coursework_01

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]`

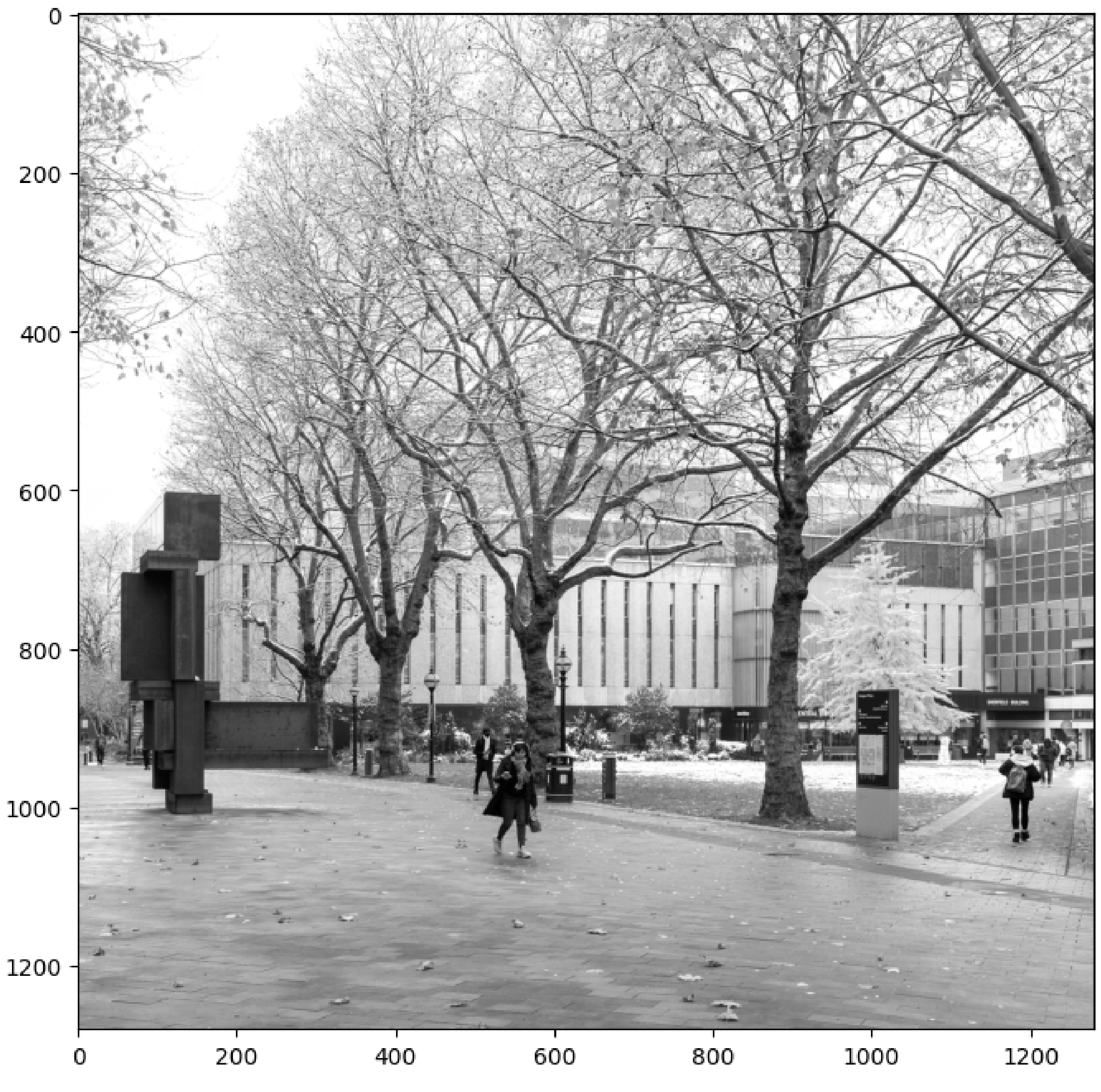
[2]:
```
# Import libaries (provided)
import imageio.v3 as imageio
import numpy as np
import matplotlib.pyplot as plt
import noise
import scipy
import scipy.signal
import math
import time
```

1.3 1. Moving average filter (20 points).

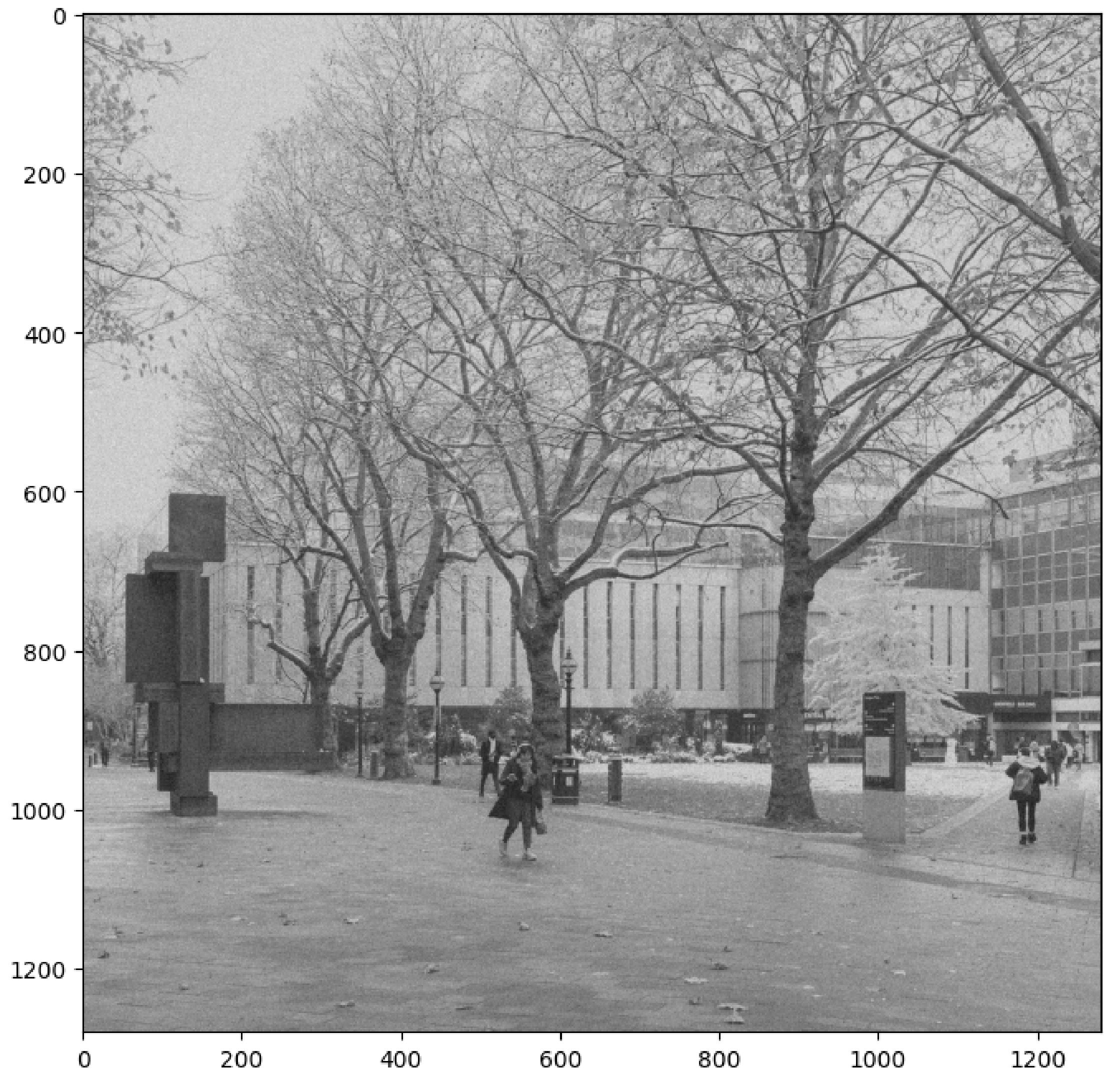
Read the provided input image, add noise to the image and design a moving average filter for denoising.

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

[3]:
```python
# Read the image (provided)
image = imageio.imread('campus_snow.jpg')
plt.imshow(image, cmap='gray')
plt.gcf().set_size_inches(8, 8)
```



[4]:
```
# 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)
```



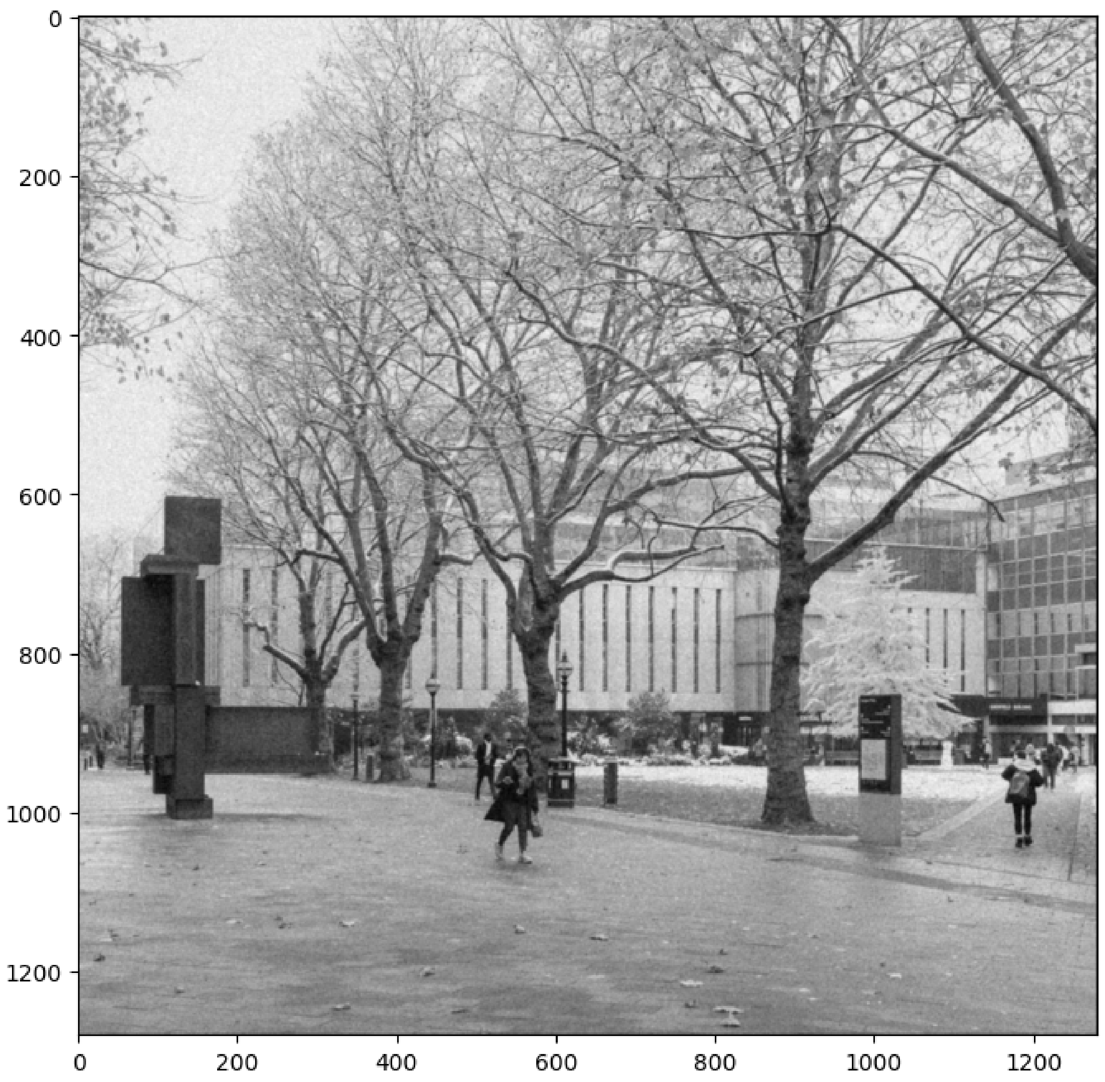
1.3.1 Note: from now on, please use the noisy image as the input for the filters.

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

[7]:
```
# Design the filter h
### Insert your code ###
window_size = 3
h = np.ones((window_size, window_size)) / (window_size ** 2)
```

```python
# Convolve the corrupted image with h using scipy.signal.convolve2d function
### Insert your code ###
image_filtered = scipy.signal.convolve2d(image_noisy, h, mode='same',␣
 ↪boundary='fill', fillvalue=0)

# 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]]
```

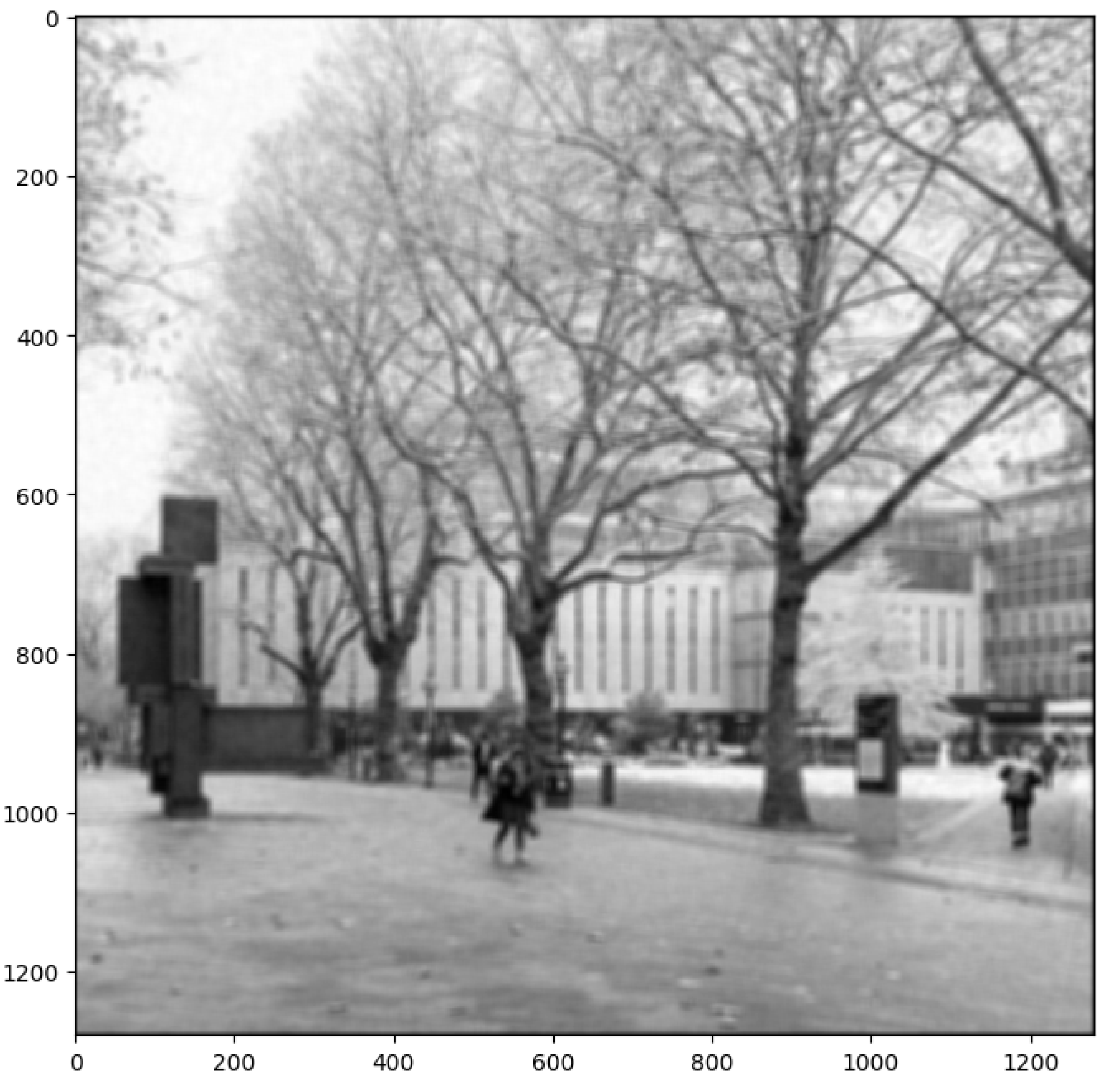


1.3.3 1.2 Filter the noisy image with a 11x11 moving average filter.

[8]:
```
# Design the filter h
### Insert your code ###
window_size = 11
h = np.ones((window_size, window_size)) / (window_size ** 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',␣
↪boundary='fill', fillvalue=0)

# Print the filter (provided)
```

```python
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

All kernels reduce the amount of noise in the image and give a smoothing/blurring effect.

Smaller kernels result in a less blurry image output than larger ones. Smaller kernels are good at preserving edges and details in the image since they average over a much smaller area. They might not be effective in images with dense Gaussian noise however.

Larger kernels result in a more blurry image but will be better at reducing the effect of noise. This is the right choice for very noisy images. Unfortunately, we get less defined edges and details due to the stronger smoothing effect. We must also take into account that larger kernels will have slower processing time for large images since each pixel has to have an entire n by n matrix computed.

We can choose our kernel size based off the trade-off of noise reduction vs detail needed.

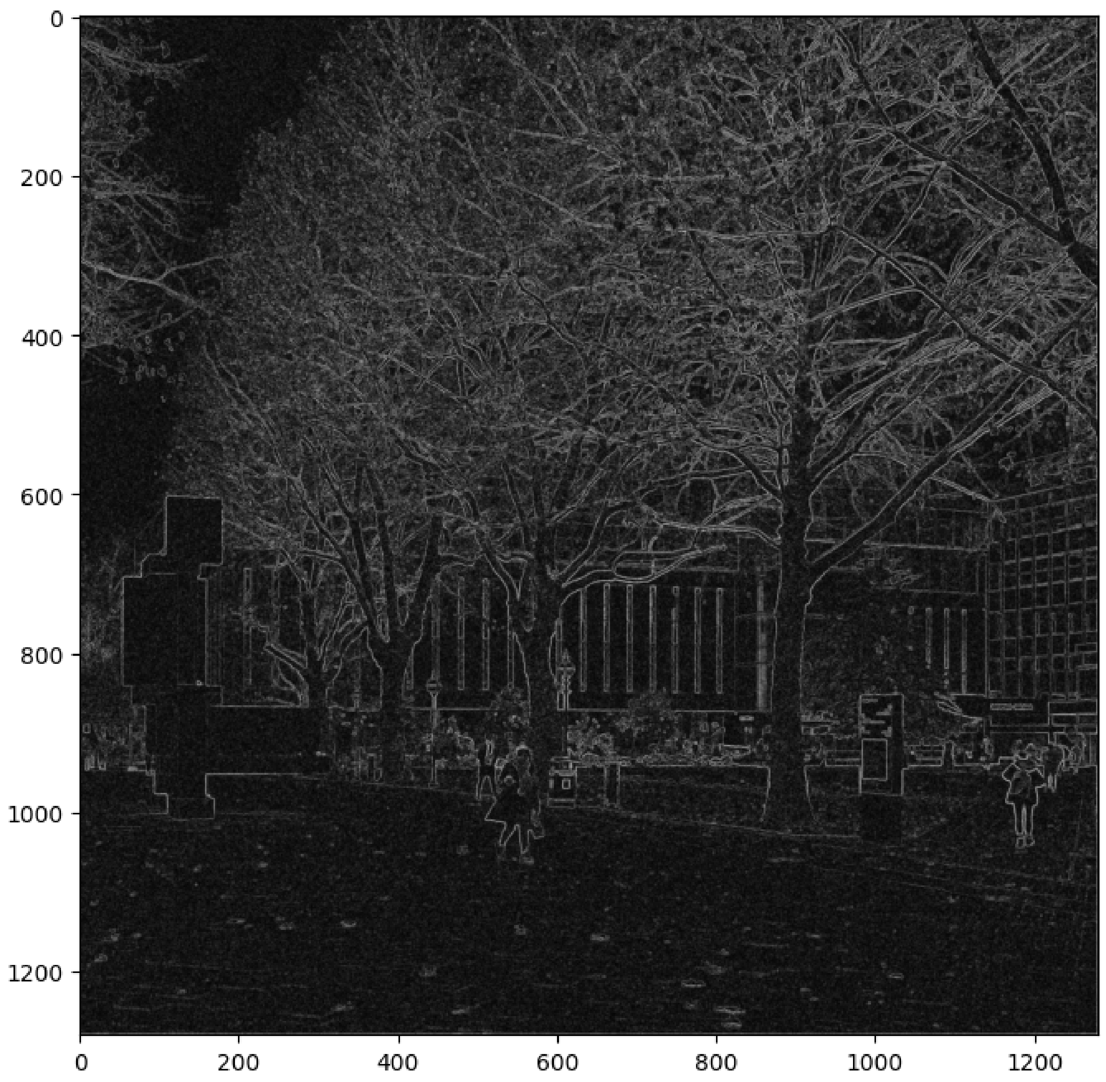
1.4 2. Edge detection (56 points).

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.

[21]:
```
# 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 ###
filtered_x = scipy.signal.convolve2d(image_noisy, sobel_x, mode='same',␣
 ↪boundary='fill', fillvalue=0)
filtered_y = scipy.signal.convolve2d(image_noisy, sobel_y, mode='same',␣
 ↪boundary='fill', fillvalue=0)

# Calculate the gradient magnitude
### Insert your code ###
grad_mag = np.sqrt(filtered_x**2 + filtered_y**2)

# 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)
```

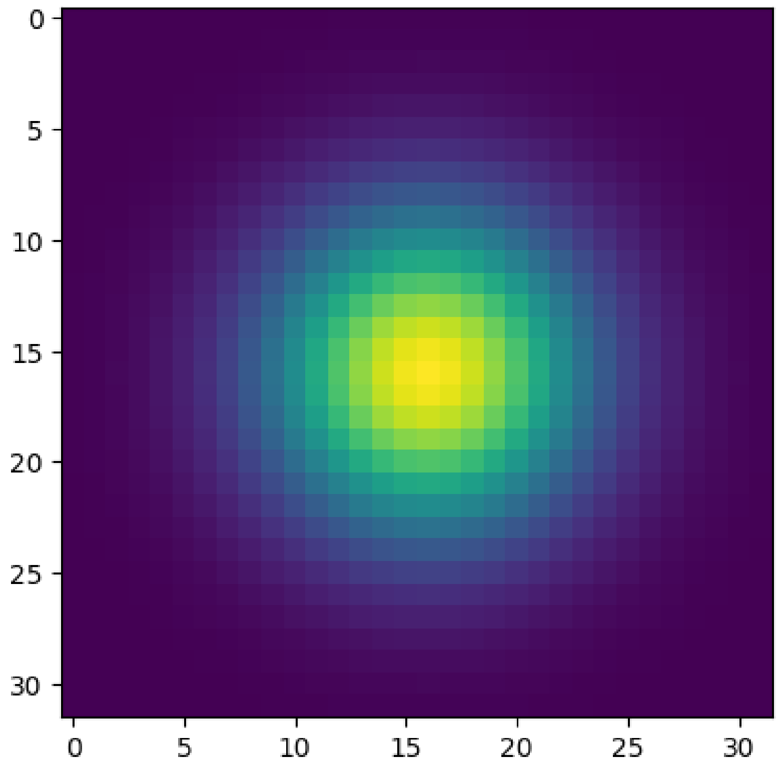
```
sobel_x:
[[ 1  0 -1]
 [ 2  0 -2]
 [ 1  0 -1]]
sobel_y:
[[ 1  2  1]
 [ 0  0  0]
 [-1 -2 -1]]
```



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

[22]:
```
# 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_size = int(2*np.ceil(3*sigma)+1)

    axis_range = np.arange(-filter_size//2, filter_size//2 + 1)
    x, y = np.meshgrid(axis_range, axis_range)

    h = (1 / (2 * np.pi * sigma**2)) * np.exp(-(x**2 + y**2) / (2 * sigma**2))

    h /= h.sum()

    return h

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

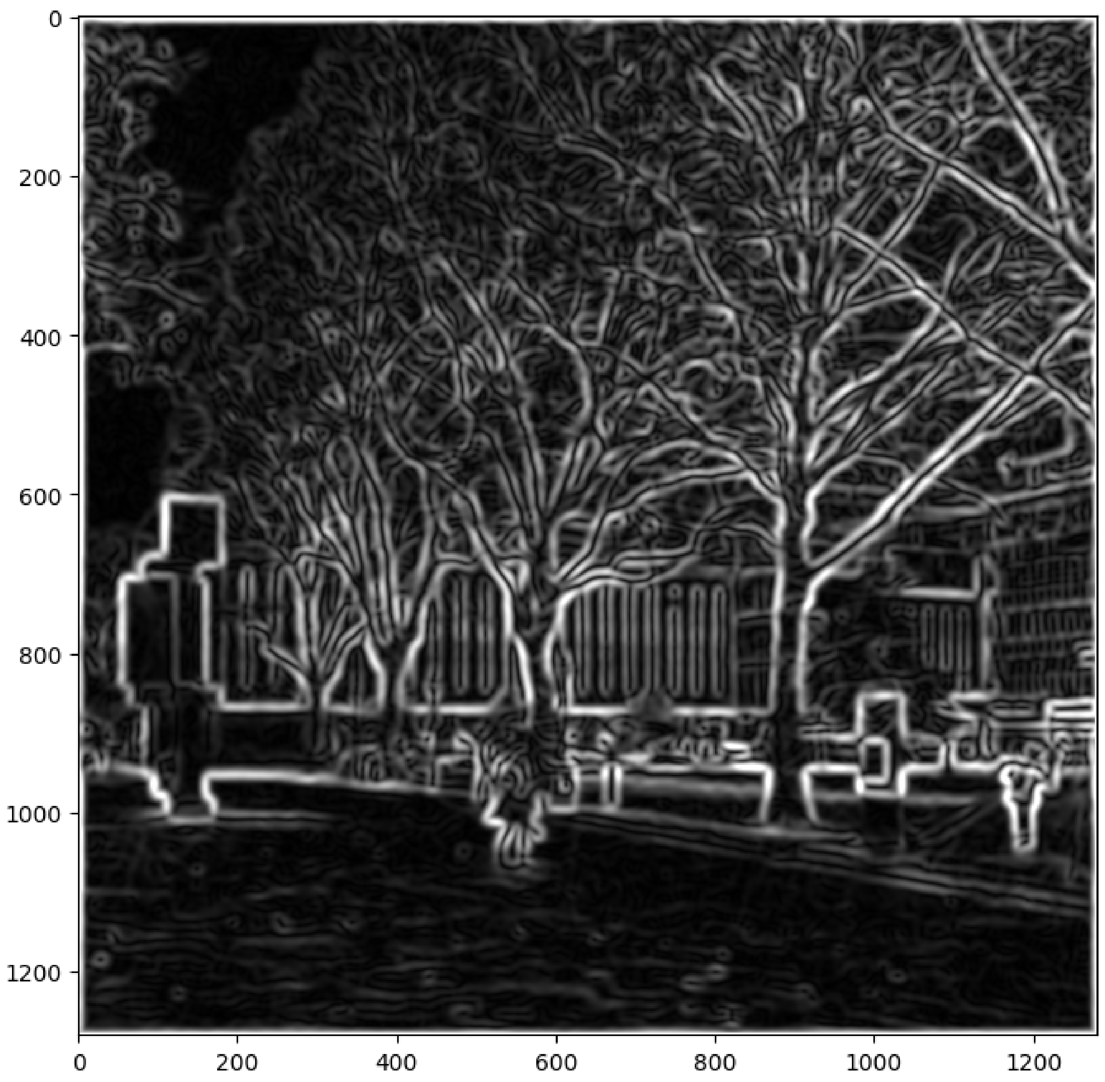
[22]:
```
<matplotlib.image.AxesImage at 0x17573b510>
```



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.

[23]:
```python
# Construct the Gaussian filter
### Insert your code ###
gaussian_filter = gaussian_filter_2d(5)

# Perform Gaussian smoothing and count time
### Insert your code ###
start_time1 = time.time()
image_smoothed = scipy.signal.convolve2d(image_noisy, gaussian_filter,␣
  ↪mode='same', boundary='fill', fillvalue=0)
end_time1 = time.time() - start_time1
print("time taken: " + str(end_time1))

# Image filtering
### Insert your code ###
filtered_x = scipy.signal.convolve2d(image_smoothed, sobel_x, mode='same',␣
  ↪boundary='fill', fillvalue=0)
filtered_y = scipy.signal.convolve2d(image_smoothed, sobel_y, mode='same',␣
  ↪boundary='fill', fillvalue=0)

# Calculate the gradient magnitude
### Insert your code ###
grad_mag = np.sqrt(filtered_x**2 + filtered_y**2)

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

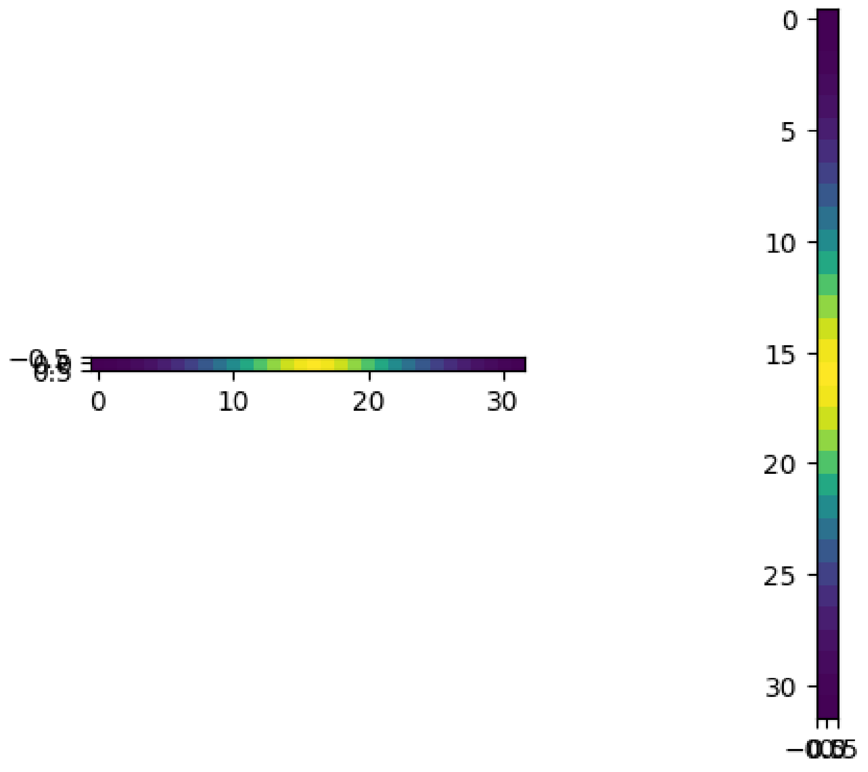
```
time taken: 2.2607219219207764
```



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.

[24]:
```python
# 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 ###
    kernel_size = int(2 * np.ceil(3 * sigma) + 1)

    x = np.arange(-kernel_size//2, kernel_size//2 + 1)
    h = (1 / (np.sqrt(2 * np.pi) * sigma)) * np.exp(-x**2 / (2 * sigma**2))

    h /= np.sum(h)
    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, -1)

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

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

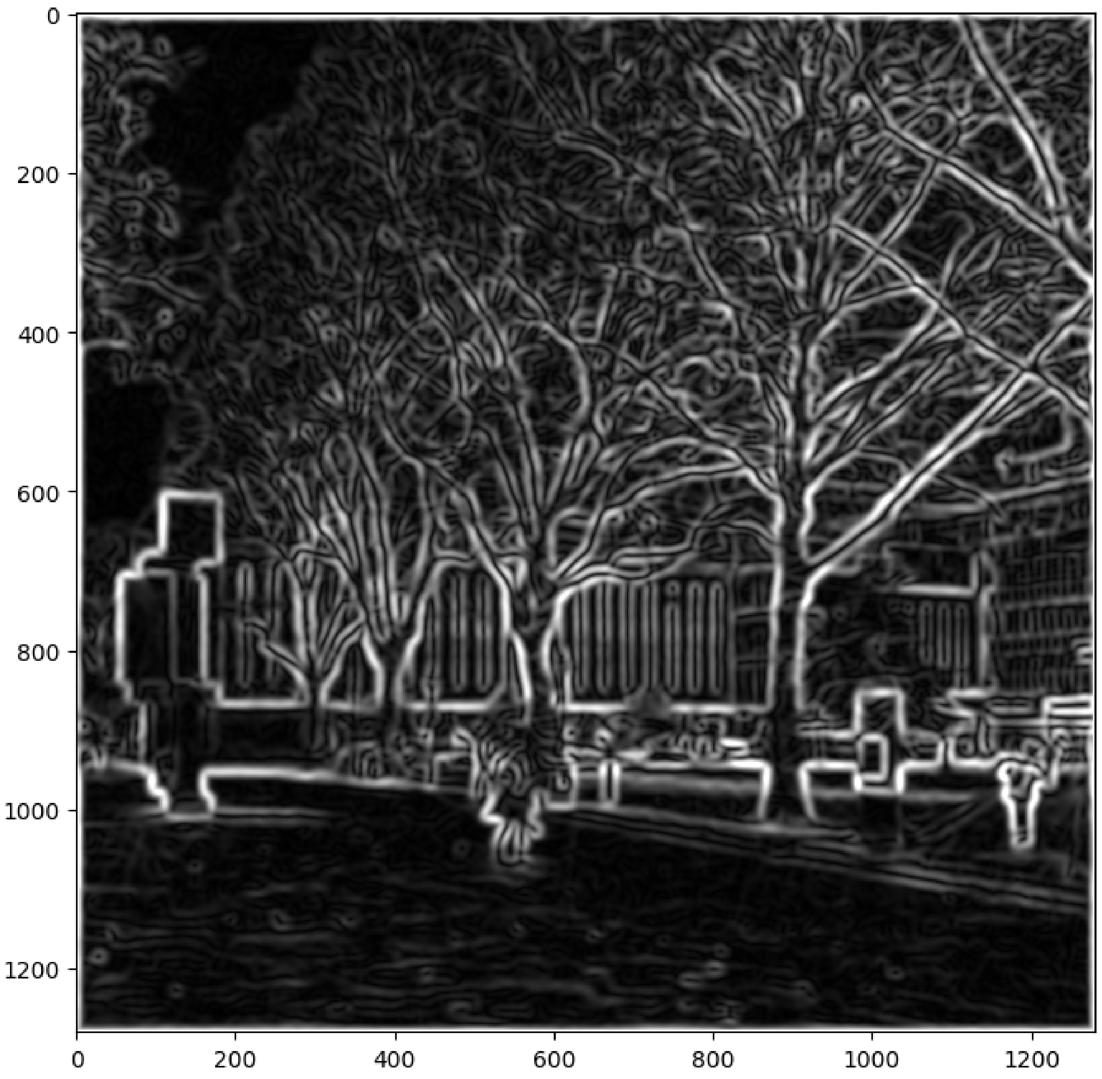
[24]:
```
<matplotlib.image.AxesImage at 0x17573ca90>
```



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.

[28]:
```
# Perform separable Gaussian smoothing and count time
### Insert your code ###
h = gaussian_filter_1d(5).reshape(1, -1)
start_time2=time.time()
gaussian_horizontal = scipy.signal.convolve2d(image_noisy, h, mode='same')
image_smoothed = scipy.signal.convolve2d(gaussian_horizontal, h.T, mode='same')
end_time2 = time.time() - start_time2
print("time taken 1: " + str(end_time1))
print("time taken 2: " + str(end_time2))

# Image filtering
### Insert your code ###
filtered_x = scipy.signal.convolve2d(image_smoothed, sobel_x, mode='same',␣
 ↪boundary='fill', fillvalue=0)
filtered_y = scipy.signal.convolve2d(image_smoothed, sobel_y, mode='same',␣
 ↪boundary='fill', fillvalue=0)

# Calculate the gradient magnitude
### Insert your code ###
grad_mag2 = np.sqrt(filtered_x**2 + filtered_y**2)

# 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 ###
difference = np.abs(grad_mag2 - grad_mag)  # Adjust shapes if necessary
mean_difference = np.mean(difference)
print("mean difference: " + str(mean_difference))
```

```
time taken 1: 2.2607219219207764
time taken 2: 0.26613521575927734
mean difference: 4.3189945070360187e-13
```



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

Insert your answer

The visual results of non-separable and separable filtering results is near identical. This is because both methods are numerically equivalent. We can clearly see the edges being highlighted and with high contrast, which shows that the effect of the smoothing and filtering works well. Separable filtering is clearly a valid substitute for non-separable filtering.

There is a significant difference in the time taken computationally between separable (2.2607219219207764 seconds) and non-separable (0.26613521575927734) filtering. The separable filtering is about 8x faster than without, since there is a huge benefit to carrying out two 1D convolutions instead of a single 2D convolution. This makes separable filtering a preffered method for real-time applications or when processing a large number of images.

The mean difference between the gradient magnitude maps is 4.3189945070360187e-13. This is almost 0, and is likely due to the limitations of floating-point arithmetic precision, and has no practical impact.

In conclusion we should always attempt to use separable Gaussian filtering since it is significantly less intensive (computationally) than non-separable and has no real tradeoff besides being slightly lengthier to implement.

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.

[29]:
```
# Import libaries (provided)
import torch
```

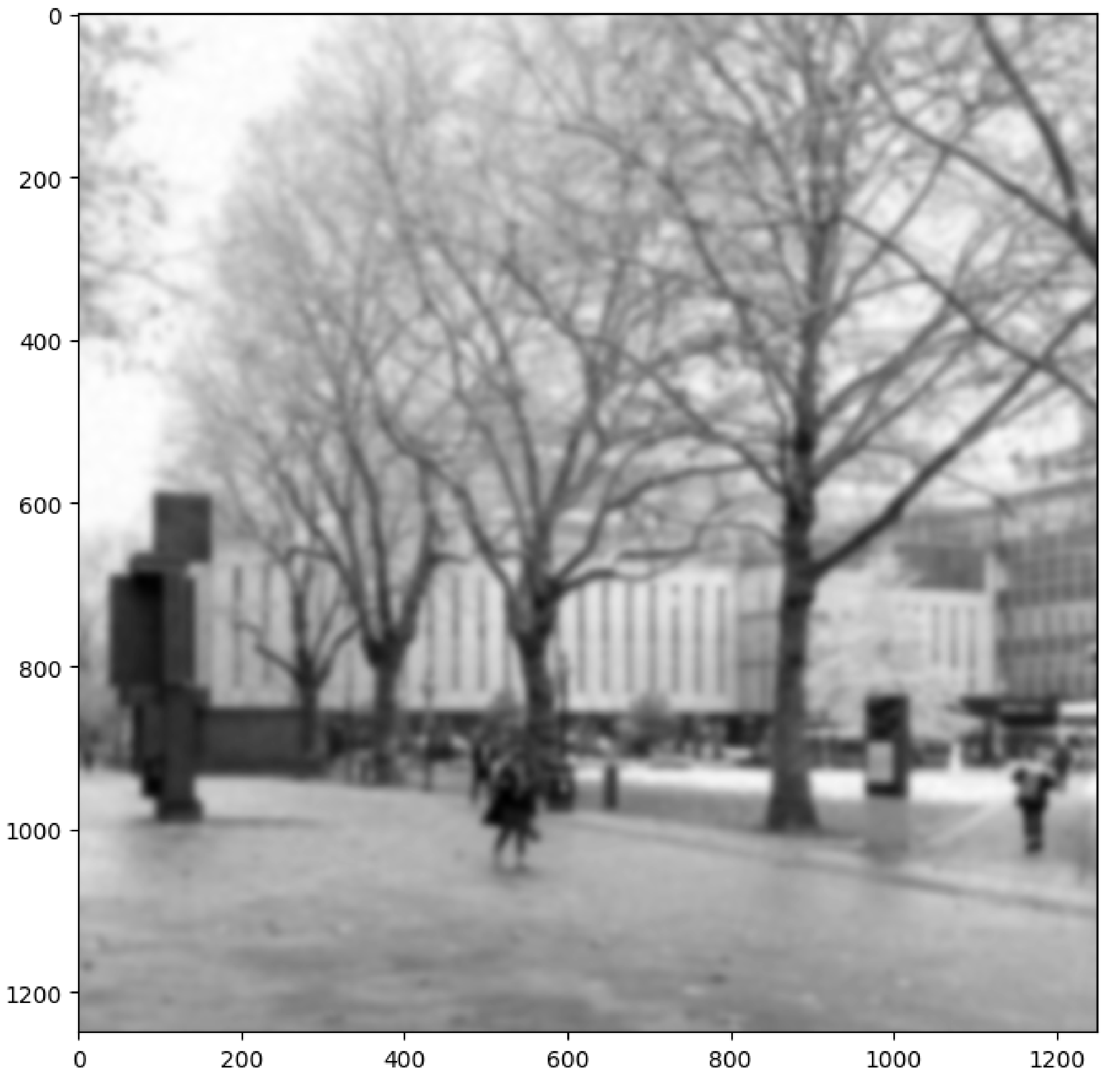
1.5.1 3.1 Expand the dimension of the noisy image into 1x1xXxY and convert it to a Pytorch tensor.

[30]:
```
# Expand the dimension of the numpy array
### Insert your code ###
noisy_image_torch = image_noisy[np.newaxis, np.newaxis, :, :]

# Convert to a Pytorch tensor using torch.from_numpy
### Insert your code ###
noisy_image_tensor = torch.from_numpy(noisy_image_torch).float()
```

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

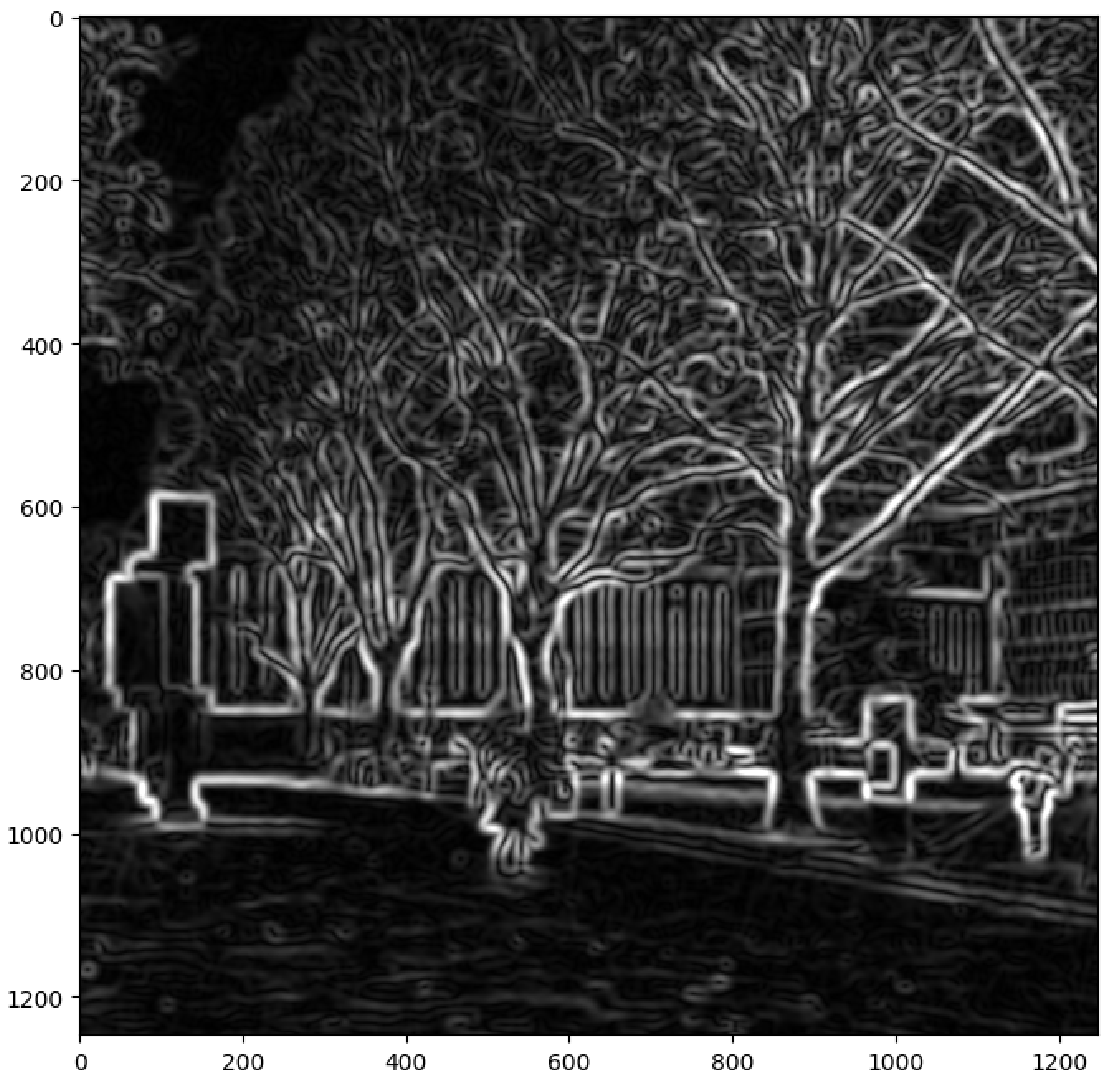
[35]:
```
# A 2D Gaussian filter when sigma = 5 pixel (provided)
sigma = 5
h = gaussian_filter_2d(sigma)
h_tensor = torch.from_numpy(h).float().unsqueeze(0).unsqueeze(0)

# Create the Conv2D filter
### Insert your code ###
conv = torch.nn.Conv2d(in_channels=1, out_channels=1, kernel_size=h_tensor.
 ↪shape[-1], bias=False)
conv.weight.data = h_tensor

# Filtering
### Insert your code ###
with torch.no_grad():
    image_filtered_tensor = conv(noisy_image_tensor.float())
```

```
image_filtered = image_filtered_tensor.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.

[38]:
```
# Create Conv2D filters
### Insert your code ###
sobel_x_kernel = np.array([[-1, 0, 1],
                           [-2, 0, 2],
                           [-1, 0, 1]], dtype=np.float32)

sobel_y_kernel = np.array([[-1, -2, -1],
                           [ 0,  0,  0],
                           [ 1,  2,  1]], dtype=np.float32)

sobel_x_tensor = torch.from_numpy(sobel_x_kernel).unsqueeze(0).unsqueeze(0)
sobel_y_tensor = torch.from_numpy(sobel_y_kernel).unsqueeze(0).unsqueeze(0)

conv_sobel_x = torch.nn.Conv2d(in_channels=1, out_channels=1, kernel_size=3,␣
 ↪bias=False)
conv_sobel_y = torch.nn.Conv2d(in_channels=1, out_channels=1, kernel_size=3,␣
 ↪bias=False)

conv_sobel_x.weight.data = sobel_x_tensor
conv_sobel_y.weight.data = sobel_y_tensor

# Perform filtering
### Insert your code ###
with torch.no_grad():
    grad_x = conv_sobel_x(image_filtered_tensor)
    grad_y = conv_sobel_y(image_filtered_tensor)

    grad_mag3_torch = torch.sqrt(grad_x**2 + grad_y**2)

# Calculate the gradient magnitude map
### Insert your code ###

grad_mag3 = grad_mag3_torch.squeeze().cpu().numpy()

# 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

```
[ ]:
```