

## 当代人工智能实验五-多模态情感分析

--10215501435 杨茜雅

### 实验任务：

- 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。

示例：



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

### 实验数据集：

- 匿名数据集（实验五数据.zip）

- data 文件夹：包括所有的训练文本和图片，每个文件按照唯一的 guid 命名。



- train.txt: 数据的 guid 和对应的情感标签。

```
guid,tag
4597,negative
26,neutral
4383,negative
212,positive
```

- test\_without\_label.txt: 数据的 guid 和空的情感标签。

```
guid,tag
8,null
1576,null
2320,null
4912,null
```

### 实验要求：

- 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。

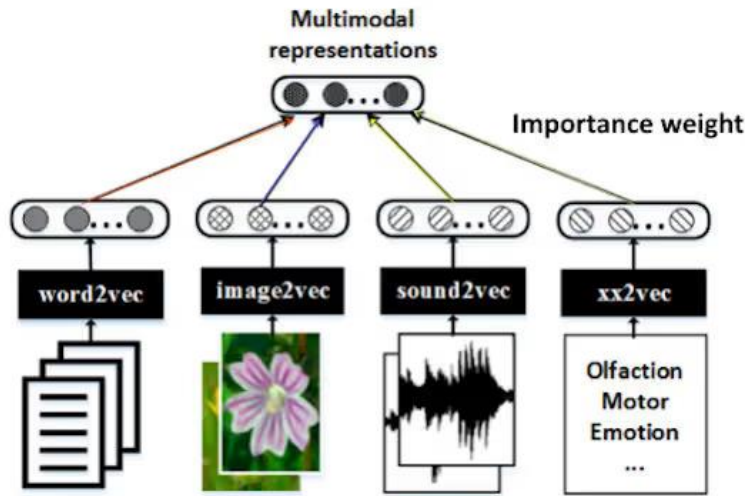
### 代码仓库地址：

<https://github.com/Lily127Yang/Contemporary-AI---Project-5>

### 实验思路：

本次实验要求实现的是一个多模态融合模型。多模态模型，顾名思义，是指能够处理多种类型数据（如图像、文本、声音等）的深度学习模型。与传统的深度学习模型相比，多模态大模型在训练过程中能够同时

处理多种类型的数据，从而提高模型的泛化能力和鲁棒性。这使得多模态大模型在自然语言处理、计算机视觉、语音识别等领域具有广泛的应用前景。



我首先对数据集进行预处理，把所有的文字数据整理到一个文件中再把图像文字转化成可以训练的数据。接着对数据集进行探索，查看各类标签的占比并且划分训练集和验证集，得出 neutral 标签占比较少的结论，后续进行探索改进。接着开始搭建模型。

### 模型结构

模型以 BERT 和 ResNet-152 为基础，分别处理文本和图像输入。接着，通过自注意力机制实现多模态信息的融合。让我们逐步解析这个模型的关键部分。除了多模态处理外，该模型还支持单一模态的处理，即只有文本输入或只有图像输入。在这两种情况下，模型分别提取单一模态的特征，并通过分类器得到输出。

### 特征提取

模型首先通过 BERT 处理文本输入，获取文本的隐藏状态。这些隐藏状态包含了文本的语义信息，为后续的多模态融合做准备。对于图像输入，模型使用了预训练的 ResNet-152 模型提取图像的特征。通过对图像特征进行池化和线性变换，得到图像的隐藏状态。

### 特征融合

接下来，模型将文本和图像的隐藏状态进行拼接，构成共同的特征表示。通过设置 attention\_mask，模型实现了对文本中 padding 部分的处理，并使用 self-attention 机制进行多模态融合。

### 分类

最后，模型分别提取多模态融合后的图像和文本特征，并通过线性变换进行分类，得到最终的输出。

不仅如此，我还将标签为 neutral 的数据条数做复制，再进行模型的训练和拟合。也进行了消融实验，分别对于仅文本和仅图像的准确率进行了探索，最后做出本次实验的总结。

## 1、数据预处理

### 1.1 观察数据集

train.txt 共有 4000 条数据。为了方便模型预测之后计算准确率，将情绪标签按分类映射为 0, 1, 2

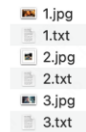
```
emo_tag = {"neutral": 0, "negative": 1, "positive": 2}
```

并且查看每种情绪的占比如下。

```
1 : 0.2983245811452863
0 : 0.10477619404851213
2 : 0.5968992248062015
```

data 文件夹中有若干对图片和 txt 文件，文件名相同的文件一一对应。两个 txt 文件分别是有情感标签的数据和需要被预测的无情感标签的数据，文件之间通过唯一的 guid 实现一一对应。

- data 文件夹：包括所有的训练文本和图片，每个文件按照唯一的 guid 命名。
- train.txt: 数据的 guid 和对应的情感标签。
- test\_without\_label.txt: 数据的 guid 和空的情感标签。



```
guid,tag
4597,negative
26,neutral
4383,negative
212,positive
```

```
guid,tag
8,null
1576,null
2320,null
4912,null
```

data.txt 中的文件是一个一个句子，语句中存在一些对情感分类无用的字符串，例如：@xxx、#、http...等等，所以不能使用 csv 格式保存。为了把所有的文字数据都整理到一个文件中，我打算使用以 tab 为分隔符的 tsv 文件格式保存。

读取文件并生成 tsv 的方法在 generate\_data.py 文件夹下。下面说明以 80% 的训练集为例，20% 的测试集和验证集做法一致。

生成训练集 (train.tsv)：写入表头后，遍历前 3200 行的训练数据（大约是 80% 的数据，用于 8:2 的训练/验证集划分）。对于每一行，它分离出标识符 (guid) 和标签 (tag)，读取对应的文本文件，并将文本内容和标签写入新的 tsv 文件。

```
# 生成训练集数据
with open(os.path.join(data_dir, 'train.tsv'), 'w', encoding='utf-8') as f:
    f.write('index\ttag\tguid\tdescription\n')
    count = 0
    # 希望按照8: 2的比例划分训练集和验证集
    for i in range(1, 3200):
        count += 1
        # 去除首尾空格
        line = lines[i].strip()
        guid, tag = line.split(',')
        file1 = './data/' + guid + '.txt'
        with open(file1, 'r', encoding='gb18030') as file_text:
            lines_text = file_text.readlines()
            text = ''
            for index in range(len(lines_text)):
                text += lines_text[index].strip() + '\t'
            polarity = labelMap[tag]
            imgid = guid
            label = str(int(polarity))
            f.write('%d\t%s\t%s\t%s\n' % (count, label, imgid, text))
```

训练集和验证集以 8: 2 的比例被划分，分别保存在 train.tsv, valid.tsv 下，可以看到，每一行的第一列是 index，第二列是 tag 转换成的数字，第三列是 guid，第四列是文字描述。

```
processed_data > cat train_legacy.tsv
1 index tag guid description
2 1 0 4597 RT @AmitSwami77: The conspirators have an evil eye & are now set to physically attack Asaram Bapu Ji! #WeDemandSafety4BapuJi http://t
3 2 1 26 Waxing trills, Chickadees calling "here sweetie", enthusiastic athletes, blue sky & snow at #ualbertafarm #UALberta
4 3 0 4383 @NYSE is looking a little despondent today...??? http://t.co/o5xikyJgT7
5 4 2 212 FERVENT | S,M,L | 140k free PLASTIC CLIP, keychain rubber AND sticker 085725737197 / 28ae36f3
6 5 2 2626 Nice day chilling in the park yesterday relieved my mood for a short while. #friends #summer #outside #depression
7 6 1 3042 Ford : F-350 Lariat 6.4L 2008 Lariat Heated Leather Rear Camera 2008 ford f 250 diesel 4 x...
8 7 2 4713 RT @MOVIEMEMORIES: Furious 7 http://t.co/CEPxf3Q1Y
9 8 0 2073 @MattSmith1230 @ProFlowers The flowers look like a dejected King Triton:
10 9 2 2020 #廢墟 #abandoned #写真撮ってると人と繋がりたい #写真好きな人と繋がりたい
11 10 0 2688 RT @Pablothemako: UPDATE!Navy discarded illegal fishing after boarding chinese vessels in #Chile's Excl Econ Zone htt...
12 11 2 4956 Arden B Womens Blue Solid Dress Sz M Medium Short Sleeve Modal Blend Above Knee http://t.co/hHbd7EnsEa http://t.co/9Dr58voDRP
13 12 2 3143 RT @dagtotdag: De eerste kilometers in open water met @Kokkie70 @G3sjaak @SwintoFC078
14 13 2 3624 DVF Diane von Furstenberg Naples Ankle Soft Canvas Blossom Black Tuxedo Pants 4
15 14 0 202 RT @SimpsonsQOTD: "Oh, don't you worry, most of you will never fall in love and marry out of fear of dying alone."
16 15 2 4486 RT @LEFLAH_KOIZUMI: Dizzy Sunfistのあやべが展示会に遊びに来てくれました??わざわざありがとう??今日は町田SDRでLIVEです?? #LEFLAH #レフラー
17 16 2 4895 Just an intense game of tic tac toe?? #tinderproblems @findrProbs http://t.co/9zAcpn5eZ2
18 17 2 2855 RT @Football oranje: Some Fenerbahce fans getting a bid too excited about Van Persie's arrival in Istanbul (via @ridvanaksu)
```

查看 train.tsv, valid.tsv 中的 tag 种类, 以及每个种类所占的个数

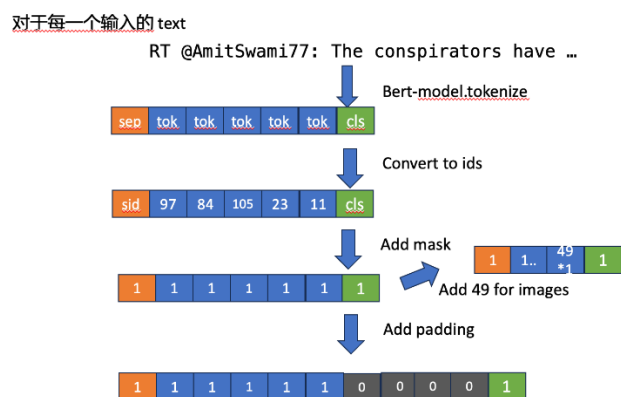
tags/file	train. tsv	valid. tsv
negative	941	252
neutral	332	87
positive	1926	462

发现 neutral 标签所占的比例略少, 如果真正在数据集上进行训练的话, 可能会更容易造成过拟合。对于样本较为缺失的标签, 一般的做法是在数据集中重复这些较少的数据较多次。尽管听上去没有什么理论支持, 但是在实际操作时, 这种方法是真实有效的。在许多 kaggle 竞赛上都得到了证实。

后续的计划是将标签为 neutral 的数据条数做复制, 再进行模型的训练和拟合, 观察对比效果。

## 1.2 把文字图像转化为可以训练的数据

我们读取刚刚得到的 tsv 文件, 并且对其中的 text 进行 tokenize, convert\_to\_ids, add\_padding 等一系列操作, 最后再加上一条数据, 这条数据会留 49 个空格给图像 (图像经过处理后长度都为 49)



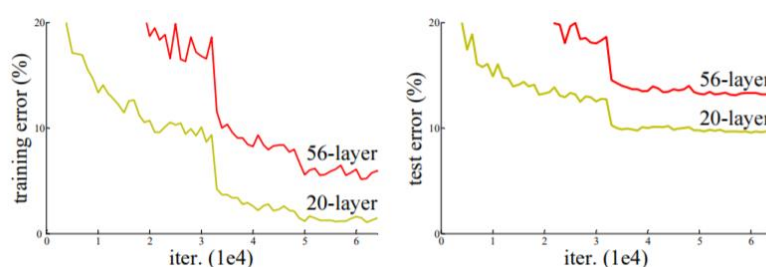
对于图像数据, 我们统一将其先转换为 224\*224 的大小, 方便模型进行训练。

```
features = []
# 默认是 224 * 224
transform = transforms.Compose([
    transforms.RandomCrop(crop_size, pad_if_needed=True),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406),
                          (0.229, 0.224, 0.225))])
```

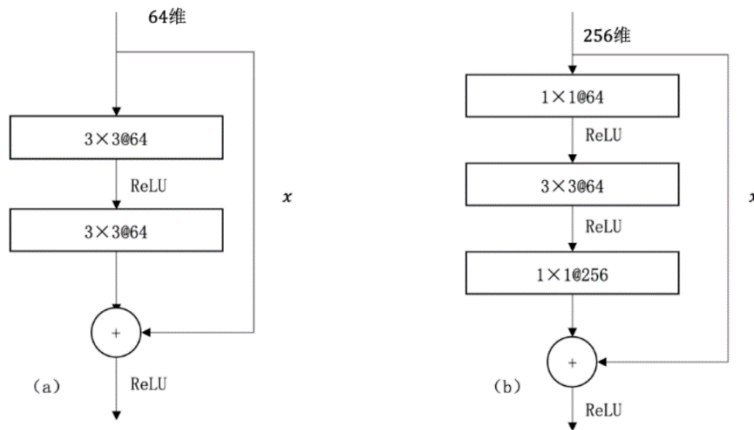
对图像使用 RandomCrop 方法统一尺寸到 crop\_size 以后, 进行随机水平翻转, 转换成 torch.tensor 数据格式, 并且进行归一化。

## 2、模型构建 BERT+RESNET-152

我们在文字部分选取的预训练模型是 bert-base-uncased, 图片部分选取的预训练模型是 resnet-152, 因为在实验三中我所得到的结论是 resnet 是一个准确率很高并且参数相对较小、不容易出现过拟合的模型。残差神经网络的主要贡献是发现了“退化现象(Degradation)”



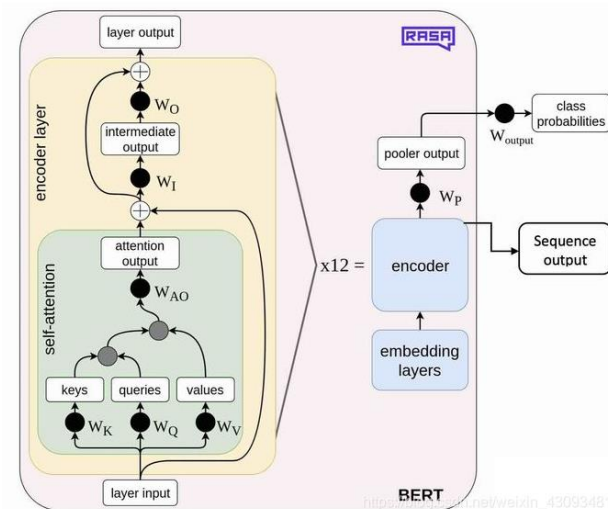
它针对退化现象发明了“快捷连接 (Shortcutconnection)”，极大的消除了深度过大的神经网络训练困难问题。神经网络的“深度”首次 突破了 100 层、最大的神经网络甚至超过了 1000 层。非线性转换极大的提高了数据分类能力，但是， 随着网络的深度不断的加大，我们在非线性转换方面已经走的太远，竟然无法实现线性转换。显然，在 神经网络中增加线性转换分支成为很好的选择，于是，ResNet 团队在 ResNet 模块中增加了快捷连接分支，在线性转换和非线性转换之间寻求一个平衡。这不仅提高了网络的性能，也为深度学习的未来发展提供了新的思路 and 方向。ResNet 的成功应用于多种计算机视觉任务，如图像分类、物体检测和图像分割，证明了其强大的功能和通用性。



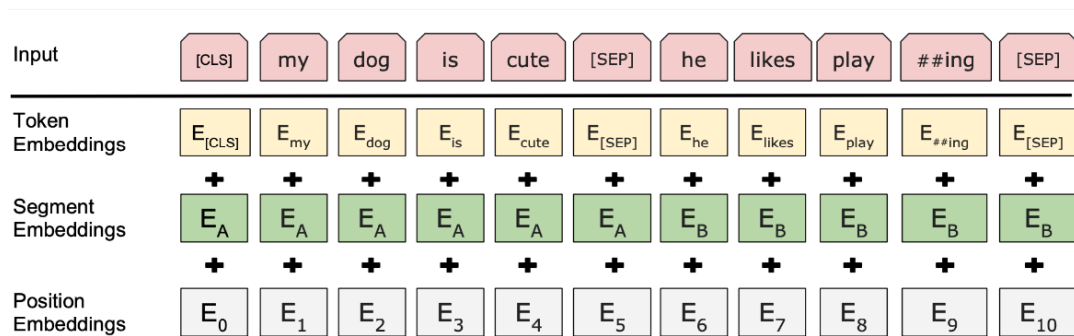
## 2.1 Bert 模型搭建

分为以下步骤：

把输入先做 embedding，并且将 token embeddings, segmentation embeddings, position embeddings 进行加和。将三种不同的 embedding 进行加和与单独的 embedding 输入相比，使用了不同的角度和方式进行了编码，可以让模型学到更多的特征信息。



- Token Embeddings: 用于将输入的单词序列转换为向量表示，作用是将每个输入单词都表示为一个向量，包含了该单词的语义信息。
- Segment Embeddings: 用于区分两个句子中相同位置的单词，作用是将不同句子中相同位置的单词进行区分，避免不同句子中的单词产生干扰。
- Position Embeddings: 用于表示单词在输入序列中的位置，作用是将输入序列中单词的位置信息编码成向量，以便模型能够理解单词在序列中的顺序关系。



之后我们添加层归一化和 dropout 防止过拟合与防止梯度爆炸或者消失，并且将输入的数据进行应用多头注意力机制。

分别定义 query、key、value

```
self.query = nn.Linear(config.hidden_size, self.all_head_size)
self.key = nn.Linear(config.hidden_size, self.all_head_size)
self.value = nn.Linear(config.hidden_size, self.all_head_size)
```

将 它们 进行multihead 拆分，拆成 12 个头，拆分后保存在 key\_layer, query\_layer, value\_layer 中，维度为 (1,12,128,64)

```
new_x_shape = x.size()[:-1] + (self.num_attention_heads, self.attention_head_size)
x = x.view(*new_x_shape)
return x.permute(0, 2, 1, 3)
```

attention\_scores 矩阵是由 Q 和 K 矩阵相乘得到的，用于计算注意力分数，注意力权重经过 Softmax 函数和 Dropout 操作后，我们可以得到每个单词对其他单词的注意力权重，并计算上下文向量。最终返回的是上下文向量，将每个注意力头的向量拼接在一起，形成一个更大的向量维度。

```
query_layer = self.transpose_for_scores(mixed_query_layer)
key_layer = self.transpose_for_scores(mixed_key_layer)
value_layer = self.transpose_for_scores(mixed_value_layer)
# Take the dot product between "query" and "key" to get the raw attention scores.
attention_scores = torch.matmul(query_layer, key_layer.transpose(-1, -2))
attention_scores = attention_scores / math.sqrt(self.attention_head_size)
# Apply the attention mask is (precomputed for all layers in BertModel forward() function)
attention_scores = attention_scores + s2_attention_mask
# Normalize the attention scores to probabilities.
attention_probs = nn.Softmax(dim=-1)(attention_scores)
attention_probs = self.dropout(attention_probs)
context_layer = torch.matmul(attention_probs, value_layer)
context_layer = context_layer.permute(0, 2, 1, 3).contiguous()
new_context_layer_shape = context_layer.size()[:-2] + (self.all_head_size,)
context_layer = context_layer.view(*new_context_layer_shape)
```

intermediate 层，也就是 Feedforward 层，用于对输入的隐藏状态向量进行非线性变换，提取更高层次的特征表示

```
class BertIntermediate(nn.Module):
    def __init__(self, config):
        super(BertIntermediate, self).__init__()
        self.dense = nn.Linear(config.hidden_size, config.intermediate_size)
        self.intermediate_act_fn = ACT2FN[config.hidden_act] \
            if isinstance(config.hidden_act, str) else config.hidden_act

    def forward(self, hidden_states):
        hidden_states = self.dense(hidden_states)
        hidden_states = self.intermediate_act_fn(hidden_states)
        return hidden_states
```

根据以上的代码，可以构建一个 bert encoder. bert encoder 层一层包括 bert input, bert layer 和 bert output, 其中 bert layer 层又包括 self-attention 层和 intermediate 和 output 层。



```
class BertLayer(nn.Module):
    def __init__(self, config):
        super(BertLayer, self).__init__()
        self.attention = BertAttention(config)
        self.intermediate = BertIntermediate(config)
        self.output = BertOutput(config)

    def forward(self, hidden_states, attention_mask):
        attention_output = self.attention(hidden_states, attention_mask)
        intermediate_output = self.intermediate(attention_output)
        layer_output = self.output(intermediate_output, attention_output)
        return layer_output
```

```
class BertEncoder(nn.Module):
    def __init__(self, config):
        super(BertEncoder, self).__init__()
        layer = BertLayer(config)
        self.layer = nn.ModuleList([copy.deepcopy(layer) for _ in range(config.num_hidden_layers)])

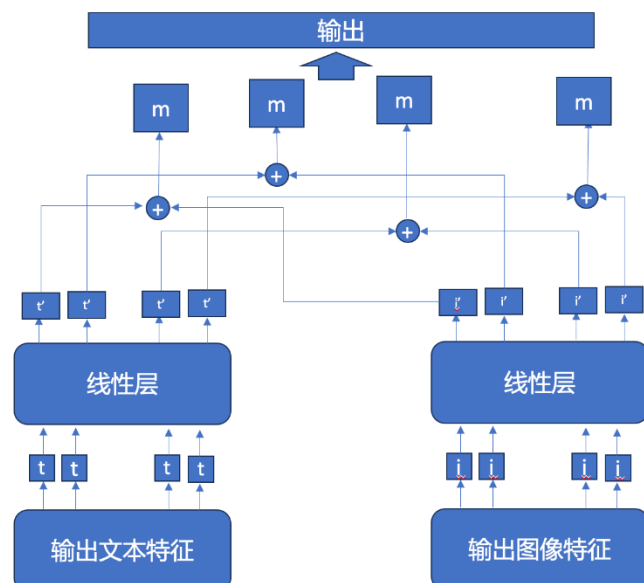
    def forward(self, hidden_states, attention_mask, output_all_encoded_layers=True):
        all_encoder_layers = []
        for layer_module in self.layer:
            hidden_states = layer_module(hidden_states, attention_mask)
            if output_all_encoded_layers:
                all_encoder_layers.append(hidden_states)
        if not output_all_encoded_layers:
            all_encoder_layers.append(hidden_states)
        return all_encoder_layers
```

定义 pooler 层，作用是将模型的最后一层输出向量转换为一个固定长度的向量，用于表示整个输入序列的语义特征，可以用于下游任务的处理，如分类、序列标注等。

```
class BertPooler(nn.Module):
    def __init__(self, config):
        super(BertPooler, self).__init__()
        self.dense = nn.Linear(config.hidden_size, config.hidden_size)
        self.activation = nn.Tanh()

    def forward(self, hidden_states):
        # We "pool" the model by simply taking the hidden state corresponding
        # to the first token.
        first_token_tensor = hidden_states[:, 0]
        pooled_output = self.dense(first_token_tensor)
        pooled_output = self.activation(pooled_output)
        return pooled_output
```

然后我们还需要接受多模态输入。我们可以将 文本+图像的输入也做一个 cross attention。Transformer 解码从完整的输入序列开始，但解码序列为空。交叉注意力将信息从输入序列引入到解码器的各层，以便它可以预测下一个输出序列标记。然后解码器将令牌添加到输出序列中，并重复此自回归过程，直到生成 EOS 令牌。



### 3、消融实验结果

分别只输入文本或图像数据，观察多模态融合模型在验证集会获得怎样的表现

输入 图像+文本，仅文本，仅图像 的准确率和 f1 值分别如下：

	图像+文本	仅文本	仅图像
Accuracy	71.34	64.67	59.8
F1 score	70.62	63.29	49.89

可以发现，我们的模型在多模态问题上的表现较好，判断正确率在 0.7 左右。并且远高于两个单模态的模型输出结果。推测文本数据表现好于图像数据的可能性是，一个图像可以表示更多范围的情绪，而文本似乎略少一些。

#### 3.1 对数据集的标签进行探索，训练

前文提到，neutral 标签所占的比例略少，如果真正在数据集上进行训练的话，可能会更容易造成过拟合。

对于样本较为缺失的标签，一般的做法是在数据集中重复这些较少的数据较多次。

定义了 fix\_overfit 方法，用于给训练集和验证集进行缺失标签的数据填补。把训练数据和验证数据里面 label 为 1 的数据先添加在末尾，然后将里面的行随机排序生成新文件。

修改以后，训练数据集中，label 为 0,1,2 的数据分别为 941 995 1926 条。验证数据集中，label 为 0,1,2 的数据分别为 252 260 461 条。相比于以前 2:1:3 的数据分布比，现在 1:1:2 的数据分布比似乎更加健壮。同时，发现数据集有多条日语、俄语数据，我将其都翻译成了英文，来提升模型的准确率。

进行训练与验证，得到的结果如下：

	图像+文本	仅文本	仅图像
Accuracy	64.74	55.6	47.99
F1 score	63.79	54.54	33.18

发现得到的准确率比原数据集更低，这也比较合理，因为原本的数据集，与其说是一个三分类问题，不如更加靠近一个二分类问题。假设模型有先验知识，知道各种标签所对应的数据比例是 2:1:3，那么准确率的期望值就是  $0.5 \times 0.5 + 0.33 \times 0.33 + 0.16 \times 0.16 = 0.375$ 。而如果知道各种数据的比例是 1:1:2，准确率的期望值是  $0.25 \times 0.25 + 0.25 \times 0.25 + 0.5 \times 0.5 = 0.365$ ，会略低一些。

但是双模态和仅文本的表现稍微还符合这样的比例，但是仅图像就太低了。F1 score 到了 33.18，不如随机猜测。因此判断这个模型的效果不好。

我们来查看预测生成的数据，在 predict 文件夹下。旧数据集和新数据集所对应的预测结果分别是 text\_without\_label\_legacy.txt 和 text\_without\_label.txt。发现前者中，标签为 positive 的较多，并且三种标签分布得较为均匀。而后者中，标签为 neutral 的过多，和原始数据的分布不太吻合，舍弃。

所以目前来讲，在这个例子中，使用原数据集表现更好。

### 4、总结

在本次实验中，我用多模态融合模型对具体数据进行了建模，使用 bert 模型 和 resne-152 模型对于文本和图像分别进行了变换并且进行向量拼接，最后喂进模型训练，取得了较好的表现。同时，我对于数据集进行了探索，发现如果重复标签为 neutral 的数据过多，无论是训练结果还是预测结果都不太理想。我们也进行了消融实验，分别对于仅文本和仅图像的准确率进行了探索，对于多模态模型的表现有了更深刻的了解。



## 5、遇到的问题及解决措施

### 5.1 读入图片描述时，出现无法解码的问题

```
main.py:32: DeprecationWarning: ANTIALIAS is deprecated and will be removed in Pillow 10 (2023-07-01). Use Resampling.LANCZOS instead.
  img = img.resize((224,224),Image.ANTIALIAS)
Traceback (most recent call last):
  File "main.py", line 35, in <module>
    des = f.read()
UnicodeDecodeError: 'gbk' codec can't decode byte 0xac in position 75: illegal multibyte sequence
```

解决方案：根据代码给出的提示信息，可以看出错误的意思是：Unicode 的解码（Decode）出现错误了，以 gbk 编码的方式去解码（该字符串变成 Unicode），但是此处通过 gbk 的方式，却无法解码（can't decode）。“illegal multibyte sequence”的意思是非法的多字节序列，也就是说无法解码了。出现这样的错误，可能是要处理的字符串本身不是 gbk 编码，却是以 gbk 编码去解码。比如，字符串本身是 utf-8 的，但用 gbk 去解码，必然出错。故采用更宽范围的编码进行解码，`with open('./data/' + str(guid) + '.txt', encoding='gb18030') as f:`，问题即可解决。

### 5.2 运行模型时，出现以下问题

```
Traceback (most recent call last):
  File "main.py", line 152, in <module>
    main()
  File "main.py", line 137, in main
    model = simpleModel.to(device)
  File "/root/miniconda3/lib/python3.8/site-packages/torch/nn/modules/module.py", line 907, in to
    return self._apply(convert)
AttributeError: 'torch.device' object has no attribute '_apply'
```

解决方案：该问题出现的原因是构建的模型类没有实例化就使用了，进行实例化之后问题解决

### 5.3 使用torch.tensor 转换时，忘记之前已经通过 torch.transformer 转换成了一系列 torch.tensor 数据类型

```
Traceback (most recent call last):
  File "/tmp/pycharm_project_417/run.py", line 92, in <module>
    main()
  File "/tmp/pycharm_project_417/run.py", line 78, in main
    train(args, train_examples, num_train_steps, label_list, optimizer, scheduler, model, encoder, global_step, tokenizer)
  File "/tmp/pycharm_project_417/processors/util.py", line 169, in train
    all_img_feats = torch.tensor([f.img_feat for f in train_features])
ValueError: only one element tensors can be converted to Python scalars
```

解决方案：list 列表中每一个元素都是一个 tensor 类型的数据，所以应该使用 `torch.stack` 方法将这些 list 里面的元素拼接起来变成一个 `torch.tensor` 类型的数据。

## 六、这样设计模型的原因和亮点

### 原因：

本项目需要分别提取文本的特征以及图片的特征，将二者特征融合之后再进情绪的分类。对于文本进行情感分析，就是需要得到整个文本的上下文的综合信息，想到 Bert 模型在处理文本信息时，内部采用自注意力机制，最后对每个字输出一个向量，并且在句子开头添加了一个特殊符号'CLS'，该符号对应的向量中包含了整个句子的综合信息。所以，进行情绪分类的文本特征考虑采用'CLS'的向量信息。对于图片的特征提取，考虑使用在 ImageNet 上预训练过的 Resnet，之所以选择 Resnet 是因为我在实验三中得出过 resnet 是一个准确率很高并且参数相对较小、不容易出现过拟合的模型这一结论。越深的网络提取的特征越抽象，越具有语义信息，Resnet 中有残差结构，该结构使得训练更深的网络时缓解梯度消失或者梯度爆炸的问题。

### 亮点：

进行实验之后，我对数据集进行了探索优化并且重新带入模型进行实验。我认为数据预处理也是“建模”的一部分，在以后的学习或者工作中我们有可能会拿到标签分布不均匀的数据，那我们是否可以对数据进行一些合理的加工而不是一定要使用 raw data 呢？带着这样的想法，我定义了 `fix_overfit` 方法，用于给训练集

和验证集进行缺失标签的数据填补，使数据分布变得更为健壮，但是最后的准确率比原本的模型更低，对此我也做出了自己认为的可能的解释。虽然结果对模型没有实际帮助，但这只证明这样的加工方法不适用于本次实验，不证明不适用于其他的情况，不管怎样，这给我以后处理数据提供了一个新的思路。