

实现功能

- 手动实现导向滤波算法
- 手动实现双边滤波算法
- 基于两种算法实现图像锐化功能

代码介绍

导向滤波

1. 计算每个窗口内I、p的均值
2. 计算I、p的方差及协方差
3. 计算系数a、b
4. 由于每一个像素处在多个窗口内，计算a、b的均值
5. 计算输出图像q

代码如下：

```
def guided_filter(I, p, r, eps):  
    """  
    实现导向滤波  
    Args:  
        I: 导向图像  
        p: 输入图像  
        r: 滤波窗口半径  
        eps: 正则化参数  
  
    Returns:  
        过滤后的图像  
    """  
    # 计算窗口内均值  
    kernel = (2 * r + 1, 2 * r + 1)  
    mean_I = cv2.blur(I, kernel) # I 的均值  
    mean_p = cv2.blur(p, kernel) # p 的均值  
    mean_Ip = cv2.blur(I * p, kernel) # I 和 p 的联合均值  
  
    # 计算协方差和方差  
    cov_Ip = mean_Ip - mean_I * mean_p # 协方差  
    mean_II = cv2.blur(I * I, kernel) # I 的平方均值  
    var_I = mean_II - mean_I * mean_I # I 的方差  
  
    # 计算线性系数 a 和 b  
    a = cov_Ip / (var_I + eps)  
    b = mean_p - a * mean_I  
  
    # 计算 a 和 b 的均值  
    mean_a = cv2.blur(a, kernel)  
    mean_b = cv2.blur(b, kernel)  
  
    # 输出图像  
    q_out = mean_a * I + mean_b
```

```
return q_out
```

双边滤波

双边滤波我们需要根据两个像素点之间的空间距离和像素值距离计算kernel，代码如下：

```
def bilateral_filter(img, d, sigma_color, sigma_space):
    """
    实现双边滤波
    Args:
        img: 输入图像
        d: 滤波窗口直径
        sigma_color: 颜色空间滤波器的 sigma 值
        sigma_space: 坐标空间滤波器的 sigma 值

    Returns:
        过滤后的图像
    """
    half_d = d // 2
    img_padded = np.pad(img, ((half_d, half_d), (half_d, half_d), (0, 0)),
mode='reflect')
    filtered_img = np.zeros_like(img)

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            for k in range(img.shape[2]):
                i_min = i
                i_max = i + d
                j_min = j
                j_max = j + d

                region = img_padded[i_min:i_max, j_min:j_max, k]
                center_pixel = img_padded[i + half_d, j + half_d, k]

                spatial_weights = np.exp(-((np.arange(d) - half_d) ** 2 +
(np.arange(d)[: , None] - half_d) ** 2) / (2 * sigma_space ** 2))
                color_weights = np.exp(-(region - center_pixel) ** 2 / (2 *
sigma_color ** 2))

                weights = spatial_weights * color_weights
                weights /= np.sum(weights)

                filtered_img[i, j, k] = np.sum(weights * region)

    return filtered_img
```

在循环中的主要操作为：

1. 取出窗口中像素
2. 计算坐标权重和颜色权重
3. 相乘得到完整滤波器
4. 与窗口中像素相乘，得到中心像素的滤波结果

锐化

锐化的大概思路为：

1. 通过滤波操作得到平滑部分
2. 原图像减去平滑部分得到高频部分
3. 将高频部分乘以锐化系数后加在原图像上

代码为：

```
def sharpen_bilateral(img, d, sigma_color, sigma_space, lambda_factor):  
    img = img.astype(np.float32) / 255  
  
    sharpened_img = np.zeros_like(img)  
  
    smooth = bilateral_filter(img, d, sigma_color, sigma_space)  
    high_freq = img - smooth  
    sharpened_img = img + lambda_factor * high_freq  
  
    sharpened_img = np.clip(sharpened_img, 0, 1) * 255  
    return sharpened_img.astype(np.uint8)
```

使用导向滤波进行锐化操作也基本相同。

图像处理网页演示工具

使用方式，在浏览器中打开<http://127.0.0.1:8088/>即可


此网页演示提供以下图像处理工具:




- 作业1: 色相/饱和度/亮度调整工具
- 作业2: 图像缩放工具
- 作业3: DCGAN图像生成工具
- 作业4: XXX工具
- 作业5: XXX工具

作业1: 色相/饱和度/亮度调整工具 作业2: 图像缩放工具 作业3: DCGAN图像生成工具 **作业4: 图像去噪工具** 作业5: XXX工具


作业四: 图像去噪工具




待去噪图像






输出图像









导向图像






将图像拖放到此处
- 或 -
点击上传



锐化图像





运行

滤波模式

☒ 双边滤波

☐ 导向滤波

窗口半径 (导向滤波)

4

10

eps (导向滤波)

0.01

1

颜色空间滤波器的 sigma 值 (双边滤波)

75

100

坐标空间滤波器的 sigma 值 (双边滤波)

75

100

锐化系数

1

5

通过 API 使用

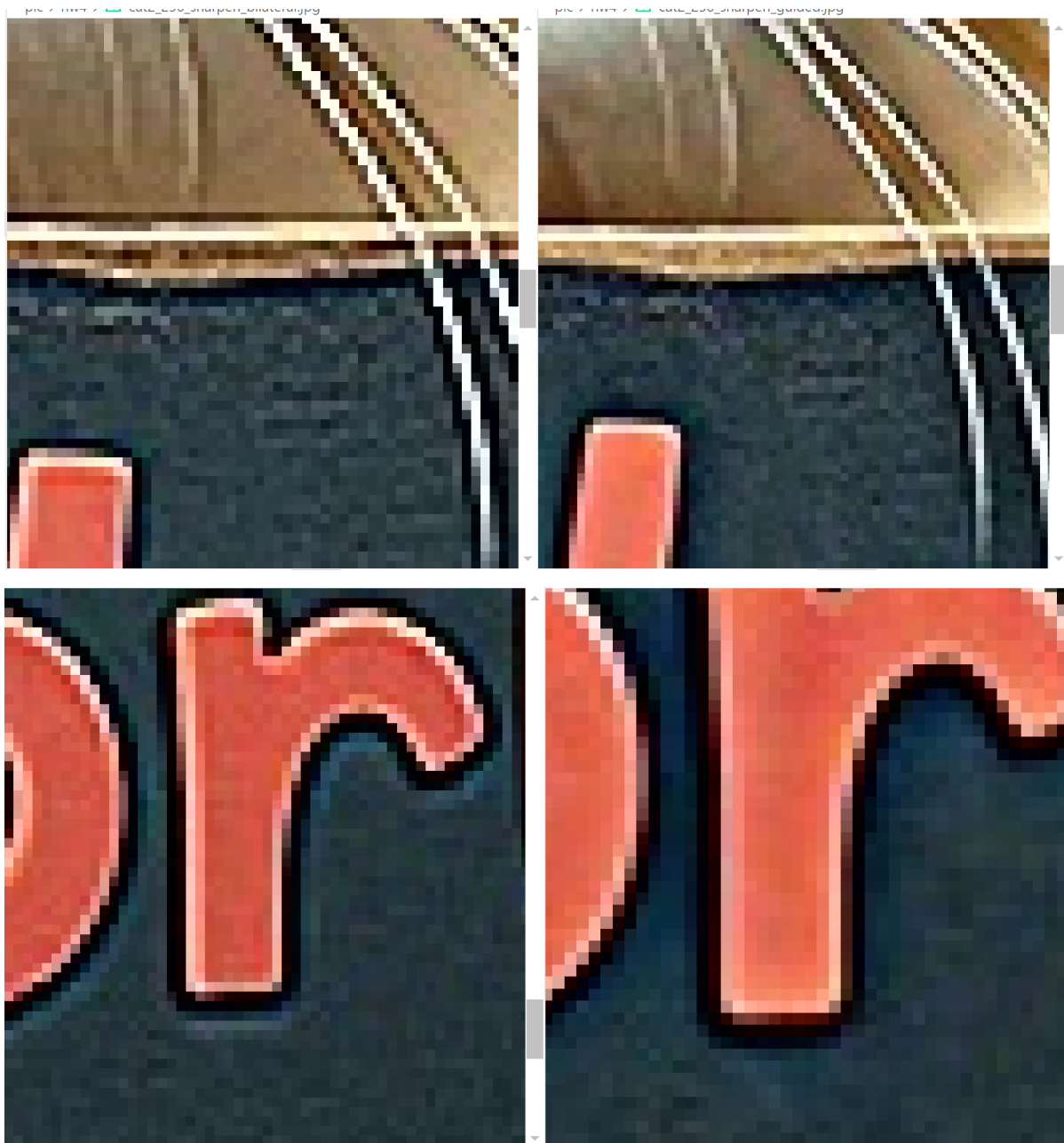
使用 Gradio 构建

一共分两栏，左栏的上半部分可以输入待去噪图像与导向图像（可选）。如果没有输入导向图像的话，则默认设置为待去噪图像本身。左栏的下半部分可以选择滤波模式，并选择相应的参数。

右栏则显示滤波图像与锐化后的图像。

实验结果

锐化



左为双边滤波，右为导向滤波，可以发现在边缘细节处二者的表现有些许的不同。双边滤波体现出了类似梯度反转的现象，表现为：上图增加了本不存在的深色边缘，下图的字体边缘更亮。

去噪



三张图片依次为：噪声图像、导向滤波和双边滤波。可以看出导向滤波相较而言效果更好：噪点的去除更加彻底，对于边缘的保护也较好。