Computer Vision hw#1
Advanced Color-to-Gray Conversion

B05902066 蔡翔陞

1. Joint bilateral filter

I implement the joint bilateral filter by following the formula as below.

$$F^T(I) = \frac{\sum_{q\in\Omega_p} G_s(p,q)G_r(T_p,T_q)I_q}{\sum_{q\in\Omega_p} G_s(p,q)G_r(T_p,T_q)}$$

, which contains spatial kernel and range kernel.

For spatial kernel, as all the computation sharing same weights, I precompute the spatial kernel with given sigma_r to speed up the process.

For range kernel, as the computation being based on the pixel value and corresponding neighbor pixels, we need to compute the weights locally. To speed up the computation, utilizing the python build-in functions and numpy package would highly improve the performance.

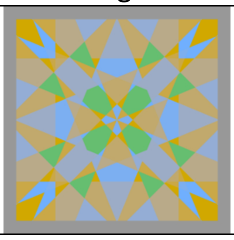
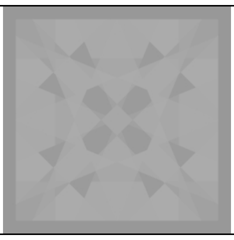
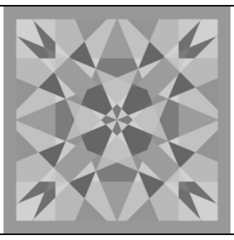
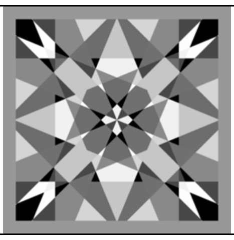
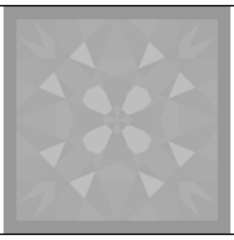
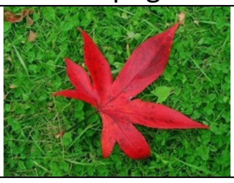
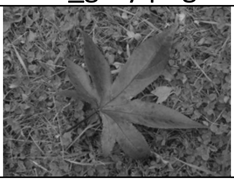
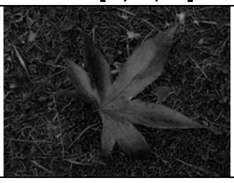
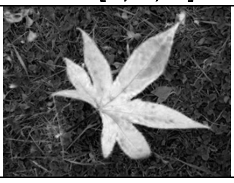
Actually, I would only use one loop to go through all the pixels on the input image. For range kernel, I use the numpy's broadcast computation find the weight in parallel. For combining spatial kernel and range kernel and applying on the input image, I use the matrix multiplication and also the broadcast computation to find the corresponding output value.

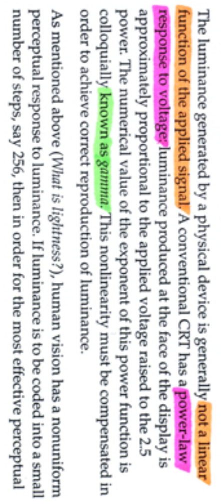
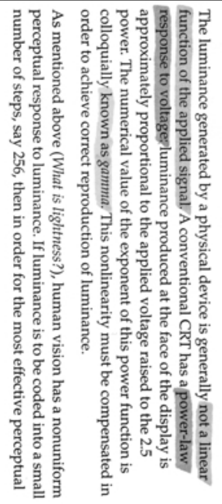
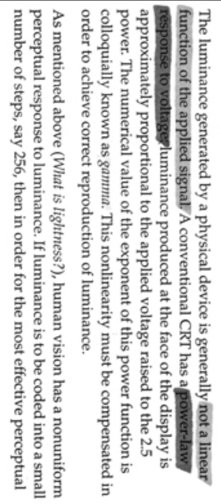
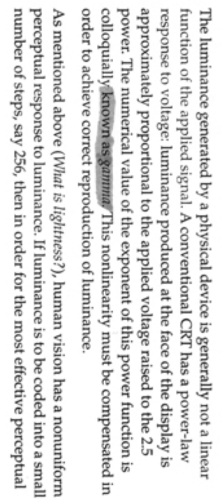
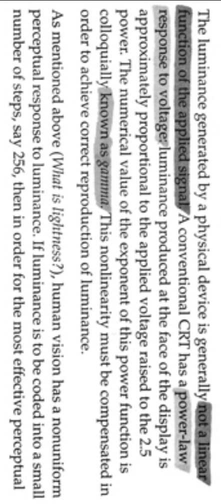
With these tricks, I could complete the computation less than 20 seconds per image.

2. Local minimal

The method to find the local minimal is intuitive. For any combination of weight $\mathrm{w} = [\mathrm{w}_b, \mathrm{w}_\mathrm{g}, \mathrm{w}_\mathrm{r}]$, the neighbors would be w plusing each of $[[0,-0.1,0.1],[0,0.1,-0.1],[0.1,0,-0.1],[0.1,-0.1,0],[-0.1,0.1,0],[-0.1,0,0.1]]$. We could simply check the error value of a given weight and the corresponding neighbor to find the local minimal.

3. result (the weight $\mathrm{w} = [\mathrm{w}_b, \mathrm{w}_\mathrm{g}, \mathrm{w}_\mathrm{r}]$)

tag	Origin	BGR2GRAY	#1	#2	#3
a					
	0a.png	0a_gray.png	W = [0, 0, 1]	W = [1, 0, 0]	W = [0, 1, 0]
b					x
	0b.png	0b_gray.png	W = [1, 0, 0]	W = [0, 0, 1]	x

c	The luminance generated by a physical device is generally not a linear function of the applied signal. A conventional CRT has a power-law response to voltage: luminance produced at the face of the display is approximately proportional to the applied voltage raised to the 2.5 power. The numerical value of the exponent of this power function is colloquially known as *gamma*. This nonlinearity must be compensated in order to achieve correct reproduction of luminance. As mentioned above (*What is lightness?*), human vision has a nonuniform perceptual response to luminance. If luminance is to be coded into a small number of steps, say 256, then in order for the most effective perceptual	The luminance generated by a physical device is generally not a linear function of the applied signal. A conventional CRT has a power-law response to voltage: luminance produced at the face of the display is approximately proportional to the applied voltage raised to the 2.5 power. The numerical value of the exponent of this power function is colloquially known as *gamma*. This nonlinearity must be compensated in order to achieve correct reproduction of luminance. As mentioned above (*What is lightness?*), human vision has a nonuniform perceptual response to luminance. If luminance is to be coded into a small number of steps, say 256, then in order for the most effective perceptual	The luminance generated by a physical device is generally not a linear function of the applied signal. A conventional CRT has a power-law response to voltage: luminance produced at the face of the display is approximately proportional to the applied voltage raised to the 2.5 power. The numerical value of the exponent of this power function is colloquially known as *gamma*. This nonlinearity must be compensated in order to achieve correct reproduction of luminance. As mentioned above (*What is lightness?*), human vision has a nonuniform perceptual response to luminance. If luminance is to be coded into a small number of steps, say 256, then in order for the most effective perceptual	The luminance generated by a physical device is generally not a linear function of the applied signal. A conventional CRT has a power-law response to voltage: luminance produced at the face of the display is approximately proportional to the applied voltage raised to the 2.5 power. The numerical value of the exponent of this power function is colloquially known as *gamma*. This nonlinearity must be compensated in order to achieve correct reproduction of luminance. As mentioned above (*What is lightness?*), human vision has a nonuniform perceptual response to luminance. If luminance is to be coded into a small number of steps, say 256, then in order for the most effective perceptual	The luminance generated by a physical device is generally not a linear function of the applied signal. A conventional CRT has a power-law response to voltage: luminance produced at the face of the display is approximately proportional to the applied voltage raised to the 2.5 power. The numerical value of the exponent of this power function is colloquially known as *gamma*. This nonlinearity must be compensated in order to achieve correct reproduction of luminance. As mentioned above (*What is lightness?*), human vision has a nonuniform perceptual response to luminance. If luminance is to be coded into a small number of steps, say 256, then in order for the most effective perceptual
	0c.png	0c_gray.png	W = [0, 1, 0]	W = [0, 0, 1]	W = [0.7, 0.3, 0]

4. Discussion

It is surprise to see that reducing the colored image to the red channel (w = [0, 0, 1]) would create the closest result with the bilateral filter. To have the better visual experience, we often use Y = 0.299 R + 0.587 G + 0.114 B to produce the grayscale image (this weight is also used by OpenCV), which gives green channel more than half proportion; however, the result shows that to segment an image, the red channel would be useful than others.

Maybe this result would also depend on the input image or other factors, but it's still quite interesting.

5. How to run the code?

The implementation of joint bilateral filter would be contained in the *class Joint_bilateral_filter* in joint_bilateral_filter.py

To find the difference between bilateral filter and joint bilateral filter, use hw1.py as *python3 hw1.py [group]*, and the result would be store in cvs format in file error[group].cvs.

To find the local minimal, use find_localminimal.py as *python3 find_localminimal.py [error_file]*, and the voting result would be sorted and print line by line.

To generate the result image, use generate_img.py as *python3 generate_img.py* and set the config and group at top of the code, the result images would store in *output_dir*.