图像分类系统实验报告

王拓

学号：2021213490

班级：2021219112

一、算法流程说明

1. 使用 `opencv-python` 分别读取每个文件夹的图片，划分训练集和测试集，根据文件夹名记录其分类的 `lable`

特征提取和词袋生成：

2. 分别对训练集和测试集的图片提取 SIFT 特征点和其 `descriptors` 向量，对向量归一化。
3. 对于训练集的所有图片的 SIFT 特征向量进行 `MiniBatchKmeans` 聚类，以减少迭代次数，提高效率，最终获得 n 个聚类中心。
4. 将 n 个聚类中心作为视觉词汇，生成词袋。

词袋投票获得每个图片的向量表示

5. 使用 SPM 算法生成图片的向量表示

a. 将图片划分维不同的尺度：1*1，2*2，4*4，对于每个划分的区域，按照 L^2 距离统计特征直方图，组成一个 n 维的向量。
b. 将以上尺度的特征直方图对应的向量拼接，组成一个长度为 `（1+4+16）* n` 的行向量，作为这张图片的向量表示。
c. 以上操作要对训练集和测试集都进行向量表示。

执行分类

6. 将训练集的词袋向量表示作为输入，label 作为预测，使用 SVM 支持向量机进行分类，获得模型。
7. 使用训练好的 SVM 对测试集的词袋向量做预测，获得其对应的类别。
8. 对每一类的结果做分析，计算准确率。

二、函数功能说明和对应参数说明

具体代码见 `jupyter notebook`，此处只给出参数和功能说明

2.1 Image_SIFT 类

```
class Image_SIFT:
    def __init__(self,image,label: Optional[int] = None):
    def sift_transform(self):
    def sift_normalize(self):
    def show_sift_keypoints(self):
```

1. `__init__`：

功能：

- 将每个输入图片，转换为灰度图，初始化变量

参数：

- `image`：cv2 读取的图片
- `label`：对应的类别标签

2. `sift_transform`：

功能：

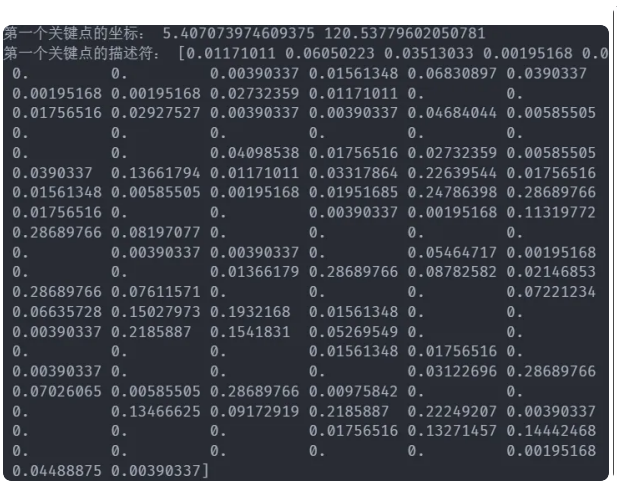
- 提取图片的 SIFT 特征，获得坐标和特征点向量

3. `sift_normalize`

```python
    def sift_normalize(self):
        normalized_descriptors = []
        for descriptor in self.descriptors:
            norm = np.linalg.norm(descriptor)
            if norm > 0:
                normalized_descriptor = descriptor / norm
            else:
                normalized_descriptor = descriptor
            normalized_descriptors.append(normalized_descriptor)
        self.descriptors = normalized_descriptors
        return normalized_descriptors
```

归一化向量的目的是为了提高算法的尺度不变性和匹配的准确性

以下是可视化结果：



2.1 load_image_get_set 加载数据集，进行训练集和测试集的划分

```python
def load_image_get_set(image_path: Optional[str] = None,
                       descriptors_save_path: Optional[str] ='./data/descriptors.dat'):
    """
    Args:
        image_path (Optional[str], optional): 文件夹目录. Defaults to None.
    """
    return train_set, test_set
```

功能：

- 加载图片，遍历文件夹，对前150张图片创建之前的类，根据类函数提取sift特征，然后加入到list，同理对 test_set 也进行相同的处理。
- 之后对这2250*feature特征聚类，得到聚类中心，之后对每张图片提取sift特征，然后用这个图片

的sift特征与聚类中心进行比较，得到这个图片的特征向量，之后用这个特征向量进行训练，**当然这是之后函数做的工作了**

结果：

```
(1133764, 128)
```

这是训练集的形状

2.3 BagOFVisualWords 类

```
class BagOFVisualWords:
    def __init__(self,
                 descriptors_list_path: Optional[str] = './data/descriptors.dat'):
    def kmeans_get_center(self,
                          descriptors_list: Optional[np.ndarray] = None,
                          k: Optional[int] =  200,
                          random_state: Optional[int] = 42,
                          center_save_path: Optional[str] = './data/center.dat'):
    def load_kmeans_center(self,
                           center_path: Optional[str] = './data/center.dat'):
    def spm_vote_get_vector(self,
                            image,
                            level: Optional[int] = 2,
                            point_descriptors: Optional[np.ndarray] = None,
                            point_keypoints: Optional[np.ndarray] = None,
                            image_path: Optional[str] = None
                            ):
    def process_dataset(self, data):
```

1. `__init__`：初始化一些类的变量
2. `kmeans_get_center`：

参数：

- `descriptors_list: Optional[np.ndarray]`：之前提取的所有训练集的descriptors向量。
- `k: Optional[int] =  200`：聚类中心的数量
- `random_state`：随机数种子
- `center_save_path: Optional[str] = './data/center.dat'`：保存路径

功能：

- 获取给定数量的聚类中心，以这些中心作为词袋，进行后续投票。

3. `spm_vote_get_vector` **使用 SPM 算法划分图像，进行词袋的投票，生成图片的词袋向量**

```python
def spm_vote_get_vector(self,
                        image,
                        level: Optional[int] = 2,
                        point_descriptors: Optional[np.ndarray] = None,
                        point_keypoints: Optional[np.ndarray] = None,
                        image_path: Optional[str] = None
                        ):
    """
    根据SPM算法，生成不同尺度下对我们视觉中心的投票向量，以此来代表图片
    """
    if self.center is None:
        raise ValueError("未初始化K均值中心。请先调用'kmeans_get_center'或'load_kmeans_center'方法。")
    num_of_vector = len(point_descriptors)  # 一个图片的特征点数量
    feature_vector2 = np.zeros((2**(level+2),self.k))  # 从小向大投票,
    image_y, image_x = image.shape
    step_x = math.ceil(image_x / (level**2))
    step_y = math.ceil(image_y / (level**2))

    # 开始对于每个向量投票
    for i in range(num_of_vector):
        x, y = point_keypoints[i].pt
        index_1 = math.floor(x / step_x) * (level**2) + math.floor(y / step_y)  # 第几个图片区域
        index_2 = np.argmin(np.linalg.norm(self.center - point_descriptors[i], axis=1))
        feature_vector2[index_1][index_2] += 1

    # 填充上一层
    feature_vector1 = np.zeros((2**level,self.k))
    for i in range(level**2):
        for j in range(level**2):
            feature_vector1[i] += feature_vector2[i*(level**2)+j]

    # 再填充一层
    feature_vector0 = np.zeros((1,self.k))
    for i in range(4):
        feature_vector0 += feature_vector1[i]
    feature_vector = np.hstack((feature_vector0.flatten(),feature_vector1.flatten(),feature_vector2.flatten()))

    return feature_vector
```

算法步骤：

- 针对每张图片，先把他划分成 4*4=16 的区域。
- 对于每个 SIFT 特征点，我们先判断他属于哪个 4*4 的区域。

- `index_2 = np.argmin(np.linalg.norm(self.center - point_descriptors[i], axis=1)` 计算这个特征点向量和哪个聚类中心距离最小，然后投票+1
- 对于所有的点都这么操作。获得 `feature_vector2`
- 接着把图片划分成 2*2=4 的区域进行同样的操作 `feature_vector1`
- 接着对整张图片进行同样的操作。 `feature_vector0`
- `feature_vector = np.hstack((feature_vector0.flatten(),feature_vector1.flatten(),feature_vector2.flatten()))` ： 把向量展平并拼接成一个向量。

参数:

- `level: Optional[int] = 2` 图片划分的层次
- `point_descriptors: Optional[np.ndarray] = None` ： 特征点向量
- `point_keypoints: Optional[np.ndarray] = None` 特征点

4. `process_dataset(self, data):` 批量处理所有图片

功能:

- 处理所有的图片的特征点和特征向量，生成 SVM 的训练向量和对应的标签

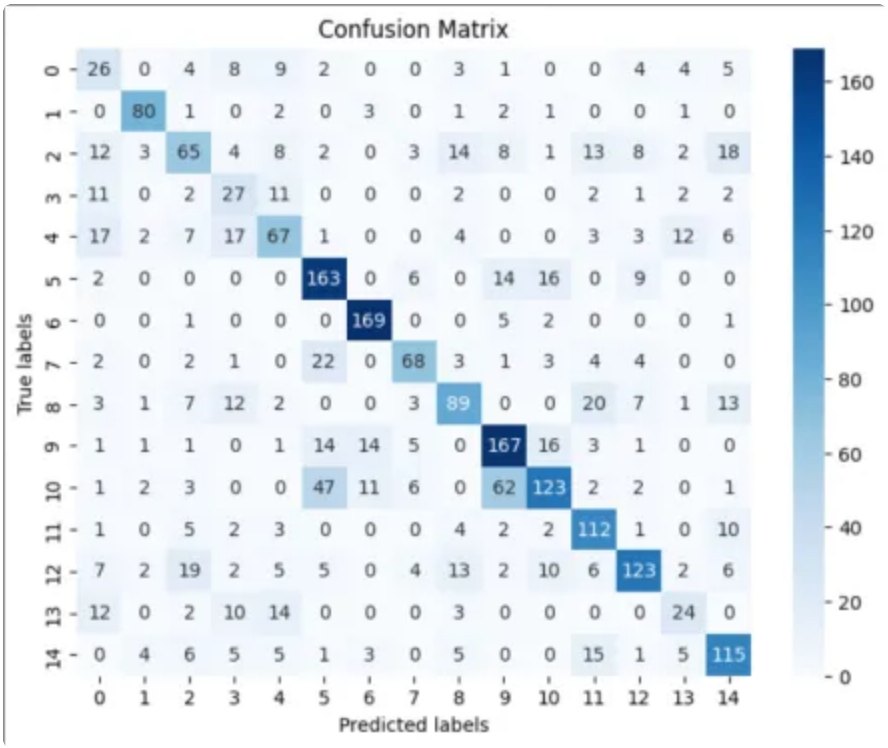
2.4 SVM_model 类

```
class SVM_model:
    def __init__(self):
        self.model = None

    def train(self,
              data: Optional[np.ndarray]=None,
              labels: Optional[np.ndarray]=None):
    def save_model(self,
                   model_path: Optional[str] = './data/model.dat'):
    def load_model(self,
                   model_path: Optional[str] = './data/model.dat'):
    def predict(self, data):
```

- SVM 类进行分类的训练
- SVM 进行模型的保存和加载
- SVM 进行标签的预测，都比较简单

三、结果的可视化和分析

3.1 混淆矩阵



3.2 详细信息

```
Classification Report:
              precision    recall  f1-score   support

           0       0.27      0.39      0.32        66
           1       0.84      0.88      0.86        91
           2       0.52      0.40      0.45       161
           3       0.31      0.45      0.36        60
           4       0.53      0.48      0.50       139
           5       0.63      0.78      0.70       210
           6       0.84      0.95      0.89       178
           7       0.72      0.62      0.66       110
           8       0.63      0.56      0.60       158
           9       0.63      0.75      0.68       224
          10       0.71      0.47      0.57       260
          11       0.62      0.79      0.70       142
          12       0.75      0.60      0.66       206
          13       0.45      0.37      0.41        65
          14       0.65      0.70      0.67       165

    accuracy                           0.63      2235
   macro avg       0.61      0.61      0.60      2235
weighted avg       0.64      0.63      0.63      2235

0.6344519015659955
```

最后的准确率是 0.6344519015659955

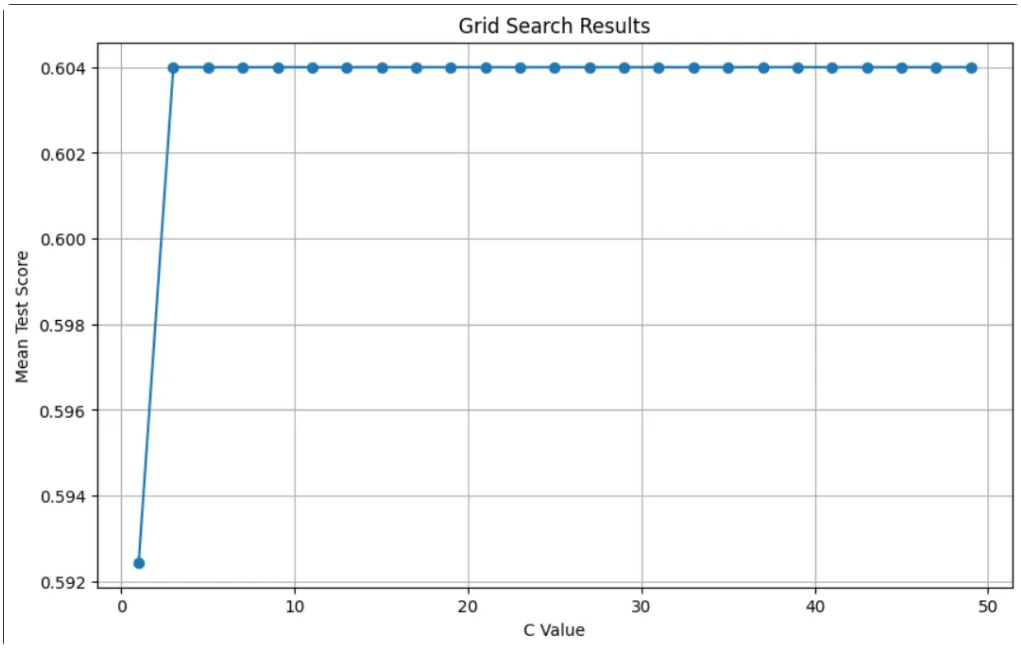
四、实验设置和超参数分析

4.1 随机选取聚类中心数量和 SVM 惩罚系数

聚类中心数量（词袋维度）	SVM 的惩罚系数 C	测试集准确率
k=64	1000	0.5825503355704698
k=100	1000	0.5843400447427293
k=200	1000	0.6098434004474272
	10	0.611185682326622
	15	0.6125279642058166
	c=20	**0.6138702460850112**
	c=30	0.611185682326622
	c=50	0.6120805369127517

4.2 对超参数 C 进行超参数进行【0-50，2】的超参数搜索

注意：在训练集上超参数搜索




```
开始训练模型
最佳C值： 3
```

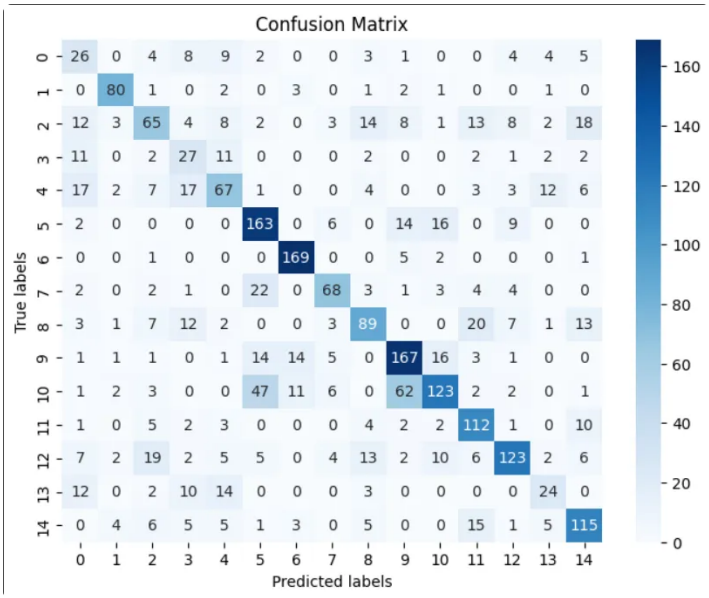
提高准确率非常关键的一步：

因为后来我反思了词袋投票的环节，大的图片所获得的投票肯定多，所以在拼接向量的时候应该归一化一下：

```
# 对每个特征向量进行归一化
normalized_feature_vector0 = feature_vector0 / np.linalg.norm(feature_vector0)
normalized_feature_vector1 = feature_vector1 / np.linalg.norm(feature_vector1)
normalized_feature_vector2 = feature_vector2 / np.linalg.norm(feature_vector2)

# 展平归一化后的特征向量
flatten_normalized_feature_vector0 = normalized_feature_vector0.flatten()
flatten_normalized_feature_vector1 = normalized_feature_vector1.flatten()
flatten_normalized_feature_vector2 = normalized_feature_vector2.flatten()

# 将展平后的特征向量进行拼接
feature_vector = np.hstack((flatten_normalized_feature_vector0, flatten_normalized_feature_vector1, flatten_normalized_feature_vector2))

```

想到这是因为当时学支持向量机做作业的时候做了这么一个归一化效果很好，所以在此尝试，果然又提升了几个点。

4.3 测试集表现




```
Classification Report:
              precision    recall  f1-score   support

           0       0.27      0.39      0.32        66
           1       0.84      0.88      0.86        91
           2       0.52      0.40      0.45       161
           3       0.31      0.45      0.36        60
           4       0.53      0.48      0.50       139
           5       0.63      0.78      0.70       210
           6       0.84      0.95      0.89       178
           7       0.72      0.62      0.66       110
           8       0.63      0.56      0.60       158
           9       0.63      0.75      0.68       224
          10       0.71      0.47      0.57       260
          11       0.62      0.79      0.70       142
          12       0.75      0.60      0.66       206
          13       0.45      0.37      0.41        65
          14       0.65      0.70      0.67       165

    accuracy                           0.63      2235
   macro avg       0.61      0.61      0.60      2235
weighted avg       0.64      0.63      0.63      2235

0.6344519015659955
```

4.4 分析

最后我选择词袋数量 N=200,SVM 的惩罚系数为 3。

词袋数量：增加词袋数量可以更好地表现图片特征，带来更细粒度的特征表示，有助于提高模型的泛化能力和性能，精度更高。

SVM 的惩罚系数：通过在**训练集**的超参数搜索和交叉验证，我发现 C=3 的准确率最高，在 C 值较大时，模型对训练数据的拟合程度更高，可能会过拟合；而 C 值较小时，模型对训练数据的拟合程度较低，可能会欠拟合。因此支持向量机的 C 选择也比较重要。