

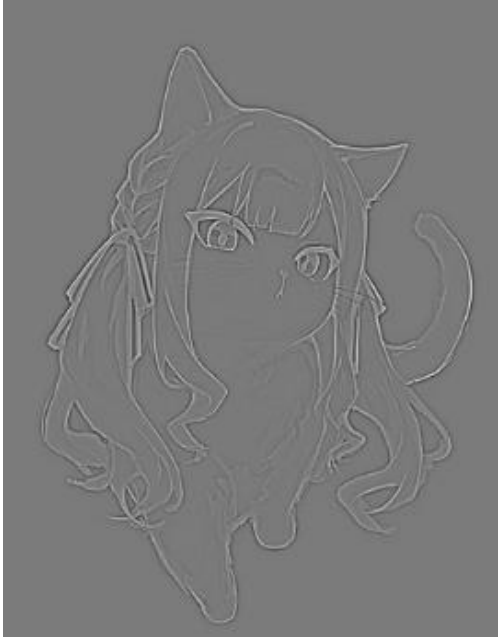



# Computer Vision HW1 Report





Student ID: R13943124

Name: 施伯儒




## 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	

從上面的比較可以看出，當 threshold 增加時，所偵測到的 keypoints 數量會減少。  
 當 threshold 等於 2.0 時，所捕捉到的 keypoints 非常雜亂，並且包含了大量的 edge points；  
 而當 threshold 等於 5.0 時，所捕捉到的 keypoints 較少 edge points，也更加具有代表性。  
 而當 threshold 等於 7.0 時，圖像上的 keypoints 幾乎都是最具代表性的點。






## Part 2.

- Report the cost for each filtered image.





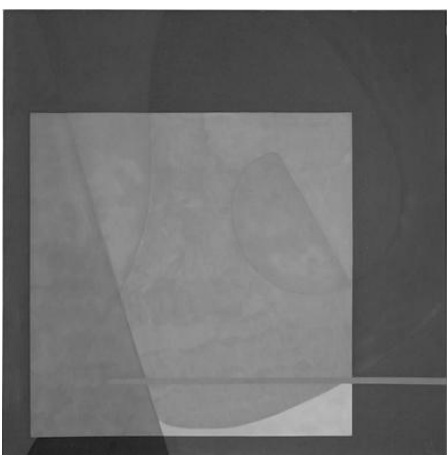
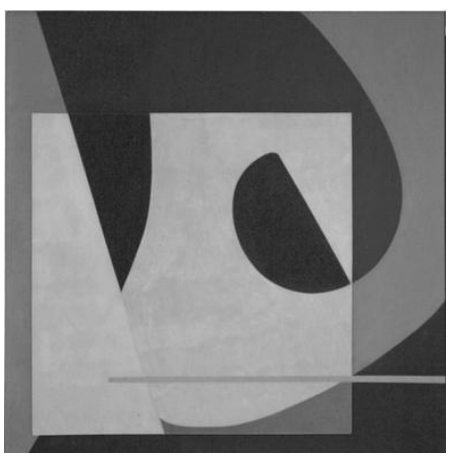
Gray Scale Setting	Cost (1.png)
cv2.COLOR_BGR2GRAY	1207799
$R*0.0+G*0.0+B*1.0$	1439568
$R*0.0+G*1.0+B*0.0$	1305961
$R*0.1+G*0.0+B*0.9$	1393620
$R*0.1+G*0.4+B*0.5$	1279697
$R*0.8+G*0.2+B*0.0$	1127913

Gray Scale Setting	Cost (2.png)
cv2.COLOR_BGR2GRAY	183850
$R*0.1+G*0.0+B*0.9$	77883
$R*0.2+G*0.0+B*0.8$	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 <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
		
		

From the grayscale image with the highest cost, We hardly differentiate the object and the something next to. While from the grayscale image with the lowest cost, we can simply differentiate the object and the something next to. That means the grayscale image generated by the weight of (0.8,0.2,0.0) is the best parameter to transform the color image to the grayscale image.

Original RGB image (2.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
		
		

From the grayscale image with the highest cost, We hardly differentiate each object from it. While from the grayscale image with the lowest cost, we can even distinguish some objects from it. That means the grayscale image generated by the weight of (0.1,0.0,0.9) is the best parameter to transform the color image to the grayscale image.



- Describe how to speed up the implementation of bilateral filter.
- 1. Using LUT for both spatial and range kernel

```
class Joint_bilateral_filter(object):
    def __init__(self, sigma_s, sigma_r):
        self.sigma_r = sigma_r
        self.sigma_s = sigma_s
        self.wndw_size = 6*sigma_s+1
        self.pad_w = 3*sigma_s

        scaleFactor_s = 1 / (2 * sigma_s * sigma_s)
        # Pixel values should be normalized to [0, 1] (divided by 255) to construct range kernel
        scaleFactor_r = 1 / (2 * sigma_r **2 * 255 ** 2)

        self.spatial_kernel = np.array(
            [ [np.exp(-(i**2+j**2) * scaleFactor_s) \
              for i in range(-(self.wndw_size // 2), (self.wndw_size +1) // 2)] \
              for j in range(-(self.wndw_size // 2), (self.wndw_size +1) // 2) ]
        )

        # Generate look up table for range kernel
        # Since the result of subtraction of intensity must fall in [0,255]
        # generate a list: (0^2 ~ 255^2) * scaleFactor_r
        self.LUT = np.exp(-np.arange(256) * np.arange(256) * scaleFactor_r)
```

在 initialization 就初始化相關的 kernel，使得到時候可以直接呼叫，而不需要再重複計算一次 Spatial Kernel 因為上下左右的差值固定，所以可以直接利用 np functions 先算完 Range Kernel 則是因為 pixel 相減的 value 落在 0~255 之間，可以直接把 0~255 平方的表建立起來。等到要用的時候查表，不須重複計算。

- 2. Using np function instead of for loop as much as possible

```
#RGB scale
for i in range(self.pad_w, padded_img.shape[0] - self.pad_w):
    for j in range(self.pad_w, padded_img.shape[1] - self.pad_w):
        patch_img = padded_img[i - self.pad_w : i + self.pad_w + 1, j - self.pad_w : j + self.pad_w + 1]
        patch_guide_r = padded_guidance[i - self.pad_w : i + self.pad_w + 1, j - self.pad_w : j + self.pad_w + 1, 0]
        center_val_r = padded_guidance[i, j, 0]

        patch_guide_g = padded_guidance[i - self.pad_w : i + self.pad_w + 1, j - self.pad_w : j + self.pad_w + 1, 1]
        center_val_g = padded_guidance[i, j, 1]

        patch_guide_b = padded_guidance[i - self.pad_w : i + self.pad_w + 1, j - self.pad_w : j + self.pad_w + 1, 2]
        center_val_b = padded_guidance[i, j, 2]

        Gr = self.LUT[ np.abs(patch_guide_r - center_val_r) ] * self.LUT[ np.abs(patch_guide_g - center_val_g) ] * self.LUT[ np.abs(patch_guide_b - center_val_b) ]
        Gr_with_Gs = Gr * self.spatial_kernel
        output[i - self.pad_w, j - self.pad_w] = np.sum(Gr_with_Gs[:, np.newaxis] * patch_img, axis=(0,1)) / np.sum(Gr_with_Gs)
```

所有 function 可以呼叫 np function 都盡量呼叫，不使用自己的 for loop。  
非必要不使用 for loop。