第 5 次实验报告（github 地址：https://github.com/ChicagoCubsOfficial/DaSE-homework）

陈存衡 新闻学（双学位） 10223330408

实验目的：

本实验旨在设计一个多模态融合模型，通过文本和图像数据的联合学习，进行情感分类任务（positive, neutral, negative），同时通过消融实验验证单模态和多模态的性能差异。

实验过程：

数据加载与预处理:

从 train.txt 加载 GUID 和标签，过滤无效标签；文本数据采用 TF-IDF 向量化，图像数据通过 torchvision.transforms 进行归一化和尺寸调整；将数据划分为训练集和验证集，确保实验结果的可复现性。

数据集设计:

定义 MultimodalDataset 数据集类，支持文本和图像特征的动态加载；通过标志位控制是否启用文本或图像数据，便于实现消融实验。

多模态模型构建:

模型包含文本特征和图像特征的子网络；文本通过线性层降维，图像特征提取基于 ResNet18；将两种模态特征融合后通过全连接层输出预测结果。

训练和验证:

训练阶段采用交叉熵损失优化模型；验证阶段引入精确率度量指标，以评估模型性能，并通过早停机制优化训练时间。

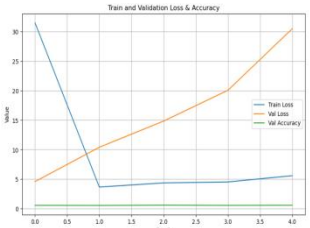
测试集预测:

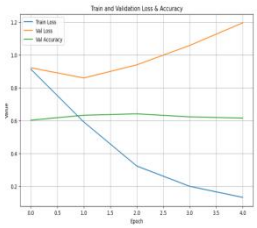
使用训练好的模型对测试集进行推断；将预测标签保存到 CSV 文件中，以便后续分析。

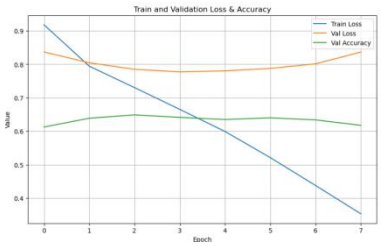
实验分析：

1 学习率问题

Lr 参数上，笔者选取 0.1,、0.001、0.00001（下图从左至右）三类作比较。





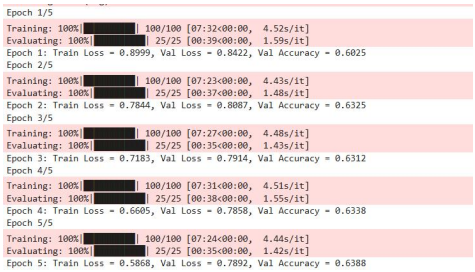


观察到，在 lr=0.1 时，训练损失很高，Train 和 Val 的 loss 在第一轮为 31 和 4，在 epoch5 之后，这一数值仍然很高，远高于其余两个。0.00001 的验证准确率最好，达到了 65%左右，其他两个在 50-60%。**学习率的变化（越小）有助于准确率提升。**

Lr=0.1,0.001 时也出现了相当明显的过拟合。

2 EPOCH 问题

由于本人设置了早停机制，在 Epoch=5,10,30 都有所实验。



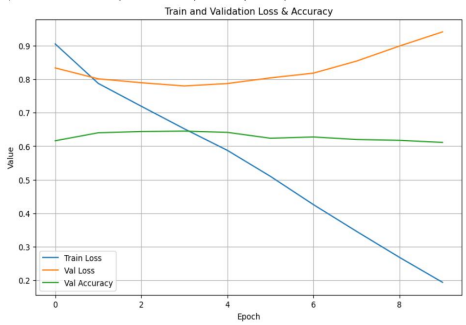


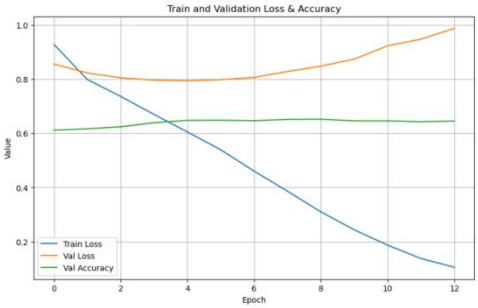
图 1 lr=1e-5,epoch=10

表 1 lr=1e-5,epoch=5

在表中，我们可以与 lr=1e-5,epoch=30(早停在 13）比较，***epoch 的增加确实可以减少损失，增加模型的准确度，提升性能***。但在 10 轮之后，这样的优势就比较微弱，每一轮的增加带来的***收益几乎可以忽略不计***。

3 划分比例

划分训练集验证集比例是会影响到模型的一个参数。本代码中我们选取训练验证 82 开（test_size=0.2)和 64（test_size=0.4)开作比较。下图为 0.4,而 0.2 的情况为学习率模块的最右边图，可自行查看。



可以看到，**训练集比例大（0.2）的情况下，epoch 运行轮数少，训练得到较好效果的速度更快**；在验证准确率问题上，训练集比例大（0.2）的初期准确率好于 ts=0.4，而在早停机制加持下，最后准确率都可以达到 65%左右。可以说训练集划分比例高有助于效率。

4 CPU/GPU

由于笔者使用的硬件设备是轻薄本，CPU 运行。

在实验效率上，与笔者室友的游戏本（GPU 运行）相对比：训练验证速度上，笔者速度如下图，时间如下图。在同等条件仅改变硬件环境下，室友的速度是本人的 6-8 倍左右。

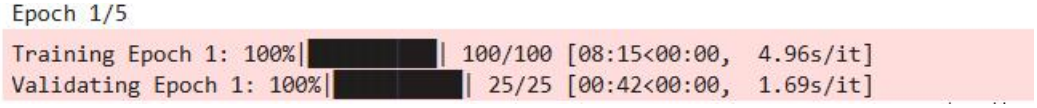


表 2 运行速度

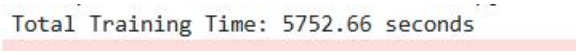


表 3 运行时间（Epoch=13,早停，TFIDF+RESNET）

5 Batchsize、图像、维度等

除了以上几种参数或硬件的“控制变量”实验以外，笔者还考虑了：BatchSize，图像输入尺寸,TF-IDF 的最大特征数量（max_features)等参数。但这些参数的改变并未实现笔者的猜想，似乎对实验结果的影响不是很大：例如改变 Batchsize128 到 256（EPOCH 早停+Lr0.00001),

在训练、验证方面的结果和曲线图并未产生太大差距。

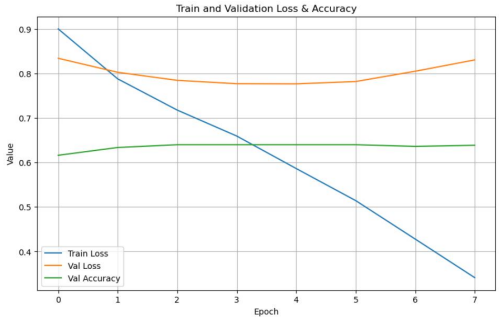


图 2 batchsize=256(几乎无差异)

6 消融实验结果

笔者增加两个实验标志，用于控制消融实验的运行：use_text = True# 使用文本数据 use_image = False # 不使用图像数据 TRUE--用 FALSE--不用

（1）仅文本

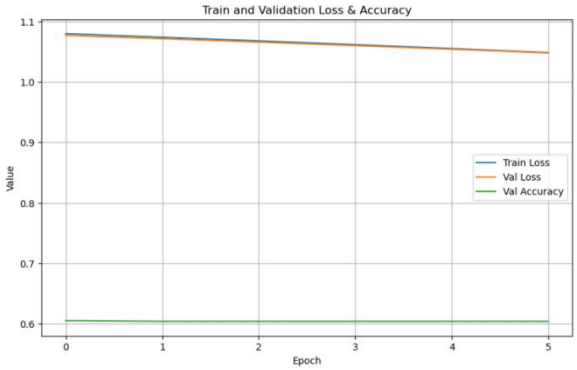


图 3 仅让模型学习文本（最初）

在仅输入文本的情况下，笔者在实验之初可以从图中观测到，与输出文本、图像的情况居然差不多。这很显然是代码本身少了一些考量。观测我们的 train 文本，大部分标记都是 positive，这就给了模型浑水摸鱼的机会。在观测“仅输入文本”的输出后，笔者发现 test 到的全是 positive，那自然会有比较高的正确率（模型蒙对了）。由此，笔者在代码中添加了 F1 度量，防止其“浑水摸鱼”。结果发现，仅输入文本的正确率很低。

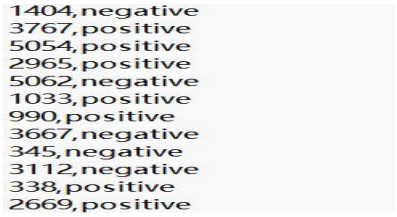


图 4 大部分都是 positive 比例超过 60%

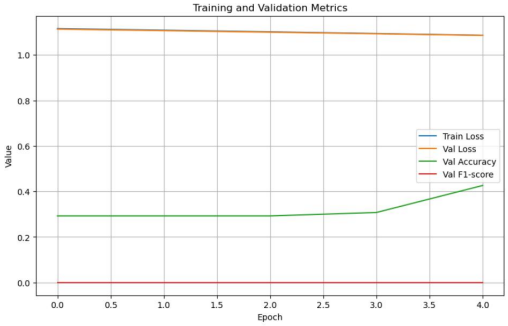


图 5 改良后的仅文本曲线图

图中可知，仅**文本的验证准确率仅在 20%-40%**之间。

（2）仅图像

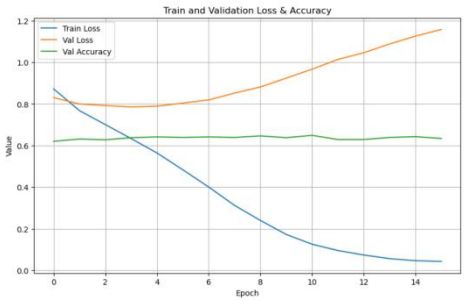


图 6 仅图像

仅用图像，早停情况下运行了 16 轮，Val loss 和 Accuracy 已经过拟合，曲线有所上涨，训练效果在后期已经较为乏力。然而 Val Accuracy 的程度在 0.6 左右，验证准确度表现尚可。

实验途中遇到的 BUG/困难：

1 为什么设计了这样的模型？（选取的预处理模型效率较低之下的无奈选择）

在实验开始前笔者选取 Bert 作为文本特征提取器。由于笔者的电脑仅有 CPU 处理能力，采用 Bert+ResNet 的运行效率很差，因此决定作出改变。

转变思路后，使用 TF-IDF + 图像特征进行多模态情感分类。文本处理方面，将使用 TfidfVectorizer 来对文本进行处理。TF-IDF 是一种经典的文本向量化方法，将文本转化为数值特征。图像处理则继续使用 ResNet 来提取图像特征。模型便是使用一个简单的多层感知机（MLP）来融合文本和图像特征。

从速度来说 TF-IDF 计算比 BERT 快很多，训练和推理时计算量也显著减小，尤其是对于小型数据集，速度提升明显；资源消耗方面，BERT 需要大量的内存和计算资源，尤其是大规模文本处理时，相比之下 TF-IDF 的计算开销小，适合低资源环境。***从这方面来说，这样的设计是一个小亮点，很适合硬件设备差的同学***，在有限资源里获取一个较好的结果。此外，笔者在后面还加入了“Patience 早停”和 F1 度量等元素，增强了其训练能力，也能确保试验的结果更精准，毕竟本次几千张图片里有超过 60%的图片是 positive，模型存在“浑水摸鱼”的可能性。

真实情况	预测结果	
	正例	反例
正例	TP(真正例)	FN(假反例)
反例	FP(假正例)	TN(真反例)

不过文本理解能力和模型表达能力，自然因为 TF-IDF 是基于词频的浅层模型，无法捕捉上下文和词义的变化而在能力上打折扣：BERT 是现阶段很强的预训练语言模型，能够理解文本的深层次含义，但 TF-IDF 仅仅考虑单个词的频率信息，无法捕捉词汇间的语义关系，因此性能会明显下降；BERT 能够学习更为复杂的上下文关系，而 TF-IDF 只能利用静态的词频，因此无法处理多义词、上下文的变化等复杂语言现象，这两个方面会差一些。

2 报错“No such file or directory: '/data\\guid.txt'”

错误原因在于路径，代码的文件结构中，train.txt 和 test_without_label.txt 包含了每个样本的 guid 和标签，而笔者初始的代码试图加载 guid.txt，但实际上文件路径应该是通过 guid 动态生成的，例如 4595.txt 等。进而从文件读取上讲，train.txt 和 test_without_label.txt 中的 guid 应该与 data 文件夹中的文本文件名对应，因此文件路径应根据 guid 来动态生成，而不是硬编码为 guid.txt。将 train.txt 中的 guid 和标签读取为 train_df，在读取文本和图片时，构造正确的路径，确保文件存在，处理文件路径时，确保每次根据 guid 动

态加载对应的文本和图片文件，问题得到了解决。

```
# 加载训练集标签
train_df = pd.read_csv(train_file, header=None, names=["guid", "label"])

# 文本预处理 (使用 TF-IDF)
tfidf_vectorizer = TfidfVectorizer(max_features=5000)  # 限制最大特征数量
train_texts = []

# 将每个文本读取并加入train_texts中
for guid in train_df["guid"]:
    text_path = os.path.join(data_dir, f"{guid}.txt")
    with open(text_path, "r") as file:
        train_texts.append(file.read())

# 对文本进行 TF-IDF 转换
X_text = tfidf_vectorizer.fit_transform(train_texts).toarray()

# 图像预处理
image_transform = transforms.Compose([
    transforms.Resize((224, 224)),                    截图(Alt + A)
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

3 报错“UnicodeDecodeError: 'utf-8' codec can't decode byte 0xa1 in position 139: invalid start byte”

遇到此 UnicodeDecodeError，笔者认为，在使用 utf-8 编码读取文件时，某些文件包含了无法解码的字节。笔者想过尝试使用其他编码格式读取文件——在读取文件时添加对其他常见编码格式的支持（例如 gbk）；当然，笔者认为也可跳过无法解码的文件。最终，笔者采用了前者模式，然后做了占位符处理（避免返回空文本并减轻对模型性能的影响），但文本信息的丢失的确无法避免。不过这是权衡了所有因素之后所处的决定。

```
    # 尝试读取文本文件，首先尝试 utf-8 编码
    try:
        with open(text_path, "r", encoding="utf-8") as file:
            text = file.read()
    except UnicodeDecodeError:
        # 如果 utf-8 编码失败，尝试 gbk 编码
        try:
            with open(text_path, "r", encoding="gbk") as file:
                text = file.read()
        except Exception as e:
            print(f"Warning: Unable to read text file {text_path} due to error: {e}")
            text = "[TEXT_ERROR]"  # 用占位符代替无法读取的文本
    except FileNotFoundError:
        print(f"Warning: Text file not found - {text_path}")
        text = "[TEXT_MISSING]"  # 用占位符代替缺失的文本

    # 使用 tfidf_vectorizer 对文本进行处理
    text_features = self.tfidf_vectorizer.transform([text]).toarray().flatten()
```

笔者还新增 chardet 模块检测文件编码。

```
import chardet

# 检测文件编码并读取文件
def read_file_with_auto_encoding(file_path):
    with open(file_path, "rb") as f:
        raw_data = f.read()
        result = chardet.detect(raw_data)
        encoding = result["encoding"] if result["encoding"] else "utf-8"  # 默认用 utf-8
        try:
            return raw_data.decode(encoding)
        except Exception as e:
            print(f"Warning: Unable to decode file {file_path} with detected encoding {encoding}: {e}")
            return "[TEXT_ERROR]"
```

4 报错“IndexError: Target 3 is out of bounds.”

这个报错花费了大量的修改时间，问题在于：模型的输出类别数和标签的取值范围不匹配，在计算损失时需要目标标签的值是 0, 1, 2，而如果标签的值为 3，***意味着目标标签超出了类别范围***。

最终调整：

1 清理标签数据，仅保留 ["positive", "neutral", "negative"]。

2 动态设置模型的输出维度为 num_classes = len(label_encoder.classes_)。

3 验证数据加载和标签处理部分的完整性，确保无越界标签，而且笔者设置了功能：打印调试信息验证每个步骤；tqdm 等可视化步骤功能为笔者支持这一点。