Exp2.计算机视觉实验报告

一、实验环境：Python2.7

一、分工

[redacted]：TODO1~4、选做非最大抑制算法设计、代码优化

[redacted]：TODO5~8、非最大抑制代码优化

工作量比例大致 1:1

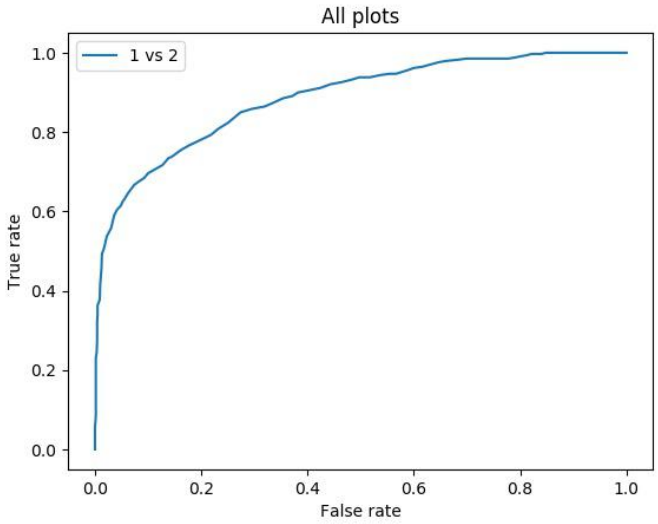
三、实验结果

（一）统一阈值

Simple ssd

Threshold = 10^-2.0

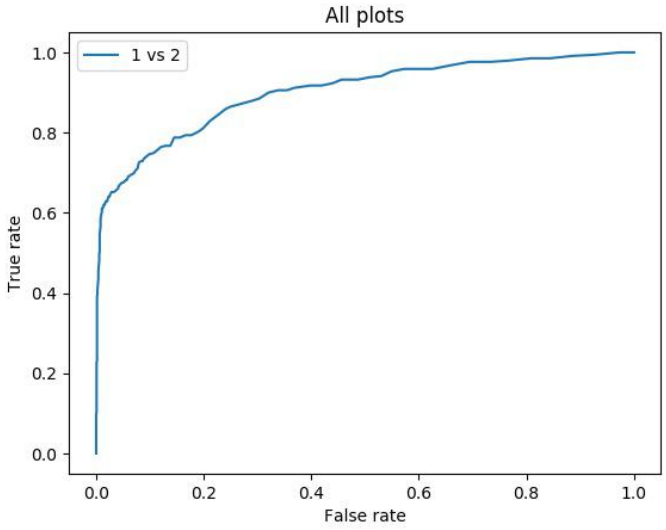
```
Average distance between true and actual matches: 269.805777021;        Average AUC: 0.885511871947
```



Simple ratiotest

Threshold = 10^-2.0

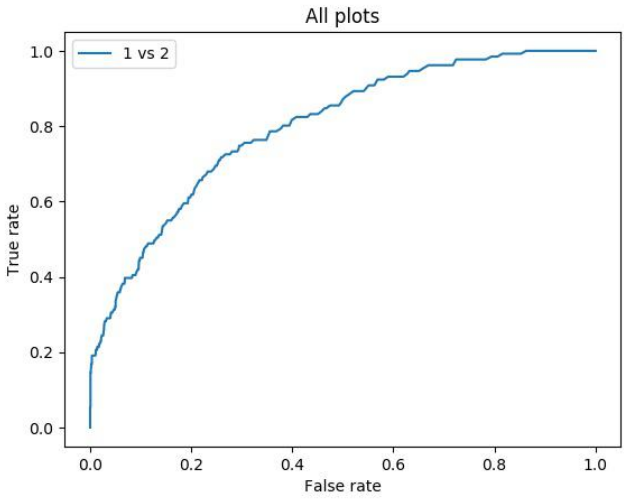
```
Average distance between true and actual matches: 269.805777021;        Average AUC: 0.900708448184
```



Mops ssd

Threshold = 10^-2.0

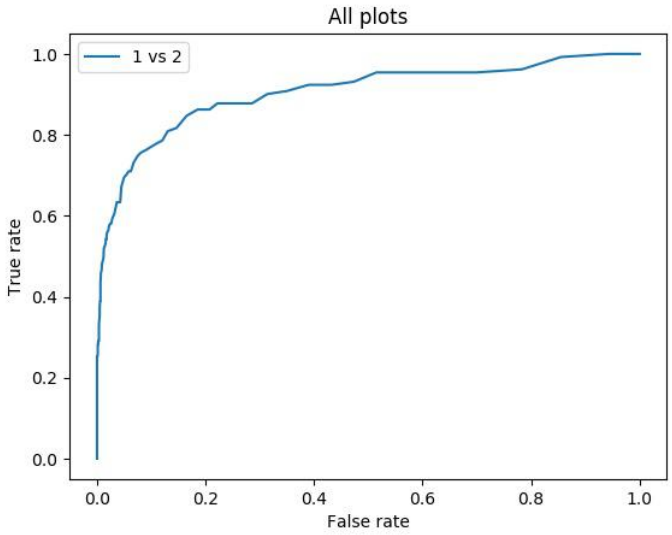
Average distance between true and actual matches: 332.872162991; Average AUC: 0.798800436205



Mops ratiotest

Threshold = 10^-2.0

Average distance between true and actual matches: 332.872162991; Average AUC: 0.903853144311



结果分析

1. 在 Threshold = 10^-2.0 的条件下：simple 和 mops 两种描述符均表明：比率测试的匹配效果均明显优于 SSD，AUC 提高了 0.087,记

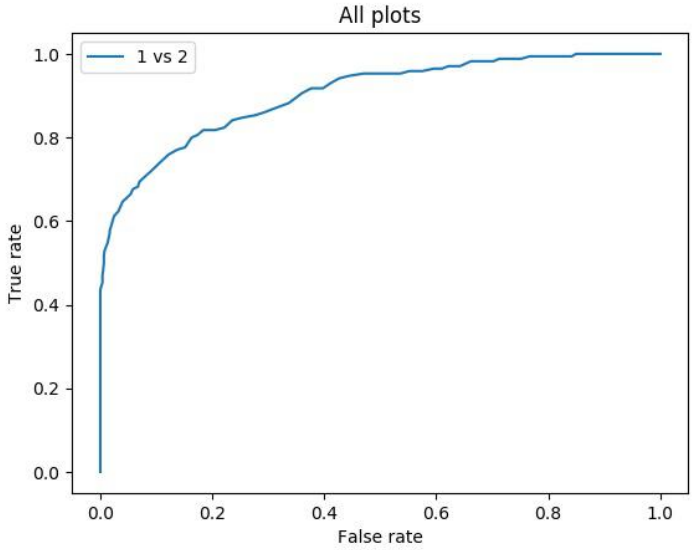
2. 在 Threshold = 10^-2.0 的条件下：应用 SSD 匹配算法，simple 描述符明显优于 mops 描述符；而应用比率测试匹配算法，mops 描述符较 simple 描述符稍有优势，AUC 提高了 0.0031

（二）、最优阈值

Simple ssd

Threshold = 10^-1.5

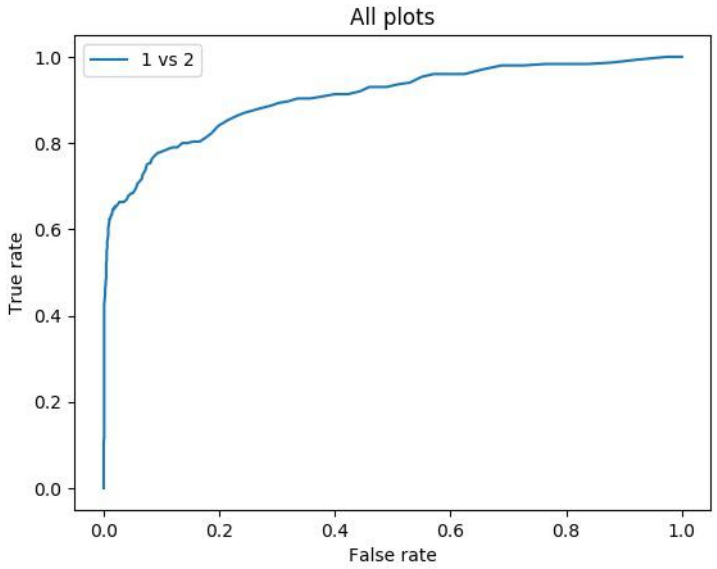
Average distance between true and actual matches: 302.192677072; Average AUC: 0.902617342274



Simple ratiotest

Threshold = 10^-1.9

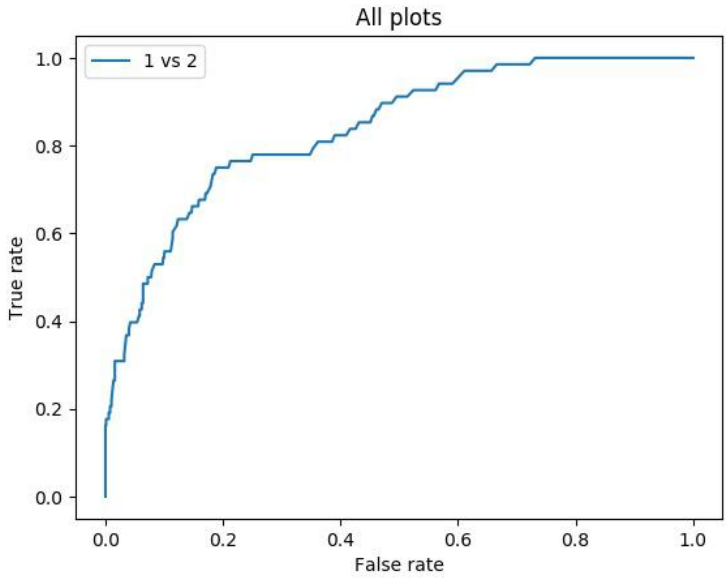
Average distance between true and actual matches: 274.820941733; Average AUC: 0.90657594381



Mops ssd

Threshold = 10^-1.5

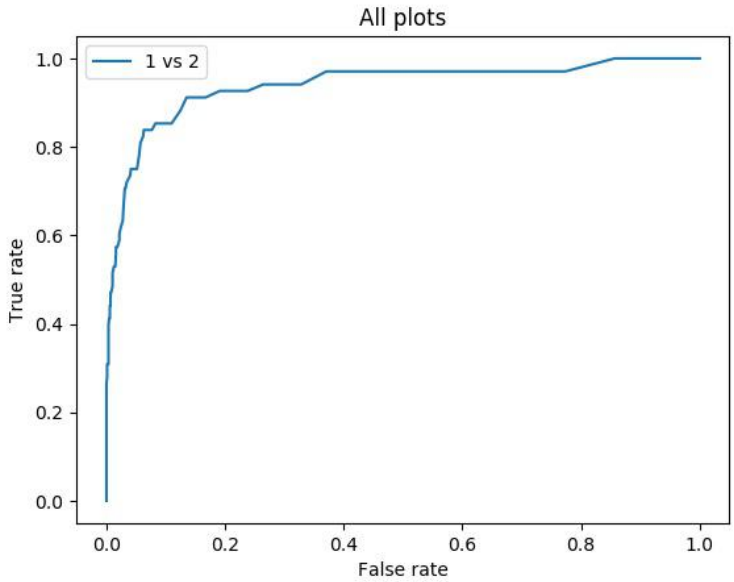
Average distance between true and actual matches: 369.342842227; Average AUC: 0.838496096379



Mops ratiotest

Threshold = 10^-1.5

Average distance between true and actual matches: 369.342842227; Average AUC: 0.938804011307

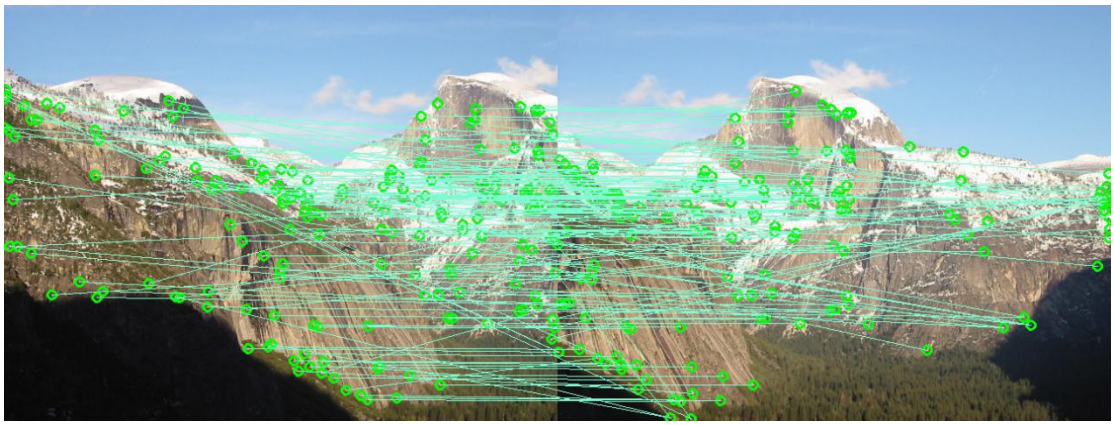


结果分析：

分析指标同上，一.1 的分析结果由 0.087 下降为 0.064，一.2 的分析结果由 0.0031 提高到 0.032.

（三）匹配结果

参数 Match percent：11.4%



四、选做

1、选做题目（2）：非最大抑制

2、问题分析：

MOPS 论文中提到用自适应非极大值抑制算法对 Harris 角点检测所得到的大量关键点做进一步筛选，以达到①减少关键点、②尽可能保持角点响应大的点、③关键点分布均匀这三个目的

3、算法设计

设置筛选指标：响应优度，用以衡量每个角点的优先级

Step1：

得到 Harris 角点检测结果集 X，X 大小为 N，设置选择系数 c、d

Step2：

寻找全局最大的角点响应值，即 fmax=max（f）

Step3：

遍历 X，计算每个角点的响应优度 ri：

$$r_i = \begin{cases} \min_j \left(\| x_i - x_j \| \right), & f(x_i) \le c \cdot f_{\max} \wedge f(x_i) < c \cdot f(x_j) \\ \infty, & f(x_i) > c \cdot f_{\max} \end{cases}$$

Step4：

对 ri 进行降序排序、选择前 d*N 个 ri 对应的角点作为返回的结果集

Step5：

返回 step1，直到对筛选效果满意，确定系数 c、d

4、核心代码

```
def anms(self, keypoints):
        max_h = 0.0
 r = []
 feature = []
```

```
    x1 = np.zeros((n, 2))
    for i, f in enumerate(keypoints):
        x1[i] = [f.pt[0], f.pt[1]]
    x2 = x1.copy()

    d = scipy.spatial.distance.cdist(x1, x2)

    for i, f in enumerate(keypoints):

        if f.response > max_h :
            max_h = f.response
            continue

    for i, f1 in enumerate(keypoints):
        #print(i)
        rad = float('inf')
        if f1.response >c*max_h:
            r.append([i,rad])
            continue
        for j, f2 in enumerate(keypoints):
            if j == i : continue
            if (f1.response<c*f2.response):
                d1 = d[i][j]
                if d1<rad:
                    rad=d1
            r.append([i,rad])

    r = sorted(r,key = lambda x:x[1],reverse=True)

    length =d*len(r)
    for i, f in enumerate(r):
        if i>length:break
        feature.append(keypoints[f[0]])

    return feature
```

5、代码&算法分析

因为本算法需要对 xi、xj 的响应进行条件判断，所以双层 for 循环不可避免，输入规模很大的 Harris 结果集会产生极大开销，所以要应用此算法，必须对输入集进行过滤，降低其规模，可选择的方案有：随机过滤、优先级过滤。我们选择了优先级过滤，即把不小于输入集响应强度均值的特征点留下，作为过滤后的输入集，添加代码如下：

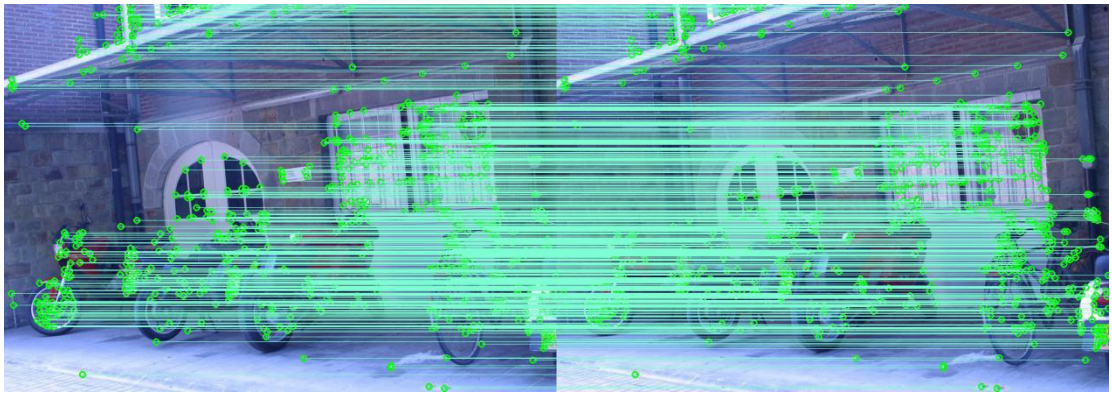
```
    def anms(self, keypointss):
    # print(len(keypointss))
     keypoints = []
     n = len(keypointss)
```

```
    keyharris = np.zeros((n, 1))
    for i, f in enumerate(keypointss):
            keyharris[i] = f.response

    harrismean = np.mean(keyharris)
    harrisstd = np.std(keyharris)

    for i, f in enumerate(keypointss):
            if f.response>=harrismean:
                    keypoints.append(f)
```

6、运行结果

（1）非最大抑制

（2）不进行非最大抑制



结果分析：

（1）因为该算法依据响应强度进行了输入预处理，所以图中只留下了响应较强的特征点，如图片上方亮度差异大的区域。

（2）将引入非最大抑制前后的匹配结果作对比：抑制后，图片上方特征点变得均匀、稀疏，达到了目的。

7、算法优缺点

缺点：该算法的输入集不能过大，实验而得：输入的特征点超过 2000 时，cpu 被完全占用。

优点：对于可计算的输入，该算法达到了预设的目标。

注：提交的 feature.py 文件中，anms（）函数的调用已被注释