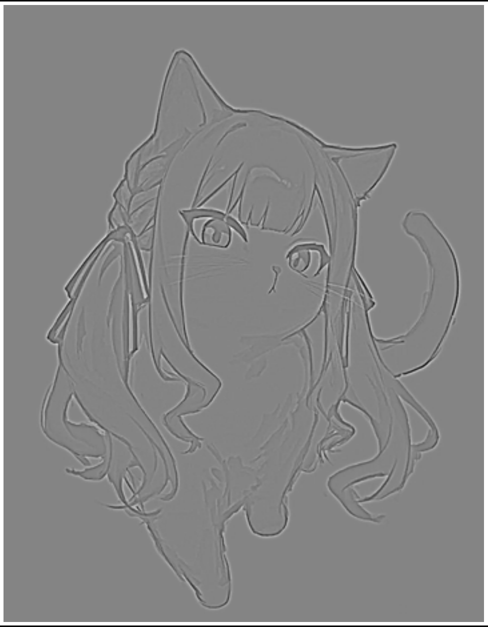
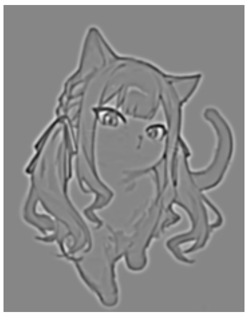
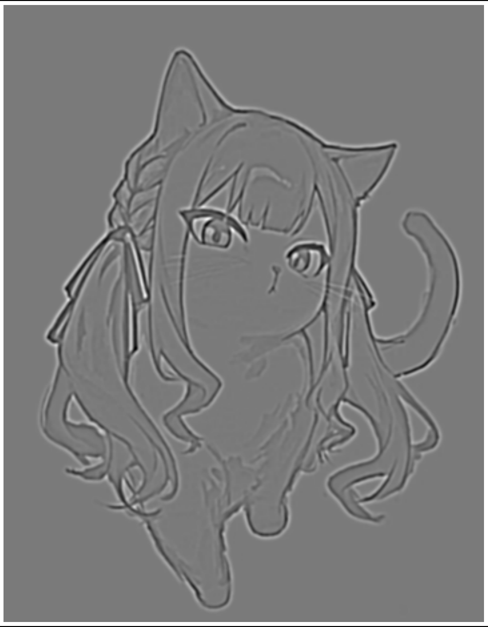
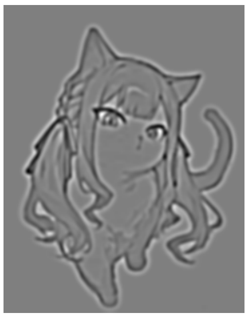
Computer Vision HW1 Report

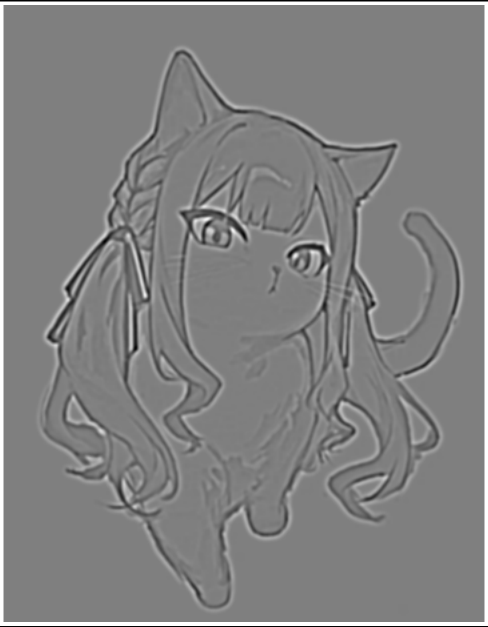
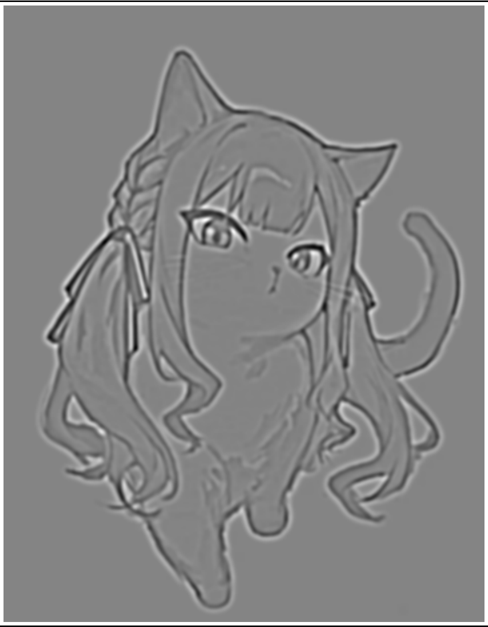
Student ID: R12942159

Name: 邱亮茗

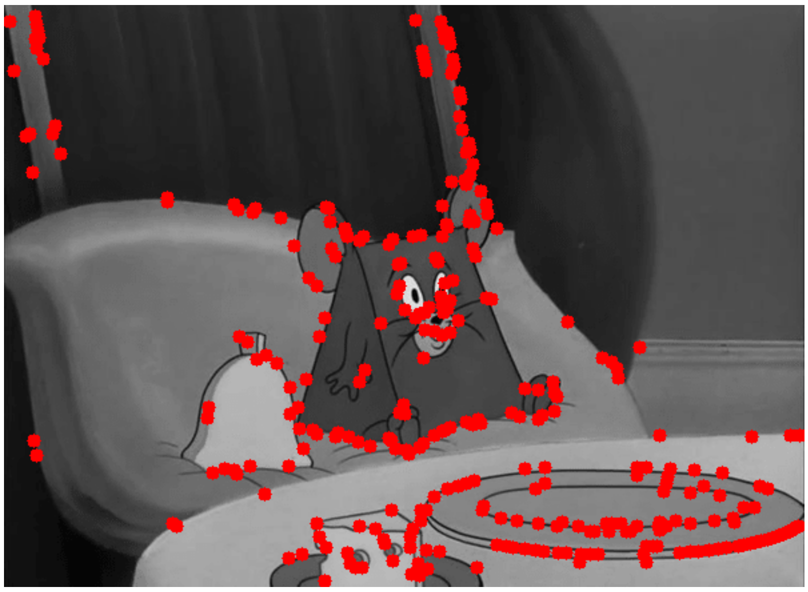
Part 1.

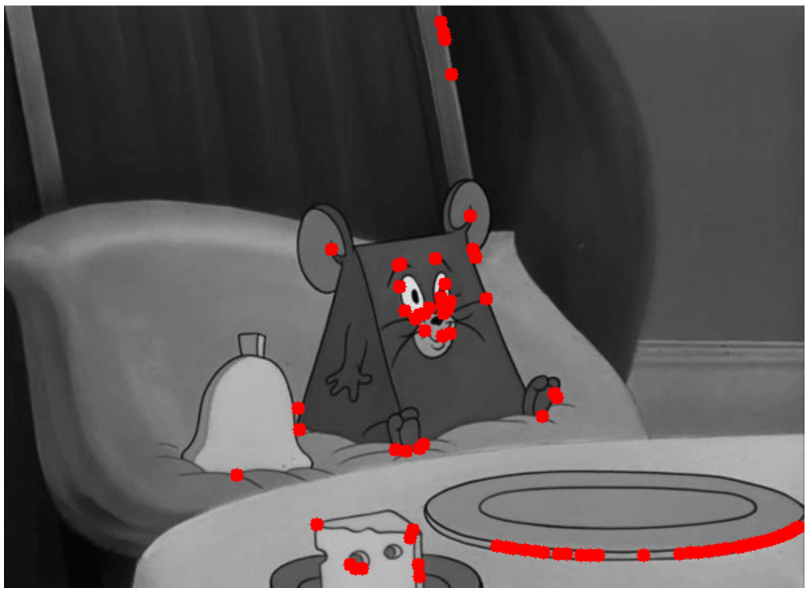
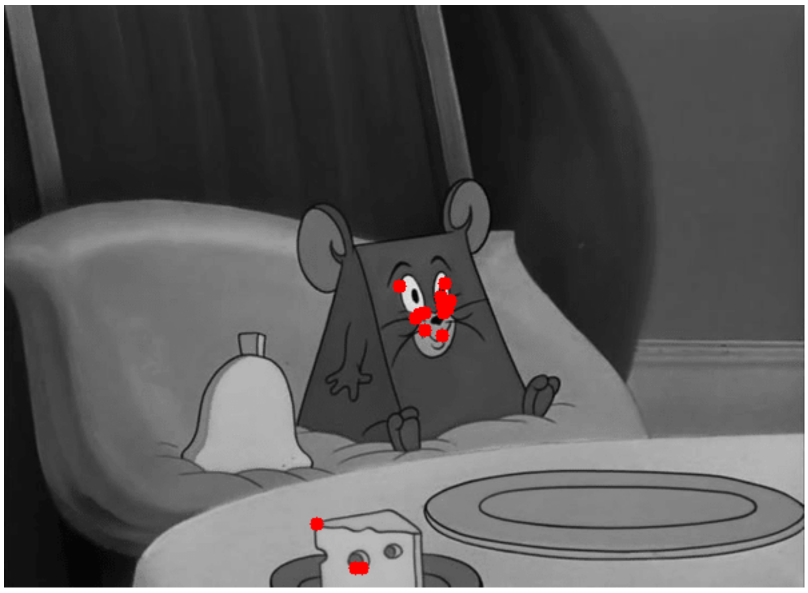
- **Visualize the DoG images of 1.png.**

	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)

These three images demonstrate the effect of different thresholds (2, 5, 7) when applying the **Difference of Gaussian (DoG)** for keypoint detection.

Threshold = 2

- Detects the most keypoints, covering almost the entire scene with red dots.
- A lower threshold allows more keypoints to be detected, including noise and weak features.

Threshold = 5

- The number of detected keypoints is significantly reduced, but important structures like the character’ s face, outlines, and object edges remain.
- A higher threshold filters out weaker keypoints, retaining only stronger features.

Threshold = 7

- Further reduces the number of detected keypoints, leaving only the most critical regions, such as the character's eyes, nose, mouth, and high-contrast areas (e.g., cheese edges).
- A stricter threshold selects only the most prominent keypoints, improving feature quality but potentially missing some details.

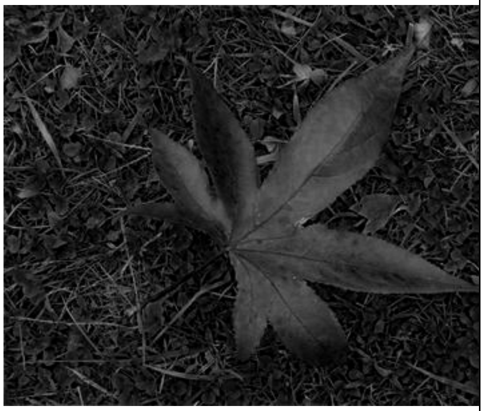
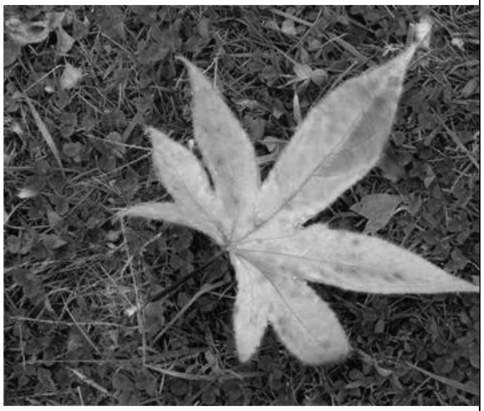
Part 2.

- **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

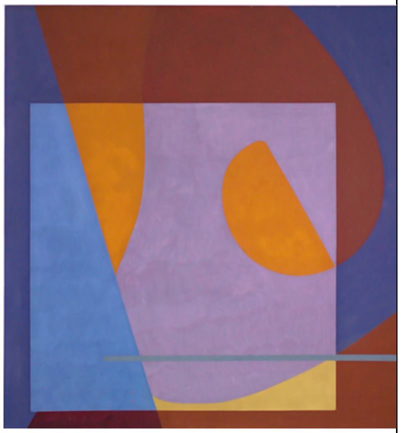
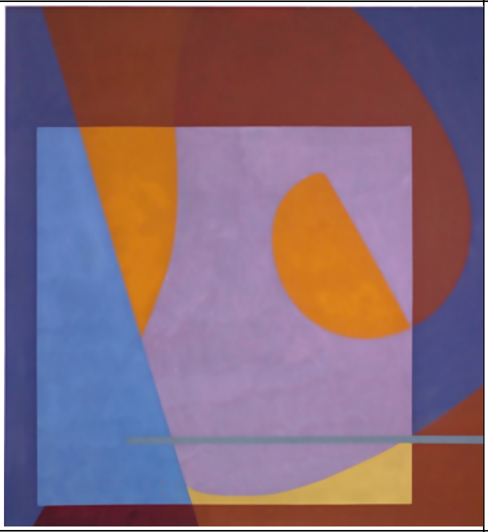
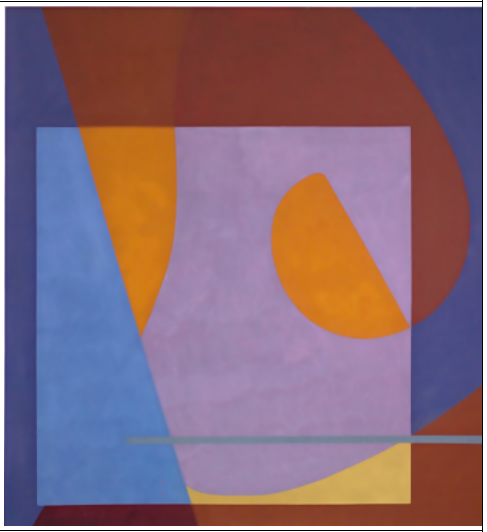
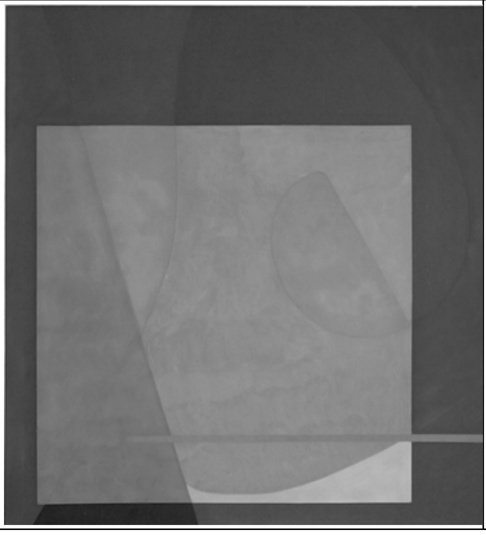
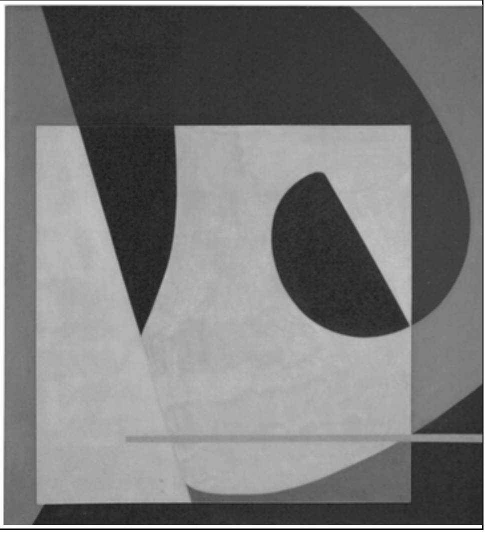
Gray Scale Setting	Cost (2.png)
cv2.COLOR_BGR2GRAY	183850
R*0.1+G*0.0+B*0.9	77882
R*0.2+G*0.0+B*0.8	86023
R*0.2+G*0.8+B*0.0	188019
R*0.4+G*0.0+B*0.6	128341
R*1.0+G*0.0+B*0.0	110862

- **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 differences between the two grayscale images can be attributed to the original image's color composition, which primarily consists of red and green. The grayscale setting with R0.0 + G0.0 + B1.0 results in the highest L1-norm because it relies entirely on the blue channel, making the guidance image significantly different from the input image. Conversely, the setting with R0.8 + G0.2 + B0.0 has the lowest L1-norm, as it emphasizes the red component, which is more aligned with the original image. This distinction affects how joint bilateral filtering is applied, as it generally preserves edges better than standard bilateral filtering. As a result, the grayscale image with the highest L1-norm retains more pronounced edges, whereas the one with the lowest L1-norm appears smoother. However, when these grayscale images are used as guidance for joint bilateral filtering, the final filtered RGB images exhibit minimal visible differences.

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)

In the grayscale image with the lowest L1-norm, the guidance image emphasizes the blue channel (0.1R + 0.9B), preserving more edge details and resulting in higher contrast and clearer features, whereas in the grayscale image with the highest L1-norm, the guidance image emphasizes the green channel (0.2R + 0.8G), which is less present in the original image, leading to a lighter appearance with less defined edges.

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

To speed up the implementation of the bilateral filter while adhering to the given constraints, we optimize the computation by leveraging NumPy vectorization and adaptive filtering based on the guidance image. Instead of processing all pixels in a window, we consider only neighboring pixels with similar intensity values, reducing unnecessary calculations. We eliminate nested for-loops by utilizing sliding window view, which allows efficient batch processing of image patches. Additionally, NumPy broadcasting is used to compute Gaussian weights in parallel, minimizing redundant operations and enhancing efficiency. These optimizations significantly accelerate the filtering process while maintaining accuracy, without relying on Cython, multi-threading, or GPU acceleration.