代码地址：https://github.com/Daserxqc/ai-lab5/issues

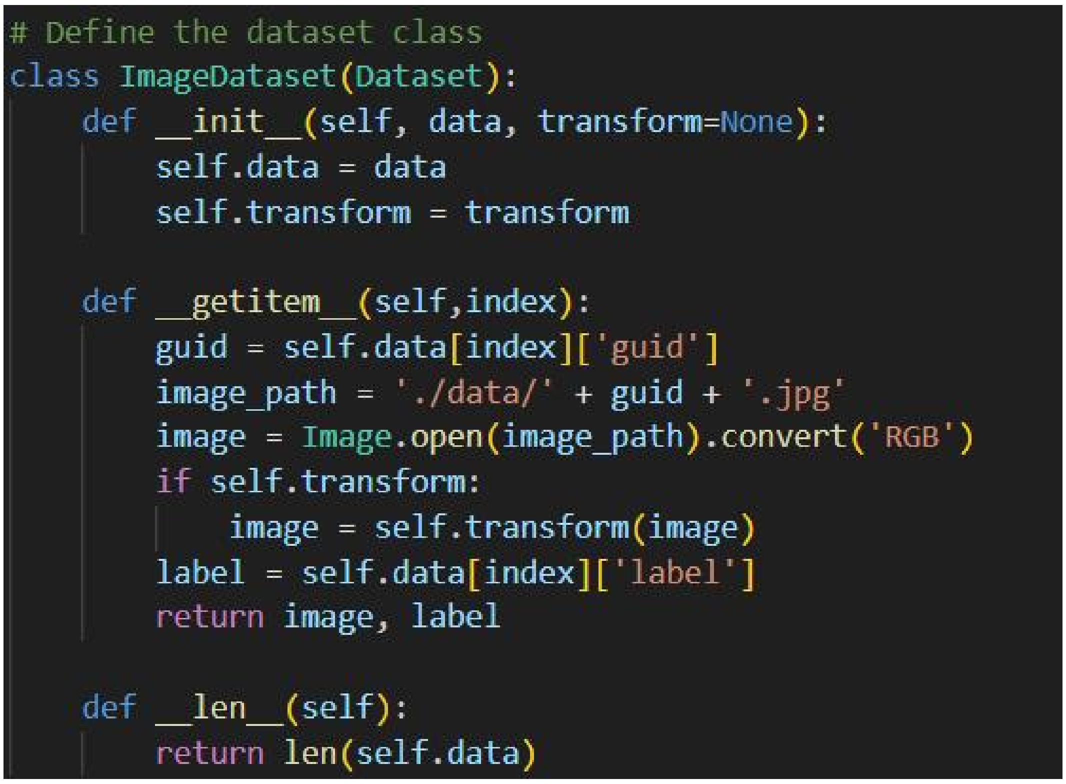
实验过程

我本次实验代码分为图像分类、文本分类、多模态融合三部分

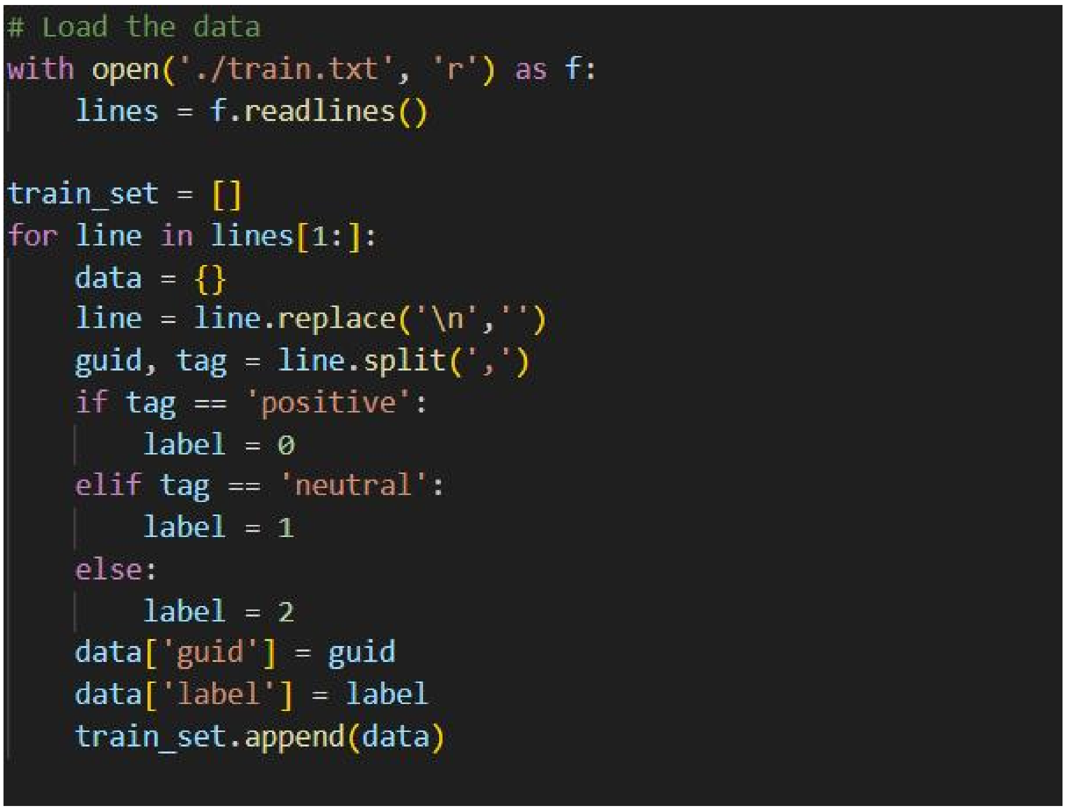
图像分类：

我使用了 efficientnet_pytorch 的 EfficientNet 模型进行图像分类，因为这个没用过所以这次用一下看看效果，亮点的话一是比较高效吧，也正如其名。它使用了一种新的复合缩放方法，通过增加网络深度、宽度和分辨率来提高模型的表现，同时保持模型参数和计算成本的相对较低。二是可扩展性强，EfficientNet 的设计具有良好的可扩展性，通过增加网络深度、宽度和分辨率，可以轻松地将 EfficientNet 扩展到不同的规模，以适应不同的应用需求。还有可解释性也不错，可以帮助研究人员更好地理解卷积神经网络的设计和优化原理。

数据预处理

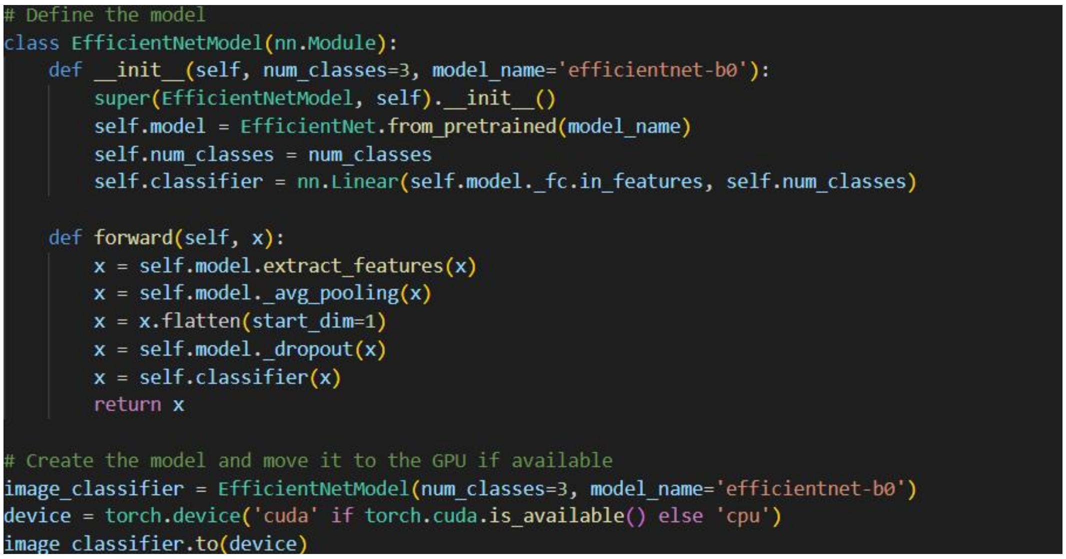


```python
# Define the dataset class
class ImageDataset(Dataset):
    def __init__(self, data, transform=None):
        self.data = data
        self.transform = transform

    def __getitem__(self,index):
        guid = self.data[index]['guid']
        image_path = './data/' + guid + '.jpg'
        image = Image.open(image_path).convert('RGB')
        if self.transform:
            image = self.transform(image)
        label = self.data[index]['label']
        return image, label

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



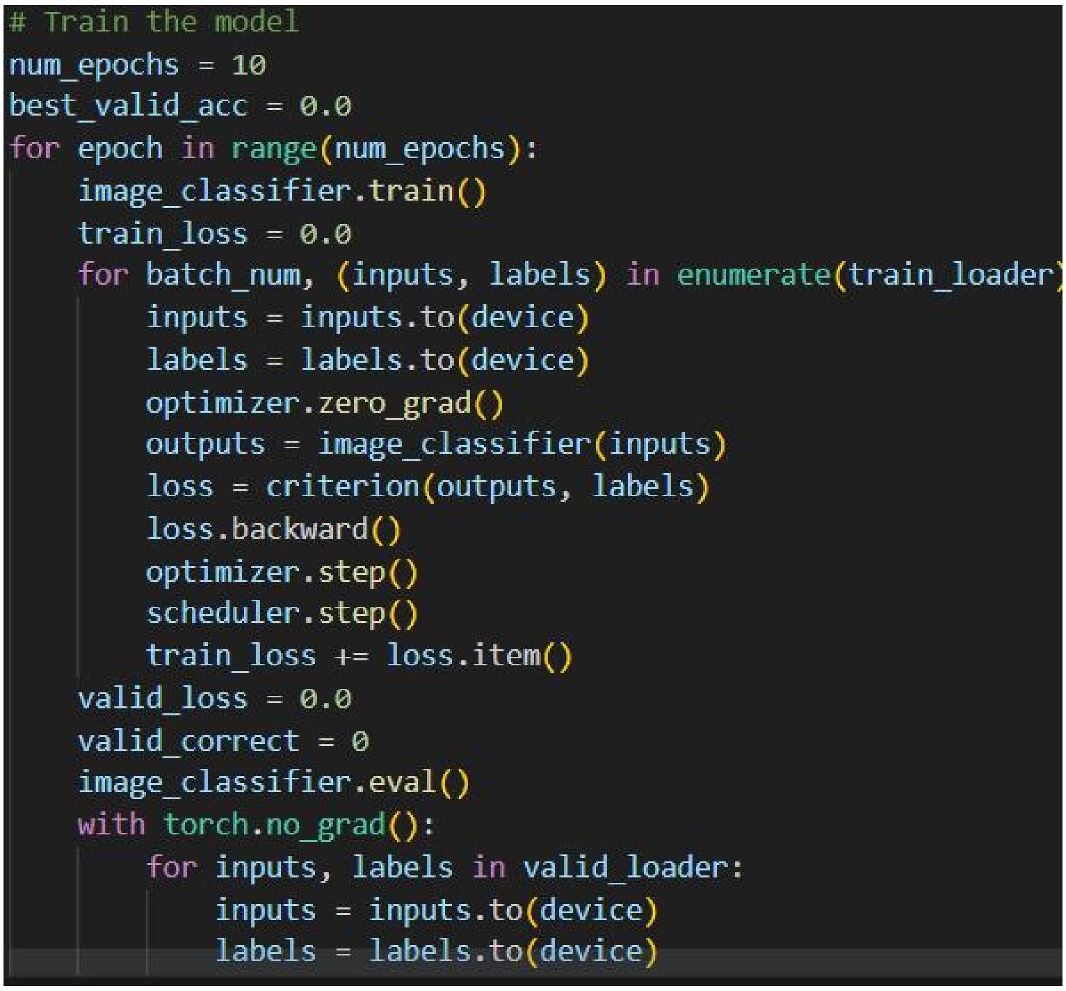
```python
# Load the data
with open('./train.txt', 'r') as f:
    lines = f.readlines()

train_set = []
for line in lines[1:]:
    data = {}
    line = line.replace('\n','')
    guid, tag = line.split(',')
    if tag == 'positive':
        label = 0
    elif tag == 'neutral':
        label = 1
    else:
        label = 2
    data['guid'] = guid
    data['label'] = label
    train_set.append(data)
```

模型定义



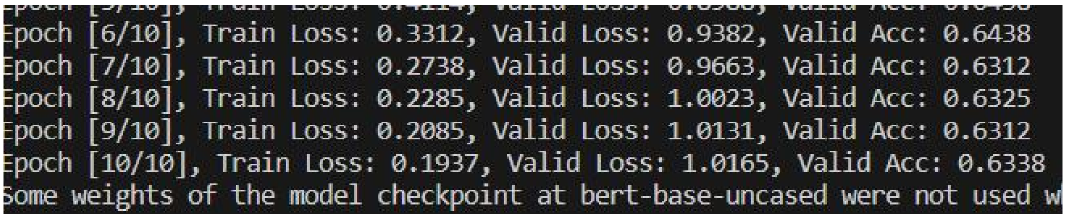
```python
# Define the model
class EfficientNetModel(nn.Module):
    def __init__(self, num_classes=3, model_name='efficientnet-b0'):
        super(EfficientNetModel, self).__init__()
        self.model = EfficientNet.from_pretrained(model_name)
        self.num_classes = num_classes
        self.classifier = nn.Linear(self.model._fc.in_features, self.num_classes)

    def forward(self, x):
        x = self.model.extract_features(x)
        x = self.model._avg_pooling(x)
        x = x.flatten(start_dim=1)
        x = self.model._dropout(x)
        x = self.classifier(x)
        return x

# Create the model and move it to the GPU if available
image_classifier = EfficientNetModel(num_classes=3, model_name='efficientnet-b0')
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
image_classifier.to(device)
```

模型训练



```
# Train the model
num_epochs = 10
best_valid_acc = 0.0
for epoch in range(num_epochs):
    image_classifier.train()
    train_loss = 0.0
    for batch_num, (inputs, labels) in enumerate(train_loader)
        inputs = inputs.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = image_classifier(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        scheduler.step()
        train_loss += loss.item()
    valid_loss = 0.0
    valid_correct = 0
    image_classifier.eval()
    with torch.no_grad():
        for inputs, labels in valid_loader:
            inputs = inputs.to(device)
            labels = labels.to(device)
```

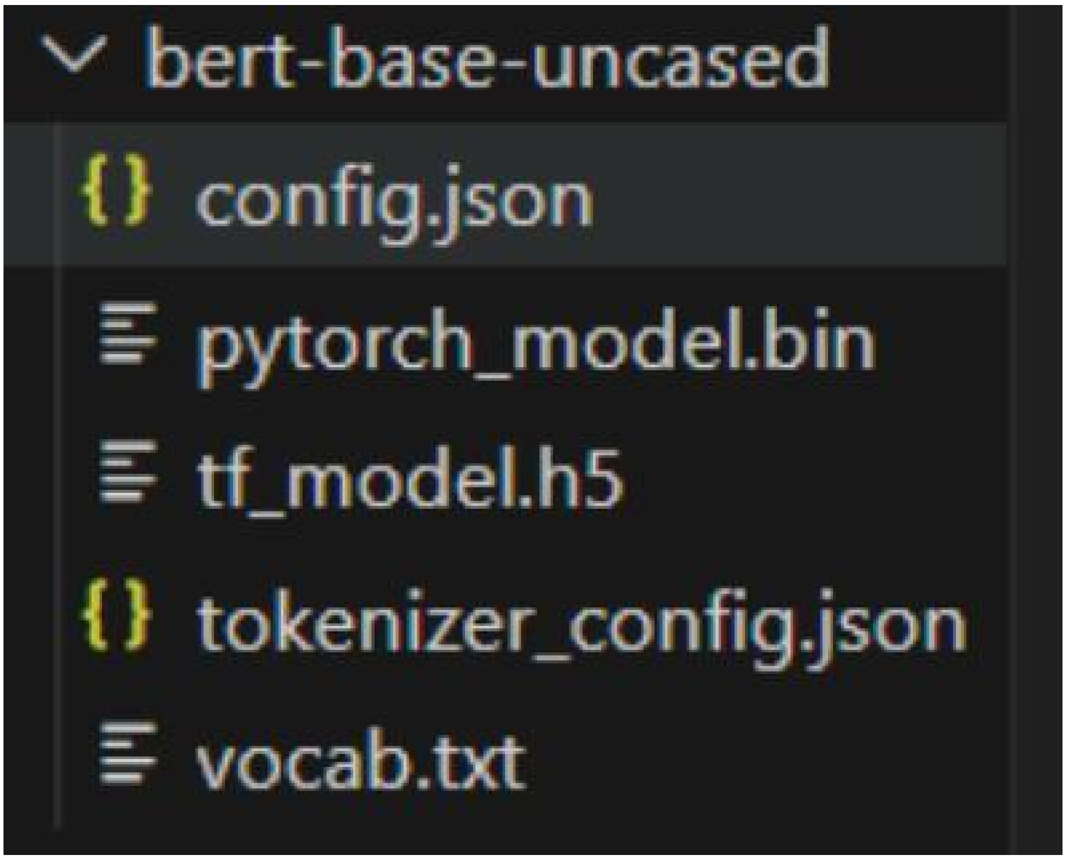
输出结果



```
Epoch [5/10], Train Loss: 0.4114, Valid Loss: [illegible], Valid Acc: [illegible]
Epoch [6/10], Train Loss: 0.3312, Valid Loss: 0.9382, Valid Acc: 0.6438
Epoch [7/10], Train Loss: 0.2738, Valid Loss: 0.9663, Valid Acc: 0.6312
Epoch [8/10], Train Loss: 0.2285, Valid Loss: 1.0023, Valid Acc: 0.6325
Epoch [9/10], Train Loss: 0.2085, Valid Loss: 1.0131, Valid Acc: 0.6312
Epoch [10/10], Train Loss: 0.1937, Valid Loss: 1.0165, Valid Acc: 0.6338
Some weights of the model checkpoint at bert-base-uncased were not used w
```

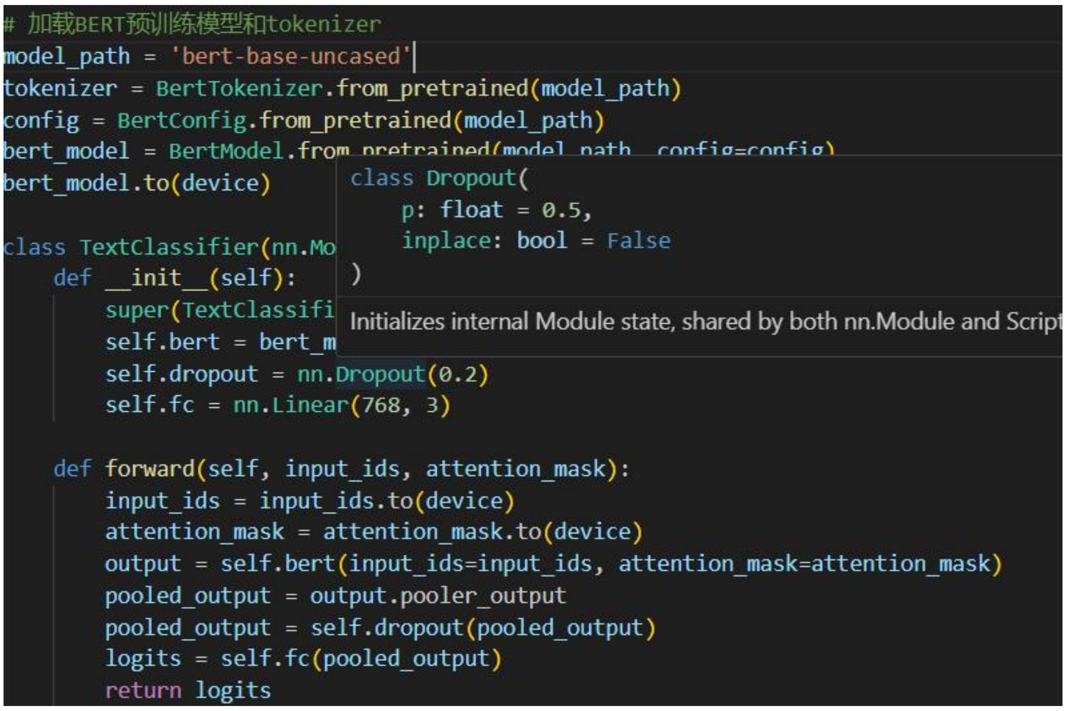
文本分类：

我使用了 bert-base-uncased 预训练模型，这个模型在我上次的实验里就想用了但是出了一些没解决的问题最后没用成功，这次把它用上了。不过在加载的时候还是加载不了，于是只能手动去官网把文件给下了下来。亮点的话在于它是个大模型嘛，然后泛化能力比较强，也支持除了英语以为的其他语言（像法语、德语之类）的。



```
bert-base-uncased
  config.json
  pytorch_model.bin
  tf_model.h5
  tokenizer_config.json
  vocab.txt
```

加载预训练模型

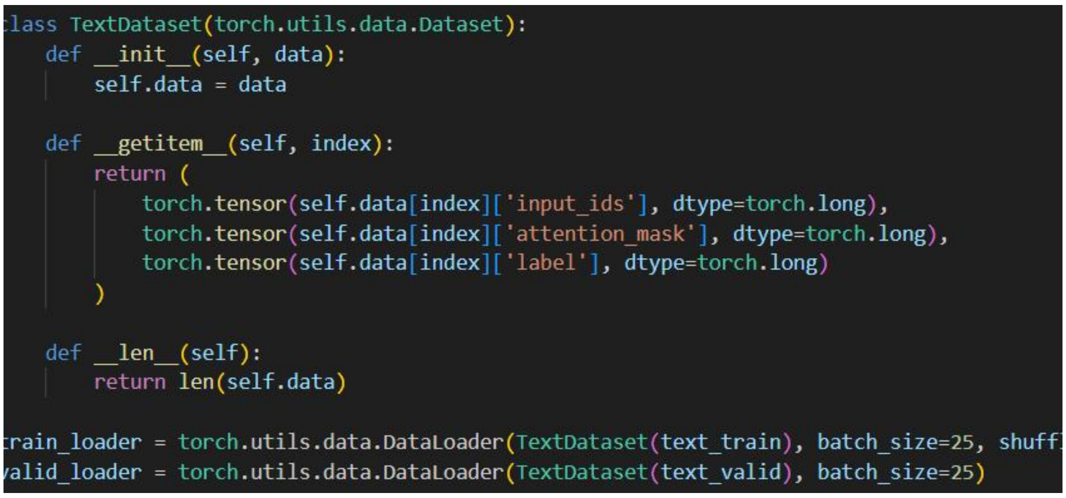


```
# 加载BERT预训练模型和tokenizer
model_path = 'bert-base-uncased'
tokenizer = BertTokenizer.from_pretrained(model_path)
config = BertConfig.from_pretrained(model_path)
bert_model = BertModel.from_pretrained(model_path, config=config)
bert_model.to(device)

class TextClassifier(nn.Mo
    def __init__(self):
        super(TextClassifi
        self.bert = bert_m
        self.dropout = nn.Dropout(0.2)
        self.fc = nn.Linear(768, 3)

    def forward(self, input_ids, attention_mask):
        input_ids = input_ids.to(device)
        attention_mask = attention_mask.to(device)
        output = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        pooled_output = output.pooler_output
        pooled_output = self.dropout(pooled_output)
        logits = self.fc(pooled_output)
        return logits
```

```
class Dropout(
    p: float = 0.5,
    inplace: bool = False
)
```

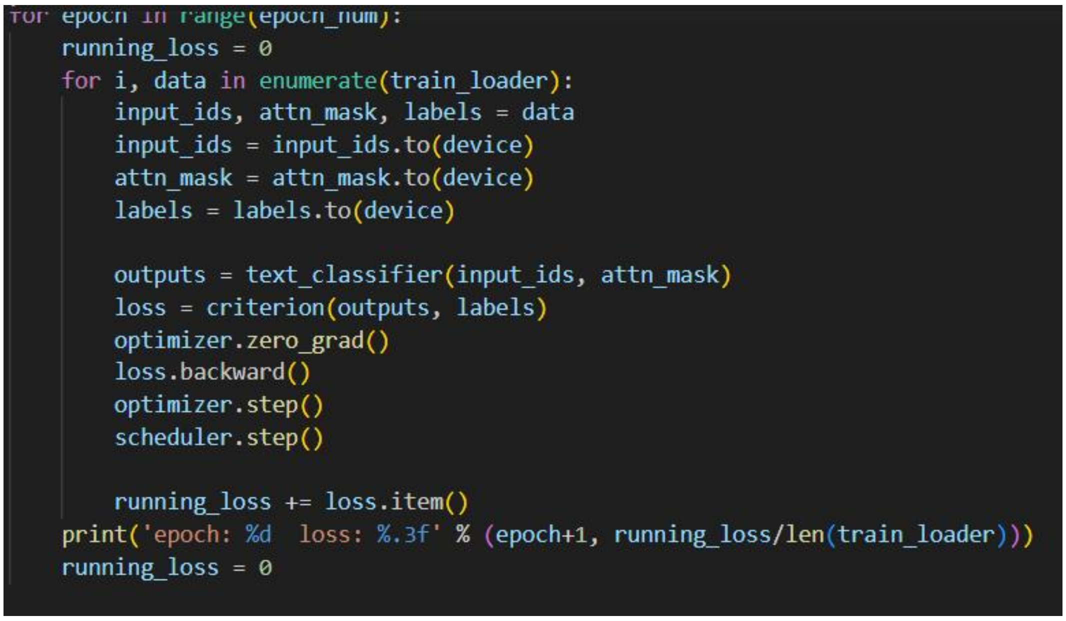
Initializes internal Module state, shared by both nn.Module and Script

进行文本分类



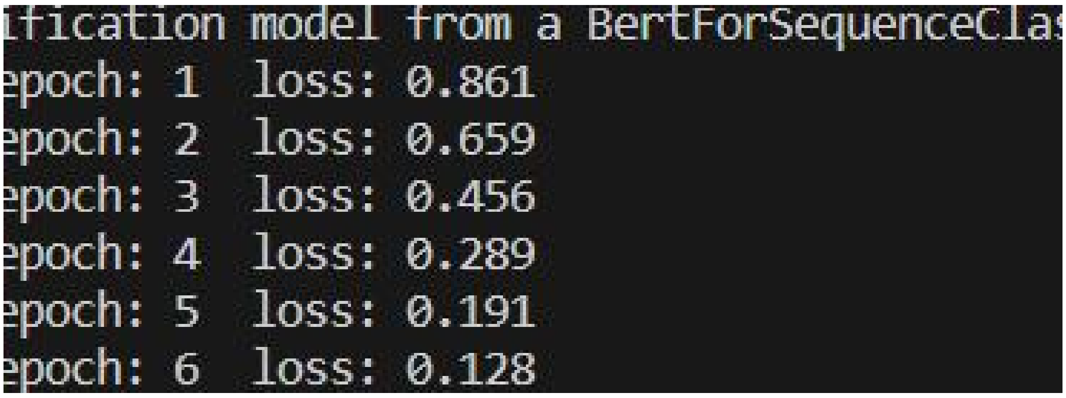
```
lass TextDataset(torch.utils.data.Dataset):
    def __init__(self, data):
        self.data = data

    def __getitem__(self, index):
        return (
            torch.tensor(self.data[index]['input_ids'], dtype=torch.long),
            torch.tensor(self.data[index]['attention_mask'], dtype=torch.long),
            torch.tensor(self.data[index]['label'], dtype=torch.long)
        )

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

rain_loader = torch.utils.data.DataLoader(TextDataset(text_train), batch_size=25, shuff
alid_loader = torch.utils.data.DataLoader(TextDataset(text_valid), batch_size=25)
```

测试结果



```
for epoch in range(epoch_num):
    running_loss = 0
    for i, data in enumerate(train_loader):
        input_ids, attn_mask, labels = data
        input_ids = input_ids.to(device)
        attn_mask = attn_mask.to(device)
        labels = labels.to(device)

        outputs = text_classifier(input_ids, attn_mask)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        scheduler.step()

        running_loss += loss.item()
    print('epoch: %d  loss: %.3f' % (epoch+1, running_loss/len(train_loader)))
    running_loss = 0
```

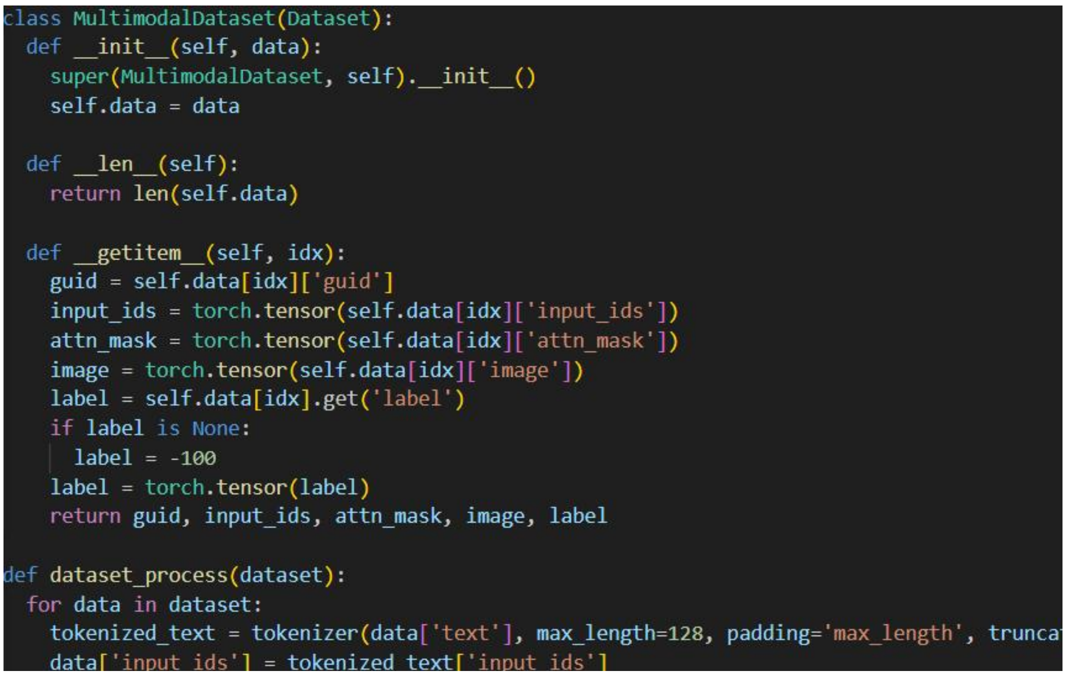
在验证集上的结果



```
ification model from a BertForSequenceClas
epoch: 1  loss: 0.861
epoch: 2  loss: 0.659
epoch: 3  loss: 0.456
epoch: 4  loss: 0.289
epoch: 5  loss: 0.191
epoch: 6  loss: 0.128
```

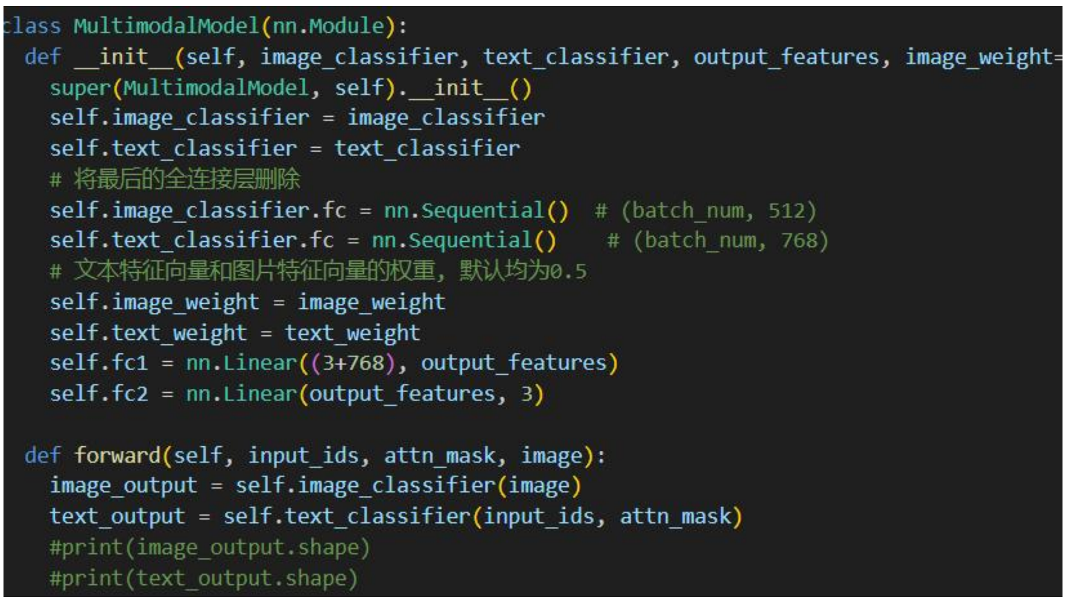
多模态融合：

数据集处理



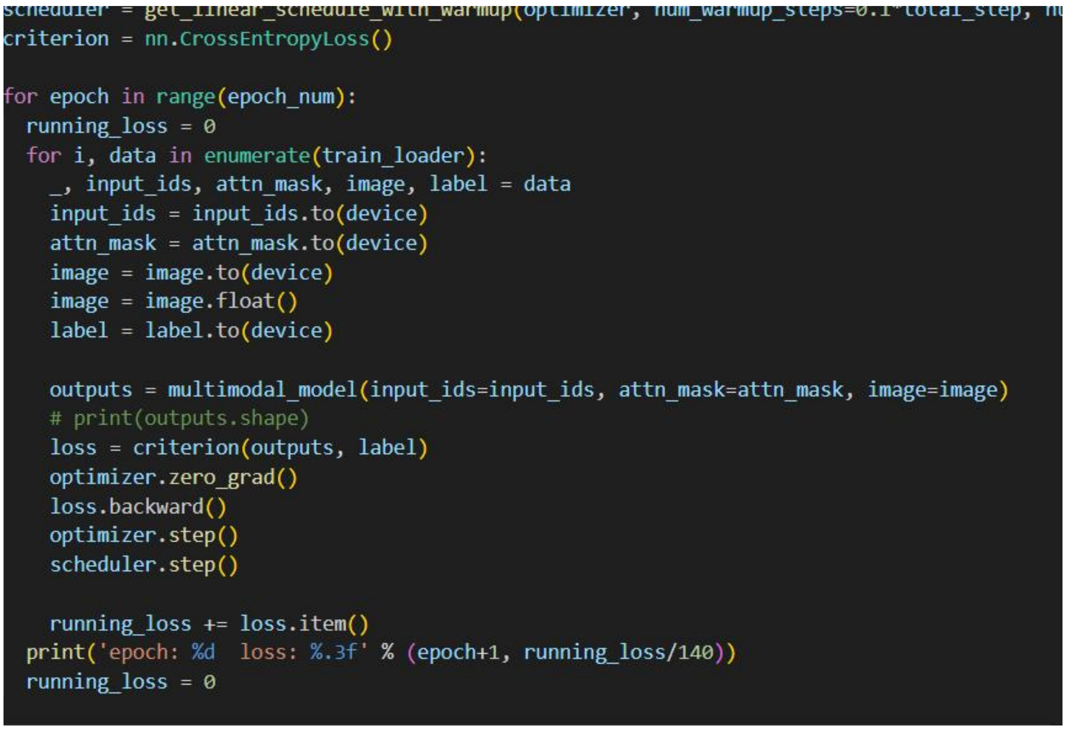
```python
class MultimodalDataset(Dataset):
  def __init__(self, data):
    super(MultimodalDataset, self).__init__()
    self.data = data

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

  def __getitem__(self, idx):
    guid = self.data[idx]['guid']
    input_ids = torch.tensor(self.data[idx]['input_ids'])
    attn_mask = torch.tensor(self.data[idx]['attn_mask'])
    image = torch.tensor(self.data[idx]['image'])
    label = self.data[idx].get('label')
    if label is None:
      label = -100
    label = torch.tensor(label)
    return guid, input_ids, attn_mask, image, label

def dataset_process(dataset):
  for data in dataset:
    tokenized_text = tokenizer(data['text'], max_length=128, padding='max_length', trunca
    data['input_ids'] = tokenized_text['input_ids']
```

模型定义



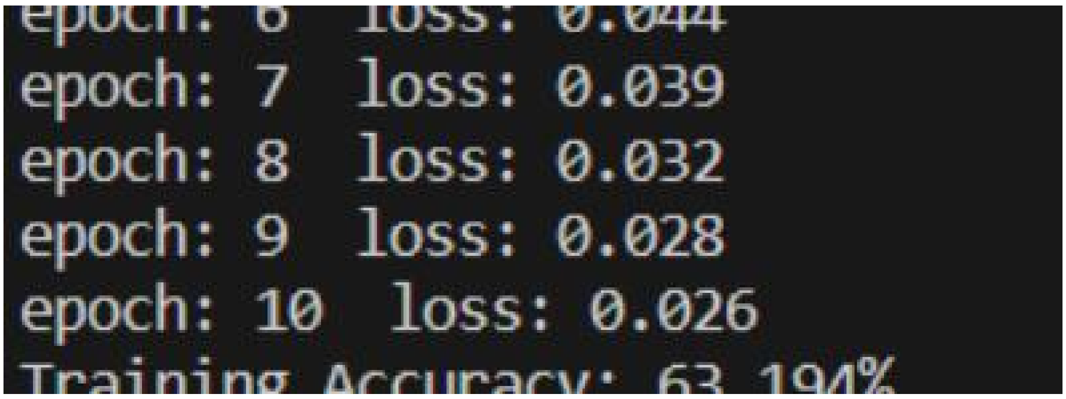
```python
class MultimodalModel(nn.Module):
  def __init__(self, image_classifier, text_classifier, output_features, image_weight=
    super(MultimodalModel, self).__init__()
    self.image_classifier = image_classifier
    self.text_classifier = text_classifier
    # 将最后的全连接层删除
    self.image_classifier.fc = nn.Sequential()  # (batch_num, 512)
    self.text_classifier.fc = nn.Sequential()    # (batch_num, 768)
    # 文本特征向量和图片特征向量的权重, 默认均为0.5
    self.image_weight = image_weight
    self.text_weight = text_weight
    self.fc1 = nn.Linear((3+768), output_features)
    self.fc2 = nn.Linear(output_features, 3)

  def forward(self, input_ids, attn_mask, image):
    image_output = self.image_classifier(image)
    text_output = self.text_classifier(input_ids, attn_mask)
    #print(image_output.shape)
    #print(text_output.shape)
```

测试结果



```
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0.1*total_step, nu
criterion = nn.CrossEntropyLoss()

for epoch in range(epoch_num):
  running_loss = 0
  for i, data in enumerate(train_loader):
    _, input_ids, attn_mask, image, label = data
    input_ids = input_ids.to(device)
    attn_mask = attn_mask.to(device)
    image = image.to(device)
    image = image.float()
    label = label.to(device)

    outputs = multimodal_model(input_ids=input_ids, attn_mask=attn_mask, image=image)
    # print(outputs.shape)
    loss = criterion(outputs, label)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    scheduler.step()

    running_loss += loss.item()
  print('epoch: %d  loss: %.3f' % (epoch+1, running_loss/140))
  running_loss = 0
```

在验证集上的结果



```
epoch: 6  loss: 0.044
epoch: 7  loss: 0.039
epoch: 8  loss: 0.032
epoch: 9  loss: 0.028
epoch: 10  loss: 0.026
Training Accuracy: 63.194%
```