coursework-01-1

February 1, 2024

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]`

[45]:
```
# Google colab filesys mounting and changing current dir
from google.colab import drive
drive.mount('/content/drive')
%cd /content/drive/MyDrive/'Colab Notebooks'/'Computer_Vision_2024'/
 ↪coursework_01

# 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
```

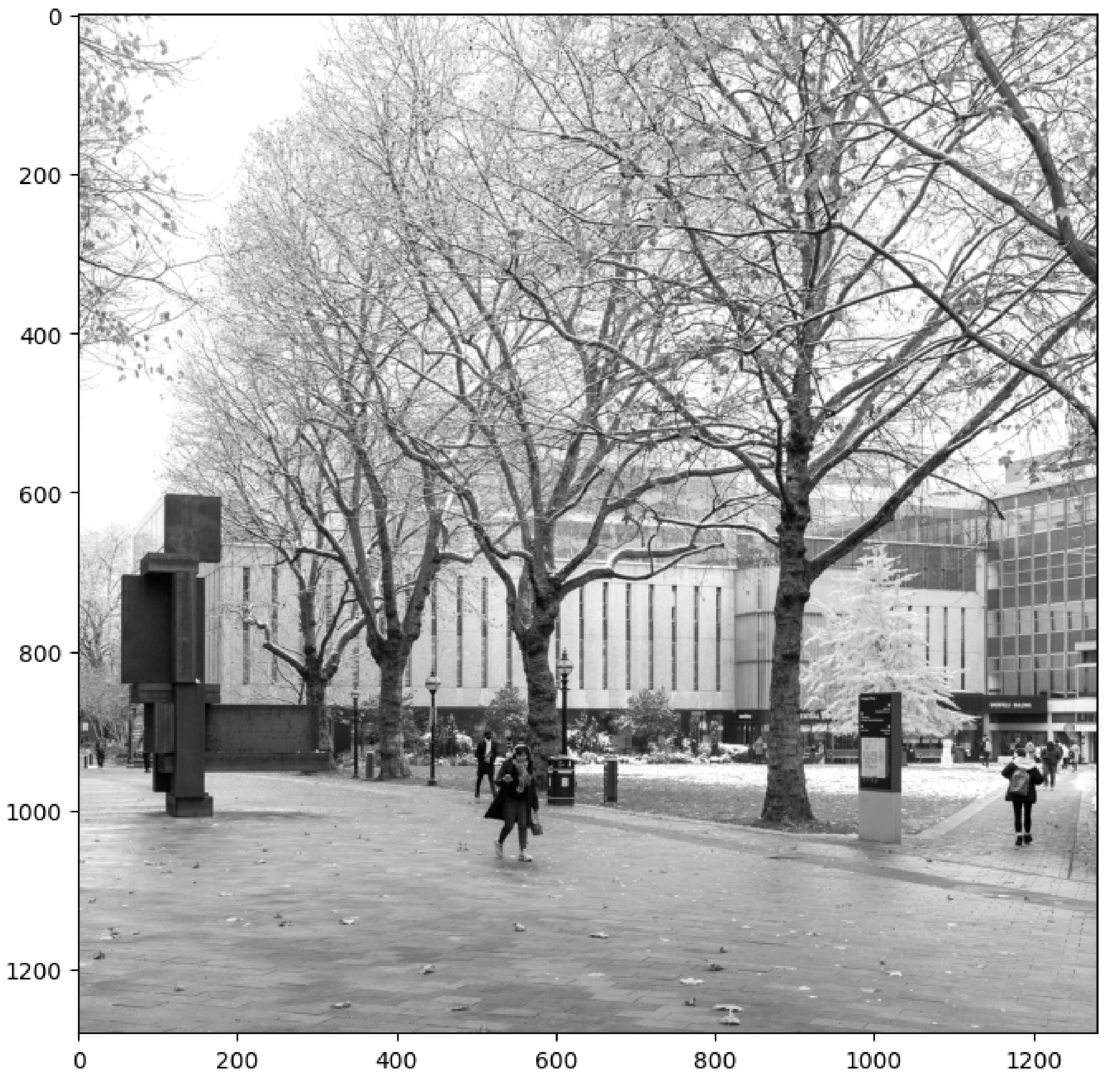
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
/content/drive/MyDrive/Colab Notebooks/Computer_Vision_2024/coursework_01
```

1.3 1. Moving average filter (20 points).

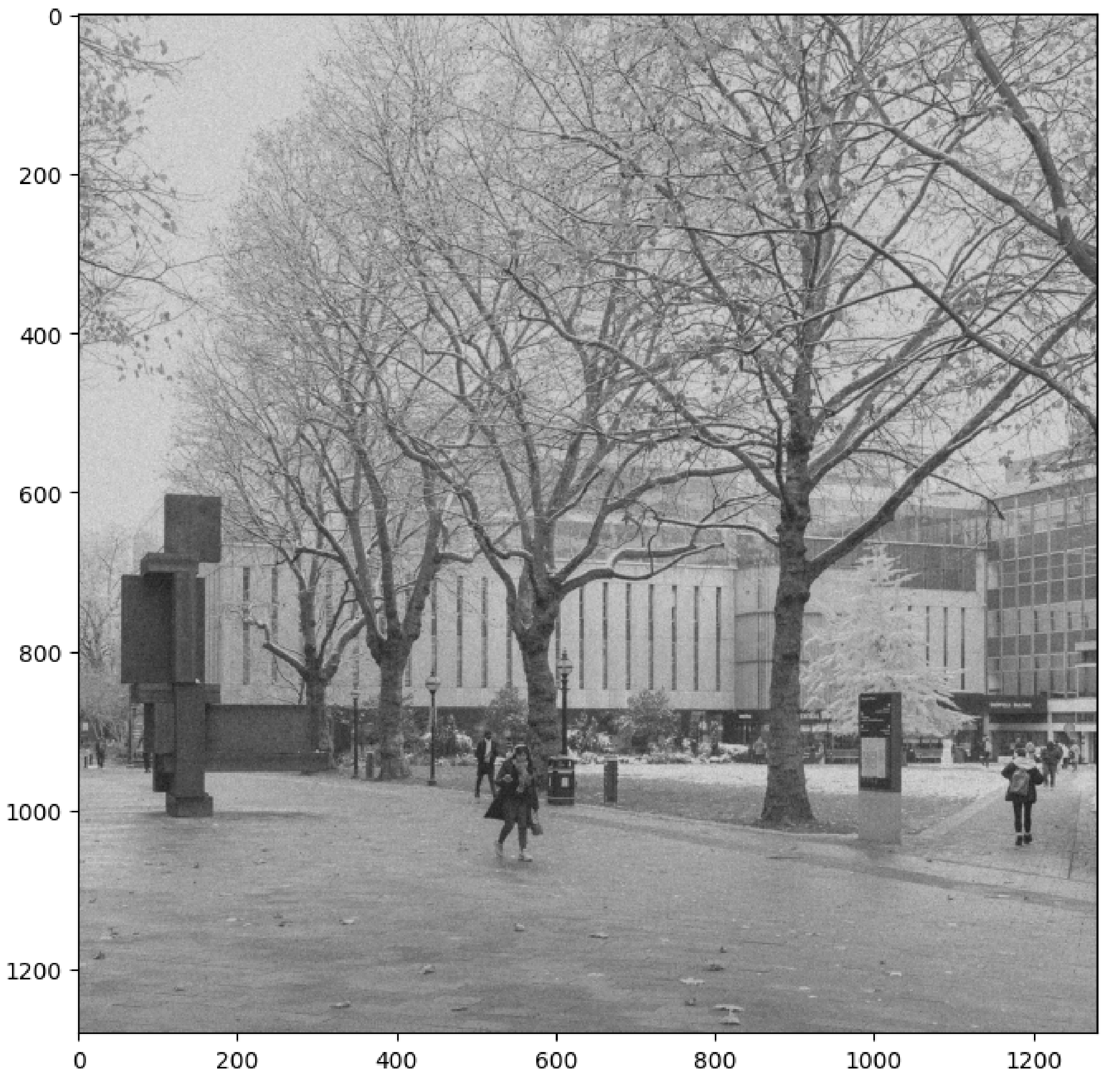
Read the provided input image, add noise to the image and design a moving average filter for denoising.

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

[46]:
```
# Read the image (provided)
image = imageio.imread('campus_snow.jpg')
plt.imshow(image, cmap='gray')
plt.gcf().set_size_inches(8, 8)
```



[47]:
```
# 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)
```



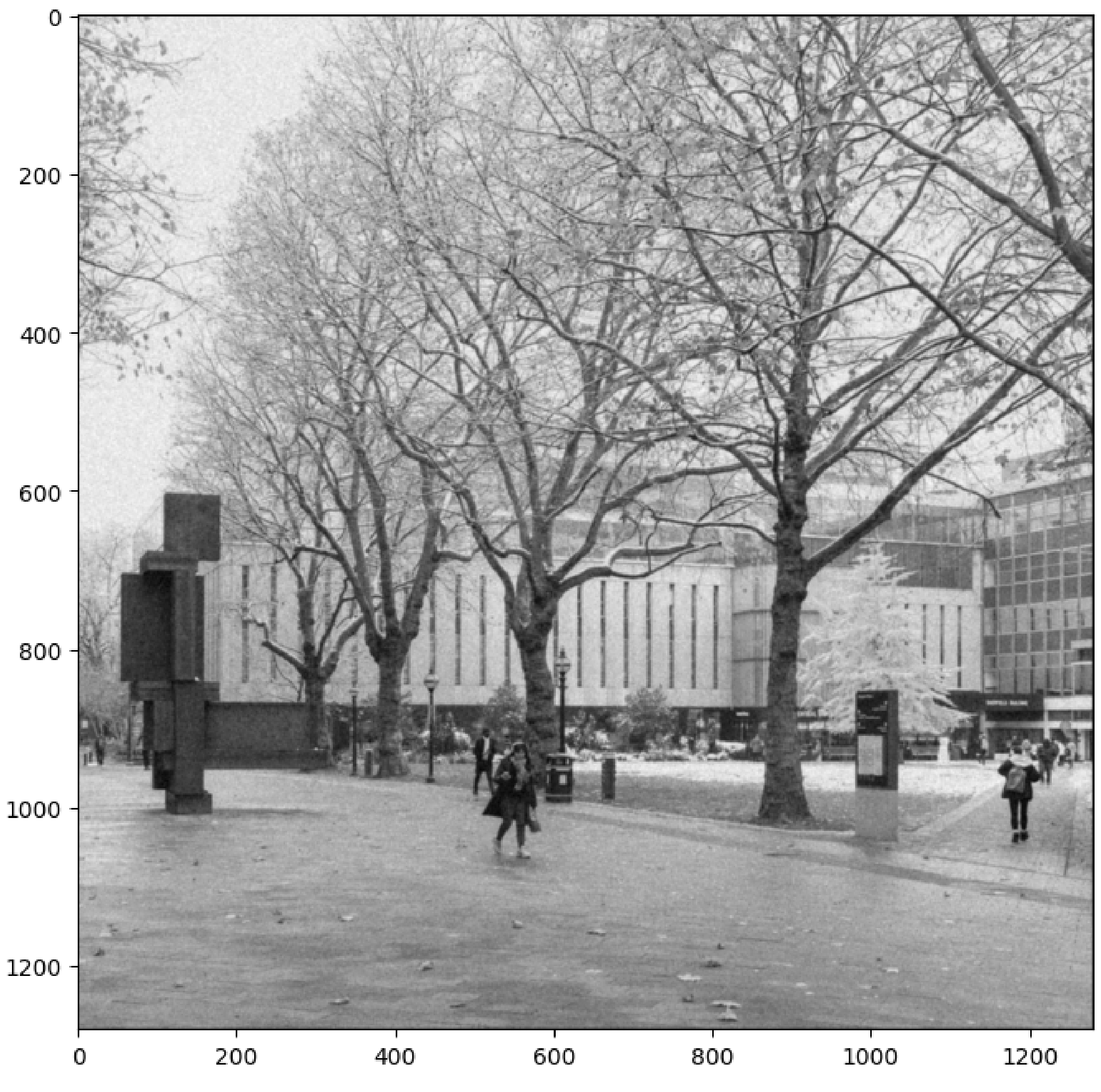
1.3.1 Note: from now on, please use the noisy image as the input for the filters.

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

[48]:
```
# Design the filter h
### Insert your code ###
h = [[1/9]*3]*3

# 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)
```

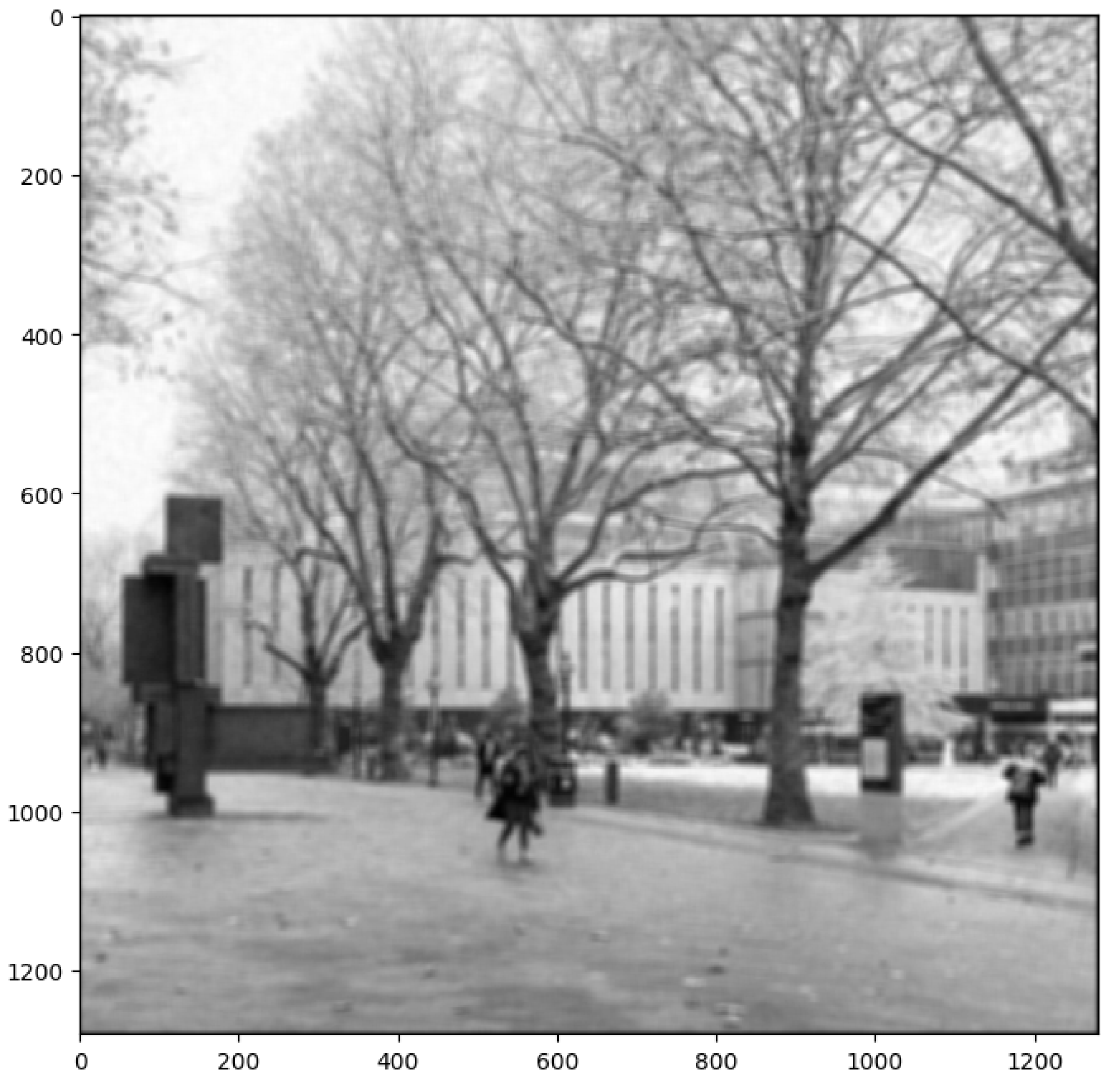
```
Filter h:
[[0.1111111111111111, 0.1111111111111111, 0.1111111111111111],
[0.1111111111111111, 0.1111111111111111, 0.1111111111111111],
[0.1111111111111111, 0.1111111111111111, 0.1111111111111111]]
```



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

[50]:
```
# Design the filter h
### Insert your code ###
h = [[1/11**2]*11]*11

# 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:
[[0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356], [0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356], [0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356],
[0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356], [0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356], [0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356],
[0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356], [0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356], [0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356],
[0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356], [0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356]]
```



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

### Insert your answer ### A larger kernel gives a smoother image since noise is suppressed more. However, it also leads to a more blurry image since high-value cells are more smoothed out (as they are set to the average of a larger number of surrounding cells).

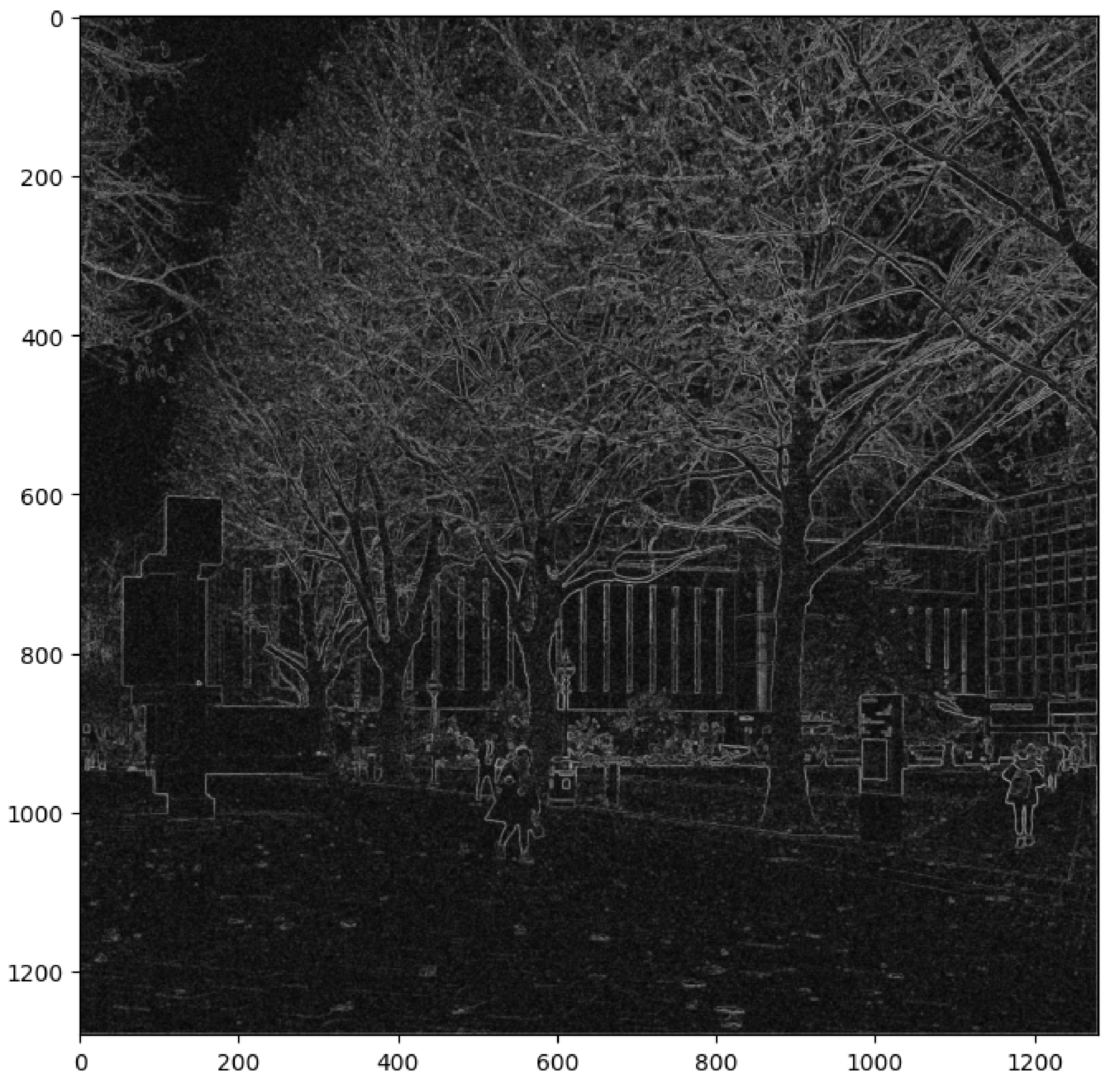
1.4 2. Edge detection (56 points).

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.

[51]:
```python
# Design the filters
### Insert your code ###
sobel_x = [[1,0,-1],[2,0,-2],[1,0,-1]]
sobel_y = [[1,2,1],[0,0,0],[-1,-2,-1]]

# Image filtering
### Insert your code ###
filtered_sobel_x = scipy.signal.convolve(image_noisy, sobel_x, mode="same")
filtered_sobel_y = scipy.signal.convolve(image_noisy, sobel_y, mode="same")


# Calculate the gradient magnitude
### Insert your code ###
grad_mag = (filtered_sobel_x**2+filtered_sobel_y**2)**0.5

# 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)
```

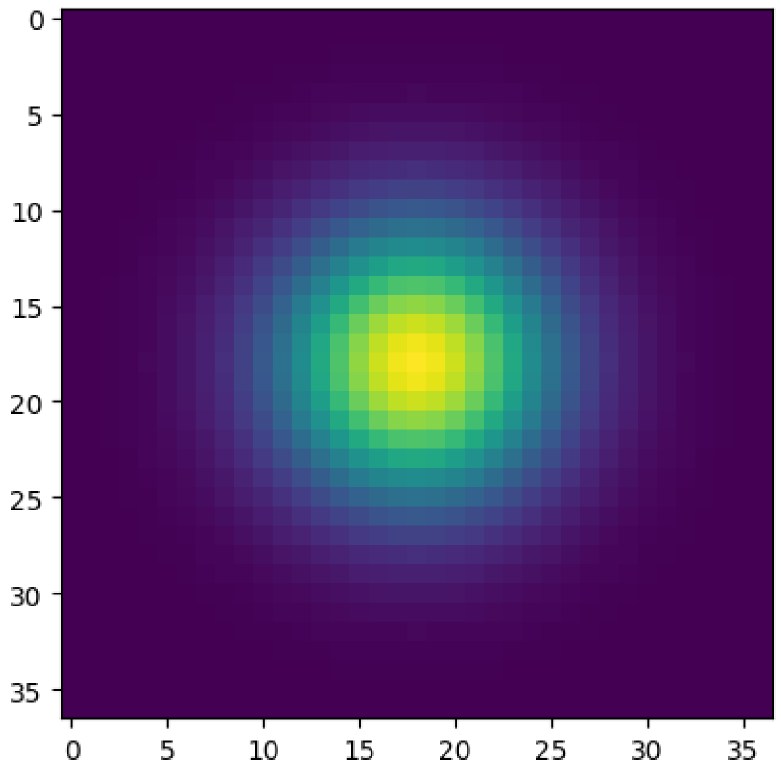
```
sobel_x:
[[1, 0, -1], [2, 0, -2], [1, 0, -1]]
sobel_y:
[[1, 2, 1], [0, 0, 0], [-1, -2, -1]]
```



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

[52]:
```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

    ### Insert your code ###
    # Filter radius is 3.5 * sigma
    radius = int(math.ceil(3.5 * sigma))
    size = 2 * radius + 1
    h=[]
    for i in range(size):
        h.append([])
        for j in range(size):
            x = math.exp((-(i-radius)**2-(j-radius)**2)/(2*sigma**2))/(2*math.
 ↪pi*sigma**2)
            h[i].append(x)
    return h

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

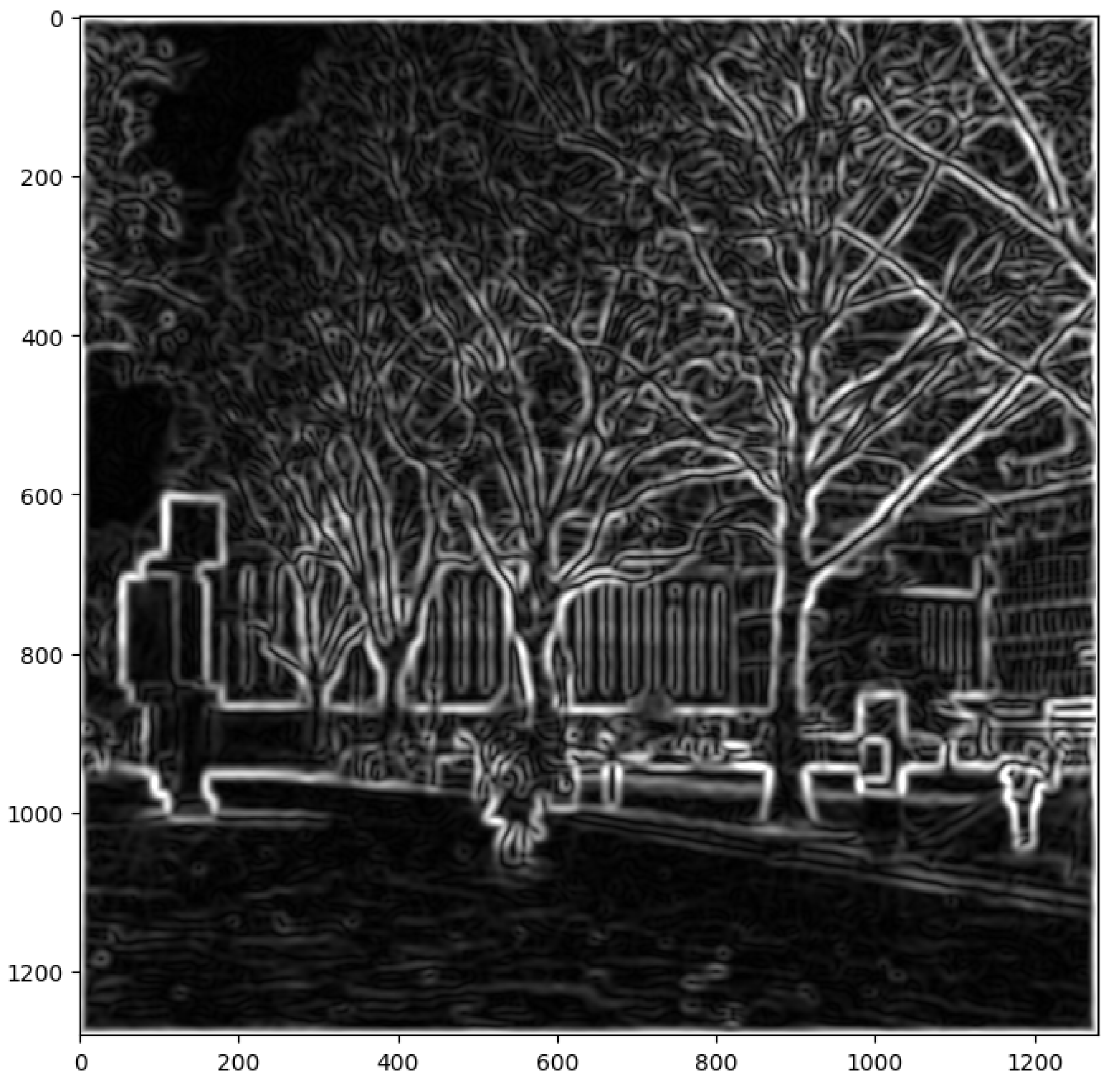
[52]:
```
<matplotlib.image.AxesImage at 0x7b8652b0f640>
```



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.

[56]:
```
# Construct the Gaussian filter
### Insert your code ###
sigma = 5
h = gaussian_filter_2d(sigma)

# Perform Gaussian smoothing and count time
### Insert your code ###
start = time.time()
gaussian_smoothed_image = scipy.signal.convolve2d(image_noisy, h, mode='same')
duration_non_separable = time.time() - start
print('Compute time for Gaussian smoothing with one filter (no separable␣
 ↪filters) is {0:.6f} seconds.'.format(duration_non_separable))

# Image filtering
### Insert your code ###
filtered_sobel_x = scipy.signal.convolve2d(gaussian_smoothed_image, sobel_x,␣
 ↪mode='same')
filtered_sobel_y = scipy.signal.convolve2d(gaussian_smoothed_image, sobel_y,␣
 ↪mode='same')

# Calculate the gradient magnitude
### Insert your code ###
grad_mag1 = (filtered_sobel_x**2 + filtered_sobel_y**2)**0.5

# Display the gradient magnitude map (provided)
plt.imshow(grad_mag1, cmap='gray', vmin=0, vmax=100)
plt.gcf().set_size_inches(8, 8)
```

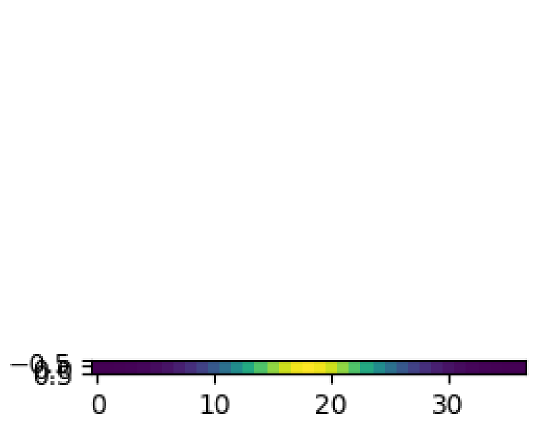
```
Compute time for Gaussian smoothing with one filter (no separable filters) is
7.034118 seconds.
```

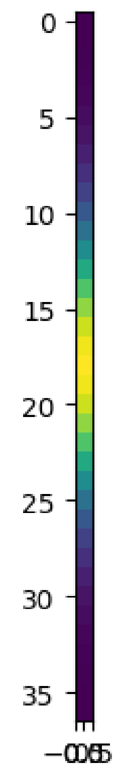


1.4.4 2.4 Implement a function that generates a 1D Gaussian filter given the parameter σ. Generate 1D Gaussian filters along x-axis and y-axis respectively.

[54]:
```
# 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

    # Filter radius is 3.5 * sigma
    radius = int(math.ceil(3.5 * sigma))
    size = 2 * radius + 1
    h = []
    for i in range(-radius,radius+1):
        x = math.exp(-(i**2)/(2*sigma**2))/(math.sqrt(2*math.pi)*sigma)
        h.append(x)
    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)
h_x = np.expand_dims(h_x, axis=0)

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

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

```
[54]: <matplotlib.image.AxesImage at 0x7b865290a7a0>
```



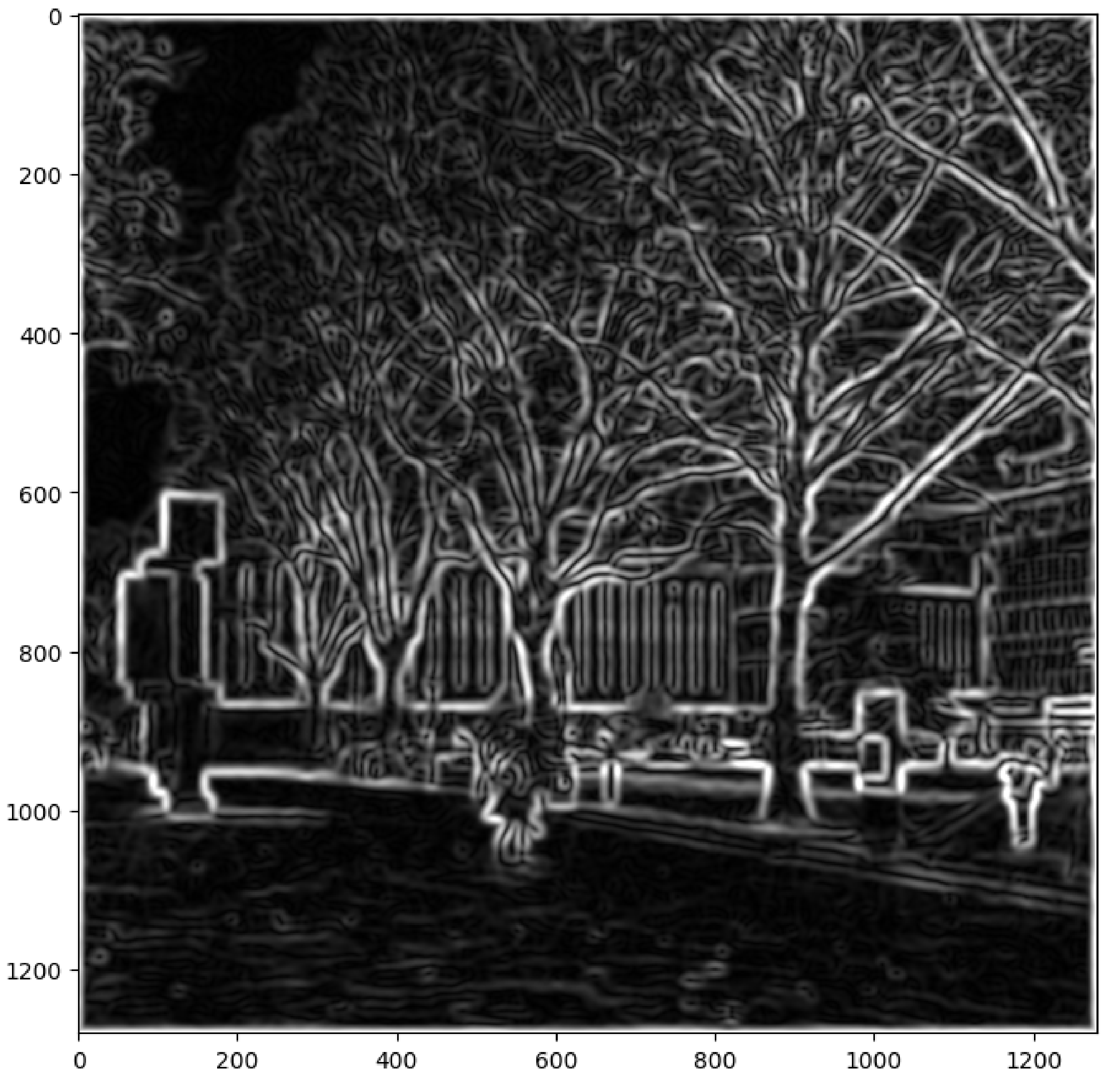


1.4.5 2.6 Perform Gaussian smoothing ($\sigma = 5$ pixels) using two separable filters and evaluate the computational time for separable Gaussian filtering. After that, perform Sobel filtering, show the gradient magnitude map and check whether it is the same as the previous one without separable filtering.

[55]:
```python
# Perform separable Gaussian smoothing and count time
### Insert your code ###
start = time.time()
image_smoothed = scipy.signal.convolve2d(image_noisy, h_x, mode='same')
image_smoothed = scipy.signal.convolve2d(image_smoothed, h_y, mode='same')
duration_separable = time.time() - start
print('Compute time for Gaussian smoothing with separable filters is {0:.6f}␣
 ↪seconds.'.format(duration_separable))

# Image filtering
### Insert your code ###
filtered_sobel_x = scipy.signal.convolve2d(image_smoothed, sobel_x, mode='same')
filtered_sobel_y = scipy.signal.convolve2d(image_smoothed, sobel_y, mode='same')

# Calculate the gradient magnitude
### Insert your code ###
grad_mag2 = (filtered_sobel_x**2 + filtered_sobel_y**2)**0.5
```

```
# 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_difference = np.mean(np.abs(grad_mag1 - grad_mag2))
print('The mean difference between the current gradient magnitude map and␣
 ↪previous one is {0:.10f}.'.format(mean_difference))
```

```
Compute time for Gaussian smoothing with separable filters is 0.735762 seconds.
The mean difference between the current gradient magnitude map and previous one
is 0.0000000000.
```



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

Gaussian smoothing aims to eliminate noise in the image. Sobel filters are used to detect edges. Using one filter for Gaussian smoothing takes 7.034118 seconds, whereas using two separate filters (one for x-axis and one for y-axis) takes 0.735762 seconds. Separate gaussian filters result in much shorter computational time.

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.

[57]:
```
# Import libaries (provided)
import torch
```

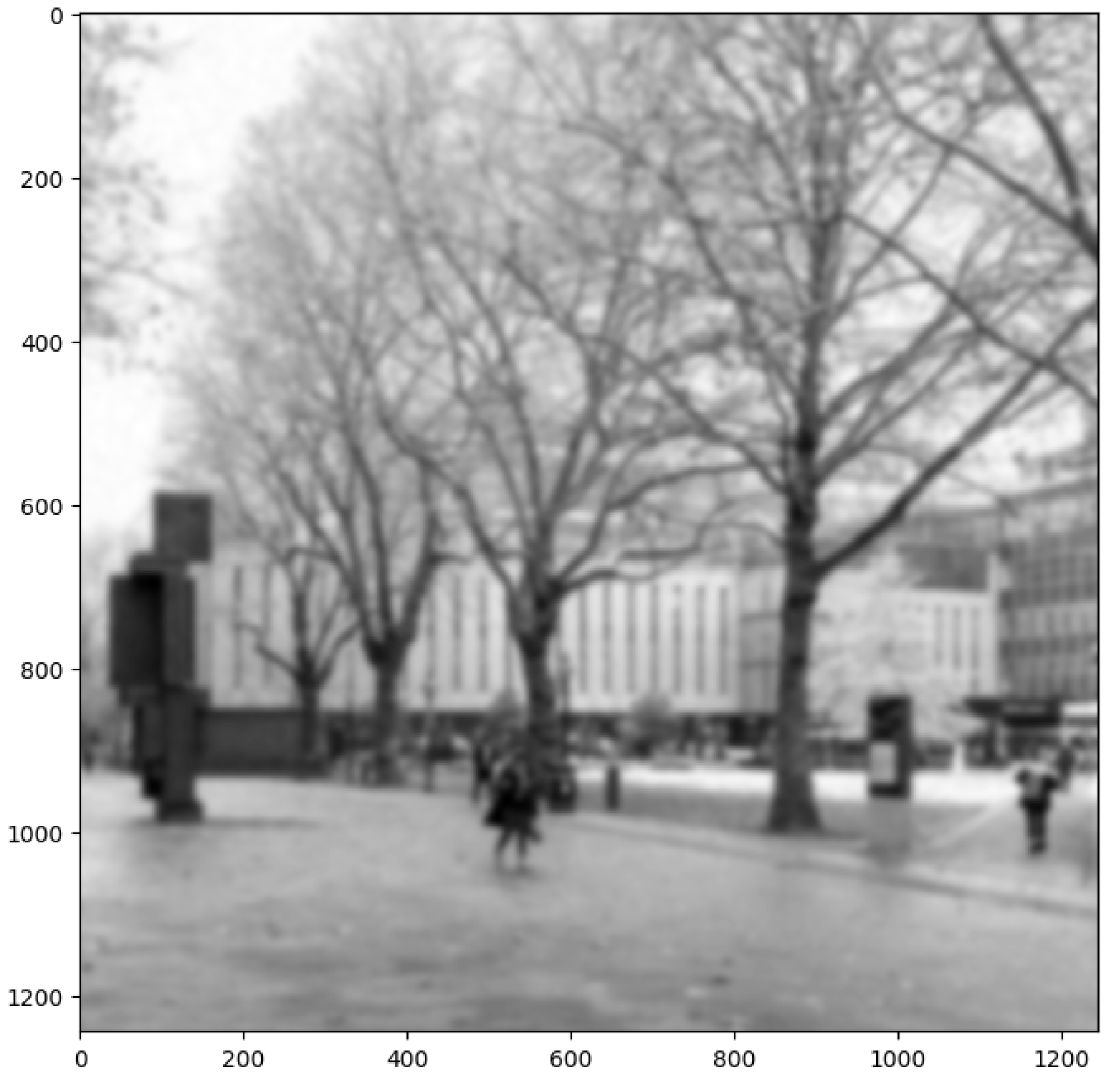
1.5.1 3.1 Expand the dimension of the noisy image into 1x1xXxY and convert it to a Pytorch tensor.

[58]:
```
# Expand the dimension of the numpy array
expanded_image_noisy = np.expand_dims(np.expand_dims(image_noisy, axis=0),␣
 ↪axis=0)
# Convert to a Pytorch tensor using torch.from_numpy
tensor_image = torch.from_numpy(expanded_image_noisy).float()
```

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

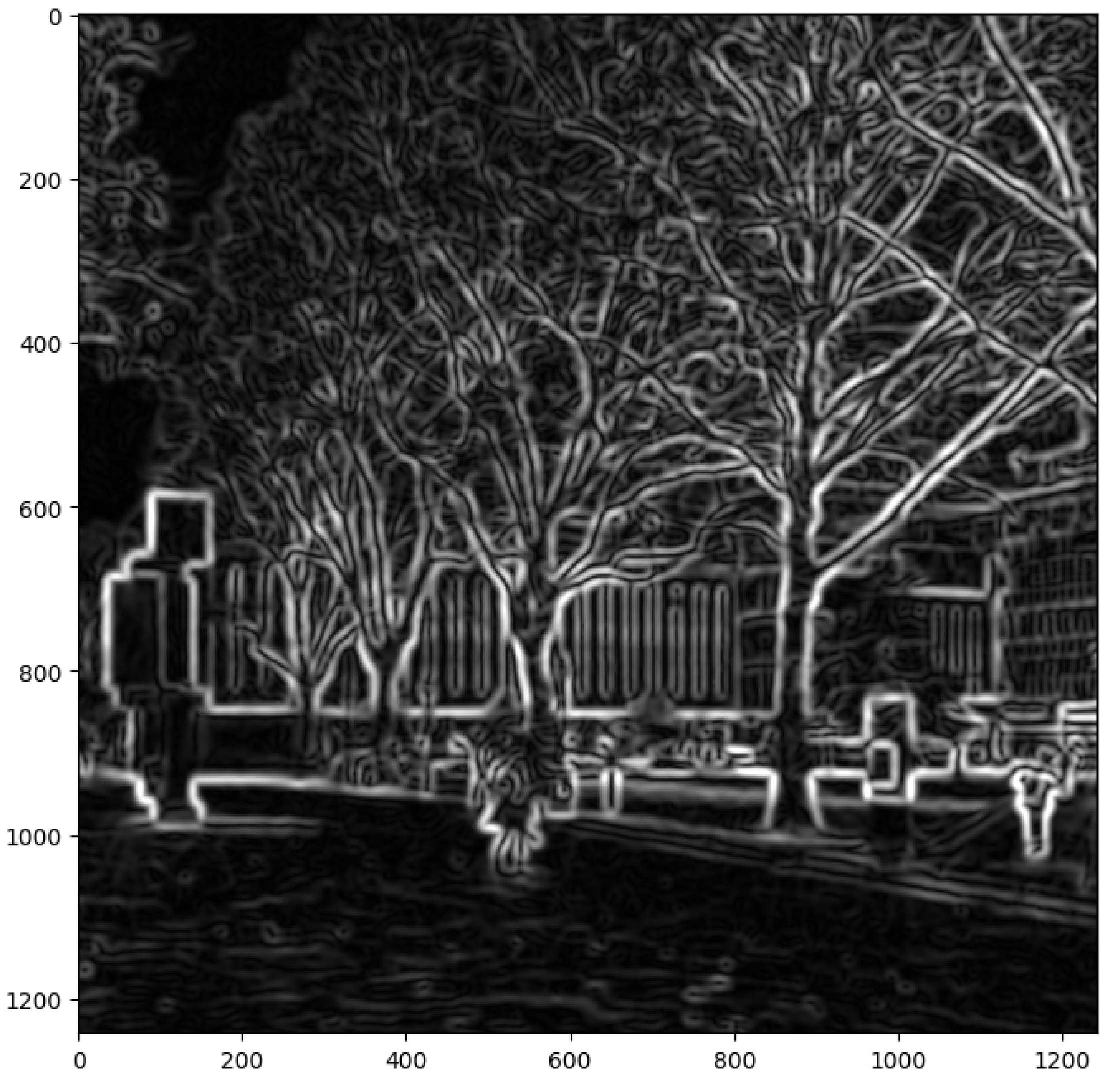
[59]:
```
# A 2D Gaussian filter when sigma = 5 pixel (provided)
sigma = 5
h = gaussian_filter_2d(sigma)

# Create the Conv2D filter
np_array_h = np.array(h)
sum_h = np.sum(np_array_h)
normalised_h = np_array_h / sum_h
tensor_h = torch.FloatTensor(normalised_h)
conv2d_gaussian_filter = torch.nn.Conv2d(in_channels=1, out_channels=1,␣
 ↪kernel_size=normalised_h.shape, bias=False)

expanded_tensor_h = tensor_h.expand(conv2d_gaussian_filter.weight.size())
```

```
conv2d_gaussian_filter.weight = torch.nn.Parameter(expanded_tensor_h,␣
↪requires_grad = False)

# Filtering
image_gaussian_filtered_tensor = conv2d_gaussian_filter(tensor_image)
image_filtered = image_gaussian_filtered_tensor.squeeze(0).squeeze(0).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.

[60]:
```
# Create Conv2D filters

# Sobel_x Conv2D filter
np_array_sobel_x = np.array(sobel_x)
tensor_sobel_x = torch.FloatTensor(np_array_sobel_x)
conv2d_sobel_x_filter = torch.nn.Conv2d(in_channels=1, out_channels=1,␣
  ↪kernel_size=np_array_sobel_x.shape, bias=False)

expanded_tensor_sobel_x = tensor_sobel_x.expand(conv2d_sobel_x_filter.weight.
  ↪size())
conv2d_sobel_x_filter.weight = torch.nn.Parameter(expanded_tensor_sobel_x,␣
  ↪requires_grad = False)

# Sobel_y Conv2D filter
np_array_sobel_y = np.array(sobel_y)
tensor_sobel_y = torch.FloatTensor(np_array_sobel_y)
conv2d_sobel_y_filter = torch.nn.Conv2d(in_channels=1, out_channels=1,␣
  ↪kernel_size=np_array_sobel_y.shape, bias=False)

expanded_tensor_sobel_y = tensor_sobel_y.expand(conv2d_sobel_y_filter.weight.
  ↪size())
conv2d_sobel_y_filter.weight = torch.nn.Parameter(expanded_tensor_sobel_y,␣
  ↪requires_grad = False)


# Perform filtering
image_sobel_x_filtered_tensor =␣
  ↪conv2d_sobel_x_filter(image_gaussian_filtered_tensor)
image_sobel_x_filtered = image_sobel_x_filtered_tensor.squeeze(0).squeeze(0).
  ↪detach().numpy()

image_sobel_y_filtered_tensor =␣
  ↪conv2d_sobel_y_filter(image_gaussian_filtered_tensor)
image_sobel_y_filtered = image_sobel_y_filtered_tensor.squeeze(0).squeeze(0).
  ↪detach().numpy()


# Calculate the gradient magnitude map
grad_mag3 = (image_sobel_x_filtered**2 + image_sobel_y_filtered**2)**0.5

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


0
200
400
600
800
1000
1200
0
200
400
600
800
1000
1200

```
[ ]:
```