计算机视觉作业-1：基于直方图的自适应阈值分割

王天锐

20120318 信号 1 班

摘　要：本次实验主要是根据老师上课所讲知识，利用基于直方图的自适应阈值分割方法来对图像进行分割。在本报告中，先是设定三个不同阈值，直接观察分割结果。然后利用统计直方图，得到一个自适应的阈值，再观测实验结果。所用语言为 python 语言，利用 numpy 矩阵运算包实现

1　方法原理及实现

1.1　阈值分割法

阈值分割可以看做一种函数操作

$$g(x,y)=\begin{cases}1, f(x,y)>T\\0, f(x,y)\le T\end{cases} \tag{1}$$

其中 T 是阈值，x 与 y 为图像横纵坐标，$f(x,y)$ 对应相应的原始图像像素值，$g(x,y)$ 为处理后的对应的像素值。对应代码实现如下

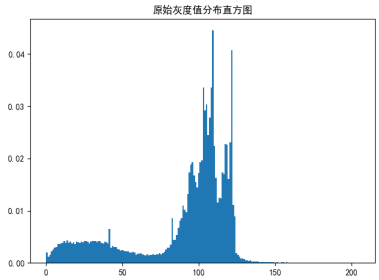
```
def process_img(threshold, img):
    image = img.copy()
    image[img>threshold] = 1
    image[img<=threshold] = 0
    return image
```

1.2　基于直方图的自适应阈值分割法

由上述阈值分割法的原理可知，阈值的设置尤其重要。但是由于在实际任务中往往不能够每张图去手动设置一个阈值，所以需要一个较为合理的阈值求取方法。在这里讲述基于直方图的自适应阈值分割法。

对于每一张灰度图，我们可以将其所有灰度值进行一个概率分布统计，可以绘制出一张灰度值分布直方图。如下所示：

(a) 原始图像



(b) 灰度值分布

图 a 是原始图像，图 b 是其对应的灰度值分布直方图。得到直方图后，我们利用混合高斯分布来对其进行拟合。为更加方便拟合，在拟合之前需先进行一个平滑操作。本实验采用均值平滑方法，代码实现如下:

```
def smooth(x, win_len):
    pad_len = (win_len - 1) // 2
    padded = np.pad(x, (pad_len, pad_len), "constant")
    n_frame = len(padded) - (win_len - 1)
    strides = padded.itemsize * np.array([1, 1])
    win_wrapped = np.lib.stride_tricks.as_strided(padded, shape=(
                                    n_frame, win_len), strides=strides)
    return np.mean(win_wrapped, axis=1)
```

平滑后，需要将分布拟合为两个高斯分布的组合分布。所以需要用两个高斯分布 (分割物和背景) 来进行分布的模拟:

$$f_o(g) = \frac{1}{\sqrt{2\pi}\sigma_o}\exp(-\frac{(g-\mu_o)^2}{\sigma_o^2}) \quad f_b(g) = \frac{1}{\sqrt{2\pi}\sigma_b}\exp(-\frac{(g-\mu_b)^2}{\sigma_b^2})$$

高斯分布对应实现代码为:

```
def gaussian(mu, theta, x):
    theta = theta + np.array(1e-12)
    return (1/((2*np.pi)**0.5*theta))*np.exp(-(x-mu)**2/(2*theta**2))
```

f_o 指的是分割物 (Object) 的分布函数，f_b 为背景 (background) 的分布函数，二者分别分布在阈值的两侧，其均值和方差均可根据阈值两侧的数据进行计算得到，计算公式如下:

$$\mu_o = \sum_{t}^{g=0} gf(g) \quad \mu_b = \sum_{max}^{g=t+1} gf(g)$$

$$\sigma_o = \sum_{t}^{g=0} (g-\mu_o)f(g) \quad \sigma_b = \sum_{max}^{g=t+1} (g-\mu_b)f(g)$$

代码实现为:

```
threshold_left_bins = bin_center[bin_center <= threshold]
left_p = p_of_bin[bin_center <= threshold] / np.sum(p_of_bin[
                                    bin_center <= threshold])
left_gaussian_mu = np.sum(threshold_left_bins * left_p)
left_gaussian_theta = np.mean((threshold_left_bins - left_gaussian_mu
                                    ) ** 2 * left_p)
threshold_right_bins = bin_center[bin_center > threshold]
right_p = p_of_bin[bin_center > threshold] / np.sum(p_of_bin[
                                    bin_center > threshold])
right_gaussian_mu = np.sum(threshold_right_bins * right_p)
right_gaussian_theta = np.mean((threshold_right_bins -
                                    right_gaussian_mu) ** 2 * right_p)
```

然后可以根据先验概率计算得到后验概率：

$$p_o(t) = \sum_{t}^{g=0} f(g) \quad p_b(t) = 1 - p_o(t)$$

$$P_t(g) = p_o f_o(g) + p_b f_b(g)$$

代码实现为:

```
p_l = np.sum(p_of_bin[bin_center <= threshold])
p_r = 1 - p_l
P = p_l * gaussian(left_gaussian_mu, left_gaussian_theta, bin_value)
                                    + p_r * gaussian(right_gaussian_mu,
                                    right_gaussian_theta, bin_value)
```

最后得到整体后验概率后经过同样的平滑操，再根据散度损失函数来选择最佳阈值即可:

$$K(t) = \sum_{g=0}^{max} f(g) \log[\frac{f(g)}{P_t(g)}]$$

散度损失函数代码实现为:

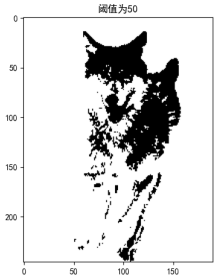
```
def divergence(f, p):
    eps = np.array(1e-9)
    p = p + eps
    return np.mean(f * np.log(f / p + eps))
```

2 实验分析

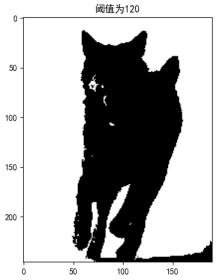
2.1 三种阈值观察结果

本小节实验分别设置了 50、150 以及 200 为阈值进行实验观察，结果如下:

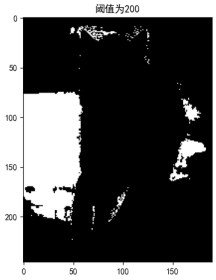
(c) 原始图像



(d) 阈值 50 结果



(e) 阈值 150 结果

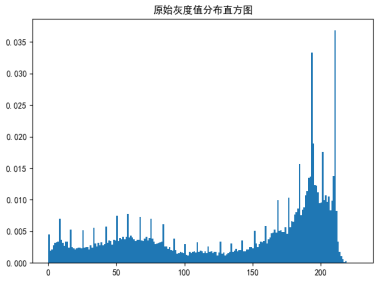


(f) 阈值 200 结果

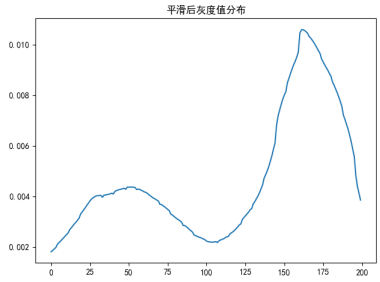
从结果来看，50 阈值过小，将狼身体上部分偏亮的部分给抹去了；150 比较合理；200 过大，将部分背景保留了下来。

2.2 直方图自适应阈值分割

本小节将利用直方图自适应阈值分割来处理 2.1 节的样本。首先是统计原始图片的灰度值分布情况, 并利用平滑函数将其平滑:

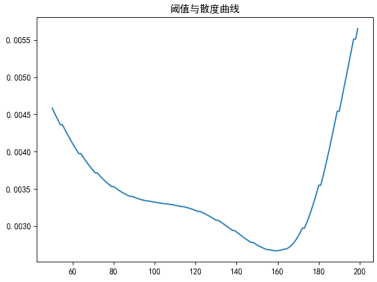


(g) 直方图分布

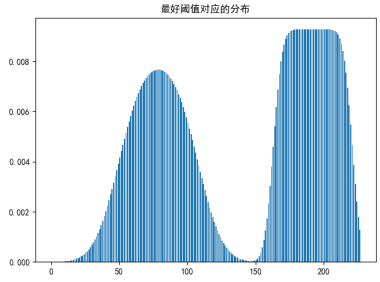


(h) 平滑后的直方图分布

图 g 为图 c 的灰度值分布直方图，图 h 为对其进行窗长为 51 的平滑后的结果。利用混合高斯分布对平滑后的分布进行拟合，并结合损失函数对阈值进行选择:

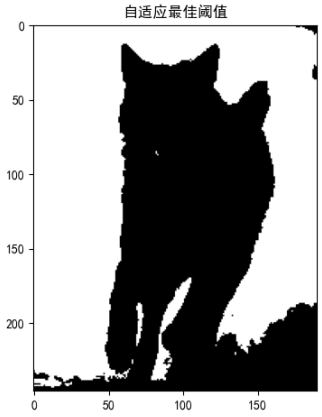


(i) 阈值 Loss 曲线



(j) 最佳阈值对应曲线

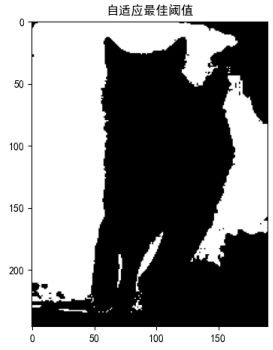
可以看到，根据 Loss 函数计算得最佳阈值为 159。所以选取 159 作为阈值进行处理，得到最终结果:



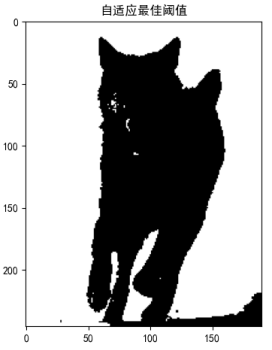
(k) 自适应得到的阈值所对应的处理结果

可以看到通过该算法，只需放入图片即可自动地计算出较为合适的阈值进行处理。效果虽然没有 150(前面测试的) 好，但是整体来讲还算能够接受。此处分析可能是由于该实验所用的损失函数是散度函数，更加关注的是曲线的分布情况。而且该图中狼腿的颜色和狼鼻子的颜色比较“浅”和背景颜色较为接近导致自适应拟合不够完美。

经过实验发现，平滑窗的长度对结果会有比较大的影响。可见下图：



(l) 窗长 31 平滑的最终结果 (阈值为 179)



(m) 窗长 71 平滑的最终结果 (阈值为 126)

可以看到，平滑窗长越大，效果越好，二者分别计算得到的最佳阈值为 179 和 126，可见差距非常之大。分析原因是平滑窗越长会导致分布更加平滑与均匀。使得混合高斯拟合得有效 (均值找得更准)。也从另一个角度说明直方图自适应算法比较受噪声影响。平滑相当于是一个低通滤波可以一定程度上滤到一部分噪声，窗越长滤除得越多，所以效果更好。

3 总结

经过此次实验，让我更加深入了解了传统的基于像素的图像分割算法。将老师上课讲的用代码自己复现一遍对我的数学理解能力和编程代码能力都有所提升。通过实验我也更加理解直方图自适应阈值计算的思路和其中需要注意的技术点，比如：平滑窗长设置的重要性 (降噪) 等。除开编码这次实验也锻炼了我 latex 文档编辑能力，非常感谢朱老师给我这次实验的机会，让我受益匪浅。