CAP 4453
PA-1 Discussion

Aayush Rana

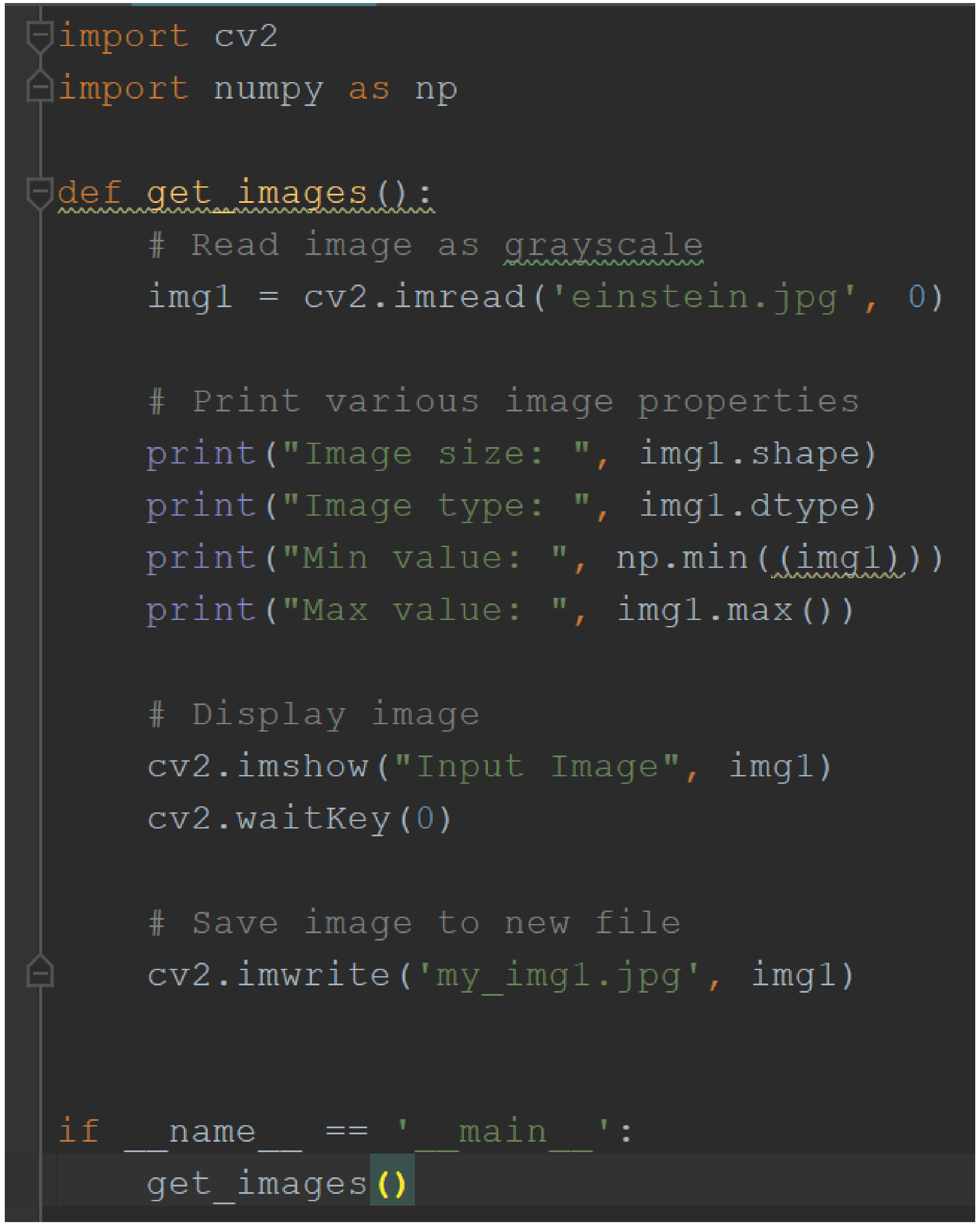
2/17/2021

OpenCV Basic

- Image operations (read, write, show)
- Edit images

OpenCV Basic

- Image operations (read, write, show)

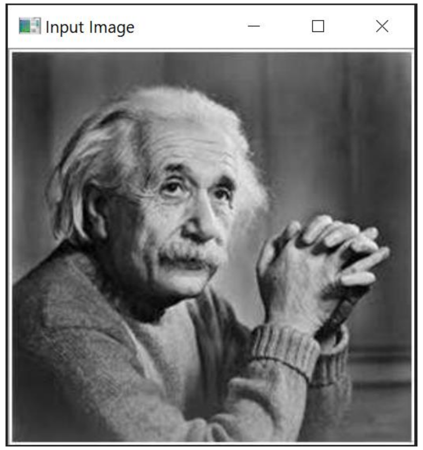


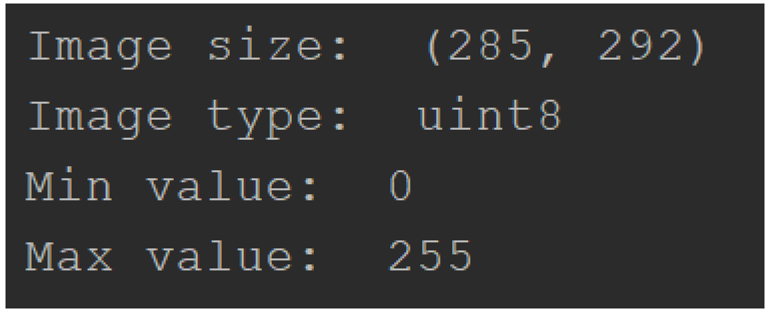
```python
import cv2
import numpy as np

def get_images():
    # Read image as grayscale
    img1 = cv2.imread('einstein.jpg', 0)

    # Print various image properties
    print("Image size: ", img1.shape)
    print("Image type: ", img1.dtype)
    print("Min value: ", np.min((img1)))
    print("Max value: ", img1.max())

    # Display image
    cv2.imshow("Input Image", img1)
    cv2.waitKey(0)

    # Save image to new file
    cv2.imwrite('my_img1.jpg', img1)


if __name__ == '__main__':
    get_images()
```

OpenCV Basic

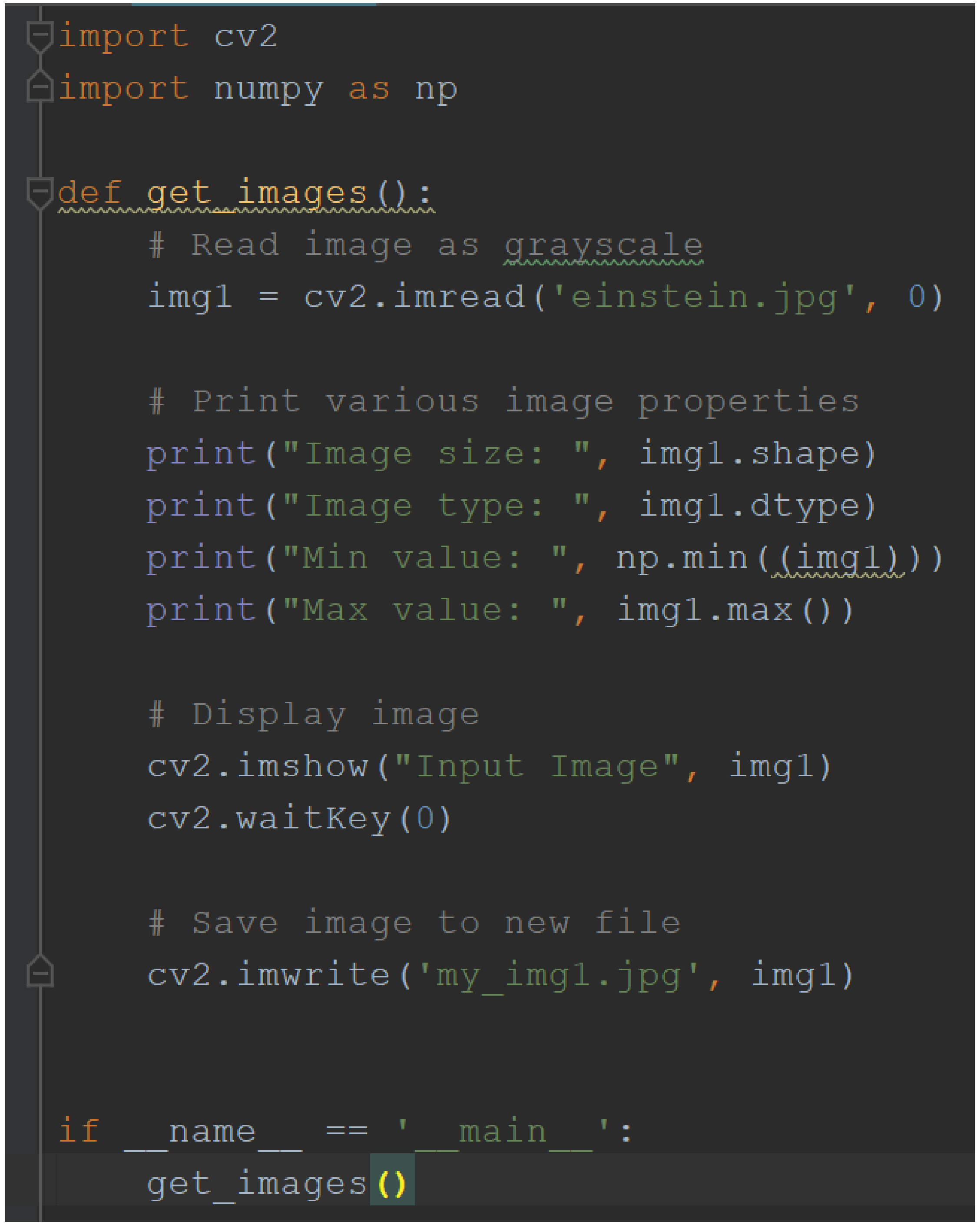
- Image operations (read, write, show)

Output





```
Image size:  (285, 292)
Image type:  uint8
Min value:  0
Max value:  255
```

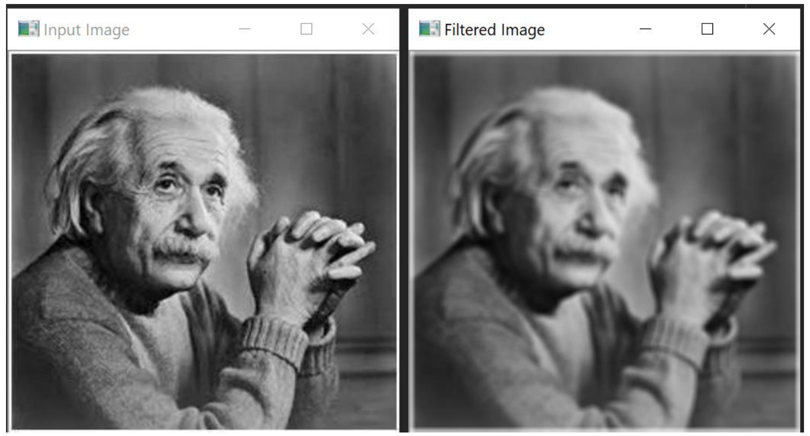


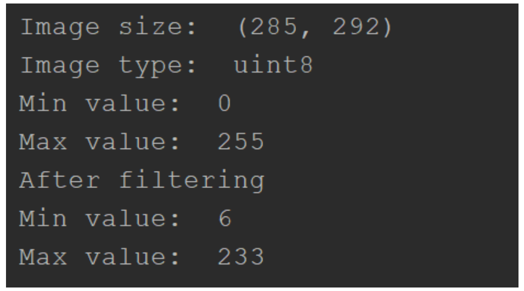
```
import cv2
import numpy as np

def get_images():
    # Read image as grayscale
    img1 = cv2.imread('einstein.jpg', 0)

    # Print various image properties
    print("Image size: ", img1.shape)
    print("Image type: ", img1.dtype)
    print("Min value: ", np.min((img1)))
    print("Max value: ", img1.max())

    # Display image
    cv2.imshow("Input Image", img1)
    cv2.waitKey(0)

    # Save image to new file
    cv2.imwrite('my_img1.jpg', img1)


if __name__ == '__main__':
    get_images()
```

OpenCV Basic

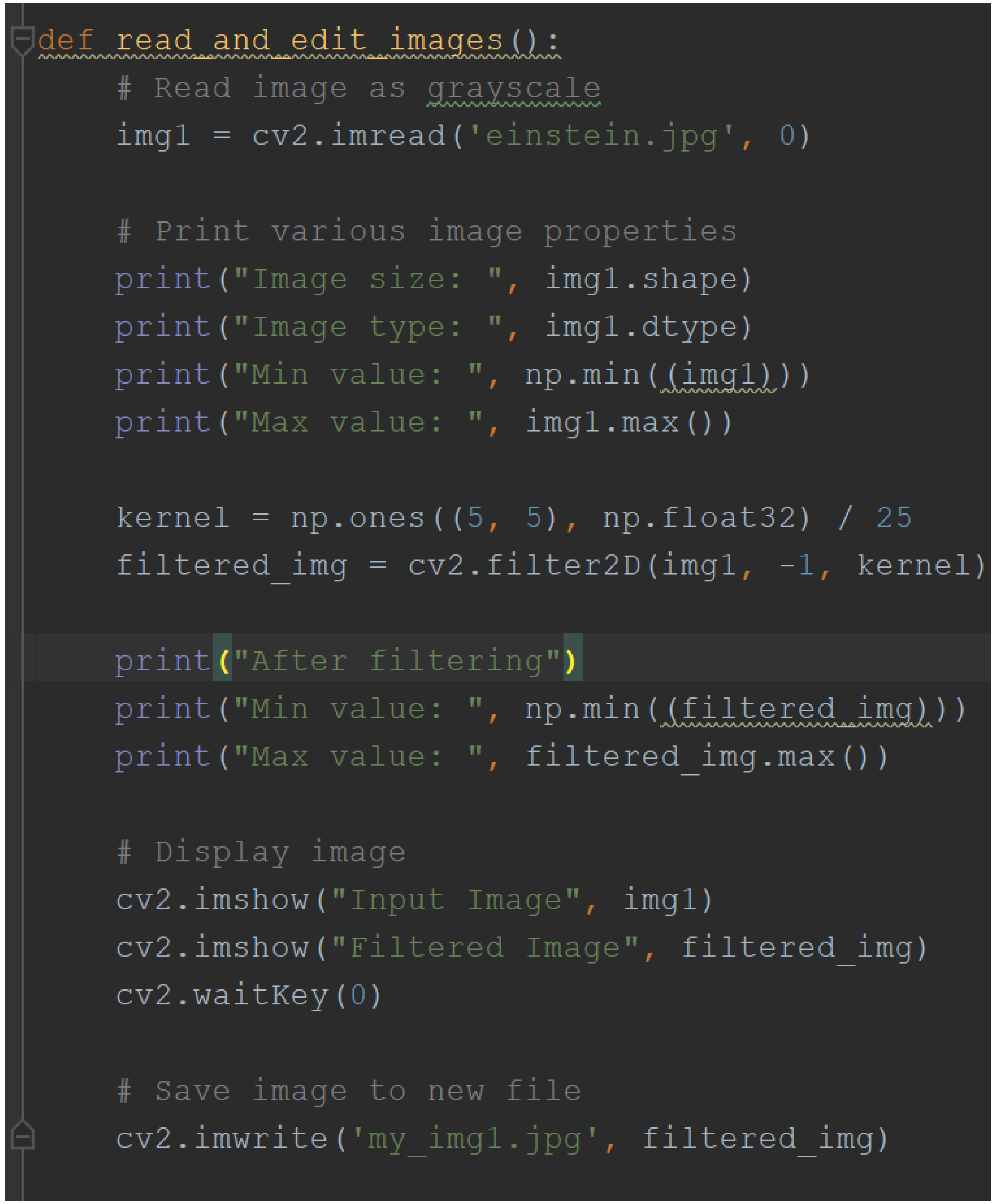
- Image operations (read, write, show)
- Edit image

Output



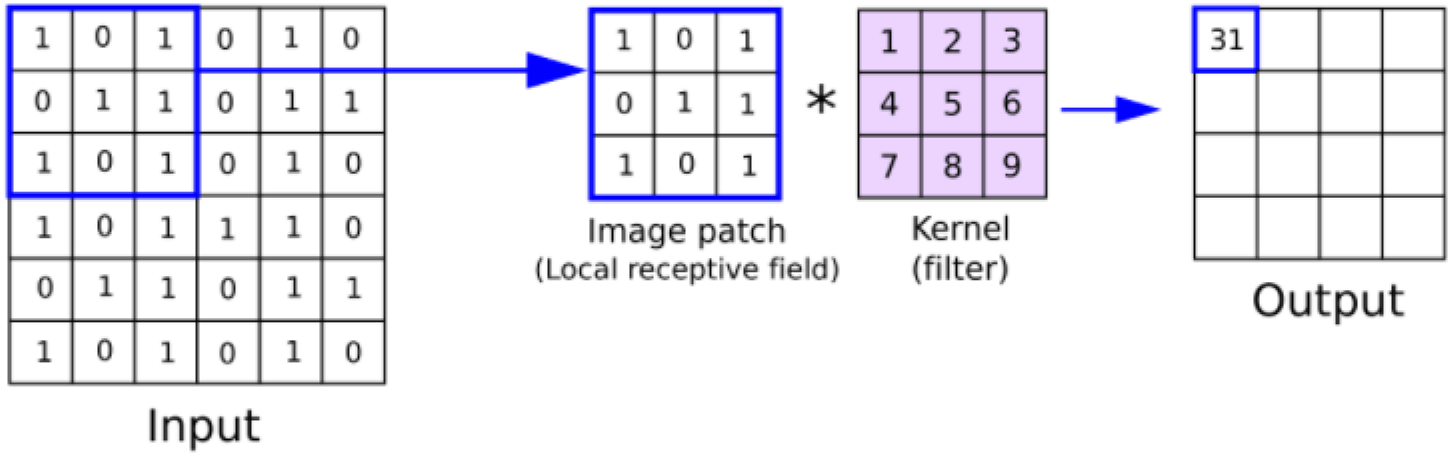


```
Image size:  (285, 292)
Image type:  uint8
Min value:  0
Max value:  255
After filtering
Min value:  6
Max value:  233
```



```python
def read_and_edit_images():
    # Read image as grayscale
    img1 = cv2.imread('einstein.jpg', 0)

    # Print various image properties
    print("Image size: ", img1.shape)
    print("Image type: ", img1.dtype)
    print("Min value: ", np.min((img1)))
    print("Max value: ", img1.max())

    kernel = np.ones((5, 5), np.float32) / 25
    filtered_img = cv2.filter2D(img1, -1, kernel)

    print("After filtering")
    print("Min value: ", np.min((filtered_img)))
    print("Max value: ", filtered_img.max())

    # Display image
    cv2.imshow("Input Image", img1)
    cv2.imshow("Filtered Image", filtered_img)
    cv2.waitKey(0)

    # Save image to new file
    cv2.imwrite('my_img1.jpg', filtered_img)
```

Filtering array



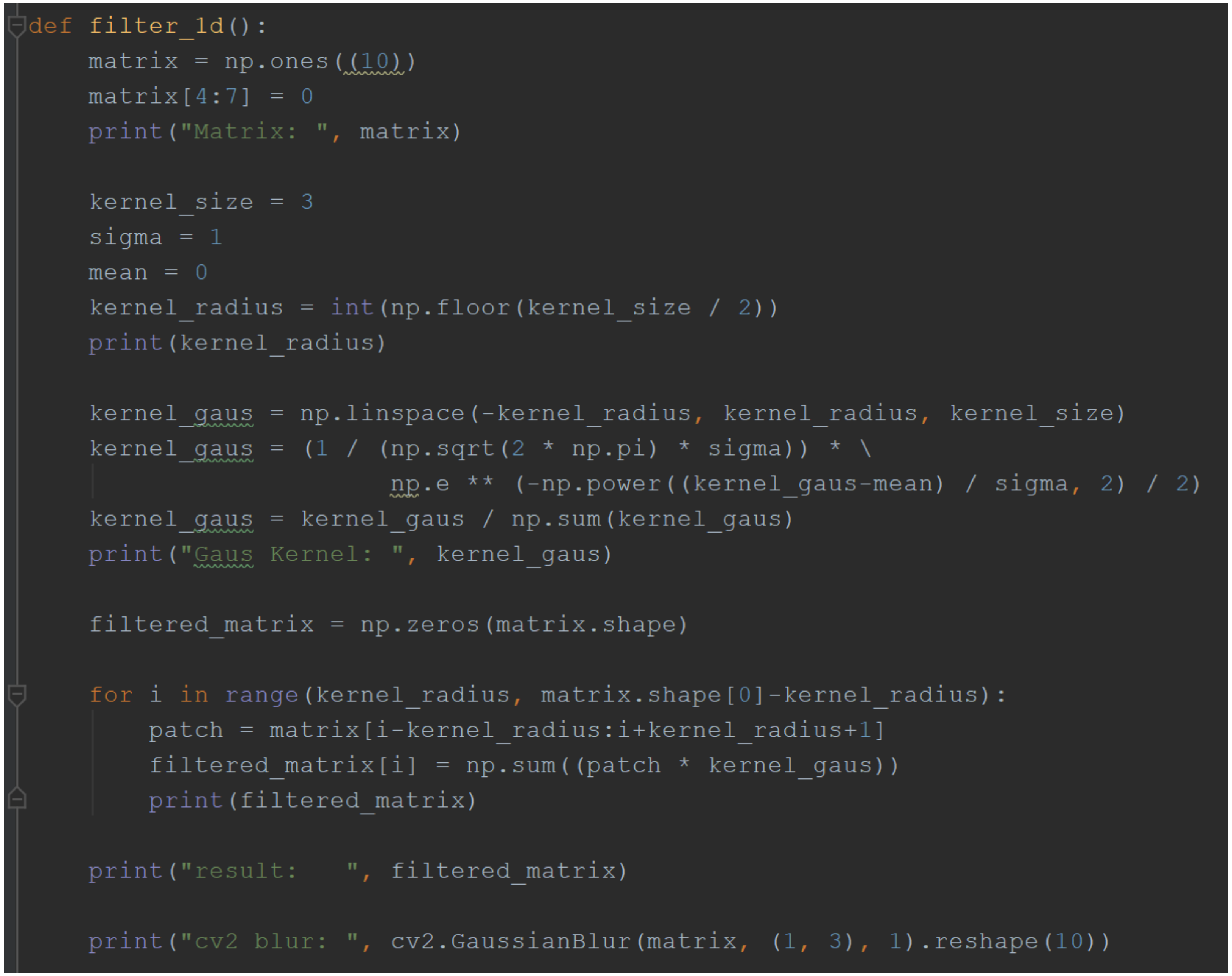
(1*1) + (0*2) + (1*3) + (0*4) + (1*5) + (1*6) + (1*7) + (0*8) + (1*9) = 31

This uses correlation formula: $F \circ I(x,y) = \sum_{j=-N}^{N} \sum_{i=-N}^{N} F(i,j) I(x+i, y+j)$

For convolution, the kernel is rotated by 180 degrees

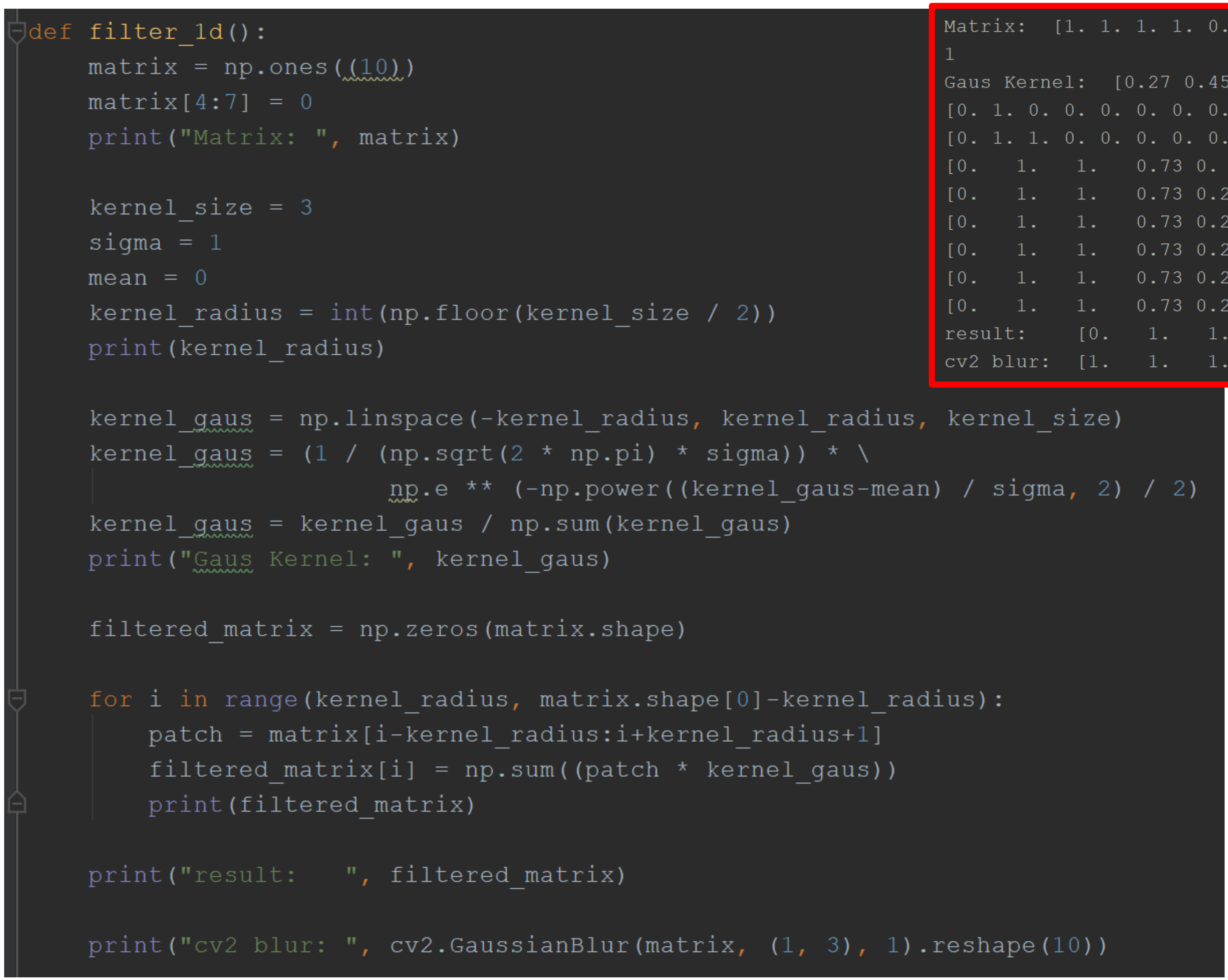
Image credit: Anh H. Reynolds https://anhreynolds.com/blogs/cnn.html

Filtering array – 1D

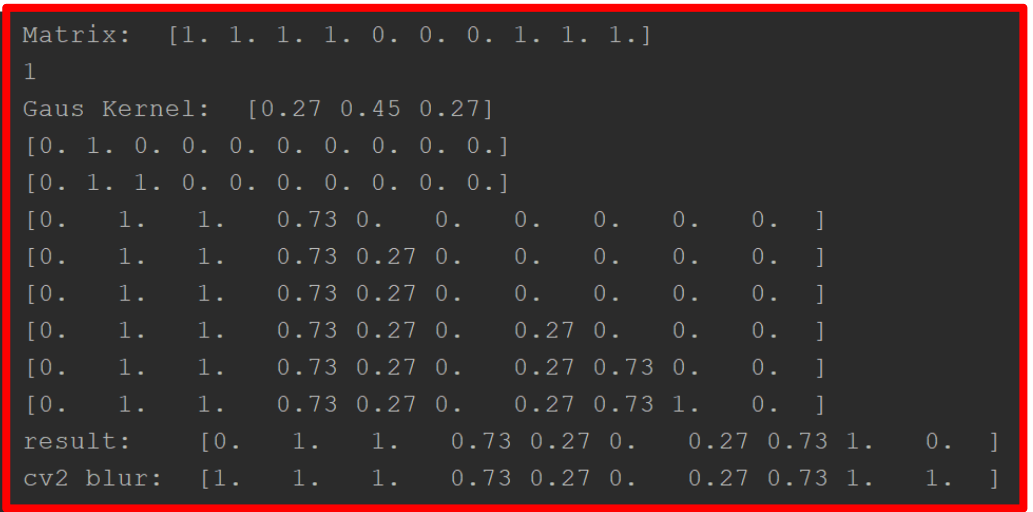


```
def filter_1d():
    matrix = np.ones((10))
    matrix[4:7] = 0
    print("Matrix: ", matrix)

    kernel_size = 3
    sigma = 1
    mean = 0
    kernel_radius = int(np.floor(kernel_size / 2))
    print(kernel_radius)

    kernel_gaus = np.linspace(-kernel_radius, kernel_radius, kernel_size)
    kernel_gaus = (1 / (np.sqrt(2 * np.pi) * sigma)) * \
                    np.e ** (-np.power((kernel_gaus-mean) / sigma, 2) / 2)
    kernel_gaus = kernel_gaus / np.sum(kernel_gaus)
    print("Gaus Kernel: ", kernel_gaus)

    filtered_matrix = np.zeros(matrix.shape)

    for i in range(kernel_radius, matrix.shape[0]-kernel_radius):
        patch = matrix[i-kernel_radius:i+kernel_radius+1]
        filtered_matrix[i] = np.sum((patch * kernel_gaus))
        print(filtered_matrix)

    print("result:   ", filtered_matrix)

    print("cv2 blur: ", cv2.GaussianBlur(matrix, (1, 3), 1).reshape(10))
```

Filtering array – 1D



```python
def filter_1d():
    matrix = np.ones((10))
    matrix[4:7] = 0
    print("Matrix: ", matrix)

    kernel_size = 3
    sigma = 1
    mean = 0
    kernel_radius = int(np.floor(kernel_size / 2))
    print(kernel_radius)

    kernel_gaus = np.linspace(-kernel_radius, kernel_radius, kernel_size)
    kernel_gaus = (1 / (np.sqrt(2 * np.pi) * sigma)) * \
                  np.e ** (-np.power((kernel_gaus-mean) / sigma, 2) / 2)
    kernel_gaus = kernel_gaus / np.sum(kernel_gaus)
    print("Gaus Kernel: ", kernel_gaus)

    filtered_matrix = np.zeros(matrix.shape)

    for i in range(kernel_radius, matrix.shape[0]-kernel_radius):
        patch = matrix[i-kernel_radius:i+kernel_radius+1]
        filtered_matrix[i] = np.sum((patch * kernel_gaus))
        print(filtered_matrix)

    print("result:   ", filtered_matrix)

    print("cv2 blur: ", cv2.GaussianBlur(matrix, (1, 3), 1).reshape(10))
```



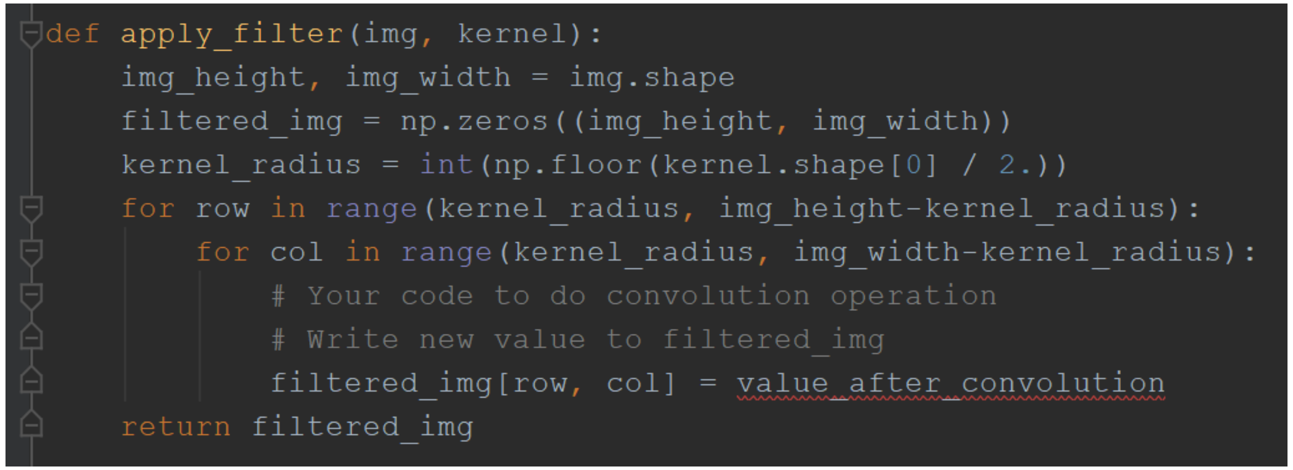
```
Matrix:  [1. 1. 1. 1. 0. 0. 0. 1. 1. 1.]
1
Gaus Kernel:  [0.27 0.45 0.27]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 1. 1. 0. 0. 0. 0. 0. 0. 0.]
[0.   1.   1.   0.73 0.   0.   0.   0.   0.   0.  ]
[0.   1.   1.   0.73 0.27 0.   0.   0.   0.   0.  ]
[0.   1.   1.   0.73 0.27 0.   0.   0.   0.   0.  ]
[0.   1.   1.   0.73 0.27 0.   0.27 0.   0.   0.  ]
[0.   1.   1.   0.73 0.27 0.   0.27 0.73 0.   0.  ]
[0.   1.   1.   0.73 0.27 0.   0.27 0.73 1.   0.  ]
result:    [0.   1.   1.   0.73 0.27 0.   0.27 0.73 1.   0.  ]
cv2 blur:  [1.   1.   1.   0.73 0.27 0.   0.27 0.73 1.   1.  ]
```

Question discussions

Question 1: Box Filtering [1 pt]

Implement two box filters (one with 3 by 3, the other one is 5 by 5 kernel size), and apply them to the given images (i.e., image1.png and image2.png) separately. Show the resulting images, and explain the resulting images, and their differences. (For convolution operation, you can use built-in function. Do not use built-in function for box filtering.)

- Prepare box filters as shown before
- Perform 2D convolution for each filter on the image



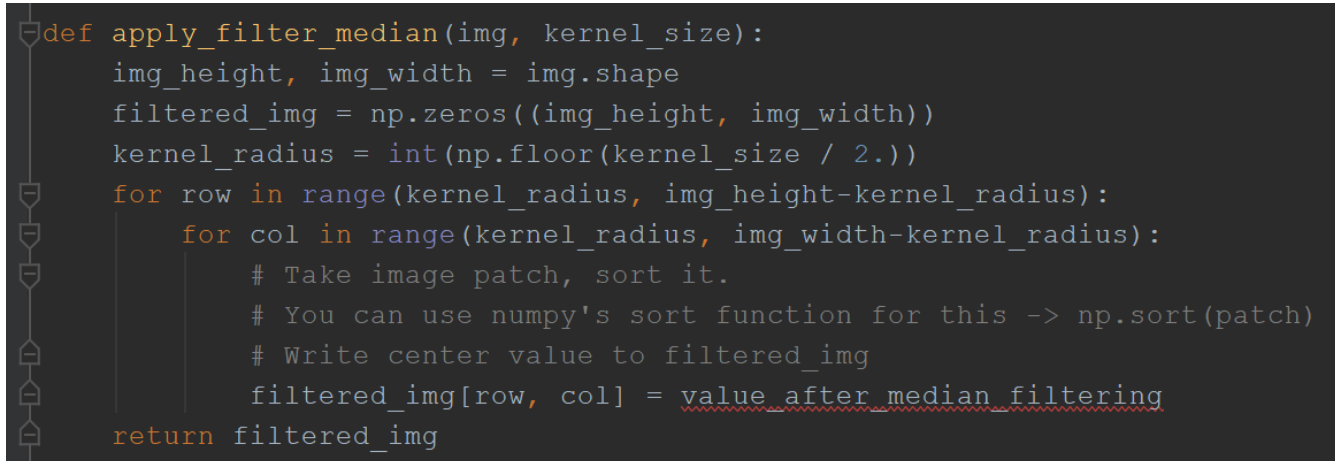
```python
def apply_filter(img, kernel):
    img_height, img_width = img.shape
    filtered_img = np.zeros((img_height, img_width))
    kernel_radius = int(np.floor(kernel.shape[0] / 2.))
    for row in range(kernel_radius, img_height-kernel_radius):
        for col in range(kernel_radius, img_width-kernel_radius):
            # Your code to do convolution operation
            # Write new value to filtered_img
            filtered_img[row, col] = value_after_convolution
    return filtered_img
```

Question discussions

Question 2: Median Filtering [1 pt]

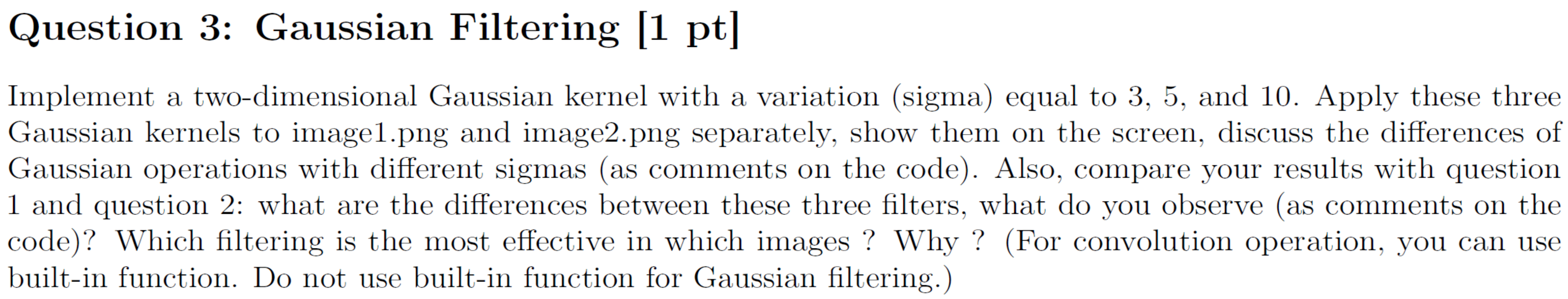
Implement three median filters (3 by 3, 5 by 5, and 7 by 7 kernel size), and apply these filters to image1.png and image2.png separately. Show (and discuss as comments) the resulting differences for each kernel on the screen, explain where median filters are most effective. (For convolution operation, you can use built-in function. Do not use built-in function for Median filtering.)

- Take image patch
- Sort and find median for each patch



```python
def apply_filter_median(img, kernel_size):
    img_height, img_width = img.shape
    filtered_img = np.zeros((img_height, img_width))
    kernel_radius = int(np.floor(kernel_size / 2.))
    for row in range(kernel_radius, img_height-kernel_radius):
        for col in range(kernel_radius, img_width-kernel_radius):
            # Take image patch, sort it.
            # You can use numpy's sort function for this -> np.sort(patch)
            # Write center value to filtered_img
            filtered_img[row, col] = value_after_median_filtering
    return filtered_img
```

Question discussions

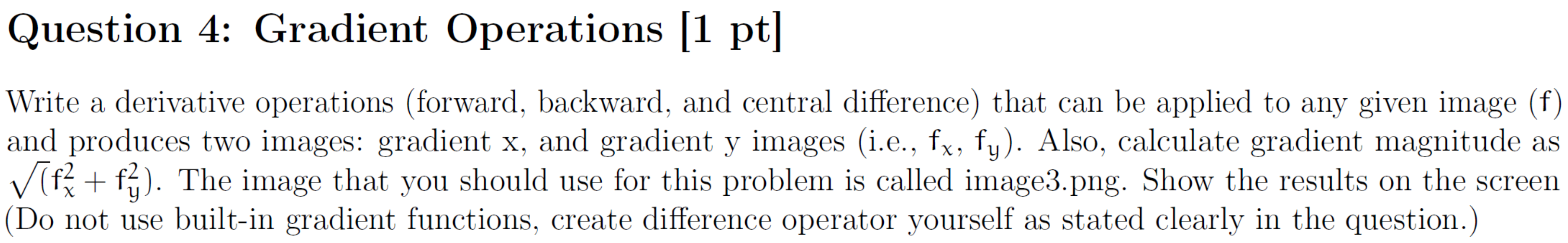


Question 3: Gaussian Filtering [1 pt]

Implement a two-dimensional Gaussian kernel with a variation (sigma) equal to 3, 5, and 10. Apply these three Gaussian kernels to image1.png and image2.png separately, show them on the screen, discuss the differences of Gaussian operations with different sigmas (as comments on the code). Also, compare your results with question 1 and question 2: what are the differences between these three filters, what do you observe (as comments on the code)? Which filtering is the most effective in which images ? Why ? (For convolution operation, you can use built-in function. Do not use built-in function for Gaussian filtering.)

- Prepare 2D Gaussian filter (similar to before)
  - Option 1
    - Prepare 1D Gaussian and reshape [np.reshape(kernel_gaus, (3,1))]
    - Do dot product [np.dot(kernel_gaus, kernel_gaus.T)]
  - Option 2
    - Use the Gaussian formula for two variables (x, y)
- Apply filter similar to filter_1d function from before
  - Patch and filter will be 2D

Question discussions

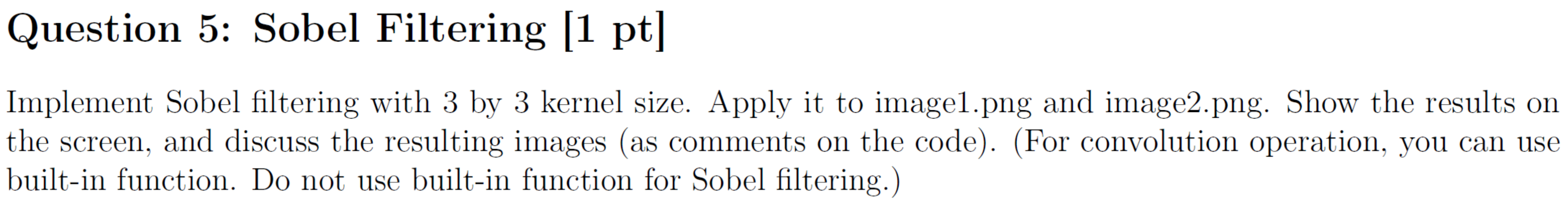


Question 4: Gradient Operations [1 pt]

Write a derivative operations (forward, backward, and central difference) that can be applied to any given image (f) and produces two images: gradient x, and gradient y images (i.e., f_x, f_y). Also, calculate gradient magnitude as $\sqrt{(f_x^2 + f_y^2)}$. The image that you should use for this problem is called image3.png. Show the results on the screen (Do not use built-in gradient functions, create difference operator yourself as stated clearly in the question.)

- Make three 1D filters [-1, +1] , [+1, -1] , [-1, 0 ,+1]
- Apply horizontally (similar to filter_1d). Gives f_x
- Get transpose of those filters, apply vertically. Gives f_y
- Get magnitude
  - Use element-wise matrix operation as np.sqrt(f_x*f_x + f_y*f_y)
- Scale image
  - Get max value of magnitude -> np.max((magnitude))
  - Divide magnitude by max value -> Makes between 0-1
  - Multiply by 255 -> Makes between 0-255
  - Convert to integer and save/show magnitude image

Question discussions



Question 5: Sobel Filtering [1 pt]

Implement Sobel filtering with 3 by 3 kernel size. Apply it to image1.png and image2.png. Show the results on the screen, and discuss the resulting images (as comments on the code). (For convolution operation, you can use built-in function. Do not use built-in function for Sobel filtering.)

- Prepare Sobel kernel
  - For x and y direction (S_x and S_y)

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

- Similar to Q3 and Q4
  - Apply S_x and S_y to image
  - Combine to get final gradient -> np.sqrt(S_x*S_x + S_y*S_y)

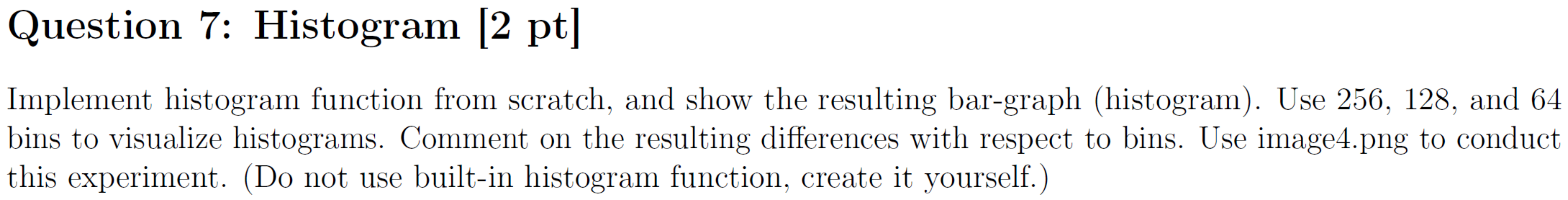
Question discussions

Question 6: Fast Gaussian Filtering [1 pt]

Implement question 3, but this time, use only 1D Gaussian to do filtering in 2D. The trick is the following: since Gaussians are separable, you can use 1D Gaussian to filter the image in one direction first, and then the other direction can be filtered. Show the results on the screen and compare the efficiency of both methods (question 3 and question 6) as comments on the code. Use image1.png and image2.png for smoothing and show results too. (For convolution operation, you can use built-in function. Do not use built-in function for fast Gaussian filtering.)

- Similar to Q3
  - Use only 1D filter
  - Apply in first direction
  - Transpose the filter, apply in another direction
  - Combine using magnitude formula

Question discussions

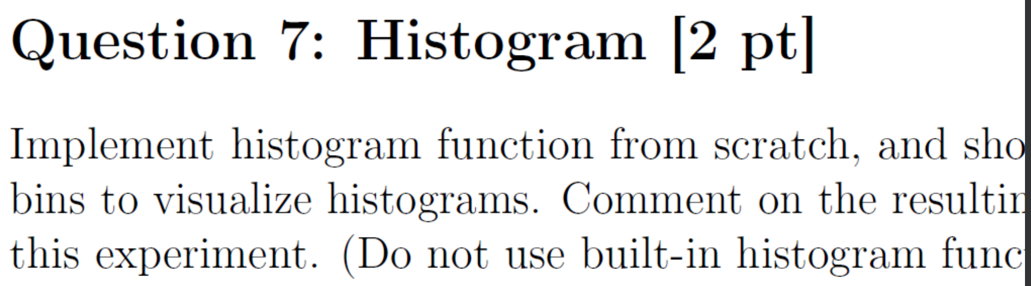


Question 7: Histogram [2 pt]

Implement histogram function from scratch, and show the resulting bar-graph (histogram). Use 256, 128, and 64 bins to visualize histograms. Comment on the resulting differences with respect to bins. Use image4.png to conduct this experiment. (Do not use built-in histogram function, create it yourself.)

- Loop through each pixel
- Find bin for that pixel and add 1 to that bin

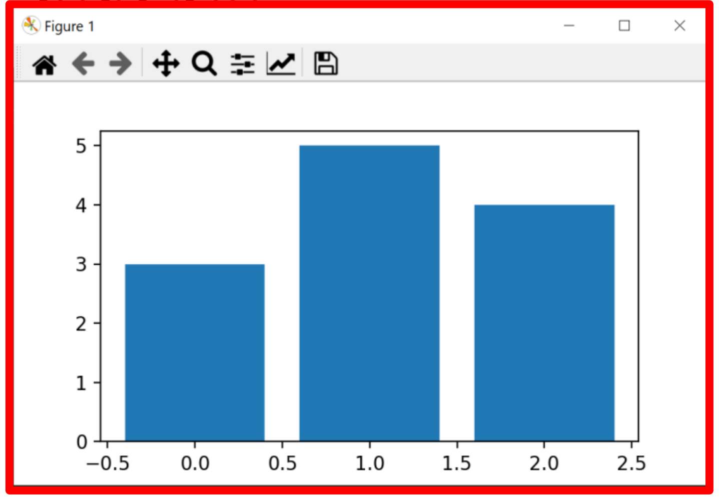
Question discussions

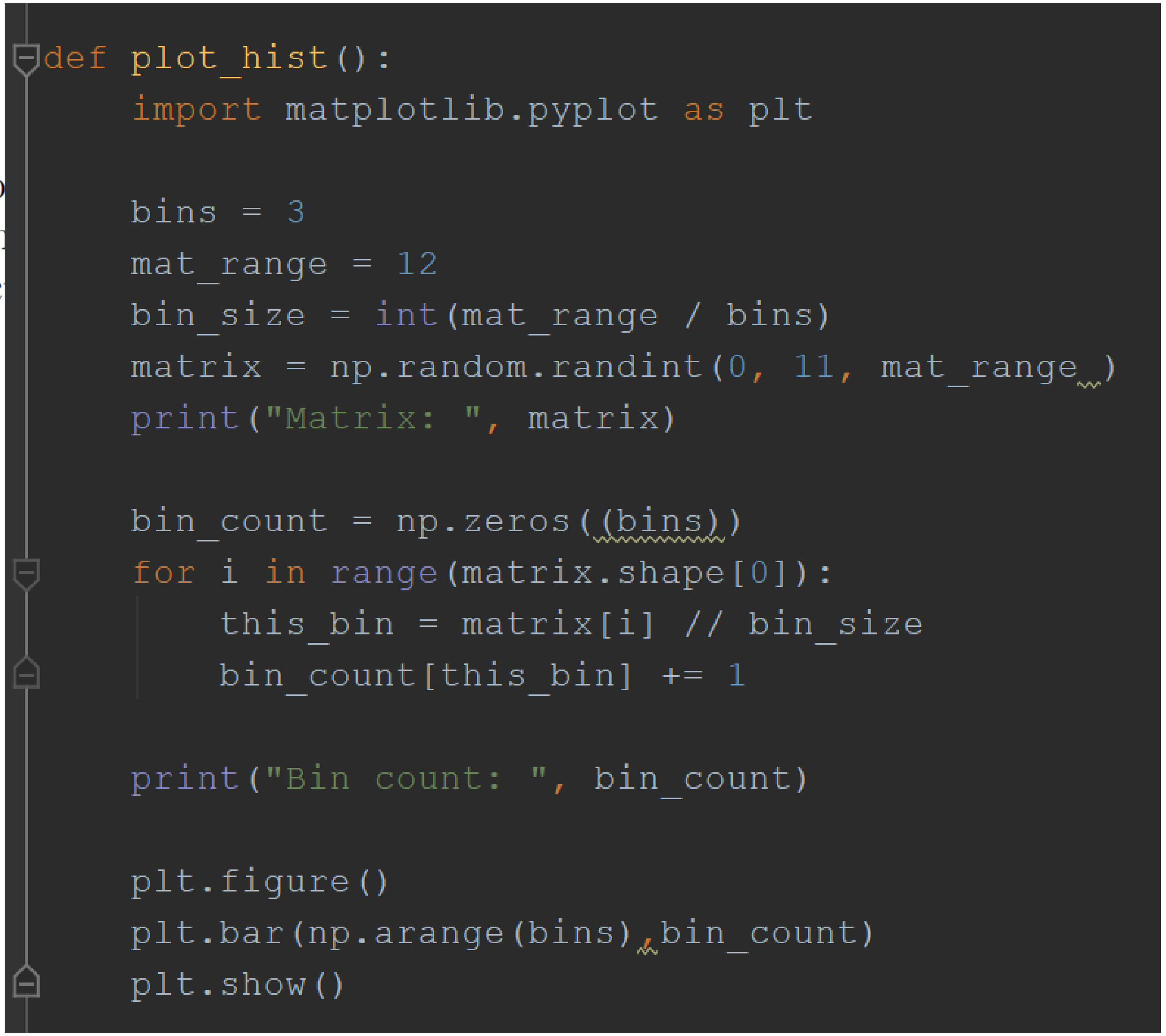


Question 7: Histogram [2 pt]

Implement histogram function from scratch, and sho
bins to visualize histograms. Comment on the resultin
this experiment. (Do not use built-in histogram func

- Loop through each pixel
- Find bin for that pixel and add 1 to that bin





```
def plot_hist():
    import matplotlib.pyplot as plt

    bins = 3
    mat_range = 12
    bin_size = int(mat_range / bins)
    matrix = np.random.randint(0, 11, mat_range )
    print("Matrix: ", matrix)

    bin_count = np.zeros((bins))
    for i in range(matrix.shape[0]):
        this_bin = matrix[i] // bin_size
        bin_count[this_bin] += 1

    print("Bin count: ", bin_count)

    plt.figure()
    plt.bar(np.arange(bins),bin_count)
    plt.show()
```

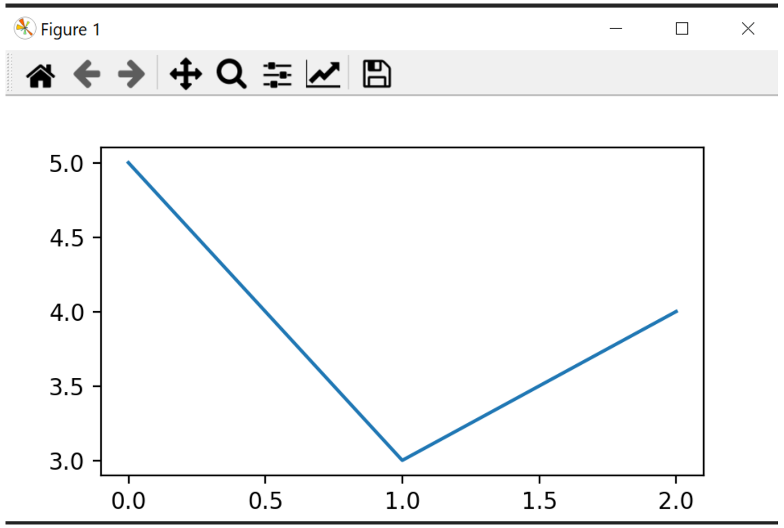
Question discussions

Question 7: Histogram [2 pt]

Implement histogram function from scratch, and show the resulting bar-graph (histogram). Use 256, 128, and 64 bins to visualize histograms. Comment on the resulting differences with respect to bins. Use image4.png to conduct this experiment. (Do not use built-in histogram function, create it yourself.)

- Loop through each pixel
- Find bin for that pixel and add 1 to that bin

```
plt.figure()
plt.plot(bin_count)
plt.show()
```



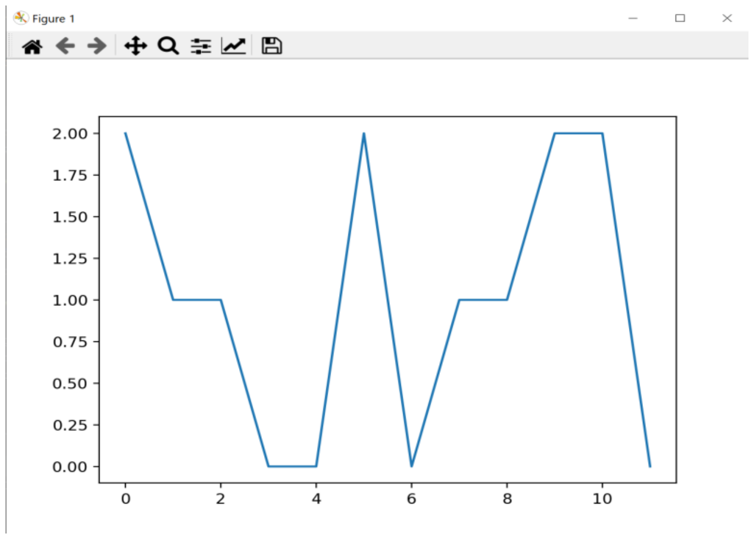
Question discussions

Question 7: Histogram [2 pt]

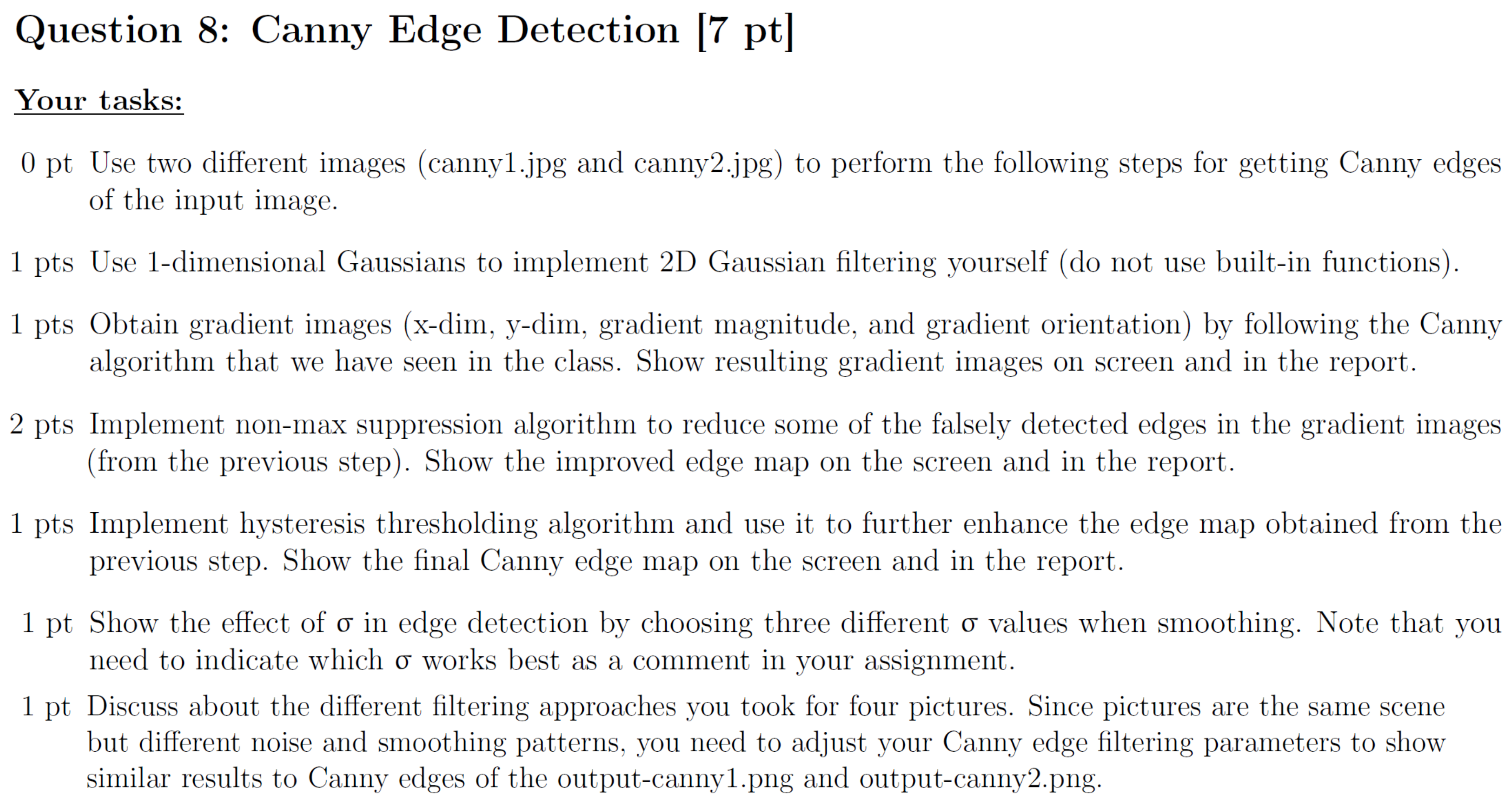
Implement histogram function from scratch, and show the resulting bar-graph (histogram). Use 256, 128, and 64 bins to visualize histograms. Comment on the resulting differences with respect to bins. Use image4.png to conduct this experiment. (Do not use built-in histogram function, create it yourself.)

- Loop through each pixel
- Find bin for that pixel and add 1 to that bin
- bins=12

```
plt.figure()
plt.plot(bin_count)
plt.show()
```



Question discussions



Question 8: Canny Edge Detection [7 pt]

Your tasks:

0 pt Use two different images (canny1.jpg and canny2.jpg) to perform the following steps for getting Canny edges of the input image.

1 pts Use 1-dimensional Gaussians to implement 2D Gaussian filtering yourself (do not use built-in functions).

1 pts Obtain gradient images (x-dim, y-dim, gradient magnitude, and gradient orientation) by following the Canny algorithm that we have seen in the class. Show resulting gradient images on screen and in the report.

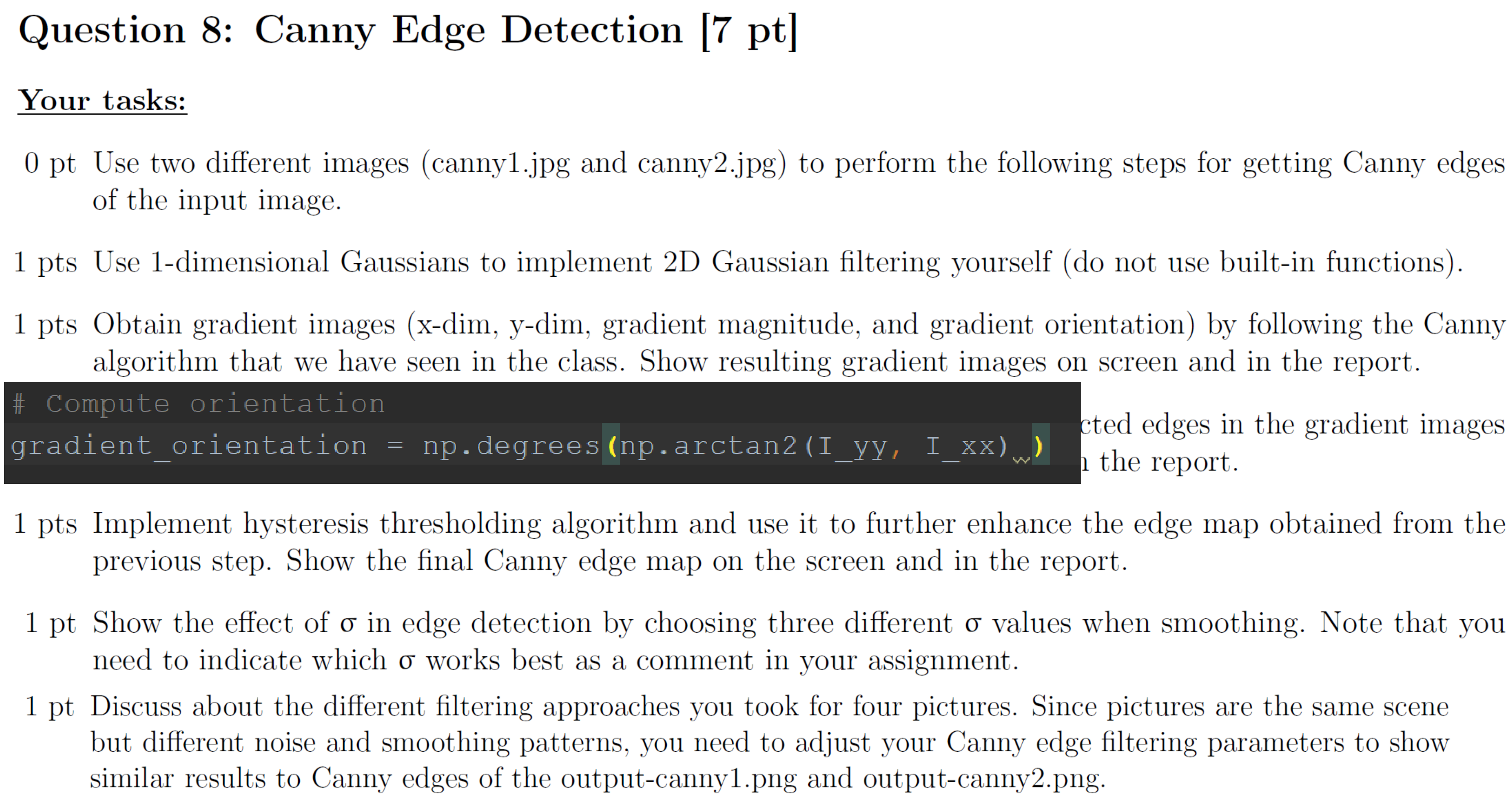
2 pts Implement non-max suppression algorithm to reduce some of the falsely detected edges in the gradient images (from the previous step). Show the improved edge map on the screen and in the report.

1 pts Implement hysteresis thresholding algorithm and use it to further enhance the edge map obtained from the previous step. Show the final Canny edge map on the screen and in the report.

1 pt Show the effect of σ in edge detection by choosing three different σ values when smoothing. Note that you need to indicate which σ works best as a comment in your assignment.

1 pt Discuss about the different filtering approaches you took for four pictures. Since pictures are the same scene but different noise and smoothing patterns, you need to adjust your Canny edge filtering parameters to show similar results to Canny edges of the output-canny1.png and output-canny2.png.

Question discussions



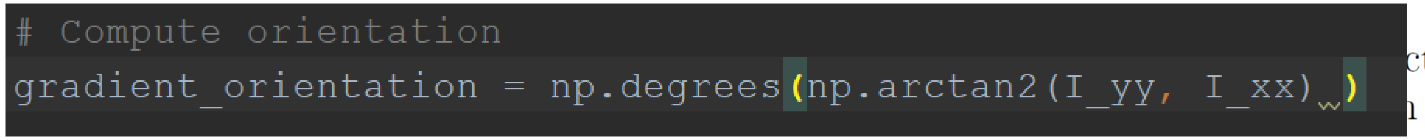
Question 8: Canny Edge Detection [7 pt]

<u>Your tasks:</u>

0 pt Use two different images (canny1.jpg and canny2.jpg) to perform the following steps for getting Canny edges of the input image.

1 pts Use 1-dimensional Gaussians to implement 2D Gaussian filtering yourself (do not use built-in functions).

1 pts Obtain gradient images (x-dim, y-dim, gradient magnitude, and gradient orientation) by following the Canny algorithm that we have seen in the class. Show resulting gradient images on screen and in the report.



```
# Compute orientation
gradient_orientation = np.degrees(np.arctan2(I_yy, I_xx))
```

cted edges in the gradient images
n the report.

1 pts Implement hysteresis thresholding algorithm and use it to further enhance the edge map obtained from the previous step. Show the final Canny edge map on the screen and in the report.

1 pt Show the effect of σ in edge detection by choosing three different σ values when smoothing. Note that you need to indicate which σ works best as a comment in your assignment.

1 pt Discuss about the different filtering approaches you took for four pictures. Since pictures are the same scene but different noise and smoothing patterns, you need to adjust your Canny edge filtering parameters to show similar results to Canny edges of the output-canny1.png and output-canny2.png.

Question discussions

Question 8: Canny Edge Detection [7 pt]

Your tasks:

0 pt Use two different images (canny1.jpg and canny2.jpg) to perform the following steps for getting Canny edges of the input image.

1 pts Use 1-dimensional your

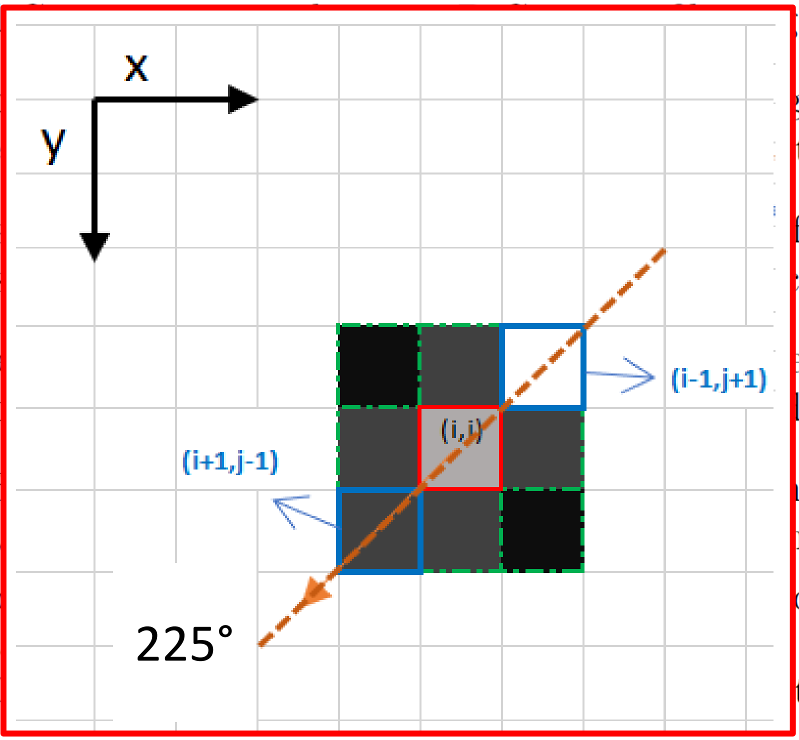
1 pts Obtain gradient in gradie
algorithm that we t ima

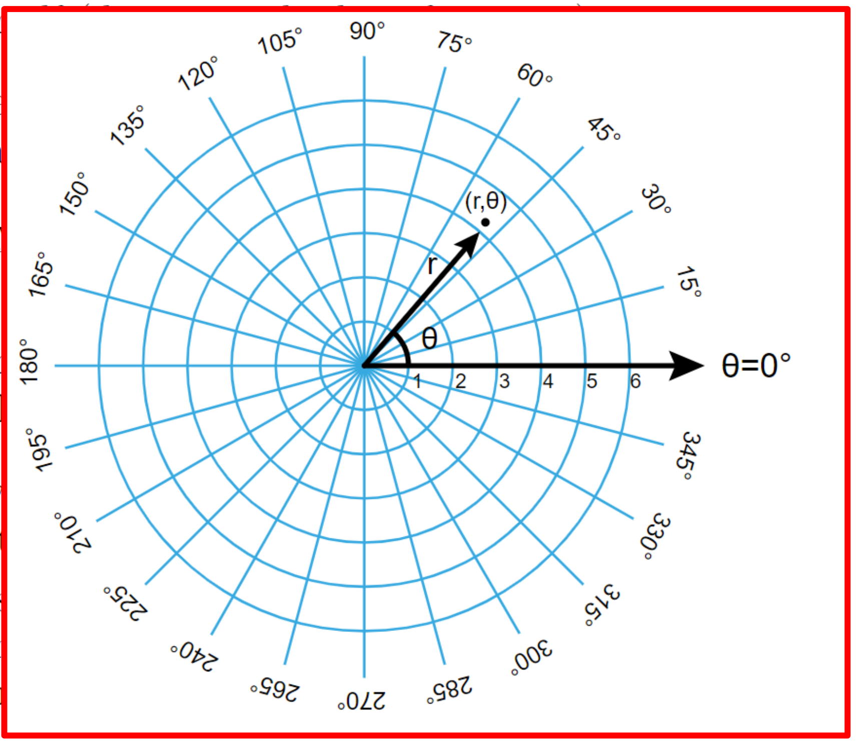
2 pts Implement non-ma falsely
(from the previous reen

1 pts Implement hyster er enh
previous step. Sho l in th

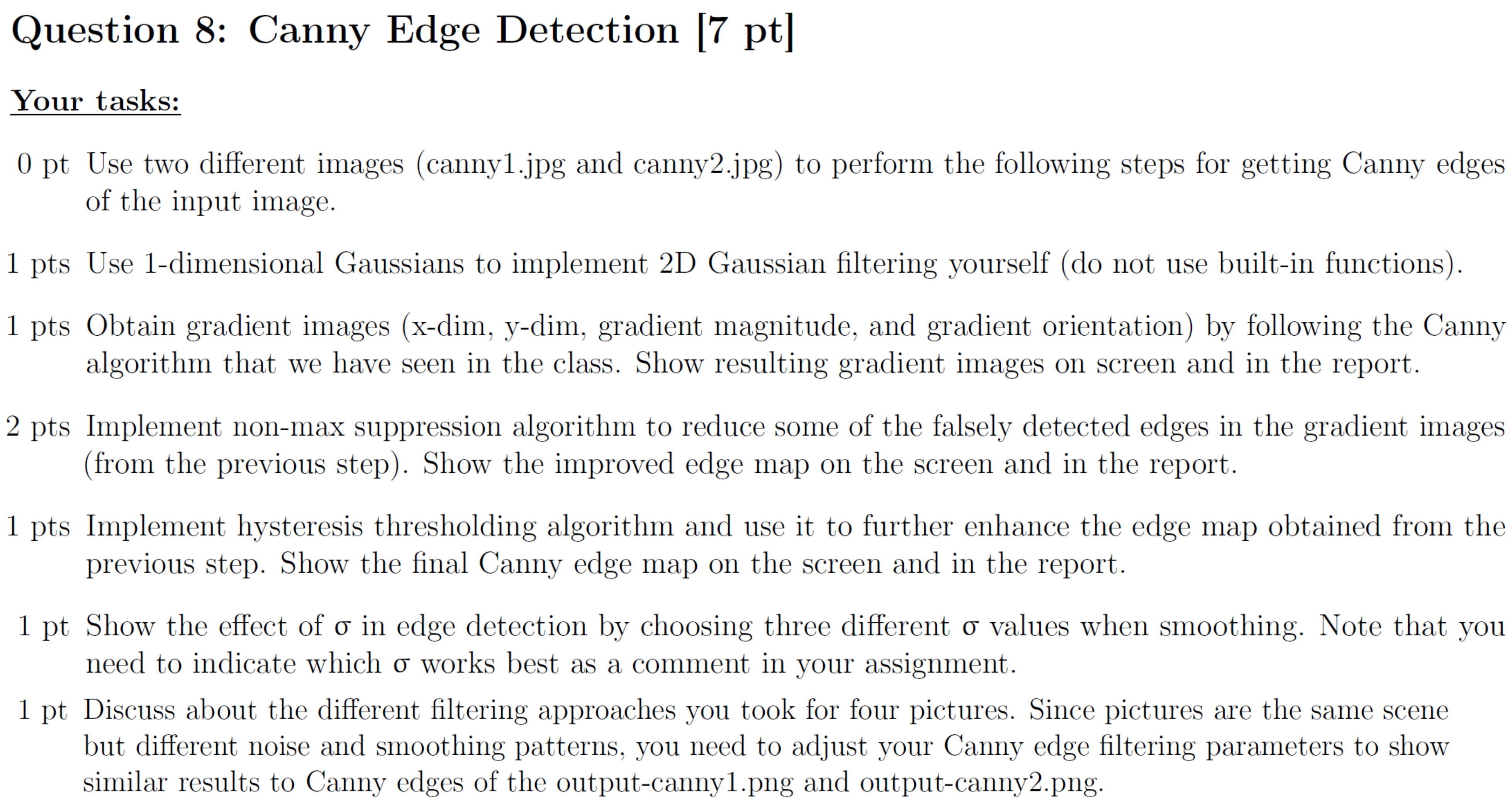
1 pt Show the effect of t σ v
need to indicate w ment

1 pt Discuss about the ctures
but different noise Cann
similar results to C t-cann





Question discussions



Question 8: Canny Edge Detection [7 pt]

Your tasks:

0 pt Use two different images (canny1.jpg and canny2.jpg) to perform the following steps for getting Canny edges of the input image.

1 pts Use 1-dimensional Gaussians to implement 2D Gaussian filtering yourself (do not use built-in functions).

1 pts Obtain gradient images (x-dim, y-dim, gradient magnitude, and gradient orientation) by following the Canny algorithm that we have seen in the class. Show resulting gradient images on screen and in the report.

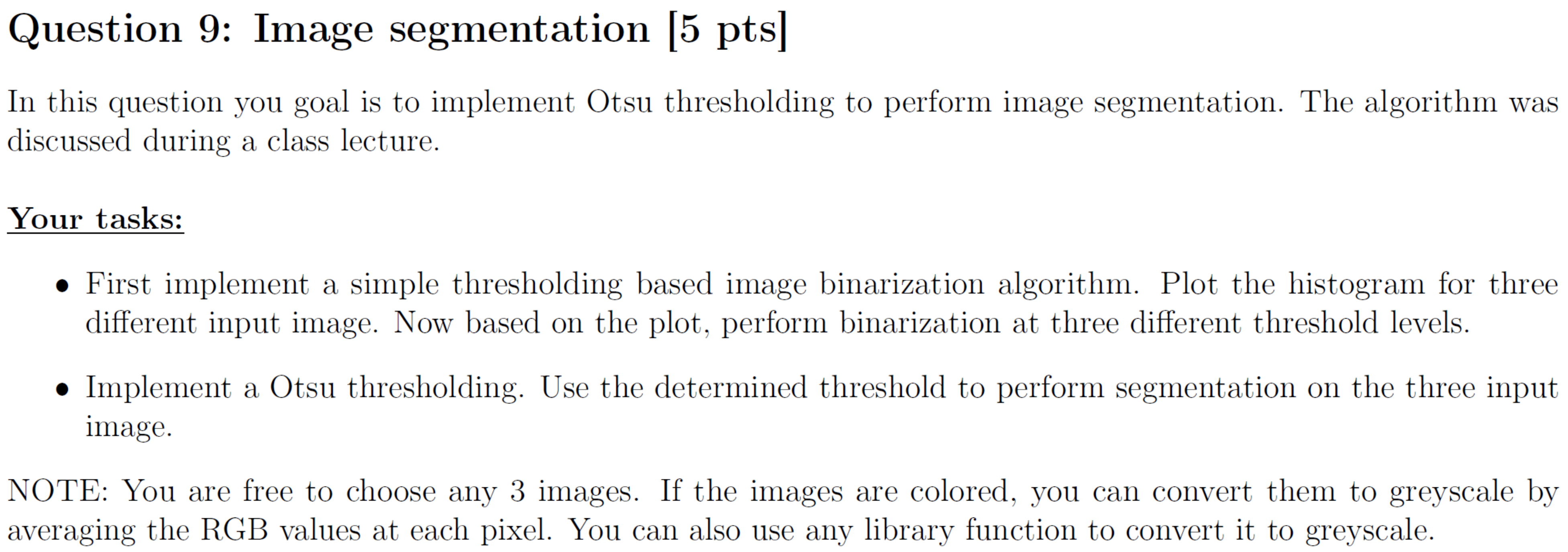
2 pts Implement non-max suppression algorithm to reduce some of the falsely detected edges in the gradient images (from the previous step). Show the improved edge map on the screen and in the report.

1 pts Implement hysteresis thresholding algorithm and use it to further enhance the edge map obtained from the previous step. Show the final Canny edge map on the screen and in the report.

1 pt Show the effect of σ in edge detection by choosing three different σ values when smoothing. Note that you need to indicate which σ works best as a comment in your assignment.

1 pt Discuss about the different filtering approaches you took for four pictures. Since pictures are the same scene but different noise and smoothing patterns, you need to adjust your Canny edge filtering parameters to show similar results to Canny edges of the output-canny1.png and output-canny2.png.

Question discussions



Question 9: Image segmentation [5 pts]

In this question you goal is to implement Otsu thresholding to perform image segmentation. The algorithm was discussed during a class lecture.

Your tasks:

- First implement a simple thresholding based image binarization algorithm. Plot the histogram for three different input image. Now based on the plot, perform binarization at three different threshold levels.
- Implement a Otsu thresholding. Use the determined threshold to perform segmentation on the three input image.

NOTE: You are free to choose any 3 images. If the images are colored, you can convert them to greyscale by averaging the RGB values at each pixel. You can also use any library function to convert it to greyscale.