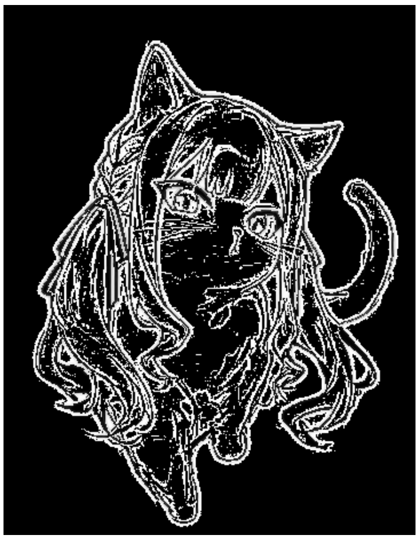
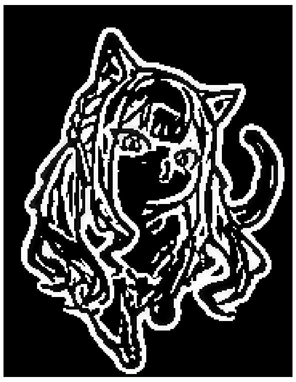
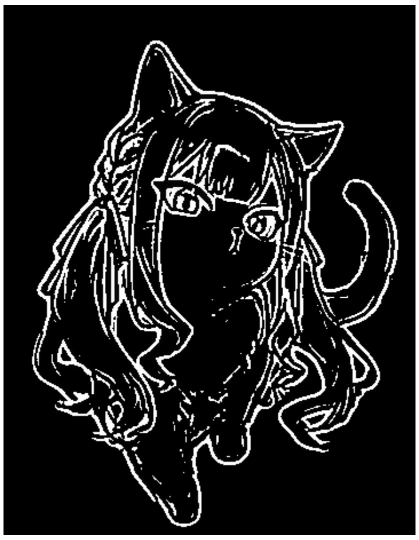
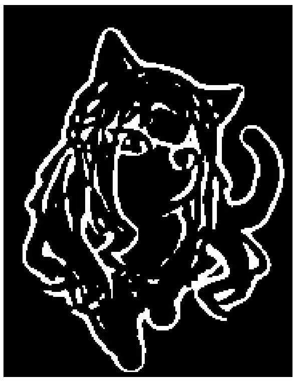
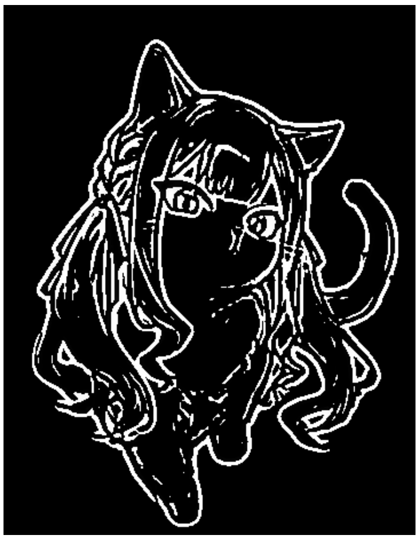
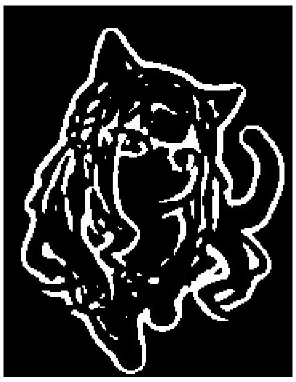
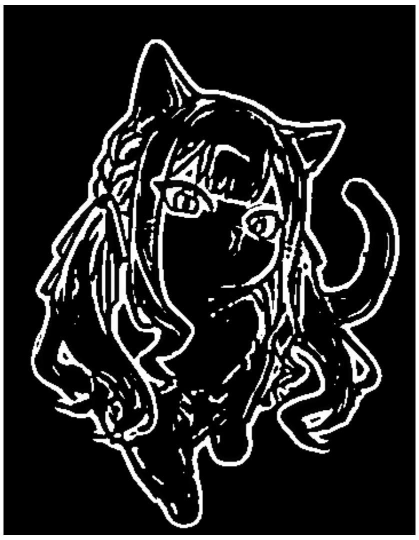
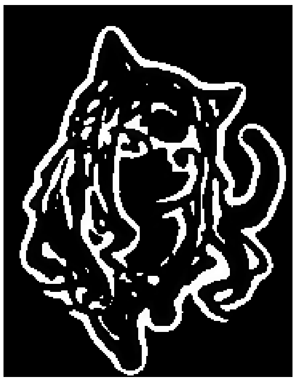
Computer Vision HW1 Report

Student ID: R12K41025
Name: 杜冠廷

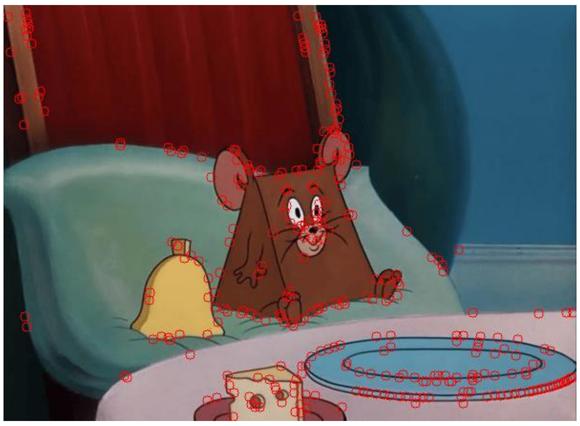
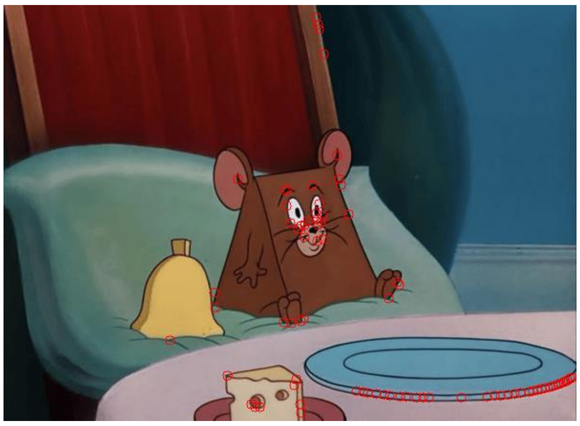
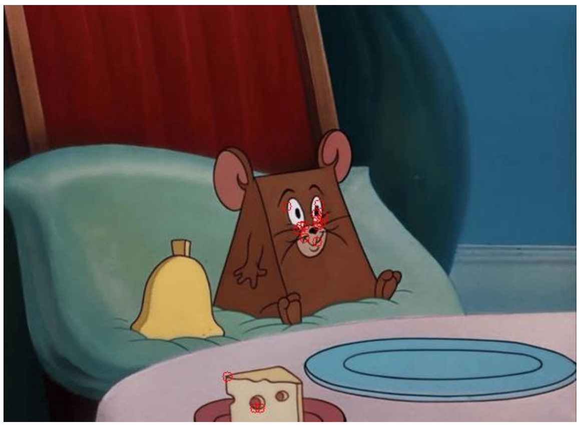
Part 1.

- **Visualize the DoG images of 1.png.**

	DoG Image (threshold = 5)		DoG Image (threshold = 5)
DoG1-1.png		DoG2-1.png	
DoG1-2.png		DoG2-2.png	
DoG1-3.png		DoG2-3.png	

DoG1-4.png		DoG2-4.png	

- **Use three thresholds (1,2,3) on 2.png and describe the difference.**

Threshold	Image with detected keypoints on 2.png
2	
5	
7	

(describe the difference)

We can find that as the threshold is large, the total number of detected keypoints will be less, and the remaining keypoints are greater feature points to describe the original image. There are two types of greater feature points:

(1). Facial Features of the Mouse: Due to the greater pixel differences between the mouse's facial features and its body, we can observe more feature points on the mouse's eyes, nose, and mouth.

(2). Color Differences: The pixel differences are more significant due to variations in color. We can also see more feature points at the boundaries of different colors, such as the plate, bell, cheese, mouse, and chair back.

When we set a higher threshold, other types of feature points almost disappear.

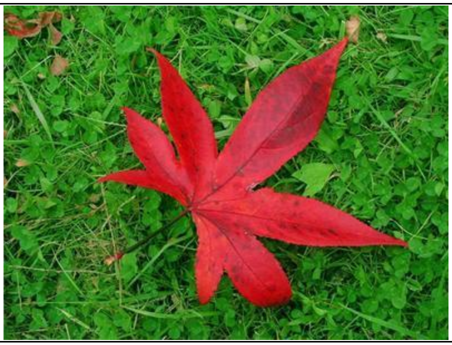
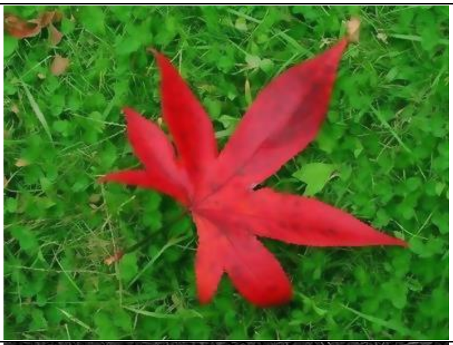
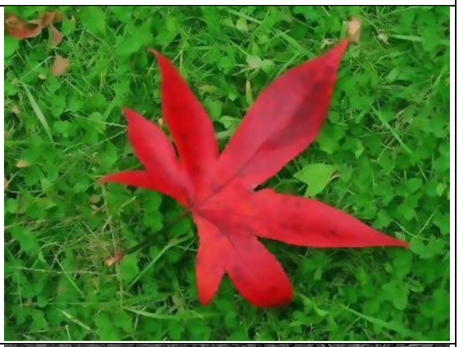
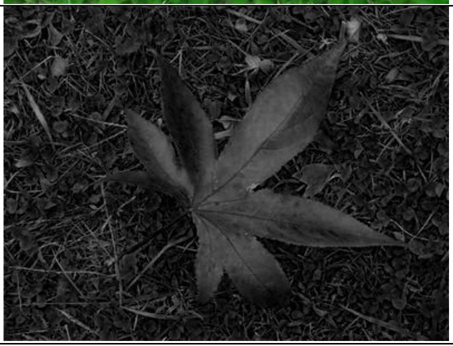
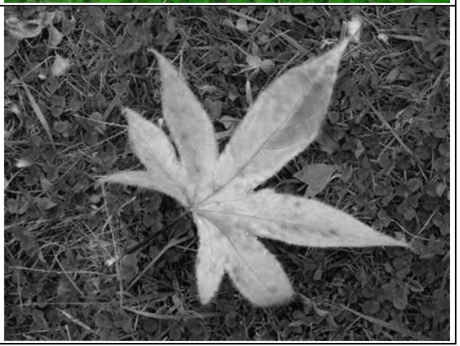
Part 2.

- **Report the cost for each filtered image.**

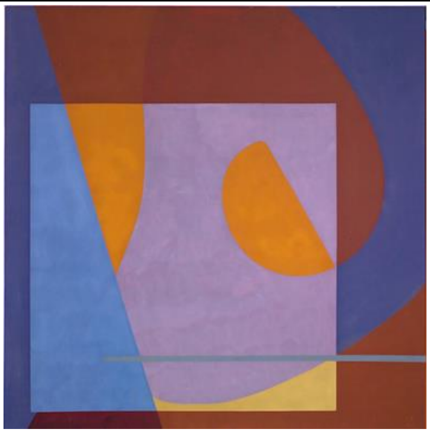
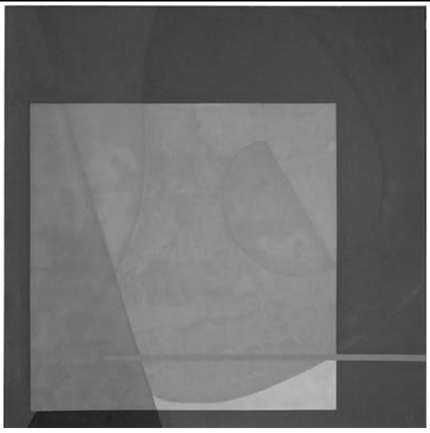
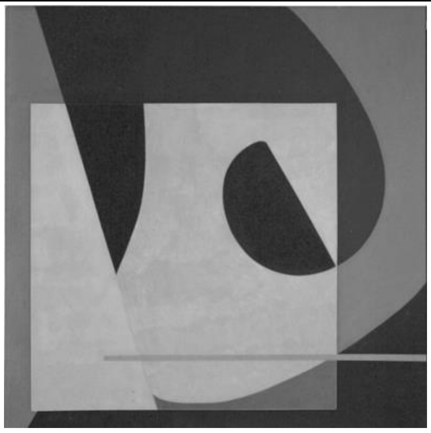
Gray Scale Setting	Cost (1.png)
cv2.COLOR_BGR2GRAY	1207799
R*0.0+G*0.0+B*1.0	1439568
R*0.0+G*1.0+B*0.0	1305961
R*0.1+G*0.0+B*0.9	1393620
R*0.1+G*0.4+B*0.5	1279697
R*0.8+G*0.2+B*0.0	1127913

Gray Scale Setting	Cost (2.png)
cv2.COLOR_BGR2GRAY	183851
R*0.1+G*0.0+B*0.9	77884
R*0.2+G*0.0+B*0.8	86023
R*0.2+G*0.8+B*0.0	188019
R*0.4+G*0.0+B*0.6	128341
R*1.0+G*0.0+B*0.0	110862

- **Show original RGB image / two filtered RGB images and two grayscale images with highest and lowest cost.**

Original RGB image (1.png)	Filtered RGB image and Grayscale image of Highest cost	Filtered RGB image and Grayscale image of Lowest cost
		
		

(Describe the difference between those two grayscale images)
The grayscale image tends to be more sensitive to the channel with larger weights. This means that if we assign a higher weight to the red channel, differences in red will be more noticeable in the grayscale image. Since the original image is primarily composed of red and green, using higher weights for R and G results in a clearer grayscale image. Therefore, the grayscale image with (R, G, B) = (0.8, 0.2, 0) appears clearer than the one with (R, G, B) = (0, 0, 1). As a result, using the former as the guidance for the Joint Bilateral Filter reduces the cost.

Original RGB image (2.png)	Filtered RGB image and Grayscale image of Highest cost	Filtered RGB image and Grayscale image of Lowest cost
		
		

(Describe the difference between those two grayscale images)
We can see that the original image can be divided into brighter and darker regions. The brighter areas have higher intensity (pixel values) and are predominantly occupied by blue, while the green elements are relatively scarce in the original image. As a result, the original image is mainly composed of the blue channel (B > R > G). When we use a larger weight for the B channel, the grayscale image becomes clearer. Consequently, the grayscale image with (R, G, B) = (0.1, 0, 0.9) appears clearer than the one with (R, G, B) = (0.2, 0.8, 0). Therefore, using the former as the guidance for the Joint Bilateral Filter results in lower cost.

- **Describe how to speed up the implementation of bilateral filter.**

  (1). Instead of using a for loop to slide a window across the entire image, we use a for loop to shift the image and iterate over all pixels within the convolution kernel. As a result, the number of iterations in the for loop is reduced from 316×316 to 19×19. The idea is that since the kernel size is 19×19, we can consider the output image as the sum of 19×19 filtered sub-images. In each iteration of the for loop, we extract the relevant region from the padded image and the padded guidance image. These two images are shifted versions of the original image and guidance image (for example, when i, j = (0, 0), all pixels in these two images correspond to the upper-left pixels of the original image). Next, we compute the range kernel values by calculating the difference between the guidance image and the shifted guidance image. Then, we multiply the shifted image by the weights of the spatial kernel and the range kernel. Finally, the result of this multiplication contributes $1/(19 \times 19)$ of the output image value.

  (2). Instead of repeatedly computing the exponential function and sum of squares within the for loop, we can precompute a one-dimensional lookup table. For the spatial kernel, since its value depends only on the offset and the default offset range is (-9, 9), we can build a two-dimensional lookup table of size (19, 19) to store the spatial kernel weights. For the range kernel, since its value depends only on the pixel intensity difference, and the difference range is (0, 256), we can build a one-dimensional lookup table of size 256 to store the range weights. During each iteration in the for loop, we can use the offset in the (x, y) direction and the intensity difference between pixels in (r, g, b) as table indices to retrieve the corresponding weights for the spatial kernel and range kernel, thereby accelerating the computation.