计算机视觉作业 -3：模板匹配技术

王天锐　20120318

2020 年 12 月 16 日

摘要

本次实验作业主要是根据老师上课所讲知识，利用 1. 相关匹配 (Correlation Matching)、2. 基于 Hausdorff 距离匹配方法；以及 3. 考虑对场景图像距离变换 (Distance Transform) 的 Hausdorff 距离匹配方法，实现模版目标在场景图像中的定位。对于每个模版分别给出最后的目标定位位置坐标 (左下角坐标为 (0,0))，对于方法 1 和 2 对比定位精度的偏差；对于方法 2 和 3 比较定位效率。

1　方法原理及实现

1.1　相关匹配算法

设场景图片为 S，模版图片为 T。由于是计算相关性，所以需要中心化处理:

$$S_{ij}^{*} = S_{i,j} - \overline{S}$$

$$T_{ij}^{*} = T_{i,j} - \overline{T}$$

然后对中心化后的场景图片 S 做模版等大窗的滑窗处理，相应的代码实现为:

```
# 中心化
template = template - np.mean(template)
scene = scene - np.mean(scene)
template_h, template_w = template.shape
scene_h, scene_w = scene.shape
# 分窗
strides = scene.itemsize * np.array([scene_w, 1, scene_w, 1])
new_h = scene_h - (template_h - 1)
new_w = scene_w - (template_w - 1)
win_wrap_img = np.lib.stride_tricks.as_strided(scene, shape=(new_h, new_w, template_h, template_w),
strides=strides)
```

然后计算每个窗和模版之间的相关性，相关性计算公式为：

$$correlation = \frac{\sum_{i=1}^{m}\sum_{i=1}^{n} S_{i,j}^{*} T_{i,j}^{*}}{\sqrt{\sum_{i=1}^{m}\sum_{i=1}^{n} {S_{i,j}^{*}}^{2} \sum_{i=1}^{m}\sum_{i=1}^{n} {T_{i,j}^{*}}^{2}}}$$

其代码实现为:

```
def corr(frame1, frame2):
        return np. sum(frame1 * frame2) / np.sqrt(np. sum(frame1 ** 2) * np. sum(frame2 ** 2))
```

最后选择出最大相关性的窗口作为匹配结果。

1.2 Hausdorff 距离匹配方法

Hausdorff 距离是用来度量两个点集之间的最大不匹配程度，设集合 A=$\alpha_1, \ldots, \alpha_n$,B=$\beta_1, \ldots, \beta_m$，Hausdorff 距离计算公式为:

$$h(A,B) = \max_{\alpha \subseteq A}\{\min_{\beta \subseteq B}\{D(\alpha,\beta)\}\}$$

$$H(A,B) = \max\{h(A,B), h(B,A)\}$$

$h(A,B)$ 是首先计算出 A 集合中的每个点 α_i 距离 B 集合中的最近的 β_i；然后选出对应最小距离最大的 α_i 的距离作为 $h(A,B)$ 的值。

本实验距离采用欧式距离：

$$D(\alpha,\beta) = (\alpha - \beta)^2$$

Hausdorff 距离计算代码为:

```
def hausdorff_dis(frame1, frame2):
        frame1_2 = np. min(
        (
                 abs(frame1.reshape([-1, 1]).repeat(frame2.shape[0], 1) - frame2)
                ).reshape([ len(frame1.flatten()), -1]), axis=-1
        )
        frame2_1 = np. min(
        (
                 abs(frame2.reshape([-1, 1]).repeat(frame1.shape[0], 1) - frame1)
                ).reshape([ len(frame2.flatten()), -1]), axis=-1
        )
        return np. max((np. max(frame1_2), np. max(frame2_1)))
```

1.3 图像距离变换

图像距离变换主要是针对二值化后的图像进行处理。给每个像素赋值为离它最近的背景像素点与其距离，从而得到距离矩阵，本实验采用绝对值距离作为度量标准，对应距离变换计算公式为:

$$dis_{i,j} = |i - a| + |j - b|$$

其中 a 和 b 是指离当前计算点最近的边缘点的横纵坐标值。代码实现为:

```
def hausdorff_distance_trans(bk_edge):
        row, col = bk_edge.shape
        min_dis = (row + col) * np.ones((row, col))
        a, b = np.where(bk_edge == 255)
        for i in  range(row):
                for j in  range(col):
                        min_dis[i, j] = np. min( abs(a - i) +  abs(b - j))
        return min_dis
```

2 实验结果及分析

2.1 原始样图展示

下列是原始样图，我们的目标是在场景图像 1 中定位模版目标 2 和 3。

图 1: 场景图

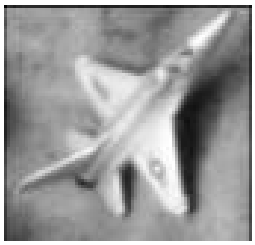

图 2: 模版 1

图 3: 模版 2

2.2 相关匹配

下图是两个模版在整张图上相关性的分布示意图 (分布图是逆时针旋转了 90°)：

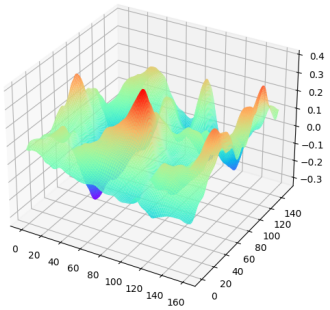

图 4: 模版 1 的相关值分布

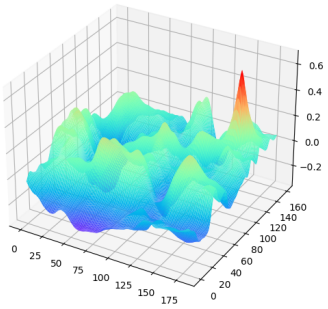

图 5: 模版 2 的相关值分布

选取出其中的最大值，得到对应的坐标。

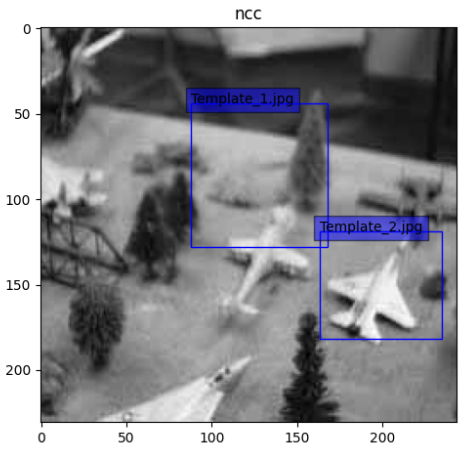

图 6: 相关匹配算法结果

```
左下角为(0,0)情况下,模版1和2目标框的左上角坐标分别为:
 [[187, 88], [112, 163]]
ncc耗时: 6.651214361190796
```

图 7: 左上角坐标和运行耗时

从结果可以看出模版 2 成功匹配上了，但是模版 1 没有匹配上。分析原因是因为模版 1 是将一个相似内容进行了一定的旋转，而相关性计算分子是对应坐标位置相乘，则对像素点分布有一定的要求，可能抗旋转性较差，最终导致匹配模版 1 失败。

2.3 Hausdorff 距离匹配方法

在该小节实验中，首先使用了 Canny 算子来提取图像的边界信息。图 8、图 9 和图 10 分别是模版 1、模版 2 和场景的 Canny 算子提取结果:

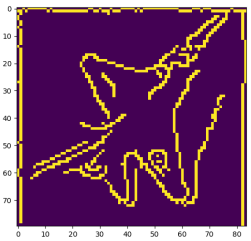

图 8: 模版 1 边界提取结果

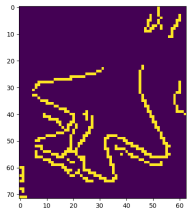

图 9: 模版 2 边界提取结果

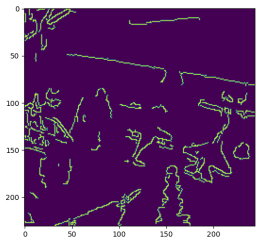

图 10: 场景边界提取结果

然后通过计算滑窗和模版的边缘坐标之间的 Hausdorff 距离，图 11 和图 12 分别是模版 1 和模版 2 在场景里的距离分布图 (分布图是逆时针旋转了 90°)。

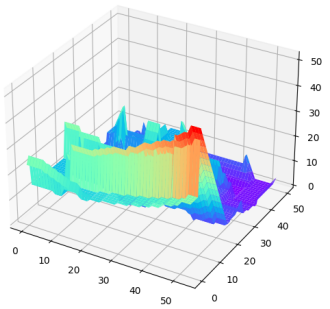

图 11: 模版 1 在场景中的 Hausdorff 距离分布

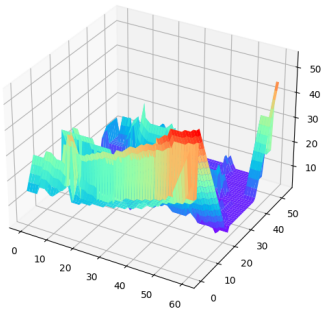

图 12: 模版 2 在场景中的 Hausdorff 距离分布

选取出其中的最小值，得到对应的坐标:

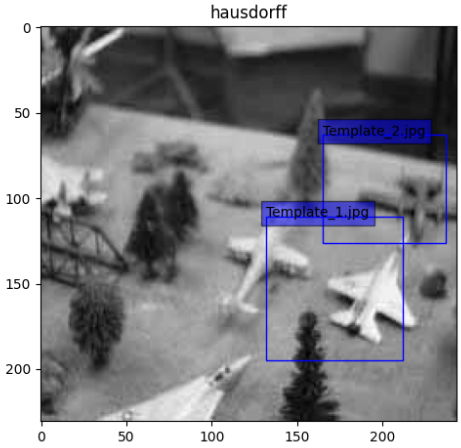

图 13: Hausdorff 距离匹配方法结果

```
左下角为(0,0)情况下,模版1和2目标框的左上角坐标分别为:
 [[120, 132], [168, 165]]
hausdorff耗时: 74.62949228286743
```

图 14: 左上角坐标和运行耗时

从结果可以看到模版 1 的结果变好了，但是模版 2 的结果变差了。分析原因是因为在边缘提取时设置的阈值过高，导致边缘较少。观察场景边缘图可以看到模版 2 的边缘主要集中在左下角，而右上角较空，分析其匹配结果在场景边缘图中也可以看出，匹配到的位置主要边界分布都在下侧，所以匹配错误。但是对于模版 1 提取到的边界较多的情况来讲，效果明显比相关性匹配更好。整体来看运行时间耗时 74.6 秒，远远高于相关性匹配的耗时 (并且此处用的是 3 步长的滑窗，相关性滑窗步长为 1)。分析原因是因为 Hausdorff 距离在计算时涉及到一个多层循环，导致速度大幅下降。

2.4 基于距离转换的距离匹配方法

首先是对图像进行 Canny 算子边界提取，然后对场景图像进行距离转换，结果示意图如下图所示：

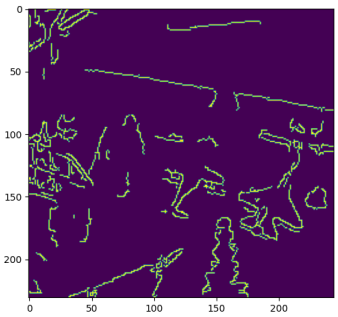

图 15: 距离转换场景的边界图

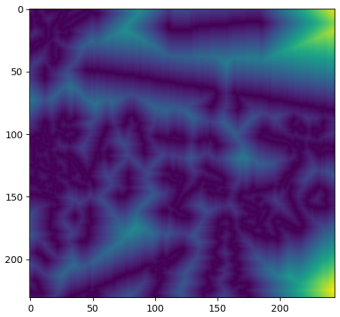

图 16: 场景距离转换结果

然后利用模版的边界图与场景的转换图进行匹配，下图分别是模版 1 和模版 2 的距离分布示意图:

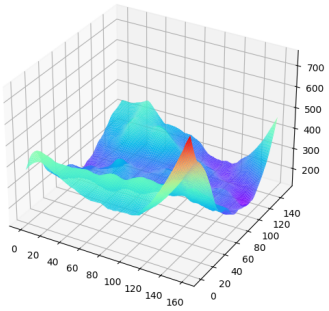

图 17: 模版 1 距离分布

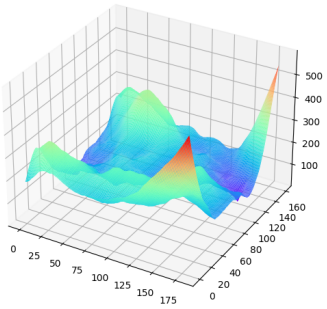

图 18: 模版 2 距离分布

选择出最小距离对应的坐标作为结果：

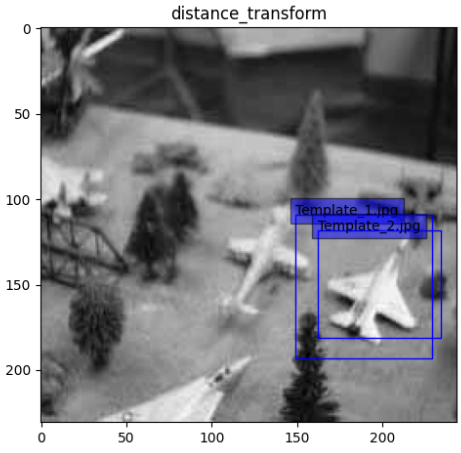

图 19: 距离转换结果

```
左下角为(0,0)情况下,模版1和2目标框的左上角坐标分别为:
 [[122, 149], [113, 162]]
distance_transform+hausdorff耗时: 9.351962089538574
```

图 20: 左上角坐标和运行耗时

从结果可以看到基于距离转换后的效果比单纯基于 Hausdorff 距离计算的效果好得多，两个模版都

较好地匹配上了，而且运行时间也远远缩短了。分析原因是因为距离转换后让边界加粗了，就有了一定的轻微抗旋转能力，所以效果上去了。

3 总结

经过这次实验作业，我更加深刻地理解了图像模版匹配的相关知识。运用了三种不同的方法层层递进进行实验。了解到各个方法他们的优点和缺点。也从实验知道了距离转换的巧妙性和鲁棒性。在做任务时需要同时考虑模型的能力和运行速度，才能做到真正的实用。