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**：2020 级	**专业：新闻系**
指导教师：李翔	**姓名**：陈佳琪	**学号**：10203330403
上机实践名称：多模态情感分析		**上机实践日期**：23.7
上机实践编号：	**组号**：	**上机实践时间**：23.7

Github 地址：https://github.com/10203330403/-23-.git

一、实验过程:

1 数据读入：

首先将带标签的 train.txt 进行处理，将文字标签转换为数字标签存储为 data . c v s 文件。然后处理 data 文件夹中图文多模态数据，首先按照文件名顺序分别读入 jpg 和 txt 文件，然后分开存储图片和文档，其中图片在读入时已变为 r g b 三维的不同大小的矩阵，此时将图片矩阵先统一维度。接下来将 txt 文件中的特殊符号去除，对文档进行向量化，使用 gensim 库进行向量化，最终转换为 numpy 文件存储起来。同时读入 test 文件，处理同上。

2.模型建立：

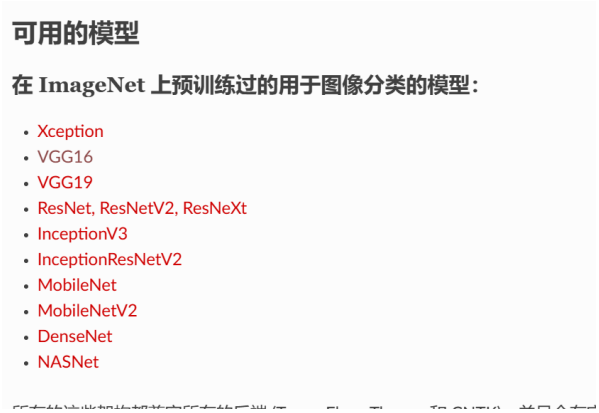
模型训练主要使用 tensorflow，对图片的处理网络使用 VGG，流程如下。

首先分析数据部分：对于文本数据使用向量归一化，而对图片数据使用 vgg 模型进行分析，其中网络构架采用 keras 中的在 ImageNet 上预训练过的用于图像分类的可用模型——VGG16。使用该模型提取特征，为了获得固定大小的 224×224 ConvNet 输入图像，从归一化的训练图像中随机裁剪。vgg 网络特点是深度较深有 16 层，使用 3*3 卷集核，第一个全连接层使用与输入相同尺寸的卷集核取代传统全连接方式以达到降低参数的目的，因此 vgg16 的参数数量相对于其他相似深度的网络更少。

11 weight	11 weight	13 weight	16 weight	16 weight	19 weight
layers	layers	layers	layers	layers	layers
imput (224x224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256

			conv1-256	conv3-256	conv3-256 conv3-256
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
			conv1-512	conv3-512	conv3-512 conv3-512
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512 conv1-512	conv3-512 conv3-512	conv3-512 conv3-512
					conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

使用 keras 的 vgg16 模型进行特征提取，
参 考 了 https://blog.csdn.net/qq_37588821/article/details/97174536



已训练模型爬取过程：未对模型进行微调

以上数据处理完毕，进入 transformer 框架：使用多头注意力机制，通过 Q,K,V 的模态间特征提取，进行训练。自注意力机制的缺陷是模型在对当前位置的信息进行编码时，会过度的将注意力集中于自身的位置，因此使用多头注意力机制来解决这一问题。

首先对输入的文本和图片数据进行位置编码，生成位置向量，用于表示输入序列中不同位置的信息。位置编码 position_embeddding 公式为 PE(pos,2i) = sin(pos/10000^2i/dmodel) PE(pos,2i+1) = cos(pos/10000^2i/dmodel)。首先根据输入张量的形状确定位置向量的大小，然后计算位置向量的值。最后，根据模式将位置向量与输入张量相加或拼接，并返回结果。其中包含两种模式：sum 和 concat。sum 模式将位置向量与输入张量相加，concat 模式将位置向量与输入张量在特征维度上拼接。具体过程是，首先根据输入张量的形状，确定位置向量的大小（self.size）。然后，计算位置向量的值。使用 K.arange 生成一个浮点数数组，表示位置索引。然后通过对 10000 的 2 倍幂次运算，得到一个表示位置向量权重的数组 position_j。使用 K.expand_dims 在维度 0 上扩展 position_j，使其与输入张量的第二个维度大小一致。使用 tf.cumsum 计算一个与输入张量第二个维度相同大小的张量 position_i，表示累积的位置索引。使用 K.expand_dims 在维度 2 上扩展 position_i，使其与输入张量的第三个维度大小一致。使用 K.dot 进行矩阵乘法运算，得到位置向量的结果 position_ij。其中，position_i 表示每个位置的索引，position_j 表示每个位置的权重。将 position_ij 分别进行余弦和正弦函数的计算，并通过 K.concatenate 在特征维度上进行拼接得到最终的位置向量 position_ij。

注意力机制使用了多头注意力（multi-head attention）的思想，将输入张量分别作为查询（Q_seq）、键（K_seq）和值（V_seq）进行处理。在初始化函数__init__中，定义了注意力头数(nb_head)和每个头的大小(size_per_head)，然后使用三个权重矩阵 WQ、WK 和 WV，分别用于将输入张量映射到查询、键和值空间。这些权重矩阵的形状根据输入张量的形状动态确定。最后再通过 Mask 对输入张量进行掩码操作。掩码可以用来忽略序列中的特定位置，以防止无关信息的干扰。综上，多头注意力机制通过权重矩阵将输入张量映射到多头空间，并进行维度重排。然后，计算注意力得分矩阵 A，通过 softmax 函数将其归一化。最后，根据注意力得分和值张量计算输出张量。

如果输入参数的长度为 3，表示没有序列长度信息，即输入为 Q_seq, K_seq, V_seq。通过权重矩阵 self.WQ 将 Q_seq 映射到查询空间，得到查询张量 Q_seq。将查询张量 Q_seq 进行维度重排，以便进行多头注意力的计算。通过权重矩阵 self.WK 将 K_seq 映射到键空间，得到键张量 K_seq。将键张量 K_seq 进行维度重排，以便进行多头注意力的计算。剩下过程同理。通过计算查询和键的点积，并除以每个头的大小的平方根，得

到注意力得分矩阵 A。使用 softmax 函数将注意力得分矩阵 A 归一化，得到注意力权重矩阵 A。通过注意力权重矩阵 A 和值张量 V_seq 计算输出张量 O_seq。对输出张量 O_seq 进行维度重排和形状调整，得到最终的输出结果。返回输出张量 O_seq。

多头注意力机制的优点是每个头都是注意力，每个头筛选到的信息不同，信息更加丰富，有利于最终模型取得更好的效果。

由于对输入进行了改造，使得模型可能有多个句子 segment，为了识别字 token 属于哪个句子，需要加入 segment 的嵌入编码，所以 BERT 的输入融合了 word embedding、position encoding 和 segment embedding。

以上代码参考了以下两篇文章：Transformer 1. Attention 中的 Q，K，V 是什么 - 知乎 (zhihu.com)。(41 条消息) 【Pytorch】BERT+LSTM+多头自注意力（文本分类）_NLP 饶了我的博客-CSDN 博客

transformer 模型训练时采用 RELU 的激活函数，拼接三种张量 O_seq = tf.concat((O_seq_1, O_seq_2, O_seq_3), axis=1)，再进行平均池化 O_seq = tf.keras.layers.GlobalAveragePooling1D()(O_seq)，再正则化防止过拟合，O_seq = tf.keras.layers.Dropout(0.3)(O_seq)，最后增加全连接层 outputs = tf.keras.layers.Dense(3, activation='softmax')(O_seq)最终输出三维的类别概率向量，通过与原先标签进行对比得到损失大小。

综上，本代码使用 Transformer 在多模态情感分析中具有如下优点，多模态情感分析涉及不同模态的数据，如文本、图像、音频等。Transformer 模型具有灵活的输入表示方式，可以同时处理多种类型的输入数据。通过多头注意力机制，Transformer 能够对不同模态的特征进行交互和融合，从而更好地捕捉多模态数据中的语义和情感信息。Transformer 模型中的自注意力机制允许对不同模态之间的关系进行建模。通过计算注意力权重，Transformer 可以根据模态之间的相关性来加权融合不同模态的特征表示。这有助于更好地理解和利用多模态数据中的相关信息。另外，Transformer 模型的可扩展性和模块化设计使得它适用于不同规模和复杂度的多模态情感分析任务。可以根据数据集的大小和特征的复杂性来增加 Transformer 模型的层数或调整模型的大小，以满足任务需求。

```
3500/3500 - 2s - loss: 0.1417 - acc: 0.9474 - val_loss: 0.2781 - val_acc: 0.9058
Epoch 33/50
3500/3500 - 2s - loss: 0.1337 - acc: 0.9511 - val_loss: 0.2635 - val_acc: 0.9058
Epoch 34/50
3500/3500 - 2s - loss: 0.1346 - acc: 0.9477 - val_loss: 0.2756 - val_acc: 0.9038
Epoch 35/50
3500/3500 - 2s - loss: 0.1241 - acc: 0.9494 - val_loss: 0.2721 - val_acc: 0.9078
Epoch 36/50
3500/3500 - 2s - loss: 0.1258 - acc: 0.9531 - val_loss: 0.2633 - val_acc: 0.9138
Epoch 37/50
3500/3500 - 2s - loss: 0.1221 - acc: 0.9514 - val_loss: 0.2687 - val_acc: 0.9098
Epoch 38/50
3500/3500 - 2s - loss: 0.1184 - acc: 0.9543 - val_loss: 0.2747 - val_acc: 0.9018
Epoch 39/50
3500/3500 - 2s - loss: 0.1150 - acc: 0.9526 - val_loss: 0.2790 - val_acc: 0.9098
Epoch 40/50
...
Epoch 49/50
3500/3500 - 2s - loss: 0.0822 - acc: 0.9717 - val_loss: 0.2970 - val_acc: 0.9038
Epoch 50/50
3500/3500 - 1s - loss: 0.0769 - acc: 0.9706 - val_loss: 0.3044 - val_acc: 0.9038
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

三、训练结果及参数调整：

```
3500/3500 - 2s - loss: 0.1445 - acc: 0.9443 - val_loss: 0.5363 - val_acc: 0.8677
Epoch 35/50
3500/3500 - 2s - loss: 0.1427 - acc: 0.9460 - val_loss: 0.4839 - val_acc: 0.8657
Epoch 36/50
3500/3500 - 2s - loss: 0.1443 - acc: 0.9451 - val_loss: 0.4828 - val_acc: 0.8617
Epoch 37/50
3500/3500 - 2s - loss: 0.1368 - acc: 0.9460 - val_loss: 0.4945 - val_acc: 0.8717
Epoch 38/50
3500/3500 - 2s - loss: 0.1317 - acc: 0.9469 - val_loss: 0.4959 - val_acc: 0.8657
Epoch 39/50
3500/3500 - 2s - loss: 0.1285 - acc: 0.9520 - val_loss: 0.5867 - val_acc: 0.8697
Epoch 40/50
3500/3500 - 2s - loss: 0.1325 - acc: 0.9503 - val_loss: 0.5395 - val_acc: 0.8677
...
Epoch 49/50
3500/3500 - 2s - loss: 0.0999 - acc: 0.9626 - val_loss: 0.5791 - val_acc: 0.8677
Epoch 50/50
3500/3500 - 2s - loss: 0.0942 - acc: 0.9674 - val_loss: 0.6015 - val_acc: 0.8677
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

如上图，调整前损失为 0.09 在验证集上的准确率为 0.8677，训练速度很快大约为 8.0s，此时 epoch 为 50，batchsize 为 64

```
Epoch 34/50
3500/3500 - 2s - loss: 0.0749 - acc: 0.9734 - val_loss: 0.6589 - val_acc: 0.8597
Epoch 35/50
3500/3500 - 2s - loss: 0.0772 - acc: 0.9709 - val_loss: 0.6385 - val_acc: 0.8697
Epoch 36/50
3500/3500 - 2s - loss: 0.0777 - acc: 0.9743 - val_loss: 0.6465 - val_acc: 0.8737
Epoch 37/50
3500/3500 - 2s - loss: 0.0697 - acc: 0.9749 - val_loss: 0.6836 - val_acc: 0.8677
Epoch 38/50
3500/3500 - 2s - loss: 0.0618 - acc: 0.9763 - val_loss: 0.6342 - val_acc: 0.8597
Epoch 39/50
3500/3500 - 2s - loss: 0.0665 - acc: 0.9774 - val_loss: 0.7617 - val_acc: 0.8637
Epoch 40/50
3500/3500 - 2s - loss: 0.0675 - acc: 0.9751 - val_loss: 0.7447 - val_acc: 0.8617
...
Epoch 49/50
3500/3500 - 2s - loss: 0.0369 - acc: 0.9886 - val_loss: 0.8619 - val_acc: 0.8537
Epoch 50/50
3500/3500 - 2s - loss: 0.0405 - acc: 0.9869 - val_loss: 0.8833 - val_acc: 0.8557
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

调整 batch 为 16 以后，下降到 0.04，但是可以看到在验证集上为 0.088，正确率为 0.8557。通过对数据查看发现，训练数据中，正面情绪占比过大，所以导致预测结果不甚准确。

四、消融实验结果：即分别只输入文本或图像数据，你的多模态融合模型在验证集会获得怎样的表现。通过对输入数据的改变，发现只输入图片和图像，会使训练效果变差。只输入图片表现如下：

```python
#O_seq_1 = attention_keras.Attention(8, 16)([visual_embeddings, text_embeddings, text_embeddings])
O_seq_1 = attention_keras.Attention(8, 16)([text_embeddings, text_embeddings, text_embeddings])
O_seq_1 = tf.keras.layers.Activation('relu')(O_seq_1)

O_seq_2 = attention_keras.Attention(8, 16)([text_embeddings, visual_embeddings, visual_embeddings])
O_seq_2 = tf.keras.layers.Activation('relu')(O_seq_2)

O_seq_3 = attention_keras.Attention(8, 16)([visual_embeddings, visual_embeddings, visual_embeddings])
O_seq_3 = tf.keras.layers.Activation('relu')(O_seq_3)
#O_seq = tf.concat((O_seq_1, O_seq_2, O_seq_3), axis=1)
#O_seq = tf.concat((O_seq_1), axis=1)

O_seq = tf.keras.layers.GlobalAveragePooling1D()(O_seq_3)
```

```
Epoch 32/50
3500/3500 - 1s - loss: 0.3055 - acc: 0.9003 - val_loss: 0.4123 - val_acc: 0.8597
Epoch 33/50
3500/3500 - 1s - loss: 0.3036 - acc: 0.9003 - val_loss: 0.4175 - val_acc: 0.8597
Epoch 34/50
3500/3500 - 1s - loss: 0.3019 - acc: 0.9003 - val_loss: 0.4215 - val_acc: 0.8597
Epoch 35/50
3500/3500 - 1s - loss: 0.3018 - acc: 0.9003 - val_loss: 0.4216 - val_acc: 0.8597
Epoch 36/50
3500/3500 - 1s - loss: 0.2961 - acc: 0.9003 - val_loss: 0.4168 - val_acc: 0.8597
Epoch 37/50
3500/3500 - 1s - loss: 0.2947 - acc: 0.9006 - val_loss: 0.4266 - val_acc: 0.8597
Epoch 38/50
3500/3500 - 1s - loss: 0.2898 - acc: 0.9006 - val_loss: 0.4170 - val_acc: 0.8577
Epoch 39/50
3500/3500 - 1s - loss: 0.2914 - acc: 0.9000 - val_loss: 0.4544 - val_acc: 0.8597
Epoch 40/50
...
Epoch 49/50
3500/3500 - 1s - loss: 0.2707 - acc: 0.9000 - val_loss: 0.4633 - val_acc: 0.8537
Epoch 50/50
3500/3500 - 1s - loss: 0.2693 - acc: 0.9034 - val_loss: 0.4573 - val_acc: 0.8557
```

四、错误:

1.使用 predict_classes()报错：'Model' object has no attribute 'predict_classes'。
解决：使用 predict 再取三维中的最大值即为分类的所属类别。
2.TimeoutError: [WinError 10060] 由于连接方在一段时间后没有正确答复或连接的主机没有反应，连接尝试失败。
解决：(41 条消息) 已解决 TimeoutError: [WinError 10060] 由于连接方在一段时间后没有正确答复或连接的主机没有反应，连接尝试失败。_袁袁袁袁满的博客-CSDN 博客
3. 保存模型时出错：**TypeError**: ('Not JSON Serializable:', b'\n\x06concat\x12\x08ConcatV2\x1a\x0factivation/Relu\x1a\x11activation_1/Relu\x1a\x11activation_2/Relu\x1a\x0bconcat/axis*\n\n\x04Tidx\x12\x020\x03*\x07\n\x01T\x12\x020\x01*\x07\n\x01N\x12\x02\x18
解决：Keras v 2.2.4-tf 保存模型 值错误：无法打包序列。·问题 #12961 ·凯拉斯团队/凯拉斯 ·GitHub