

# 实验 3 LSH

姓名 学号 班级

日期

## 1 实验概览

### 1.1 实验内容

LSH, 即 **L**ocality-**S**ensitive **H**ashing (局部敏感哈希)。本次实验要求 LSH 算法在图片数据库中搜索与目标图片最相似的图片, 并与暴力 NN 的结果与运行时间进行对比。

### 1.2 实验原理

在本次实验中, 我们统一使用 12 维向量  $\mathbf{p}$  来表示图像特征。

将图像分割为左上、右上、左下、右下四个大小相等的子图像, 每个子图像的颜色通道占比为 3 维向量  $(r, g, b)$ 。将四个子图像的 3 维向量拼接, 得到  $\mathbf{p}$ 。

对于暴力 NN, 直接求解最近的向量

$$k_{\text{match}} = \arg \max_k \langle \mathbf{p}_{\text{target}}, \mathbf{p}_k \rangle.$$

对于 LSH, 我们做如下处理。

首先, 我们将向量进行量化。

$$p_{i,\text{quantized}} = \begin{cases} 0, & 0 \leq p_i < 0.3, \\ 1, & 0.3 \leq p_i < 0.6, \\ 2, & 0.6 \leq p_i. \end{cases}$$

然后, 我们将向量转化为 Hamming 码表示。由于  $\mathbf{p}_{\text{quantized}}$  是 12 维 3-bit 向量, 我们得到一个  $12 \times (3 - 1) = 24$  位的 Hamming 码  $v(\mathbf{p})$ 。

我们取  $\{1, 2, \dots, 24\}$  的  $L$  个子集  $\{I_i\}_{i=1}^L$ , 其中  $I_i = \{i_1, i_2, \dots, i_m\}$ 。定义  $\mathbf{p}$  在  $I_i$  上的投影  $g_i(\mathbf{p}) = v_{i_1} v_{i_2} \dots v_{i_m}$ , 其中  $v_{i_j}$  表示  $v(\mathbf{p})$  的第  $j$  位。

我们认为与目标图片的特征向量相似的特征向量有很大的概率满足:

$$\exists i : g_i(\mathbf{p}) = g_i(\mathbf{p}_{\text{target}})$$

于是, 我们对所有在任一个子集上的投影与目标图片特征向量在该子集上的投影相等的特征向量运行暴力 NN, 以此减少运算数量。

## 2 解决思路

计算用于表示图像特征的 12 维向量。

```
def generate_feature_vector(img: cv2.typing.MatLike):
    h, w, _ = img.shape
    vector = []
    for i in range(4):
        crop_img = img[(i // 2) * h // 2 : (((i // 2) + 1) * h) // 2, (i %
        ↪ 2) * w // 2 : (((i % 2) + 1) * w) // 2]
        totals = crop_img.sum(axis=(0, 1)).astype('float64')
        vector.extend(totals / totals.sum())
    return np.array(vector)
```

对于 NN，直接暴力求解距离，返回最近的结果。

```
def match(self):
    distances = []
    for index, d_vec in enumerate(self.dataset_vectors):
        dst = get_distance(self.target_vector, d_vec)
        distances.append((index, dst))
    distances.sort(key=lambda x: x[1])
    return distances[0][0]
```

对于 LSH，为了避免重复运算，总是先进行量化操作，并计算 Hamming 码表示。

```
def _quantize_feature_vector(self, feature_vector: np.ndarray):
    v = feature_vector.copy()
    v[v >= 0.6] = 2
    v[v < 0.3] = 0
    v[(0.3 <= v) & (v < 0.6)] = 1
    return v.astype(np.int64)

def _get_hamming(self, feature_vector: np.ndarray):
    hamming = []
    for p in self._quantize_feature_vector(feature_vector):
        hamming.extend([1] * p + [0] * (2 - p))
    return np.array(hamming)
```

```
dataset_hamming = [self._get_hamming(vec) for vec in self.dataset_vectors]
target_hamming = self._get_hamming(self.target_vector)
```

得到 Hamming 码表示后，我们计算每个子集  $I_i$  上的投影。

```
def _get_hash_vector(self, hamming: np.ndarray, subset: tuple):
    return tuple(hamming[subset])

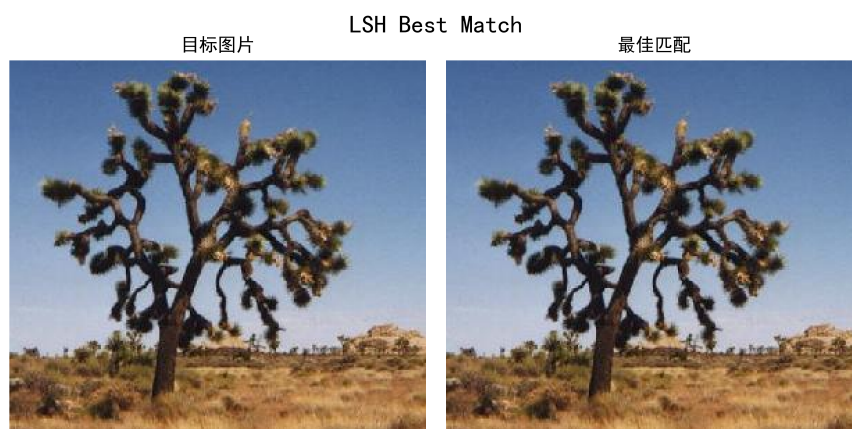
for proj in self.projectors:
    dataset_hashes = [self._get_hash_vector(ham, proj) for ham in
        ↪ dataset_hamming]
    target_hash = self._get_hash_vector(target_hamming, proj)
```

对在任一子集上投影与目标投影相同的，加入候选。

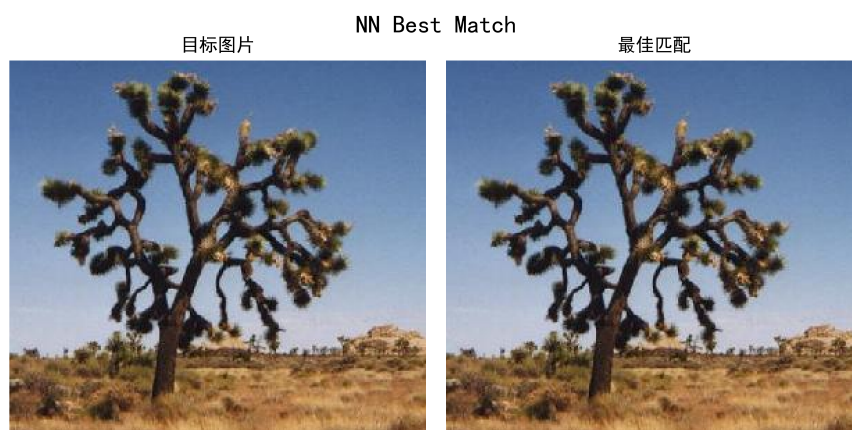
```
for index, _hash in enumerate(dataset_hashes):
    if _hash != target_hash:
        continue
    candidates.add(index)
```

在候选图像中运行暴力 NN 算法，返回最近的结果。

### 3 运行结果



(a) LSH 匹配结果



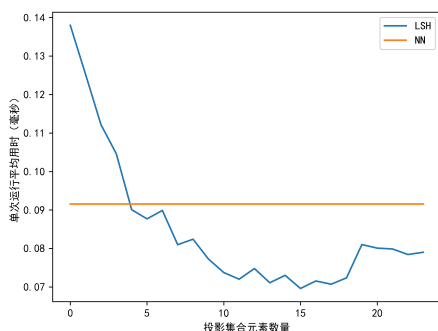
(b) NN 匹配结果

图 1: 运行结果

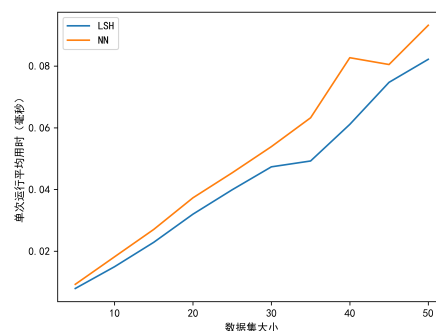
由于目标图像与数据库中的一张图像是完全一致的，匹配结果对于任何投影集合都必定是正确的。

下面我们取子集数量  $L = 1$ ，并对探究数据库大小、投影集合元素数量对 LSH、NN 运行时间的影响。为了排除干扰，我们随机地取投影集合，并运行多次取平均值。

LSH 的运行时间主要分为三部分：计算 Hamming 码、计算投影并选取候选、暴力匹配最近向量。其中计算 Hamming 码这一步是离线的（可预计算的）；计算投影这一步由于投影向量可能会变动，不能看作离线；暴力匹配是在线的。为了与 NN 的比较更贴近应用场景，我们在运行时间中去掉计算 Hamming 码这一步的耗时。



(a) 运行时间与投影集合元素数量的关系



(b) 运行时间与数据集大小的关系

图 2: 运行时间影响因素分析

## 4 结果分析与思考

### 4.1 结果分析

从结果来看, LSH 和 NN 都可以准确地匹配出正确的图像。

观察运行时间与投影集合元素数量的关系图, 可以发现, 投影集合元素数量越多, 运行时间越短, 在  $k = 7$  之后保持不变。这是由于投影集合元素数量越多, 同一哈希桶内图像数量越少, 进而需要进行的距离计算也越少。同时, 这可能会使相似的图像落入不同的哈希桶, 进而降低匹配准确度。受限于实验提供的数据集, 我们不对准确性进行测试。在  $k < 4$  附近, 运行时间实际劣于 NN 算法, 这是由于计算投影和选择候选导致了额外的时间开销。在  $k > 20$  附近, 运行时间呈上升趋势, 这是由于需要进行的距离计算不再减少, 但参与计算的 Hamming 码位数变多。

观察运行时间与数据集大小的关系图 (取  $k = 7$ ), 可以发现, 在  $n < 50$  处, 运行时间与数据集大小大致呈线性关系, LSH 的运行时间总是优于 NN 的运行时间。受限于实验提供的数据集, 无法对更大的数据集进行测试。从理论角度来看, NN 算法、LSH 算法的复杂度都应当是关于数据集大小线性的。

### 4.2 思考

1. 本练习中使用了颜色直方图特征信息, 检索效果符合你的预期吗? 检索出的图像与输入图像的相似性体现在哪里?

检索效果符合预期。检索出的图像与输入图像完全一致; 从数据库中去掉完全一致的图像后, 检索出的图像在场景上非常类似: 黄沙、蓝天、树。从原理上看, 两张图像的色调也是接近的。



(a) 目标图片



(b) 最佳匹配

图 3: 去除完全相同的图片后的最佳匹配

## 2. 能否设计其他的特征?

可以使用梯度特征、角度特征等等。

## 5 实验感想

通过本次实验，我初步了解了 LSH 及其中的哈希思想，完成了 LSH 于图像匹配中的应用。同时，我将 LSH 与 NN 进行多方面的对比，并进行简单分析，初步掌握对比算法优劣的方法。