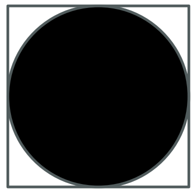
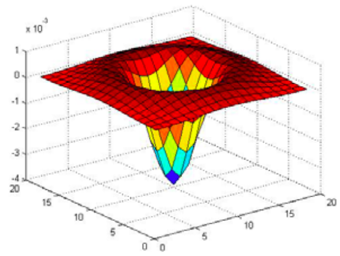
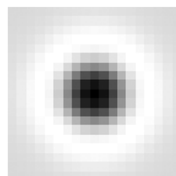
Part I

Q1.1:

D







The graph has 2 parts. One part below 0 and one part above 0.

Since pixel value for black is 0, and in order to achieve extrema, the black circle need to convolve with all the negative portion of Laplacian so that the sum would be largest. (Because the negative portion is at the center of filter)

That means 0 of LoG has to occur at edge of the circle

$$\nabla^2 G(x, y, \sigma) = 0 \Rightarrow \frac{x^2+y^2}{2\sigma^2} - 1 = 0$$

$$x^2+y^2 = 2\sigma^2$$

$$r^2 = 2\sigma^2$$

$$r = \sqrt{2}\sigma$$

$$\sqrt{2}\sigma = \frac{1}{2}D$$

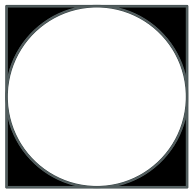
$$\sigma = \frac{1}{2\sqrt{2}}D$$

so the scale should be $\frac{1}{2\sqrt{2}}D$

Part I

1.2

D



Similar to 1.1, the white circle is at the negative portion of the filter $\Rightarrow$ We need to achieve a minimum.

In order to achieve minimum, circle need to just cover the negative portion of the circle.

That means the 0 of LoG occurs at the edge of the circle.

So $\sigma = \frac{1}{2\sqrt{2}} D$

Part I 2.1

$$\lambda_1 \lambda_2 = \det(N) = I_x^2 I_y^2 - I_x I_y I_x I_y = 0$$

$$\lambda_1 + \lambda_2 = \mathrm{trace}(N) = I_x^2 + I_y^2$$

$$\Rightarrow \quad \begin{cases} \lambda_1 = 0 \\ \lambda_2 = I_x^2 + I_y^2 \end{cases}$$

Part I 2.2

Since $\lambda_1 = 0 \geq 0$, $\lambda_2 = I_x^2 + I_y^2 \geq 0$

So N is positive semi-definite

$w(x, y)$ N is positive semi-definite

Since $w(x, y)$ is constant

Need to show the sum of positive semi-definite is also

positive semi-definite.

Let $v \in \mathbb{R}^2$, A . B be 2 positive semi-definite matrixs

$v^T (A+B) v = v^T A v + v^T B v$

$v^T A v \geq 0$, $v^T B v \geq 0$

$\Rightarrow v^T (A+B) v \geq 0$

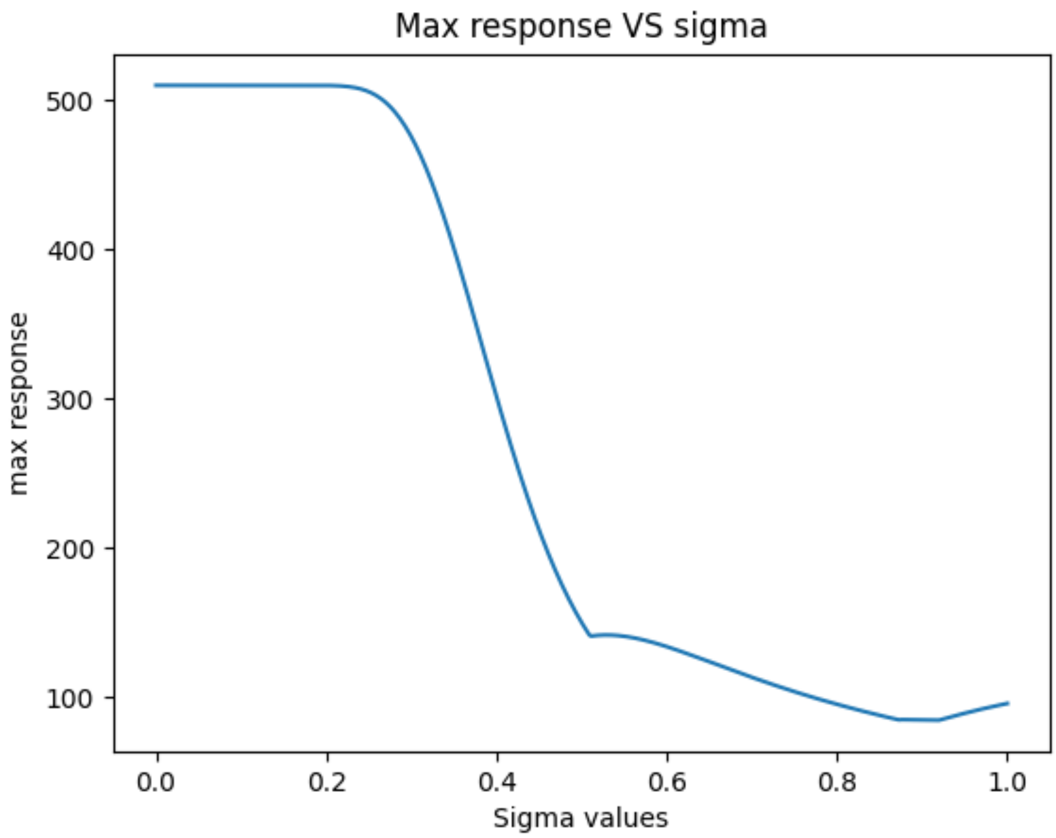
$\Rightarrow$ A + B is positive semi-definite.

Thus, the sum of positive semi-definite is also

positive semi-definite

Thus, M is positive semi-definite.

Part 1
Q1.3



I've created a 100x100 black square in the center of 200x200 white background. I've tried different values in log-domain and it turns out that the maximum occurs some where between 0 and 1. So I plot the sigma vs response graph and the maximum occurs at around 0.004

Part 2

Q3.1

```python
def retriving_mag_and_angle(image, t):
    # Calculate derivatives
    gx = cv2.Sobel(image, cv2.CV_32F, 1, 0)
    gy = cv2.Sobel(image, cv2.CV_32F, 0, 1)
    # Calculate the magnitude and angle for sourc.e image
    mag, angle = cv2.cartToPolar(gx, gy, angleInDegrees=True)
    # Convert direct angle to indirect angle with range of -15 ~ 165

    def convert_to_indirect_angle(x):
        if x >= 165 and x < 345:
            return x - 180
        elif x >= 345:
            return x - 360
        else:
            return x
    convert_to_indirect_angle = np.vectorize(convert_to_indirect_angle)
    angle = convert_to_indirect_angle(angle)
    # Threshold magnitude
    threshold = np.vectorize(lambda x: x if x >= t else 0)
    mag = threshold(mag)
    return mag, angle
```

```python
    # Threshold for each images are defined here
    THRESHOLD = [50, 70, 80, 60, 60]
```

I use cv2.cartToPolar to get the magnitude and the angle of the image. Thresholds are set empirically based on the result of HoG.

Q3.2

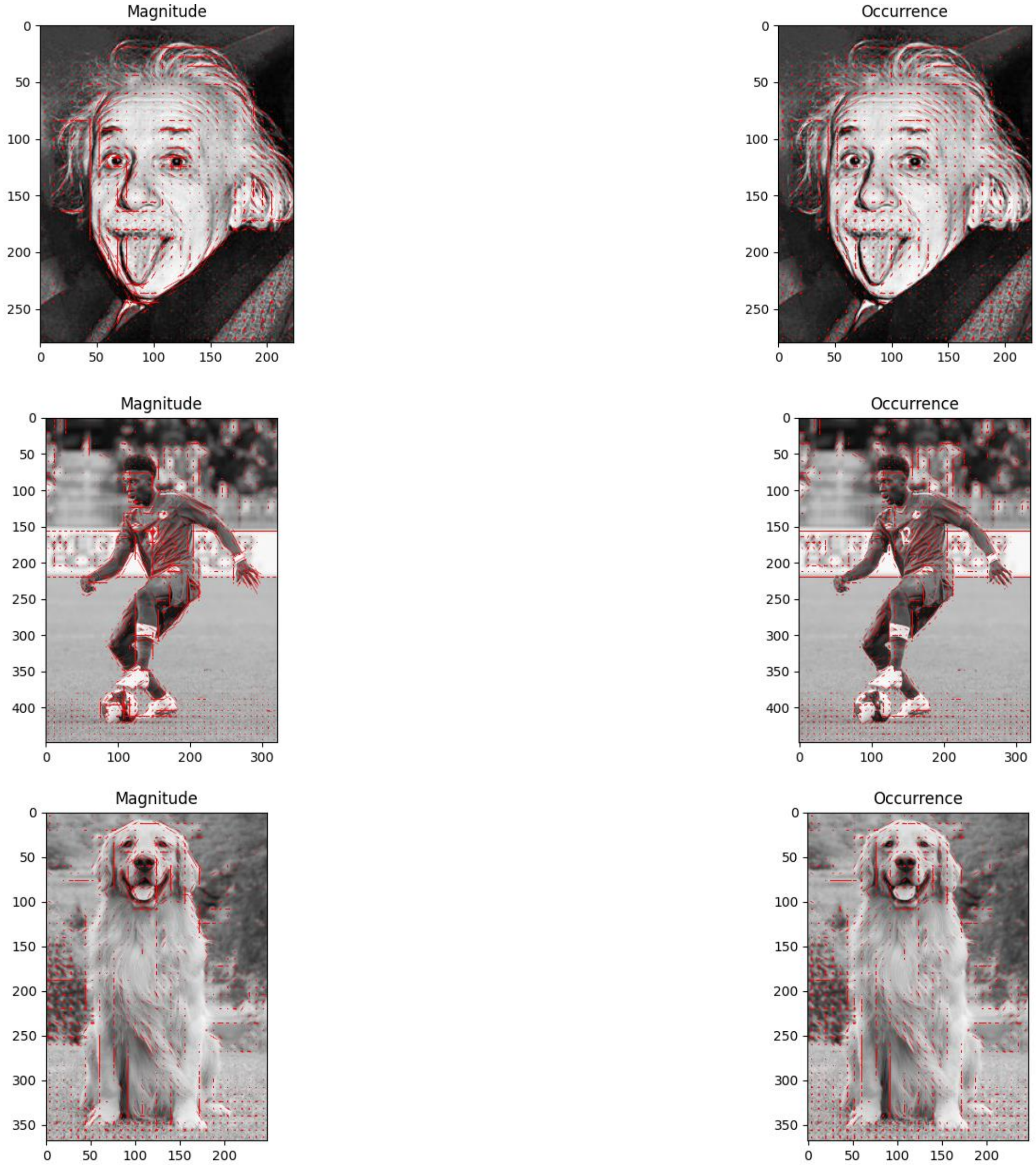
```python
def create_grid(image, size, mag, angle):
    # crop the image, gradient abd angle based on grid size
    m, n = image.shape
    new_m = m - m % size
    new_n = n - n % size
    image = image[:new_m, :new_n]
    mag = mag[:new_m, :new_n]
    angle = angle[:new_m, :new_n]
    return image, mag, angle, int(new_m / size), int(new_n / size)
```

This function is used to crop the image, magnitude and angle based on grid cell size.

Q3.3

```python
def process_cell(angle, mag, mode):
    # Define 6 if function for each bin
    bin_conditions = [(lambda i: lambda x: True if x >= -15 +
                       i * 30 and x < 15 + i * 30 else False)(i) for i in range(6)]
    # Define histogram list
    histogram = [0 for _ in range(6)]
    # Function for processing each pixel

    def process_pixel(x, y, mode):
        for m, bin_condition in enumerate(bin_conditions):
            if bin_condition(x):
                # Accumulate either magnitude or occurrence
                if mode == 'magnitude':
                    histogram[m] += y
                elif mode == 'occurrence' and y != 0:
                    histogram[m] += 1
                break
    process_pixel = np.vectorize(process_pixel, otypes=[])
    process_pixel(angle, mag, mode)
    return histogram
```

This function will process each grid cell. Each grid will have a descriptor with length 6. The value of either magnitude or occurrence will be accumulated based on the bin the angle belongs to. Also, if the magnitude does not pass the threshold, it will not be counted in occurrences as well.

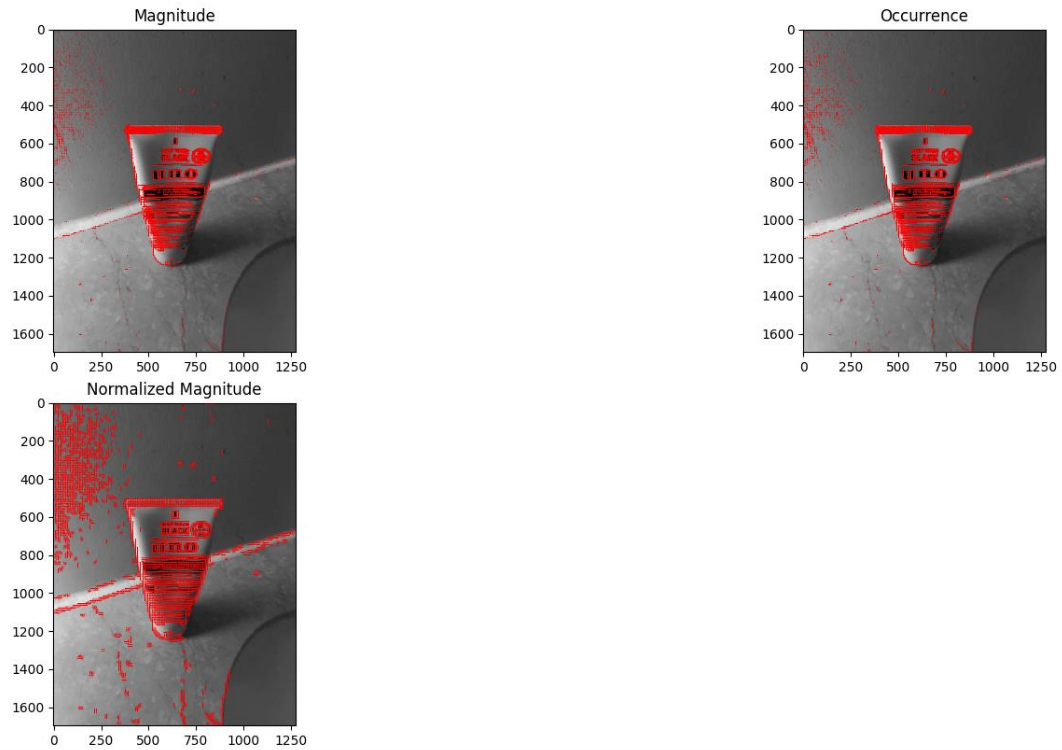


Above are the result images

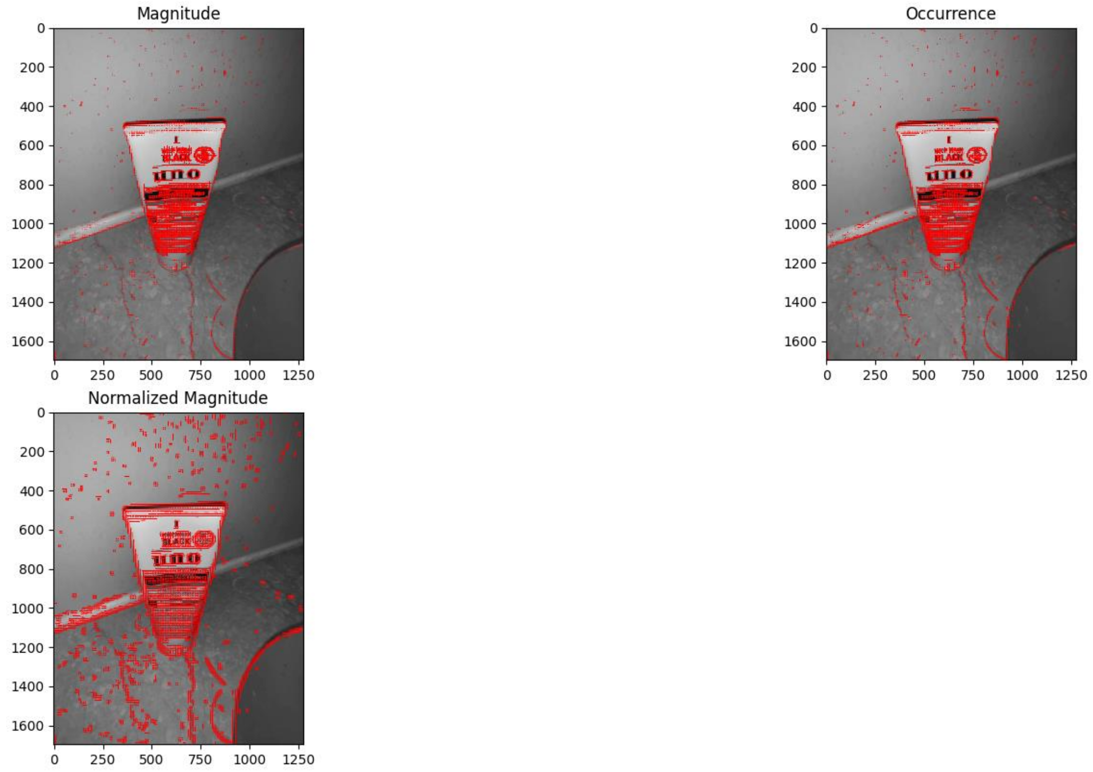
Q3.4

```python
def normalize_descriptor(descriptor):
    m, n, _ = descriptor.shape
    normalized_descriptor = np.zeros((m - 1, n - 1, 24))
    for i in range(m - 1):
        for j in range(n - 1):
            # putting 4 descriptors together into 24 x 1 vector
            result = np.concatenate([descriptor[i][j],
                                     descriptor[i][j + 1],
                                     descriptor[i + 1][j],
                                     descriptor[i + 1][j + 1]])
            # Normalize descriptor and store in matrix
            normalized_descriptor[i][j] = result / \
                np.sum(np.sqrt(np.square(result) + np.square(0.001)))
    return normalized_descriptor
```

The normalized descriptors stored as 1.txt, 2.txt … 5.txt. The above function is used to calculate the normalized descriptor. I choose to use magnitude descriptor to normalize.



This is the image that is taken without flashlight.



This is the image that is taken with flashlight. I choose to use visualization approach to compare normalized HoG. Basically, we have 24 entries in each descriptor. I split this 24 into 4 groups of 6 and taking the sum of these four vectors. Then, I draw this vector of length 6 the same way as I did for Q3.3.

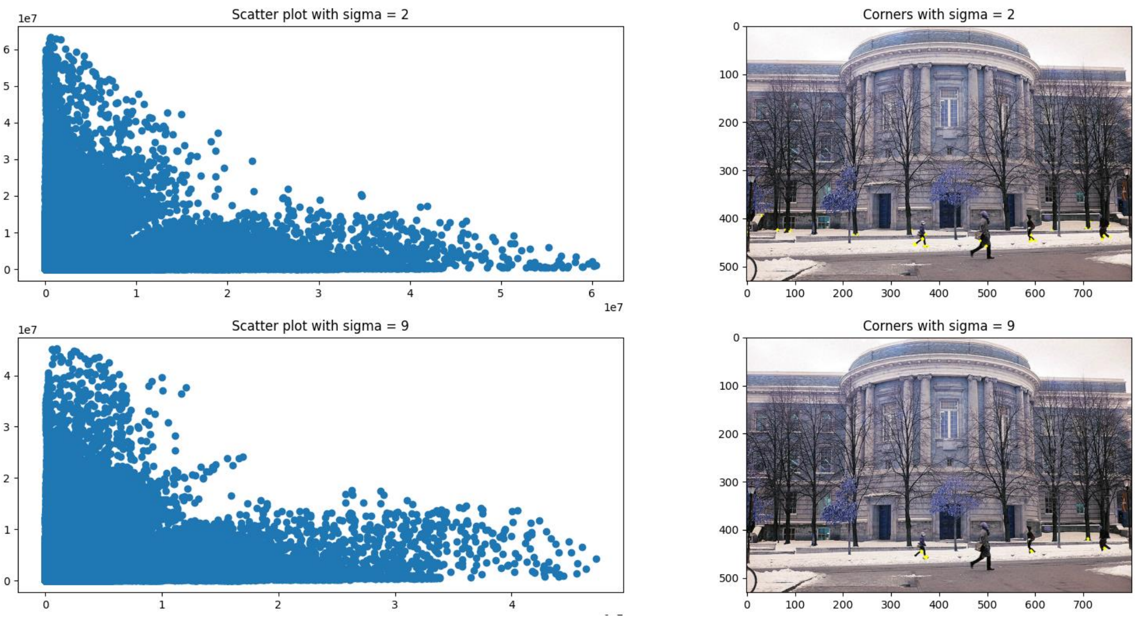
As we can see from the result, in Magnitude image, there are lots of extremely long lines. Normalizing can help us reduce the effect of those massive magnitude. Also, as you notice that there are lots of small dots in the background of magnitude image. Those are produced by illuminant. Even we do have a threshold, they still pass it. In Normalized magnitude image, it seems that the illuminant effect is stronger, however, the advantage is that if we clamp gradients for > 0.2 or 0.3 and renormalize it, those dots will be gone. So the normalization can help us reduce the effect of illuminant.

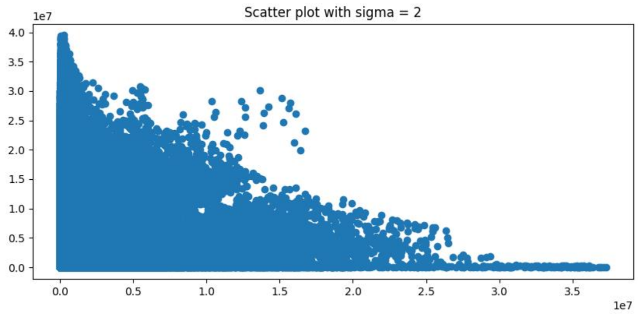
Q4

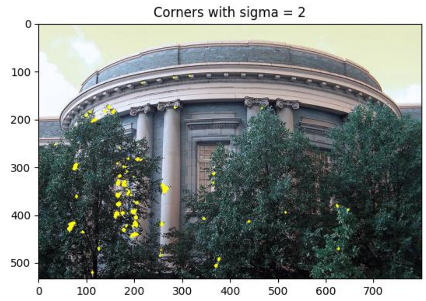
```python
def second_moment_matrix(image, sigma):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    m, n = gray.shape
    # Get Ix and Iy
    Ix = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=5)
    Iy = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=5)

    IxIy = np.multiply(Ix, Iy)
    Ix2 = np.multiply(Ix, Ix)
    Iy2 = np.multiply(Iy, Iy)

    # Kernel size can be defined here
    k_size = 11
    Ix2_blur = cv2.GaussianBlur(Ix2, (k_size, k_size), sigma)
    Iy2_blur = cv2.GaussianBlur(Iy2, (k_size, k_size), sigma)
    IxIy_blur = cv2.GaussianBlur(IxIy, (k_size, k_size), sigma)
    # Create Eigen value matrix
    eigen_values = np.zeros((m, n, 2))

    for i in range(m):
        for j in range(n):
            eigen_values[i][j] = LA.eigvals(np.array([[Ix2_blur[i][j], IxIy_blur[i][j]],
                                                      [IxIy_blur[i][j], Iy2_blur[i][j]]]))

    return eigen_values
```

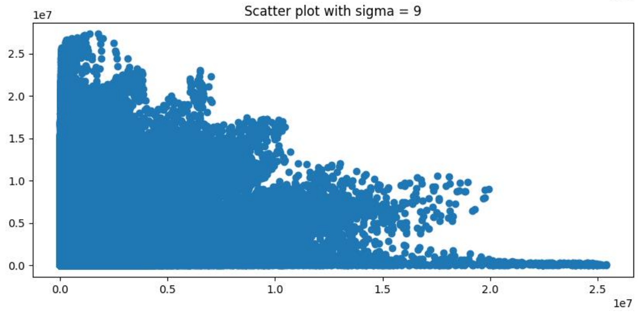
This is the function used to calculate the eigen values of the second moment matrix for each pixel.

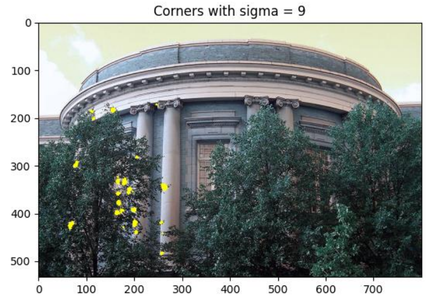
Here are results of two images











The thresholds for the two images are 13000000 and 8000000. As we can see that, if we use the same threshold values, less corner will be detected if sigma is larger. The reason is that, if we have larger sigma size in gaussian filter, it means the boundary will weight more, the center will weight less. The response becomes smaller when using a larger sigma so that less corner points are detected in image.