第十四周报告

1. 实验概览

本实验旨在于学习 LSH（Locality-sensitive Hashing）的基本原理，理解 LSH 在检索中的优势和思想，并在实验中实现对于图片的预处理、哈希函数和检索过程。

2. 实验环境

Docker: sjtumic/ee208

3. 解决思路

a) 图片的预处理

- **提取颜色直方图**

首先对图片分块处理

```
height, width, _ = np.shape(img)
pts = []
pts.append(img[0:(height // 2), 0:(width // 2)])
pts.append(img[0:(height // 2), (width // 2):])
pts.append(img[(height // 2):, 0:(width // 2)])
pts.append(img[(height // 2):, (width // 2):])
```

基于之前 lab 的结果，对图片进行分块提取颜色直方图拼接为 12 维图片特征

```
    feature = []
    for pt in pts:
        col = []
        col.append(
            sum(i * cv2.calcHist([pt], [0], None, [256], [0, 256])[i] for i in range(256))[0])
        col.append(
            sum(i * cv2.calcHist([pt], [1], None, [256], [0, 256])[i] for i in range(256))[0])
        col.append(
            sum(i * cv2.calcHist([pt], [2], None, [256], [0, 256])[i] for i in range(256))[0])
        col /= sum(col)

        feature.append(col[0])
        feature.append(col[1])
        feature.append(col[2])

    return compute_one_hot(feature)
```

根据特征每一维特征值整数赋值得到最终特征

```
def compute_one_hot(array):
    res = []
    for i in array:
        if i < 0.3:
            res.append(0)
        elif i < 0.6:
            res.append(1)
        elif i > 0.6:
            res.append(2)

    return res
```

- **利用 HOG 算法进行特征提取**

实验中，作为补充，基于 skimage 库利用了 hog 方法对图片计算特征，得到 10000 位的图像编码作为特征

```
def get_feature2(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    feature = skimage.feature.hog(img)
    feature = compute_one_hot2(feature)
    return feature[:10000]
```

映射到整数域函数如下：

```
def compute_one_hot2(array):
    max_val = max(array)
    res = []
    for i in array:
        if i < max_val / 3:
            res.append(0)
        elif i < max_val * 2 / 3:
            res.append(1)
        elif i > max_val * 2 / 3:
            res.append(2)

    return res
```

b) 构建 LSH 算法

- **定义 hash 函数**

根据参考得到 hash 函数，其中投影集合在每次初始化时随机取得

```
def __init__(self, tables_num: int, C: int):
    self.tables_num = tables_num
    self.C = C
    self.num = []
    for i in range(C + 1):
        self.num.append('1' * i + '0' * (C - i))
    self.hash_tables = [dict() for _ in range(tables_num)]
    self.I = np.random.randint(12, size=int(np.log2(tables_num)))

def hash(self, input):
    project = ''.join([self.num[input[i]] for i in range(len(input))])
    hash_val = ''.join([project[i] for i in self.I])
    hash_val = int(hash_val, base=2)

    return hash_val
```

- **定义插入数据和查询操作**

```python
def insert(self, inputs, index):
    inputs = np.array(inputs)

    hash_index = self.hash(inputs)
    self.hash_tables[hash_index].setdefault(tuple(inputs.tolist()), []).append(index)


def query(self, inputs, nums=3):
    inputs = np.array(inputs)
    hash_val = self.hash(inputs)
    candidate_dict = self.hash_tables[hash_val]

    candidates = sorted(
        candidate_dict, key=lambda x: self.euclidean_dis(x, inputs))

    res = []
    for i in candidates[:nums]:
        res += candidate_dict[i]

    return res[:nums]
```

其中，每一类初始化为字典是防止出现 feature 相同的情况（以颜色直方图作为特征容易发生）；利用特征之间的欧几里得距离作为定义图像相似度的指标

4. 代码运行结果

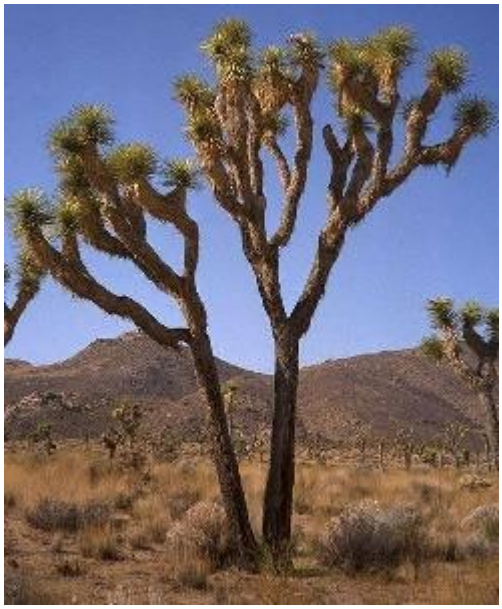
基于颜色直方图的结果:

```python
tg = cv2.imread('./target.jpg')
feature = get_feature(tg)
res = np.array(lsh.query(feature))
print(res)
```

```
['12.jpg' '38.jpg' '23.jpg']
```

其中 38.jpg 和 target.jpg 图像一致

而 12.jpg 图像内容和 target 为同一类，且整体颜色分布和 target 相似度较高，可以发现以颜色直方图作为特征，将提取出和目标图片颜色分布类似的图片，但无法对细节进一步细化



基于 HOG 特征提取结果：

target.jpg

```
['38.jpg' '7.jpg' '11.jpg']
```

在实验中对于数据库中每一张图都进行了检索操作，整体上 HOG 得到的特征可以将数据库中的图片较好地区分开来