

CSC420 Report 3

Owner's Information

First Name: Kwan Kiu

Last Name: Choy

Student No: 1005005879

UtorId: choykwan

[Question 1](#)

[Part 1](#)

[Part 2](#)

[Part 3](#)

[Question 2](#)

[Part 1](#)

[Part 2](#)

[Question 3](#)

[Part 1](#)

[Part 2](#)

[Part 3](#)

[Question 4](#)

[Part 1](#)

[Part 2](#)

[Part 3](#)

[Part 4](#)

Question 1

Part 1

- 1) Response of the LoG filter to an image of a black circle with diameter D on a white background.
Let $R(\sigma, D)$ denote the response.

Since we are using normalized

$$R(\sigma, D) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \sigma^2 \nabla^2 G(x, y, \sigma) I[x^2 + y^2 \leq (D/2)^2] dx dy$$

$$= \sigma^2 \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{1}{\pi \sigma^4} \left(\frac{x^2 + y^2}{2\sigma^2} - 1 \right) e^{-\frac{x^2 + y^2}{2\sigma^2}} I[x^2 + y^2 \leq (D/2)^2] dx dy$$

$$= \sigma^2 \int_0^{D/2} \int_0^{D/2} r \frac{1}{\pi \sigma^4} \left(\frac{r^2}{2\sigma^2} - 1 \right) e^{-\frac{r^2}{2\sigma^2}} dr d\theta$$

$$= \sigma^2 \frac{1}{\pi \sigma^4} \int_0^{2\pi} \int_0^{D/2} r \left(\frac{r^2}{2\sigma^2} - 1 \right) e^{-\frac{r^2}{2\sigma^2}} dr d\theta$$

$$= \frac{1}{\pi \sigma^2} \int_0^{2\pi} \int_0^{D/2} \left(\frac{r^3}{2\sigma^2} e^{-\frac{r^2}{2\sigma^2}} - r e^{-\frac{r^2}{2\sigma^2}} \right) dr d\theta$$

$$= \frac{1}{2\pi \sigma^4} \int_0^{2\pi} \int_0^{D/2} r^3 e^{-\frac{r^2}{2\sigma^2}} - 2\sigma^2 r e^{-\frac{r^2}{2\sigma^2}} dr d\theta$$

$$= \frac{1}{2\pi \sigma^4} \int_0^{2\pi} 2\sigma^4 \left[-\frac{1}{2} e^{-\frac{r^2}{2\sigma^2}} \right]_0^{D/2} - 2\sigma^2 \left[-e^{-\frac{r^2}{2\sigma^2}} \right]_0^{D/2} d\theta$$

$$= \frac{1}{2\pi \sigma^4} \int_0^{2\pi} \left(-\frac{1}{2} e^{-\frac{D^2}{8\sigma^2}} + \frac{1}{2} \right) 2\sigma^4 (D^2 + 8\sigma^2) + 2\sigma^4 e^{-\frac{D^2}{8\sigma^2}} d\theta$$

$$= \frac{1}{2\pi \sigma^4} \int_0^{2\pi} \left(-\frac{1}{4} e^{-\frac{D^2}{8\sigma^2}} \sigma^4 (D^2 + 8\sigma^2) + 2\sigma^4 \right) d\theta$$

$$= \frac{-(D^2 e^{-\frac{D^2}{8\sigma^2}})}{4\sigma^2}$$

$$u = r^2 / 2\sigma^2$$

$$du = \frac{2r}{2\sigma^2} dr$$

$$dr = \frac{du \sigma^2}{r}$$

$$\int_0^{D/2} r^3 e^{-u} dr$$

$$= \int_0^{D/2} r^2 e^{-u} \frac{du \sigma^2}{r}$$

$$= \sigma^2 \int_0^{D/2} 2\sigma^2 u e^{-u} du$$

$$= 2\sigma^4 \int_0^{D/2} u e^{-u} du$$

$$= 2\sigma^4 (-u e^{-u} - \int e^{-u} du)$$

$$= 2\sigma^4 (-u e^{-u} + e^{-u}) \Big|_0^{D/2}$$

$$= 2\sigma^4 - \left(\frac{D^2}{8\sigma^2} e^{-\frac{D^2}{8\sigma^2}} + e^{-\frac{D^2}{8\sigma^2}} \right)$$

$$= 2\sigma^4 - \frac{1}{4} e^{-\frac{D^2}{8\sigma^2}} \sigma^2 (D^2 + 8\sigma^2)$$

$$\int_0^{D/2} r e^{-r^2/2\sigma^2}$$

$$= \int_0^{D/2} \sigma^2 e^{-u} du$$

$$= \sigma^2 (-e^{-u}) \Big|_0^{D/2}$$

$$= \sigma^2 (-e^{-\frac{D^2}{8\sigma^2}} + 1)$$

$$= \sigma^2 (1 - e^{-\frac{D^2}{8\sigma^2}})$$

To get the sigma that maximizes the response,

$$R(\sigma, D) = \frac{D^2 e^{-\frac{D^2}{8\sigma^2}}}{4\sigma^2}$$

$$\frac{dR}{d\sigma} = \frac{\frac{d}{d\sigma} (D^2 e^{-\frac{D^2}{8\sigma^2}}) 4\sigma^2 - 8\sigma (D^2 e^{-\frac{D^2}{8\sigma^2}})}{16\sigma^4} = \frac{\frac{D^2}{4\sigma^3} D^2 e^{-\frac{D^2}{8\sigma^2}} \cdot \frac{1}{\sigma^2} - 8\sigma D^2 e^{-\frac{D^2}{8\sigma^2}}}{16\sigma^4}$$

$$= \frac{\frac{D^4}{4\sigma^5} e^{-\frac{D^2}{8\sigma^2}} - 8\sigma D^2 e^{-\frac{D^2}{8\sigma^2}}}{16\sigma^4} = \frac{D^4 e^{-\frac{D^2}{8\sigma^2}} - 32\sigma^5 e^{-\frac{D^2}{8\sigma^2}}}{16\sigma^5}$$

$$= \frac{D^2 e^{-\frac{D^2}{8\sigma^2}} (D^2 - 32\sigma^2)}{16\sigma^5}$$

Set $\frac{dR}{d\sigma} = 0$ and solve for σ .

$$D^2 e^{-\frac{D^2}{8\sigma^2}} (D^2 - 32\sigma^2) = 0$$

$$D^2 - 32\sigma^2 = 0$$

$$(D + \sqrt{32}\sigma)(D - \sqrt{32}\sigma) = 0$$

$$D + \sqrt{32}\sigma = 0 \text{ or } D - \sqrt{32}\sigma = 0$$

$$\sigma = -\frac{D}{\sqrt{32}} \text{ this is rejected } \frac{D}{\sqrt{32}} = \sigma$$

So the σ that gives us the max response is $\sigma = D/\sqrt{32}$

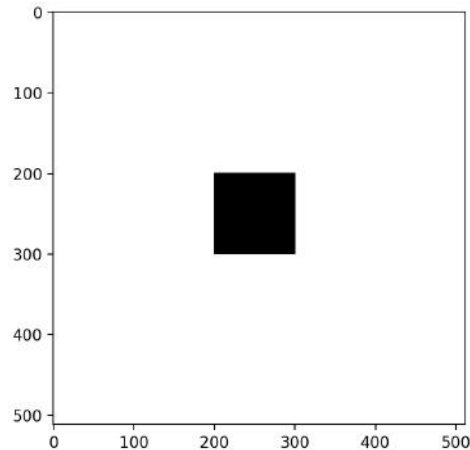
Part 2

The scale used should remain the same since there is only 1 solution when we set $\frac{dR}{d\sigma} = 0$. However, instead of looking for the maximum, we should be looking at the minimum since the pixel inside the circle is white but not black.

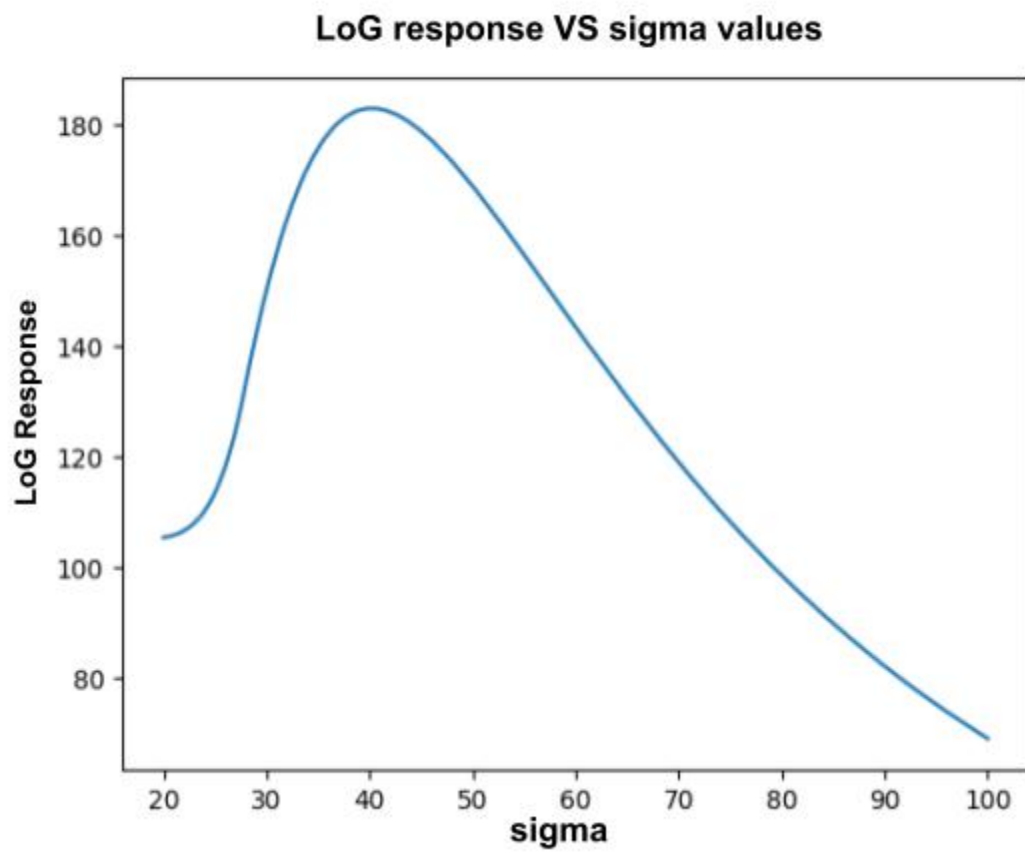
Part 3

I used the code below to generate a 500 * 500 large image, where the

```
fig = plt.figure()
whiteblackimage = 255 * np.ones(shape=[500, 500, 3])
cv2.rectangle(whiteblackimage, pt1=(200,200), pt2=(300,300), color=(0,0,0),
thickness=-1)
whiteblackImage = np.array(whiteblackimage)
```



I have plotted the LoG response VS the sigma values, and obtained the sigma value that produces the maximum LoG response value.



The maximum LoG response is 183.07166263121093, and the sigma used is 40.50251256281407.

Question 2

Part 1

$$2) \quad M = \sum_x \sum_y w(x, y) \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix}$$

a) Compute the eigenvalues of N where $N = \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix}$

$$N = \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix}$$

$$N - \lambda I = \begin{pmatrix} I_x^2 - \lambda & I_x I_y \\ I_x I_y & I_y^2 - \lambda \end{pmatrix}$$

$$|N - \lambda I| = 0$$

$$(I_x^2 - \lambda)(I_y^2 - \lambda) - (I_x I_y)^2 = 0$$

$$\cancel{I_x^2 I_y^2} - I_y^2 \lambda - I_x^2 \lambda + \lambda^2 - \cancel{I_x^2 I_y^2} = 0$$

$$\lambda^2 - \lambda(I_y^2 + I_x^2) = 0$$

$$\lambda(\lambda - (I_y^2 + I_x^2)) = 0$$

$$\lambda_1 = 0 \quad \text{or} \quad \lambda_2 = I_y^2 + I_x^2$$

Part 2

b)

Prove that the matrix M is a positive semi definite

A positive semi definite matrix is a Hermitian matrix such that all eigenvalues are non negative.

First of all, we know that the two eigen values of N are either 0 or $I_y^2 + I_x^2$. Since $I_y^2 + I_x^2$ is a sum of squares, $I_y^2 + I_x^2 \geq 0$. Then, we know that the 2 eigen values of N are all non negative.

$$M = \sum_x \sum_y w(x,y) \begin{pmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{pmatrix} \quad \leftarrow \text{this is positive semi definite}$$

By the definition of $w(x,y)$, which is a window function that could be :

- 1) a box function that outputs 1 if (x,y) is in window and 0 otherwise
- 2) or a gaussian distribution that ranges from 0 to 1

Since the output of $w(x,y)$ is also nonnegative, we know that $w(x,y) \cdot N$ will also be a positive semi definite matrix.
Proof is shown below:

Let $c = w(x,y)$ where we know that $0 \leq c \leq 1$

$$c \begin{pmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{pmatrix} = \begin{pmatrix} c I_x^2 & c I_x I_y \\ c I_y I_x & c I_y^2 \end{pmatrix}$$

$$\begin{aligned} (c I_x^2 - \lambda)(c I_y^2 - \lambda) - c^2 I_x^2 I_y^2 &= 0 \\ \cancel{c I_x^2 I_y^2} - c I_x^2 \lambda - c I_y^2 \lambda + \lambda^2 - \cancel{c^2 I_x^2 I_y^2} &= 0 \\ \lambda^2 - \lambda(c I_x^2 + c I_y^2) &= 0 \\ \lambda(\lambda - (c I_x^2 + c I_y^2)) &= 0 \end{aligned}$$

$$\text{So } \Rightarrow \lambda_1 = 0 \text{ or } \lambda_2 = c I_x^2 + c I_y^2$$

Since c is non negative, λ_2 is also non negative and hence $w(x,y)N$ is a positive semi definite matrix

Next, since M is a sum of positive semi definite matrix, I would like to prove that the sum of 2 positive semi definite matrix is also a positive semi definite matrix.

Let A and B be two positive semi definite matrices, this means for any $\vec{v} \in \mathbb{R}^n$ that $\vec{v} \neq \vec{0}$:

$$\vec{v}^T A \vec{v} > 0 \text{ and } \vec{v}^T B \vec{v} > 0$$

We want to show that $\vec{v}^T (A+B) \vec{v} > 0$.

$$\begin{aligned} &\vec{v}^T (A+B) \vec{v} \\ &= (\vec{v}^T A + \vec{v}^T B) \vec{v} \quad \text{By the distributive properties} \\ &= \vec{v}^T A \vec{v} + \vec{v}^T B \vec{v} \quad \text{of matrix multiplication.} \end{aligned}$$

Since $\vec{v}^T A \vec{v} > 0$ and $\vec{v}^T B \vec{v} > 0$ by assumption, we can conclude that $\vec{v}^T (A+B) \vec{v} > 0$.

hence we know that the sum of two positive semi definite matrix is also a positive semi definite matrix.

Since M is just a sum of positive semi-definite matrices and we've shown that the sum will also be a positive semi-definite matrix, we've shown that M is a positive semi-definite matrix.

Question 3

Part 1

```
def center_crop(image, m, n, tau):
    center_x, center_y = image.shape[0] // 2, image.shape[1] // 2
    return image[center_x - m*tau//2: center_x+ m*tau//2, center_y -
n*tau//2: center_y + n*tau//2]

def construct_HOG(orientation, gradient, m, n, tau):
    bins = [(-15, 15), (15, 45), (45, 75), (75, 105), (105, 135), (135,
165)]
    matrix = np.zeros((m,n,len(bins)))
    bincounts = np.zeros((m, n, len(bins)))
    for i in range(orientation.shape[0]):
        for j in range(orientation.shape[1]):
            i_m = i // tau
            j_n = j // tau
            # handle the case when orientation is larger than 165
            if orientation[i][j] >= 165:
                matrix[i_m][j_n][0] += gradient[i][j]
                bincounts[i_m][j_n][0] += 1
                continue
            for k in range(len(bins)):
                if bins[k][0] <= orientation[i][j] < bins[k][1]:
                    matrix[i_m][j_n][k] += gradient[i][j]
                    bincounts[i_m][j_n][k] += 1
                    break
    return matrix, bincounts

def compute_image_gradient_and_threshold(image, tau, epsilon):
    # center crop an image
    dim = image.shape
    print(dim)
    m, n = dim[0] // tau, dim[1] // tau
    print(f"M, N:{m, n}")
    image = center_crop(image, m, n, tau)
    print(image.shape)
    # compute gradients of x and y directions
    g_x = cv2.Sobel(image, cv2.CV_64F, 1, 0)
    g_y = cv2.Sobel(image, cv2.CV_64F, 0, 1)
    # compute the magnitude of gradients
```

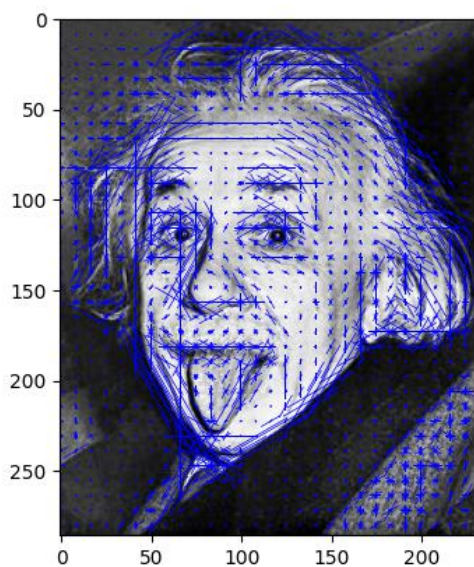
```

gradient = np.sqrt(np.square(g_x) + np.square(g_y))
orientation = np.arctan2(g_y, g_x) * (180 / np.pi)
# Thresholding gradient magnitude
gradient = np.where(gradient < epsilon, 0, gradient)
# Convert the negative orientation
orientation = np.where(orientation < 0, 180+orientation, orientation)
# construct HOG
print(orientation.shape)
gradient_mtx, bincounts = construct_HOG(orientation, gradient, m, n,
tau)
return gradient, orientation, gradient_mtx, bincounts

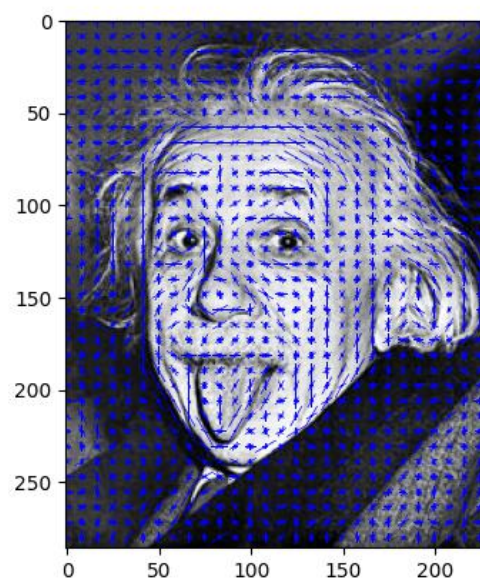
def generate_quiver_plot(gradient_mtx, image, tau, name):
    m, n, bincount = gradient_mtx.shape
    print(m, n)
    bins = [(-15, 15), (15, 45), (45, 75), (75, 105), (105, 135), (135,
165)]
    # X and Y are 2D
    X, Y = np.meshgrid(np.linspace(0, n*tau, n), np.linspace(0, m*tau, m))
    fig, ax = plt.subplots()
    ax.imshow(image, cmap="gray")
    # Create a quiver plot for each of the bin
    # Make U and V 2D as well => have to loop over each bin
    # Obtain the U and V with sin and cos functions
    for k in range(bincount):
        mean_angle = (bins[k][0] + bins[k][1])/2 * np.pi/180
        # Compute x component by multiplying |gradient|cos(\theta)
        U = np.sin(mean_angle) * gradient_mtx[:, :, k]
        # Compute y component by multiplying |gradient|sin(\theta)
        V = np.cos(mean_angle) * gradient_mtx[:, :, k]
        #ax.quiver(X, Y, U, V, color="blue", pivot="middle")
        ax.quiver(X, Y, U, V, color="blue", linewidth=.5, headlength=0,
headwidth=1, headaxislength=0, pivot="middle")
    # save fig
    fig.savefig(f"{name}_quiver.png")
    # show figure
    #fig.show()

```

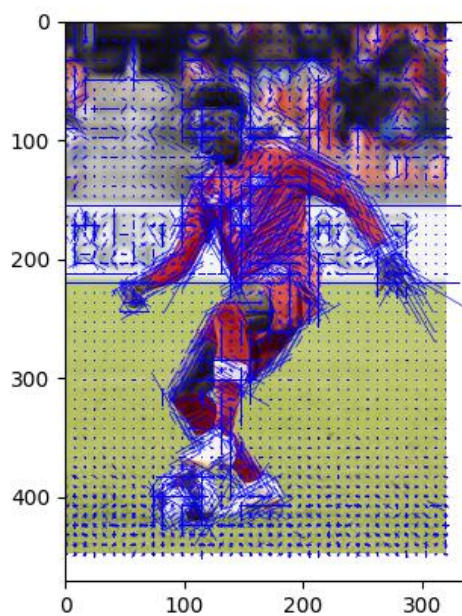
cell=8, epsilon=0.1 (Gradient Magnitude)



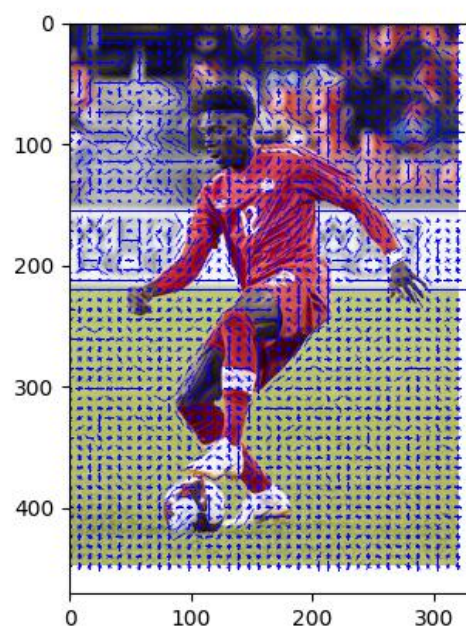
cell=8 (Orientation Count)

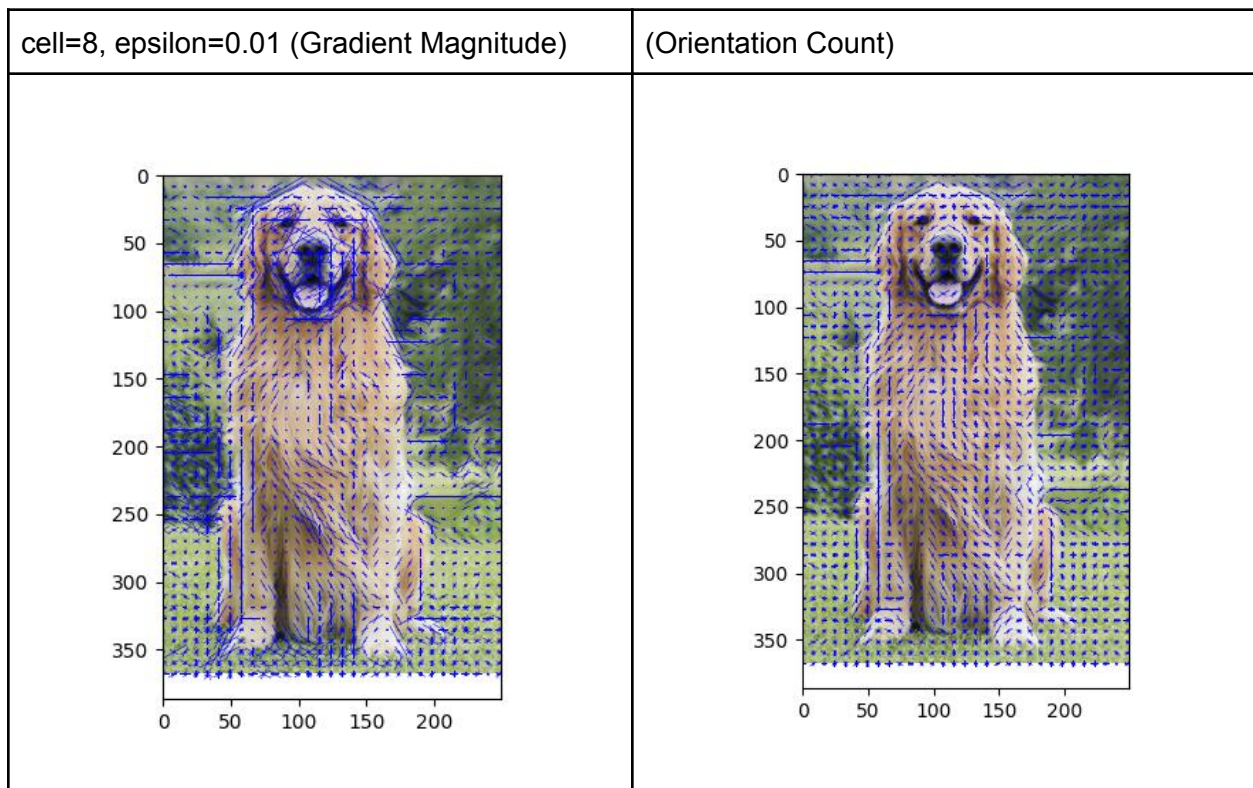


cell=8, epsilon=0.01 (Gradient Magnitude)



(Orientation Count)





Comparing between the Gradient Magnitude and the Orientation Count approach:

Both approaches display the orientations for each of the pixels in a cell. By using the gradient magnitude approach, we can observe the change of the intensity through the length of the arrow. The orientation count approach, on the other hand, produces arrows of the same length and we are only counting the frequencies of a specific orientation in that cell block. The quiver plot produced by the orientation count approach is much clearer as there are no overlapping arrows, but with less information (since we couldn't see the gradient magnitude in this plot). Therefore, if knowing the gradient magnitude is not needed, then the orientation count approach creates quiver plots that are easier to visualize.

Part 2

Code for normalizing the gradient magnitude

```
def get_normalized_histogram(gradient_mtx, name, block_size=2, e=0.0001,
bin_count=6):
    m, n, _ = gradient_mtx.shape
    normalized_histogram = []
    txt_array = []
    for i in range(m - block_size + 1):
        for j in range(n - block_size + 1):
```

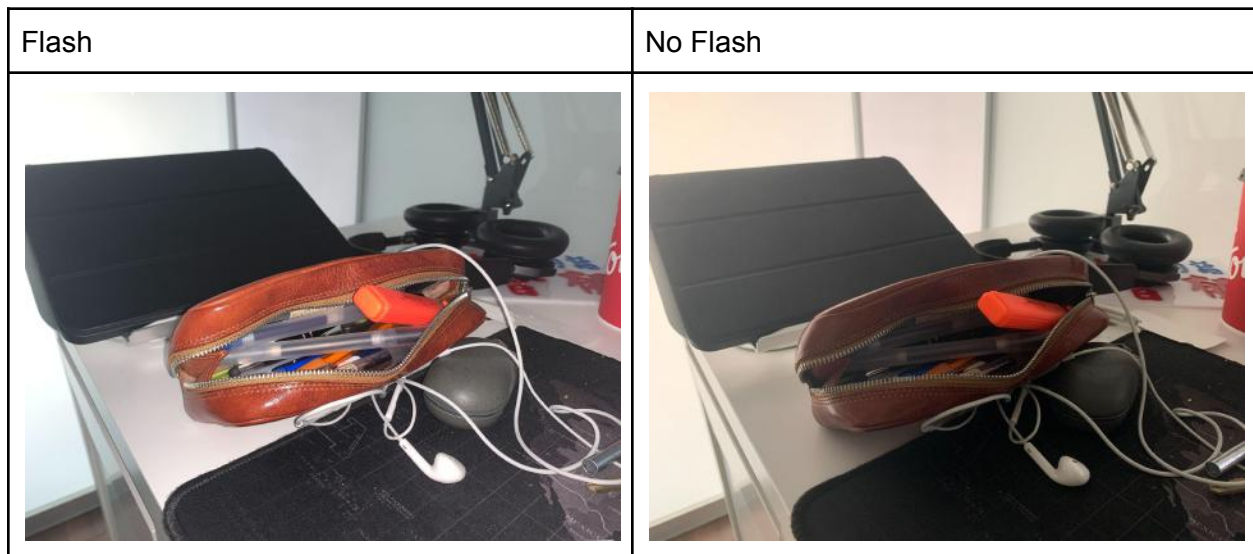


```

        h_i = gradient_mtx[i:i+block_size, j:j+block_size].flatten()
        normalized_descriptor = h_i/np.sqrt(np.sum(np.square(h_i)) +
np.square(e))
        normalized_histogram.append(normalized_descriptor)
        txt_array.append(normalized_descriptor)
    normalized_histogram = np.array(normalized_histogram).reshape((m-1,
n-1, block_size*block_size*bin_count))
    np.savetxt(f"./{name}.txt", np.array(txt_array), delimiter=",")
    return normalized_histogram

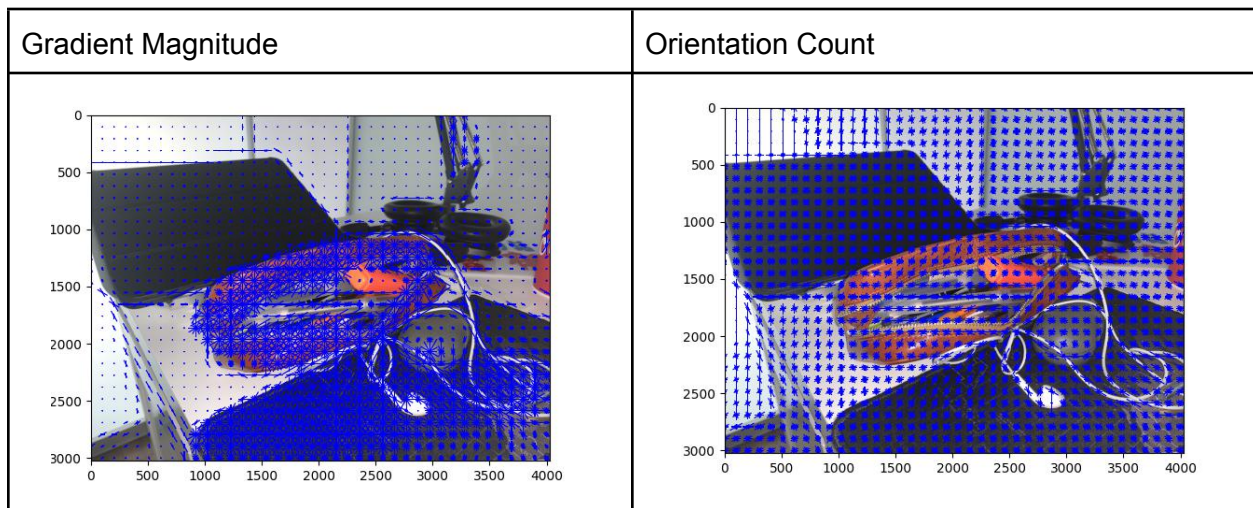
```

Part 3

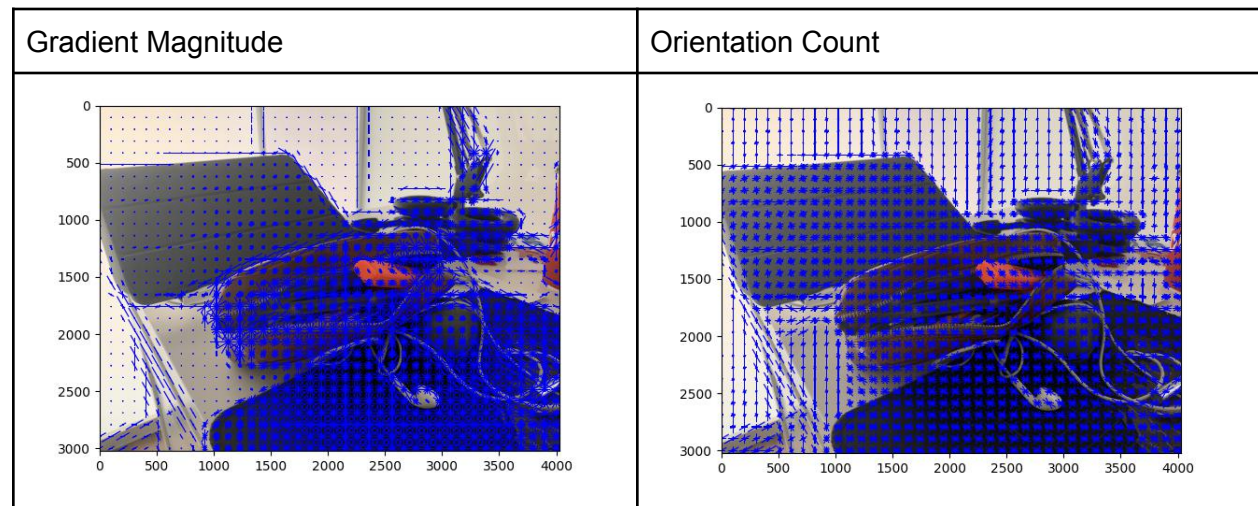


Before Normalization

Flash's HOG

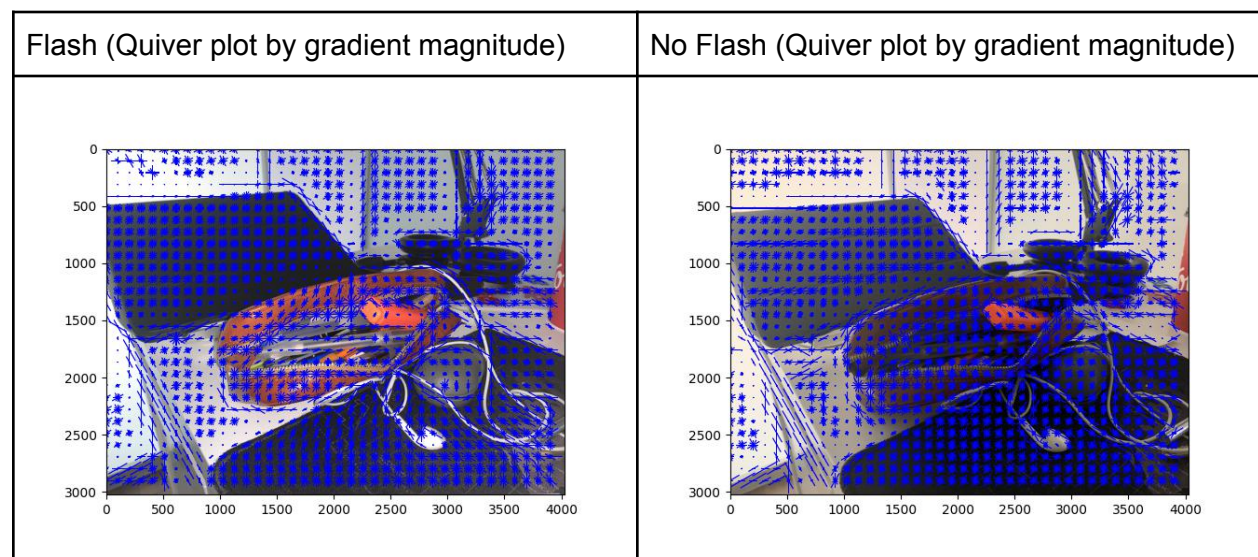


No Flash's HOG



After Normalization

To visualize the normalized histograms, I am slicing the normalized matrix. Instead of getting all of the 24 bins, we are only getting the first 6, since the rest are normalized gradient magnitudes of neighbourhood pixels, which are less relevant for visualization purposes.



After normalization, the two quiver plots look more similar to each other. Before the normalization, there are a lot of blue arrows around areas of the pencil case, above and below the opened zipper, as it was where that was illuminated by the flash. This is a significant contrast compared to that of the quiver plot of the no flash image. However, after normalization, the pencil case has less blue arrows, meaning that the distribution of blue arrows around the

center part of the image is more even. This change makes the plot look more similar to the quiver plot in the no flash image.

To measure this difference quantitatively, I have also attempted to measure the differences of the two images using the sum of least squares. Using the sum of least squares prevents differences from cancelling with each other. To make sure that this is a fair comparison, when I was computing the least square differences of the gradient magnitude matrices before normalization, I made sure the shape of the gradient magnitude matrix is of shape (m-1, n-1, 6).

```
print(np.sum(np.square(gradient_mtx4[:-1, :-1, :] - gradient_mtx5[:-1, :-1, :])))
print(np.sum(np.square(normalized4[:, :, :6] - normalized5[:, :, :6])))
```

At the end, the differences are as shown below

```
[1] 32256318465600.69
[2] 60.38666264144986
```

As you can see then, the differences between the two gradient magnitude matrices are much higher before normalization. Consequently, we can say that normalization can reduce the effects of illumination and contrast and hence normalizing HoG could be beneficial.

All the txt files generated from the normalization function are in the txtfiles directory.

Q3_1.txt => normalized gradient magnitude of Einstein's image

Q3_2.txt => normalized gradient magnitude of Man-Playing-Soccer image

Q3_3.txt => normalized gradient magnitude of Dog image

Q3_4.txt => normalized gradient magnitude of Flash image

Q3_5.txt => normalized gradient magnitude of No Flash image

Question 4

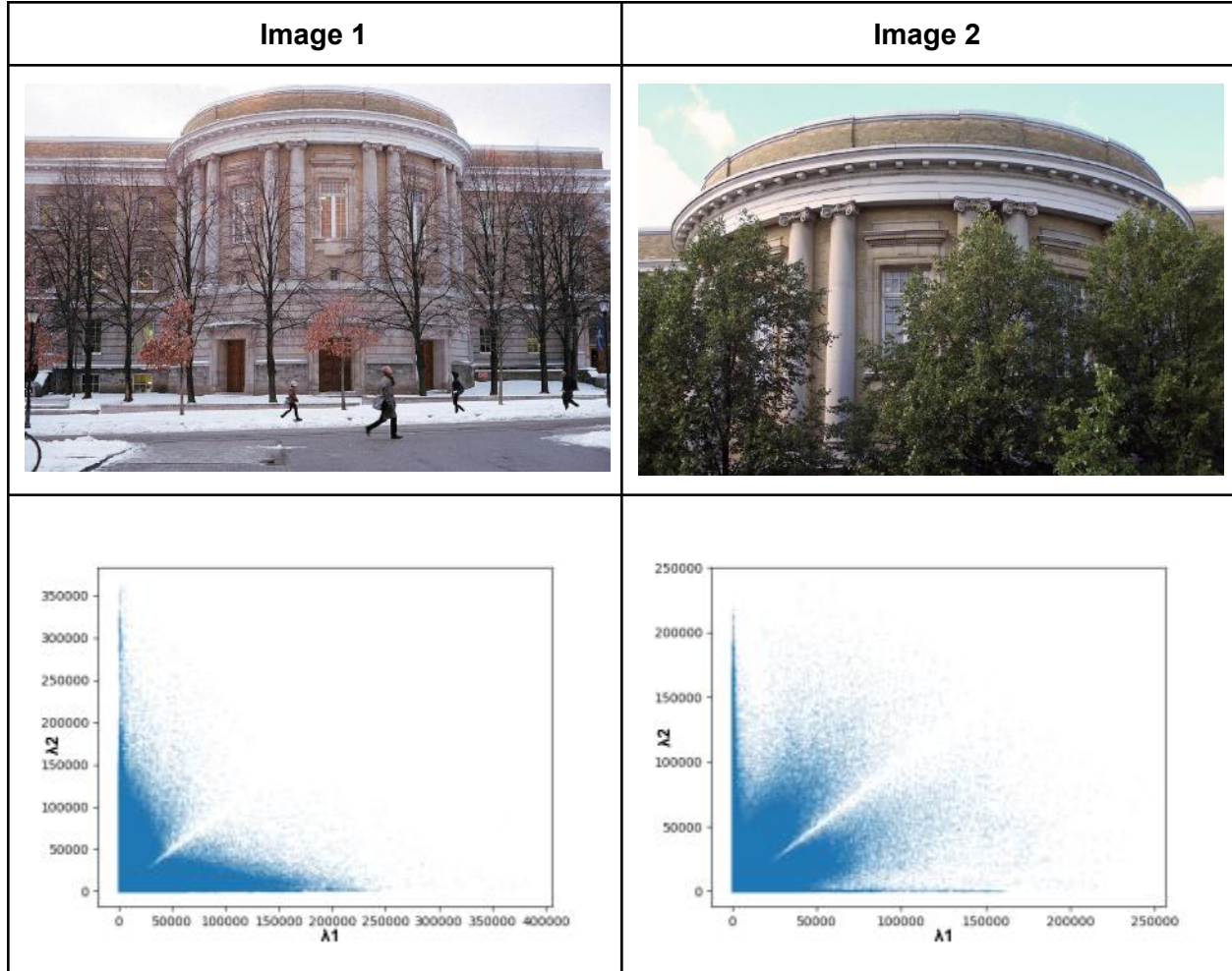
Part 1

Code for generating the second moment matrix

```
def generate_second_moment_matrix(image, ksize=7, sigma=10):
    m, n = image.shape
    eigenvalues = np.zeros((m, n, 2))
    # compute gradients of x and y directions
    Ix = cv2.Sobel(image, cv2.CV_64F, 1, 0)
    Iy = cv2.Sobel(image, cv2.CV_64F, 0, 1)
    IxIy = np.multiply(Ix, Iy)
    Ix2 = np.multiply(Ix, Ix)
```

```
Iy2 = np.multiply(Iy, Iy)
Ix2_blur = cv2.GaussianBlur(Ix2, ksize=(ksize, ksize), sigmaX=sigma)
Iy2_blur = cv2.GaussianBlur(Iy2, ksize=(ksize, ksize), sigmaX=sigma)
IxIy_blur = cv2.GaussianBlur(IxIy, ksize=(ksize, ksize), sigmaX=sigma)
for i in range(m):
    for j in range(n):
        matrix = np.array([[Ix2_blur[i, j], IxIy_blur[i, j]],
                             [IxIy_blur[i, j], Iy2_blur[i, j]]])
        eigenvalues[i, j] = np.linalg.eigvals(matrix)
    det = np.multiply(Ix2_blur, Iy2_blur) -
np.multiply(IxIy_blur, IxIy_blur)
    trace = Ix2_blur + Iy2_blur
return eigenvalues, det, trace
```

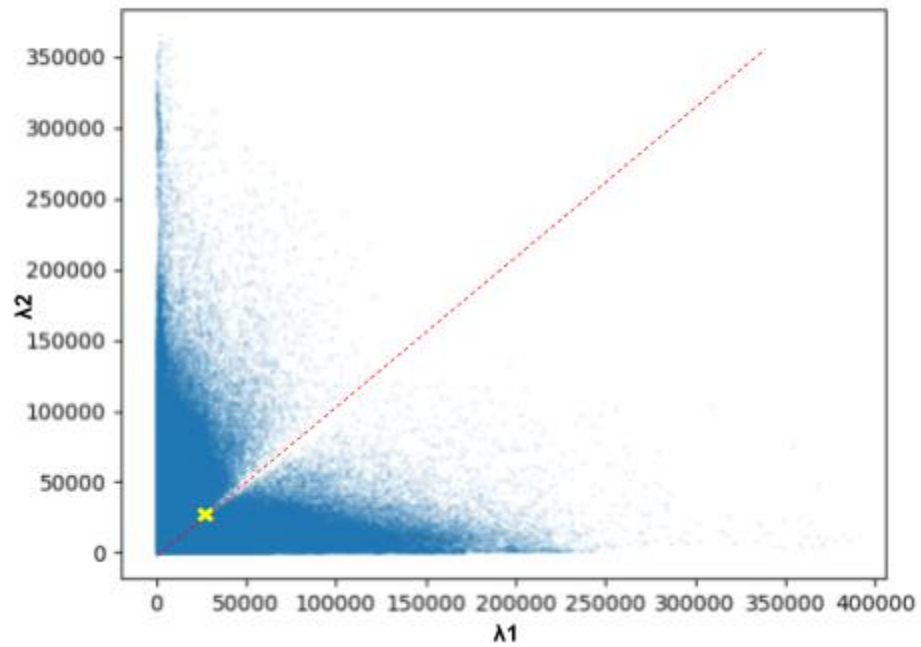

Part 2



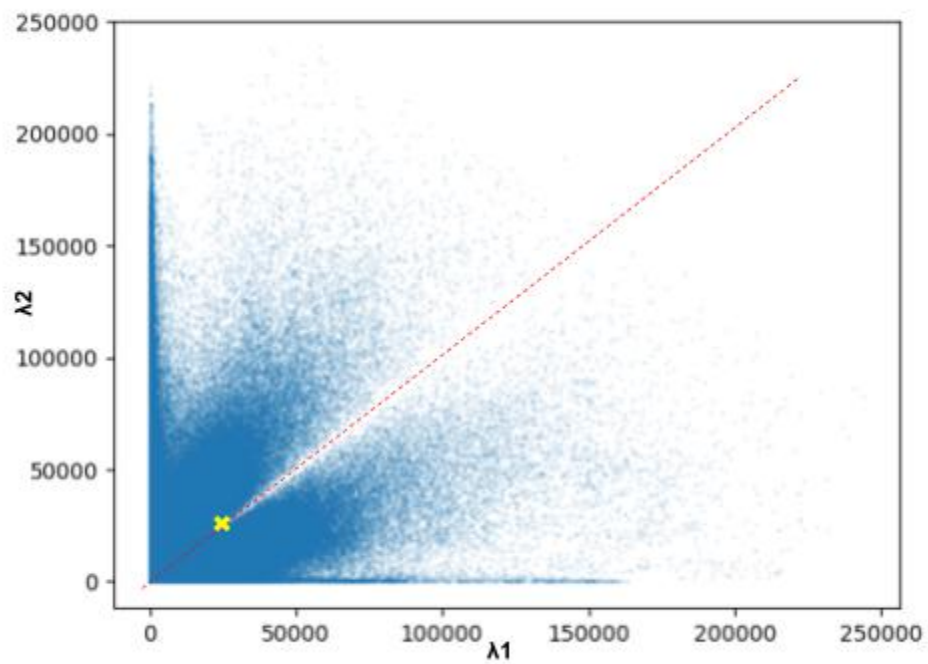
Part 3

We know that from the lectures slides that if that pixel is a corner, then $\lambda_0 \sim \lambda_1$ and both λ_0, λ_1 have to be large. Therefore, to obtain a reasonable threshold value based on the scatterplots, we can plot a line with $y=x$ on top of the scatterplot, and get the dot that lies on the line with the biggest x and y values.

For image1, the threshold is marked as a yellow cross in the scatter plot below, which is **25000**.



For image 2, the threshold is marked as a yellow cross in the scatterplot below, which is **30000**.


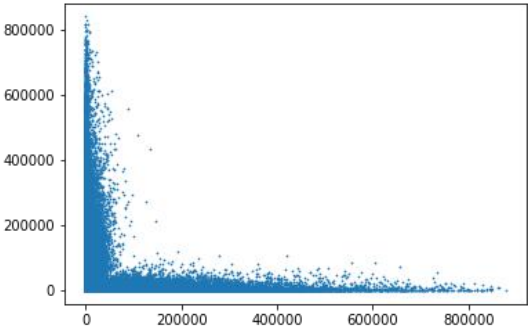


With a kernel size of 7 and sigma of 10 for the gaussian filter, the two images generated with detected corners are shown below.

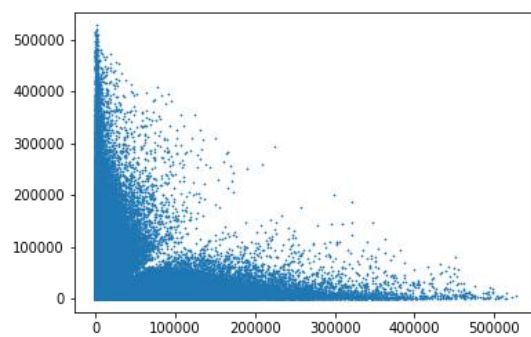


Part 4

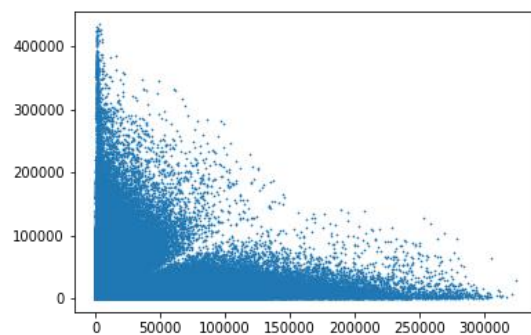
Image 1

σ	Ksize=7	Scatterplots: x- λ_1 and y- λ_2
0.5		

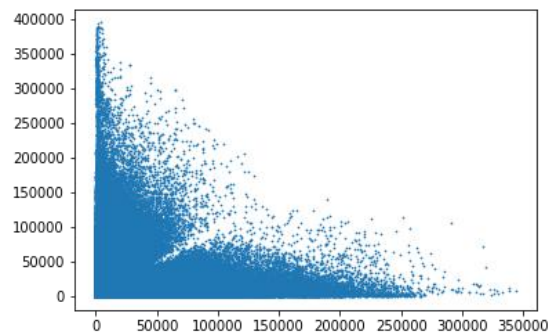
1



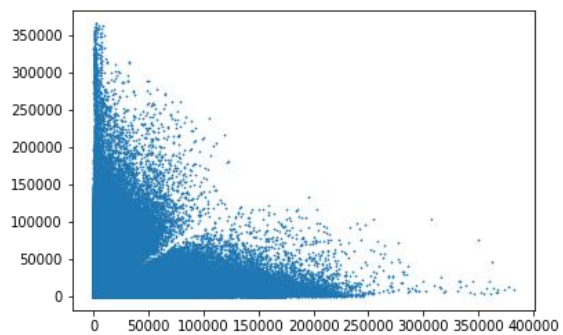
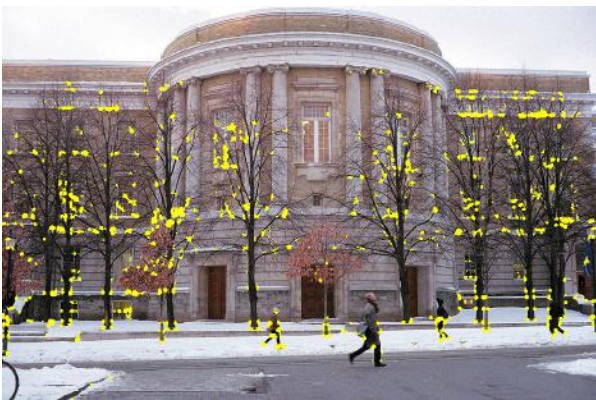
2



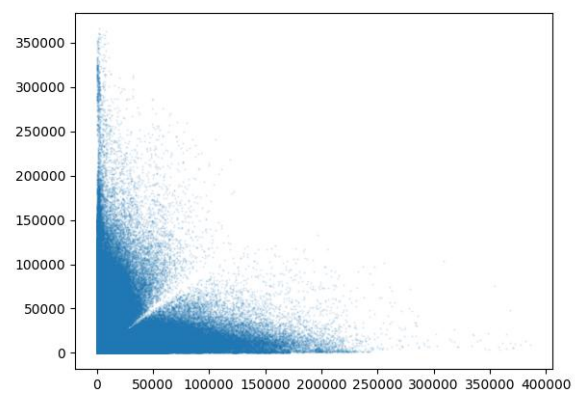
3



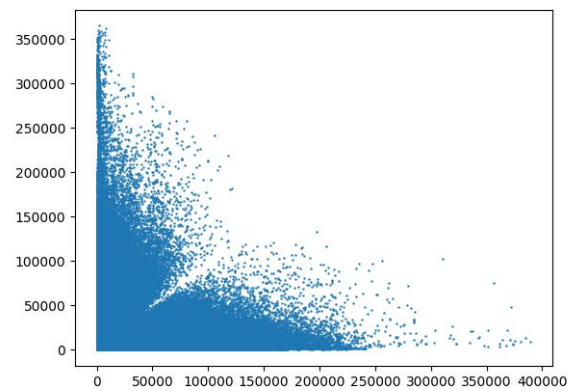
7

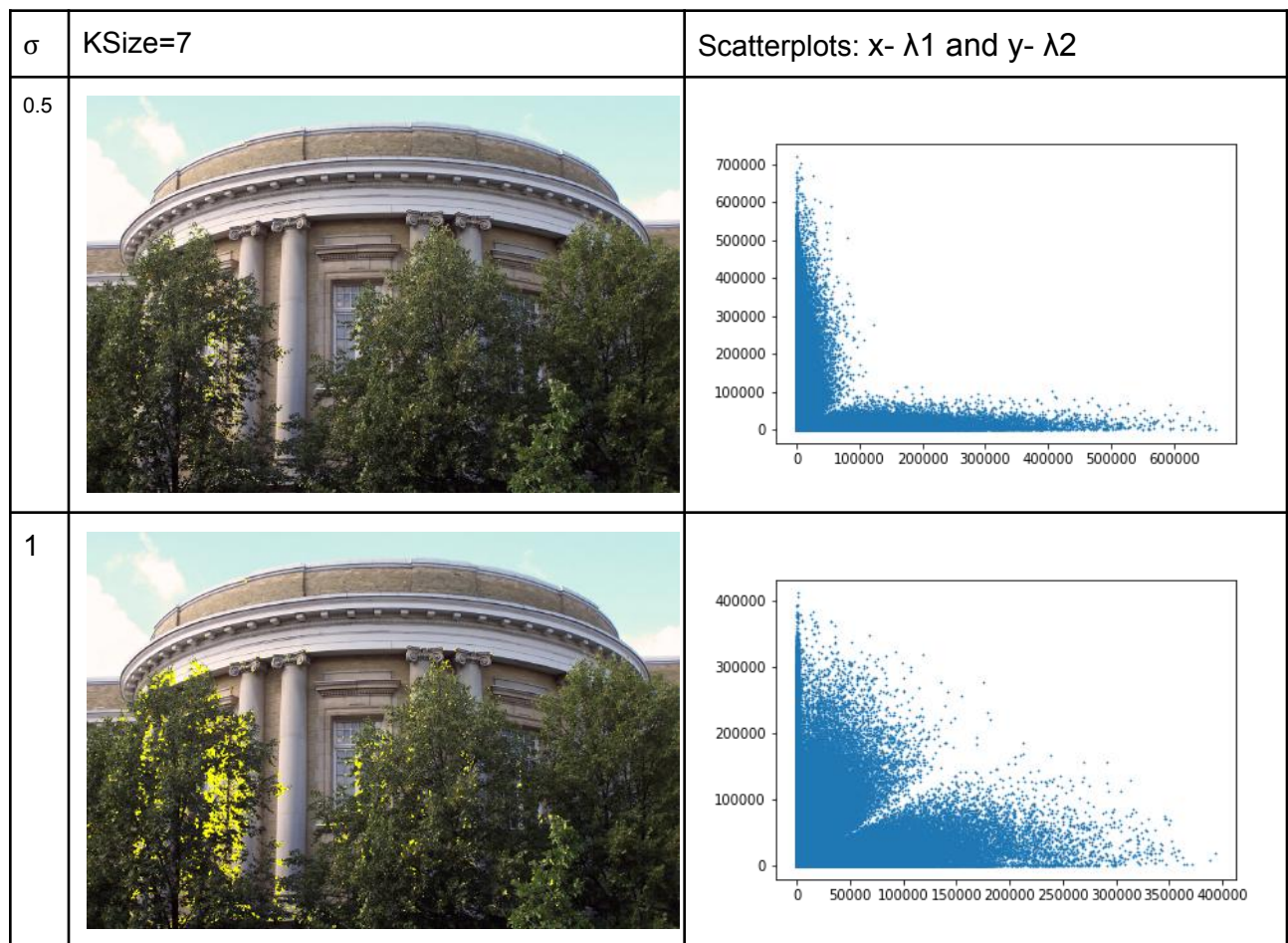
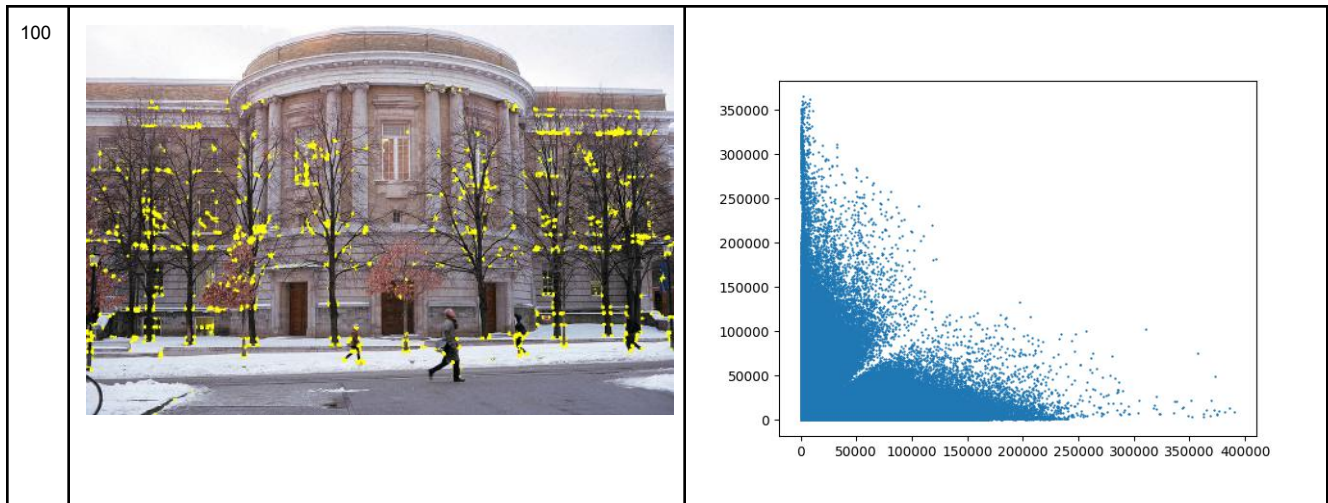


10

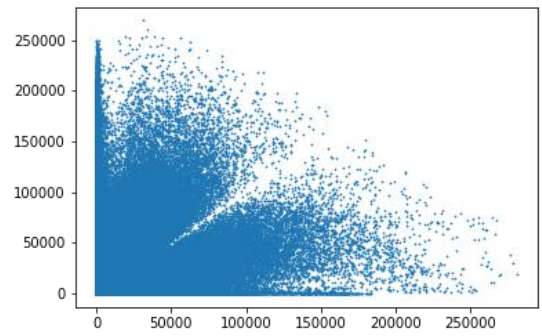
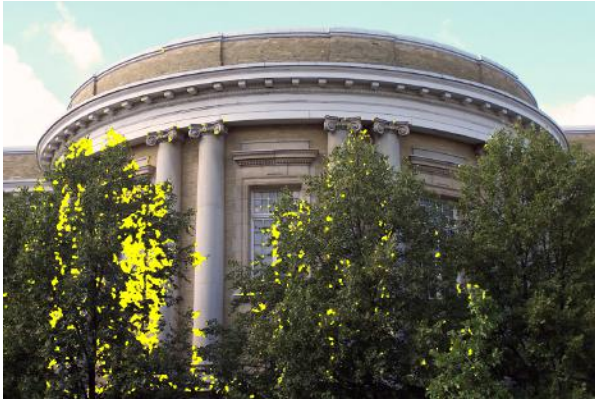


20

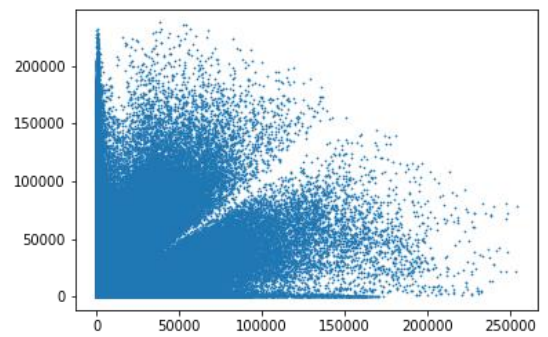
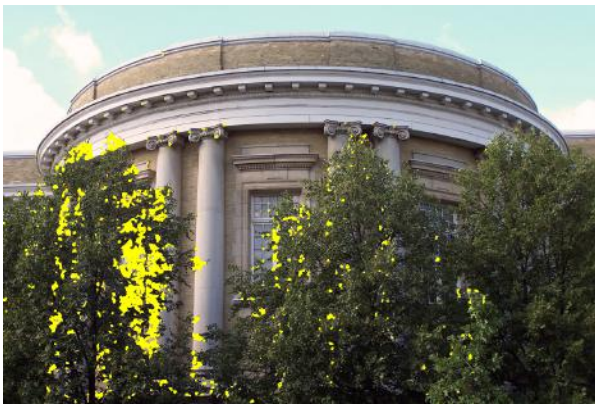




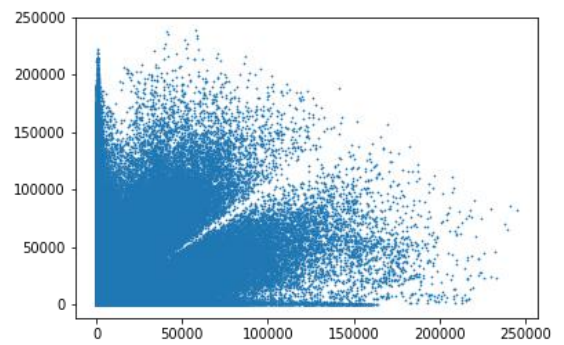
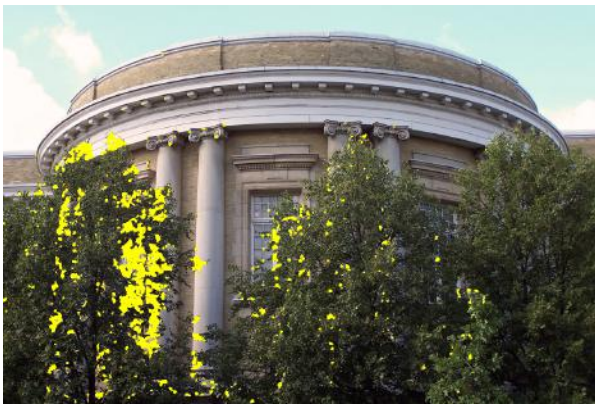
2

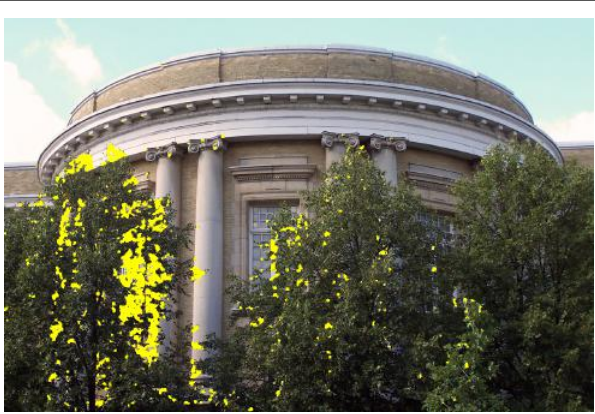
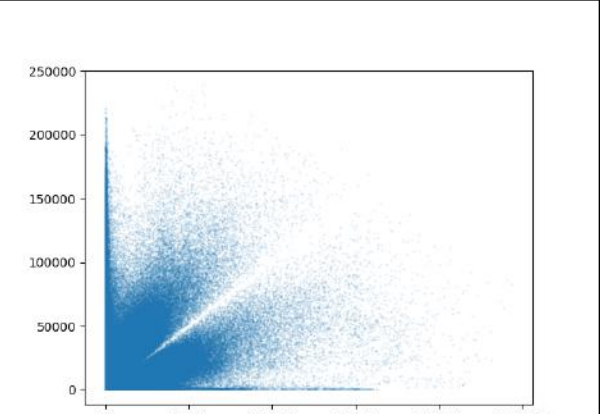
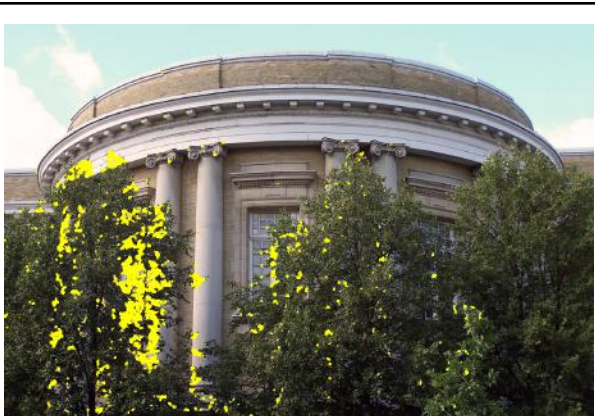
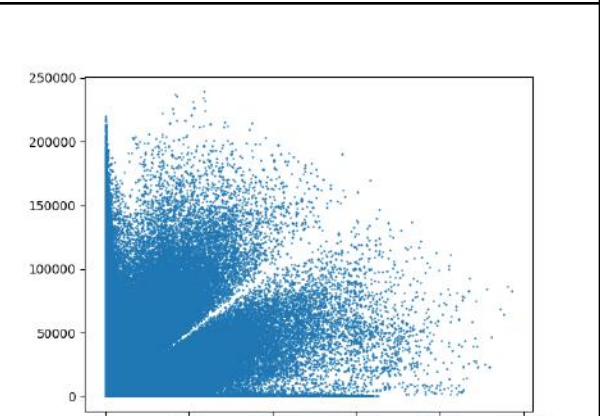
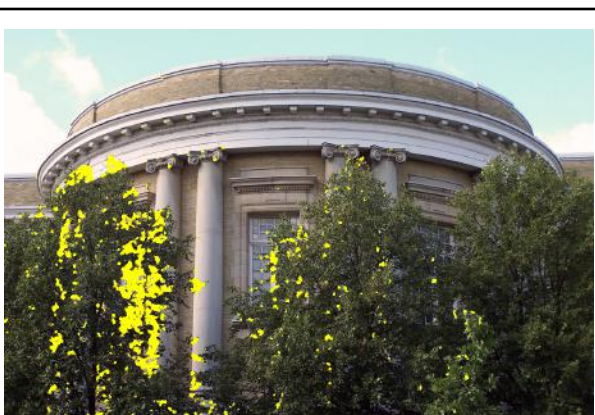
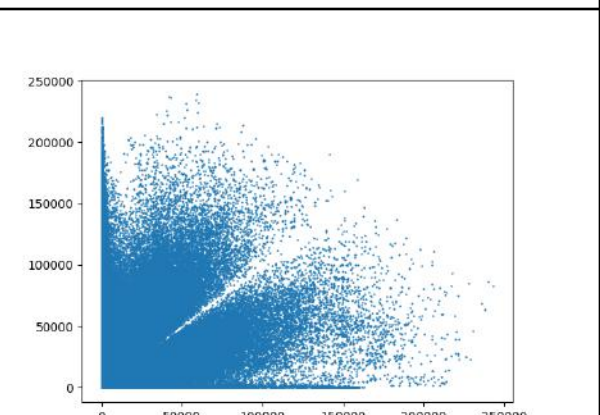


3



7



10		
20		
100		

From both of the images, the corners detected seem to have no significant difference. Comparing the corners detected for a sigma of 20 and sigma of 100 for both image 1 and image 2, the number and the position of corners are pretty much the same. However, if we compare the visualization results of corners detected by sigma = 0.5 and sigma=100, we can see that the dots that represent the corners are bigger with a larger sigma, which makes the corners more easily visible.

Moreover, the range of the eigenvalues also decreases as σ increases up to 7. After σ reaches 7, no matter how much we increase the σ , the range of the eigenvalues remains the same.