coursework_01

January 29, 2025

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]`

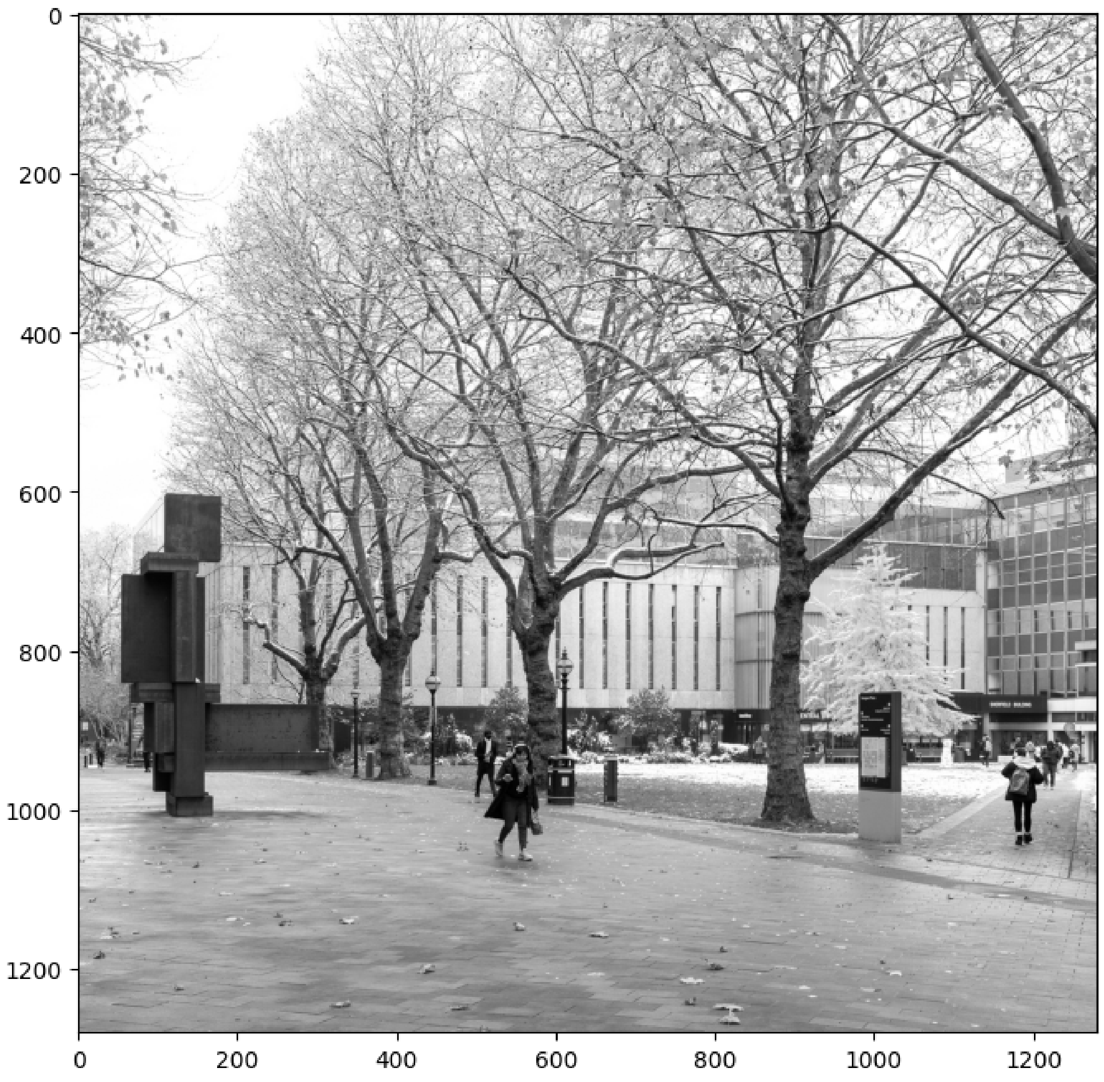
[1]:
```python
# Import libaries (provided)
import imageio.v3 as imageio
import numpy as np
import matplotlib.pyplot as plt
import noise
import scipy
import scipy.signal
import math
import time
```

1.3 1. Moving average filter (20 points).

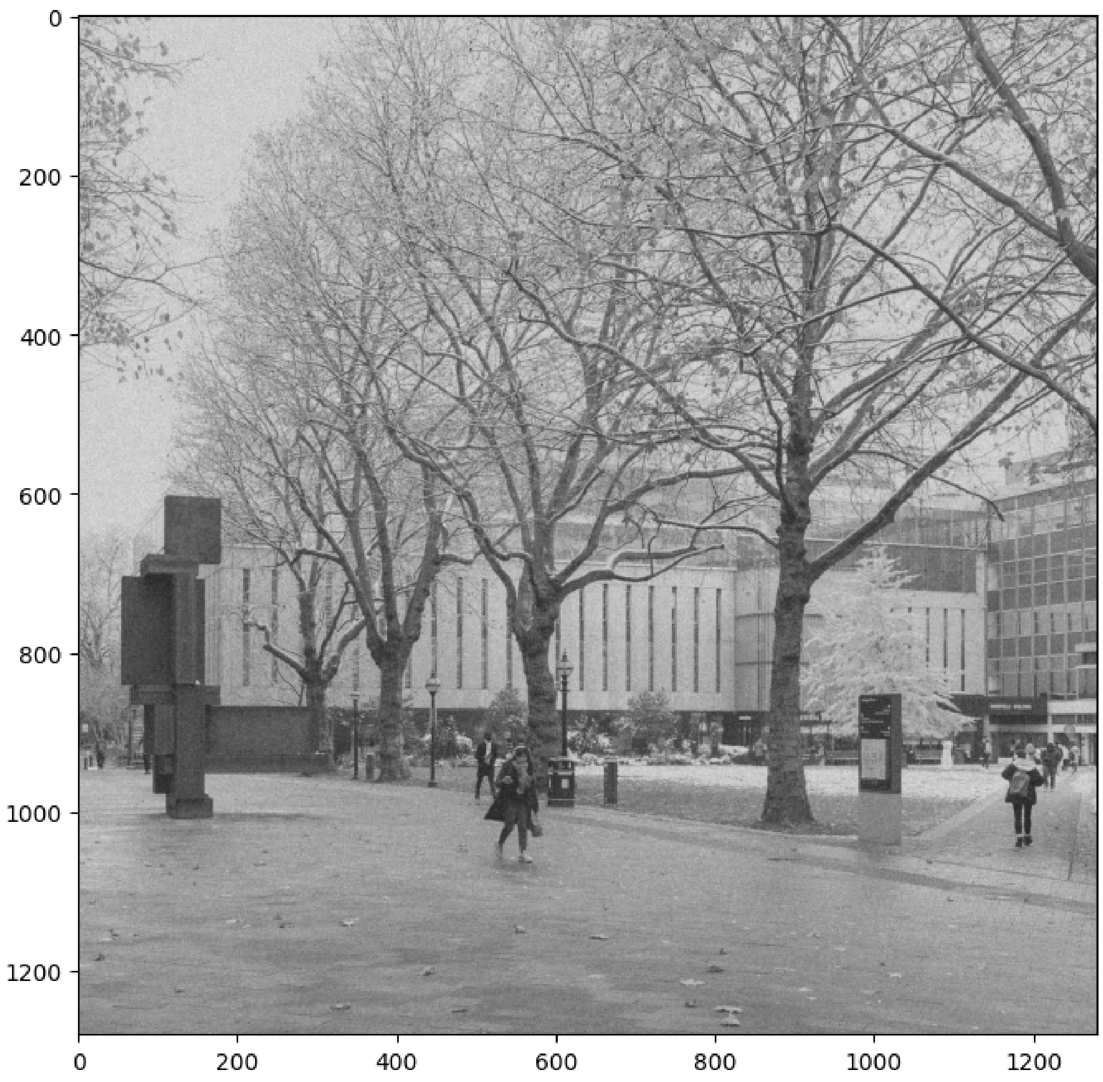
Read the provided input image, add noise to the image and design a moving average filter for denoising.

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

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



[3]:
```python
# 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)
```



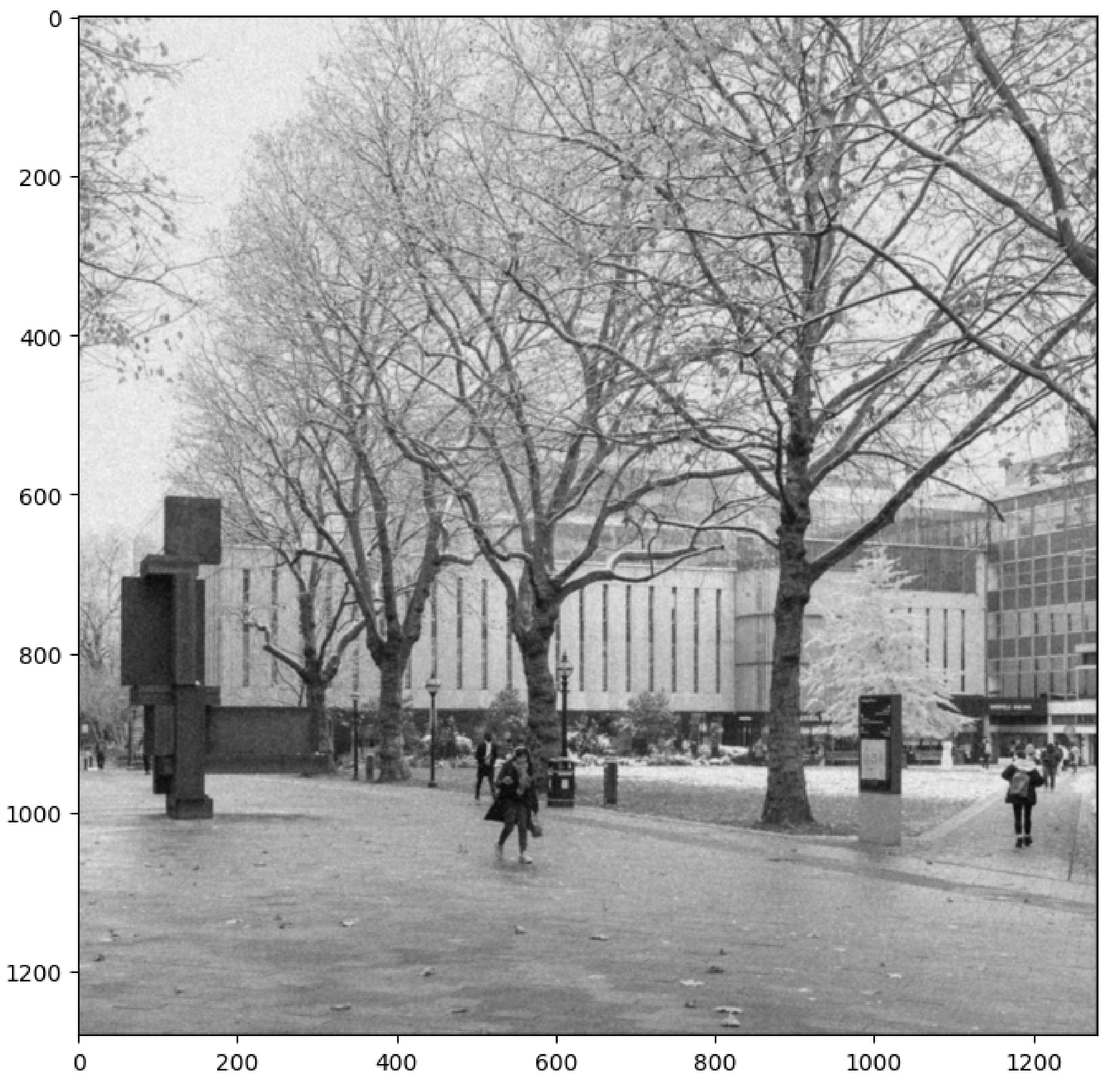
1.3.1 Note: from now on, please use the noisy image as the input for the filters.

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

[4]:
```python
# Design the filter h
### Insert your code ###
h = np.ones((3, 3)) /9
h1 = np.array([[0, 0, 0], [1, 1, 1], [0, 0, 0]]) / 3
```

```python
h2 = np.array([[0, 1, 0], [0, 1, 0], [0, 1, 0]]) / 3

# Convolve the corrupted image with h using scipy.signal.convolve2d function
### Insert your code ###
image_filtered_temp = scipy.signal.convolve2d(image_noisy, h1, mode='same')
image_filtered = scipy.signal.convolve2d(image_filtered_temp, h2, mode='same')

# 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]]
```

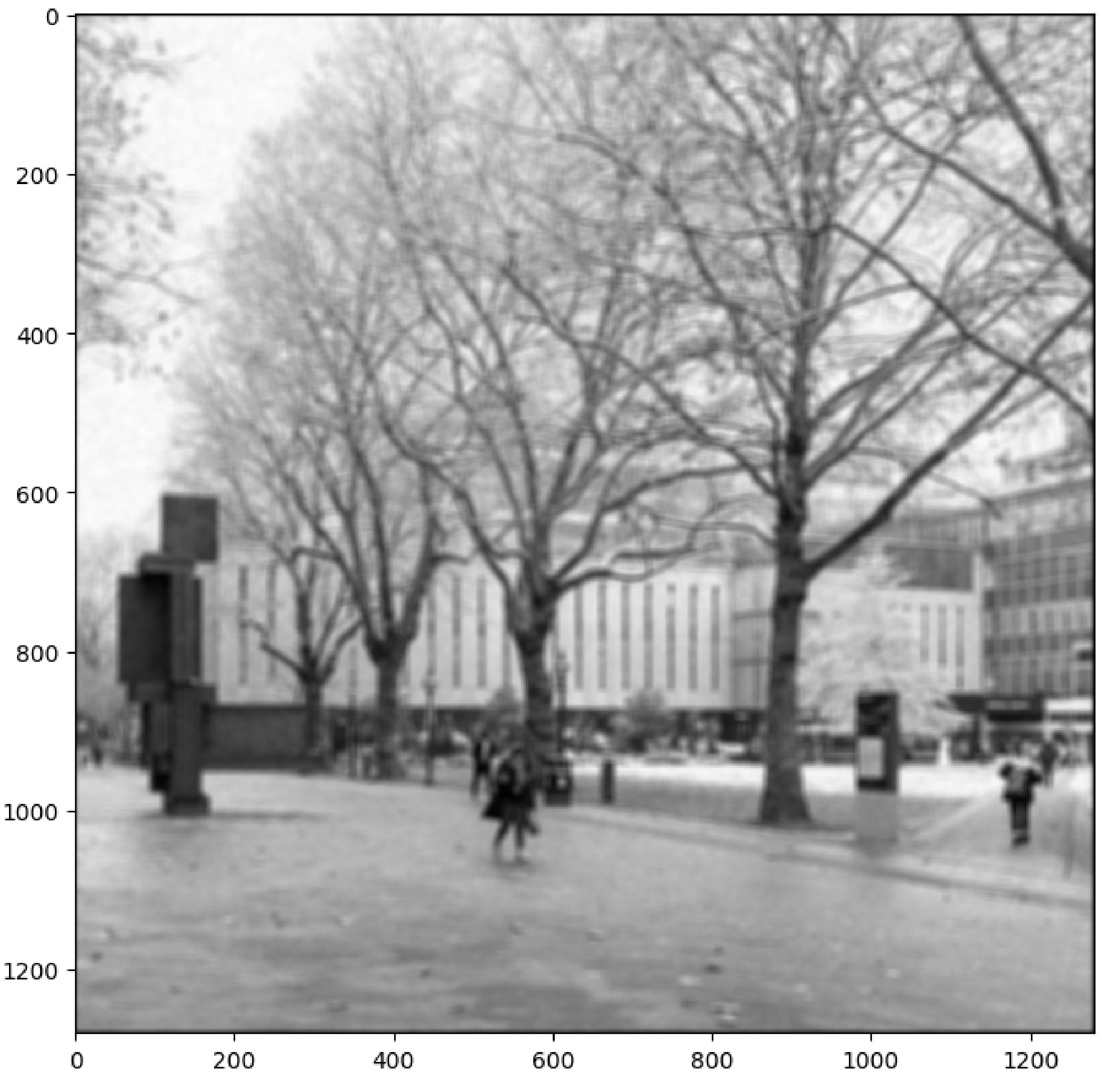


1.3.3 1.2 Filter the noisy image with a 11x11 moving average filter.

[5]:
```
# Design the filter h
### Insert your code ###
h = np.ones((11, 11)) / (11**2)

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

# 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?

As the kernel size is increased, a larger window size is considered and avereged for each pixel. Thus, the effect of this is greater smoothing of the image, resulting in a more blurry image. Furthermore, with 0-same padding we are getting a border around the image with a larger kernel size as this slowly fades to black.

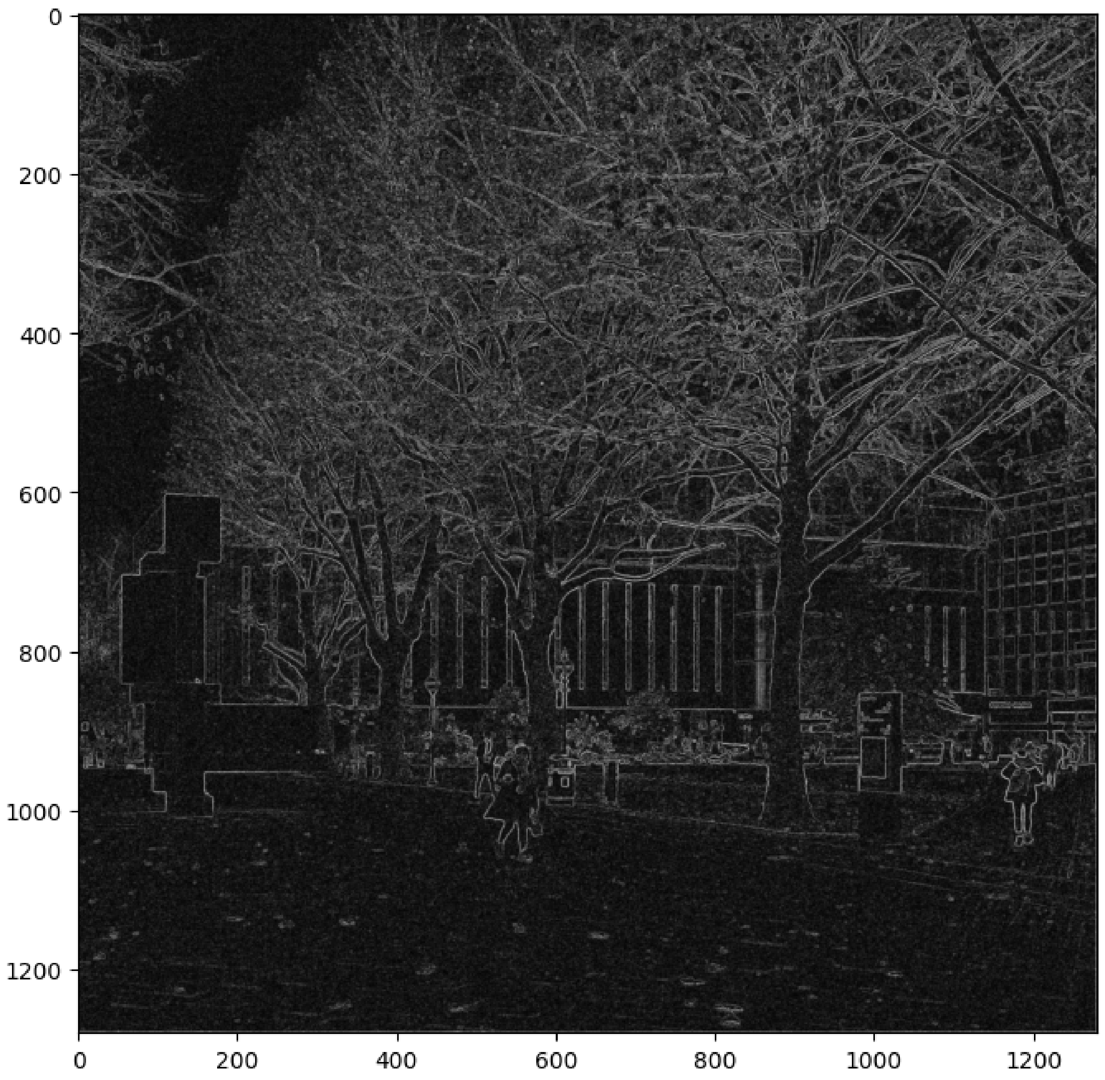
1.4 2. Edge detection (56 points).

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
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
image_filtered_x = scipy.signal.convolve2d(image_noisy, sobel_x, mode='same')
image_filtered_y = scipy.signal.convolve2d(image_noisy, sobel_y, mode='same')

# Calculate the gradient magnitude
grad_mag = np.sqrt(image_filtered_x**2 + image_filtered_y**2) * 255

# 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)
```

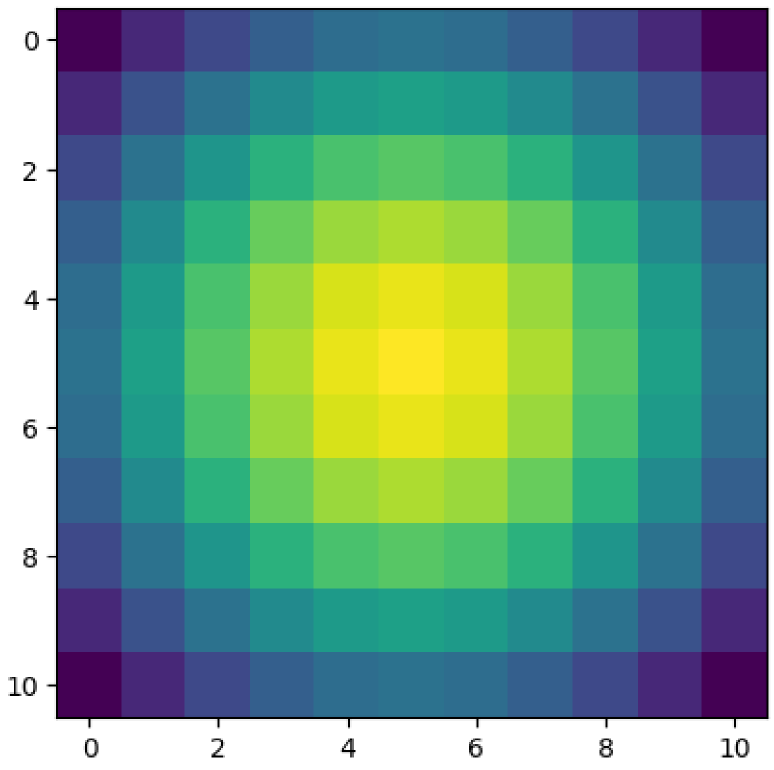
```
sobel_x:
[[ 1  0 -1]
 [ 2  0 -2]
 [ 1  0 -1]]
sobel_y:
[[ 1  2  1]
 [ 0  0  0]
 [-1 -2 -1]]
```



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

[7]:
```python
# 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
    size = 2 * sigma + 1
    h = np.zeros((size, size))
    for i in range(size):
        for j in range(size):
            h[i, j] = np.exp(-((i-sigma)**2 + (j-sigma)**2) / (2*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 0x2547c6af470>
```

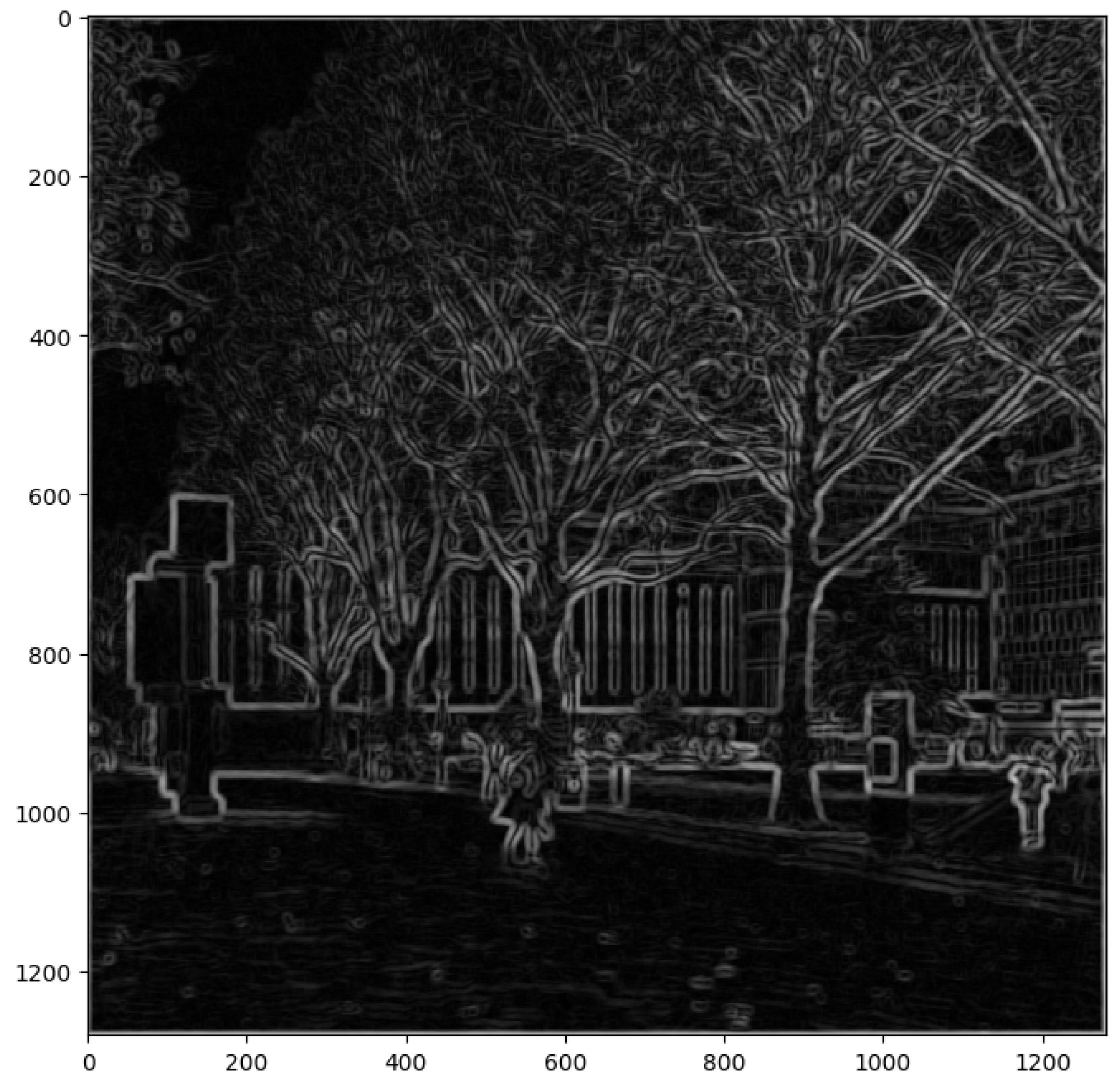


1.4.3 2.3 Perform Gaussian smoothing ($\sigma = 5$ pixels) and evaluate the computational time for Gaussian smoothing. After that, perform Sobel filtering and show the gradient magintude map.

[8]:
```
# Construct the Gaussian filter
sigma = 5
h = gaussian_filter_2d(sigma)

# Perform Gaussian smoothing and count time
start_time = time.time()
image_filtered = scipy.signal.convolve2d(image_noisy, h, mode='same')
end_time = time.time()
```

```
print('Time for Gaussian smoothing: {:.2f} seconds'.format(end_time -␣
 ↪start_time))

# Image filtering
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_filtered_x = scipy.signal.convolve2d(image_filtered, sobel_x, mode='same')
image_filtered_y = scipy.signal.convolve2d(image_filtered, sobel_y, mode='same')

# Calculate the gradient magnitude
grad_mag = np.sqrt(image_filtered_x**2 + image_filtered_y**2)
grad_mag = grad_mag / grad_mag.max() * 255

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

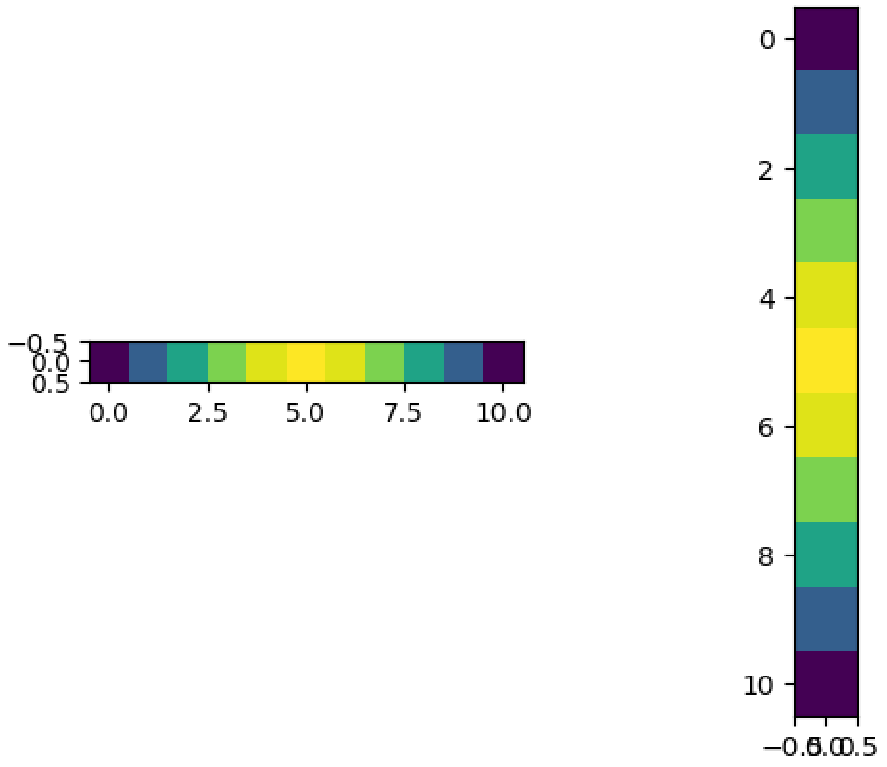
```
Time for Gaussian smoothing: 0.29 seconds
```



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 ###
    size = 2 * sigma + 1
    h = np.zeros((1, size))
    for i in range(size):
        h[0][i] = np.exp(-((i-sigma)**2) / (2*sigma**2))
    return h

# sigma = 5 pixel (provided)
sigma = 5

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

# The Gaussian filter along y-axis. Its shape is (sz, 1).
h_y = np.transpose(gaussian_filter_1d(sigma))

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

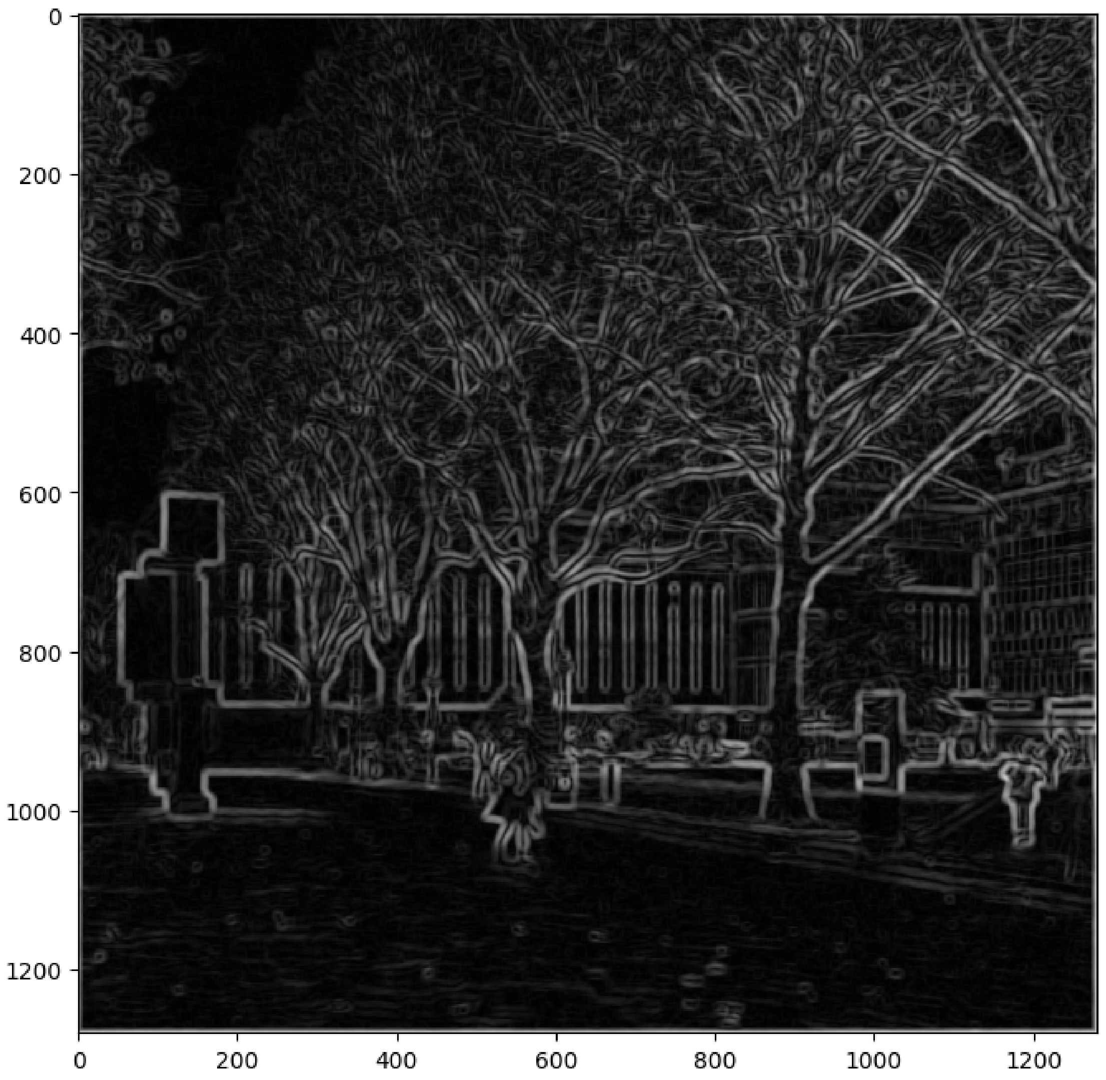
[9]:
```
<matplotlib.image.AxesImage at 0x2547ceef650>
```



1.4.5 2.5 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]:
```python
# Perform separable Gaussian smoothing and count time
start_time = time.time()
image_filtered_temp = scipy.signal.convolve2d(image_noisy, h_x, mode='same')
image_filtered = scipy.signal.convolve2d(image_filtered_temp, h_y, mode='same')
end_time = time.time()
print('Time for separable Gaussian smoothing: {:.2f} seconds'.format(end_time -␣
 ↪start_time))

# Image filtering
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_filtered_x = scipy.signal.convolve2d(image_filtered, sobel_x, mode='same')
image_filtered_y = scipy.signal.convolve2d(image_filtered, sobel_y, mode='same')

# Calculate the gradient magnitude
grad_mag2 = np.sqrt(image_filtered_x**2 + image_filtered_y**2)
grad_mag2 = grad_mag2 / grad_mag2.max() * 255

# 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.
diff = np.mean(np.abs(grad_mag - grad_mag2))
print('Mean difference between the two gradient magnitude maps: {:.2f}'.
 ↪format(diff))
```

```
Time for separable Gaussian smoothing: 0.12 seconds
Mean difference between the two gradient magnitude maps: 0.00
```



1.4.6 2.6 Comment on the Gaussian + Sobel filtering results and the computational time.

For a significant decrease in computational time (about 3 times), we see absolutely no difference in the filter results from changing the kernel from a 3x3 kernel to using 2 kernels of size 1x3 and 3x1. This is due to them being mathematically the same kernel due to the properties of the convolution operator and allowing us to seperate them. This allows us to decrease our computation complexity from O((N$^{2)(K}$2)) to O((N^2)(K))

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 NxC1xXxY, applies the filter and outputs an array of dimension NxC2xXxY. Here, since we only have one image with one colour channel, we will set N=1, C1=1 and C2=1. You can read the documentation of Conv2D for more detail.

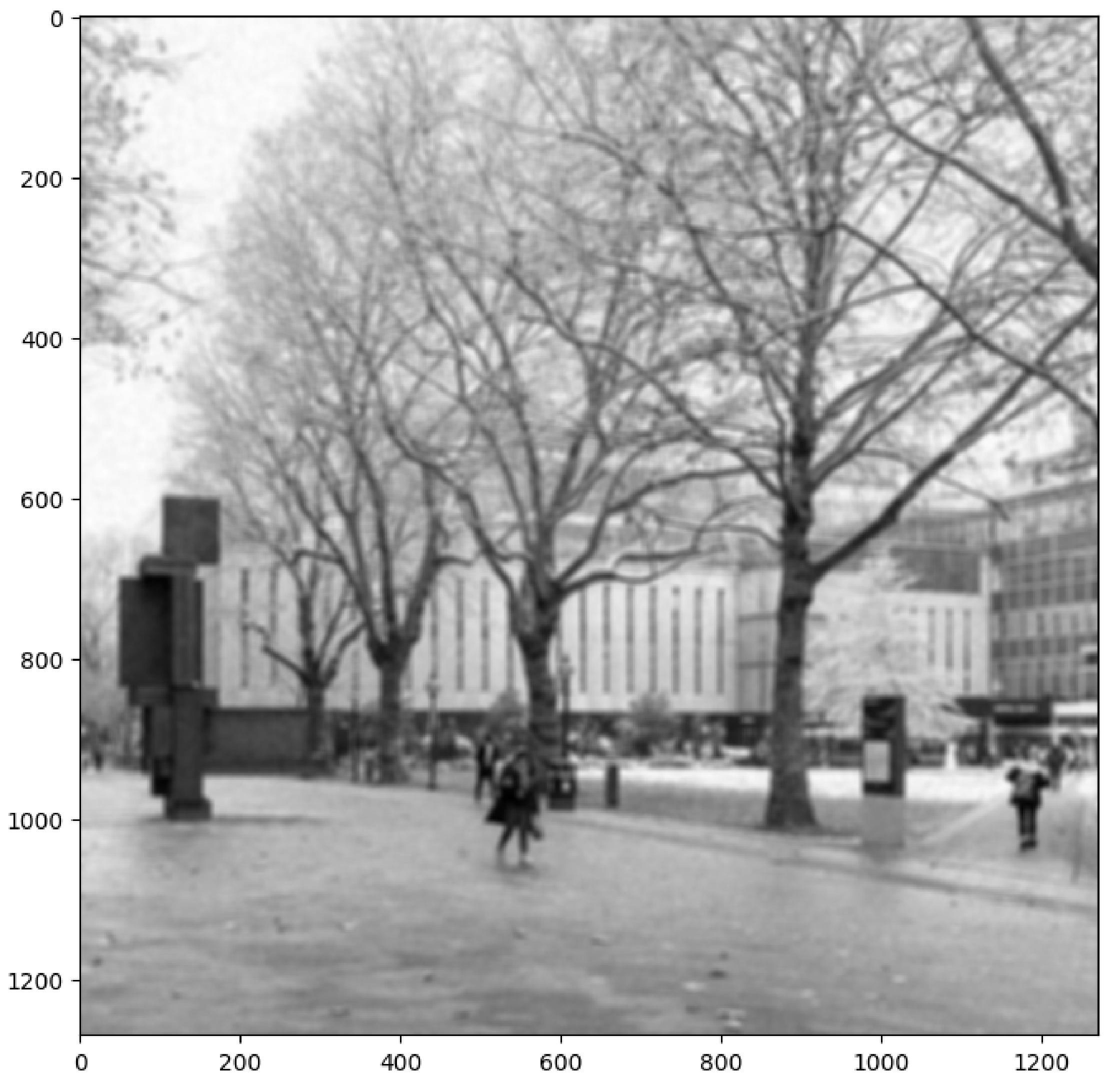
[11]:
```
# Import libaries (provided)
import torch
```

1.5.1 3.1 Expand the dimension of the noisy image into 1x1xXxY and convert it to a Pytorch tensor.

[12]:
```
# Expand the dimension of the numpy array
extended_image = np.reshape(image_noisy, (1, 1) + image_noisy.shape)

# Convert to a Pytorch tensor using torch.from_numpy
torch_tensor = torch.from_numpy(extended_image)
```

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

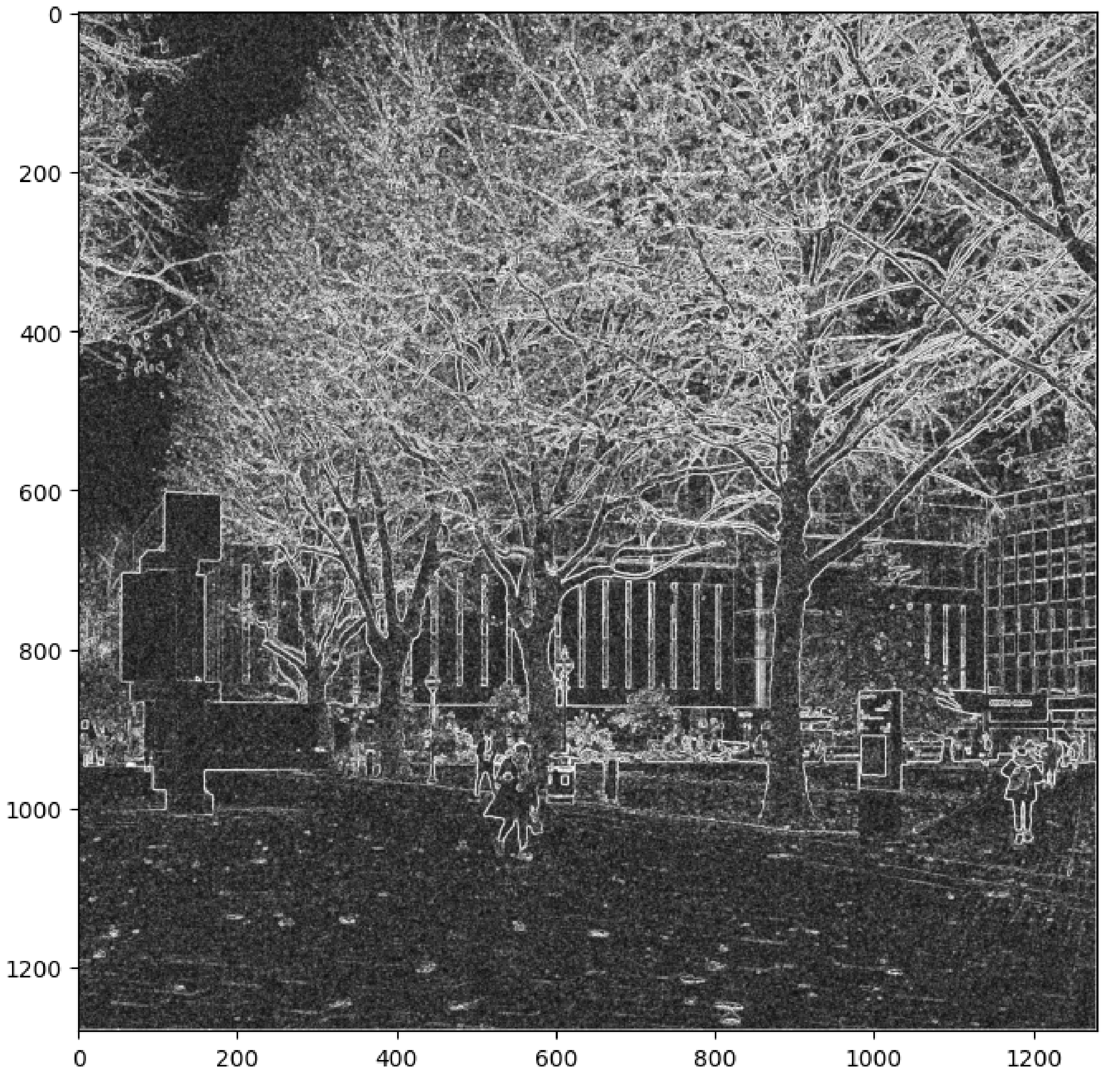
[13]:
```
# A 2D Gaussian filter when sigma = 5 pixel (provided)
sigma = 5
h = gaussian_filter_2d(sigma)

# Create the Conv2D filter
conv2d = torch.nn.Conv2d(1, 1, kernel_size=(h.shape[0], h.shape[1]), padding=0,␣
 ↪bias=False)

# Filtering
conv2d.weight.data = torch.from_numpy(h).float().unsqueeze(0).unsqueeze(0)
image_filtered = conv2d(torch_tensor.float())
image_filtered = image_filtered.squeeze().detach().numpy()

# 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
conv2d_x = torch.nn.Conv2d(1, 1, kernel_size=(3, 3), padding=1, bias=False)
conv2d_y = torch.nn.Conv2d(1, 1, kernel_size=(3, 3), padding=1, bias=False)

# Perform filtering
conv2d_x.weight.data = torch.from_numpy(sobel_x).float().unsqueeze(0).
 ↪unsqueeze(0)
conv2d_y.weight.data = torch.from_numpy(sobel_y).float().unsqueeze(0).
 ↪unsqueeze(0)
image_filtered_x = conv2d_x(torch_tensor.float()).squeeze().detach().numpy()
```

```
image_filtered_y = conv2d_y(torch_tensor.float()).squeeze().detach().numpy()


# Calculate the gradient magnitude map
grad_mag3 = np.sqrt(image_filtered_x**2 + image_filtered_y**2)
grad_mag3 = grad_mag3 / grad_mag3.max() * 255

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



```
[ ]:
```