

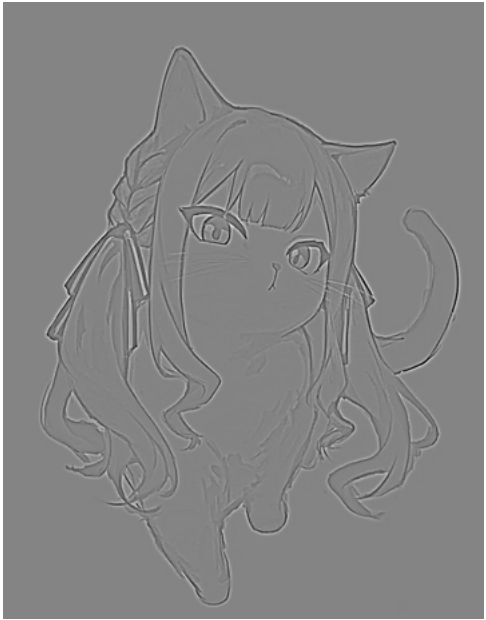



Computer Vision HW1 Report





Student ID: B11705048

Name: 林杰

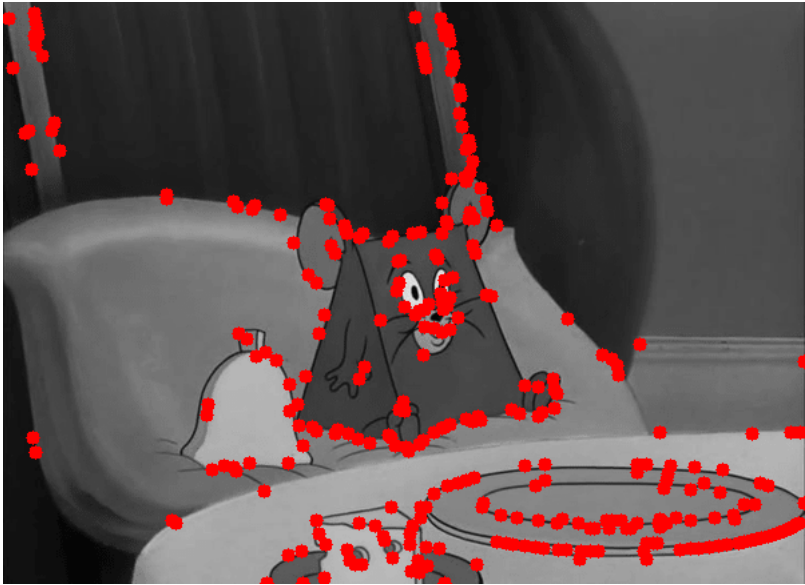
Part 1.

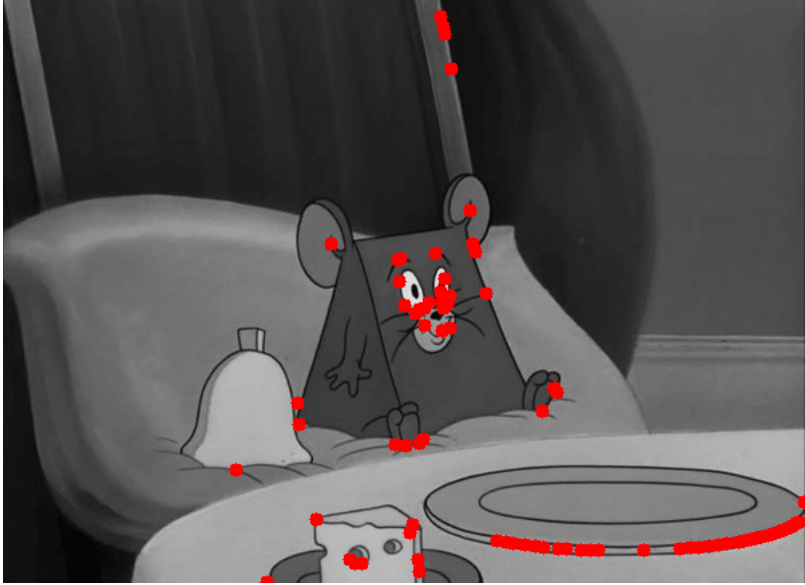
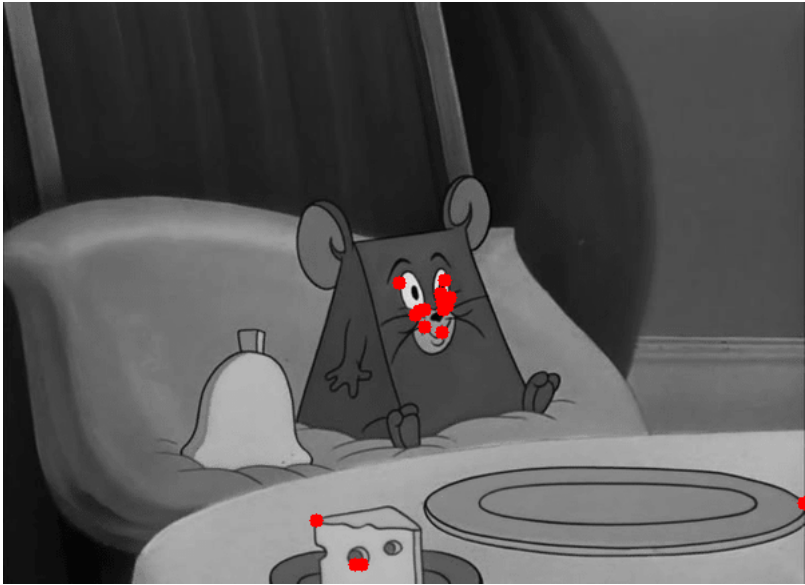
- Visualize the DoG images of 1.png.

	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 (2,5,7) on 2.png and describe the difference.

Threshold	Image with detected keypoints on 2.png
2	

5	
7	

(describe the difference)

When using the Difference of Gaussians approach to detect keypoints in a figure, different thresholds can give different results.

- 1. Number of Keypoints Detected:** A lower threshold is more permissive, meaning even small DoG responses qualify as keypoints. This leads to an increased number of keypoints, but also more false positives. In our experiment, using a threshold of 7 produced the fewest keypoints, while a threshold of 2 produced the most.
- 2. Keypoint Stability:** With a low threshold, the code picks up noise or minor texture variations, so many of those keypoints could vanish if you slightly change lighting or perspective. With a high threshold, the code may miss fine details. However, the keypoints you do detect are typically “stronger” features (like sharp corners).




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
$R*0.2+G*0.0+B*0.8$	86024
$R*0.2+G*0.8+B*0.0$	188019
$R*0.4+G*0.0+B*0.6$	128341
$R*1.0+G*0.0+B*0.0$	110862

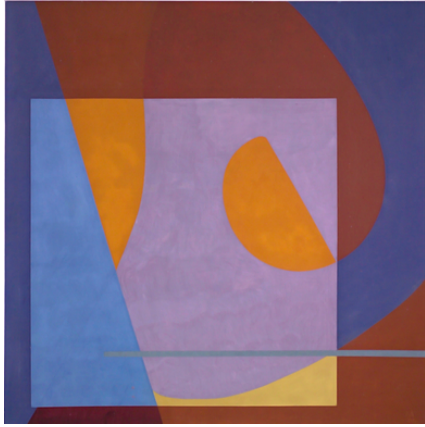
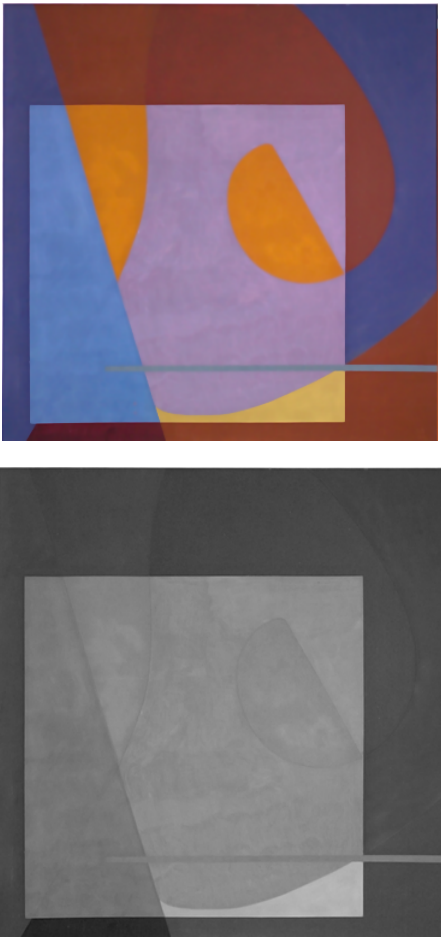
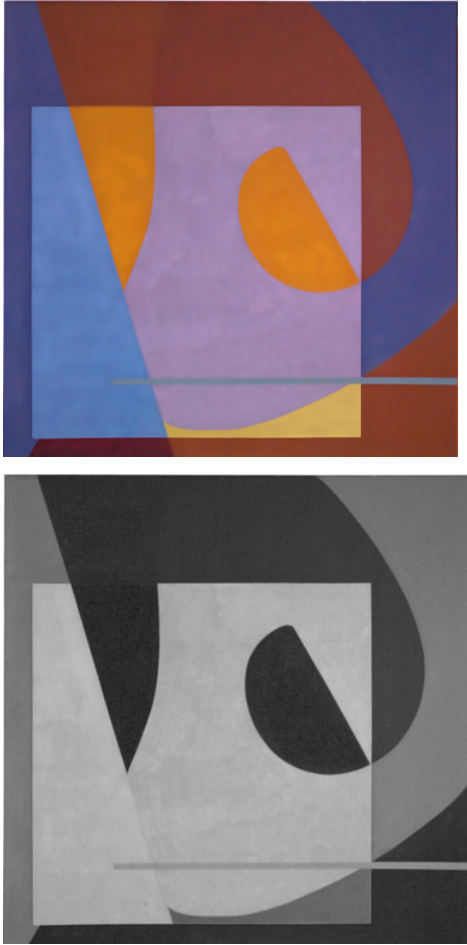
- Show original RGB image / two filtered RGB images and two grayscale images with highest and lowest cost.

Original RGB image (1.png)	Filtered <u>RGB image</u> and <u>Grayscale image</u> of Highest cost	Filtered <u>RGB image</u> and <u>Grayscale image</u> of Lowest cost
		

(Describe the difference between those two grayscale images)

From the images, we can see that the left grayscale image is much darker overall (especially the leaf), while the right grayscale image is lighter and shows more contrast between leaf and grass. This suggests that the left image's grayscale conversion weights darker channels (or the red channel in a way that turns the leaf nearly black), whereas the right image's conversion gives the leaf higher intensity.

Furthermore, since the right grayscale image has higher contrast, it provides stronger intensity gradients that serve as better guidance for the Joint Bilateral Filter (JBF). That means the filter can more clearly distinguish the leaf from the background, preserving edges and detail more effectively than the darker grayscale version.

Original RGB image (2.png)	Filtered <u>RGB image</u> and <u>Grayscale</u> <u>image</u> of Highest cost	Filtered <u>RGB image</u> and <u>Grayscale</u> <u>image</u> of Lowest cost
		

(Describe the difference between those two grayscale images)

Similar with the result from 2.png, the right grayscale image (with the lowest cost) has higher contrast, making both the **edges** and **color** differences more distinct. This provides stronger intensity gradients, which serve as better guidance for the Joint Bilateral Filter (JBF), allowing it to preserve edges and structures more effectively.

Describe how to speed up the implementation of bilateral filter.

1. Precomputed Look-Up Tables (LUTs) for Faster Weight Computation

The code speeds up the spatial and range weight calculations by using Look-Up Tables instead of computing the values repeatedly for each pixel.

- Spatial weight LUT (table_G_s) stores precomputed Gaussian spatial weights for all possible distances up to pad_w, avoiding redundant exponential calculations during filtering.
- Range weight LUT (table_G_r) allows direct lookup of the Gaussian range weights, eliminating expensive `exp()` computations.

2. Faster Neighborhood Shifting Using `np.roll()`

The filter requires shifting both the input image and the guidance image multiple times across different spatial offsets. Instead of iterating through each pixel manually, the code uses `np.roll`, which is an optimized NumPy operation for shifting arrays efficiently.

3. Accelerating Matrix Computation with NumPy Broadcasting

When processing color images, the code utilizes NumPy broadcasting to avoid additional for loops.

```
if c > 1:
    result += shifted_img * t_w[..., None]
    wgt_sum += t_w[..., None]
else:
    result[..., 0] += shifted_img[..., 0] * t_w
    wgt_sum[..., 0] += t_w
```