模式识别大作业报告

明陈林 2019011457 自96

1. 问题分析与形式化

1.1 问题背景

在各大电商平台上，衣物一般都是流通量较大的商品类目。一款衣物，可能根据其尺码、颜色、图案等等特征分为更细的子款式。在这次作业中，我们主要关注"颜色"所带来的款式区分。我们需要处理的是一个折衷的颜色识别和区分问题——假设商家上传一个衣服所有款式的图片，和可供选择的一些描述颜色的文字标签，设计一套算法，将这些图片和这些文字标签进行匹配。

1.2 问题分析

抽象地来说，本次任务是一个较为粗粒度的图像和文本两模态数据匹配的任务。最终选择的方案为文字标签和图片均用模型处理的方案。理由有两点，参考方案中使用基于规则的匹配会非常考验规则的制订和标准库的泛化性，很容易就会出现细分的颜色没办法区分，或者出现标准库中没有出现的形容颜色的类别（淘宝为了吸引顾客眼球，常常会起一些萌萌的、或者形容词类别的词语）此外，片面的以标准库中匹配得到的颜色作为整个图片的类别也可能导致很大一类都会没有区别，使得数据集数据参考意义下降。综上所述，在思索了许久之后没有想到较好的基于规则的匹配后，我决定转而采用文字标签和图片均用模型处理的方案。将图片和文本进行编码，进行之间特征数值的计算，最终根据相似数值的大小关系确定图片的标签。

1.3 问题形式化

输入：一个商品的所有图片：$T_1 = I_1, I_2, \dots I_n$，以及可选的描述颜色的字符标签 $T_2 = I_1, I_2, \dots I_m$。

输出：图片和文字标签的匹配关系，即需要输$(I_1, T_1), (I_2, T_2), \dots, (I_n, T_n)$。自然地，每张图的匹配标签需要在可选颜色字符标签集中做选取，即需要满足约束$\forall i \in 1, 2, \dots, n, I_i \in I, T_i \in T$。最终输出到$json$文件进行网站测评。

2. 数据处理流程

经过上述分析，本题输入为两种模态，一个是同一种商品的所有图片，一个是所有的可选的标签，一个走图片编码通道，一个走自然语言处理通道。在对可选标签进行处理的时候，我发现如果用中文直接进行编码，会出现字符太长的问题（淘宝会加上各种来迎合关键词搜索，所以有的衣服标签就会很长）。所以在文本处理前，我会将中文先翻译成英文（这里用了其他人做的英汉互译的两个对应的txt），再扔进文本处理通道。两通道分别多双模态进行编码，提取特征，进行特征相似度计算，经过模型得到图片和文本之间的相似度矩阵，取每张图片对文本中相似度最大的作为模型预测的文本标签，训练过程中，将模型输出的每张图片对文本中相似度列表与训练集上给的正确结果转化的one-hot比对，用交叉熵做损失函数，训练网络。测试集就直接取每张图片对文本中相似度最大的作为模型预测的文本标签作为预测的tag结果。

2.1 dataset处理

因为给了总的json文件，直接读取json文件然后按照地址定位寻找到图片，读取即可，经过clip的图片编码通道后得到处理后的图片数据。可选标签经过汉译英之后，经过clip的文本处理通道，得到处理后的文本标签。而训练集上的图片的真实标签我将其按照可选标签的参考，转换为one-hot向量，即假设一共有3个可选标签，此图片的真是标签是第二个，则独热向量为$[0,1,0]$。一开始我将每张图片作为一个训练数据，但是在后来我发现，这样子是不恰当的。因为如果以每张图片作为一个训练数据，那么在后续的以batch输入网络进行训练的时候，如果batch大于1则会有可选标签不一致，独热向量有歧义的问题，这是不利于我们的训练的。但如果batch为1，整个网络的训练就会巨慢无比。最后我修改了getitem的复写，将同种商品的文件夹作为一个数据，里面包括了所有的图片和真实标签，这样子我在后续的网络训练中batch为1就可以对同一可选标签的进行训练，并且一定程度上提高了训练速度。

2.2 train部分数据的处理

一开始我将标注的数据集分为了训练集和验证集，比例为9：1。但是由于所给的数据实在是太大了，一开始跑一个eopch要好久好久，导致很久很久才能出现结果，而且也不一定能取得比较好的结果。经过考虑，每次训练从原本的训练数据中随机切出10%部分进行训练，加速网络的训练，可视化更好，将验证集每次也分出500个来做验证，加速整体运行的速度和效果。

```
for i in range(10):
    small_train_data, _ = random_split(train_data, [int(len(train_data) * 0.1), len(train_data) - int(len(train_data) * 0.1)])
    small_val_data, _ = random_split(val_data, [500, len(val_data) - 500])
    small_train_loader = DataLoader(small_train_data, batch_size=1, shuffle=True)
    small_val_loader = DataLoader(small_val_data, batch_size=1, shuffle=True)
    train(5, criterion=criterion, optimizer=optimizer, train_loader=small_train_loader, val_loader=small_val_loader)
```

3. 算法原理

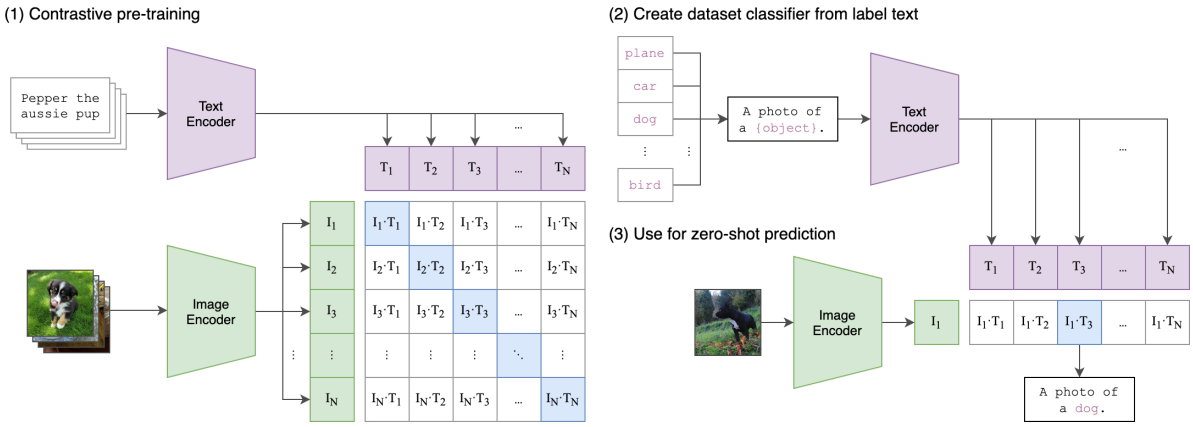
经过前期调研，最终选择clip模型，因为其就是利用text信息监督视觉任务自训练，本质就是将分类任务化成了图文匹配任务，模态处理和任务目标都非常贴合本次的大作业。

论文信息：**Title**: Learning transferable visual models from natural language supervision

作者：Alec Radford * 1 Jong Wook Kim * 1 Chris Hallacy 1 Aditya Ramesh 1 Gabriel Goh 1 Sandhini Agarwal Girish Sastry 1 Amanda Askell 1 Pamela Mishkin 1 Jack Clark 1 Gretchen Krueger 1 Ilya Sutskever 1

clip优势：1.因为本身也是采用的图文匹配，使用非常方便；2.clip使用了超过4亿个（高清图像、文本），本身就有很强的图文匹配能力。下面就具体模型和算法进行说明：

先上一个最经典的图，也是论文中的图。



可以看到模型是双模态输入，2个encoder分别处理文本和图片数据，text encoder使用Transformer，image encoder用了2种模型，ResNet和Vision Transformer(ViT)；encoder representation直接线性投影到multi-modal embedding space。接下来计算2模态之间的cosine similarity，让N个匹配的图文对相似度最大，不匹配的图文对相似度最小。即矩阵主对角线上即是相似度最匹配的图片和标签的配对。训练过程将模型输出和真实的独热码标签做交叉熵loss训练。

4. 实现过程

全局变量加载模型

$$model, preprocess = clip.load("ViT-B/32", device = device, jit = False)$$

首先复写dataset，核心代码转化为伪代码如下：

```
class Cloth_Dataset(Dataset):
    def __init__() :
        1.是否是训练数据isTrain
        2.数据存储的位置
        3.存储整个json文件的数据地址
        4.汉译英字典
    def __len__() :
        返回存储整个json文件的数据地址的大小
    def __getitem__(self, index):
        按照index读取字典中第index + 1个数据，进入数据文件地址，取出图片
        对每一张图片，应用preprocess编码，将其加入处理好的图片队列；对应的标签根据可选标签进行one-hot编码；
        对可选标签进行汉译英（提前以字典形式存储好），再进行clip.tokenize的Transform的编码，加入处理好的文本队列；
        返回元组（所有处理好的图片tensor，所有处理好的one-hot的tensor，处理后的可选标签）
```

从训练数据集读取数据，按照9：1比例随机划分训练集和验证集

核心train部分伪代码：

```
for epoch in range(epochs):
    从dataloder中取出1个作为batch训练
    扔进模型得到输出
    将其与真实one-hot编码的标签放进交叉熵loss
    然后用优化器更新参数
    计算准确率和loss

    输出本次epoch后的loss和acc结果，打印出来，每5次epoch绘制一次曲线
```

Test部分和train部分比较像，区别就是不用进行优化器更新参数。

5. 实验结果和分析

详细的实验结果可以看上交的.ipynh文件中的记录。

训练100个epoch（因为划分了数据集，其实相当于对大的数据集只训练了10遍）后的训练集准确率最高达到了：0.952，验证集上的准确率最高达到了0.946。并且从曲线上可以看出，其实在震荡上升的，过拟合不严重。

但是会发现虽然在训练集上的准确率还是以上升为主，但是验证集上准确率却有可能跌回0.93，这个时候我再训练了25个epoch（尽力了，没时间训了），发现基本上靠近了上限，估计再增加epoch数量大概只能提升1%之内，最后在网站的评测结果达到了0.951

最终网站上结果：

$$ACC = 0.9519, EM = 0.8413$$