学生实验报告

学号	1120192933	学院	计算机学院
姓名	李桐	专业	人工智能

搜索匹配与模型融合

1 实验目的

（1）理解 faiss 库的应用。
（2）理解搜索匹配的目的。
（3）理解搜索匹配和模型融合的过程。

2 实验原理

（1）faiss 库
（2）faiss 库中的 IndexFlatIP 函数

3 实验条件与环境

要求	名称	版本要求	备注
编程语言	python	3.6 以上	
开发环境	dsw	无要求	
第三方工具包/库/插件	sklearn	无要求	
第三方工具包/库/插件	tqdm	4.32	
第三方工具包/库/插件	faiss	无要求	

其他工具	无	无要求	
硬件环境	台式机、笔记本均可	无要求	

4 实验步骤及操作

步骤序号	1
步骤名称	faiss 搜索匹配
步骤描述	使用 faiss 减少搜索空间。

代码及讲解	faiss 是为稠密向量提供高效相似度搜索和聚类的框架。由 Facebook AI Research 研发，具有以下特性：提供多种检索方法、速度快、可存在内存和磁盘中、C++实现，提供 Python 封装调用、大部分算法支持 GPU 实现。
	使用流程大概是这样的:
	1.准备数据（train，query） 比如 train[10000, d], query[100, d]
	2.建立索引 index（多种方式可选）
	3.index.add(train)
	4.distance, index = index.search (query, k) 。 其中 k 是邻居数。train 和 query 可以相同，也可以不同。
	5.也可以加聚类（提升速度）。
	
	Demo 上分别进行暴力搜索以及聚类加速:
	`#_____`
	`#建立数据后`
	`#, 将维度d输入, 得到索引index`
	`index = faiss.IndexFlatL2(d)  # build the index  暴力检索, 欧氏距离.`
	`print(index.is_trained)`
	
	`#将train数据添加到索X引index中`
	`index.add(xb)  # add vectors to the index`
	`print(index.ntotal)`
	
	`#对query数据进行搜索得到D和I.`
	`k = 4  # we want to see 4 nearest neighbors`
	`#[nq, k]  对query的每行找到k个相应的distance和index`
	`D, I = index.search(xq, k)  # actual search (距离, ID)`
	`print(I[:5])  # 输出前5行`
	`print(D[-5:])  # 输出后5行`
	`#_____`
	
	`#_____`
	`nlist = 3  # 聚类质心`
	`quantizer = faiss.IndexFlatL2(d)  # IndexFlatL2 & IndexFlatIP -> 欧式距离 & 内积搜索`
	`index = faiss.IndexIVFFlat(quantizer, d, nlist, faiss.METRIC_L2)`
	
	`assert not index.is_trained`
	`index.train(xb)`
	`assert index.is_trained`
	
	`index.add(xb)`
	`# METRIC_L2 & METRIC_INNER_PRODUCT -> 欧式距离 & 内积搜索`
	
	`k = 4  # we want to see 4 nearest neighbors`
	`#[nq, k]  对query的每行找到k个相应的distance和index`
	`D, I = index.search(xq, k)  # actual search (距离, ID)`
	`print(I[:5])  # 输出前5行`
	`print(D[-5:])  # 输出后5行`
	`#_____`
	到了真正的搜索和模型融合阶段，代码是这样构成的:

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
# gpu数量
ngpu = torch.cuda.device_count()
def parse_args():
    parser=argparse.ArgumentParser(description='input checkpont')
    parser.add_argument('--checkponit',default='./results2/1-optimizer-Adam.pth')
    parser.add_argument('--I_json_path',default='./data3/i_train.json')
    parser.add_argument('--V_json_path',default='./data3/v_train.json')
    parser.add_argument('--I_path',default='./data3/I/')
    parser.add_argument('--V_path',default='./data3/V/')
    parser.add_argument('--output_path',default='./results2/topk.txt')

    args=parser.parse_args()

    return args
```

首先是承接输入。

```
def Query(q,g,topK,nlist=24):
    #nlist=# 聚类质心有多少
    d=q.shape[1]#维度
    quantizer = faiss.IndexFlatL2(d)  # IndexFlatL2 & IndexFlatIP -> 欧式距离 & 内积搜索
    index = faiss.IndexIVFFlat(quantizer, d, nlist, faiss.METRIC_L2)

    assert not index.is_trained
    index.train(q)
    assert index.is_trained

    index.add(q)
    # METRIC_L2 & METRIC_INNER_PRODUCT -> 欧式距离 & 内积搜索


    k = 4                          # we want to see 4 nearest neighbors
    #[nq, k]  对query的每行找到k个相应的distance和index
    D, I = index.search(g, k)     # actual search (距离, ID)
    # print(I[:5])    # 输出前5行
    # print(D[-5:])   # 输出后5行
    return I[:topK]
```

定义查询方法。

```
class MYDataset(Dataset):
    def __init__(self,jsonpath,path):
        """
        :参数 root_dir : 数据集所在文件夹路径
        :参数 img_dir  : 图片所在文件夹路径
        :参数 label_dir: 标签文件名
        :参数 enhancement: 数据增强的方法,包括None、以及方法1-7(int)
        """
        self.path='./data4/i_train.json'
        self.path=jsonpath

        self.rpath='./data4/I/'
        self.rpath=path

        with open(self.path, 'r') as f:
            self.x = json.load(f)

        #数据处理
        self.transform = transforms.Compose([
                transforms.Resize((224, 224)),
                transforms.ToTensor(),
                transforms.Normalize(mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0.225])])  # 归一化
```

构建数据集，对图片数据以及视频帧进行处理。

```
    model = nn.DataParallel(model, device_ids=list(range(ngpu)))
    # 准备完毕，开始测试了
    model.eval()
    feature_list=[]
    with torch.no_grad():
        for data in data_loader:

            images, true_labels = data
            images = images.to(device)
            true_labels = true_labels.to(device)

            # 前向传播
            model_feature = model(images)
            model_feature=model_feature.to('cpu')
            model_feature=model_feature.numpy()
            for i in model_feature:
                feature_list.append(i)



    return feature_list
```

构建函数，获得特征。

```
model= resnest50_m(pretrained=True)
for param in model.parameters():
    param.requires_grad=False
    model.fc=nn.Linear(model.fc.in_features,24)


    # parser.add_argument('--I_json_path')
    # parser.add_argument('--V_json_path')
    # parser.add_argument('--I_path')
    # parser.add_argument('--V_path')
bs=200
dataset_train = MYDataset(jsonpath=args.I_json_path,path=args.I_path)
dataset_val = MYDataset(jsonpath=args.V_json_path,path=args.V_path)

train_loader = torch.utils.data.DataLoader(dataset_train,batch_size=bs,shuffle=False,
                                           num_workers=4,pin_memory=True,drop_last=False)
val_loader = torch.utils.data.DataLoader(dataset_val,batch_size=bs,shuffle=False,
                                           num_workers=4,pin_memory=True,drop_last=False)

q=validation(data_loader=train_loader,model=model,checkpoint=args.checkponit)
g=validation(data_loader=val_loader,model=model,checkpoint=args.checkponit)
print('准备！')
ans=Query(q=q,g=g,topK=5)
np.savetxt(args.output_path,ans,delimiter=',') #frame: 文件 array:存入文件的数组
```

总体上，加载模型、构建数据集，将传入进来的图片、视频帧提取特征，然后进行搜索和匹配，如果传入多个模型，那就使用多个模型的结果进行融合。

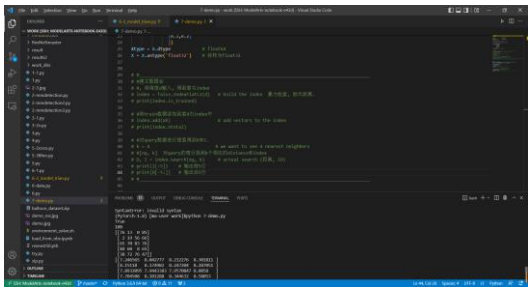
```
for i in check_list:
    q=validation(data_loader=train_loader,model=model,checkpoint=i)
    print('训练数据准备好了')
    g=validation(data_loader=val_loader,model=model,checkpoint=i)
    print('准备！')
    ans=Query(q=q,g=g,topK=2)
    ans_list.append(ans.reshape(-1))
np.savetxt(args.output_path,ans_list,delimiter=',') #frame: 文件 array:存入文件的数组
```

5 实验结果及分析讨论

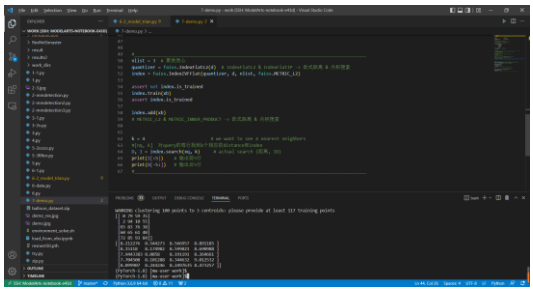
（1）最终结果的具体结果（文字说明）

成功进行了搜索、匹配以及模型融合。

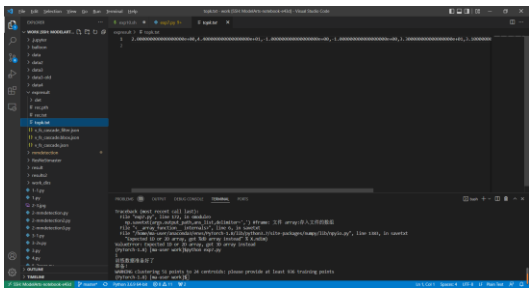
（2）最终结果界面截图（界面截图）



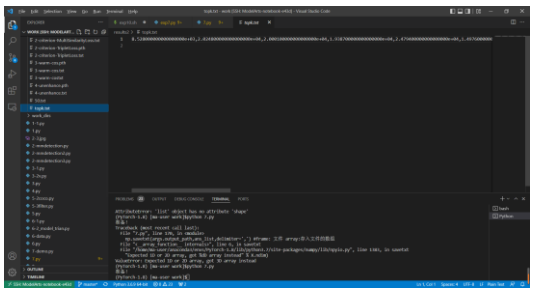
(1)demo 暴力搜索



(2)demo 聚类加速



(3)Demo 数据集



(4)大数据集

（3）最终结果的说明（注意事项或提醒）

上面两个图是我去理解 fassis 的时候做的一个 demo 的匹配。后两个是视频和图像的匹

配结果。

需要注意，为了获得中间的特征的输出，我更改了 resnest 的代码，返回的不再是标签，而是之前的特征。

（4）最终结果的解读与讨论

我先跑了一下 demo 数据集，然后才跑得大数据集。我的天呀，我发现消耗时间真的是指数级的上升。这还是开 GPU 呢，还用了这么久。

另外对老师的代码进行了一些修改，老师的代码是对照片寻找视频、对视频寻找照片。我感觉这样做的效果会好一些，但是其实不太方便理解，我一开始就没有懂老师的意思。其实可以这样去解释流程一下流程：先检测视频帧，有物体就进行识别，识别之后利用识别的特征在图片库进行匹配，最后得到想要的商品。这是我的理解，我觉得这样可能也会更方便其他同学理解一些。

另外因为我将 np 保存为了 txt，所以里面都是科学计数，哈哈哈，好蠢。改成 list，数字变成 int，再搞成 str 保存为 txt 效果应该会更好一些。

6 收获与体会

宏观上，理解了 faiss 库的应用、理解了搜索匹配的目的、理解了搜索匹配和模型融合的过程。并且由于分辨率的原因，这一部分基本都是自己写的。

还有一点小吐槽：

校园网下载这个库好慢啊。尤其是使用 smart art，每次进入环境都需要安装。又不能一次开太长时间，要不花钱很多。但是来回换 cpu、gpu 还要用好久时间安装环境。好不容易舍得开 gpu 了，结果安装了半个小时环境。

还有多个模型融合的时候也有一点问题，一开始写成了这样：

```
check_list=args.checkponit.split(',')
ans_list=[]
for i in check_list:
    q=validation(data_loader=train_loader,model=model,checkpoint=i)
    g=validation(data_loader=val_loader,model=model,checkpoint=i)
    print('准备！')
    ans=Query(q=q,g=g,topK=5)
    ans_list.append(ans)
    np.savetxt(args.output_path,ans_list,delimiter=',') #frame: 文件 array:存入文件的数组
```

但是这样的话，会报错：

```
WARNING clustering 51 points to 24 centroids: please provide at least 936 training points
Traceback (most recent call last):
  File "exp7.py", line 172, in <module>
    np.savetxt(args.output_path,ans_list,delimiter=',') #frame: 文件 array:存入文件的数组
  File "<__array_function__ internals>", line 6, in savetxt
  File "/home/ma-user/anaconda3/envs/PyTorch-1.8/lib/python3.7/site-packages/numpy/lib/npyio.py", line 1383, in savetxt
    "Expected 1D or 2D array, got %dD array instead" % X.ndim)
ValueError: Expected 1D or 2D array, got 3D array instead
```

因为是三维了，不符合我们的要求。所以进行了更改：

```
for i in check_list:
    q=validation(data_loader=train_loader,model=model,checkpoint=i)
    print('训练数据准备好了')
    g=validation(data_loader=val_loader,model=model,checkpoint=i)
    print('准备！')
    ans=Query(q=q,g=g,topK=2)
    ans_list.append(ans.reshape(-1))
np.savetxt(args.output_path,ans_list,delimiter=',') #frame: 文件 array:存入文件的数组
```

其实改法不难，但是我太蠢了，改了好久，用了一个贼蠢的方法。后来想到可以这样改，所以还是放在了这里。

7 备注及其他

无。