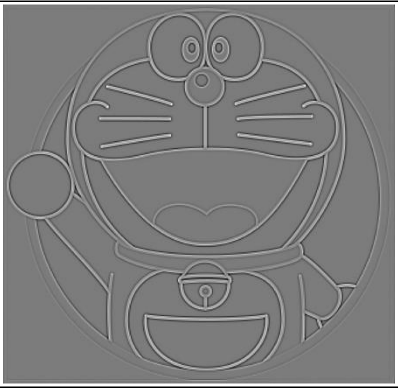
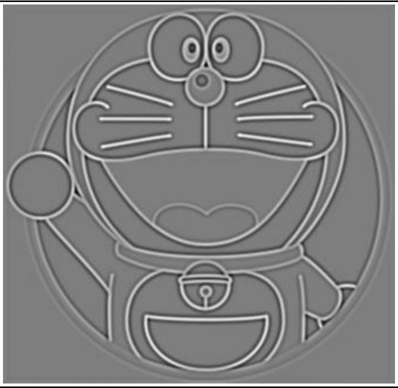
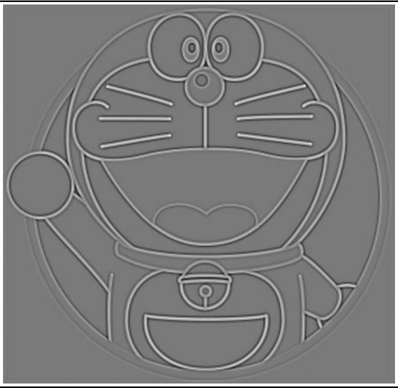
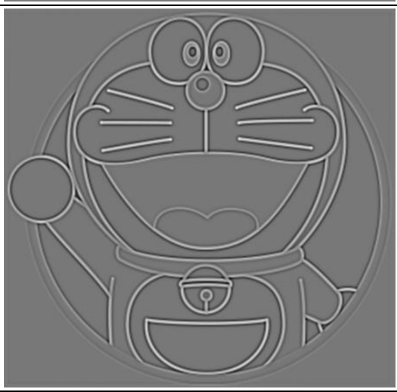
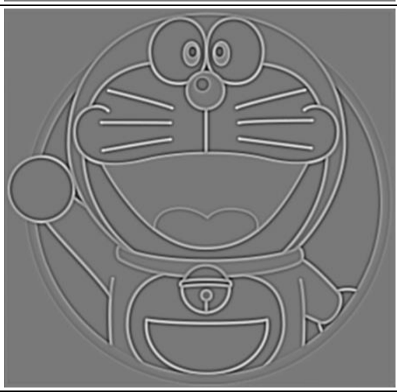
Computer Vision HW1 Report

Student ID: R10921a36
Name: 石子仙

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)

Threshold 是 2 的時候有很多在文字、盤子邊緣的點，往 threshold 越大變越少，而且留下來的 keypoint 與周圍顏色差距比較大（灰白對比黑白）

Part 2.

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

Gray Scale Setting	Cost (1.png)	Gray Scale Setting	Cost (2.png)
cv2.COLOR_BGR2GRAY	1207799	cv2.COLOR_BGR2GRAY	183850
R*0.0+G*0.0+B*1.0	1439568	R*0.1+G*0.0+B*0.9	77883
R*0.0+G*1.0+B*0.0	1305961	R*0.2+G*0.0+B*0.8	86023
R*0.1+G*0.0+B*0.9	1393620	R*0.2+G*0.8+B*0.0	188019
R*0.1+G*0.4+B*0.5	1279697	R*0.4+G*0.0+B*0.6	128341
R*0.8+G*0.2+B*0.0	1127913	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.**

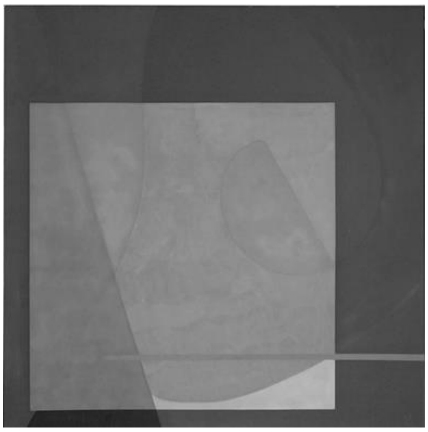
Original RGB image (1.png)	Filtered RGB image and	Filtered RGB image and

	Grayscale image of Highest cost	Grayscale image of Lowest cost
	 	 

(Describe the difference between those two grayscale images)

Highest cost: 葉子本身與草地黑白差距小，做出來的 image 邊緣較模糊（葉子的斑點看不太清楚）

Lowest cost: 葉子本身與草地黑白差距大，做出來的 image 邊緣較清楚，保留較多細節

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)

Highest cost: 相鄰色塊黑白差距小，中間橘紫顏色看起來差不多，做出來的 image 邊緣較模糊
Lowest cost: 相鄰色塊黑白差距大，做出來的 image 邊緣較清楚

- **Describe how to speed up the implementation of bilateral filter.**
  - 我在自己的電腦上測試的結果：
    - JBF.so:1.3 sec
    - My result: 0.87~0.93 sec
  - 一開始是用 for (r, c) in (row, column)的形式來算每個 pixel 的值，spatial kernel 用查表的，range kernel 用算的，大約是 8 秒。
  - 後來把 range kernel 也改用查表呼叫，大約是 3 秒。
  - 到此時我發現我是做

    for (r, c) in (row, column):

    output[r, c] = spatial[r, c] … range[r, c] … image[r, c] / spatial[r, c] … range[r, c]

    類似於這種取值方式，就覺得應該可以改成全部一起算。
  - 改成先得到一個 kernel matrix (是 spatial kernel 跟 range kernel 相乘的結果)，shape = (r, c, window, window)，以 sample 為例是(314, 314, 19, 19)。

    跟 original image 的 sliding window(314, 314, 19, 19, 3)去做 element wise 相乘再加起來得到原公式的分子。

    就只是把原本查表的方法改成包進(314, 314)的矩陣做運算，不使用 for (r, c) in (row, column)來加速，沒用到新的數學方法或定理，只是要花很多時間解決 array shape, broadcast 等問題。