

Final Project: Haze Removal

2.1 Basic Task

1. Implement the approach described in the paper

1) Introduction of the approach described in the paper

导向滤波器由下面三个式子（记为式①②③）定义：

$$a_k = \frac{\frac{1}{|\omega|} \sum_{i \in \omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon}$$

$$b_k = \bar{p}_k - a_k \mu_k$$

$$q_k = \frac{1}{|\omega|} \sum_{k|i \in \omega_k} (a_k I_i + b_k)$$

它们的推导过程如下：

导向滤波器是一个在导向图 I 和滤波输出 q 之间的局部线性模型。假设 q 是 I 经过以像素 k 为中心的窗口 ω_k 的线性变换得到的：

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

(a_k, b_k) 是假定在 ω_k 中恒定的一些线性系数。使用半径范围为 r 的正方形窗口。由于 $\nabla q = a \nabla I$ ，这个局部线性模型确保了只有 I 有边缘的时候 q 才有边缘。

为了确定 (a_k, b_k) ，我们需要限制滤波输入图像 p 。将输出 q 表示为输入 p 减去一些诸如噪声的不想要的元素：

$$q_i = p_i - n_i$$

为了求得在维持线性模型的同时使得 q 和 p 之间的差值最小化的解，问题转化为求下面这个在窗口 ω_k 下的价值函数最小化：

$$E(a_k, b_k) = \sum_{i \in \omega_k} ((a_k I_i + b_k - p_i)^2 + \epsilon a_k^2)$$

上式中 ϵ 是正则化参数，为了防止 a_k 过大而设置。

解上述方程即可得式①②。

$$a_k = \frac{\frac{1}{|\omega|} \sum_{i \in \omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon}$$

$$b_k = \bar{p}_k - a_k \mu_k$$

μ_k 和 σ_k^2 分别是 I 在 ω_k 中的均值和方差， $|\omega|$ 是 ω_k 中的像素数， $\bar{p}_k = \frac{1}{|\omega|} \sum_{i \in \omega_k} p_i$ 是 p 在 ω_k 中的均值。

考虑到上面的计算中每个像素 i 都被牵涉进覆盖过它的窗口的计算中，通过不同窗口计算的 q_i 是不同的。我们要的结果应当是所有可能的 q_i 的均值（式③）：

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

注意，由于窗口是对称的，有 $\sum_{k|i \in \omega_k} a_k = \sum_{k \in \omega_i} a_k$ ，至此我们可以重新将 q_i 表示成

$$q_i = \bar{a}_i I_i + \bar{b}_i$$

其中， $\bar{a}_i = \frac{1}{|\omega|} \sum_{k \in \omega_i} a_k$, $\bar{b}_i = \frac{1}{|\omega|} \sum_{k \in \omega_i} b_k$ 是所有覆盖 i 的窗口的平均系数。

2) My implementation details

通过上面的分析及定义中的三个式子，论文中给出了实现公式的算法

Algorithm 1 Guided Filter.

```

1: meanI = fmean(I, r)
   meanp = fmean(p, r)
   corrI = fmean(I .* I, r)
   corrIp = fmean(I .* p, r)
2: varI = corrI - meanI .* meanI
   covIp = corrIp - meanI .* meanp
3: a = covIp ./ (varI + ε)
   b = meanp - a .* meanI
4: meana = fmean(a, r)
   meanb = fmean(b, r)
5: q = meana .* I + meanb

```

根据算法一实现function $q = \text{guided_filter}(I, p, r, \text{eps})$ ， eps 即 ϵ 。程序调用了 f_{mean} ，它是时间复杂度为 $O(N)$ 的均值滤波器。导向滤波器的主要计算量都来自于 f_{mean} 。为了达到 $O(N)$ 的时间复杂度，可以采用integral image的技术或一种简单的moving sum的方法。论文中作者给出了moving sum的一维算法。我选择实现的是integral image的方法。

Integral Image是一种快速有效计算一幅图像中指定矩形子区域的像素值之和的方法。对于灰度图而言，这个和除以矩形域内的像素数是计算区域内平均灰度的过程，正是均值滤波器的主要步骤。这也是Integral Image目前的主要应用。

可以注意到，矩形域内像素数的计算是求指定矩形子区域的像素值之和的一种特殊情况，令输入的图像每个像素值为1即得。

因此令function $out = \text{box_filter}(in, r)$ 为求指定矩形子区域的像素值之和的函数。 f_{mean} 通过调用两次该函数实现。第一次 $A = \text{box_filter}(in, r)$ ；

获得没有归一化的像素值之和。第二次 $B = \text{box_filter}(\text{ones}(\text{size}(in, 1), \text{size}(in, 2)), r)$ ；获得归一化的除数。最终返回 $out = A ./ B$ ；

※ box filter具体实现Integral Image的方法和过程。

(1) 计算Summed Area Table

表中的任意点 (x, y) 的值是该点上面和左边所有像素点的值之和：

$$I_{\Sigma}(x, y) = \sum_{\substack{x' \leq x \\ y' \leq y}} i(x', y')$$

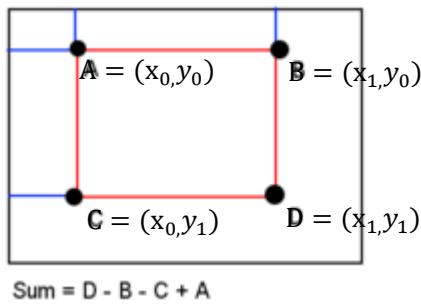
注意不需要每次重新遍历整个左上角区域，而是从左上角开始计算，每计算完一个点都存表，之后计算右下角的点时就可以利用这个结果：

$$I(x, y) = i(x, y) + I(x - 1, y) + I(x, y - 1) - I(x - 1, y - 1)$$

因此每个点的计算都仅读取输入图像中该点的像素值以及已生成的Summed Area Table中的三个值。一共需要三个+/-运行，可在常数时间内完成。

(2) 计算一个区域内的像素值之和

有了第一步的准备，我们只需要取用Summed Area Table中的四个值就可以计算一个区域内的像素值之和了。



上图中，

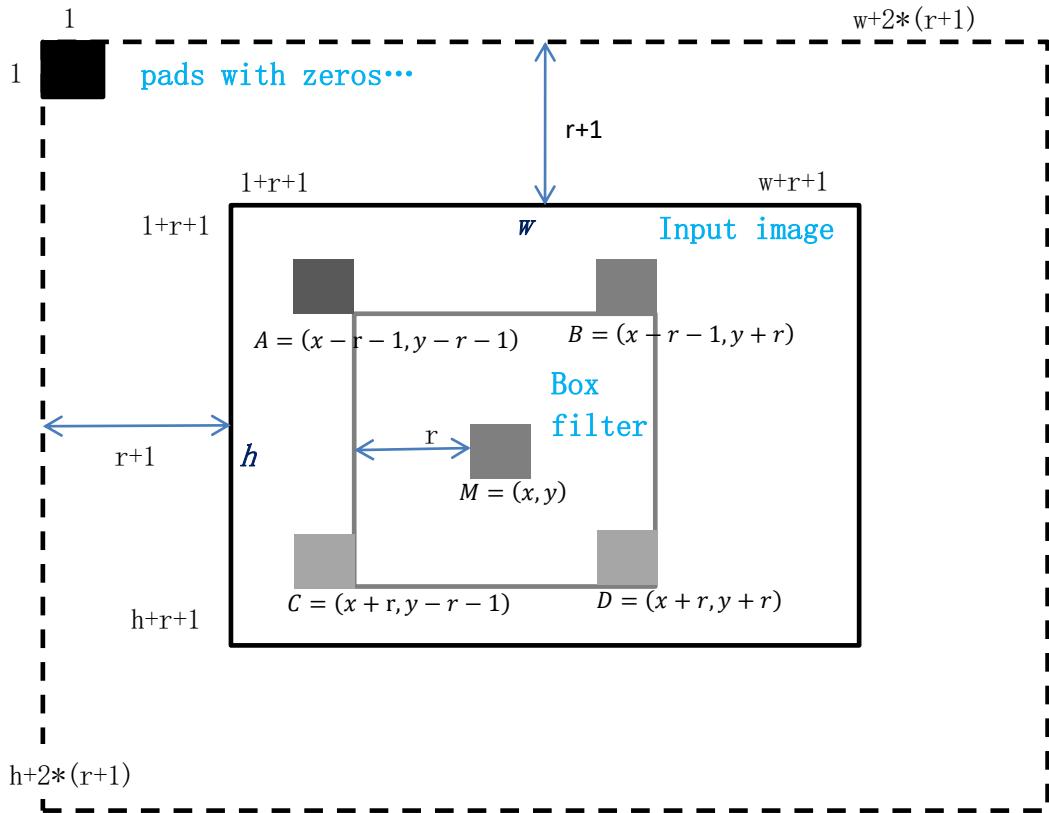
$$\sum_{\substack{x_0 < x \leq x_1 \\ y_0 < y \leq y_1}} i(x, y) = I(D) + I(A) - I(B) - I(C)$$

注意到上面的计算过程只用了常数时间且和矩形区域的大小无关。

针对我们的算法，矩形区域是以计算像素点为中心，到中心的棋盘距离为r以内的区域。

※ 针对该问题的实现(Integral Image):

下图是示意图及相应的符号表示。



(1) 为了对图像中的所有像素统一处理而不区分边界和非边界，我将输入图像填充0之后再进行上面的两个步骤。填充的标准是使得步骤中的所有操作都不会越界。一个最极端的情况是上图中的M点表示原输入图像中的最左上角的一个点A = (x_0, y_0) , 则D = $(x_1, y_1) = (x_0 - r - 1, y_0 - r - 1)$, 因此需要在原图像的基础上向外填充 $r + 1$ 层0.

(2) 生成**Summed Area Table**, 注意边界和上图一致

```

for x = 1 + r + 1 : h + 2 * (r + 1)
    for y = 1 + r + 1 : w + 2 * (r + 1)
        SAT(x, y) = in(x, y) + SAT(x - 1, y) + ...
                    SAT(x, y - 1) - SAT(x - 1, y - 1);
    end
end

```

(3) 计算一个区域内的**像素值之和**

```

for x = 1 + r + 1 : h + r + 1
    for y = 1 + r + 1 : w + r + 1
        out(x, y) = SAT(x + r, y + r) + SAT(x - r - 1, y - r - 1) ...
                    - SAT(x - r - 1, y + r) - SAT(x + r, y - r - 1);
    end
end

```

(4) 最后返回的是去除填充0的部分。

2. Evaluate my implementation by conducting experiments.

1) edge-preserving filtering

IMPLEMENTATION:

当 guide I 和要进行滤波的输入 p 相同时, 导向滤波器可用来进行边缘保持的滤波。

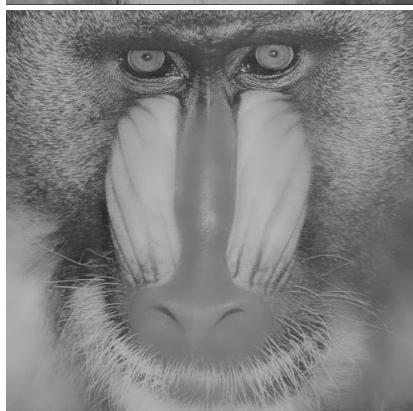
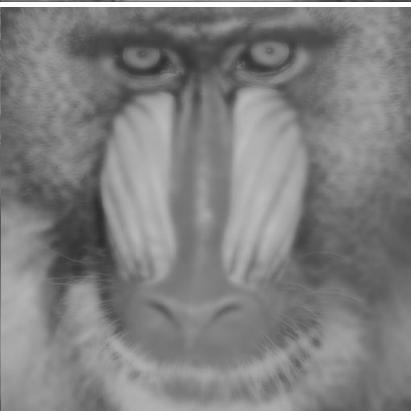
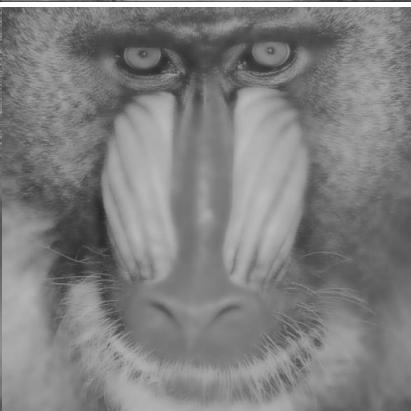
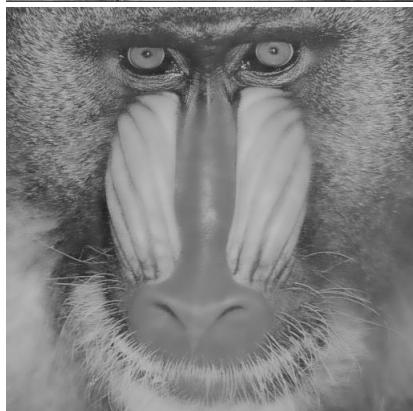
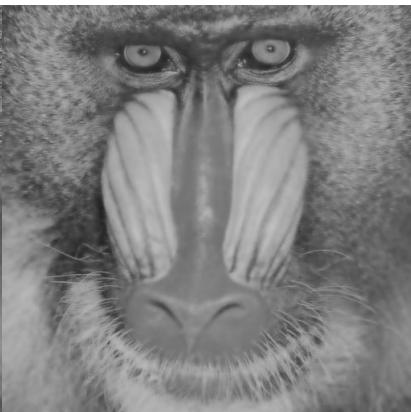
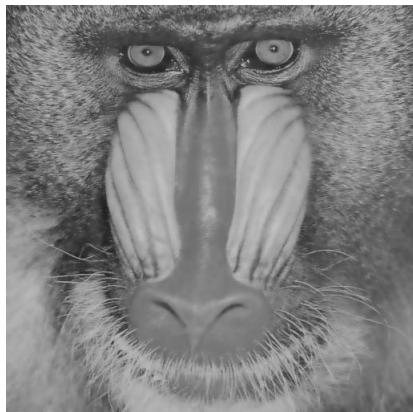
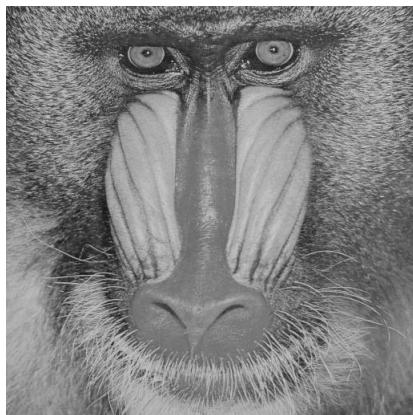
PROCESSING RESULTS:

对每张处理的图片，先贴一张原图，接下来九张图的参数设置如下：

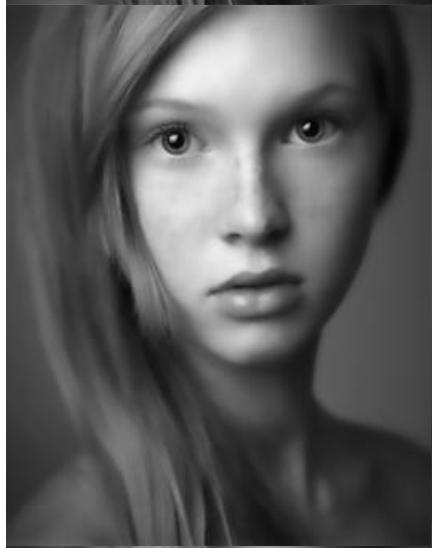
$r = 2, \epsilon = 0.1^2$	$r = 2, \epsilon = 0.2^2$	$r = 2, \epsilon = 0.4^2$
$r = 4, \epsilon = 0.1^2$	$r = 4, \epsilon = 0.2^2$	$r = 4, \epsilon = 0.4^2$
$r = 8, \epsilon = 0.1^2$	$r = 8, \epsilon = 0.2^2$	$r = 8, \epsilon = 0.4^2$

注：所有图片以一定比例（注明在图片名后面的括号内）缩小后粘贴在下面。

baboon(40%)

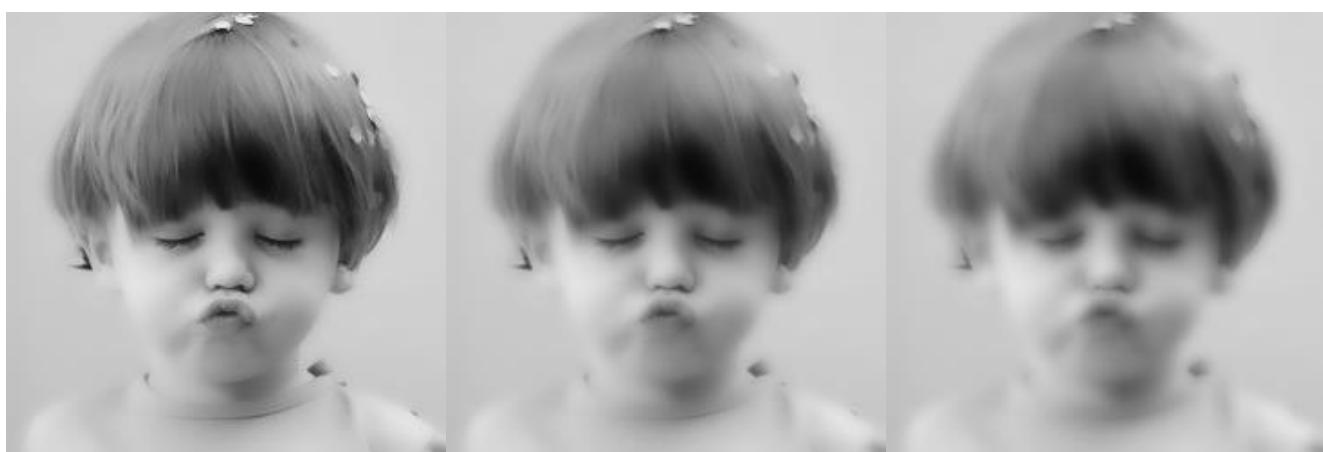
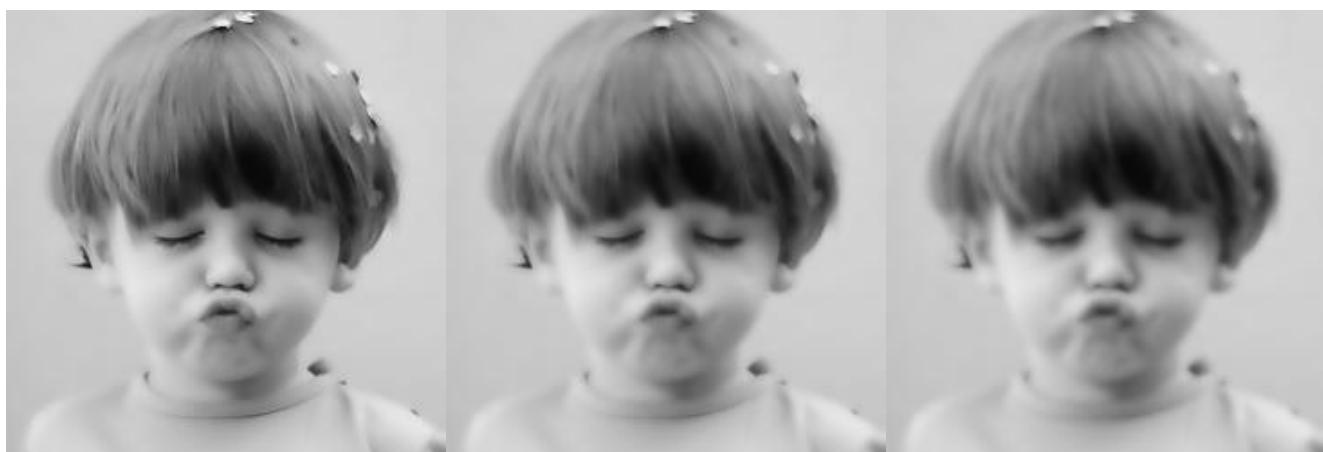


beauty with freckle(80%)





boy(90%)



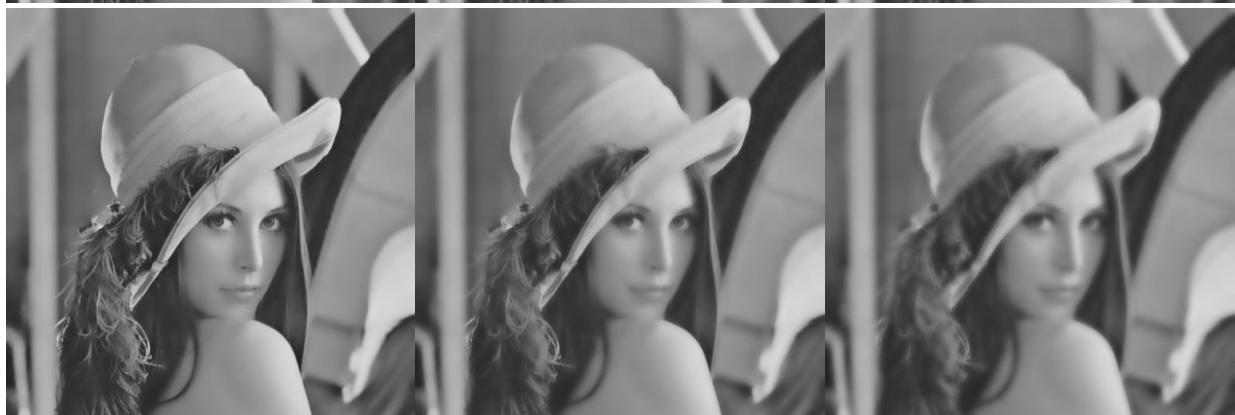
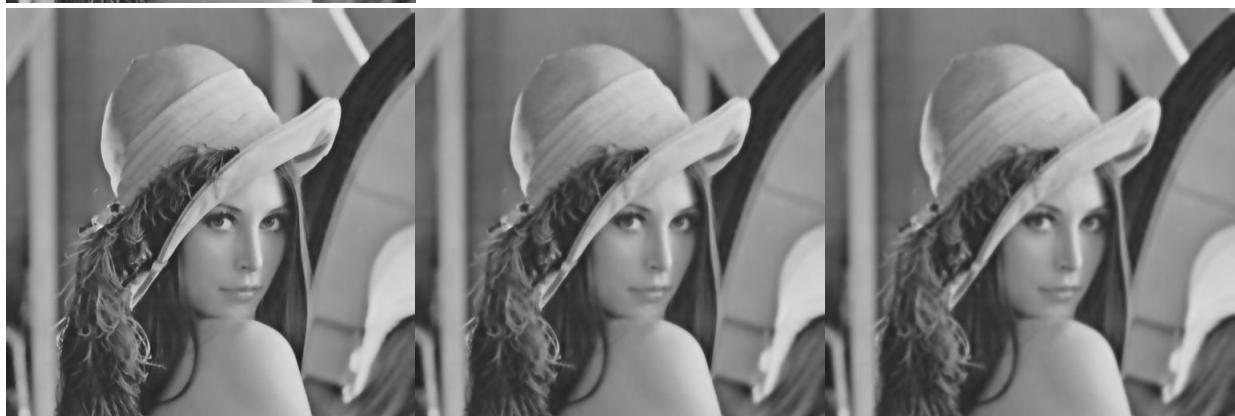


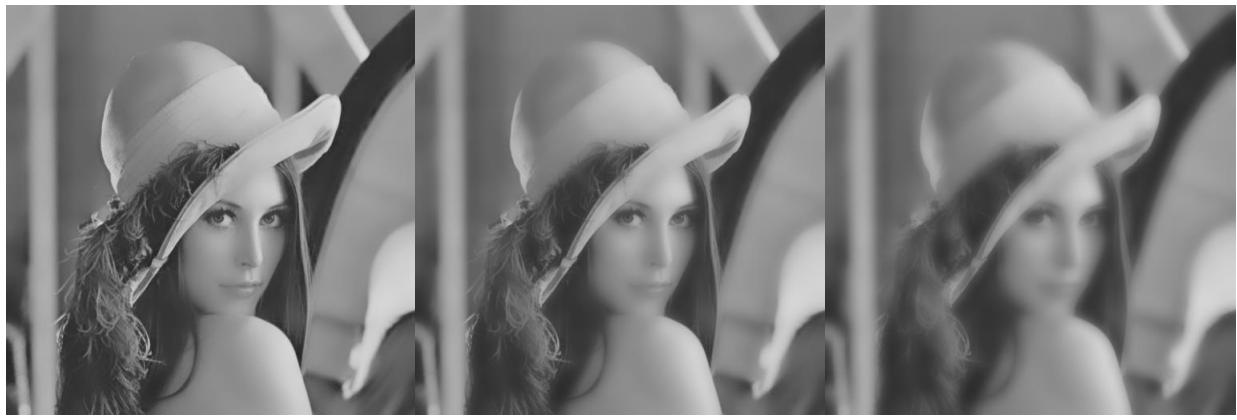
cat(80%)





lena(40%)





ANALYSES:

r 相同时， ϵ 越大，图像越平滑/模糊； ϵ 相同时， r 越大，图像越平滑/模糊。对于上述参数的设置来说， ϵ 的改变对平滑/模糊效果的影响比 r 更明显。

首先 r 和滤波器的大小有关， r 越大，滤波器越大，则越大区域的像素点参与到均值计算中，平滑/模糊效果越强。而参数 ϵ 的影响以及为什么 $I \equiv p$ 可以进行边缘保持滤波在论文的 3.2 节给出了详细的证明和解释。作者分“High variance”和“Flat patch”两种情况进行讨论。可以证明当一个像素在“high variance”区域中间时，它的值是不变的；而在“flat patch”区域中间时，它的值是附近像素值的均值。因为边缘（edge）和“high variance”区域紧密相关，可以做到边缘保持。

具体到公式中表现为图像中 σ^2 远小于 ϵ 的块被平滑，而 σ^2 大于 ϵ 的块保持下来。对于同一图像， σ^2 不变，因此 ϵ 越大，图像被平滑的块越多，保持的边缘越少，图像整体越模糊。

感性角度看，平滑的效果是非常好的，下图分别是美丽的带着雀斑的女孩的原图和参数为 $r = 2, \epsilon = 0.1^2$ 的输出图像（截取并放大了一部分），可以看到雀斑几乎完全被去除且眼睛、头发等边缘仍然得到了一定的保持。



2) detail enhancement

IMPLEMENTATION:

- a) 由于输入的图像是彩色的，而我们实现的 guided filter 只能处理灰度图，因此需要进行拓展。
其实 guided filter 很容易拓展到彩色图像。当 p 是多通道的图像时，直接分别对每个通道运用导向滤波即可。

```
q(:,:,1) = guided_filter(I(:,:,1), p(:,:,1), r, eps);
q(:,:,2) = guided_filter(I(:,:,2), p(:,:,2), r, eps);
q(:,:,3) = guided_filter(I(:,:,3), p(:,:,3), r, eps);
```

下一部分展示的处理结果均采用的是这种方法。

但是要注意的是当任意的单个通道的边缘或细节难以辨别时，这种分别对单通道处理的方法就不是很好了。需要对 guided filter 进行改写以适合处理彩色图像。

首先，重写局部线性模型为

$$q_i = \mathbf{a}_k^T \mathbf{I}_i + b_k, \forall i \in \omega_k$$

其中 \mathbf{I}_i 是 3×1 的颜色向量， \mathbf{a}_k 是 3×1 的系数向量。

针对彩色图的引导滤波器如下：

$$\begin{aligned} \mathbf{a}_k &= \left(\sum_k + \epsilon U \right)^{-1} \left(\frac{1}{|\omega|} \sum_{i \in \omega_k} \mathbf{I}_i p_i - \mu_k \bar{p}_k \right) \\ b_k &= \bar{p}_k - \mathbf{a}_k^T \mu_k \\ q_i &= \bar{\mathbf{a}}_i^T \mathbf{I}_i + \bar{b}_i \end{aligned}$$

其中， Σ_k 是 I 在 ω_k 中的 3×3 的协方差矩阵， U 是 3×3 的单位矩阵。

- b) 利用导向滤波器对图像进行细节增强可以避免梯度反转效应，细节增强的算法如下：

输入图像（input signal）为 p ，利用导向滤波器对它进行边缘保持的平滑处理得到基础层（base layer） q ，细节层（detail layer） $d = p - q$ 。放大细节层以增强细节 $d_{\text{boosted}} = c * d$ ， c 是放大的系数。输出的增强了细节的图像（enhanced signal）是增强细节层和基础层的结合 $p_{\text{enhanced}} = q + d_{\text{boosted}}$ 。

```
d = p - q; % detail layer
d_boosted = 5*d; % boosted detail layer
p_enhanced = q + d_boosted; % enhanced signal
```

导向滤波器能更好地保持图像中的梯度信息是因为基础层在边缘附近的梯度 $\nabla q \approx \bar{a}\nabla I$ ，边缘的形状在重新结合的细节加强层中得到了很好的保持。更细节的证明可以参考“pami12guidedfilter”的 3.4 节。

PROCESSING RESULTS:

对每张处理的图片，先贴一张原图，接下来十二张图的参数设置如下：

$r = 2, \epsilon = 0.1^2$	$r = 2, \epsilon = 0.2^2$	$r = 2, \epsilon = 0.4^2$
$r = 4, \epsilon = 0.1^2$	$r = 4, \epsilon = 0.2^2$	$r = 4, \epsilon = 0.4^2$
$r = 8, \epsilon = 0.1^2$	$r = 8, \epsilon = 0.2^2$	$r = 8, \epsilon = 0.4^2$
$r = 16, \epsilon = 0.1^2$	$r = 16, \epsilon = 0.2^2$	$r = 16, \epsilon = 0.4^2$

放大细节的系数 $c = 5$

注：所有图片以一定比例（注明在图片名后面的括号内）缩小后粘贴在下面。

bird(50%)



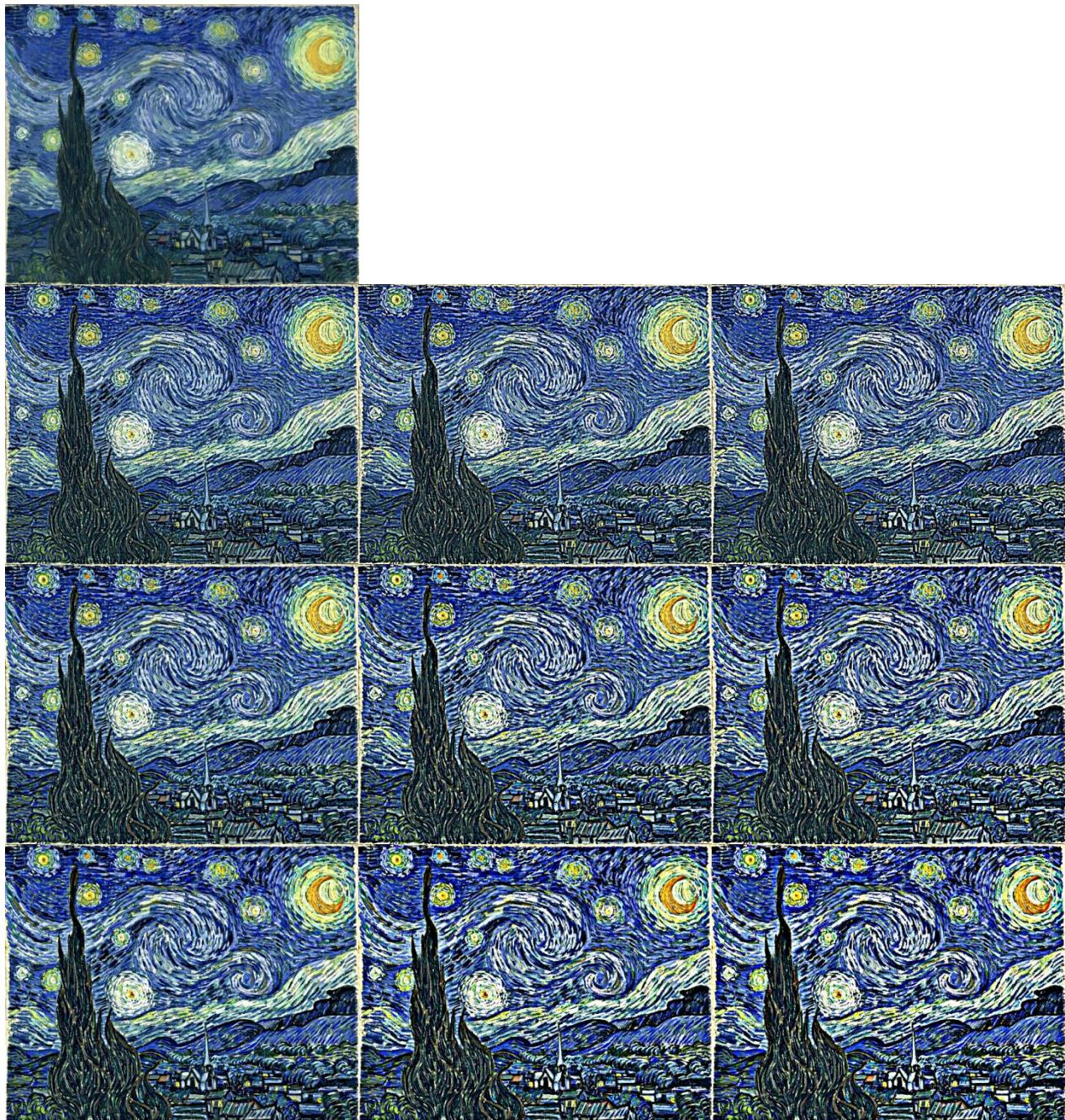


monarch(40%)





starynight(35%)



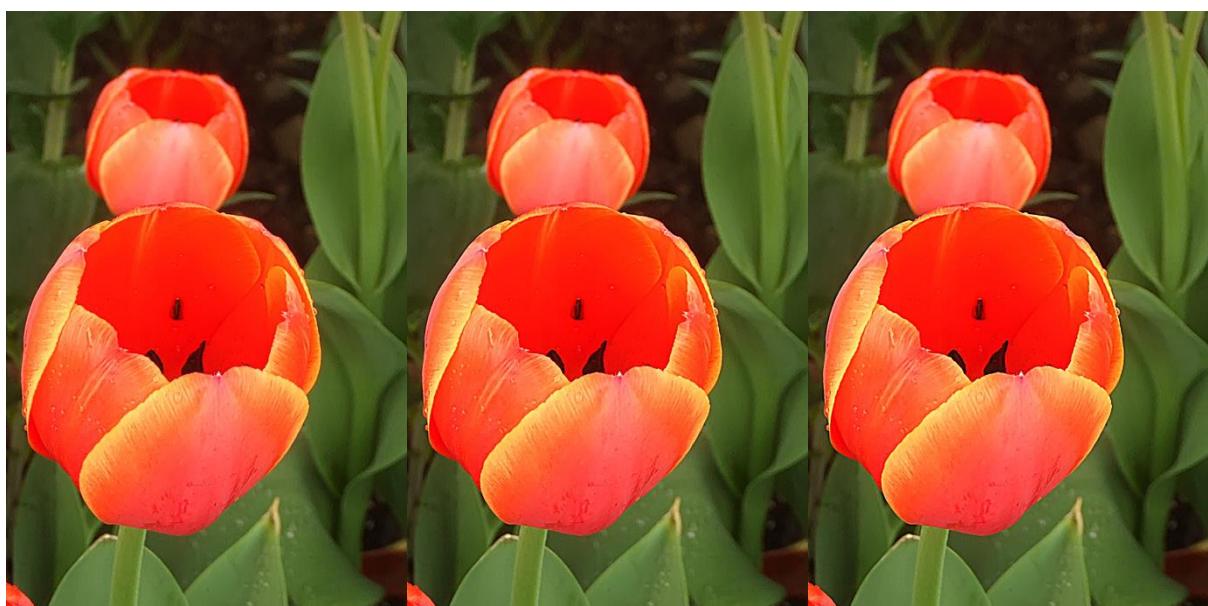
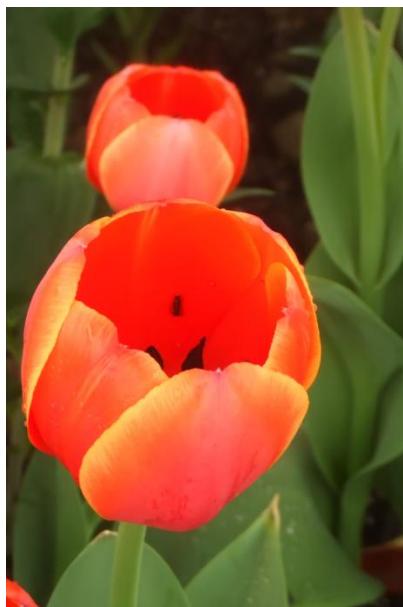


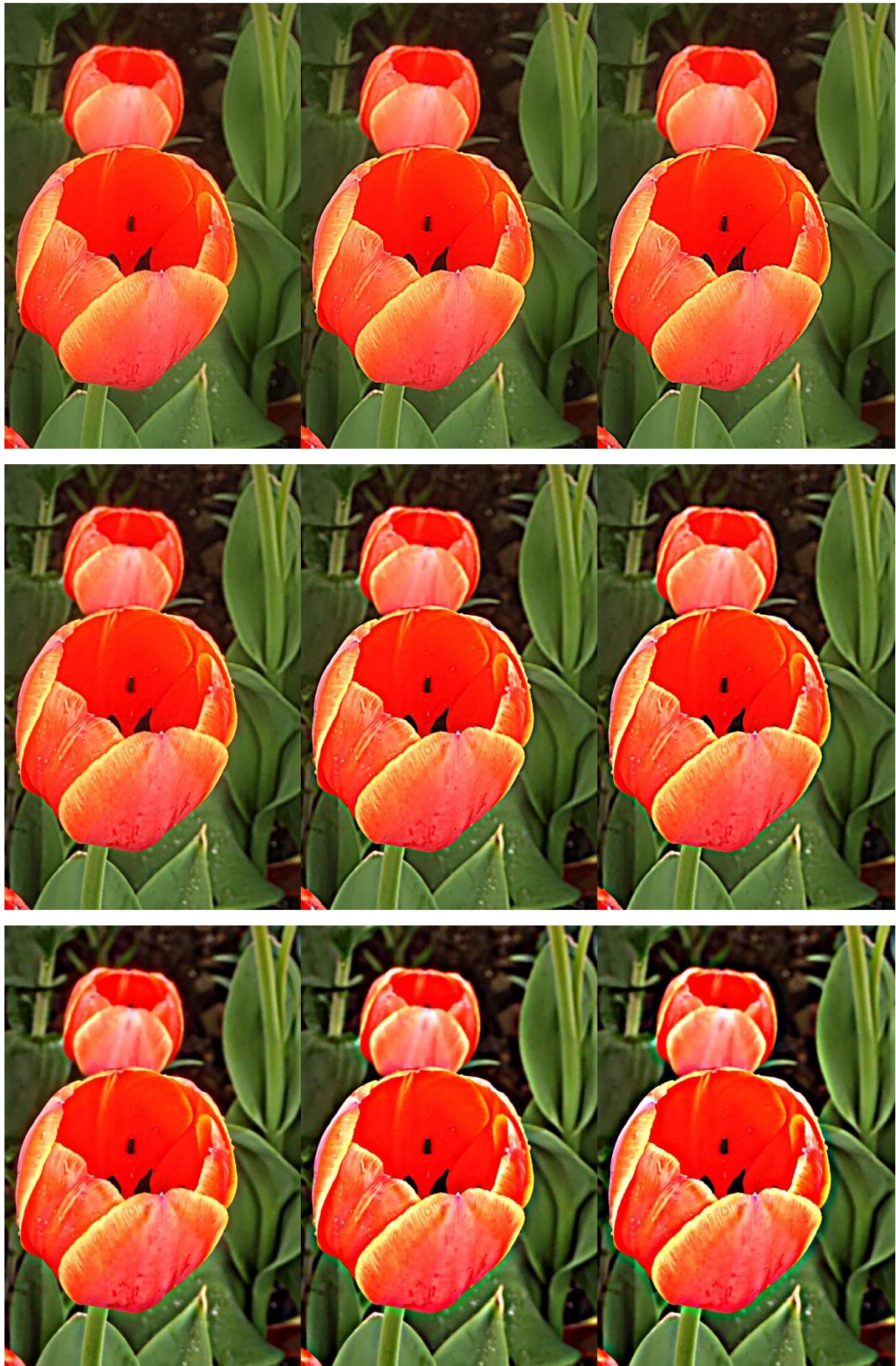
tomato(50%)





tulips(25%)



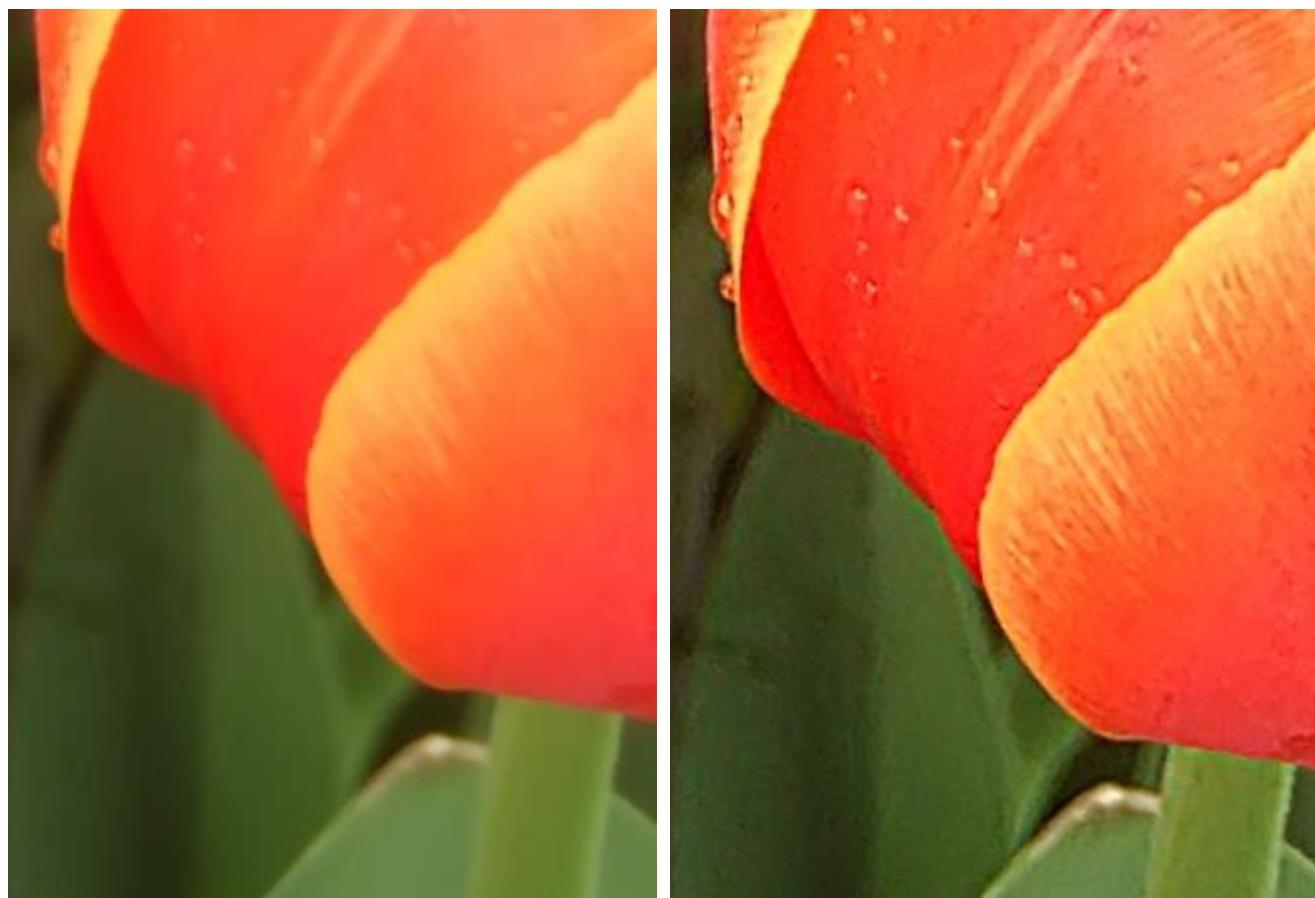


ANALYSES:

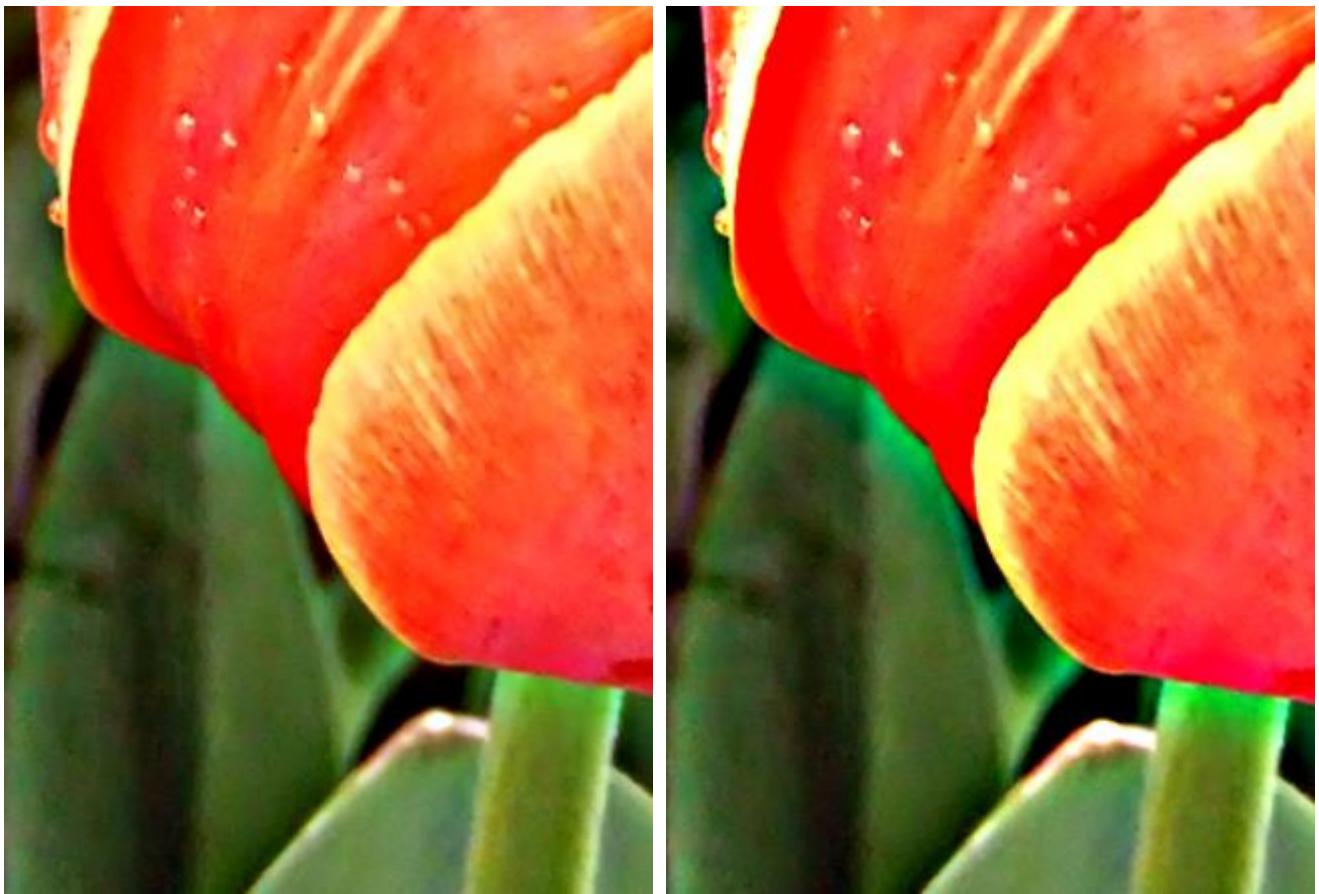
r 相同时， ϵ 越大，图像越清晰； ϵ 相同时， r 越大，图像越清晰。对于上述参数的设置来说， ϵ 的改变对细节增强的效果的影响比 r 更明显。

这个结果和平滑的分析是一致的，因为细节增强就是建立在平滑的基础上，第一步进行平滑时效果越强，则相减后得到的细节层效果越强。

再进一步观察图像的效果：上文所说的“清晰”表现为色彩明亮鲜艳，对比度、饱和度高，展现的细节更多，图像中的物品立体感更强、层次更分明。（下图分别是原图和参数为 $r = 2, \epsilon = 0.1^2, c = 5$ 的细节增强图）



但同时，边缘处出现“光环”、晕轮，在测试图像 *tulips* 中尤为明显，在我测试的参数中， $r = 2$ 时光晕不明显， $r > 2$ 时在边缘部分有非常明显的光晕，随着 r 和 ϵ 的增大而显著增大。（下图分别是参数为 $r = 16, \epsilon = 0.1^2, c = 5$ 和 $r = 16, \epsilon = 0.4^2, c = 5$ 的细节增强图）



3. Explore the disadvantages of the paper by experiments, and try to handle those issues.

从上面的实验结果来看，光晕（Halos）即是这种方法的一个较大弊端和限制。“光晕”指不希望但却被平滑的边缘上的看起来不自然的效果（"Halos" refer to the artifacts of unwanted smoothing of edges.），相反，“梯度反转”指不希望但却被锐化的边缘上的看起来不自然的效果（"gradient reversal" refers to the artifacts of unwanted sharpening of edges.）导向滤波器相比于双边滤波器的一个很大的优势是避免了边缘处的梯度反转，但是却存在“光晕”的问题。事实上，很多学者并不区分上面两种，而统称为"halos"。

可以证明，当局部滤波器被用来平滑一些边缘时 halos 是不可避免的。例如，当较强的纹理被平滑时，较弱的边缘也会被平滑。像导向滤波器、双边滤波器等局部滤波器会因为对这些边缘局部地进行平滑而引入 halos。基于全局优化的滤波器能从全局角度弱化这种模糊的效果，减少 halos，但会导致全局灰度的偏移。下图是作者做出的比较图：

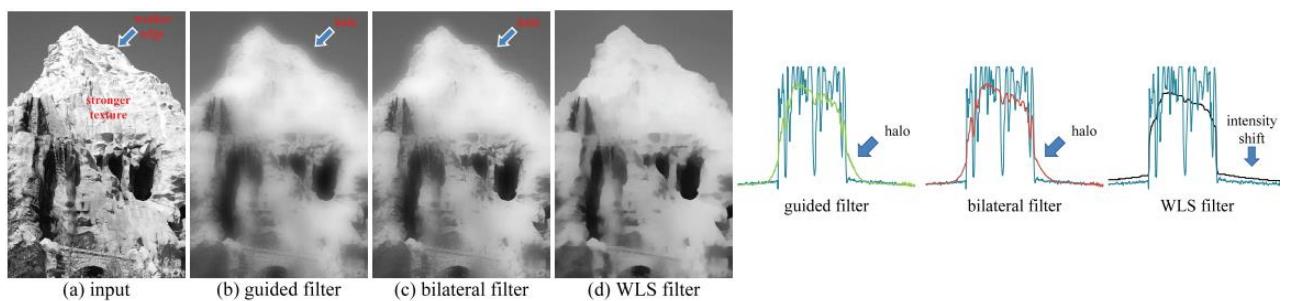


Fig. 18. The halo artifacts. The parameters are $r = 16$, $\epsilon = 0.4^2$ for the guided filter, $\sigma_s = 16$, $\sigma_r = 0.4$ for the bilateral filter, and $\alpha = 1.2$, $\lambda = 5$ for the WLS filter. On the right we show the input signals and filtering results on a scanline.

确实在边缘保持上有些滤波器（如 WLS filter）做得更好，基于优化的方法也通常能产生更高质量的结果，但是它们也更耗时且会有其他问题。

前面有提到，对彩色/多通道图像进行导向滤波有两种方法。第一种是直接分别对每个通道运用导向滤波。第二种是重写局部线性模型得到彩色图像的导向滤波器。第二种方法不再孤立地处理颜色通道，而是进行了结合，

因此能更好地利用彩色通道之间的关联，保持在灰度图/单个通道中难以辨别的边缘和细节。

据此可以推断第二种方法可以有效减少 halos。事实也是如此，下图是作者做出的对比图：

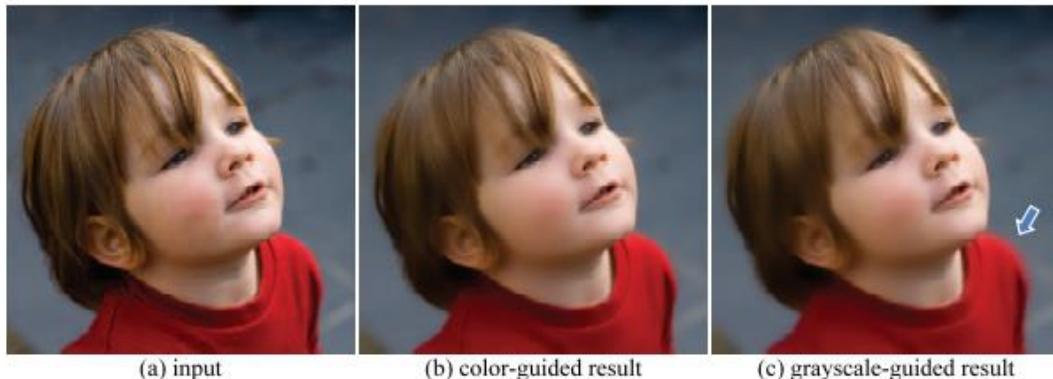


Fig. 8. Guided filtering results guided by the color image (b) and guided by its gray-scale version (c). The result in (c) has halos because the edge is not undistinguishable in gray-scale.

因此彩色图像建议用彩色引导滤波器，可以有效减少 halos，当然，处理时间增加了。

基于全局优化的滤波器也可以减少 halos，同样，要以时间复杂度为代价。

2.2 Advanced Task

1. Implement the faster guided filter.

1) Introduction of the approach described in the paper

首先回顾一下：

导向滤波器是一个在导向图 I 和滤波输出 q 之间的局部线性模型。假设 q 是 I 经过以像素 k 为中心的窗口 ω_k 的线性变换得到的：

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

(a_k, b_k) 是假定在 ω_k 中恒定的一些线性系数。使用半径范围为 r 的正方形窗口。

为了最小化 q 和 p 之间的重建误差（reconstruction error），算出：

$$a_k = \frac{\frac{1}{|\omega|} \sum_{i \in \omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon}$$

$$b_k = \bar{p}_k - a_k \mu_k$$

μ_k 和 σ_k^2 分别是 I 在 ω_k 中的均值和方差， ϵ 是正则化参数，为了防止 a_k 过大而设置。 $|\omega|$ 是 ω_k 中的像素数，

$$\bar{p}_k = \frac{1}{|\omega|} \sum_{i \in \omega_k} p_i$$

最后滤波的输出表示成：

$$q_i = \bar{a}_i I_i + \bar{b}_i$$

其中， $\bar{a}_i = \frac{1}{|\omega|} \sum_{k \in \omega_i} a_k$, $\bar{b}_i = \frac{1}{|\omega|} \sum_{k \in \omega_i} b_k$ 是所有覆盖 i 的窗口的平均系数。

算法如下：

Algorithm 1 Guided Filter.

- 1: $\text{mean}_I = f_{\text{mean}}(I, r)$
 $\text{mean}_p = f_{\text{mean}}(p, r)$
 $\text{corr}_I = f_{\text{mean}}(I \cdot I, r)$
 $\text{corr}_{Ip} = f_{\text{mean}}(I \cdot p, r)$
- 2: $\text{var}_I = \text{corr}_I - \text{mean}_I \cdot \text{mean}_I$
 $\text{cov}_{Ip} = \text{corr}_{Ip} - \text{mean}_I \cdot \text{mean}_p$
- 3: $a = \text{cov}_{Ip} ./ (\text{var}_I + \epsilon)$
 $b = \text{mean}_p - a \cdot \text{mean}_I$
- 4: $\text{mean}_a = f_{\text{mean}}(a, r)$
 $\text{mean}_b = f_{\text{mean}}(b, r)$
- 5: $q = \text{mean}_a \cdot I + \text{mean}_b$

上图中等式四的 \bar{a} 和 \bar{b} 是两个平滑了的图，而 q 中的边缘和结构主要由调整引导图像 I 得到，但是主要的计算开销却在计算 \bar{a} 和 \bar{b} 上，实际上它们的计算不需要在全分辨率下计算。据此思想可以得到一个进行了下采样版本的快速引导滤波算法：

Algorithm 2 Fast Guided Filter.

- 1: $\underline{I'} = f_{\text{subsample}}(I, s)$
 $\underline{p'} = f_{\text{subsample}}(p, s)$
 $\underline{r'} = r/s$
- 2: $\text{mean}_I = f_{\text{mean}}(I', r')$
 $\text{mean}_p = f_{\text{mean}}(p', r')$
 $\text{corr}_I = f_{\text{mean}}(I' \cdot I', r')$
 $\text{corr}_{Ip} = f_{\text{mean}}(I' \cdot p', r')$
- 3: $\text{var}_I = \text{corr}_I - \text{mean}_I \cdot \text{mean}_I$
 $\text{cov}_{Ip} = \text{corr}_{Ip} - \text{mean}_I \cdot \text{mean}_p$
- 4: $a = \text{cov}_{Ip} ./ (\text{var}_I + \epsilon)$
 $b = \text{mean}_p - a \cdot \text{mean}_I$
- 5: $\text{mean}_a = f_{\text{mean}}(a, r')$
 $\text{mean}_b = f_{\text{mean}}(b, r')$
- 6: $\underline{\text{mean}_a} = f_{\text{upsample}}(\text{mean}_a, s)$
 $\underline{\text{mean}_b} = f_{\text{upsample}}(\text{mean}_b, s)$
- 7: $q = \text{mean}_a \cdot I + \text{mean}_b$

首先，以比率 s 下采样（最近邻或双线性插值）输入图像 p 和引导图像 I ，则所有的 box filter 都在低分辨率下进行。之后将 \bar{a} 和 \bar{b} 上采样（双线性插值）到原来的大小。最后输出的 q 仍然通过 $q = \bar{a}I + \bar{b}$ （ I 是全分辨率的引导图）计算。

容易看出，所有 box filter 的计算复杂度从 $O(N)$ 降到了 $O(N/s^2)$ 。虽然最后一步的上采样和输出的步骤是 $O(N)$ 的复杂度，但是由于只占了整体计算量的一小部分而可以忽略它们的影响。

2) My implementation details

根据算法二实现 `function q = fast_guided_filter(I, p, r, eps, s)`, s 即是缩放比例。除了用 `imresize` (我测试了作业 1 中自己写的 `scale_linearInterpolation` 来进行缩放, 但是效率非常不尽如人意, 快速导向滤波的结果反而比导向滤波慢很多, 为了不影响实验应有的效果, 采用 MATLAB 内置的 `imresize` 方法进行测试和比较) 进行 `subsample` 和 `upsample`, 其他代码和图中的伪代码几乎完全一样。调用的函数也是导向滤波器中已实现的, 不再赘述。

至于上/下采样方法的选取我实际比较了一下:

下面五个图对应的是下面表格中的操作

原图	$r = 4, \epsilon = 0.2^2$ 进行导向滤波	$r = 4, \epsilon = 0.2^2, s = 4$ <code>subsample = 'nearest'</code> <code>upsample = 'bilinear'</code> 进行快速导向滤波
$r = 4, \epsilon = 0.2^2, s = 4$ <code>subsample = 'bilinear'</code> <code>upsample = 'bilinear'</code> 进行快速导向滤波	$r = 4, \epsilon = 0.2^2, s = 4$ <code>subsample = 'bilinear'</code> <code>upsample = 'nearest'</code> 进行快速导向滤波	$r = 4, \epsilon = 0.2^2, s = 4$ <code>subsample = 'nearest'</code> <code>upsample = 'nearest'</code> 进行快速导向滤波



很明显, `subsample = 'nearest'`, `upsample = 'bilinear'` 的设置使得快速导向滤波和导向滤波的结果差异最小, 肉眼看不出来差别。因此采用这个设置。

2. Compare the running time of the faster guided filter to the original one on both smoothing and enhancement tasks respectively, also paste your processing results in your report.

时间比较:

smoothing:

对五张灰度图在参数为 $r = [4 8 16]$, $\epsilon = [0.1^2 0.2^2 0.4^2]$ 下进行导向滤波所用时间为 16.864427 秒, 同样条件下 (另设 $s = 4$) 进行快速导向滤波所用时间 1.163830 秒。加速了约 14.5 倍。

detail enhancement:

对五张灰度图在参数为 $r = [4 \ 8 \ 16]$, $\epsilon = [0.1^2 \ 0.2^2 \ 0.4^2]$ 下进行导向滤波所用时间为 8.811079 秒, 同样条件下 (另设 $s = 4$) 进行快速导向滤波所用时间 127.631152 秒。同样加速了约 **14.5** 倍。

结果:

下面仅各粘贴一组结果说明

每九张图的参数设置如下:

$r = 4, \epsilon = 0.1^2$	$r = 4, \epsilon = 0.2^2$	$r = 4, \epsilon = 0.4^2$
$r = 8, \epsilon = 0.1^2$	$r = 8, \epsilon = 0.2^2$	$r = 8, \epsilon = 0.4^2$
$r = 16, \epsilon = 0.1^2$	$r = 16, \epsilon = 0.2^2$	$r = 16, \epsilon = 0.4^2$

smoothing:



detail enhancement:



和之前的结果对比，肉眼看不出来差别，说明这种算法非常高效。

References

1. Guided Image Filtering (<http://kaiminghe.com/eccv10/index.html>)
2. Computer Vision – The Integral Image
(<https://computersciencesource.wordpress.com/2010/09/03/computer-vision-the-integral-image/>)
3. integral image
(<https://www.quora.com/How-integral-image-is-used-in-image-processing-and-how-improves-the-computation-time>)
4. Summed area table (https://en.wikipedia.org/wiki/Summed_area_table)