Lab5:多模态情感分析

实验要求：

给定配对的文本和图像，预测对应的情感标签，三分类任务：positive, neutral, negative。

设计一个多模态融合模型。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。

代码仓库在GuanZhuang10/AI_lab5 · GitHub

代码实现

1 加载数据集

`MultiModalDataset` 类定义了数据集的组织方式，该数据集包括文本和图像数据。

1.1 MultiModalDataset 类

这个类用于加载文本和图像数据，将它们组织成模型可以接受的格式。以下是该类的主要部分的解释：

- `__init__` 方法：初始化数据集类的实例，接收文本数据路径 `text_data_path`，图像数据路径 `image_data_path`，训练文件名 `train_file`，BERT tokenizer `tokenizer`，图像变换函数 `image_transform`（可选），以及最大文本长度 `max_length`。
- `load_data` 方法：从文本文件中加载数据。每一行包含一个样本的GUID和标签，通过这些信息组织文本和图像数据。标签（'positive'，'neutral'，'negative'）映射为整数标签（2，1，0）。加载的数据以字典形式存储，包含文本、图像和标签信息。
- `__len__` 方法：返回数据集的总样本数。
- `__getitem__` 方法：根据给定的索引 `idx` 返回对应的样本。这个方法使用tokenizer对文本进行编码，将图像转换为PyTorch张量，并返回一个包含文本输入、图像和标签的字典。

```
class MultiModalDataset(Dataset):
    def __init__(self, text_data_path, image_data_path, train_file, tokenizer,
image_transform=None, max_length=128):
        #初始化函数，接受文本数据路径、图像数据路径、训练文件、BERT tokenizer、图像变换、最
大长度等参数
        # 将这些参数保存为实例变量
        self.text_data_path = text_data_path
        self.image_data_path = image_data_path
        self.train_file = train_file
        self.tokenizer = tokenizer
        self.image_transform = image_transform
        self.max_length = max_length

        self.data = self.load_data()

    def load_data(self):
        # 加载数据的方法
        # 从文本文件中读取行，解析每行中的标识和标签，加载对应的文本和图像数据
        with open(os.path.join(self.text_data_path, self.train_file), 'r',
encoding='utf-8', errors='ignore') as f:
            lines = f.readlines()

        multimodal_set = []

        for line in lines[1:]:
            data = {}
            line = line.replace('\n', '')
            guid, tag = line.split(',')
            text_file_path = os.path.join(self.text_data_path, 'data', guid + '.txt')
            with open(text_file_path, 'r', encoding='utf-8', errors='ignore') as
text_file:
                text = text_file.read()

            image_file_path = os.path.join(self.image_data_path, 'data', guid +
'.jpg')
            image = Image.open(image_file_path).convert('RGB')
            if self.image_transform:
                image = self.image_transform(image)

            if tag == 'positive':
                label = 2
            elif tag == 'neutral':
                label = 1
            else:
                label = 0

            data['text'] = text
            data['image'] = image
            data['label'] = label
            multimodal_set.append(data)
```

```
        return multimodal_set

    def __len__(self):
        # 返回数据集的长度
        return len(self.data)

    def __getitem__(self, idx):
        # 根据给定索引返回数据集中的一个样本
        # 返回一个包含文本、图像和标签的字典
        data = self.data[idx]
        text = data['text']
        image = data['image']
        label = int(data['label'])

        inputs = self.tokenizer.encode(text, truncation=True, padding='max_length',
max_length=self.max_length,
                                       return_tensors='pt')

        return {'text_input': inputs.squeeze(),
                'image': image,
                'label': torch.tensor(label)}
```

1.2 数据加载和划分训练集/验证集的代码

数据加载和划分流程：

- `BERT Tokenizer`设置：通过 `BertTokenizer.from_pretrained('bert-base-uncased')` 创建一个BERT tokenizer。
- 数据集实例创建：利用 `MultiModalDataset` 类创建一个文本和图像数据集实例。指定文本数据路径、图像数据路径、训练文件、BERT tokenizer，以及图像预处理操作。
- 训练集和验证集划分：获取数据集的总样本数 `total_size`。指定验证集的大小 `val_size`，计算训练集的大小 `train_size`。使用 `random_split` 函数将数据集划分为训练集和验证集。
- 数据加载器创建：利用 `DataLoader` 类创建训练集和验证集的数据加载器。指定批量大小、是否打乱数据以及多线程加载数据的工作进程数。

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

# 创建文本和图像数据集实例
text_image_dataset = MultiModalDataset(text_data_path, image_data_path, train_file,
tokenizer,

image_transform=transforms.Compose([transforms.Resize((224, 224)),

transforms.ToTensor()]))

# 划分训练集和验证集
total_size = len(text_image_dataset)
val_size = 500
train_size = total_size - val_size

text_image_train_dataset, text_image_val_dataset = random_split(text_image_dataset,
[train_size, val_size])

# 创建文本和图像数据加载器
text_image_train_loader = DataLoader(text_image_train_dataset, batch_size=batch_size,
shuffle=True,
                                     num_workers=num_workers)
text_image_val_loader = DataLoader(text_image_val_dataset, batch_size=batch_size,
shuffle=False,
                                   num_workers=num_workers)
```

2. 模型设计

由于在实验一和实验三中对比了图像分类和文本分类各种方法的优劣，所以在进行多模态分类前，我先设计了一个 `ResNet50` 模型以实现纯图像分类，和一个 `BERT` 模型以实现纯文本分类。

2.1 图像分类

在预训练过程中，我首先定义了一个ResNet50模型，将其最后一层全连接层修改为适应问题的三个类别，并将模型移动到GPU。随后，选择了交叉熵损失函数和Adam优化器。通过迭代训练数据集，该模型进行了五个epoch的训练，对每个批次进行前向传播、损失计算和反向传播，以优化模型参数。在每个epoch结束后，通过验证数据集评估模型性能，输出平均验证损失和准确度。最终，训练好的ResNet50模型参数被保存到文件 'resnet50_model.pth' 中。这段代码的目标是进行图像分类任务，通过迭代训练和验证，使模型能够学习并适应数据集中的图像特征，以实现准确的分类预测。

```
# 定义ResNet50模型
resnet50 = models.resnet50()  # 使用weights参数替代pretrained参数
resnet50.fc = nn.Linear(resnet50.fc.in_features, 3)
resnet50 = resnet50.to(device)
# 定义损失函数和优化器
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(resnet50.parameters(), lr=0.001)
```

在`FusionModel`类中的图像模型部分，加载了预训练的ResNet模型的状态字典，即`resnet50_model.pth`

```
        # 创建一个未初始化的ResNet模型
        self.image_model = models.resnet50(pretrained=False)
        # 修改模型的全连接层为Identity，以便匹配FusionModel
        self.image_model.fc = nn.Identity()
        # 加载ResNet模型的部分状态字典，不包括最终全连接层
        state_dict = torch.load(image_model_path)
        self.image_model.load_state_dict({k: state_dict[k] for k in state_dict if 'fc'
not in k})
```

这段代码中，首先创建了一个未初始化的ResNet50模型，并将其全连接层替换为`nn.Identity()`，以保留模型的特征提取部分而去掉分类层。然后，通过加载预训练的ResNet模型的状态字典，仅保留不包含最终全连接层的权重信息。这样，`FusionModel`类中的`self.image_model`成为了一个包含ResNet50特征提取部分的模型，准备用于多模态融合模型中。

在使用预训练的ResNet50模型的时候，我一开始遇到了这样的问题：

```
Traceback (most recent call last):
  File "D:\homework\AI\project5\catmodel.py", line 204, in <module>
    train_multimodal_model(text_data_path, image_data_path, train_file, text_model_path,
     image_model_path, num_labels)
  File "D:\homework\AI\project5\catmodel.py", line 142, in train_multimodal_model
    fusion_model = FusionModel(text_model_path, image_model_path, num_labels).to(device)
  File "D:\homework\AI\project5\catmodel.py", line 85, in __init__
    self.image_model.load_state_dict(torch.load(image_model_path))
  File "C:\Users\granger\AppData\Roaming\Python\Python310\site-packages\torch\nn\modules
   \module.py", line 2152, in load_state_dict
    raise RuntimeError('Error(s) in loading state_dict for {}:\n\t{}'.format(
RuntimeError: Error(s) in loading state_dict for ResNet:
    Unexpected key(s) in state_dict: "fc.weight", "fc.bias".
```

遇到的错误表明在加载ResNet模型时，状态字典中存在意外的键。我一开始使用的ResNet模型具有修改过的最终层（全连接层），当尝试加载状态字典时，它期望具有原始最终层的架构。这种不匹配导致错误。

我使用了部分加载模型的方法解决了这个问题，即只加载ResNet的特征提取器部分，并初始化`FusionModel`的其余部分。另外由于`nn.Identity()`没有`in_features`属性，不改变输入维度，所以需要根据需求手动指定`in_features`。

2.2 **文本分类**

在预训练过程中，我使用BERT模型进行文本分类。首先，通过BERT的tokenizer对文本进行标记化，然后创建了文本数据集实例，并将其划分为训练集和验证集。随后，加载BERT模型并将其移动到GPU上，定义了优化器和损失函数。通过迭代训练集，对BERT模型进行训练，其中每个epoch包括前向传播、损失计算、反向传播和参数更新。在每个epoch结束后，通过验证集评估模型性能，输出平均验证损失和准确度。最终，保存训练好的BERT模型参数到文件'bert_model.pth' 中。

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

# 创建文本数据集实例
text_data_path = '.'  # 修改为文本数据所在的文件夹
text_train_file = 'train.txt'
text_dataset = TextDataset(text_data_path, text_train_file, tokenizer)

# 划分训练集和验证集
total_size = len(text_dataset)
val_size = 500
train_size = total_size - val_size

text_train_dataset, text_val_dataset = random_split(text_dataset, [train_size,
val_size])

# 创建文本数据加载器
text_batch_size = 32
text_num_workers = 4
text_train_loader = DataLoader(text_train_dataset, batch_size=text_batch_size,
shuffle=True, num_workers=text_num_workers)
text_val_loader = DataLoader(text_val_dataset, batch_size=text_batch_size,
shuffle=False, num_workers=text_num_workers)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# 创建并将模型移动到设备
bert_model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=3).to(device)

# # 定义BERT模型，并将其移动到GPU
# bert_model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=3).to(device)

# 定义优化器和损失函数
bert_optimizer = optim.AdamW(bert_model.parameters(), lr=2e-5)
bert_criterion = nn.CrossEntropyLoss()
```

在`FusionModel`类中的文本模型部分，加载了预训练的BERT模型的状态字典，即`bert_model.pth`

```
# 加载BERT模型
        self.text_model = BertForSequenceClassification.from_pretrained('bert-base-
uncased', num_labels=num_labels)
        self.text_model.load_state_dict(torch.load(text_model_path))
        self.text_model = self.text_model.bert
```

首先加载了预训练的BERT模型，设计用于文本分类任务，其中 `num_labels` 参数指定了输出类别的数量。接着，通过 `torch.load` 加载了预训练BERT模型的状态字典。最后，通过 `self.text_model.bert` 提取了加载的BERT模型的特征提取部分，去除了预训练模型的分类头。这个过程的目的是在融合多模态信息的模型中，将BERT模型的文本特征提取部分与其他模态（例如图像）的特征一起使用，以实现联合学习的效果。

针对ResNet和BERT模型，将其参数设置为不可训练，以防止在融合模型的训练过程中被更新。

```
        # 冻结模型参数，防止在训练中被更新
        for param in self.image_model.parameters():
            param.requires_grad = False

        for param in self.text_model.parameters():
            param.requires_grad = False
```

创建一个线性层 `fusion_layer`，由于 `nn.Identity()` 没有 `in_features` 属性，不改变输入维度，所以需要根据需求手动指定 `in_features`，即为ResNet的最后一层输出（2048维）和BERT的隐藏层输出维度的总和。

```
        # 定义融合的线性层
        # 手动指定in_features
        in_features = 2048 + self.text_model.config.hidden_size  # 2048是ResNet50最后一层的输出维度
        self.fusion_layer = nn.Linear(in_features, num_labels)
```

2.3 联合处理图像和文本信息

下面的代码定义了 `FusionModel` 类的前向传播方法（ `forward` ），用于联合处理图像和文本信息。

2.3.1 向量相加

图像模型的前向传播：

- 输入图像数据 `image_input` 到 `self.image_model` 中，获取图像特征表示 `image_features` 。
- 通过 `torch.unsqueeze` 在第一个维度上扩展 `image_features` 的维度，以便与文本特征进行连接。

文本模型的前向传播：

- 输入文本数据 `text_input` 到 `self.text_model` 中，获取文本的输出 `text_outputs` 。
- 从 `text_outputs` 中提取最后一层隐藏状态（ `last_hidden_states` ）。

提取文本特征向量：

- 从 `last_hidden_states` 中提取文本特征向量，即CLS标记对应的隐藏状态。
- 使用 `torch.unsqueeze` 在第一个维度上扩展 `text_features` 的维度。

连接图像和文本特征：

- 使用 `torch.cat` 在第三个维度上连接扩展后的 `image_features` 和 `text_features`，形成联合的特征表示 `fused_features`。

融合后的特征传入线性层：

- 将联合特征 `fused_features` 传入线性层 `self.fusion_layer` 进行融合。
- 通过 `squeeze` 操作，将输出的张量维度中的大小为1的维度去除，得到最终的分类 logits。

通过这个前向传播过程，模型能够同时处理图像和文本信息，通过联合学习获得融合的特征表示，并最终将其用于分类任务。这种结构允许模型在处理多模态输入时进行信息的有机整合。

```
    def forward(self, image_input, text_input):
        # 图像模型的前向传播
        image_features = self.image_model(image_input)
        # 扩展 image_features 的维度
        image_features = torch.unsqueeze(image_features, dim=1)
        # 文本模型的前向传播
        text_outputs = self.text_model(text_input)
        last_hidden_states = text_outputs.last_hidden_state
        # 提取特征向量（CLS对应的隐藏状态）
        text_features = last_hidden_states[:, 0, :]
        text_features = torch.unsqueeze(text_features, dim=1)
        # 连接 image_features 和 text_features
        fused_features = torch.cat([image_features, text_features], dim=2)
        # 融合后的特征传入线性层
        logits = self.fusion_layer(fused_features).squeeze()
        return logits
```

在进行向量拼接的时候，我遇到了一些向量维度的问题。

```
Traceback (most recent call last):
  File "D:\homework\AI\project5\catmodel.py", line 211, in <module>
    train_multimodal_model(text_data_path, image_data_path, train_file, text_model_path,
image_model_path, num_labels)
  File "D:\homework\AI\project5\catmodel.py", line 165, in train_multimodal_model
    logits = fusion_model(image_input, text_input_ids, text_attention_mask)
  File "C:\Users\granger\AppData\Roaming\Python\Python310\site-
packages\torch\nn\modules\module.py", line 1518, in _wrapped_call_impl
    return self._call_impl(*args, **kwargs)
  File "C:\Users\granger\AppData\Roaming\Python\Python310\site-
packages\torch\nn\modules\module.py", line 1527, in _call_impl
    return forward_call(*args, **kwargs)
  File "D:\homework\AI\project5\catmodel.py", line 113, in forward
    fused_features = image_features + text_features
RuntimeError: The size of tensor a (2048) must match the size of tensor b (3) at non-singleton
dimension 1
```

因为 `image_features` 的维度是 (2048,)，而 `text_features` 的维度是 (batch_size, num_labels)，不能直接相连，而`torch.cat` 在拼接时要求除了拼接维度之外的维度必须相等，最后扩展 image_features 的维度，并在维度2上进行了拼接（ `dim=2` ）。

2.3.2 注意力机制

注意力机制的作用是使模型能够动态地关注输入中的不同部分，从而更好地捕捉输入之间的关系和重要信息。在多模态模型中，跨模态注意力机制允许模型在处理图像和文本特征时关注彼此之间最相关的部分。

定义交叉注意力层

- `CrossModalAttention` 是一个继承自 `nn.Module` 的PyTorch模块，用于实现交叉注意力机制。
- 在初始化中，它包含一个线性层 `self.fc` ，该层的输入大小是 `input_size*2` ，输出大小是 `input_size` ，这是为了确保输出的形状与输入相同。
- `self.softmax` 是一个Softmax激活函数，用于计算注意力权重。
- 在 `forward` 方法中，输入 `input_1` 和 `input_2` 被拼接在一起，然后通过线性层和Softmax激活函数产生注意力权重。
- 最后，通过将这些注意力权重应用于输入，得到了经过注意力机制调整后的输入 `attended_input_1` 和 `attended_input_2` 。

```python
class CrossModalAttention(nn.Module):
    def __init__(self, input_size):
        super(CrossModalAttention, self).__init__()
        # 用于跨模态注意力的线性层
        self.fc = nn.Linear(input_size*2, input_size)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, input_1, input_2):
        # 拼接输入
        concatenated_input = torch.cat([input_1, input_2], dim=1)
        # 应用线性层和softmax
        fc_output = self.fc(concatenated_input)
        attention_weights = self.softmax(fc_output)
        # 将注意力权重应用到输入上
        attended_input_1 = input_1 * attention_weights
        attended_input_2 = input_2 * attention_weights
        return attended_input_1, attended_input_2
```

定义双流多模态层

- `MultiModalModel` 是一个继承自 `nn.Module` 的PyTorch模块，实现了双流多模态模型。
- 在初始化中，加载了预训练的ResNet-50图像模型和BERT文本模型。
- 包含两个全连接层 `text_fc` 和 `image_fc`，用于处理文本和图像特征。
- 使用 `CrossModalAttention` 模块进行跨模态注意力。
- 包含其他全连接层 `fc1` 和 `fc2`，用于最终的分类输出。
- 在 `forward` 方法中，处理图像和文本输入，应用全连接层，通过跨模态注意力调整特征，最后输出分类结果。

```
class MultiModalModel(nn.Module):
    def __init__(self, text_model_path, image_model_path, num_labels):
        super(MultiModalModel, self).__init__()
        # 加载预训练模型
        self.resnet_model = resnet_model
        self.bert_model = bert_model
        # 用于文本和图像特征的全连接层
        self.text_fc = nn.Sequential(
            nn.Linear(768, 512),
            nn.ReLU(),
            nn.Dropout(0.1)
        )
        self.image_fc = nn.Sequential(
            nn.Linear(2048, 512),
            nn.ReLU(),
            nn.Dropout(0.1)
        )
        # 跨模态注意力层
        self.cross_modal_attention = CrossModalAttention(input_size=512)
        # 其他全连接层
        self.fc1 = nn.Linear(512*2, 512)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(512, num_labels)

    def forward(self, image_input, text_input):
        # 处理图像输入
        image_output = self.resnet_model(image_input)
        # 获取文本输入的最大序列长度
        max_length = self.bert_model.config.max_position_embeddings
        tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
        tokens = tokenizer.batch_decode(text_input, skip_special_tokens=True)
        text = " ".join(tokens)
        # 对文本输入进行分词
        inputs = tokenizer(text, return_tensors='pt', padding='max_length',
truncation=True, max_length=max_length)
        # 提取CLS token表示
        text_output = self.bert_model(**inputs).last_hidden_state[:,
:image_output.size(0), :].squeeze(0)
        # 对文本和图像特征应用全连接层
        image_output = self.image_fc(image_output)
        text_output = self.text_fc(text_output)
        # 应用跨模态注意力
        attended_image, attended_text = self.cross_modal_attention(image_output,
text_output)
        # 拼接注意力后的特征
        merged_representation = torch.cat([attended_image, attended_text], dim=1)
        # 应用其他全连接层
        x = self.fc1(merged_representation)
        x = self.relu(x)
```

```
        output = self.fc2(x)
        return output
```

在使用注意力机制融合文本向量和图像向量时，为了避免过拟合，我在文本和图像的线性层之后添加了Dropout层。训练后的结果显示每个epoch的输出结果相同。

3 模型训练

下面是用于训练双流多模态模型的代码。

```
def train_multimodal_model(text_data_path, image_data_path, train_file,
text_model_path, image_model_path, num_labels,
                           batch_size=32, num_workers=4, num_epochs=5):
    # 设置BERT tokenizer
    ...
    # 创建文本和图像数据集实例
    ...
    # 划分训练集和验证集
    ...
    # 创建文本和图像数据加载器
    ...
    # 创建并将融合模型移动到设备
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    fusion_model = FusionModel(text_model_path, image_model_path,
num_labels).to(device)

    # 定义优化器和损失函数
    fusion_optimizer = optim.AdamW(fusion_model.parameters(), lr=2e-5)
    fusion_criterion = nn.CrossEntropyLoss()

    # 训练融合模型
    for epoch in range(num_epochs):
        # 训练模型
        fusion_model.train()
        for batch in text_image_train_loader:
            text_input = batch['text_input'].to(device)
            image_input = batch['image'].to(device)
            labels = batch['label'].to(device)

            fusion_optimizer.zero_grad()

            logits = fusion_model(image_input, text_input)
            loss = fusion_criterion(logits, labels)

            loss.backward()
            fusion_optimizer.step()

        # 验证模型
        fusion_model.eval()
        val_loss = 0.0
        correct = 0
        total = 0

        with torch.no_grad():
            for batch in text_image_val_loader:
                text_input = batch['text_input'].to(device)
                image_input = batch['image'].to(device)
                labels = batch['label'].to(device)

                logits = fusion_model(image_input, text_input)
```

```
                val_loss += fusion_criterion(logits, labels).item()

                _, predicted = logits.max(1)
                total += labels.size(0)
                correct += predicted.eq(labels).sum().item()

        avg_val_loss = val_loss / len(text_image_val_loader)
        accuracy = correct / total

        print(f'Epoch [{epoch+1}/{num_epochs}], Avg Val Loss: {avg_val_loss},
Accuracy: {accuracy}')
```

在`train_multimodal_model`中，实现了创建数据集和数据加载器，定义优化器和损失函数，训练融合模型的功能，关于创建数据集和数据加载器，已经在加载数据集一节中具体解释过了。

这段代码定义了优化器（`fusion_optimizer`）和损失函数（`fusion_criterion`）。优化器采用AdamW算法，用于更新模型的权重，以最小化模型在训练集上的损失。损失函数使用交叉熵损失，适用于多类别分类问题。

接下来，通过循环遍历每个epoch，在每个epoch内使用训练集进行模型训练。在训练阶段，通过计算损失和反向传播，优化器更新模型的权重。

最后，在每个epoch结束时，切换到模型的验证模式，计算在验证集上的平均损失和准确度，对验证集进行评估。

在使用注意力机制的时候，我通过调整优化器的权重衰减（weight decay）参数来实现 L2 正则化，在创建优化器时设置`weight_decay`参数。

```
def train_multimodal_model(text_data_path, image_data_path, train_file,
text_model_path, image_model_path, num_labels,
                           batch_size=32, num_workers=4, num_epochs=5,
weight_decay=1e-5):  # 添加 weight_decay 参数
    # ...
    # 创建并将融合模型移动到设备
    fusion_model = MultiModalModel(text_model_path, image_model_path, num_labels)
    # 定义优化器和损失函数，添加 weight_decay 参数
    fusion_optimizer = optim.AdamW(fusion_model.parameters(), lr=2e-5,
weight_decay=weight_decay)  # 添加 weight_decay
    # ...
    # 训练融合模型
    for epoch in range(num_epochs):
        # 训练模型
        fusion_model.train()
        for batch in text_image_train_loader:
            # ...
            logits = fusion_model(image_input, text_input)
            loss = fusion_criterion(logits, labels)
            # 添加 weight_decay 到优化器的 step 中
            loss += weight_decay * sum(p.norm(2) ** 2 for p in
fusion_model.parameters())
            loss.backward()
            fusion_optimizer.step()
        # ...
if __name__ == '__main__':
    torch.multiprocessing.freeze_support()
    # 调用训练函数，可以通过调整 weight_decay 参数来控制正则化的强度
    train_multimodal_model(text_data_path, image_data_path, train_file,
text_model_path, image_model_path, num_labels, weight_decay=1e-5)
```

数据分析

Model	Acc
catmodel	82.4%
CMAmodel	64.6%

我的代码有以下一些亮点：

1. **注意力机制**:
    - 实现了跨模态注意力机制（`CrossModalAttention`），增强了文本和图像特征之间的交互，在多模态融合任务中是至关重要的。

2. **模型架构**:
    - `MultiModalModel` 类清晰地定义了多模态融合模型的架构，通过注意力和全连接层将文本和图像特征结合在一起。
3. **训练循环**:
    - 训练循环结构良好，清晰划分了训练和验证阶段。
    - 使用 AdamW 优化器、交叉熵损失和权重衰减，这是训练神经网络的良好选择。
4. **可复现性**:
    - 设置随机种子确保了代码的可复现性，使得在不同运行中结果一致，更易于调试和验证。

消融实验

Model	Acc
BERT	74.8%
Resnet50	60.2%

在多模态情感分类任务中，通过比较catmodel和CMAmodel两种不同的多模态模型，以及BERT和Resnet50的消融实验结果，得出了以下结论。catmodel表现最好，其通过简单的向量连接策略成功捕捉了图像和文本信息之间的关联性，取得了82.4%的准确率。相比之下，CMAmodel采用跨模态注意力的方式，在这个任务中准确率较低，可能是由于注意力机制未充分捕捉到图像和文本的相关性。BERT和Resnet50分别只关注文本和图像信息，准确率也相对较低。