图像处理 第一次作业

数学三班 李岳锴 200810301

图像处理基础知识

1. 什么是像素，什么是图像的采样和量化，图像的采样和量化对图像有什么影响?

数字图像由二维的元素组成，每一个元素具有一个特定的位置(x,y)和幅值$f(x,y)$，这些元素称为像素。数字图像是由被称作像素的小块区域组成的二维矩阵。将物理图像行列划分后，每个小块区域即称为像素。对于单色即灰度图像而言，每个像素的的亮度用一个数值来表示，通常数值范围在0到255之间，即可用一个字节来表示。0表示黑，255表示白，而其他表示灰度级别。

图像采样和量化对图像的影响主要体现在图像空间分辨率和图像灰度分辨率两个方面。**图像采样是指将连续的图像信号转换为离散的像素值**。采样的过程中，每个像素的大小和位置都会影响图像的空间分辨率。**图像量化是指将图像灰度值映射到一组离散的取值范围内**。量化的过程中，每个像素的灰度值会被近似为最接近的离散值，这会影响图像的灰度分辨率。

采样率越高，即每单位长度所包含的像素数越多，图像的空间分辨率就越高，图像能够显示更多的细节和纹理；相反，采样率越低，即每单位长度所包含的像素数越少，图像的空间分辨率就越低，图像会变得模糊或失真。量化级别越高，即灰度值被近似为更多的离散值，图像的灰度分辨率就越高，图像能够显示更多的灰度层次，细节更加丰富；相反，量化级别越低，即灰度值被近似为更少的离散值，图像的灰度分辨率就越低，图像会出现灰度级别不足、颜色失真等问题。

2. 给定一副彩色图像：

(1) 读入彩色图像：

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

image = cv2.imread("dove.jpg")
```

(2) 将彩色图像转为灰度图像：

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

(3) 将灰度图像转化为二值图像,将二值图像保存成图像：

```
_, binary_image = cv2.threshold(gray_image, threshold_value, 255, cv2.THR
ESH_BINARY)

cv2.imwrite(output_path, binary_image)
```

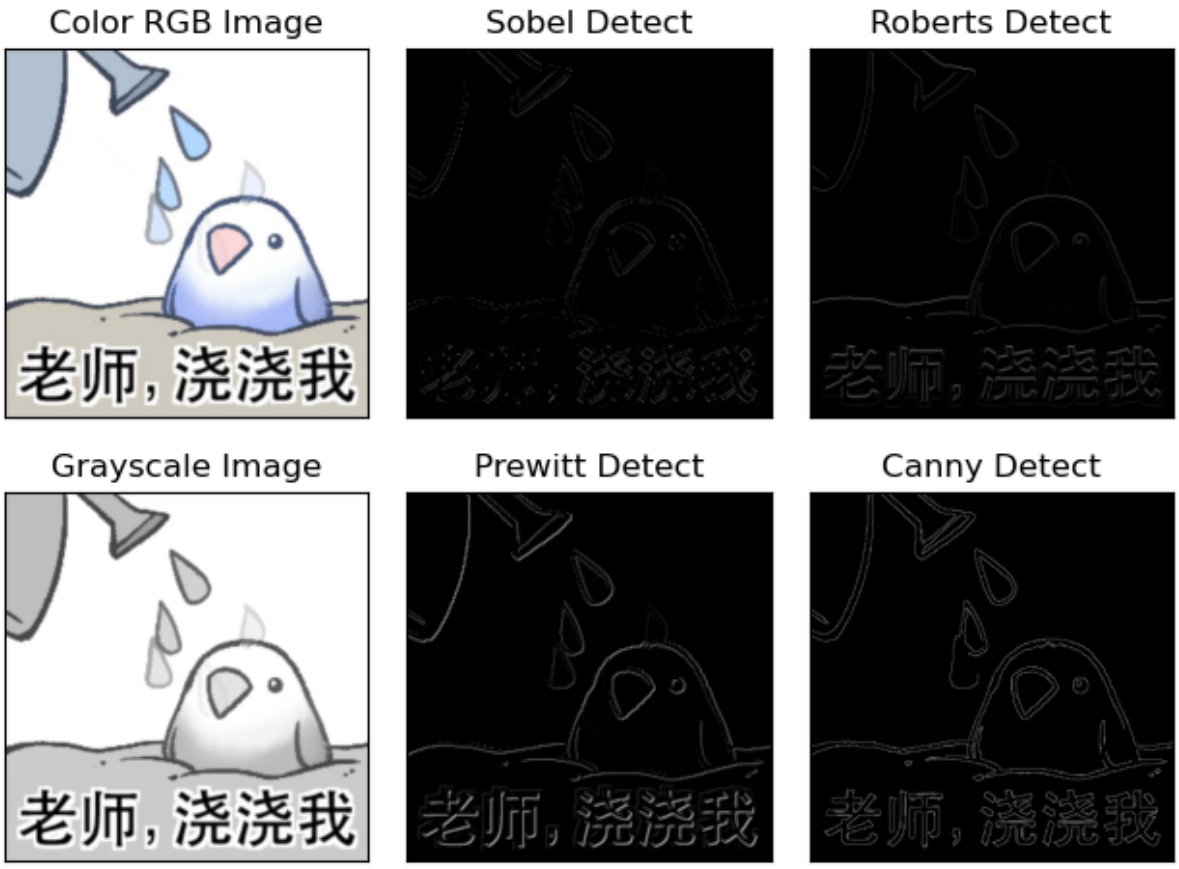
(4) 用sobel, prewitt, roberts, canny方法检测灰度图像边缘：

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg


def Sobel_detect_edges(image_path, output_path, threshold1, threshold2):
    # 读取灰度图像
    gray_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # 使用Sobel方法检测边缘
    edges = cv2.Sobel(gray_image, cv2.CV_64F, 1, 1, ksize=3)

    # # 应用阈值来获取二值化边缘图像
    # edges = cv2.Canny(edges, threshold1, threshold2)

    # 保存处理后的图像
    cv2.imwrite(output_path, edges)


def Prewitt_detect_edges(image_path, output_path):
    # 读取灰度图像
    gray_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # 定义Prewitt算子
    kernel_x = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]], dtype=np.fl
oat32)
    kernel_y = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]], dtype=np.fl
oat32)

    # 使用filter2D函数进行边缘检测
```

```
    edges_x = cv2.filter2D(gray_image, -1, kernel_x)
    edges_y = cv2.filter2D(gray_image, -1, kernel_y)

    # 将水平和垂直边缘相加
    edges = cv2.addWeighted(edges_x, 0.5, edges_y, 0.5, 0)

    # 保存处理后的图像
    cv2.imwrite(output_path, edges)


def Roberts_detect_edges(image_path, output_path):
    # 读取灰度图像
    gray_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # 定义Roberts算子
    kernel_x = np.array([[1, 0], [0, -1]], dtype=np.float32)
    kernel_y = np.array([[0, 1], [-1, 0]], dtype=np.float32)

    # 使用filter2D函数进行边缘检测
    edges_x = cv2.filter2D(gray_image, -1, kernel_x)
    edges_y = cv2.filter2D(gray_image, -1, kernel_y)

    # 将水平和垂直边缘取绝对值并相加
    edges = cv2.add(np.abs(edges_x), np.abs(edges_y))

    # 保存处理后的图像
    cv2.imwrite(output_path, edges)


def Canny_detect_edges(image_path, output_path, threshold1, threshold2):
    # 读取灰度图像
    gray_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # 使用Canny算法进行边缘检测
    edges = cv2.Canny(gray_image, threshold1, threshold2)

    # 保存处理后的图像
    cv2.imwrite(output_path, edges)


def display_picture(img1, img2, img3, img4, img5, img6):
    # 读取图片
    im1 = mpimg.imread(img1)
    im2 = mpimg.imread(img2)
    im3 = mpimg.imread(img3)
    im4 = mpimg.imread(img4)
    im5 = mpimg.imread(img5)
    im6 = mpimg.imread(img6)

    # 创建一个包含3个子图的窗口
    fig, axs = plt.subplots(2, 3)
```

```
    # 在第一个子图中显示彩色RGB图像
    axs[0][0].imshow(im1)
    axs[0][0].set_title('Color RGB Image')
    axs[0][0].set_xticks([])
    axs[0][0].set_yticks([])

    # 在第二个子图中显示灰度图像
    axs[1][0].imshow(im2, cmap='gray')
    axs[1][0].set_title('Grayscale Image')
    axs[1][0].set_xticks([])
    axs[1][0].set_yticks([])

    # 在第三个子图中显示Sobel检测图像
    axs[0][1].imshow(im3, cmap='gray')
    axs[0][1].set_title('Sobel Detect')
    axs[0][1].set_xticks([])
    axs[0][1].set_yticks([])

    # 在第四个子图中显示Prewitt检测图像
    axs[1][1].imshow(im4, cmap='gray')
    axs[1][1].set_title('Prewitt Detect')
    axs[1][1].set_xticks([])
    axs[1][1].set_yticks([])

    # 在第五个子图中显示Roberts检测图像
    axs[0][2].imshow(im5, cmap='gray')
    axs[0][2].set_title('Roberts Detect')
    axs[0][2].set_xticks([])
    axs[0][2].set_yticks([])

    # 在第六个子图中显示腐蚀图像
    axs[1][2].imshow(im6, cmap='gray')
    axs[1][2].set_title('Canny Detect')
    axs[1][2].set_xticks([])
    axs[1][2].set_yticks([])

    # 设置子图之间的间距
    plt.tight_layout()

    # 显示窗口
    plt.show()


# 调用函数进行边缘检测
Sobel_detect_edges('dove_grey.jpg', 'dove_sobel_edge.jpg',
threshold1=100, threshold2=200)
Prewitt_detect_edges('dove_grey.jpg', 'dove_prewitt_edge.jpg')
Roberts_detect_edges('dove_grey.jpg', 'dove_roberts_edge.jpg')
Canny_detect_edges('dove_grey.jpg', 'dove_canny_edge.jpg',
threshold1=100, threshold2=200)

display_picture('dove.jpg', 'dove_grey.jpg', 'dove_sobel_edge.jpg',
```

```
                'dove_prewitt_edge.jpg', 'dove_roberts_edge.jpg', 'dove_c
anny_edge.jpg')
```



第一章 相关经典模型

1. 请写出CV模型与RSF模型在水平集框架下的能量泛函，并解释能量泛函中各项的意义；另外CV模型中$c1, c2$与RSF模型中$f1(x), f2(x)$的区别，CV与RSF模型的区别，并简述CV模型与RSF模型各自的优缺点。

(1) CV模型能量泛函：

$$
\begin{aligned}
F^{CV}(c_1, c_2, \phi) = \lambda_1 \int_{\Omega} |u_0(x,y) - c_1|^2 H(\phi(x,y)) dxdy \\
+\lambda_2 \int_{\Omega} |u_0(x,y) - c_2|^2 [1 - H(\phi(x,y))] dxdy \\
+\nu \int_{\Omega} |\nabla H(\phi(x,y))| dxdy
\end{aligned}
$$

其中：

$$H(z)=\begin{cases}1, z\geq 0\\ 0, z<0\end{cases}$$

正规化后的能量泛函：

$$\begin{aligned}F_{\varepsilon}^{CV}(c_1,c_2,\phi)=&\lambda_1\int_{\Omega}|u_0(x,y)-c_1|^2H_{\varepsilon}(\phi(x,y))dxdy\\&+\lambda_2\int_{\Omega}|u_0(x,y)-c_2|^2[1-H_{\varepsilon}(\phi(x,y))]dxdy\\&+\nu\int_{\Omega}|\nabla H_{\varepsilon}(\phi(x,y))|dxdy\end{aligned}$$

其中：

$$H_{\varepsilon}(z)=\frac{1}{2}\left[1+\frac{2}{\pi}\arctan(\frac{z}{\varepsilon})\right]$$

CV模型中，λ_1和λ_2对应的第一项和第二项是数据的拟合项。在极小化的过程中，尽量保证区域内和区域外的原图像强度u_0与常数$c1, c2$贴近。ν对应的第三项为长度项，是用来保证图像边界的正规性，使得图像的边界更加简单。

(2) RSF模型能量泛函：

$$\begin{aligned}F^{RSF}(\phi,f_1,f_2)=&\sum_{i=1}^{2}\lambda_i\int_{\Omega}\left(\int_{\Omega}K_{\sigma}(x-y)|u_0(y)-f_i(x)|^2M_i(y))dy\right)dx\\&+\nu\int_{\Omega}|\nabla H(\phi(x))|dx\\&+\mu\int_{\Omega}\frac{1}{2}(|\nabla\phi(x)|-1)^2dx\end{aligned}$$

其中$M_1(\phi)=H(\phi), M_2(\phi)=1-H(\phi)$，核函数通常取：

$$K_{\sigma}(u)=\frac{1}{2\pi\sigma^2}e^{-|u|^2/2\sigma^2}$$

正则化后的能量泛函：

$$\begin{aligned}F^{RSF}(\phi,f_1,f_2)=&\sum_{i=1}^{2}\lambda_i\int_{\Omega}\left(\int_{\Omega}K_{\sigma}(x-y)|u_0(y)-f_i(x)|^2M_i^{\varepsilon}(\phi(y))dy\right)dx\\&+\nu\int_{\Omega}|\nabla H_{\varepsilon}(\phi(x))|dx\\&+\mu\int_{\Omega}\frac{1}{2}(|\nabla\phi(x)|-1)^2dx\end{aligned}$$

其中$M_1^{\varepsilon}(\phi)=H_{\varepsilon}(\phi), M_2^{\varepsilon}(\phi)=1-H_{\varepsilon}(\phi)$

RSF模型中，λ_1和λ_2对应的第一项和第二项是数据的拟合项，由核函数控制图像的局部信息，从而近似拟合区域内的图像强度。第三项为长度项，作用与CV模型中相同，用来保证曲线的简洁；

第四项为水平集正则项，是用来保持水平集函数ϕ的正规性，从而也保证了计算的精确性，同时也避免了重新初始化水平集函数的过程。

(3) CV模型中c_1, c_2与RSF模型中$f_1(x), f_2(x)$的区别：

CV模型中的c_1, c_2是常量，用于近似轮廓线C两侧的区域$inside(C)$和$outside(C)$内的图像强度。而RSF模型中的$f_1(x), f_2(x)$是拟合函数，用于拟合C两侧的区域内的图像强度，其值随着中心点x的变化而变化。因此，RSF模型更加灵活和准确，能够更好地适应图像的变化和复杂性。

(4) CV模型与RSF模型的区别：

① 模型原理：CV模型是一个基于区域的活动轮廓模型，通过拟合两个常量来近似轮廓线两侧的区域内的图像强度。而RSF模型利用核函数和拟合函数来近似区域内的图像强度，通过局部化的核函数和拟合函数来充分利用图像的局部信息。

② 图像强度与空间变化性质：CV模型通过两个常量（c_1, c_2）来近似区域内的图像强度，假设区域内的图像强度是均匀的。而RSF模型通过拟合函数（$f_1(x), f_2(x)$）来近似区域内的图像强度，能够处理图像强度不均匀的情况。另外，CV模型中的常量c_1, c_2是全局常量，不随空间位置变化。而RSF模型中的拟合函数$f_1(x), f_2(x)$是随着空间位置x的变化而变化的，具有空间变化性质。

③ 分割性能：CV模型在处理具有同质区域的图像分割任务时表现较好，对于简单纹理和边界的保留能力较强。而RSF模型在处理具有图像强度不均匀性质和具有弱边界的物体分割时效果更好。

(5) CV模型的优缺点：

优点：

- 基于Mumford-Shah分割技巧和水平集方法，能够准确检测和保留边缘位置，不需要光滑初始图像。
- 可以处理非梯度定义的边缘和光滑边缘，对于一些经典活动轮廓模型无法处理的情况更适用。
- 可以从任意位置的一条初始曲线自动检测内部轮廓，不需要围绕待检测物体的完整轮廓。

缺点：

- 对具有不均匀强度的图像分割效果较差。CV模型仅考虑全局图像强度信息，无法处理图像强度不均匀的情况。
- 对于具有不同质区域的图像分割效果较好，但对于具有不同质区域和复杂纹理的图像分割能力有限。

(6) RSF模型的优缺点：

优点：

- 利用拟合函数$f_1(x)$和$f_2(x)$来近似轮廓线两侧区域内的图像强度，充分利用了图像的局部强度信息，能够处理具有图像强度不均匀性质的图像。
- RSF模型的核函数K_σ具有局部化性质，通过控制尺度参数σ，可以在可控制尺度的局部区域内充分利用图像的强度信息，从而更好地引导活动轮廓线的移动。
- 对于具有弱边界的物体如血管等，RSF模型有较好的分割效果。

缺点：

- RSF模型仅利用图像的局部信息，可能导致能量泛函存在局部极小，因此分割结果更加依赖于轮廓线的初始化。
- 由于RSF模型是非凸的，存在局部极小解的问题，可能导致分割结果不稳定。
- RSF模型的拟合函数$f_1(x)$和$f_2(x)$的空间变化性质使得模型更加复杂，计算复杂度可能较高。

2. 请写出Split Bregman方法求解如下L1正则问题的一般步骤：

$$\min_{u} |\Phi(u)|_1 + \mathrm{H}(u)$$

① 引入辅助变量d，则原问题等价于解如下约束极小化问题：

$$\min_{u,d} |d|_1 + H(u)$$
$$s.t.\, d = \Phi(u)$$

② 引入二次约束函数，将①中约束极小化问题转化为无约束极小化问题：

$$\min_{u,d} |d|_1 + H(u) + \frac{\lambda}{2} \|d - \Phi(u)\|^2$$

③ 考虑到二次约束函数仅仅近似地或者称作弱地强制约束条件$d = \Phi(u)$,而事实上希望精确地或严格地强制该约束条件，因此考虑Split Bregman迭代算法：

$$(u^{k+1}, d^{k+1}) = \arg\min_{u,d} |d|_1 + H(u) + \frac{\lambda}{2} \|d - \Phi(u) - b^k\|^2$$
$$b^{k+1} = b^k + (\Phi(u^{k+1}) - d^{k+1})$$

将原L1 正则问题转化为求解一系列无约束优化问题和Bregman迭代的问题.

④分别关于u和d交替极小化：

$$u^{k+1} = \arg\min_{u} H(u) + \frac{\lambda}{2} \|d^k - \Phi(u) - b^k\|^2$$
$$d^{k+1} = \arg\min_{d} |d|_1 + \frac{\lambda}{2} \|d - \Phi(u^{k+1}) - b^k\|^2$$

对于第一步，因为已经将u从$L1$项中分离出来，关于u的优化问题现在是可微的，因此可以用很多种优化方法来解决该问题。用哪一种特定的优化方法来解决该问题取决于函数$H(\cdot)$的性质。

对于第二步，利用向量值shrinkage算子显式地计算出d：

$$d^{k+1} = \mathrm{shrink}(\Phi(u^{k+1}) + b^k, \frac{1}{\lambda})$$

其中$\mathbf{shink}(\mathbf{x}, \gamma) = \begin{cases} \frac{\mathbf{x}}{|\mathbf{x}|} \max(|\mathbf{x}| - \gamma, 0), \mathbf{x} \neq 0 \\ 0, \mathbf{x} = 0 \end{cases}$