计算机视觉作业 -5：背景建模

王天锐　20120318

2021 年 1 月 5 日

摘要

本次实验作业主要是根据老师上课所讲知识，利用 Scene_Data 文件夹中的视频帧序列实现基于 GMM（高斯混合模型）的背景建模。

1　方法原理及实现

1.1　混合高斯模型的背景建模

图像中每个像素点的值（或特征）短时间内都是围绕与某一中心值一定距离内分布，通常，中心值可以用均值来代替，距离呢可以用方差来代替。这种分布呢是有规律的，根据统计定律，如果数据点足够多的话，是可以说这些点呈正态分布，也称为高斯分布（取名高斯，大概是因为很多地方都用这个名字吧）。根据这个特点，如果像素点的值偏离中心值较远，那么，这个像素值属于前景，如果像素点的值偏离中心值很近（在一定方差范围内），那么可以说这个点属于背景。理论上，如果不存在任何干扰的话，是可以准确区分前景和背景的。但是，现实往往不尽如人意，如果画面中光线变化的话，这个高斯分布的中心位置是会改变的。

混合高斯模型是单高斯模型的推广，单高斯模型只能描述背景的单一模式，当背景表现为树叶晃动等多模态形式时就容易出错。以树叶晃动的背景为例，树叶晃出某个位置时，该位置的像素信息用一个高斯模型表示，树叶晃到该位置时，用另一个高斯模型表示该位置的像素信息，这样新的图片中的像素不论与哪个高斯模型匹配都将视为背景，这样就防止模型将树叶晃动也视为运动目标。

1.1.1　像素模型的定义

每个像素由多个单模型描述:$P(p) = \{w_i(x,y,t), u_i(x,y,t), \theta_i(x,y,t)^2\}, i = 1,2,\ldots,K$。K 的值一般在 3 到 5 之间，表示混合高斯模型中包含单模型的个数，$w_i(x,y,t)$ 表示每个模型的权重，满足:

$$\sum_{i=1}^{K} w_i(x,y,t) = 1$$

三个参数 (权重、均值、方差) 确定一个单高斯模型。

1.1.2　更新参数并进行前景检测

(1) 第一步**:**如果新读入的图像序列中的图片在 (x,y) 处的像素值对于 $i = 1,2,\ldots,K$ 满足 $|I(x,y,t) - u_i(x,y,t)| \leq \lambda\theta_i(x,y,t)$, 则新像素点与该单模型匹配。如果存在与新像素点匹配的单模型，则判断为背

景，并进入第二步；如果不存在则判断为前景，并进入第三步。

(2) 第二步: 修正与新像素匹配的单模型的权值，权值增量为 $dw = \alpha(1 - w_i(x, y, t-1))$，新的权值表示如下:

$$w_i(x, y, t) = w_i(x, y, t-1) + dw = w_i(x, y, t-1) + \alpha(1 - w_i(x, y, t-1))$$

修正与新像素匹配的单模型的均值和方差，同单高斯模型。完成第二步后就进入第四步。

(3) 如果新像素不与任何一个单模型匹配，则:1) 如果当前单模型的数目已经达到允许的最大数目，则去除当前多模型集合中重要性最小的单模型，重要性的计算见第三小节。2) 增加一个新的单模型，新的单模型的权重为一个较小的值，均值为新像素的值，方差为给定的较大的值。

权重归一化:

$$w_i(x, y, t) = \frac{w_i(x, y, t)}{\sum_{j=1}^{K} w_i(x, y, t)}, (i = 1, 2, \ldots, K)$$

1.1.3 多个单高斯模型的排序及删减

混合高斯模型的基本思想是用多个高斯模型作为一个像素位置的模型，使得模型在多模态背景中具有鲁棒性。我们假设背景模型具有以下特点：(1) 权重大: 背景出现的频率高；(2) 方差小: 像素值变化不大。据此，我们以

$$sort\_key = \frac{w_i(x, y, t)}{\delta_i(x, y, t)}$$

作为重要性排序的依据。排序及删减过程如下: (1) 计算每个单模型的重要性值 sort_key。(2) 对于各个单模型按照重要性的大小进行排序，重要性大的排在前面。(3) 若前 N 个单模型的权重满足 $\sum_{i=1}^{N} w_i(x, y, t) > T$，则仅用这个单模型作为背景模型，删除其他的模型，一般阈值 T=0.7。

1.2 代码实现

整体归纳则是首先初始化预先定义的几个高斯模型，对高斯模型中的参数进行初始化，并求出之后将要用到的参数。其次，对于每一帧中的每一个像素进行处理，看其是否匹配某个模型，若匹配，则将其归入该模型中，并根据新的像素值对该模型参数进行更新，若不匹配，则以该像素建立一个高斯模型，初始化参数。最后选择前面几个最有可能的模型作为背景模型，为背景目标提取做铺垫。如下为整体代码实现:

```
num_gauss = 3  # 高斯的数量
D = 2.5  # 阈值系数
sd_init = 6  # 标准差初始化
num_pic = 201  # 200张图片
lr = 0.01  # 学习率
threshold = 0.90
first = load_img(os.path.join(home, '0000.jpg'))
n_row, n_col = first.shape  # H,W
weights = np.zeros((n_row, n_col, num_gauss))  # 用来存混合高斯的参数
weights[:, :, 0] = 1
mean = np.zeros((n_row, n_col, num_gauss))  # 混合高斯的均值
mean[:, :, 0] += first[:, :]  # 将第一个初始化为第一帧的像素值
```

```
sd = np.ones_like(mean) * sd_init
mask = np.ones((num_pic, n_row, n_col))
n = 0
# 为了代码整洁，去掉了对文件名的循环
frame = load_img(os.path.join(home, name))
for row in  range(n_row):
        for col in  range(n_col):
                match = False
                # 更新参数
                for k in  range(num_gauss):
                        diff =  abs(frame[row, col] - mean[row, col, k])
                        if diff <= D * sd[row, col, k] and not match:
                                match = True
                                weights[row, col, k] = (1 - lr) * weights[row, col, k] + lr
                                p = lr / weights[row, col, k]
                                mean[row, col, k] = (1 - p) * mean[row, col, k] + p * frame[row, col
                                    ]
                                sd[row, col, k] = np.sqrt((1 - p) * (sd[row, col, k] ** 2) + p * (
                                    diff ** 2))
                        else:
                                weights[row, col, k] = (1 - lr) * weights[row, col, k]
                        # 如果没有匹配上，则用新模型
                        if not match:
                                min_arg = np.argmin(weights[row, col, :])
                                mean[row, col, min_arg] = frame[row, col]
                                sd[row, col, min_arg] = sd_init
                                # 对于各个高斯进行weight/std排序。找出前k个，作为背景模型，其余为前
                                    景
                                w_sum = np. sum(weights[row, col, :])
                                weights[row, col, :] = weights[row, col, :] / w_sum  # 归一化
                        for p in  range(num_gauss):
                                if np. sum(weights[row, col, :(p + 1)]) >= threshold:
                                        if  abs(frame[row, col] - mean[row, col, p]) <= D * sd[row,
                                             col, p]:
                                                mask[n, row, col] = 255
                                                break
                                        else:
                                                mask[n, row, col] = 1
                                                break
        save_result(mask[n], name)
```

2 实验结果及分析

2.1 序列帧展示

下列是部分时序帧的展示。

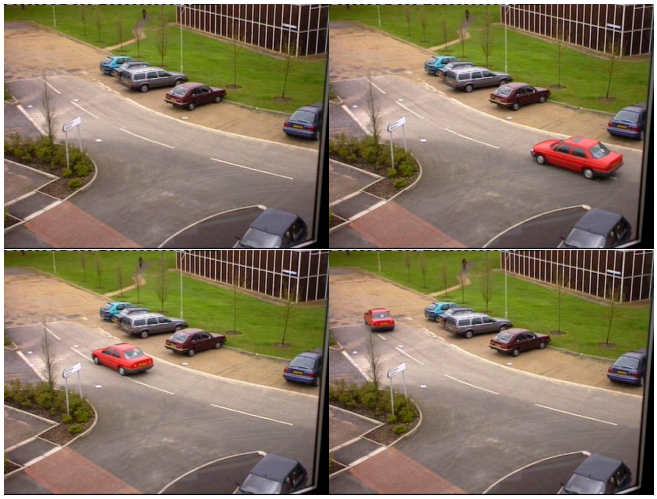

图 1: 视频序列展示

可以看到整个过程中，人眼能看到的“动态”是那个红色的车子。

2.2 GMM 背景建模结果

测试了不同阈值情况下的处理结果。分别阈值为 0.7、0.9 和 0.95。如下所示

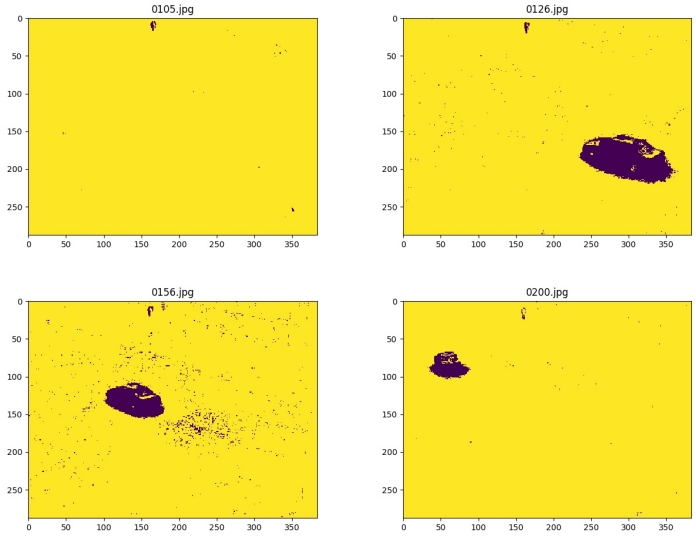

图 2: 阈值 0.7 的处理结果

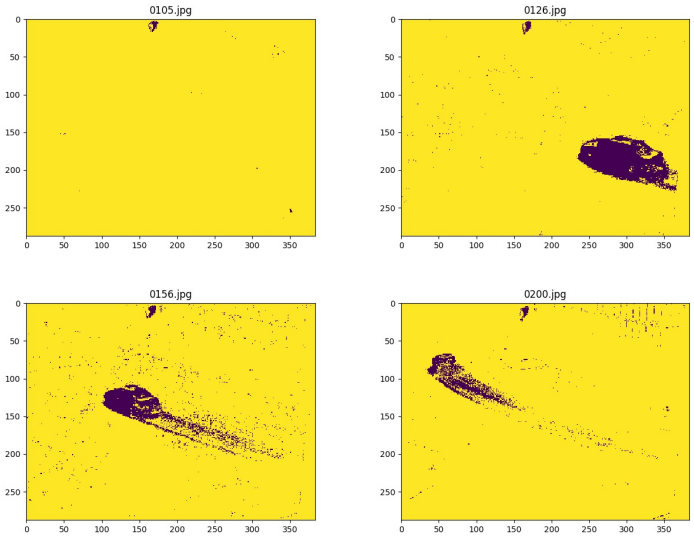

图 3: 阈值 0.9 的处理结果

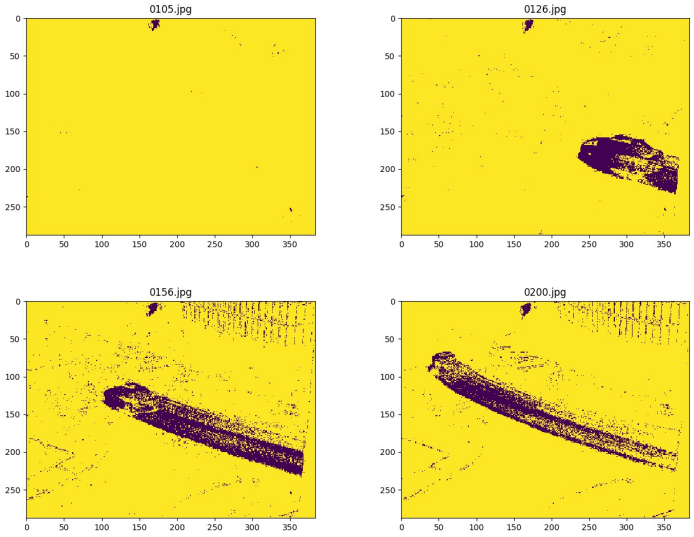

图 4: 阈值 0.95 的处理结果

从结果可以看到阈值越高出现的噪点越多，这是因为我们对于后景的判断是大于阈值的判定为背景图。阈值越大背景选择得越少，所以前景增多导致噪声增多。

2.3 实时性分析

根据实验发现，本算法实时性较差。如下是不同图片大小下的耗时情况。

图 5: 原图大小耗时情况

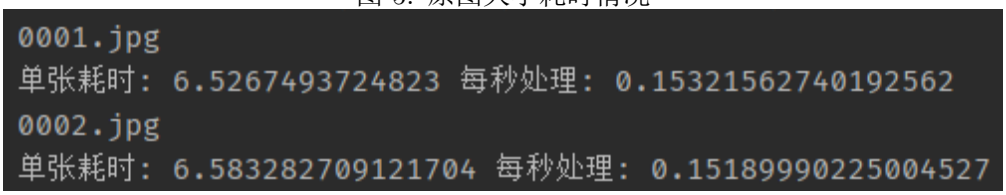

图 6: 原图 1/2 大小耗时情况

图 7: 原图 1/5 大小耗时情况

可以看到在商务笔记本算力情况下，该算法的处理速度是非常慢的。分析原因也可能是因为 python 语言本身就比 C 慢，再加上算法本身的嵌套循环非常多。所以导致最终速度很慢。

3 总结

经过这次实验作业，我更加深刻地理解了背景建模的相关知识。对 GMM 背景建模算法进行了复现和实验。通过实验将上课所学的知识应用到了实践中去。通过实验加深了一些课堂上理论的理解。还让我发现了 GMM 背景建模的阈值设置的一些问题所在。也开发了我的自主探索的意识。非常感谢朱老师给我这次实验的机会。