss: 1.00775253772735 6
Epoch [4/10], Step [200/200], Loss: 0.9982258677482605
Epoch [5/10], Step [100/200], Loss: 0.9333962202072144
Epoch [5/10], Step [200/200], Loss: 1.058244228363037
Epoch [6/10], Step [100/200], Loss: 1.0649231672286987
Epoch [6/10], Step [200/200], Loss: 1.184614896774292
Epoch [7/10], Step [100/200], Loss: 0.991972029209137
Epoch [7/10], Step [200/200], Loss: 0.7762249708175659
Epoch [8/10], Step [100/200], Loss: 0.9536113739013672
Epoch [8/10], Step [200/200], Loss: 1.0249760150909424
Epoch [9/10], Step [100/200], Loss: 0.8444915413856506
Epoch [9/10], Step [200/200], Loss: 0.8965730667114258
Epoch [10/10], Step [100/200], Loss: 0.8346863985061646
Epoch [10/10], Step [200/200], Loss: 0.8472128510475159
Epoch [10/10] Finished Training
Accuracy of the Text Model on the validation set: 59.53125 %
Epoch [1/10], Step [100/200], Loss: 0.8700272440910339
Epoch [1/10], Step [200/200], Loss: 0.8504122495651245
Epoch [2/10], Step [100/200], Loss: 0.6710808873176575
Epoch [2/10], Step [200/200], Loss: 0.7716893553733826
Epoch [3/10], Step [100/200], Loss: 0.7678555846214294
Epoch [3/10], Step [200/200], Loss: 0.678406834602356
Epoch [4/10], Step [100/200], Loss: 0.638258159160614
Epoch [4/10], Step [200/200], Loss: 0.9293013215065002
Epoch [5/10], Step [100/200], Loss: 0.6622889041900635
Epoch [5/10], Step [200/200], Loss: 0.5352687835693359
Epoch [6/10], Step [100/200], Loss: 0.6722620129585266
Epoch [6/10], Step [200/200], Loss: 1.0197017192840576
Epoch [7/10], Step [100/200], Loss: 0.46604567766189575
Epoch [7/10], Step [200/200], Loss: 0.7727079391479492
Epoch [8/10], Step [100/200], Loss: 0.5994027853012085
Epoch [8/10], Step [200/200], Loss: 0.7640274167060852
Epoch [9/10], Step [100/200], Loss: 0.802619457244873
Epoch [9/10], Step [200/200], Loss: 1.0750938653945923
Epoch [10/10], Step [100/200], Loss: 0.7786669135093689
Epoch [10/10], Step [200/200], Loss: 0.6137496829032898
Epoch [10/10] Finished Training
Accuracy of the Image Model on the validation set: 68.53125 %
```

六、部分 bug：

```
Failed to read data/4537.txt with re-detected encoding.
Failed to read data/4968.txt with re-detected encoding.
Failed to read data/3843.txt with re-detected encoding.
Failed to read data/4723.txt with re-detected encoding.
Failed to read data/3327.txt with re-detected encoding.
Failed to read data/1557.txt with re-detected encoding.
Failed to read data/4954.txt with re-detected encoding.
Failed to read data/4227.txt with re-detected encoding.
Failed to read data/4792.txt with re-detected encoding.
Failed to read data/1474.txt with re-detected encoding.
Failed to read data/4277.txt with re-detected encoding.
Failed to read data/4486.txt with re-detected encoding.
```

一开始读取数据时出现无法正确打开 txt 文件的情况，可能很多 txt 文件使用了不同的编码格式，但是文件的编码方式好像不止一种，即使再次检测编码格式，仍然会出现错误

```python
# 尝试读取文本文件
try:
    with open(text_name, 'r', encoding=encoding) as file:
        text = file.read().strip()
except UnicodeDecodeError:
    # 如果发生编码错误，重新检测编码
    with open(text_name, 'rb') as file:
        result = chardet.detect(file.read())
        encoding = result['encoding']
        try:
            with open(text_name, 'r', encoding=encoding) as file:
                text = file.read().strip()
        except UnicodeDecodeError:
            # 如果再次失败，记录错误并继续或使用默认文本
            print(f"Failed to read {text_name} with re-detected encoding.")
            text = ""
```

于是使用 errors='replace'来处理编码错误，将任何无法识别的字符替换为一个替代字符，这样即使文本中有些字符因为编码问题无法正确解读，代码也不会因此抛出异常，从而允许程序继续运行。

```python
# 读取文本数据
with open(text_file, 'r', encoding='utf-8', errors='replace') as file:
    text_data = file.read()
```

```
File e:\Python\Python311\Lib\site-packages\torch\nn\modules\module.py:1518, in
Module._wrapped_call_impl(self, *args, **kwargs)
  1516     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
  1517 else:
-> 1518     return self._call_impl(*args, **kwargs)

File e:\Python\Python311\Lib\site-packages\torch\nn\modules\module.py:1527, in
Module._call_impl(self, *args, **kwargs)
  1522 # If we don't have any hooks, we want to skip the rest of the logic in
  1523 # this function, and just call forward.
  1524 if not (self._backward_hooks or self._backward_pre_hooks or
self._forward_hooks or self._forward_pre_hooks
  1525         or _global_backward_pre_hooks or _global_backward_hooks
  1526         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1527     return forward_call(*args, **kwargs)
  1529 try:
  1530     result = None

Cell In[7], line 27
...
   414 def _check_input_dim(self, input):
   415     if input.dim() != 4:
--> 416         raise ValueError(f"expected 4D input (got {input.dim()}D input)")

ValueError: expected 4D input (got 3D input)
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell
output settings...
```

在构建模型并运行的时候出现过上面问题，出现的错误 ValueError: expected 4D input (got 3D input) 指出模型期望的输入应该是四维的，但实际上它收到了三维的输入。在使用 PyTorch 和时，输入数据通常需要包含额外的维度来表示批次大小，即使是单个数据样本。 在深度学习中，数据通常以 batches 的形式处理。即使一次只处理一个样本，也需要添加一个批次维度。所以模型期望一个四维输入（批次大小，颜色通道，高度，宽度），而不是三维（颜色通道，高度，宽度）。 为了解决这个问题，在单个样本的基础上添加了一个批次维度。

```
File e:\Python\Python311\Lib\site-packages\torch\nn\modules\module.py:1527, in Module._call_impl(self, *args, **kwargs)
  1522 # If we don't have any hooks, we want to skip the rest of the logic in
  1523 # this function, and just call forward.
  1524 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hooks or self._forward_pre_hooks
  1525         or _global_backward_pre_hooks or _global_backward_hooks
  1526         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1527     return forward_call(*args, **kwargs)
  1529 try:
  1530     result = None

Cell In[4], line 28
...
File e:\Python\Python311\Lib\site-packages\torch\nn\modules\linear.py:114, in Linear.forward(self, input)
   113 def forward(self, input: Tensor) -> Tensor:
--> 114     return F.linear(input, self.weight, self.bias)

RuntimeError: mat1 and mat2 shapes cannot be multiplied (1x1128 and 2176x64)
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

还出现过诸如此类的维度不匹配的问题，在上面这个情况下，具体的错误是在执行模型的前向传播时，尝试进行的线性层操作即全连接层时维度不匹配，为了解决这个问题，再次打印检查了self.text_fc 层的输出维度，self.image_model 的输出维度，并进行了更改。

```python
print("Text features size:", text_features.size())
print("Image features size:", image_features.size())
combined_features = torch.cat((text_features, image_features), dim=1)
print("Combined features size:", combined_features.size())
```

```
Text features size: torch.Size([1, 128])
Image features size: torch.Size([1, 1000])
Combined features size: torch.Size([1, 1128])
```

之后还出现过类似的维度匹配问题，通过同样的方式检查维度进行解决。

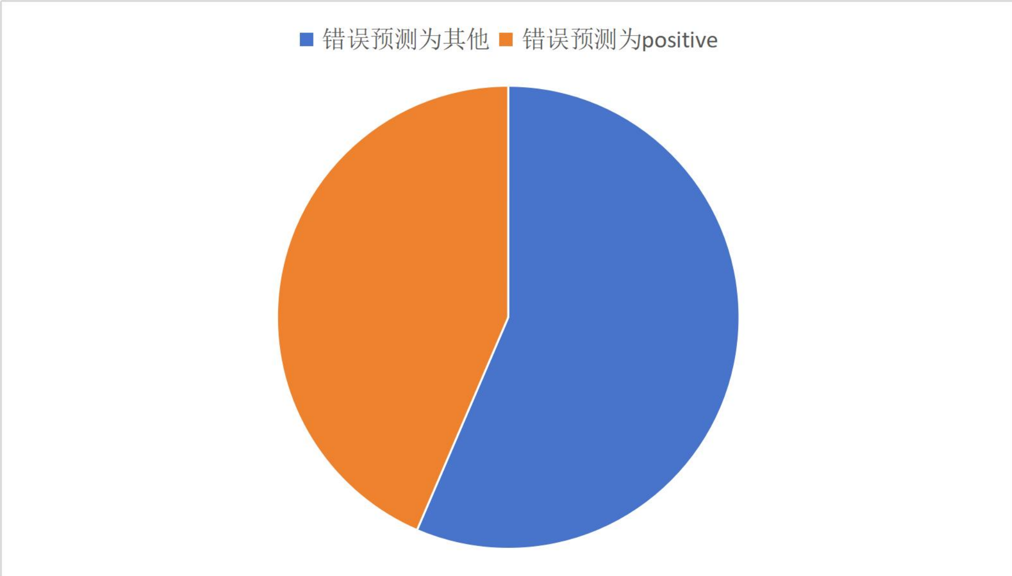
实验过程中还出现过部分如代码语法等其他错误，并未全部记录。

七、回顾总结：

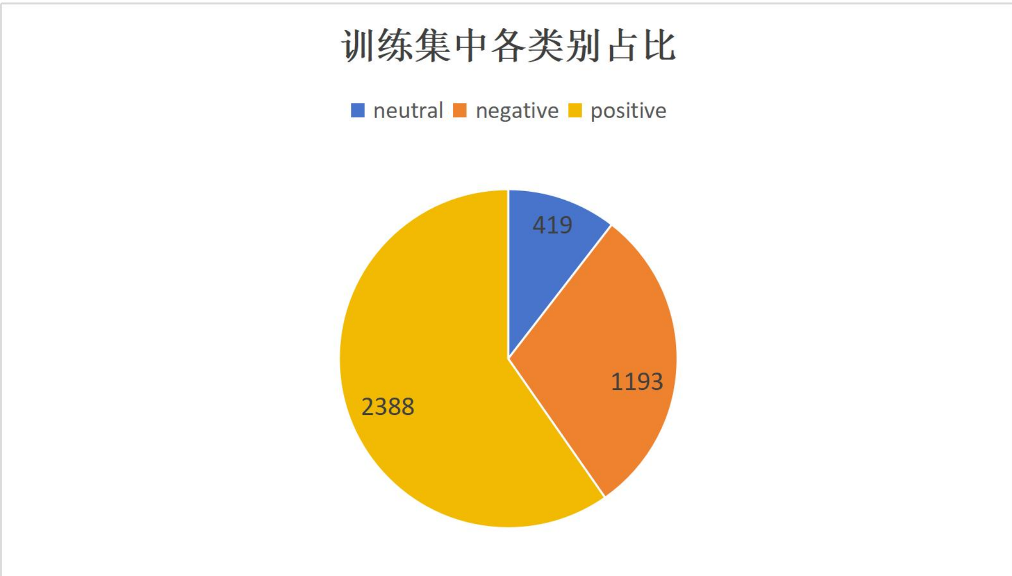
在一开始的多模态训练结果中，发现第三个 epoch 后的验证集准确度最高，并且模型在训练过程中的准确度并不稳定，且整体趋势没有显著的提升。并且根据输出错误的预测情况发现，大部分错误的标签都被预测为了 positive

实验五数据 > error_analysis.txt

```
Actual Label: 0, Predicted Label: 2
Actual Label: 0, Predicted Label: 2
Actual Label: 2, Predicted Label: 0
Actual Label: 1, Predicted Label: 0
Actual Label: 0, Predicted Label: 2
Actual Label: 0, Predicted Label: 2
Actual Label: 2, Predicted Label: 0
Actual Label: 2, Predicted Label: 0
Actual Label: 0, Predicted Label: 2
Actual Label: 0, Predicted Label: 2
Actual Label: 2, Predicted Label: 0
Actual Label: 2, Predicted Label: 0
Actual Label: 2, Predicted Label: 0
Actual Label: 0, Predicted Label: 2
Actual Label: 2, Predicted Label: 0
Actual Label: 0, Predicted Label: 2
Actual Label: 1, Predicted Label: 2
Actual Label: 0, Predicted Label: 2
Actual Label: 0, Predicted Label: 2
Actual Label: 2, Predicted Label: 0
Actual Label: 2, Predicted Label: 0
Actual Label: 2, Predicted Label: 0
Actual Label: 0, Predicted Label: 2
Actual Label: 2, Predicted Label: 0
Actual Label: 2, Predicted Label: 0
Actual Label: 0, Predicted Label: 2
Actual Label: 0, Predicted Label: 2
Actual Label: 0, Predicted Label: 2
Actual Label: 2, Predicted Label: 0
Actual Label: 2, Predicted Label: 0
Actual Label: 1, Predicted Label: 2
Actual Label: 0, Predicted Label: 2
Actual Label: 0, Predicted Label: 2
Actual Label: 2, Predicted Label: 0
Actual Label: 0, Predicted Label: 2
Actual Label: 1, Predicted Label: 2
Actual Label: 2, Predicted Label: 0
Actual Label: 1, Predicted Label: 0
Actual Label: 1, Predicted Label: 0
Actual Label: 0, Predicted Label: 2
Actual Label: 2, Predicted Label: 0
Actual Label: 2, Predicted Label: 0
Actual Label: 0, Predicted Label: 1
Actual Label: 0, Predicted Label: 2
```



观察 train.txt 训练数据发现，数据的样本类别的样本数量并不平衡，可能会导致模型偏向于预测较多样本的类别



并且由于使用了冻结的 ResNet50,模型可能无法学习到足以区分不同情感的复杂图像特征。当然还有可能有其他原因，比如学习率的值或者优化器的选择。

之后的消融实验，只进行一个 epoch 训练，发现文本模型在验证集上的准确率为 76.84%，而图像模型的准确率为 61.72%。经过 10 个 epoch 的训练，文本模型在验证集上的准确率为 59.53%，而

图像模型的准确率为 68.53%。这些结果与之前的消融实验结果相比显示了一些不同的趋势和问题。文本模型的准确度下降，可能是由于过拟合以及不适当的学习率导致的，但为了与之前多模态的结果进行对比，没有选择修改超参数。而图像模型的性能有所提升，在多个 epoch 后图像模型能够更好地学习和提取对情感分类有用的视觉特征。

与之前多模态输入进行对比，单独的文本模型在某个时点的准确率高达 76.84%，单独的图像模型也可以达到 68.53%，因此单模态模型在这个特定任务上可能比多模态模型表现更好。但因为此次模型只使用了简单的特征拼接，而多模态的表现结果还受到不同模态融合策略的影响。所以并不能简单得出在这个情况下，单模态模型就一定比多模态模型好的结论。如果条件允许的情况下，使用更复杂的融合方法，如使用注意力机制的融合可能会得到更好的结果。