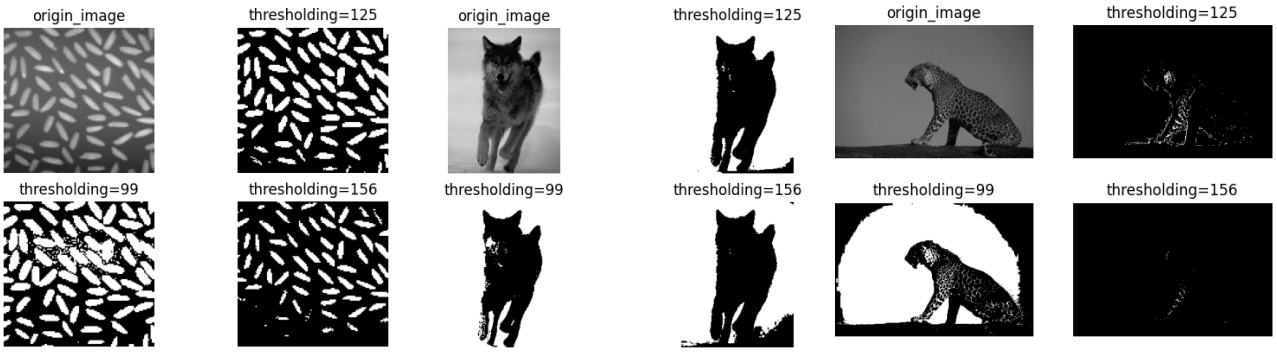
计算机视觉作业1: 基于直方图的自适应阈值分割

唐麒 21120299
qitang@bjtu.edu.cn

摘要—图像分割是是图像理解的重要组成部分，同时也是计算机视觉研究中的一个重要领域。传统的图像分割方法根据灰度、颜色、纹理、和形状等特征将图像进行划分区域，让区域间显差异性，区域内呈相似性。本文主要围绕基于阈值的分割方法进行实验，在随机设定三个不同阈值进行观察的基础上，进一步利用基于直方图的自适应阈值分割方法来对灰度图像进行分割。图像根据自适应阈值的分割结果将在本文的实验部分给出，并对部分参数设置对实验结果的影响进行简单探讨。

关键词—计算机视觉、图像分割、灰度直方图、自适应阈值、混合高斯分布



(a) (b) (c)

图 1: 原始图像及三个不同阈值下的分割结果: (a) Test_Img_1, (b) Test_Img_2, and (c) Test_Img_3. 三个阈值分别为 125（右上）、99（左下）和 156（右下）

1 简介

图像分割是指根据灰度、彩色、空间纹理、几何形状等特征把图像划分成若干个互不相交的区域，使得这些特征在同一区域内表现出一致性或相似性，而在不同区域间表现出明显的不同。简而言之就是在一副图像中，把目标从背景中分离出来。对于灰度图像来说，区域内部的像素一般具有灰度相似性，而在区域的边界上一般具有灰度不连续性。

近年来，随着硬件设备的不断提升和数据量的大幅度增长，深度学习技术在相当多的计算机视觉任务上都取得了令人瞩目的成绩，包括图像分割任务。而在此之前，研究者利用数字图像处理、拓扑学、数学等方面的知识来进行图像分割的方法，主要包括: 基于阈值的分割方法、基于聚类的分割方法、基于区域的分割方法、基于图论的分割方法和基于边缘的分割方法等。

本文所采用的实验方法为基于阈值的分割方法。这类方法的核心思想是基于图像的灰度特征来计算一个或多个灰度阈值，并将图像中每个像素的灰度值与阈值相比较，最后将像素根据比较结果分到合适的类别中。因此，该类方法最为关键的一步就是按照某个准则函数来求解最佳的灰度阈值。

基于阈值的分割方法适用于目标和背景占据不同灰度范围的图像。理想情况下，目标和背景在其所属区域内具有恒定的不同灰度，则只需选取一个阈值便可对图像进行分割。但实际上，由于拍摄时的噪声、非均匀照明和目标表面对光照的非均匀反射等因素的干扰，其灰度值并不是恒定的，因此固定阈值分割难以有出色表现。这种情况下，我们可以根据不同图像的特性分别采用不同的阈值进行分割，即自适应阈值分割。

基于阈值的分割方法大多利用了图像的灰度直方图，它很好地反映了一幅图像中的灰度分布信息，是阈值选取的重要参考依据。在对直方图区域进行划分后，可以得到相应的直方图统计信息（目标／背景的先验概率、均值等），由此可以进一步求解不同的阈值选取准则函数。

本文剩余部分将主要对基于直方图的自适应阈值分割方法的原理、实现及实验进行介绍，所有实验涉及的方法、函数实现均基于 Python 语言。

2 方法

这一章节将对基于阈值的图像分割方法和本文所进行实验的基于直方图的自适应阈值分割进行介绍。

2.1 基于阈值的图像分割方法

图像分割旨在根据一定的准则将图像 I 划分为不同区域的集合 S，集合 S 满足下列条件：

1) $\cup S_i = S$
2) $S_i \cap S_j = \emptyset,\ i \neq j$
3) $\forall S_i,\ P(S_i) = true$
4) $P(S_i \cup S_j) = false,\ i \neq j,\ S_i\ adjacent\ S_j$

对于灰度图像而言，可以将其看作由目标（前景）和背景所构成，基于阈值的分割方法就是通过一个固定阈值将图像中的目标像素与背景像素进行分割，从而实现从图像中提取目标的目的。上述过程可以用下面的公式进行描述：

$$I'(x,y) = \begin{cases} 255, & I(x,y) > \text{thresholding} \\ 0, & I(x,y) \leq \text{thresholding} \end{cases} \tag{1}$$

其中，thresholding 是设定的阈值，x 与 y 表示像素在图像中的坐标，$I(x,y)$ 为像素 (x,y) 在原始图像中的灰度值，$I'(x,y)$ 为像素 (x,y) 根据阈值分割后的二值图像对应的像素值。代码实现如下：

```python
# 对图像进行二值化处理
def segmentation_by_thresholding(image, thresholding):
    thresholding_image = image.copy()
    thresholding_image[image > thresholding] = 255
    thresholding_image[image <= thresholding] = 0
    return thresholding_image
```

main.py

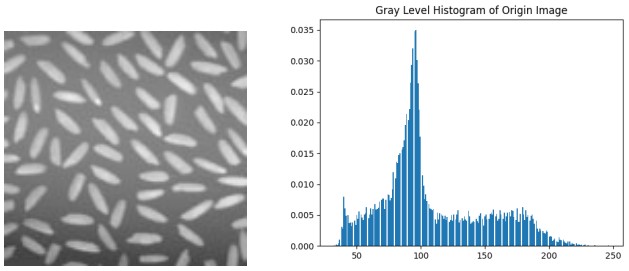
阈值的选择对于基于阈值的分割方法而言至关重要。在理想条件下，目标和背景的灰度值具有明显的划分，可以将直方图的峰值之间的数值设置为阈值。但在实际应用中，由于各种环境因素的影响，目标和背景的灰度值界限并不易于区分，如图 1 所示，阈值设置过低时，背景中的噪声也会被划分到目标类别中，而当阈值设置过高时，目标中灰度值较低的部分被划入了背景部分。此外，对于不同的图像而言，相同的阈值也会取得不同的分割效果。因此，阈值的设置是基于阈值的分割方法的一大难题。图像分割阈值的设置常用方法包括：

- 基于直方图形状：对去噪后的直方图的峰值、谷值和曲率等进行分析从而确定阈值
- 基于聚类：迭代找到阈值，并基于该阈值进行聚类
- 基于熵：选择一个使直方图信息量最大化的阈值
- 空间阈值 (高阶统计)：阈值的选择是基于空间邻域的高阶统计量
- 局部阈值：在每个邻域内利用局部统计信息寻找阈值

本文实验的方法是基于直方图的自适应阈值分割，即将图像的灰度直方图中的目标分量和背景分量分别参数化为模型（如高斯分布），然后在图像的灰度范围内迭代地设置不同的阈值，在当前阈值下，可以利用直方图的信息对模型中的未知参数进行估计（如高斯分布的均值、方差等），进而利用估计的参数对图像的整体灰度直方图进行“预测”。在同时具有“预测”分布和真实分布的情况下，就可以对不同阈值下“预测”的模型或分布进行评价，从而获得使拟合效果最好的阈值。

2.2 基于直方图的自适应阈值分割

图像的灰度直方图可以反映一幅图像中的灰度分布信息，能够为阈值的选取提供重要参考依据。基于直方图的自适应阈值分割方法，首先利用灰度直方图对图像的灰度分布进行统计，其实现利用了 matplotlib 包中的 pyplot.hist 接口。如图 2 所示：



(a) (b)

图 2: Test_Img_1 及其灰度直方图：(a) 原始图像, (b) 灰度直方图.

为了去除噪声的影响以及便于后续分布的拟合，要对直方图进行预处理，可以采用均值滤波或一维高斯滤波。本文采用了一维高斯滤波，结果如图 3 所示：

图 3: Test_Img_1 经过一维高斯滤波后的灰度直方图曲线

一维高斯滤波的实现是通过生成一个滑动窗口大小的离散序列，该序列服从 0 均值，指定方差的高斯分布（不同的方差滤波的结果不同，方差越大，滤波的结果也越平滑）。然后利用该序列对原始灰度曲线做卷积，其代码实现如下：

```python
# 一维高斯滤波
def oned_gaussian_filter(sequence, win_size):
    sigma = 128
    r = range(-int(win_size / 2), int(win_size / 2) + 1)
    gauss = [1 / (sigma * math.sqrt(2 * math.pi)) * math.exp(-float(x) ** 2 / (2 * sigma ** 2)) for x in r]
    return np.convolve(sequence, gauss, 'same')
```

main.py

为了对滤波处理后的灰度分布曲线进行拟合，同时将图像中的目标灰度值与背景灰度值进行区分，就需要分别建立符合目标和背景分量的直方图形状的参数模型，从图 3 可以看出，目标分量和背景分量的直方图形状近似于高斯分布，而图像整体的灰度直方图形状即为由两个高斯分布组成的混合高斯分布。因此，在基于直方图的自适应阈值分割方法中，用下面两个高斯分布来分别代表目标和背景的分布：

$$f_o(g) = \frac{1}{\sqrt{2\pi}\sigma_o} e^{-1/2\left(\frac{g-\mu_o}{\sigma_o}\right)^2} \tag{2}$$

$$f_b(g) = \frac{1}{\sqrt{2\pi}\sigma_b} e^{-1/2\left(\frac{g-\mu_b}{\sigma_b}\right)^2} \tag{3}$$

其中，$f_o(g)$ 和 $f_b(g)$ 分别为目标和背景的分布函数，同时也表示灰度值为 g 的像素占图像总体像素的百分比，μ_o 和 μ_b 分别为目标和背景的灰度均值，$sigma_o$ 和 $sigma_b$ 分别为目标和背景的标准差。

高斯分布用 Python 实现的代码如下：

```python
# 定义高斯模型
def gaussian(x, mu, theta):
    theta = theta + np.array(1e-10)  # 避免 0 做除数
    return (1 / ((2 * np.pi) ** 0.5 * theta)) * np.exp(-(x - mu) ** 2 / (2 * theta ** 2))
```

main.py

我们希望阈值可以将目标和背景的灰度分布进行分割，换句话来说，就是目标和背景的灰度可以分别分布在阈值的左右两侧。因此，我们可以假定一个阈值 t，然后对灰度图像直方图中阈值 t 左右两侧的灰度信息进行统计计算，从而可以利用统计数据对 $f_o(g)$ 和 $f_b(g)$ 的均值及标准差进行估计，其估计方法如下所示：

$$\mu_o(t) = \sum_{g=0}^{t} f(g)g \quad \mu_b(t) = \sum_{g=t+1}^{\max} f(g)g \tag{4}$$

$$\sigma_o = \sum_{g=0}^{t} f(g)\,(g-\mu_o) \quad \sigma_b = \sum_{g=t+1}^{\max} f(g)\,(g-\mu_b) \tag{5}$$

根据先验，我们可以得到一个像素落在目标或落在背景的概率分别为：

$$p_o(t) = \sum_{t}^{g=0} f(g) \tag{6}$$

$$p_b(t) = 1 - p_o(t) \tag{7}$$

对于假定的阈值 t，可以通过下面的公式对图像整体的灰度分布进行“预测”，并同样利用一维高斯滤波对估计的分布进行滤波处理：

$$P_t(g) = p_o f_0(g) + p_b f_b(g) \tag{8}$$

上述过程的代码实现如下：

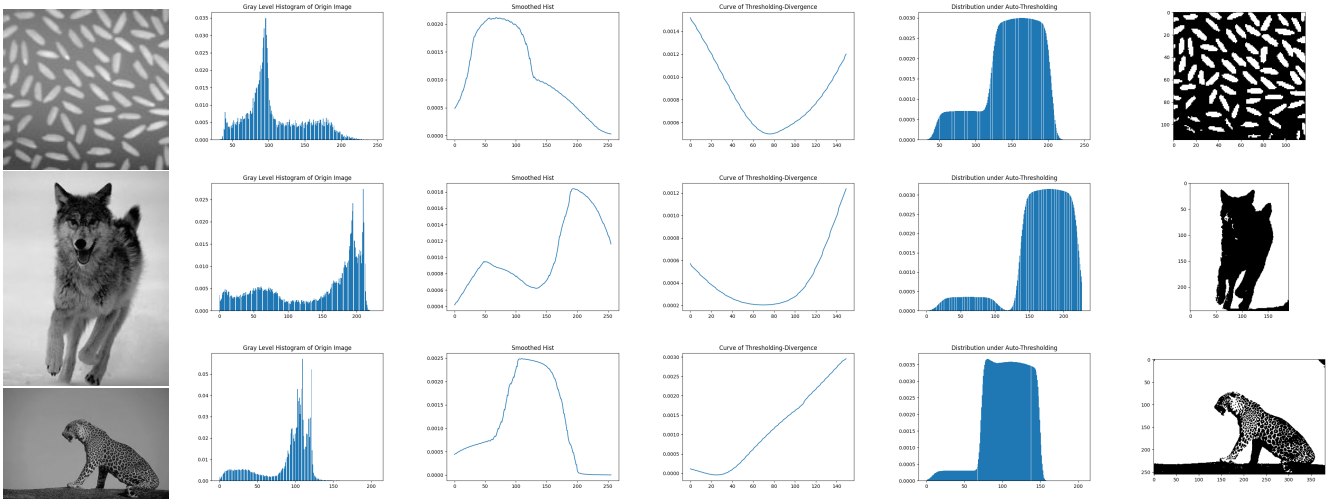
```python
# 根据阈值预测混合高斯分布
def mixture_gaussian(thresholding, n, bins_center, smooth_win):
    if thresholding > np.max(bins_center) or thresholding < 0:
        print("Invalid Value of Thresholding:", thresholding)
        return
    bins_of_fo = bins_center[bins_center <= thresholding]
    po = n[bins_center <= thresholding] / np.sum(n[bins_center <= thresholding])
    mu_of_fo = np.sum(bins_of_fo * po)
    theta_of_fo = np.mean((bins_of_fo - mu_of_fo) ** 2 * po)

    bins_of_fg = bins_center[bins_center > thresholding]
    pg = n[bins_center > thresholding] / np.sum(n[bins_center > thresholding])
    mu_of_fg = np.sum(bins_of_fg * pg)
    theta_of_fg = np.mean((bins_of_fg - mu_of_fg) ** 2 * pg)

    po_t = np.sum(n[bins_center <= thresholding])
    pb_t = 1 - po_t

    predicted_model = []
    for bin_center in bins_center:
        predicted_model.append(po_t * gaussian(bin_center, mu_of_fo, theta_of_fo)
                               + pb_t * gaussian(bin_center, mu_of_fg, theta_of_fg))
    return oned_gaussian_filter(predicted_model, smooth_win)
```

main.py



(a) (b) (c) (d) (e) (f)

图 4: 基于直方图的自适应阈值分割结果：(a) 原始图像, (b) 灰度直方图, (c) 一维高斯滤波后的灰度信息图, (d) 阈值-散度曲线图, (e) 散度最小时拟合的灰度直方图, (f) 自适应阈值的分割结果.

最后，只需要在图像的灰度值变化范围内对潜在的阈值进行迭代，并通过计算 KL 散度来度量“预测”分布与真实分布之间的相似性，从而搜索到使得二者最相似的阈值即可，KL 散度的计算公式为：

$$K(t) = \sum_{g=0}^{\max} f(g) \log \left[\frac{f(g)}{P_t(g)} \right] \tag{9}$$

其 Python 代码为：

```python
# 计算散度
def calculate_divergence(true_distribution, fit_distribution):
    eps = np.array(1e-10)  # 避免 0 做除数
    fit_distribution = fit_distribution + eps
    return np.mean(true_distribution * np.log(true_distribution / fit_distribution + eps))  # 避免 log(0)
```

main.py

3 实验

3.1 基于人工设置阈值的分割结果

设定三个不同阈值，分别为 99、125 和 156，分割结果如图 1 所示。可以看到对于同一幅图像来说，阈值设置过低时，背景中的噪声也会被划分到目标类别中，而当阈值设置过高时，目标中灰度值较低的部分被划入了背景部分。以 Test_Img_1 为例，在阈值为 125 时，分割效果较好，而在设置阈值为 99 时，分割结果中有部分背景噪声被分割到了目标中，而当阈值设置为 156 时，图像中的目标明显分割不全。此外，对于不同的图像而言，相同的阈值也会取得不同的分割效果。例如在阈值设置为 125 时，Test_Img_1 可以去的较好的分割效果，但 Test_Img_3 的分割结果质量很差。

3.2 基于直方图的自适应阈值分割结果

在一维高斯滤波滑动窗口大小为 96，$\mu = 0, \sigma = 128$ 的自适应阈值分割结果如图 4 所示，得到的阈值如表 1 所示：

表 1: 基于直方图的自适应阈值分割方法阈值表

	Test_Img_1	Test_Img_2	Test_Img_3
阈值	126	120	75

从图 4 可以看出，基于直方图的自适应阈值分割方法在三幅图像中均取得了较好的分割效果，尤其是对于 Test_Img_3 来说，该方法选取的阈值的分割结果远远好于人工设定的三个阈值的分割结果。而表 1 则反映了由于不同图像的内容不同，其灰度值的分布也不相同，导致其阈值的选取变化也较大，基于直方图的自适应阈值分割方法可以根据不同图像的灰度值分布选取合适的分割阈值，从而可以大大减少人力工作。

但基于直方图的自适应阈值分割方法的性能还会受到一些参数选择的影响，例如滤波窗口大小的选择等，下面将对部分参数对分割结果的影响进行简单讨论。

3.3 对比实验

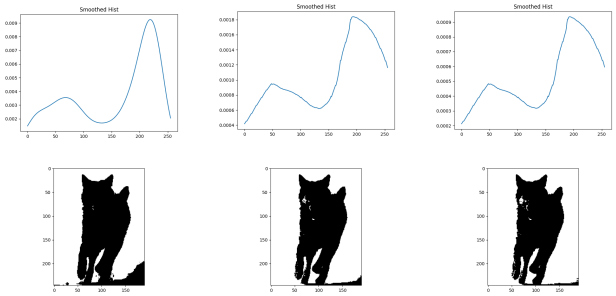
在本文实现的基于直方图的自适应阈值分割方法中，采用了一维高斯滤波器对图像的灰度直方图进行去噪，因此，除了滑动窗口大小的选择外，一维高斯滤波器的参数也对分割结果有一定的影响。这一部分，我们将对一维高斯滤波 σ 大小和一维高斯滤波窗口大小对分割结果的影响进行讨论。

3.3.1 一维高斯滤波 σ 大小的影响

在探究 σ 大小对阈值选取和分割结果的影响时，固定其他参数不变，σ 分别选取 16、128 和 256，在图像 Test_Img_2 上进行实验，阈值的选取如表 2 所示，分割结果如图 5 所示。

表 2: 不同 σ 一维高斯滤波得到的自适应阈值表

	$\sigma = 16$	$\sigma = 128$	$\sigma = 256$
阈值	145	120	114



(a) (b) (c)

图 5: 不同 σ 一维高斯滤波得到的自适应阈值分割结果：(a) $\sigma = 16$, (b) $\sigma = 128$, (c) $\sigma = 256$. 第一行为经过一维高斯滤波后的灰度分布统计图，第二行为基于自适应阈值分割的结果

结合表 2 和图 5 可以看出，随着 σ 的增大，阈值的选取逐渐减小，对图片的分割结果也有比较大的影响。图像 Test_Img_2 中目标区域灰度值整体偏低，背景区域灰度值偏高，而随着阈值的增大，可以有效地减少背景中被错误分割到目标区域的像素数量（如图像右下部分的区域），但同时，目标中灰度值偏高的部分像素也会随之被分割到背景区域中（如狼的面部、脚等）。

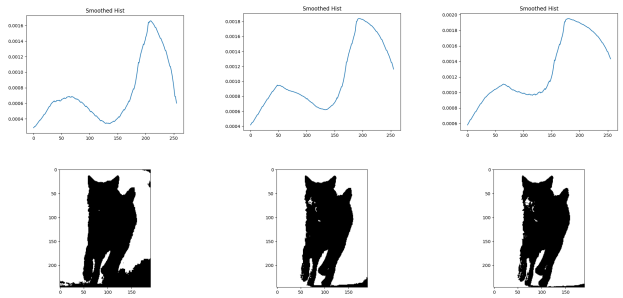
3.3.2 一维高斯滤波窗口大小的影响

在探究滤波滑动窗口大小对阈值选取和分割结果的影响时，固定其他参数不变，窗口大小分别选取 64、96 和 128，在图像 Test_Img_2 上进行实验，阈值的选取如表 3 所示，分割结果如图 6 所示。

表 3: 不同滤波窗口大小得到的自适应阈值表

	window size=64	window size=96	window size=128
阈值	160	120	112

结合表 3 和图 6 可以看出，随着滤波滑动窗口的增大，阈值的选取逐渐减小，对图片的分割结果也有比较大的影响。其对图片分割的影响在上一小节已经进行了阐述，此处不再赘述。滤波滑动窗口的增大会导致阈值的选取减小，是由于滑动窗口增大在总体趋势上会导致灰度值分布更加平滑，噪声更少，使得对混合高斯分布的拟合更加准确。



(a) (b) (c)

图 6: 不同滤波窗口大小得到的自适应阈值分割结果：(a) window size=64, (b) window size=96, (c) window size=128. 第一行为经过一维高斯滤波后的灰度分布统计图，第二行为基于自适应阈值分割的结果

综合上述实验结果来看，基于直方图的自适应阈值分割方法相比较人工设定阈值更加高效、准确，但其阈值的选取也受到其他因素的影响，如滤波滑动窗口的大小等。同时，也不难分析得出该方法对噪声敏感，鲁棒性较差。

4 总结

本文主要对基于阈值的图像分割方法进行了探讨和实验，并重点讨论了基于直方图的自适应阈值分割方法。相比较人工设置阈值而言，基于直方图的自适应阈值分割方法可以更高效地寻找到可以较为准确地将灰度图像分割为目标和背景的阈值，计算简单、效率较高。但该方法也同样存在一定的问题，一方面是依赖于图像灰度值服从混合高斯分布的假设，但真实的图像分布并不一定是服从高斯分布的，一旦这个假设不成立，那么该方法的准确性也会受到一定的影响；另一方面该方法只考虑了像素的灰度值，而一般不考虑图像的空间特征，使其对噪声比较敏感，鲁棒性不高。