

Xiang Chen  
1004704992  
October 27, 2020

# CSC420 Report

Foe more

## **Part I: Theory (40 marks)**

Q1) (10 marks)

In this question, I use linear interpolation to upsample by a factor of 4, which is the following formula.

$$G(x) = \frac{x_2 - x}{x_2 - x_1} F(x_1) + \frac{x - x_1}{x_2 - x_1} F(x_2)$$

We have five points (-2.0,4), (-1.0,1),(0.0,5),(1.0,1),(2.0,4). We can simplify calculate the linear function of each line and put x = -1.75, -1.5, -1.25 to  $f_1$ , x = -0.75, -0.5, -0.25 to  $f_2$ , 0.25, 0.5, 0.75 to  $f_3$ , 1.25, 1.5, 1.75 to  $f_4$  and plot the value.

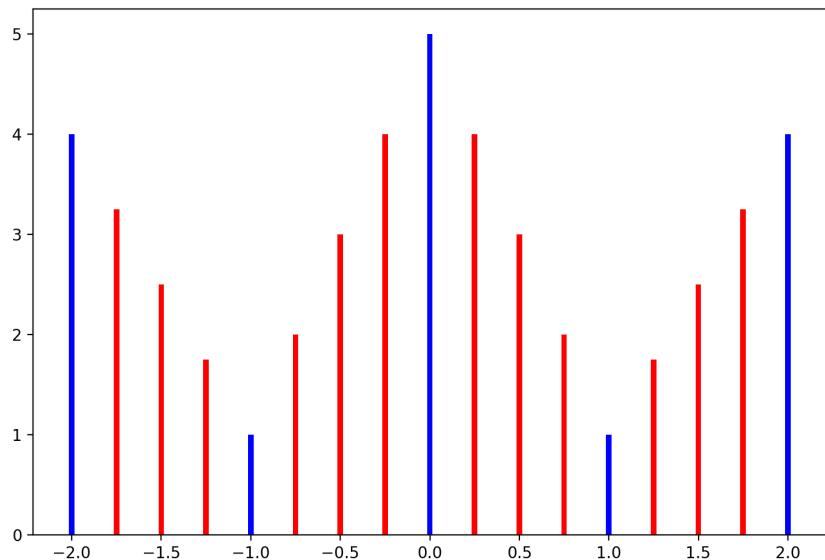
$$f_1 = -3x - 2$$

$$f_2 = 4x + 5$$

$$f_3 = -4x + 5$$

$$f_4 = 3x - 2$$

Here is the graph generated by python.



## Q2) (15 marks)

For this question, we first need to do some transformation for  $\sin^2(x)$ .

We know  $\sin^2(x) = \frac{1}{2}(1 - \cos(2x))$ .

Then, we can have  $\sin^2(2F_a\pi t) = \frac{1}{2}(1 - \cos(4F_a\pi t))$ .

We can have  $x(t) = \frac{1}{2}(1 - \cos(4F_a\pi t)) + \frac{1}{7}\cos(F_b\pi t) + \frac{1}{5}\sin(5F_c\pi t)$ .

We can remove the brackets.

Then, we have  $x(t) = \frac{1}{2} - \frac{1}{2}\cos(4F_a\pi t) + \frac{1}{7}\cos(F_b\pi t) + \frac{1}{5}\sin(5F_c\pi t)$ .

The max frequency for  $x(t)$  is  $\max(\frac{4F_a\pi}{2\pi}, \frac{F_b\pi}{2\pi}, \frac{5F_c\pi}{2\pi})$ .

By Nyqvist-Shannon sampling theorem, the sufficient sample-rate is  $2 * \max(2F_a, \frac{F_b}{2}, \frac{5F_c}{2})$ .

## Q3) (15 marks)

1.

We know  $N$  is a  $2*2$  Matrix. Therefore, we can use the formula for eigenvalue of  $2*2$

matrix, which is  $\lambda_{\pm} = \frac{1}{2}[(a_{11} + a_{22}) \pm \sqrt{4a_{12}a_{21} - a_{12}a_{21}}]$  (0.3.1).

We have  $N = \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix}$ , which  $a_{11} = I_x^2$ ,  $a_{12} = I_x I_y$ ,  $a_{21} = I_x I_y$ ,  $a_{22} = I_y^2$ .

When we put them in to (0.3.1), we get  $\lambda_1 = 0$  and  $\lambda_2 = I_x^2 + I_y^2$ .

Therefore, the second value of  $\lambda$  is  $I_x^2 + I_y^2$ .

2.

We know  $N = V \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} V^{-1}$ , and  $M = \sum_x \sum_y w(x, y) * (V \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} V^{-1})$  from lecture.

By the property of linear system, we can make some form transition.

Then, we can get  $M = \sum_x \sum_y V \begin{pmatrix} w(x, y) * \lambda_1 & 0 \\ 0 & w(x, y) * \lambda_2 \end{pmatrix} V^{-1}$ .

Since we have assumed  $N$  is a positive semi-definite, we can know  $\lambda_1 \geq 0$  and  $\lambda_2 \geq 0$ .

Since  $w(x, y) > 0$  (Gaussian Matrix), we know  $V \begin{pmatrix} w(x, y)^* \lambda_1 & 0 \\ 0 & w(x, y)^* \lambda_2 \end{pmatrix} V^{-1}$  is also positive semi-definite.

Therefore, we can know M is also positive semi-definite since it is a sum of many matrixes which are positive semi-definite.

We can follow the intuition to prove the matrix of sum of many matrixes which are positive semi-definite is also positive semi-definite. Let assume any two matrix X and Y.

We can have  $V^T X V > 0$  and  $V^T Y V > 0$ , then we can have  $V^T X V + V^T Y V > 0$ , which is  $V^T(X + Y)V > 0$ .

## Part II: Image Resizing with Seam Carving (75 marks)

1.

```
23 def Sobel_Operation(src):
24     # This function is only for grey scale image. #
25     # Please use getGreyImage first, if input is a RGB image. #
26     sobel_x = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
27     sobel_y = np.array([[1,-2,-1],[0,0,0],[1,2,1]])
28     x_length, y_length = src.shape
29     res = np.empty((x_length,y_length), dtype=float)
30     conv_x = cv.filter2D(src,-1,sobel_x)
31     conv_y = cv.filter2D(src,-1,sobel_y)
32     for i in range(x_length):
33         for j in range(y_length):
34             res[i][j] = math.sqrt(conv_x[i][j]**2+conv_y[i][j]**2)
35     return res
36
```

2. (45 marks)

```
44 def cutColoum(src):
45     i_length, j_length = src.shape
46     backtrack = np.zeros((i_length, j_length), dtype=float)
47     directions = np.zeros((i_length, j_length), dtype=int)
48     gradients = Sobel_Operation(src)
49     for j in range(j_length): backtrack[i_length-1][j] = src[i_length-1][j]
50     for i in reversed(range(i_length-1)):
51         for j in range(j_length):
52             if not j:
53                 index = np.argmin(backtrack[i+1, j:j + 2])
54                 backtrack[i][j] = gradients[i][j] + backtrack[i+1][j+index]
55                 directions[i][j] = index
56             elif j == j_length - 1:
57                 index = np.argmin(backtrack[i+1, j-1:j+1])
58                 backtrack[i][j] = gradients[i][j] + backtrack[i+1][j-1+index]
59                 directions[i][j] = index - 1
60             else:
61                 index = np.argmin(backtrack[i+1, j-1:j + 2])
62                 backtrack[i][j] = gradients[i][j] + backtrack[i+1][j-1+index]
63                 directions[i][j] = index - 1
64     binary_matrix = np.ones((i_length, j_length), dtype=int)
65     min_j = np.argmin(backtrack[0])
66     for i in range(i_length):
67         binary_matrix[i][min_j] = 0
68         min_j = min_j + directions[i][min_j]
69     return binary_matrix
```

3. (10 marks)

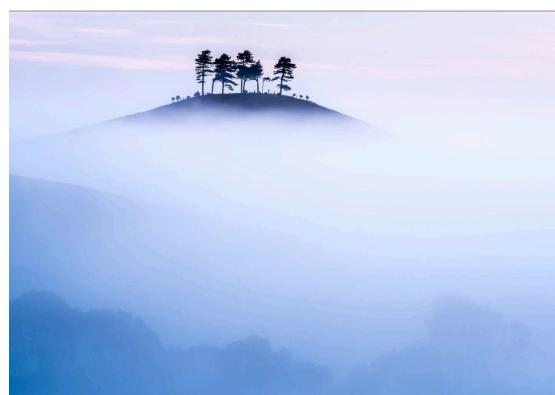
```
77 def mainCol(src,new_col):
78     row, col, _ = src.shape
79     new_image = src.copy()
80     if(new_col>=col): return src
81     for k in range(col-new_col):
82         binary_matrix = cutColoum(getGreyImge(new_image))
83         i_length,j_length = binary_matrix.shape
84         buff = new_image.copy()
85         new_image = np.zeros((i_length,j_length-1,_),dtype=np.uint8)
86         for i in range(i_length):
87             new_j = 0
88             for j in range(j_length):
89                 if binary_matrix[i][j]:
90                     new_image[i][new_j][0] = buff[i][j][0]
91                     new_image[i][new_j][1] = buff[i][j][1]
92                     new_image[i][new_j][2] = buff[i][j][2]
93                     new_j += 1
94     return new_image
```

4. (10 marks)

```
68
69 def cutRow(src):
70     i_length, j_length = src.shape
71     rotate = np.zeros((j_length, i_length), dtype=float)
72     for i in range(i_length):
73         for j in range(j_length):
74             rotate[j_length-j-1][i] = src[i][j]
75     return cutColoum(rotate)
```

5. (10 marks)







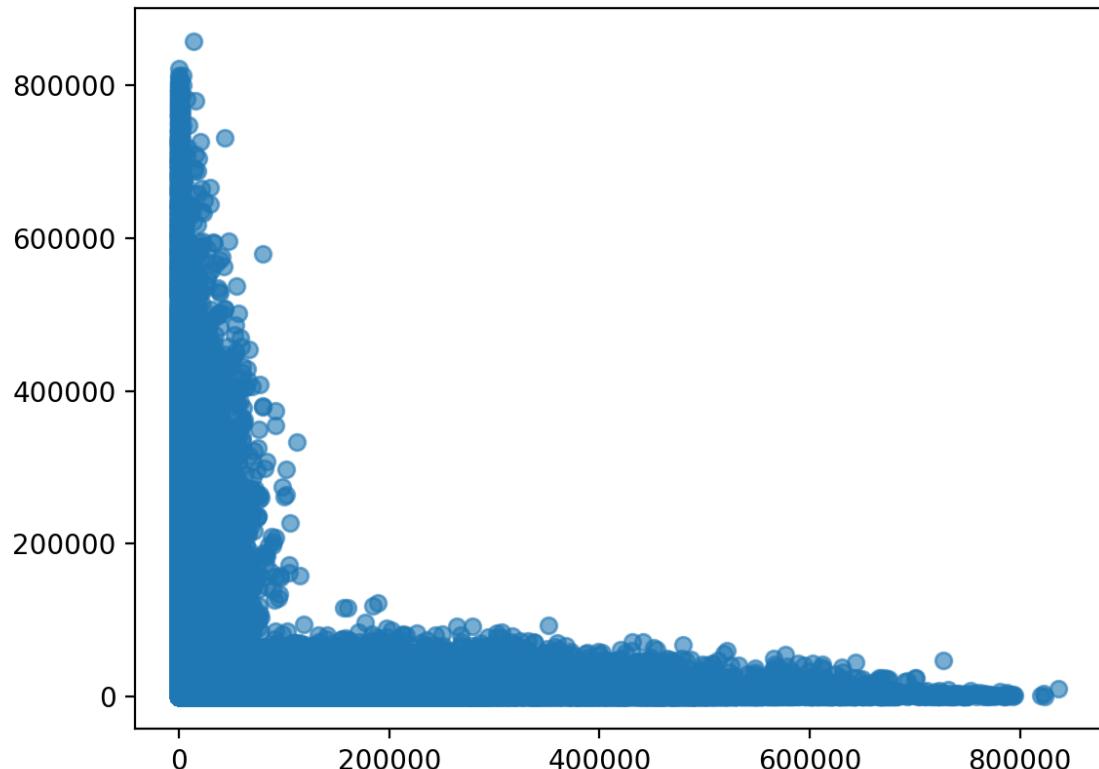
### Part III: Corner Detection (35 marks)

1.

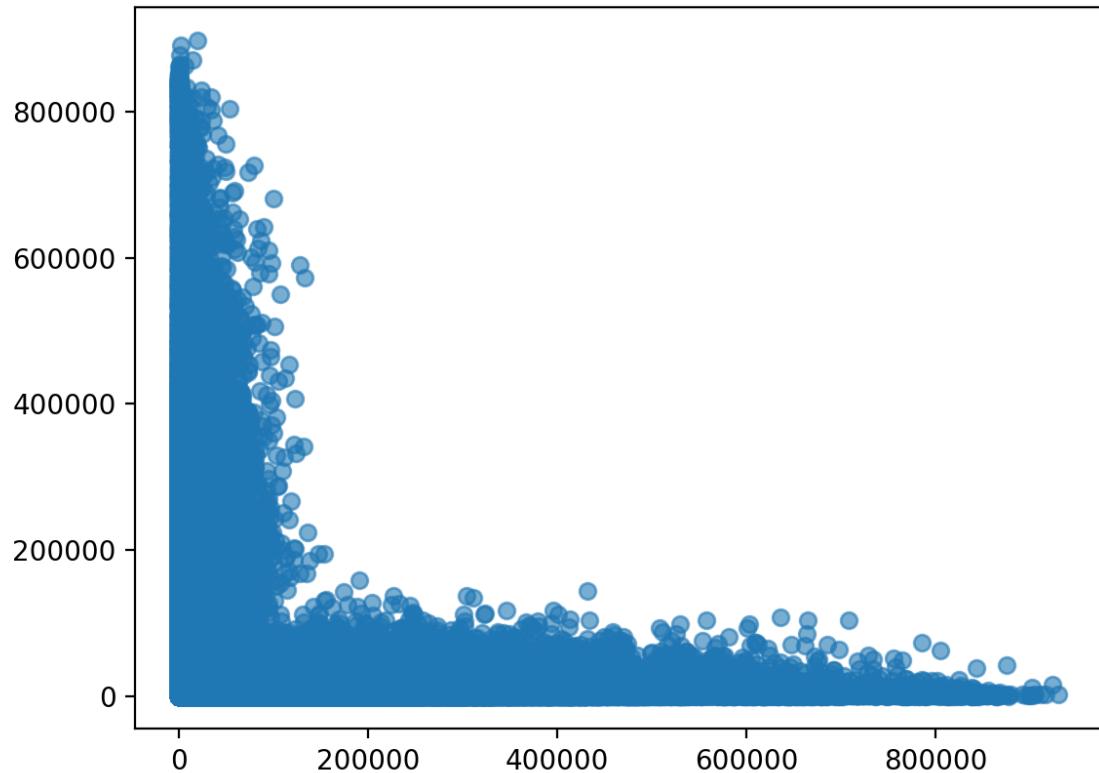
```
28 def eigenvalues(src,kernel_size):
29     sobel_x = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
30     sobel_y = np.array([[1,-2,-1],[0,0,0],[1,2,1]])
31     grad_x = cv.filter2D(src,-1,sobel_x)
32     grad_y = cv.filter2D(src,-1,sobel_y)
33     I_xy = grad_x * grad_y
34     I_x = grad_x * grad_x
35     I_y = grad_y * grad_y
36     window = Gaussian Blur(kernel_size/6,kernel_size)
37     I_x = cv.filter2D(I_x,-1,window)
38     I_y = cv.filter2D(I_y,-1,window)
39     I_xy = cv.filter2D(I_xy,-1,window)
40     x_length, y_length = src.shape
41     eigenvalues_1 = np.zeros((x_length,y_length),dtype=complex)
42     eigenvalues_2 = np.zeros((x_length,y_length),dtype=complex)
43     for i in range(x_length):
44         for j in range(y_length):
45             M = np.array([[I_x[i][j],I_xy[i][j]],[I_xy[i][j],I_y[i][j]]])
46             eigvals, eigvecs = la.eig(M)
47             # If you don't want me to use library, here is the alternative way to get eigenvalue of 2*2 Matrixx
48             # eigenvalues_1[i][j] = (1/2) * (M[0][0]+M[1][1] - math.sqrt(4*M[0][1]*M[1][0]+(M[0][0]-M[1][1])**2))
49             # eigenvalues_2[i][j] = (1/2) * (M[0][0]+M[1][1] + math.sqrt(4*M[0][1]*M[1][0]+(M[0][0]-M[1][1])**2))
50             eigenvalues_1[i][j] = eigvals[0]
51             eigenvalues_2[i][j] = eigvals[1]
52     return eigenvalues_1, eigenvalues_2
```

2.

This is the scatterplot for  $I_1(\text{ex4.jpg})$ .



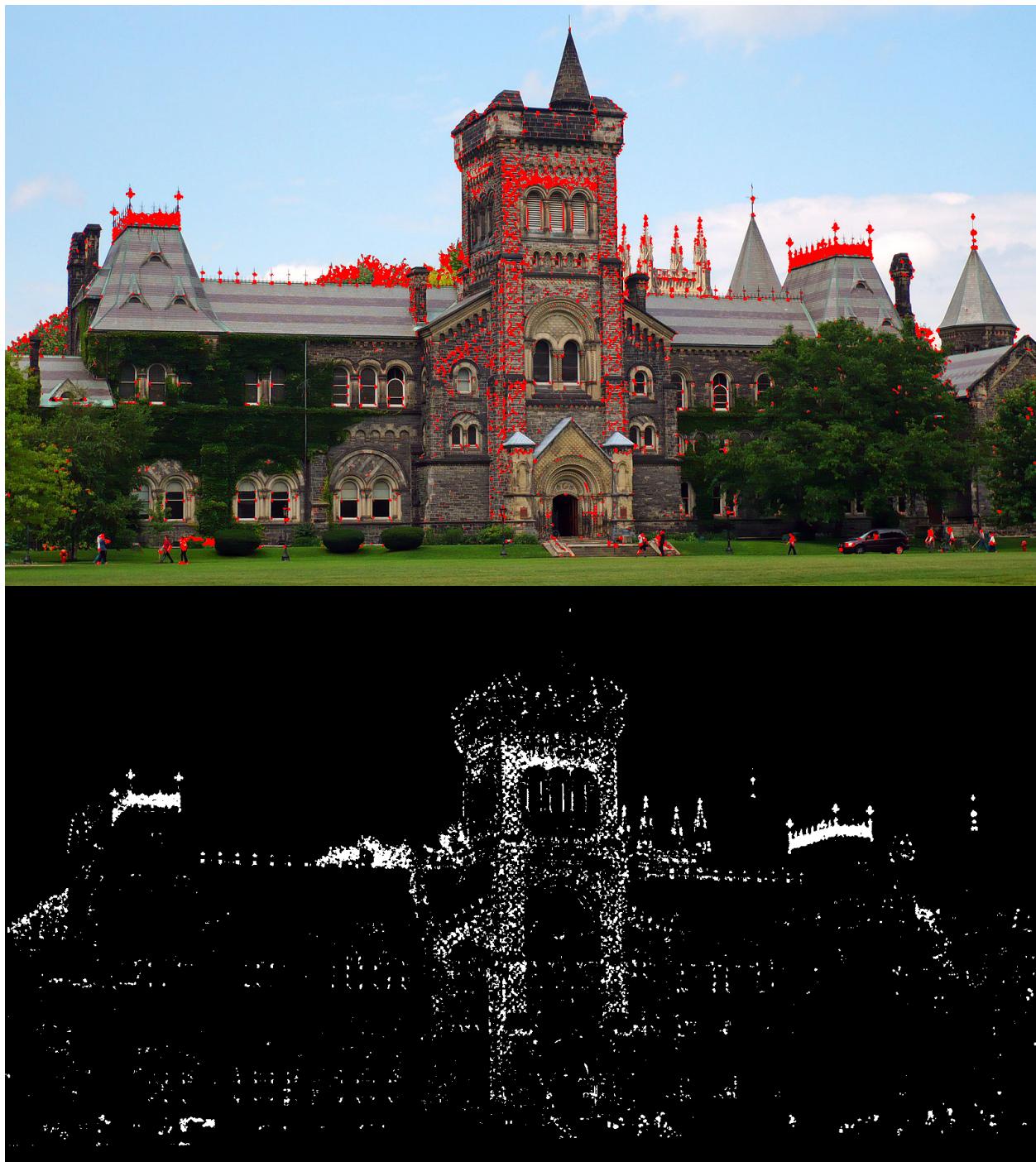
This is the scatterplot for  $I_2(\text{ex5.jpg})$ .



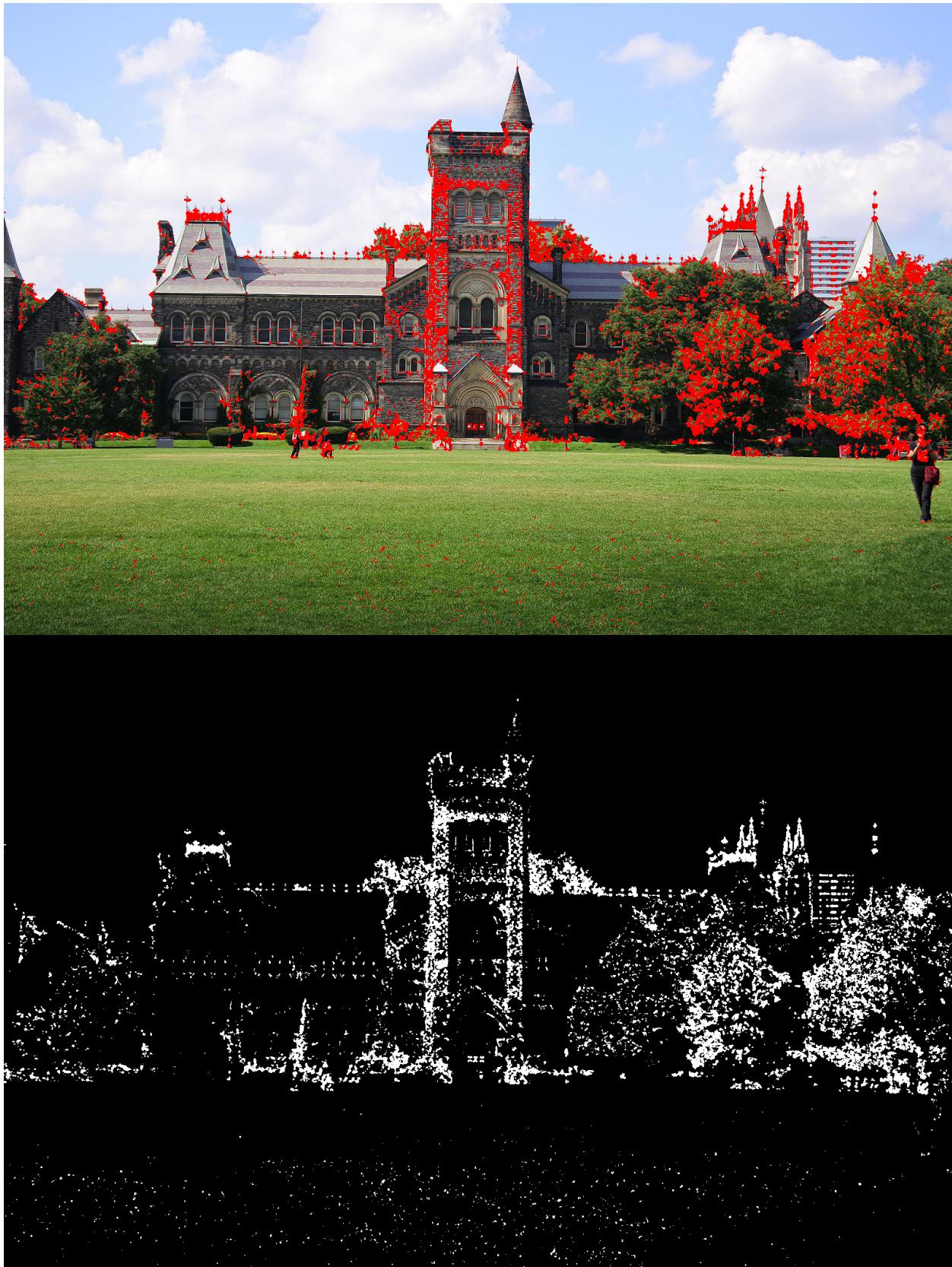
```
83 def scatterplot(src):
84     eigenvalues_1, eigenvalues_2 = eigenvalues(src, 3)
85     plt.scatter(eigenvalues_1, eigenvalues_2, alpha=0.6)
86     plt.show()
```

3.

I choose threshold value of 18888 for  $I_1$ .



I choose threshold value of 18888 for  $I_2$ .



4.

When I Choose a different value for kernel size, I find that when kernel size is bigger, the corners in the picture are bigger. This makes intuition since when we apply larger Gauss filter, we make the energy of the picture more uniform. Therefore, the pixels around around the corner pixel will have value closer to the corner, and it has a wave effect. Therefore, when the Gauss filter is bigger, we will have larger corners just like the following binary images. In this situation, we actually lose the really corners.

