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.

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.

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]`

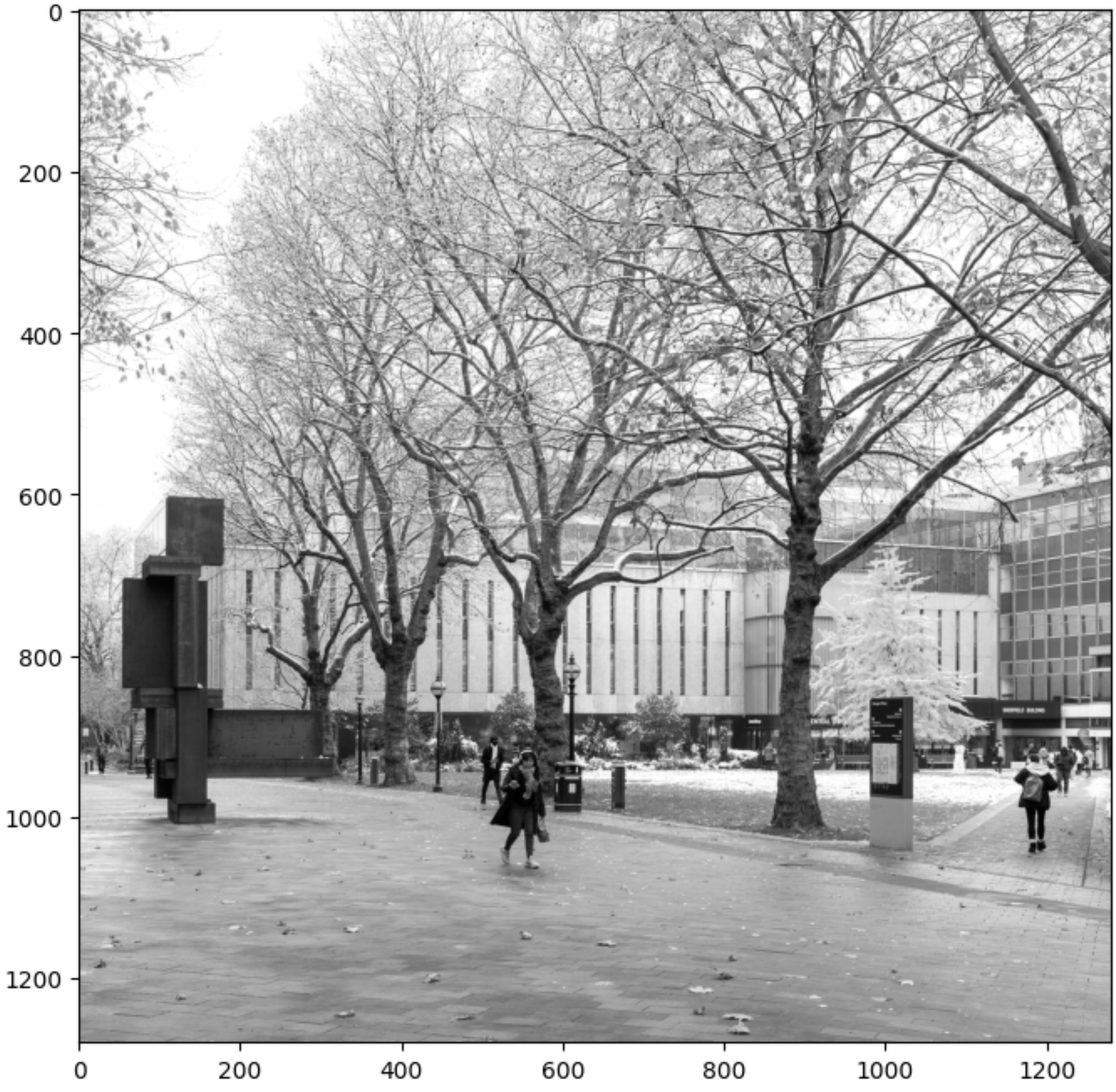
In [1]:
```
# Import libaries (provided)
import imageio.v3 as imageio
import numpy as np
import matplotlib.pyplot as plt
import noise
import scipy
import scipy.signal
import math
import time
```

1. Moving average filter (20 points).

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

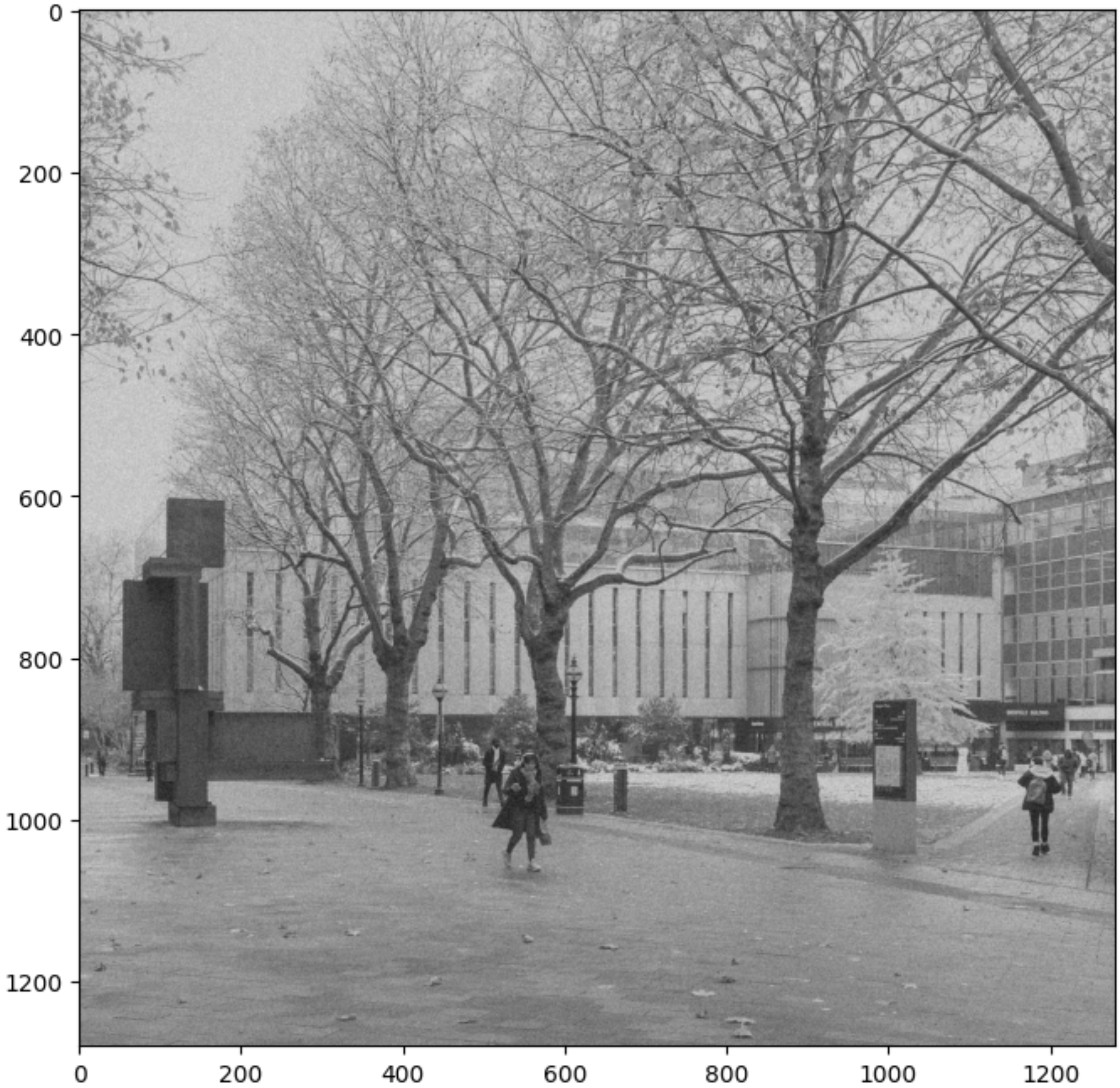
You are expected to design the kernel of the filter and then perform 2D image filtering using the function `scipy.signal.convolve2d()`.

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



In [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)
```



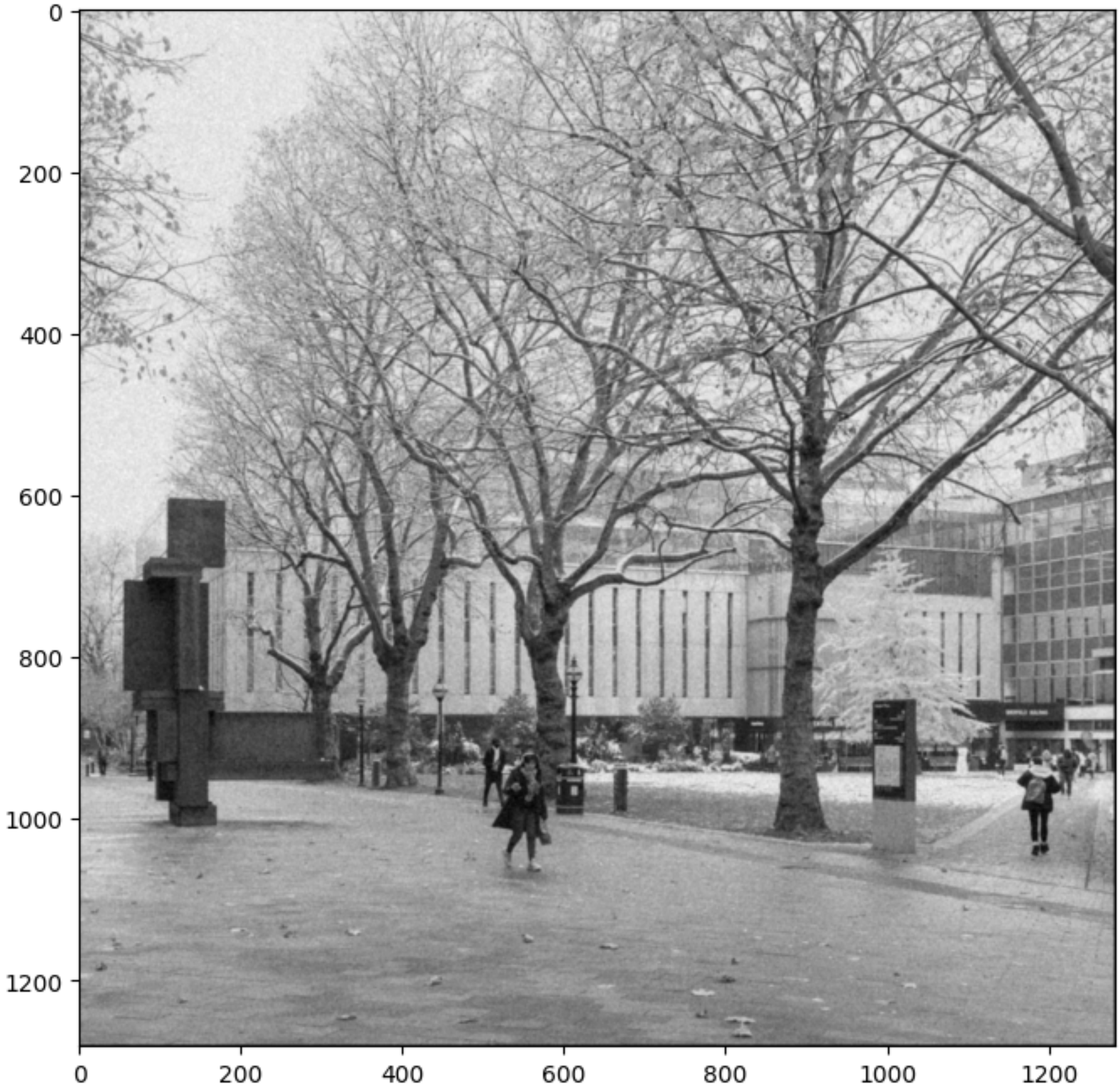
Note: from now on, please use the noisy image as the input for the filters.

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

In [4]:
```
# Design the filter h
### Insert your code ###
h_np = np.full((3,3), 1/(3*3))
h = h_np.tolist()

# Convolve the corrupted image with h using scipy.signal.convolve2d funct
### 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:
[[0.1111111111111111, 0.1111111111111111, 0.1111111111111111], [0.1111111111111111, 0.1111111111111111, 0.1111111111111111], [0.1111111111111111, 0.1111111111111111, 0.1111111111111111]]
```

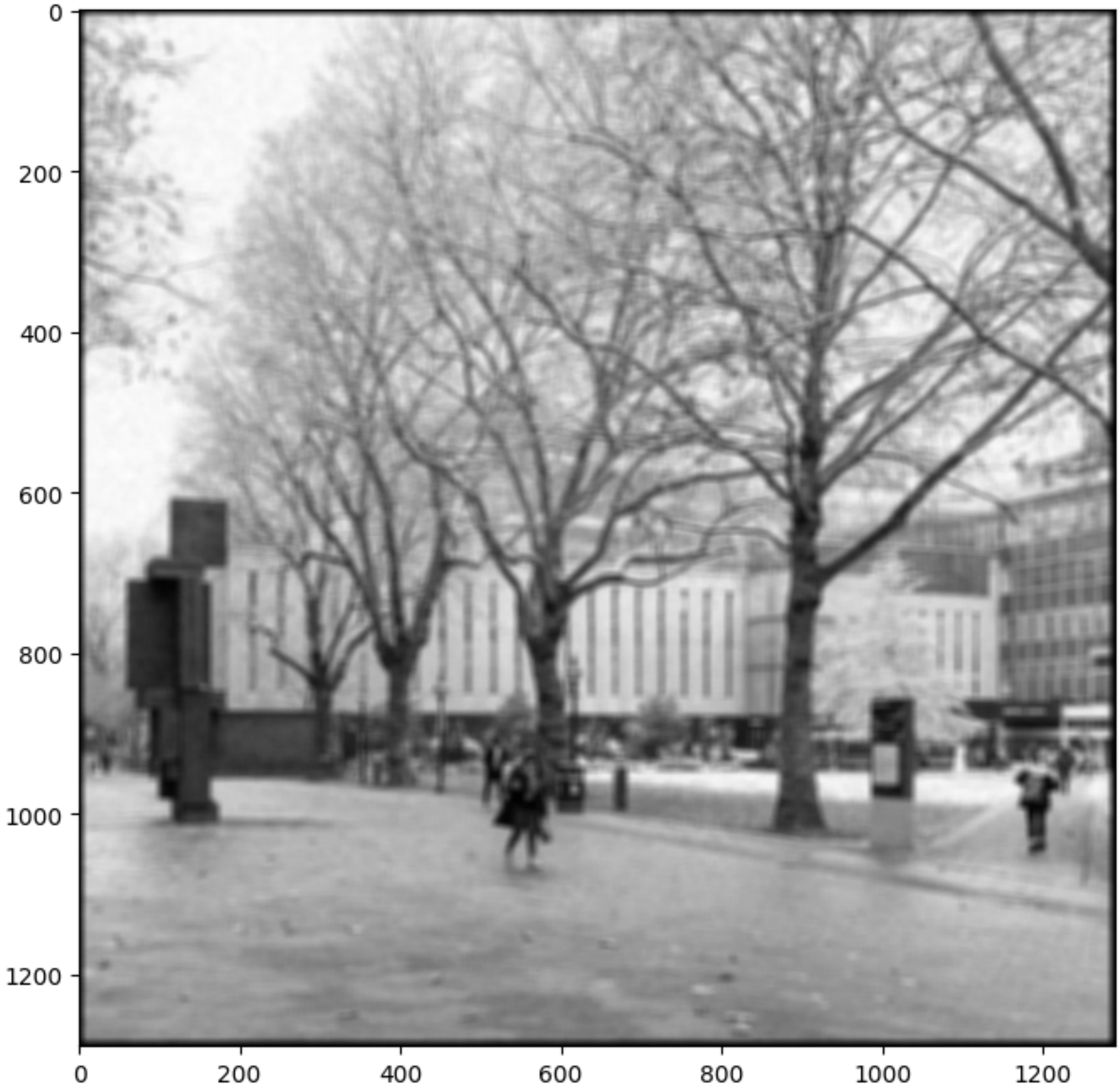


1.2 Filter the noisy image with a 11x11 moving average filter.

In [5]:
```
# Design the filter h
### Insert your code ###
h_np = np.full((11,11), 1/(11*11))
h = h_np.tolist()
```

```
# Convolve the corrupted image with h using scipy.signal.convolve2d funct
### 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:
[[0.008264462809917356, 0.008264462809917356, 0.008264462809917356, 0.0082
64462809917356, 0.008264462809917356, 0.008264462809917356, 0.008264462809
917356, 0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356], [0.008264462809917356, 0.008264462809917356, 0.0082
64462809917356, 0.008264462809917356, 0.008264462809917356, 0.008264462809
917356, 0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356], [0.008264462809917356, 0.0082
64462809917356, 0.008264462809917356, 0.008264462809917356, 0.008264462809
917356, 0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356], [0.0082
64462809917356, 0.008264462809917356, 0.008264462809917356, 0.008264462809
917356, 0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356, 0.008264
462809917356], [0.008264462809917356, 0.008264462809917356, 0.008264462809
917356, 0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356, 0.008264
462809917356, 0.008264462809917356], [0.008264462809917356, 0.008264462809
917356, 0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356, 0.008264
462809917356, 0.008264462809917356, 0.008264462809917356], [0.008264462809
917356, 0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356, 0.008264
462809917356, 0.008264462809917356, 0.008264462809917356, 0.00826446280991
7356], [0.008264462809917356, 0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356, 0.008264
462809917356, 0.008264462809917356, 0.008264462809917356, 0.00826446280991
7356, 0.008264462809917356], [0.008264462809917356, 0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356, 0.008264
462809917356, 0.008264462809917356, 0.008264462809917356, 0.00826446280991
7356, 0.008264462809917356, 0.008264462809917356], [0.008264462809917356,
0.008264462809917356, 0.008264462809917356, 0.008264462809917356, 0.008264
462809917356, 0.008264462809917356, 0.008264462809917356, 0.00826446280991
7356, 0.008264462809917356, 0.008264462809917356, 0.008264462809917356], [
0.008264462809917356, 0.008264462809917356, 0.008264462809917356, 0.008264
462809917356, 0.008264462809917356, 0.008264462809917356, 0.00826446280991
7356, 0.008264462809917356, 0.008264462809917356, 0.008264462809917356, 0.
008264462809917356]]
```



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

The larger the kernel size, the more blurred the filtered image appears.

2. Edge detection (56 points).

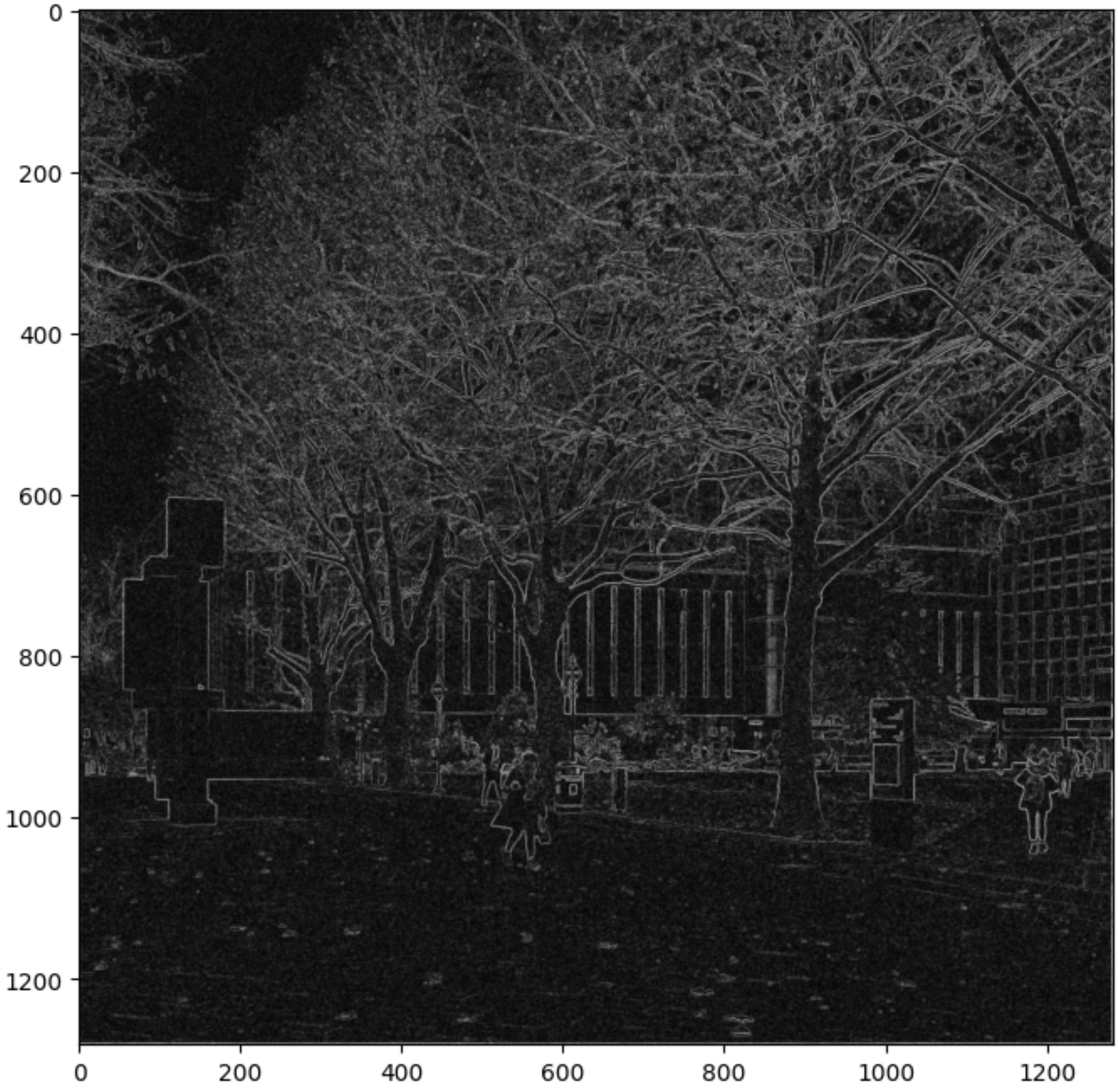
Perform edge detection using Sobel filtering, as well as Gaussian + Sobel filtering.

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

In [6]:
```
# 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 ###
g_x = scipy.signal.convolve2d(image_noisy, sobel_x)
g_y = scipy.signal.convolve2d(image_noisy, sobel_y)

# Calculate the gradient magnitude
### Insert your code ###
g_x_np = np.array(g_x)
g_y_np = np.array(g_y)
grad_mag_np = np.sqrt(np.square(g_x_np) + np.square(g_y_np))
grad_mag = grad_mag_np.tolist()

# 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]]
```

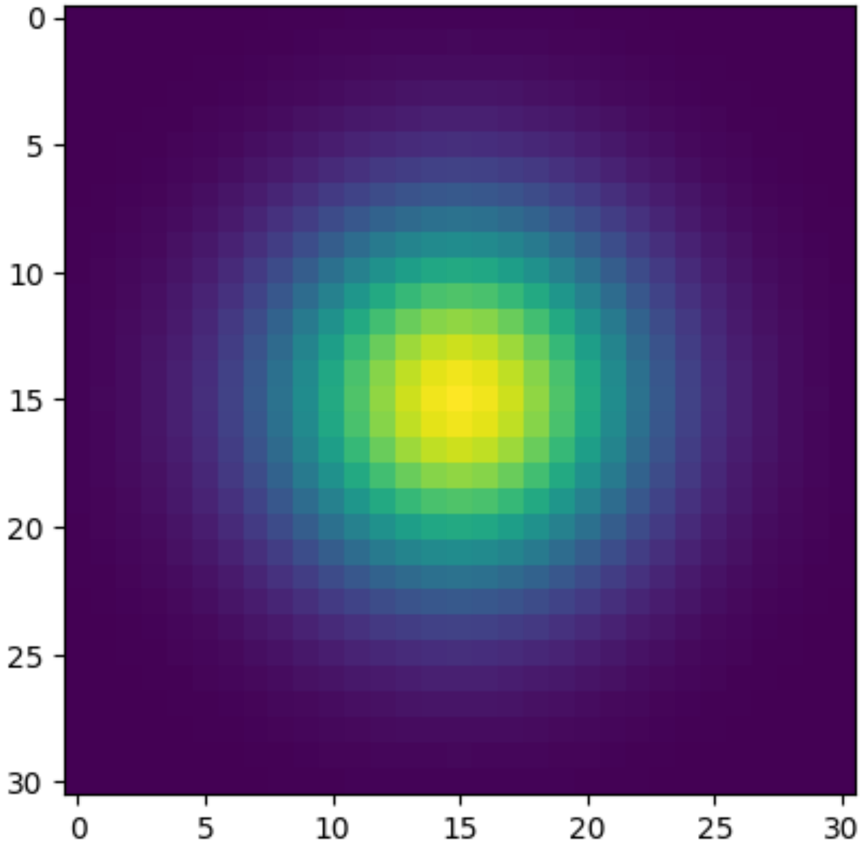


2.2 Implement a function that generates a 2D Gaussian filter given the parameter σ.

In [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 ###
    # Filter radius is k times sigma where k = 3
    k = 3
    rad = k * sigma
    sz = 2 * rad + 1
    # [..., -2, -1, 0, 1, 2, ...]
    i_row = np.arange(-(rad), (rad) + 1, 1)
    i = np.tile(i_row, (sz, 1))
```

```
    j = np.transpose(i)

    # exponent : -(i**2 + j**2) / 2 * sigma**2
    exponent = - (np.square(i) + np.square(j)) / (2 * sigma**2)
    h = np.exp(exponent) * (1 / (2 * np.pi * sigma**2))

    # Normalise (values sum to 1)
    h = h / np.sum(h)

    return h.tolist()

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

Out[7]: <matplotlib.image.AxesImage at 0x177fa50d0>

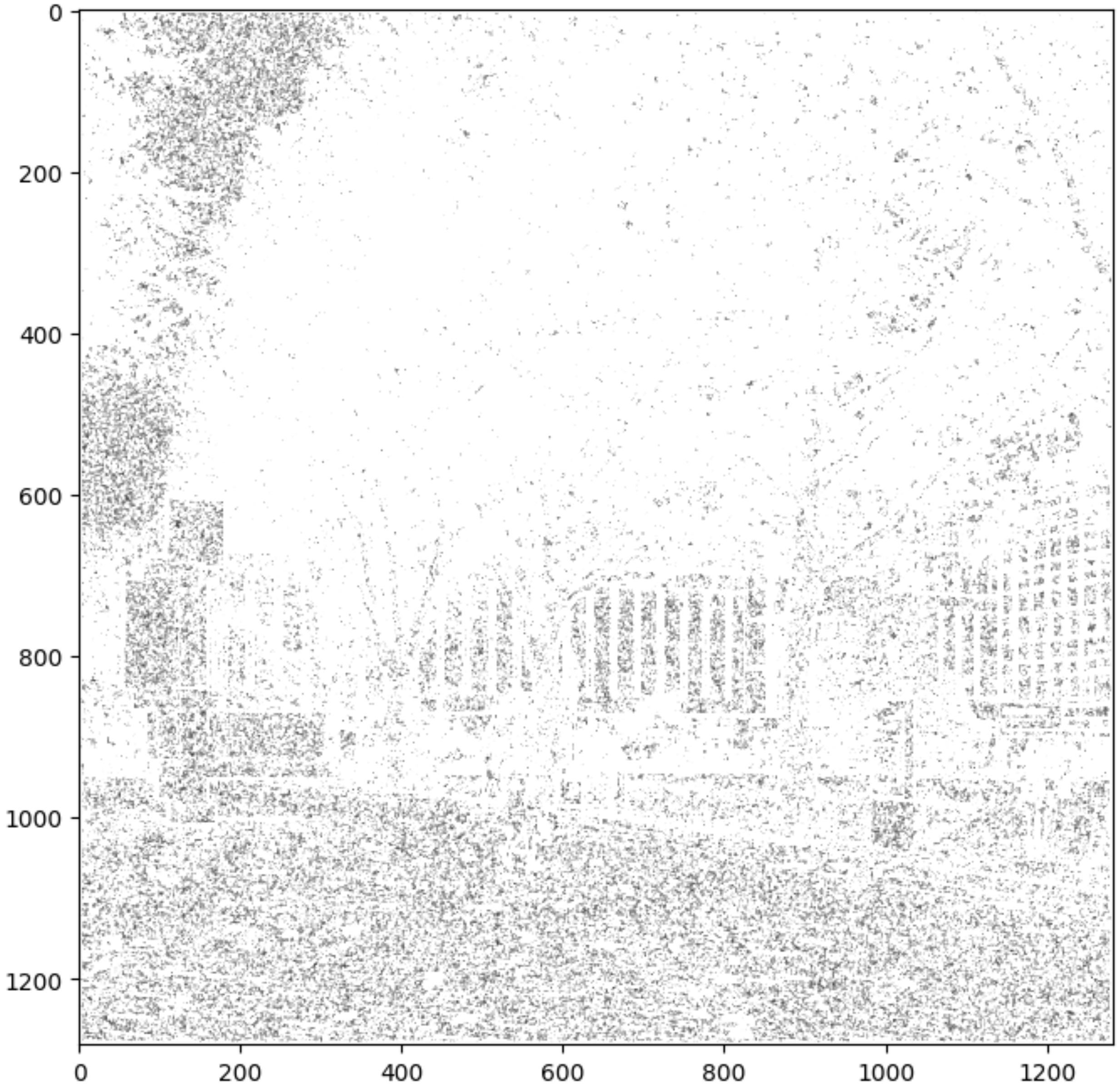


2.3 Perform Gaussian smoothing (σ = 5 pixels) and evaluate the computational time for Gaussian smoothing. After that, perform Sobel filtering and show the gradient magintude map.

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

```python
# Perform Gaussian smoothing and count time
### Insert your code ###
start = time.time()
image_smoothed = scipy.signal.convolve2d(image_noisy, g)
end = time.time()
elapsed = end - start
print(f"Elapsed time: {elapsed} seconds")

# Image filtering
### Insert your code ###
g_x = np.array(scipy.signal.convolve2d(image_smoothed, sobel_x))
g_y = np.array(scipy.signal.convolve2d(image_smoothed, sobel_y))

# Calculate the gradient magnitude
### Insert your code ###
grad_mag = np.sqrt(np.square(g_x_np) + np.square(g_y_np)).tolist()

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

```
Elapsed time: 1.8598520755767822 seconds
```

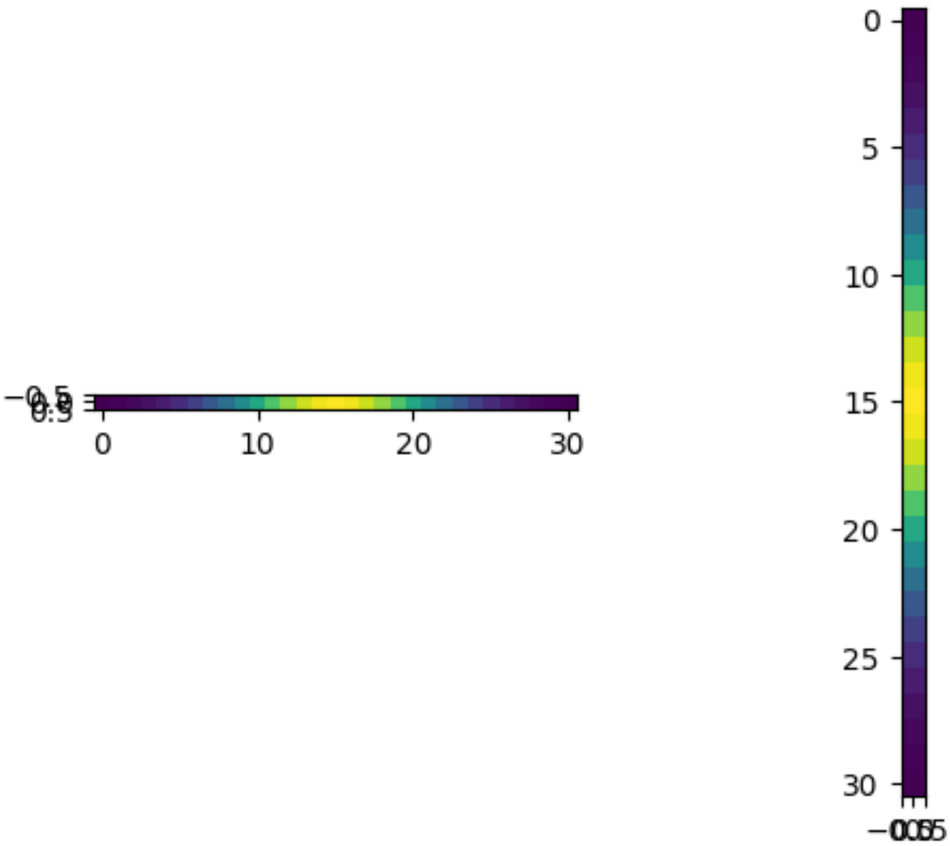


2.4 Implement a function that generates a 1D Gaussian filter given the parameter σ. Generate 1D Gaussian filters along x-axis and y-axis respectively.

In [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 ###
    # Filter radius is k times sigma where k = 3
    k = 3
    rad = k * sigma
    sz = 2 * rad + 1
    # [..., -2, -1, 0, 1, 2, ...]
    i = np.arange(-(rad), (rad) + 1, 1)
```

```
    # exponent : -(i**2) / 2 * sigma**2
    exponent = - (np.square(i)) / (2 * sigma**2)
    h = np.exp(exponent) * (1 / (math.sqrt(2 * np.pi) * sigma))

    # Normalise (values sum to 1)
    h = h / np.sum(h)

    return [h.tolist()]

# 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)

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

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

Out[9]: <matplotlib.image.AxesImage at 0x177ec5670>

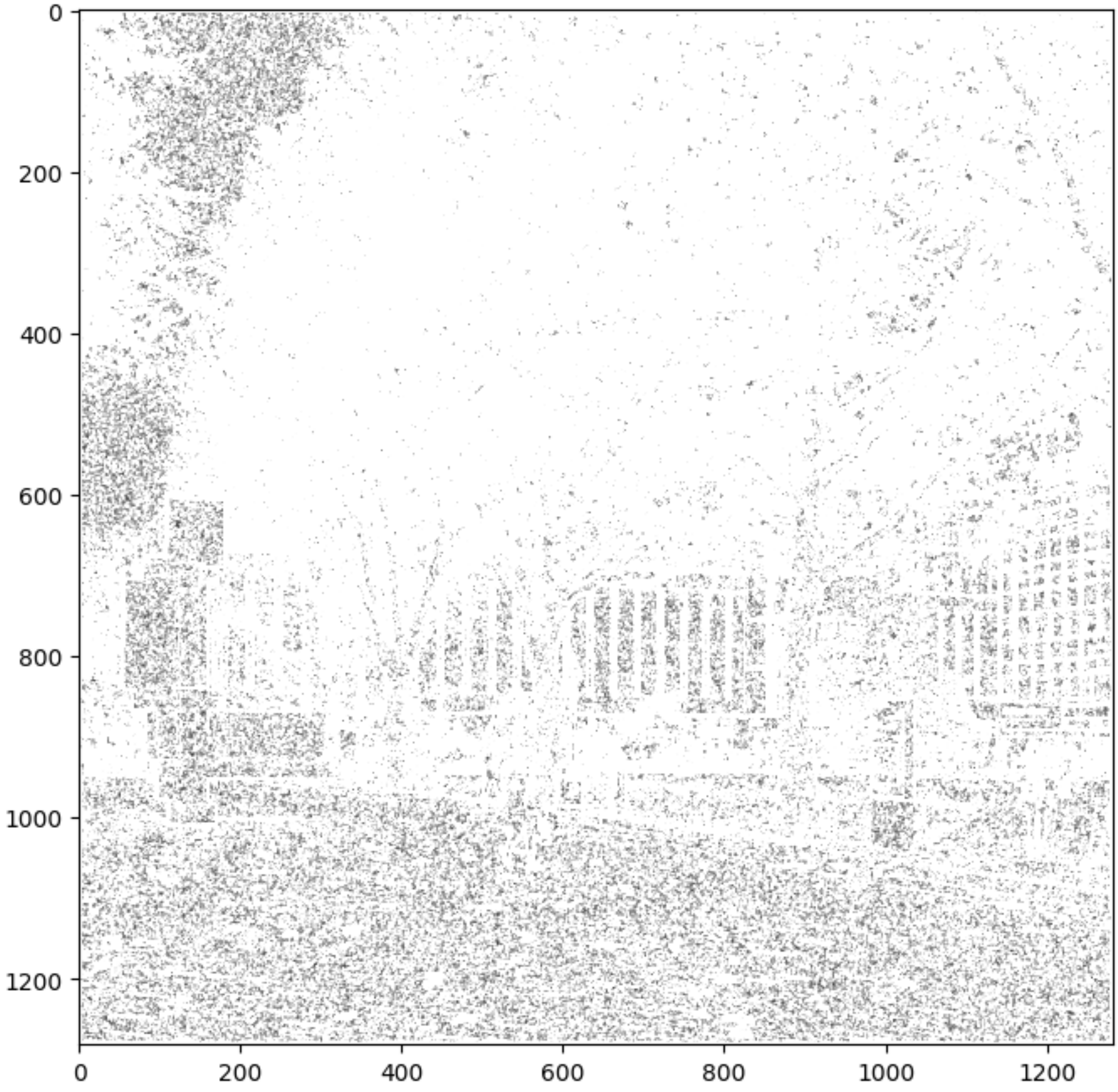


2.6 Perform Gaussian smoothing (σ = 5 pixels) using two separable filters and evaluate the computational time for separable Gaussian filtering. After that, perform Sobel filtering, show the gradient magnitude map and check whether it is the same as the previous one without separable filtering.

In [10]:

```
# Perform separable Gaussian smoothing and count time
### Insert your code ###
start = time.time()
# used h_x and h_y from 2.4
image_applied_x = scipy.signal.convolve2d(image_noisy, h_x)
image_smoothed = scipy.signal.convolve2d(image_applied_x, h_y)
end = time.time()
elapsed = end - start
print(f"Elapsed time: {elapsed} seconds")

# Image filtering
### Insert your code ###
g_x = np.array(scipy.signal.convolve2d(image_smoothed, sobel_x))
g_y = np.array(scipy.signal.convolve2d(image_smoothed, sobel_y))

# Calculate the gradient magnitude
### Insert your code ###
grad_mag2 = np.sqrt(np.square(g_x_np) + np.square(g_y_np)).tolist()

# 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_diff = np.mean(np.array(grad_mag2) - np.array(grad_mag))
print(f"Mean difference: {mean_diff}")
```

```
Elapsed time: 0.26508402824401855 seconds
Mean difference: 0.0
```



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

As shown by the mean difference, the results of both the 2D and separable Gaussian filters are the same. The key difference is that the separable filter takes significantly less computational time.

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.

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

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

In [12]:
```
# Expand the dimension of the numpy array
### Insert your code ###
image_np = np.expand_dims(np.array(image_noisy), axis=(0,1))
# image_noisy shape -> (1280, 1280), image_np shape -> (1, 1, 1280, 1280)

# Convert to a Pytorch tensor using torch.from_numpy
### Insert your code ###
image_tensor = torch.from_numpy(image_np)
```

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

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

# Create the Conv2D filter
### Insert your code ###
# kernel size is 2*rad + 1 where rad = k*sigma
# alternatively, can be taken from width of h
k = 3
sz = 2 * (k * sigma) + 1
h_np = np.array(h)
h_tensor = torch.from_numpy(h_np)
m = torch.nn.Conv2d(1, 1, sz, bias=False)
m.weight.data = h_tensor
m.weight.requires_grad = False

# Filtering
### Insert your code ###
image_filtered = m(image_tensor)

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

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
Cell In[13], line 19
     15 m.weight.requires_grad = False
     17 # Filtering
     18 ### Insert your code ###
---> 19 image_filtered = m(image_tensor)
     21 # Display the filtering result (provided)
     22 plt.imshow(image_filtered, cmap='gray')

File ~/Library/Python/3.9/lib/python/site-packages/torch/nn/modules/module.py:1511, in Module._wrapped_call_impl(self, *args, **kwargs)
   1509     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
   1510 else:
-> 1511     return self._call_impl(*args, **kwargs)

File ~/Library/Python/3.9/lib/python/site-packages/torch/nn/modules/module.py:1520, in Module._call_impl(self, *args, **kwargs)
   1515 # If we don't have any hooks, we want to skip the rest of the logic in
   1516 # this function, and just call forward.
   1517 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hooks or self._forward_pre_hooks
   1518         or _global_backward_pre_hooks or _global_backward_hooks
   1519         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1520     return forward_call(*args, **kwargs)
   1522 try:
   1523     result = None

File ~/Library/Python/3.9/lib/python/site-packages/torch/nn/modules/conv.py:460, in Conv2d.forward(self, input)
    459 def forward(self, input: Tensor) -> Tensor:
--> 460     return self._conv_forward(input, self.weight, self.bias)

File ~/Library/Python/3.9/lib/python/site-packages/torch/nn/modules/conv.py:456, in Conv2d._conv_forward(self, input, weight, bias)
    452 if self.padding_mode != 'zeros':
    453     return F.conv2d(F.pad(input, self._reversed_padding_repeated_twice, mode=self.padding_mode),
    454                     weight, bias, self.stride,
    455                     _pair(0), self.dilation, self.groups)
--> 456 return F.conv2d(input, weight, bias, self.stride,
    457                 self.padding, self.dilation, self.groups)

RuntimeError: weight should have at least three dimensions
```

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

In [ ]:
```
# Create Conv2D filters
### Insert your code ###

# Perform filtering
### Insert your code ###

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

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