实验五 多模态情感分析

GitHub 地址：https://github.com/NoExsisted/AI_project5

多模态融合模型是一种能够处理多种不同类型数据输入的深度学习模型。这些不同类型的数据可以是图像、文本、音频等。多模态融合模型的目标是将这些不同类型的数据进行融合，以便模型可以从中学习到更丰富的信息，并做出更准确的预测或决策。

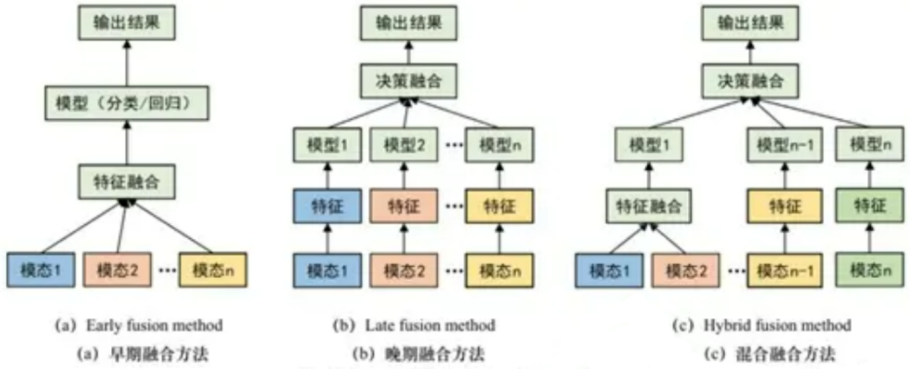
在多模态融合模型中，每种类型的数据都会经过专门的处理网络，比如图像数据可能会经过卷积神经网络（CNN），文本数据可能会经过循环神经网络（RNN）或 Transformer，音频数据可能会经过卷积神经网络或者一维卷积神经网络。（本次实验只涉及图像和文本数据）

处理完各自的数据后，这些数据的特征会被融合在一起，然后输入到一个全连接层或其他类型的层中进行最终的预测或决策。

多模态融合模型的优势在于它能够综合不同类型数据的信息，从而提高模型的性能和鲁棒性。比如在一个视听识别任务中，结合图像和音频信息可以提高识别的准确性；在一个文本图像匹配任务中，结合文本和图像信息可以提高匹配的准确性。

在实际应用中，多模态融合模型可以用于各种任务，比如图像标注、视频内容理解、自然语言处理等。它为处理多模态数据提供了一种有效的方法，使得模型可以更好地理解和利用不同类型数据之间的关联，从而提高模型的表现。

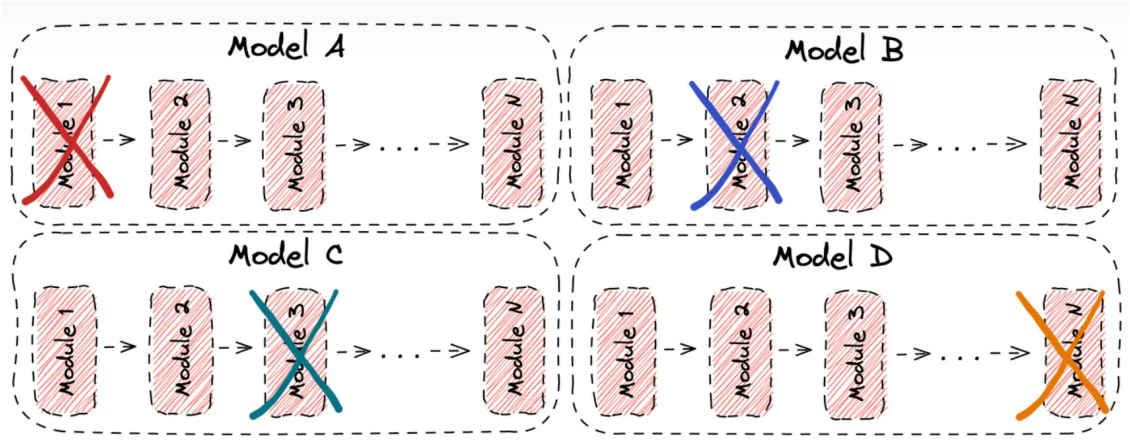
总的来说，多模态融合模型是一种强大的深度学习模型，能够处理多种不同类型的数据输入，并通过融合这些数据来提高模型的性能和应用范围。



消融实验是指在研究中逐步剥离模型中的某些部分，以便评估这些部分对模型性能的影响。在多模态融合模型中，消融实验可以用来分析不同模态数据对最终模型性能的贡献，以及各个部分之间的相互影响。例如，在一个图像和文本的多模态融合模型中，可以进行以下消融实验：

1. 图像消融：评估在去除图像输入的情况下模型的性能如何。这可以帮助确定图像对最终预测的贡献程度。
2. 文本消融：评估在去除文本输入的情况下模型的性能如何。这可以帮助确定文本对最终预测的贡献程度。
3. 融合方式消融：评估在改变不同的融合方式（如拼接、相加等）时模型的性能如何。这可以帮助确定不同融合方式对最终模型性能的影响。

通过消融实验，可以更好地理解模型中各个部分的作用，从而进行模型设计和优化。



实验目的

- 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。

数据预处理

1. 首先是文本读取，编码模式选择 `ansi`，获取目标文件中的内容。

```
def process_text(text_file_path):
    with open(text_file_path, 'r', encoding = 'ansi') as text_file_content:
        text_data = text_file_content.read()
    return text_data
```

bug1：如果编码模式用 `gbk` 或 `utf-8` 都会报错

```
UnicodeDecodeError: 'gbk' codec can't decode byte 0xac in position 75: illegal multibyte sequence
```

```
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xa1 in position 90: invalid start byte
```

解决方法1：打开数据集中的文本，发现有些文本显示的编码模式是 `ansi`，所以换了 `ansi` 试试，没有报错。

2. 然后是图片读取。将图片尺寸统一调整为 224 * 224。

```
def process_image(image_file_path):
    image = Image.open(image_file_path)
    # 调整图像尺寸为统一的大小
    resized_image = image.resize((224, 224))
    # 将图像转换为 numpy 数组并添加到 image_data 列表中
    image_data = np.array(resized_image)
    return image_data
```

bug2：如果不调整图像尺寸，接下来会无法转化成数组，因为每张图片尺寸不尽相同，所以矩阵无法对齐。

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[3], line 2
      1 labeled_data, unlabeled_data, test_data = functions.get_data()
----> 2 image_data_array, text_data_array, label_numeric_array, tokenizer = functions.get_train_array(labeled_data, unlal
      3 test_text_array, test_image_array = functions.get_test_array(test_data, tokenizer)

File c:\Users\lianxiang\Desktop\大三上\AI\exp5\functions.py:99, in get_train_array(labeled_data, unlabeled_data)
     96 # 将填充后的矩阵转换为Numpy数组
     97 padded_sequences_array = np.array(padded_sequences)
---> 99 image_data_array = np.array(image_data_list)
    100 text_data_array = padded_sequences_array
    101 label_numeric_array = np.array(label_numeric_list)

ValueError: setting an array element with a sequence. The requested array has an inhomogeneous shape after 1 dimensions.
```

解决方法2：在读取的时候就将图像尺寸调整为统一的大小。

3. 在定义一个读取整体数据集的函数，调用上面定义的两个函数。`labeled_data`，`unlabeled_data`，`test_data` 分别储存不同的数据。因为 data 中有一部分编号既不在训练集中也不在测试集中。这一部分文本在 tokenizer.fit_on_texts 时也是需要的，所以保留。区分测试数据和训练数据是为了方便训练和预测。

   将情感分析的三种结果通过 `label_mapping` 字典分别映射到 0，1，2，便于模型训练。

```python
label_mapping = {"negative": 0, "neutral": 1, "positive": 2}
def get_data():
    # 读取train.txt文件
    train_data = pd.read_csv('../实验五数据/train.txt')
    # 创建一个字典存储标签信息
    label_dict = dict(zip(train_data['guid'], train_data['tag']))
    test_data = pd.read_csv('../实验五数据/test_without_label.txt')
    test = list(test_data['guid'])

    labeled_data = []
    unlabeled_data = []
    test_data = []

    # 遍历data文件夹
    data_folder = '../实验五数据/data/'
    for file_name in os.listdir(data_folder):
        if file_name.endswith('.txt'):
            guid = file_name.split('.')[0]  # 获取文件名中的编号
            if int(guid) in label_dict:
                # 有标签的数据
                label = label_dict[int(guid)]
                label_numeric = label_mapping[label]
                # 读取文本和图像数据
                text_file = os.path.join(data_folder, file_name)
                image_file = os.path.join(data_folder, guid + '.jpg')
                text_data = process_text(text_file)
                image_data = process_image(image_file)
                labeled_data.append((text_data, image_data, label_numeric))
            else:
                # 无标签的数据，读取文本和图像数据
                text_file = os.path.join(data_folder, file_name)
                image_file = os.path.join(data_folder, guid + '.jpg')
                text_data = process_text(text_file)
                image_data = process_image(image_file)
                unlabeled_data.append((text_data, image_data))
                if int(guid) in test:
```

```
                    test_data.append((text_data, image_data))
    # labeled_data 存储有标签的训练数据，unlabeled_data 存储无标签的训练数据
    # test_data 存储测试数据
    return labeled_data, unlabeled_data, test_data
```

这样将数据读取的过程封装成函数放在 functions.py 文件中，要调用只需要执行以下两行

```
import functions
labeled_data, unlabeled_data, test_data = functions.get_data()
```

这样得到的 `labeled_data`，`unlabeled_data`，`test_data` 都是列表，接下来要分开转换成数组，才能输入模型进行训练。

4. 将训练集数据转化成数组。从 labeled_data 中分别取出 `text_data`，`image_data`，`label_numeric` 保存到 `text_data_list`，`image_data_list`，`label_numeric_list`，将 unlabeled_data 的文本数据添加到 `text_data_list` 中。

   文本处理：创建一个 Tokenizer 对象，将所有的文本数据放入，再将每个句子转换为序列并进行填充。根据填充前查找最大值，发现最大文本长度设为 35 能容纳下所有数据。将填充后的矩阵转换为 numpy 数组

   图像和标签不用再额外处理，转为数组即可。返回三个数组和 tokenizer，tokenizer用于测试集的文本向量化。

```
def get_train_array(labeled_data, unlabeled_data):
    # 从labeled_data中分别取出text_data, image_data, label_numeric
    text_data_list = [item[0] for item in labeled_data]
    image_data_list = [item[1] for item in labeled_data]
    label_numeric_list = [item[2] for item in labeled_data]

    all_text = []
    # 将unlabeled_data的文本数据添加到text_data_list中
    for item in unlabeled_data:
        all_text.append(item[0])

    # 处理文本
    # 创建一个Tokenizer对象
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(all_text)
    # 将每个句子转换为序列并进行填充
    max_length = 35
    padded_sequences = []
    for sentence in text_data_list:
        sequence = tokenizer.texts_to_sequences([sentence])[0]
        padded_sequence = pad_sequences([sequence], maxlen=max_length,
padding='post')[0]
        padded_sequences.append(padded_sequence)
    # 将填充后的矩阵转换为numpy数组
    padded_sequences_array = np.array(padded_sequences)

    image_data_array = np.array(image_data_list)
    text_data_array = padded_sequences_array
    label_numeric_array = np.array(label_numeric_list)
    return image_data_array, text_data_array, label_numeric_array, tokenizer
```

5. 将测试集数据转化成数组。与训练集类似，只是不用再创建 tokenizer 对象。

```
def get_test_array(test_data, tokenizer):
    max_length = 35
    test_text_list = [item[0] for item in test_data]
    padded_sequences = []
    for sentence in test_text_list:
        sequence = tokenizer.texts_to_sequences([sentence])[0]
        padded_sequence = pad_sequences([sequence], maxlen=max_length,
padding='post')[0]
        padded_sequences.append(padded_sequence)

    # 将填充后的矩阵转换为Numpy数组
    test_text_array = np.array(padded_sequences)
    test_image_list = [item[1] for item in test_data]
    test_image_array = np.array(test_image_list)
    return test_text_array, test_image_array
```

模型定义

可能由于任务比较简单，或者我的思路比较简单，所以早期融合、后期融合和混合级融合差异不大。以下都以早期融合为例，其他两种写在代码中，不过多赘述。

图像模态处理

- 图像输入：模型接受大小为 (224, 224, 3) 的图像输入。
- 卷积层：使用了两个卷积层，分别为 32 个 (3, 3) 大小的卷积核和 64 个 (3, 3) 大小的卷积核，激活函数为 ReLU，并通过最大池化层进行下采样。
- 全连接层：Flatten 层将卷积层输出展平后，经过一个包含 64 个神经元的全连接层，激活函数为 ReLU，得到图像模态的输出。

文本模态处理

- 文本输入：模型接受长度为 35 的文本序列输入。
- 嵌入层：对文本序列进行嵌入，将 5130 维的词汇映射到 100 维的词嵌入空间。
- LSTM层：使用具有 64 个隐藏单元的 LSTM 层对文本序列进行建模，得到文本模态的输出。

多模态融合

- 合并特征：图像模态和文本模态的输出通过 concatenate 操作进行融合。
- 全连接层：融合后的特征通过一个包含 64 个神经元的全连接层，激活函数为 ReLU。

输出层

- 分类层：最终通过一个包含 3 个神经元的全连接层，激活函数为 softmax，得到模型的输出。

```
def early_fusion():
    # 定义图像模态的输入
    image_input = Input(shape=(224, 224, 3))
    x = Conv2D(32, (3, 3), activation='relu')(image_input)
    x = MaxPooling2D((2, 2))(x)
    x = Conv2D(64, (3, 3), activation='relu')(x)
    x = MaxPooling2D((2, 2))(x)
    x = Flatten()(x)
    image_output = Dense(64, activation='relu')(x)
```

```
    # 定义文本输入
    text_input = Input(shape=(35,), name='text_input')
    text_embedding = Embedding(input_dim=5130, output_dim=100)(text_input)
    y = LSTM(64)(text_embedding)

    # 合并两个模态的输出
    combined = concatenate([image_output, y])
    z = Dense(64, activation='relu')(combined)
    output = Dense(3, activation='softmax')(z)

    model = Model(inputs=[image_input, text_input], outputs=output)
    return model
```

模型优势

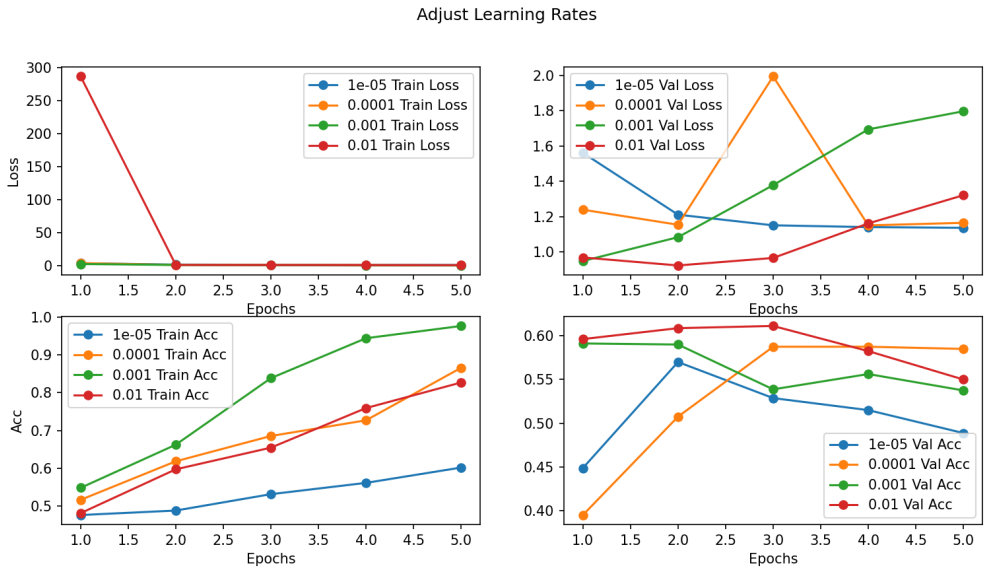
- **综合考虑多模态信息**：模型能够同时处理图像和文本信息，并在较早的阶段将它们融合，有利于模型更好地学习多模态信息之间的关联。
- **适用于小规模数据集**：模型结构相对简单，适用于小规模数据集的训练，避免了过度复杂的模型结构可能导致的过拟合问题。
- **高效处理短文本数据**：针对文本数据较短小的特点，采用了简单的文本处理模型，提高了模型的计算效率。

调参

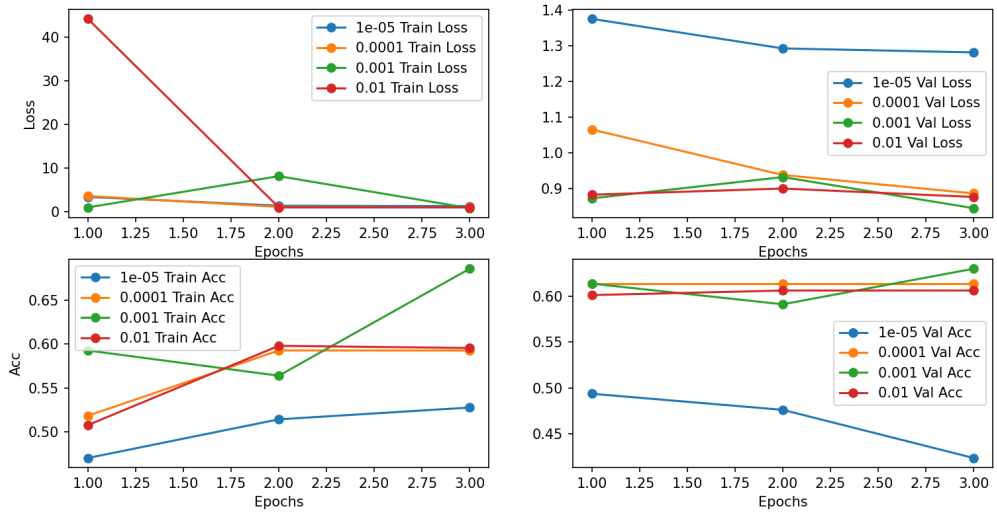
首先调学习率，优化器为 Adam，学习率取以下五个值分别试验（epoch = 5，batch_size = 64）

```
learning_rates = [1e-5, 1e-4, 1e-3, 1e-2]
```

将每一轮运行结果整理为下图，可以更直观地看到运行过程，看到验证集准确率在第三次迭代之后都出现了下降，所以重新训练，epoch = 3。



调整后运行结果如下



最终输出结果

```
最佳学习率：0.001，最佳准确率：0.6299999952316284
运行时间：454.014秒
```

(详细代码在 adjust_lr.py 中)

预测并整理结果

1. model.predict 预测的结果的尺寸是 (511, 3)

```
16/16 [==============================] - 3s 109ms/step
[[8.1332084e-03 4.5181736e-02 9.4668502e-01]
 [1.7830946e-02 1.2679212e-01 8.5537696e-01]
 [4.9205357e-03 1.5340184e-01 8.4167755e-01]
 ...
 [9.8007727e-01 6.7696169e-06 1.9915922e-02]
 [7.2916850e-02 2.0629815e-03 9.2502022e-01]
 [9.3282284e-03 1.7360168e-03 9.8893577e-01]]
```

所以从每一行选出数组最大的索引作为预测标签，再通过 label_mapping 字典映射回原本的文字情感标签。

```python
def map_labels(labels):
    key = list(label_mapping.keys())
    mapped_labels = []
    for label in labels:
        mapped_labels.append(key[label])
    return mapped_labels
```

调用上述定义的函数 map_labels

```python
predictions = model.predict([test_image_array, test_text_array])
max_indices = np.argmax(predictions, axis=1)
labels = functions.map_labels(max_indices)
```

得到 labels 如图所示

```
print(labels)
```

✓ 0.0s Python

```
['positive', 'positive', 'positive', 'neutral', 'negative', 'negative', 'negative', 'positive', 'positive', 'negative', 'posi
```

2. 将 labels 写入 test_without_label.txt 对应位置。定义 write_test 函数，传入 labels 和 file_path，将 labels 写入指定文件的对应位置。

```
def write_test(labels, file_path):
    with open(file_path, 'r') as file:
        lines = file.readlines()
        for i in range(1, len(lines)):
            parts = lines[i].split(',')
            if parts[1].strip() == "null":
                parts[1] = labels[i - 1] + "\n"
                lines[i] = ','.join(parts)

    with open(file_path, 'w') as file:
        file.writelines(lines)
```

消融实验

- 只输入文本数据

  模型

  ```
  def text_only():
      # 移除图像输入的消融实验
      text_input = Input(shape=(35,), name='text_input')
      text_embedding = Embedding(input_dim=5130, output_dim=100)
  (text_input)
      y = LSTM(64)(text_embedding)
      output = Dense(3, activation='softmax')(y)
      model_text_only = Model(inputs=text_input, outputs=output)
      return model_text_only
  ```

  运行结果

  ```
  50/50 [==============================] - 2s 21ms/step - loss: 0.9398 - accuracy: 0.5869 - val_loss: 0.8862 - val_accuracy:
  0.6137
  Epoch 2/3
  50/50 [==============================] - 1s 14ms/step - loss: 0.9141 - accuracy: 0.5928 - val_loss: 0.8871 - val_accuracy:
  0.6137
  Epoch 3/3
  50/50 [==============================] - 1s 14ms/step - loss: 0.9137 - accuracy: 0.5928 - val_loss: 0.8860 - val_accuracy:
  0.6137
  25/25 [==============================] - 0s 4ms/step - loss: 0.8860 - accuracy: 0.6137
  验证集准确率:  0.6137499809265137
  运行时间: 4.418秒
  ```

- 只输入图像数据

  模型

```python
def image_only():
    # 移除文本输入的消融实验
    image_input = Input(shape=(224, 224, 3))
    x = Conv2D(32, (3, 3), activation='relu')(image_input)
    x = MaxPooling2D((2, 2))(x)
    x = Conv2D(64, (3, 3), activation='relu')(x)
    x = MaxPooling2D((2, 2))(x)
    x = Flatten()(x)
    image_output = Dense(64, activation='relu')(x)
    output = Dense(3, activation='softmax')(image_output)
    model_image_only = Model(inputs=image_input, outputs=output)

    return model_image_only
```

运行结果

```
验证集准确率:  0.5299999713897705
运行时间: 117.523秒
```

可以看到准确率大部分是文本数据提供的。这可能是由于图像预处理时只保留了 224 * 224，而有的图像尺寸为 600 * 600，可能有效信息保存得不够多，导致图像数据的准确率较低。

总结

- 代码实现时遇到了哪些bug？如何解决的？（详见数据预处理）
  - bug1：读取文本数据报错。解决方案1：编码模式选用 `ansi`。
  - bug2：数据集转换成数组报错。解决方案2：读取时统一图像尺寸。
- 你为什么会设计这样的模型？你觉得你的模型有什么亮点？
  - 基于数据集中文本较短小且数量较少的特点，设计这样的模型是出于以下考虑：
    1. 数据特点：由于文本都比较短小，每条序列长度不超过 35，因此可以使用较简单的文本处理模型，如 Embedding 层和 LSTM 层，来捕捉文本数据的特征。这样的模型设计能够有效地处理短文本数据，避免了使用过于复杂的文本处理模型，从而提高了模型的计算效率。
    2. 多模态信息融合：通过早期融合的方式将图像和文本的特征在较早的阶段合并，有助于模型更好地学习图像和文本之间的关联。这种设计能够充分利用多模态信息，提高了模型的表征能力，有助于提高分类的准确性。
    3. 适用性：考虑到数据集规模较小，设计一个相对简单的模型结构能够更好地适应数据集的特点，避免了过度复杂的模型结构可能导致的过拟合问题。
  - 模型的亮点在于能够有效地处理短文本和图像数据，充分利用了多模态信息，提高了模型的表征能力和分类准确性。另外，模型结构相对简单，易于理解和调整，适用于小规模数据集的训练。
- 多模态融合模型在验证集上的结果。
  - 验证集准确率 0.63。
  - 损失函数及训练详情见 **调参** 部分图表。
- 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
  - 只输入文本数据：验证集准确率 0.6137。
  - 只输入图像数据：验证集准确率 0.53。