图像处理 HW4：图像去噪实操

李天宇　　2200013188　　信息科学技术学院

1　基本实现

又到了我最爱的调库就能拿 60 分时间了。

1.1　双边滤波

OpenCV 对于双边滤波的实现如下：

`cv.bilateralFilter(src, d, sigmaColor, sigmaSpace[, dst[, borderType]])->dst`

其中`d`是卷积核的大小，而对于高斯函数中的 σ，这里将颜色空间和坐标空间的 σ 分开计算了，分别为`sigmaColor`和`sigmaSpace`。

根据官方文档，d 一般取 5 或 9（需要重噪声过滤的离线应用程序，较为极端的情况）。而为了简单起见，两个`sigma`也可以设为相同，一般取 $(10, 150)$，这里就取个较为中间的值 75（其实是 Copilot 的自动补全填了个 75（乐））。

1.2　NLM 滤波

NLM 滤波使用了`fastNlMeansDenoisingColored()`函数，定义如下：

`cv.fastNlMeansDenoisingColored(src[,dst[,h[,hColor[,templateWindowSize[,searchWindowSize]]]]])->dst`（这名字太长一行都打不下了）

这个函数将图像转化为 $CIELAB$ 颜色空间，然后使用`fastNlMeansDenoising`函数分别去噪具有给定 h 参数的 L 和 AB 分量。后面的四个参数都有默认值，就直接不写了。

2　基础功能实现：导向滤波

2.1　OpenCV 接口

还是先看看官方接口吧，导向滤波的函数为

`cv.ximgproc.guidedFilter(guide,src,radius,eps[,dst[,dDepth[,scale]]])->dst`

`guide`为引导图像（或图像数组），这里为了方便，直接认为`guide`是单张图像。事实上，我们这里也只能以图像自身作为`guide`。

我设计的函数如下：

Code Listing 1: 导向滤波

```
1    def my_guided_filter(src, guide, r, eps):
2        # 具体实现
3        return dst
```

那么，为了补出这里的具体实现，我们还是需要研究导向滤波究竟如何实现。

2.2 实现方法

1. 输入与假设

   输入图像为待滤波图像 p 和引导图像 I，在局部窗口 ω_k 中假设输出图像 q 满足线性关系：

   $$q_i = a_k I_i + b_k,\ \forall i \in \omega_k$$

   其中 a_k 和 b_k 是局部窗口的线性系数。

2. 优化目标

   通过最小化以下目标函数求解 a_k 和 b_k：

   $$E(a_k, b_k) = \sum((a_k I_i + b_k - p_i)^2 + \epsilon a_k^2)$$

   其中，ϵ 是正则化参数，用于控制平滑度。

3. 解析解

   根据优化目标的闭式解：

   $$a_k = \frac{\mathrm{cov}_k(I, p)}{\mathrm{var}_k(I) + \epsilon},\ b_k = \overline{p}_k - a_k \overline{I}_k$$

   其中，$\mathrm{cov}_k(I, p)$ 和 $\mathrm{var}_k(I)$ 分别是窗口内的协方差和方差，$\overline{p}_k$ 和 $\overline{I}_k$ 是窗口均值。

4. 最终输出

   输出图像 q 是所有窗口线性预测的加权平均：

   $$q_i = \frac{1}{|\omega_i|} \sum_{k:i \in \omega_k} (a_k I_i + b_k)$$

那么，我们只要根据第三步，求出 a_k 和 b_k，然后根据第四步输出图像，这份代码就可以很轻松地完成了（雾）。

2.3 具体代码实现

在没有盒式滤波的前提下，可以直接基于局部窗口逐像素计算均值和其他统计量，这种方法虽然直观，但计算效率较低。对于每个像素，我们做如下步骤：

- 定义窗口范围
- 提取局部窗口
- 计算局部统计量（用`np.mean`）
- 计算线性系数 a 和 b
- 输出 q

3 进阶功能实现

3.1 基于盒式滤波优化的快速导向滤波

盒式滤波器即`boxFilter`，它对输入图像的局部区域进行均值计算，从而快速计算出局部均值和相关性。通过这种方式，导向滤波可以高效地进行图像平滑处理，同时保留边缘细节。实际上，`boxFilter()`函数起到的是模糊图像的作用。

在 OpenCV 的文档中，`boxFilter()`函数使用系数

$$K = \alpha \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix}$$

其中

$$\alpha = \begin{cases} \dfrac{1}{\text{ksize.width} * \text{ksize.height}} & \texttt{when normalize=true} \\ 1 & \texttt{otherwise} \end{cases}$$

未经归一化的盒式滤波器在计算每个像素邻域的各种积分特性时很有用，例如图像导数的协方差矩阵。这里我们用的应该是归一化的盒式滤波器，否则在经过几次滤波后，这些值会变得过大，从而导致图像几乎全变成白色。如下图：

图 1: 不归一化的结果

我们也可以定义一个`box_filter()`：

Code Listing 2: 快速盒式滤波

```
import cv2 as cv

def box_filter(img, r):
    # 快速盒式滤波，用于计算均值
    kernel = np.ones((2 * r + 1, 2 * r + 1), dtype=np.float32)
    return cv.filter2D(img, -1, kernel) / kernel.sum()
```

这里我就直接用官方的盒式滤波了。具体实现就是把原来的逐像素计算变为用盒式滤波分别对 I 和 p 滤波，直接以矩阵计算方差与协方差，从而算出 a, b。之后再用盒式滤波，计算`mean_a`，`mean_b`。

$$q = \texttt{mean\_a} * I + \texttt{mean\_b}$$

3.2 手动实现双边滤波

在前面的调库环节中，已经看到双边滤波函数的参数，据此设计函数：

Code Listing 3: 双边滤波

```
def my_bilateral_filter(input_image, d, sigma_color, sigma_space):
    # 具体实现
    return dst
```

定义高斯函数

$$G_\sigma(x) = \frac{1}{2\pi\sigma^2}\exp(-\frac{x^2}{2\sigma^2})$$

接下来定义两个核：

- 空间域核：$K_s = G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)$
- 灰度域核：$K_r = G_{\sigma_r}(\|I_\mathbf{p} - I_\mathbf{q}\|)$

双边滤波依赖于以下数学式：

$$I'_p = \frac{1}{W_p}\sum_{\mathbf{q}\in S} K_s K_r I_q$$

$$W_p = \sum_{\mathbf{q}\in S} K_s K_r$$

但实际上，高斯函数的系数抵消了，所以在实际计算时：

$$G_\sigma(x) = \exp(-\frac{x^2}{2\sigma^2})$$

最后限制一下范围就可以了。
这里需要注意，参数中是 d，若记 $r = \lfloor \frac{d}{2} \rfloor$，则 $x_q \in [x_p - r, x_p + d - r)$。
逐像素点计算即可。

3.3 各种滤波方法对比

以下是用`cat256.jpg`经去噪处理的结果。

(a) OpenCV (b) 手动实现

图 2: 双边滤波

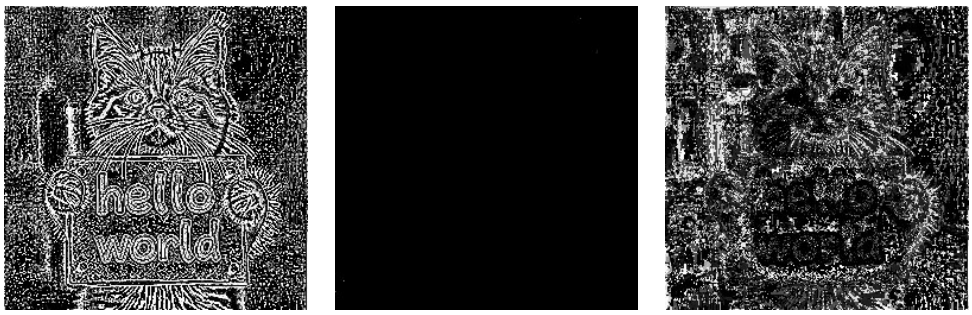
(a) OpenCV (b) 手动实现 (c) 盒式滤波优化

图 3: 导向滤波

对于双边滤波，手动实现与 OpenCV 存在较大差异，主要体现在边缘、锐化上。具体来讲，就是手动实现的更“糊”。

对于导向滤波而言，三者并没有什么肉眼可见的差异。

至于双边和导向在锐化方面的比较，以肉眼观察，可以发现相较于双边滤波而言，导向滤波在边缘处（如毛发）给出了更好的锐化。我们也可以通过图像做差来更好地看出这一点，这里我用的是 OpenCV 的滤波函数:



(a) 双边滤波 (b) 导向滤波 (c) NLM 滤波

图 4: 滤波图像差

图 b 并不是我拿一张黑色图糊弄的（笑），可以看到导向滤波更好地保留了原图的细节，另外，NLM 的画风确实比较清奇。

我也比较了手动/OpenCV 实现的图像差，基本上模式都是一致的，就不赘述了。

回顾发现，自己各个函数的代码风格不一致，实现方法也不尽相同，有的逐像素靠 for 循环，有的调 np 库多一些，下次还是修一修吧。