

coursework_01

January 31, 2023

1 Coursework 1: Image filtering

In this coursework you will practice techniques for image filtering. The coursework includes coding questions and written questions. Please read both the text and the code in this notebook to get an idea what you are expected to implement.

1.1 What to do?

- Complete and run the code using `jupyter-lab` or `jupyter-notebook` to get the results.
- Export (File | Save and Export Notebook As...) the notebook as a PDF file, which contains your code, results and answers, and upload the PDF file onto [Scientia](#).
- Instead of clicking the Export button, you can also run the following command instead:
`jupyter nbconvert coursework_01_solution.ipynb --to pdf`
- If Jupyter complains about some problems in exporting, it is likely that pandoc (<https://pandoc.org/installing.html>) or latex is not installed, or their paths have not been included. You can install the relevant libraries and retry. Alternatively, use the Print function of your browser to export the PDF file.
- If Jupyter-lab does not work for you at the end (we hope not), you can use Google Colab to write the code and export the PDF file.

1.2 Dependencies:

You need to install Jupyter-Lab (https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html) and other libraries used in this coursework, such as by running the command: `pip3 install [package_name]`

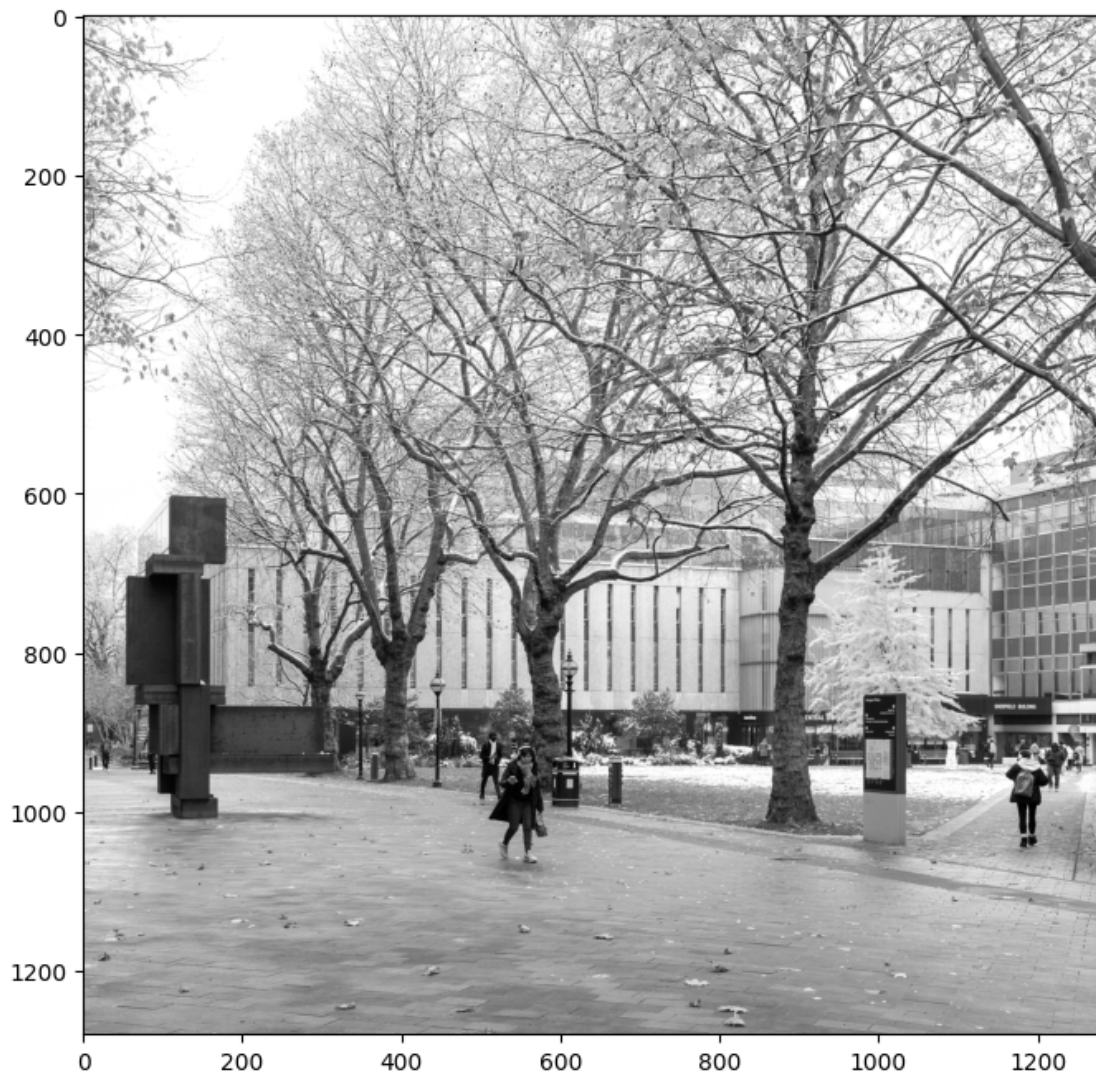
```
[1]: # Import libraries (provided)
import imageio.v3 as imageio
import numpy as np
import matplotlib.pyplot as plt
import noise
import scipy
import scipy.signal
import math
import time
```

1.3 1. Moving average filter (20 points).

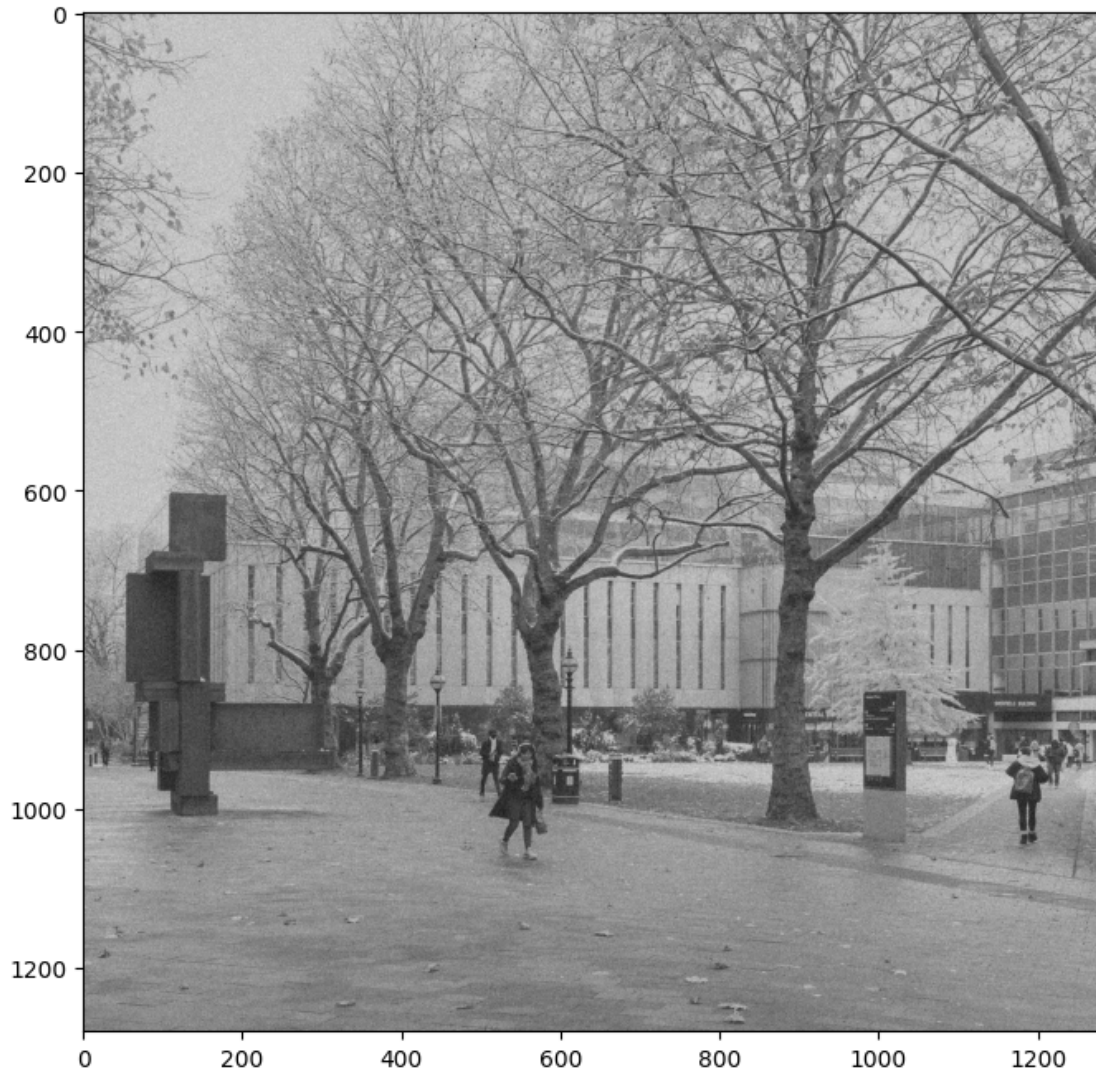
Read the provided input image, add noise to the image and design a moving average filter for denoising.

You are expected to design the kernel of the filter and then perform 2D image filtering using the function `scipy.signal.convolve2d()`.

```
[2]: # Read the image (provided)
image = imageio.imread('campus_snow.jpg')
plt.imshow(image, cmap='gray')
plt.gcf().set_size_inches(8, 8)
```



```
[3]: # Corrupt the image with Gaussian noise (provided)
image_noisy = noise.add_noise(image, 'gaussian')
plt.imshow(image_noisy, cmap='gray')
plt.gcf().set_size_inches(8, 8)
```



1.3.1 Note: from now on, please use the noisy image as the input for the filters.

1.3.2 1.1 Filter the noisy image with a 3x3 moving average filter. Show the filtering results.

```
[4]: # Design the filter h
### Insert your code ###
h = np.array([[1/9, 1/9, 1/9], [1/9, 1/9, 1/9], [1/9, 1/9, 1/9]])
```

```

# Convolve the corrupted image with h using scipy.signal.convolve2d function
### Insert your code ###
image_filtered = scipy.signal.convolve2d(image_noisy, h)

# Print the filter (provided)
print('Filter h:')
print(h)

# Display the filtering result (provided)
plt.imshow(image_filtered, cmap='gray')
plt.gcf().set_size_inches(8, 8)

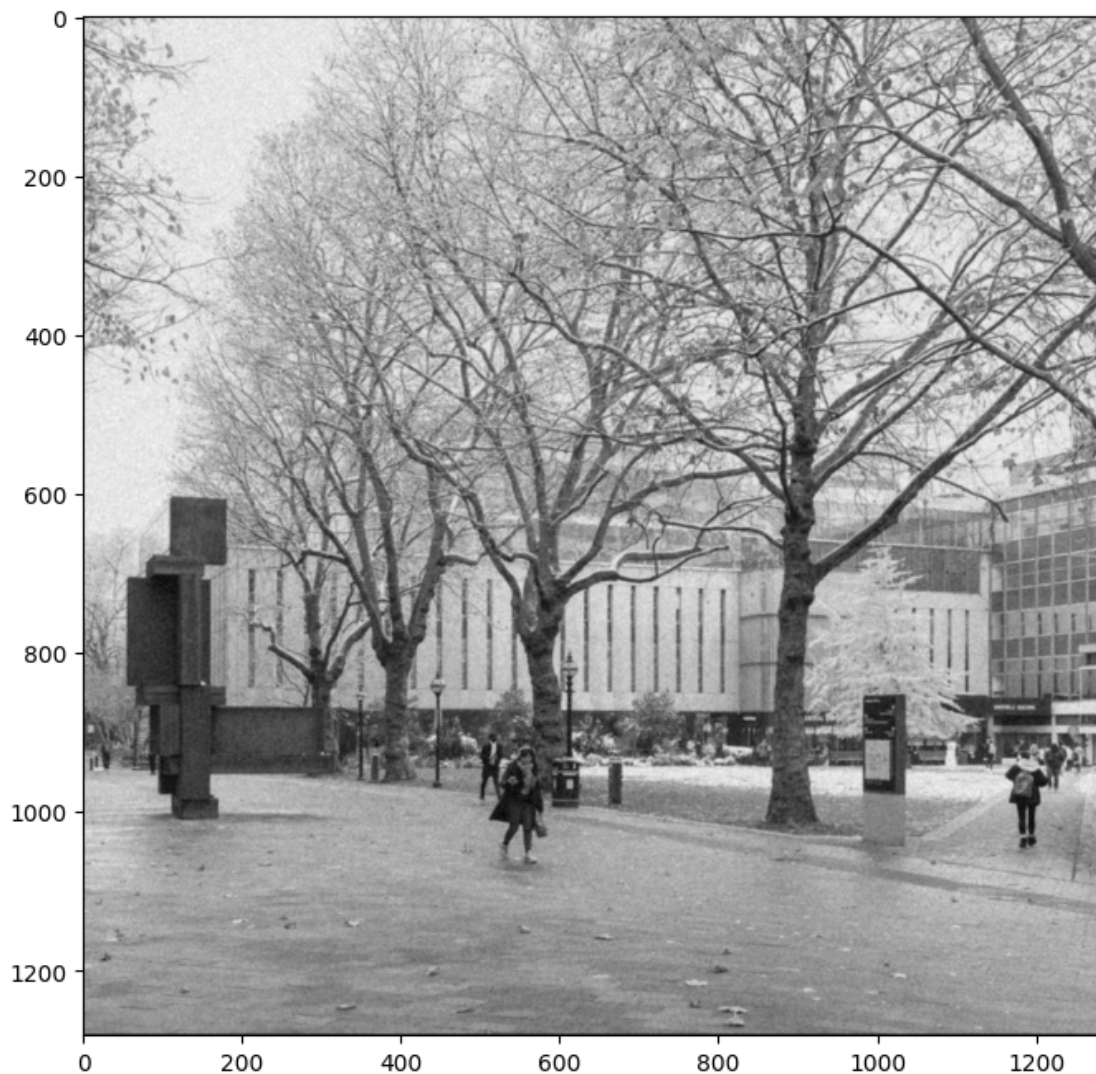
```

Filter h:

```

[[0.11111111 0.11111111 0.11111111]
 [0.11111111 0.11111111 0.11111111]
 [0.11111111 0.11111111 0.11111111]]

```



1.3.3 1.2 Filter the noisy image with a 11x11 moving average filter.

```
[5]: # Design the filter h
    ### Insert your code ###
    h = np.array([[1/121 for j in range(11)] for i in range(11)])

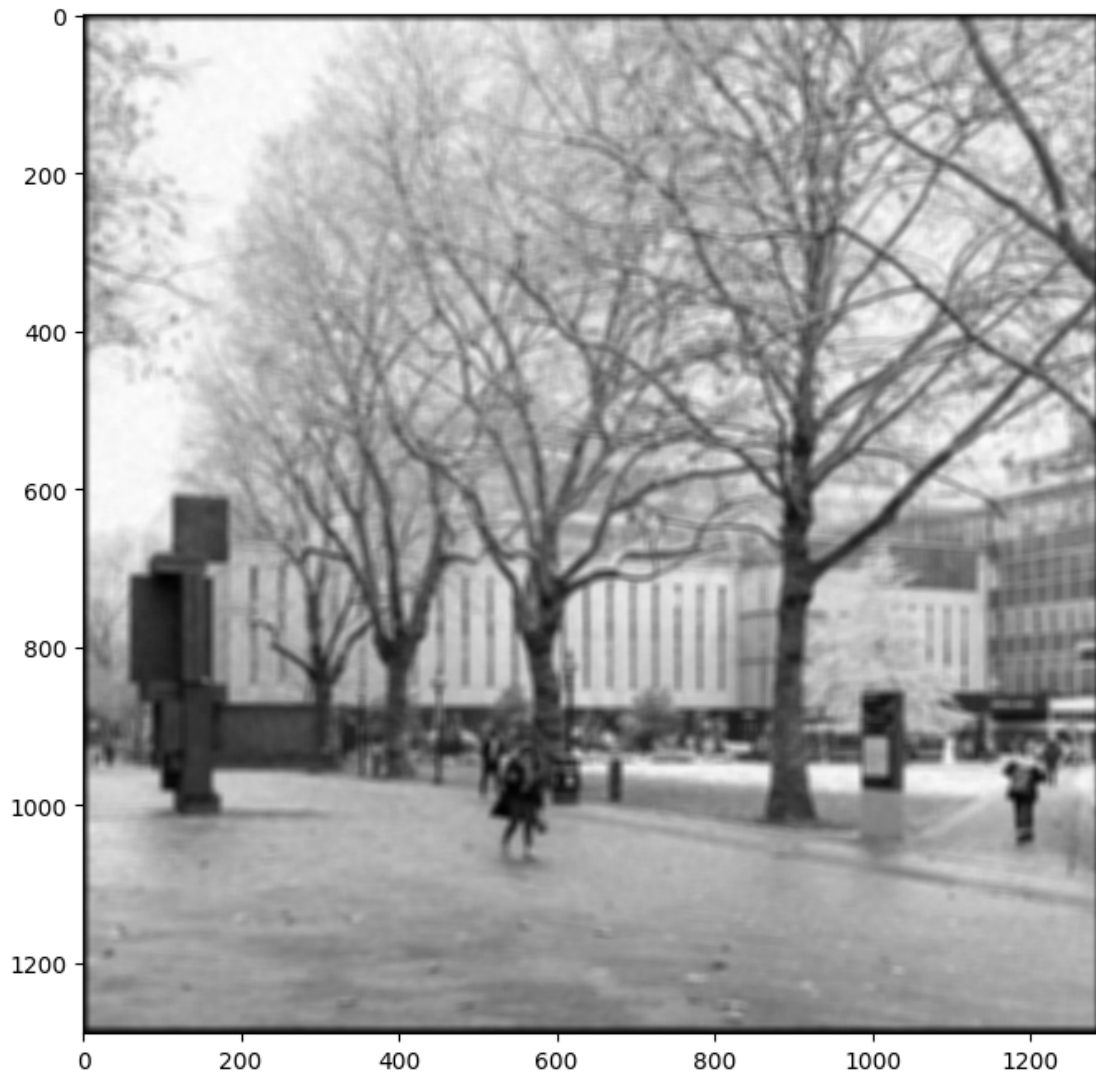
    # Convolve the corrupted image with h using scipy.signal.convolve2d function
    ### Insert your code ###
    image_filtered = scipy.signal.convolve2d(image_noisy, h)

    # Print the filter (provided)
    print('Filter h:')
    print(h)

    # Display the filtering result (provided)
    plt.imshow(image_filtered, cmap='gray')
    plt.gcf().set_size_inches(8, 8)
```

Filter h:

```
[[0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]]
```



1.3.4 1.3 Comment on the filtering results. How do different kernel sizes influence the filtering results?

Insert your answer

Both filters can remove the Gaussian noise from the noisy image. This is because moving average filter is a low-pass filter and it pass through low frequency signals while attenuating high frequency signals, since noise pixels are high frequency signals, they are removed.

The filtering kernel with 11×11 size does better in terms of identifying and removing all the noise pixels but it also blurs the image significantly. The filtering kernel with 3×3 size can remove most noise pixels but still leaves some undetected, this kernel also blurs the image compare to the original image, but the blurring effect is less significant.

1.4 2. Edge detection (56 points).

Perform edge detection using Sobel filtering, as well as Gaussian + Sobel filtering.

1.4.1 2.1 Implement 3x3 Sobel filters and convolve with the noisy image.

```
[6]: # Design the filters
      ### Insert your code ###
      sobel_x = np.array([[1, 0, -1],[2, 0, -2],[1, 0, -1]])
      sobel_y = np.array([[1, 2, 1],[0, 0, 0],[-1, -2, -1]])

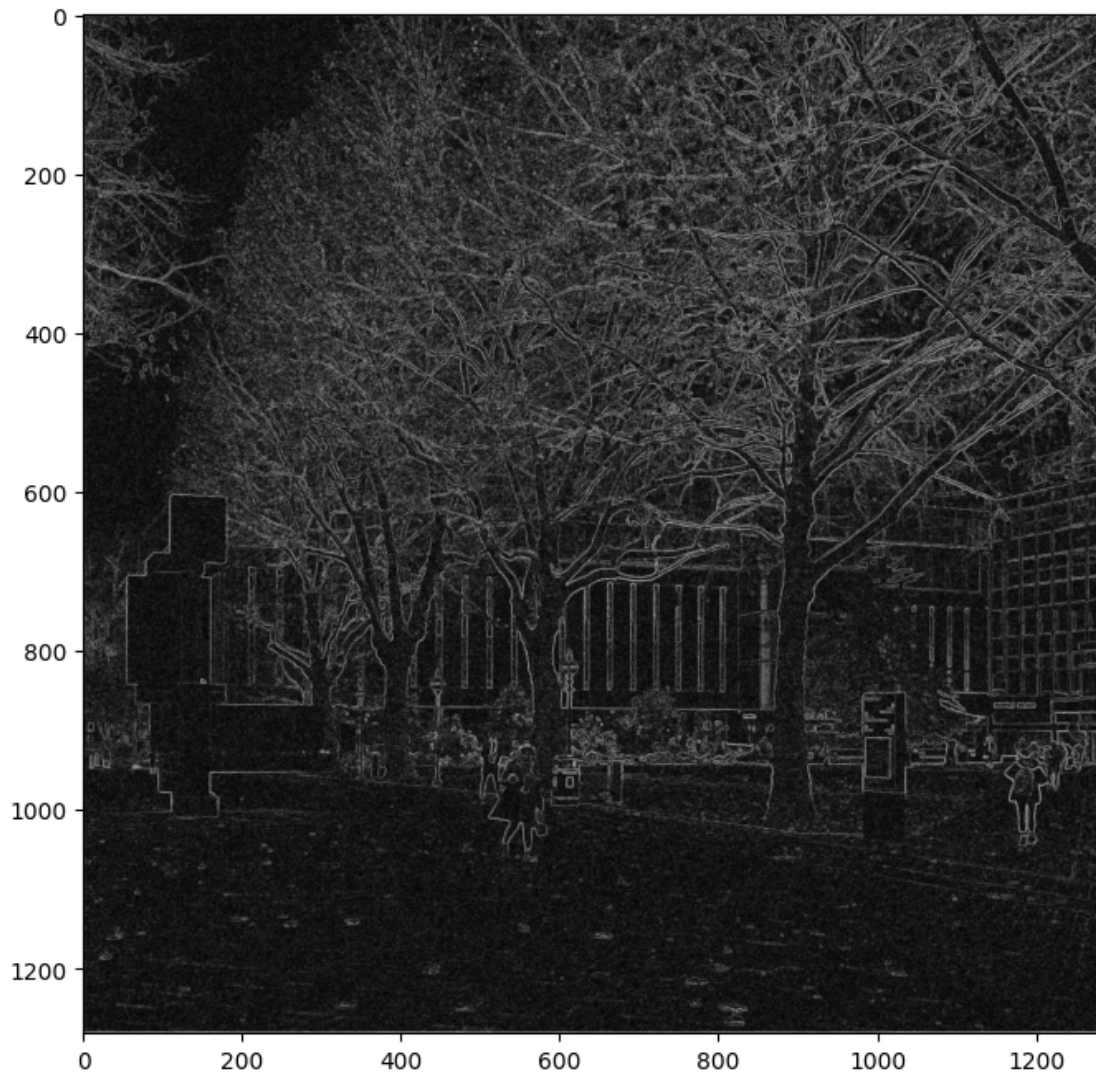
      # Image filtering
      ### Insert your code ###
      grad_x = scipy.signal.convolve2d(image_noisy, sobel_x)
      grad_y = scipy.signal.convolve2d(image_noisy, sobel_y)

      # Calculate the gradient magnitude
      ### Insert your code ###
      grad_mag = np.sqrt(np.square(grad_x) + np.square(grad_y))

      # Print the filters (provided)
      print('sobel_x:')
      print(sobel_x)
      print('sobel_y:')
      print(sobel_y)

      # Display the magnitude map (provided)
      plt.imshow(grad_mag, cmap='gray')
      plt.gcf().set_size_inches(8, 8)
```

```
sobel_x:
[[ 1  0 -1]
 [ 2  0 -2]
 [ 1  0 -1]]
sobel_y:
[[ 1  2  1]
 [ 0  0  0]
 [-1 -2 -1]]
```



1.4.2 2.2 Implement a function that generates a 2D Gaussian filter given the parameter σ .

```
[7]: # Design the Gaussian filter
def gaussian_filter_2d(sigma):
    # sigma: the parameter sigma in the Gaussian kernel (unit: pixel)
    #
    # return: a 2D array for the Gaussian kernel

    ### Insert your code ###

    #should be precise enough as the values are very small outside  $[-k\sigma, k\sigma]$  where  $k$  is 4
```



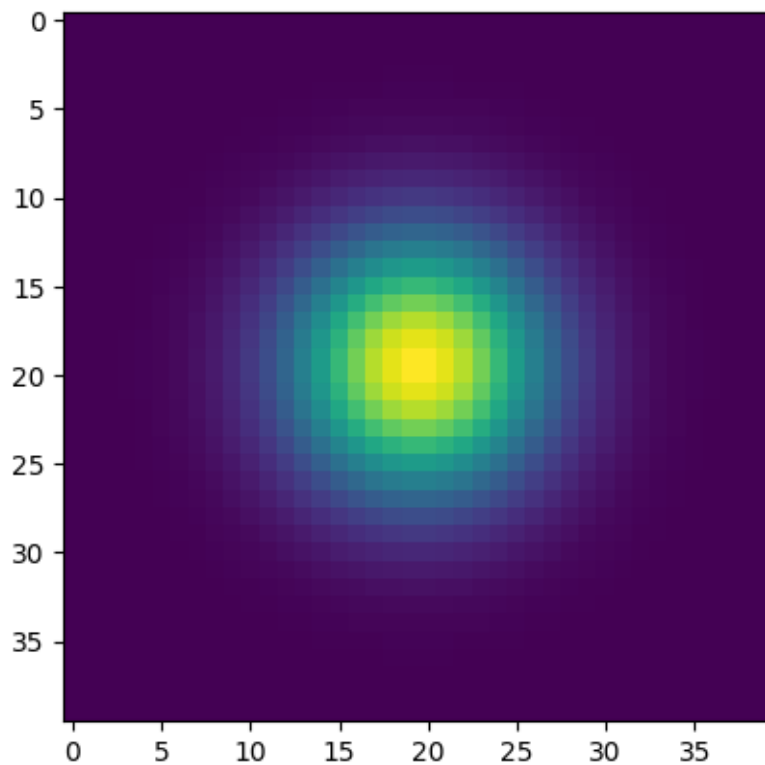
```

#generate two 1D gaussian arrays to be combined into a 2D Gaussian kernel
gaussian_a_1D = scipy.signal.gaussian(40, sigma)
gaussian_b_1D = scipy.signal.gaussian(40, sigma)
#2D Gaussian kernel produced by doing an outer product of two 1D Gaussian
↪arrays
h = np.outer(gaussian_a_1D, gaussian_b_1D)
#Normalise
h /= (2*np.pi*(sigma**2))
return h

# Visualise the Gaussian filter when sigma = 5 pixel (provided)
sigma = 5
h = gaussian_filter_2d(sigma)
plt.imshow(h)

```

[7]: <matplotlib.image.AxesImage at 0x1307b8cd0>



1.4.3 2.3 Perform Gaussian smoothing ($\sigma = 5$ pixels) and evaluate the computational time for Gaussian smoothing. After that, perform Sobel filtering and show the gradient magnitude map.

```
[8]: # Construct the Gaussian filter
### Insert your code ###
gaussian_filter = gaussian_filter_2d(5)

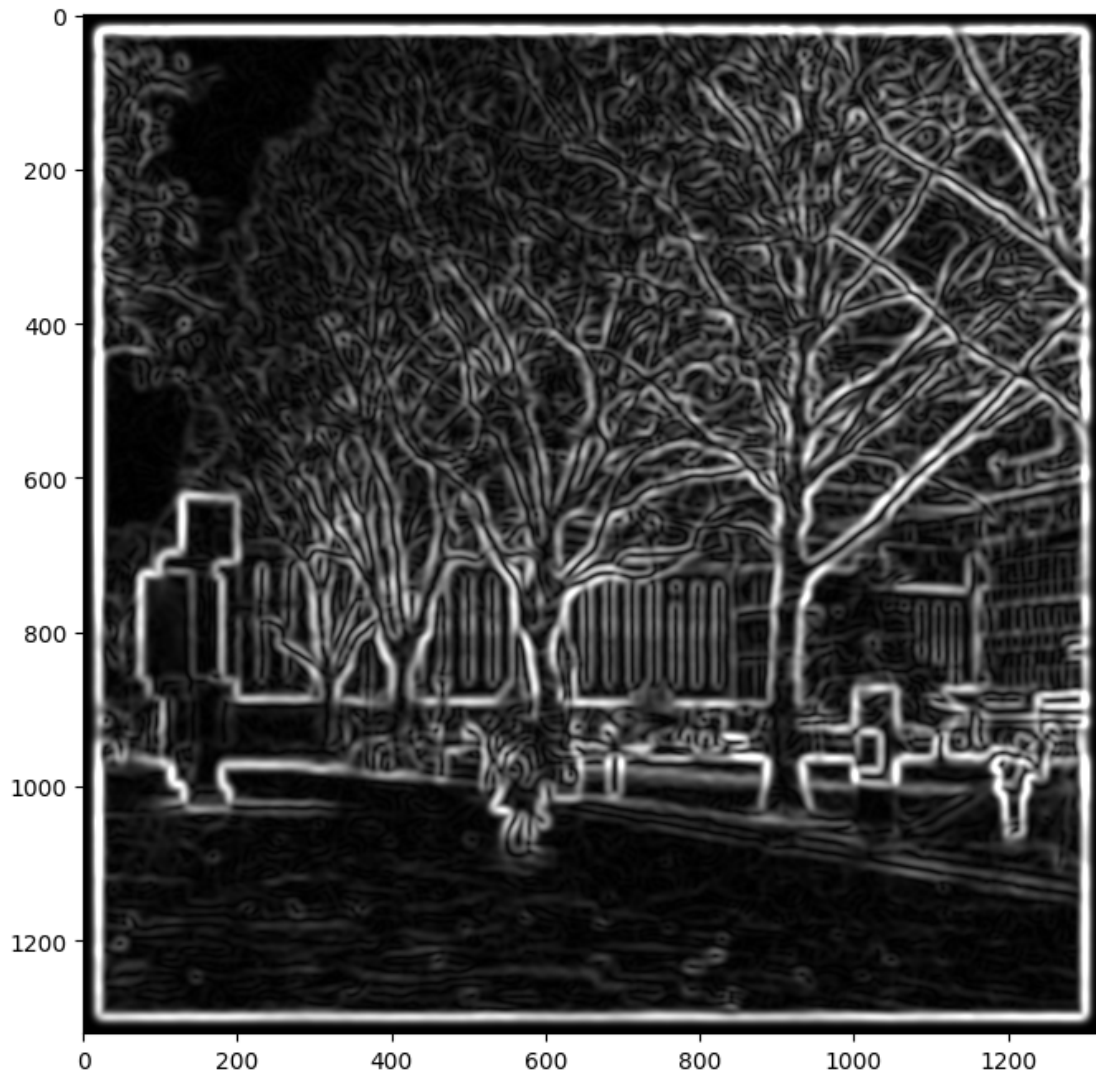
# Perform Gaussian smoothing and count time
### Insert your code ###
start_time = time.time()
image_gaussian = scipy.signal.convolve2d(image_noisy, gaussian_filter)
end_time = time.time()
comp_time = end_time - start_time
print(comp_time)

# Image filtering
### Insert your code ###
grad_x = scipy.signal.convolve2d(image_gaussian, sobel_x)
grad_y = scipy.signal.convolve2d(image_gaussian, sobel_y)

# Calculate the gradient magnitude
### Insert your code ###
grad_mag = np.sqrt(np.square(grad_x) + np.square(grad_y))

# Display the gradient magnitude map (provided)
plt.imshow(grad_mag, cmap='gray', vmin=0, vmax=100)
plt.gcf().set_size_inches(8, 8)
```

5.971627950668335



1.4.4 2.4 Implement a function that generates a 1D Gaussian filter given the parameter σ . Generate 1D Gaussian filters along x-axis and y-axis respectively.

```
[9]: # Design the Gaussian filter
def gaussian_filter_1d(sigma):
    # sigma: the parameter sigma in the Gaussian kernel (unit: pixel)
    #
    # return: a 1D array for the Gaussian kernel

    ### Insert your code ###
    h = scipy.signal.gaussian(40, sigma)
    h /= (np.sqrt(2*np.pi)*sigma)
    return h
```

```

# sigma = 5 pixel (provided)
sigma = 5

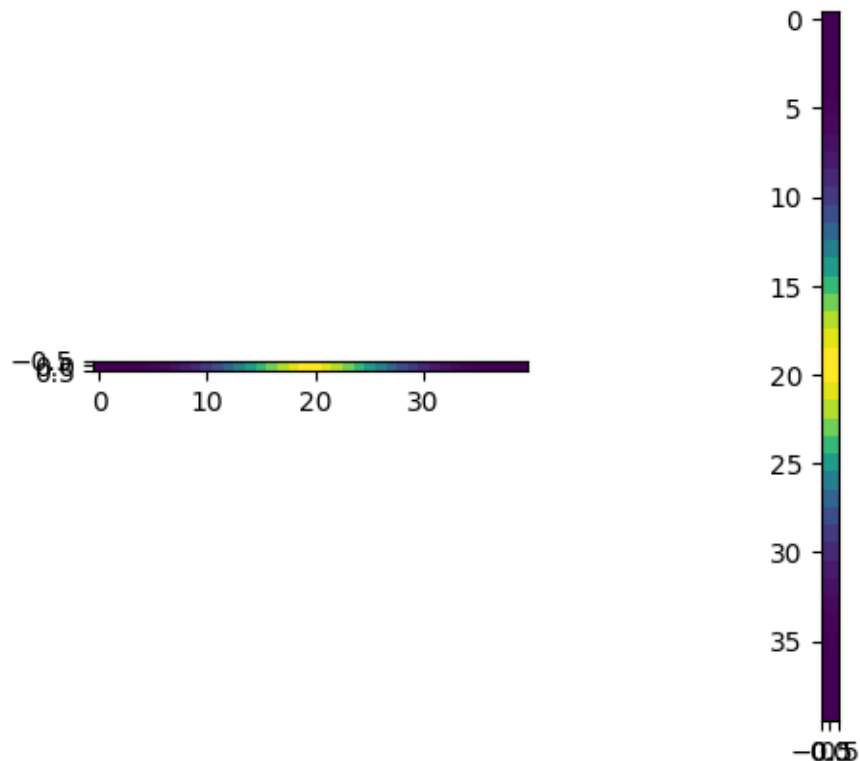
# The Gaussian filter along x-axis. Its shape is (1, sz).
### Insert your code ###
h_x = gaussian_filter_1d(sigma).reshape((1,40))

# The Gaussian filter along y-axis. Its shape is (sz, 1).
### Insert your code ###
h_y = h_x.transpose()

# Visualise the filters (provided)
plt.subplot(1, 2, 1)
plt.imshow(h_x)
plt.subplot(1, 2, 2)
plt.imshow(h_y)

```

[9]: <matplotlib.image.AxesImage at 0x130889570>



1.4.5 2.6 Perform Gaussian smoothing ($\sigma = 5$ pixels) using two separable filters and evaluate the computational time for separable Gaussian filtering. After that, perform Sobel filtering, show the gradient magnitude map and check whether it is the same as the previous one without separable filtering.

```
[10]: # Perform separable Gaussian smoothing and count time
#### Insert your code ####
start_time = time.time()
image_gaussian_x = scipy.signal.convolve2d(image_noisy, h_x)
image_gaussian = scipy.signal.convolve2d(image_gaussian_x, h_y)
end_time = time.time()
comp_time = end_time - start_time
print(comp_time)

# Image filtering
#### Insert your code ####
grad_x2 = scipy.signal.convolve2d(image_gaussian, sobel_x)
grad_y2 = scipy.signal.convolve2d(image_gaussian, sobel_y)

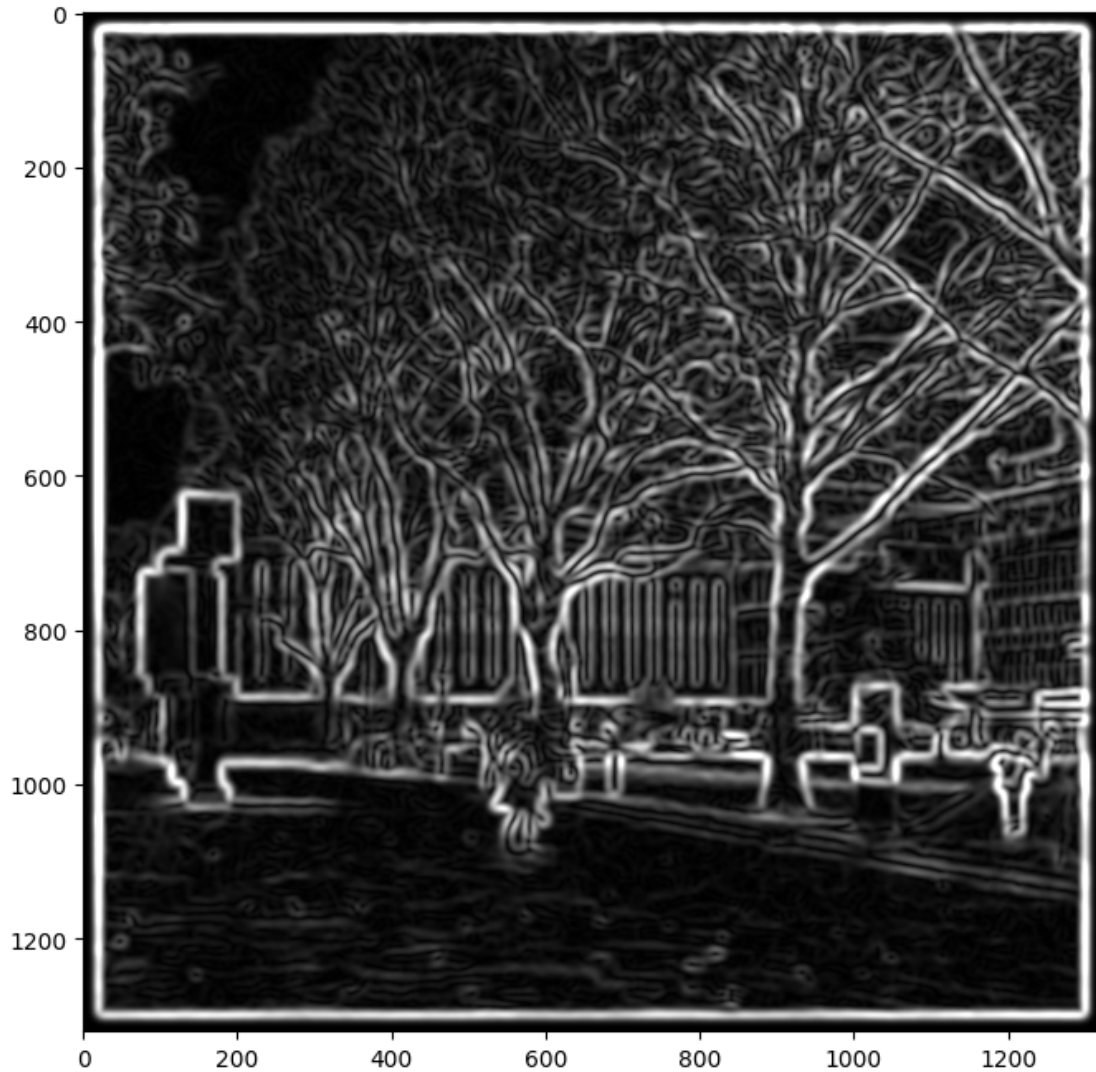
# Calculate the gradient magnitude
#### Insert your code ####
grad_mag2 = np.sqrt(np.square(grad_x2) + np.square(grad_y2))

# Display the gradient magnitude map (provided)
plt.imshow(grad_mag2, cmap='gray', vmin=0, vmax=100)
plt.gcf().set_size_inches(8, 8)

# Check the difference between the current gradient magnitude map
# and the previous one produced without separable filtering. You
# can report the mean difference between the two.
#### Insert your code ####
mean_abs_diff = (abs(grad_mag2 - grad_mag)).mean()
print(f"The mean absolute difference between the gradient map is:␣
↪{mean_abs_diff}.")
```

0.5552949905395508

The mean absolute difference between the gradient map is: 5.284452312498926e-13.



1.4.6 2.7 Comment on the Gaussian + Sobel filtering results and the computational time.

Insert your answer

Sobel's filtering itself outlines the edges in the image but it cannot remove the noise pixels while performing edge detection since there is colour discontinuity for areas with noise pixels and Sobel filters capture the discontinuity. Applying Gaussian filter before Sobel filtering smooths the image and thus suppresses the noise, but it also leads to a blurred edge detection result and may blur out weak edges.

In terms of computational time, performing Gaussian smoothing using two 1D Gaussian kernels takes significantly shorter time than using one 2D Gaussian kernel (0.536 secs compare to 5.537 secs on my machine). This is because separable filtering has a computational complexity of $O(N^2K)$ compare to $O(N^2K^2)$ where N is image size and K is filtering kernel size.

1.5 3. Challenge: Implement 2D image filters using Pytorch (24 points).

Pytorch is a machine learning framework that supports filtering and convolution.

The `Conv2D` operator takes an input array of dimension $N \times C_1 \times X \times Y$, applies the filter and outputs an array of dimension $N \times C_2 \times X \times Y$. Here, since we only have one image with one colour channel, we will set $N=1$, $C_1=1$ and $C_2=1$. You can read the documentation of `Conv2D` for more detail.

```
[11]: # Import libraries (provided)
import torch
```

1.5.1 3.1 Expand the dimension of the noisy image into $1 \times 1 \times X \times Y$ and convert it to a Pytorch tensor.

```
[12]: # Expand the dimension of the numpy array
#### Insert your code ####
image_hgt = image_noisy.shape[0]
image_len = image_noisy.shape[1]
image_noisy_4D = image_noisy.reshape((1,1,image_hgt, image_len))

# Convert to a Pytorch tensor using torch.from_numpy
#### Insert your code ####
image_noisy_4D_torch = torch.from_numpy(image_noisy_4D)

# x = image_noisy_4D_torch[0]
# image_noisy_permuted = x.permute(1,2,0)
# plt.imshow(image_noisy_permuted)
```

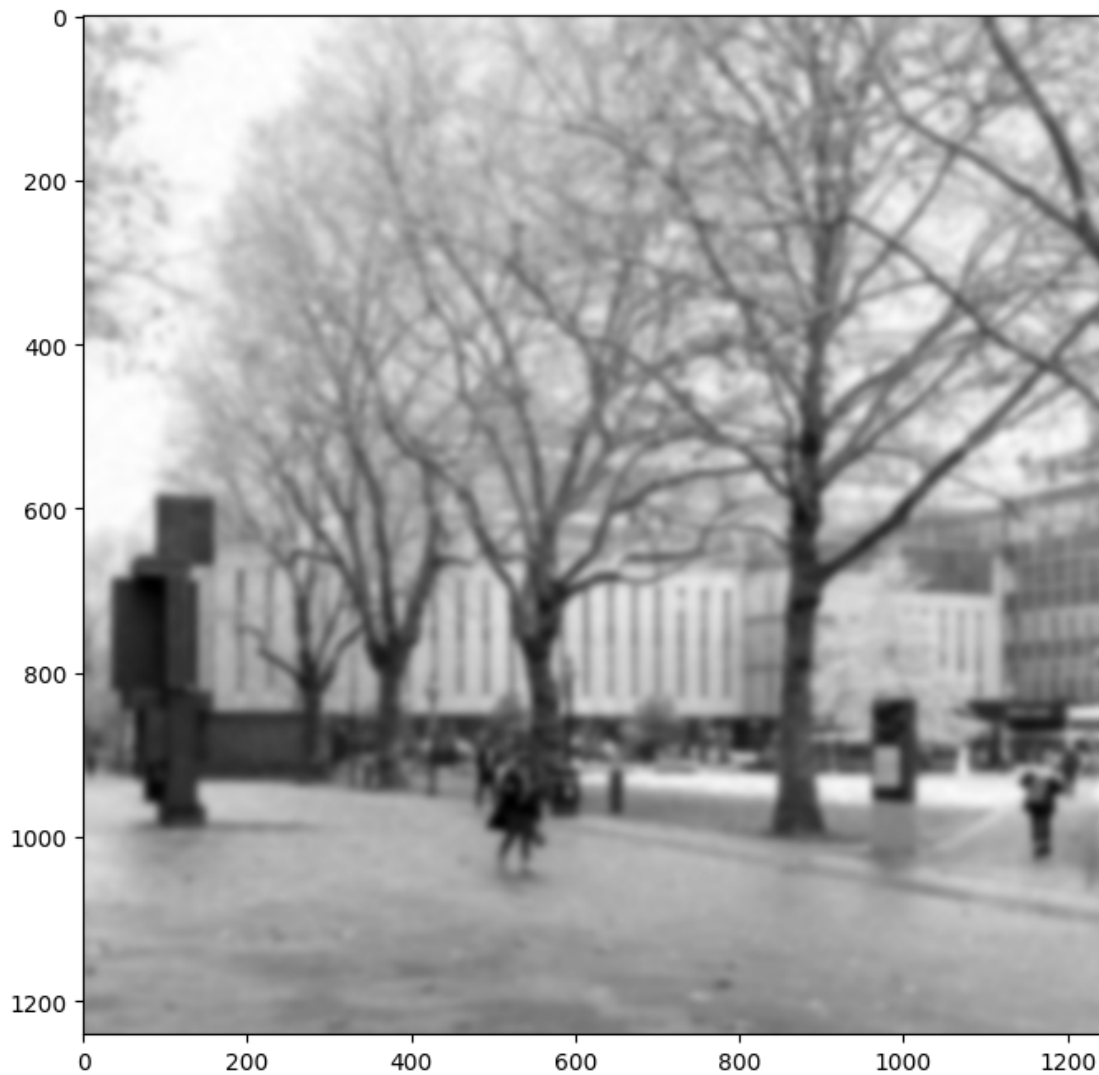
1.5.2 3.2 Create a Pytorch `Conv2D` filter, set its kernel to be a 2D Gaussian filter and perform filtering.

```
[13]: # A 2D Gaussian filter when sigma = 5 pixel (provided)
sigma = 5
h = gaussian_filter_2d(sigma)

# Create the Conv2D filter
#### Insert your code ####
h_reshaped = h.reshape((1,1, h.shape[0], h.shape[1]))
h_torch = torch.from_numpy(h_reshaped)

# Filtering
#### Insert your code ####
image_gaussian = torch.nn.functional.conv2d(image_noisy_4D_torch, h_torch)
image_filtered = image_gaussian[0].permute(1, 2, 0)
```

```
# Display the filtering result (provided)
plt.imshow(image_filtered, cmap='gray')
plt.gcf().set_size_inches(8, 8)
```



1.5.3 3.3 Implement Pytorch Conv2D filters to perform Sobel filtering on Gaussian smoothed images, show the gradient magnitude map.

```
[14]: # Create Conv2D filters
      ### Insert your code ###
      sobel_x = np.array([[1, 0, -1],[2, 0, -2],[1, 0, -1]])
      sobel_y = np.array([[1, 2, 1],[0, 0, 0],[-1, -2, -1]])
      sobel_x_4D = sobel_x.reshape((1,1,3,3))
      sobel_y_4D = sobel_y.reshape((1,1,3,3))
```

```

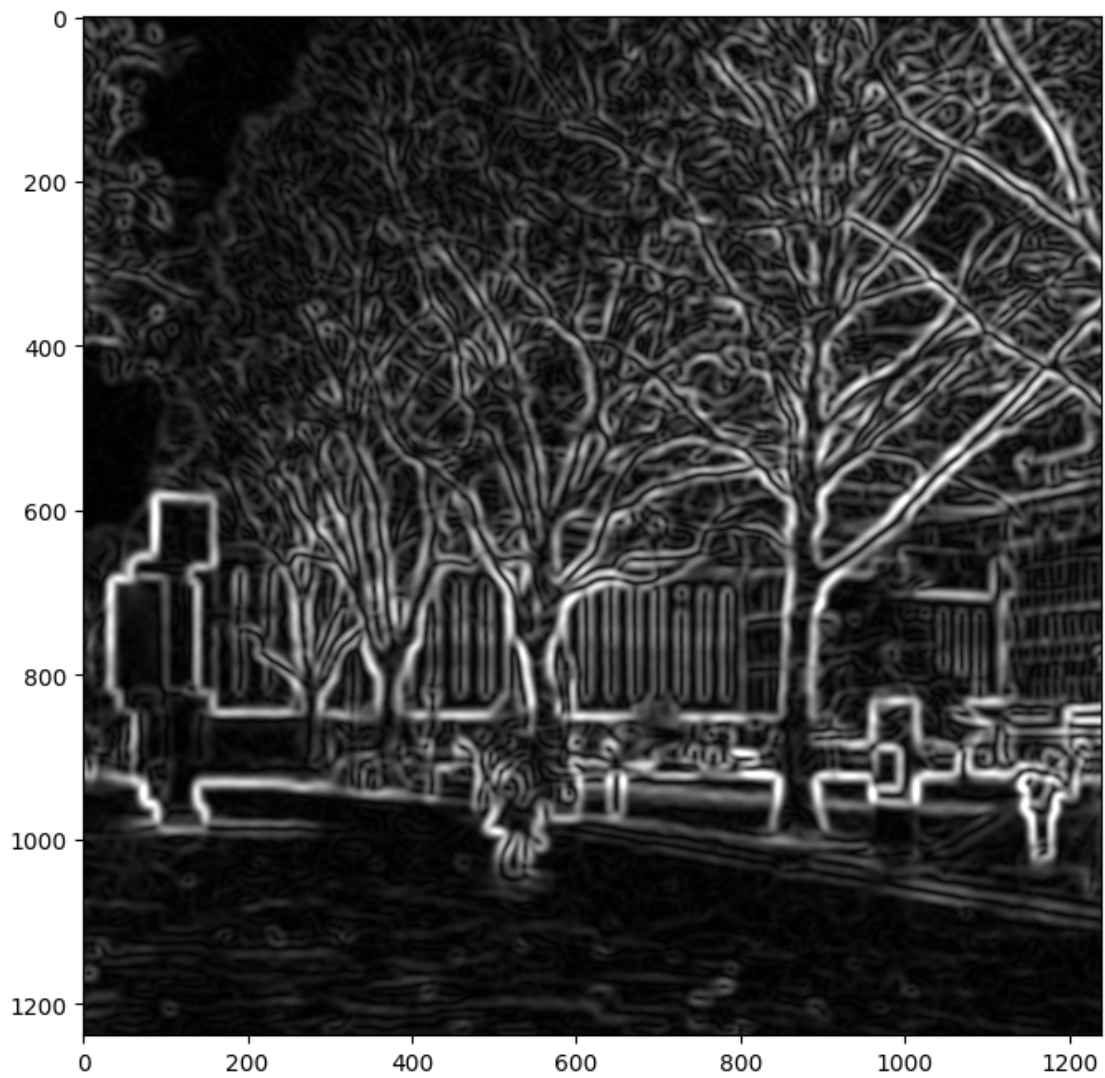
sobel_x_torch = torch.from_numpy(sobel_x_4D).double()
sobel_y_torch = torch.from_numpy(sobel_y_4D).double()

# Perform filtering
### Insert your code ###
image_sobel_x = torch.nn.functional.conv2d(image_gaussian, sobel_x_torch)
image_sobel_y = torch.nn.functional.conv2d(image_gaussian, sobel_y_torch)

# Calculate the gradient magnitude map
### Insert your code ###
grad_mag_torch = torch.sqrt(torch.square(image_sobel_x) + torch.
    ↪square(image_sobel_y))
grad_mag3 = grad_mag_torch[0].permute(1,2,0)

# Visualise the gradient magnitude map (provided)
plt.imshow(grad_mag3, cmap='gray', vmin=0, vmax=100)
plt.gcf().set_size_inches(8, 8)

```



[]: