coursework_01

January 26, 2022

1 Coursework 1: Image filtering

In this coursework you will practice image filtering techniques, which are commonly used to smooth, sharpen or add certain effects to images. The coursework includes both coding questions and written questions. Please read both the text and code comment 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 | Export Notebook As...) or print (using the print function of your browser) the notebook as a pdf file, which contains your code, results and text answers, and upload the pdf file onto Cate.
- If Jupyter-lab does not work for you, you can also use Google Colab to write the code and export the pdf file.

1.2 Dependencies:

If you do not have Jupyter-Lab on your laptop, you can find information for installing Jupyter-Lab here.

There may be certain Python packages you may want to use for completing the coursework. We have provided examples below for importing libraries. If some packages are missing, you need to install them. In general, new packages (e.g. imageio etc) can be installed by running

```
pip3 install [package_name]
```

in the terminal. If you use Anaconda, you can also install new packages by running `conda install [package_name]` or using its graphic user interface.

[ ]:
```
# Import libaries (provided)
import 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.

Read a specific input image and add noise to the image. Design a moving average filter of kernel size 3x3 and 11x11 respectively. Perform image filtering on the noisy image.

Design the kernel of the filter by yourself. Then perform 2D image filtering using the function `scipy.signal.convolve2d()`.

[ ]:
```
# Read the image (provided)
image = imageio.imread('hyde_park.jpg')
plt.imshow(image, cmap='gray')
plt.gcf().set_size_inches(10, 8)
```

[ ]:
```
# 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(10, 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. (5 points)

[ ]:
```
# Design the filter h
h = [[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
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(10, 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. (5 points)

[ ]:
```
# Design the filter h
h = np.ones((11, 11))/121

# Convolve the corrupted image with h using scipy.signal.convolve2d function
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(10, 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? (10 points)

The 3×3 kernal does remove some of the noise present in the image while maintaining a relatively acceptable level of image detail.

While the 11×11 kernal may remove more noise in the image, it also removes more image detail that results in a more blurry image compared to the 3×3 kernal.

Since larger kernels have more values factored into the average, and this woudl result in that a larger kernel will blur the image more than a smaller kernel which is seen in the comparison between kernal sizes of 3×3 and 11×11.

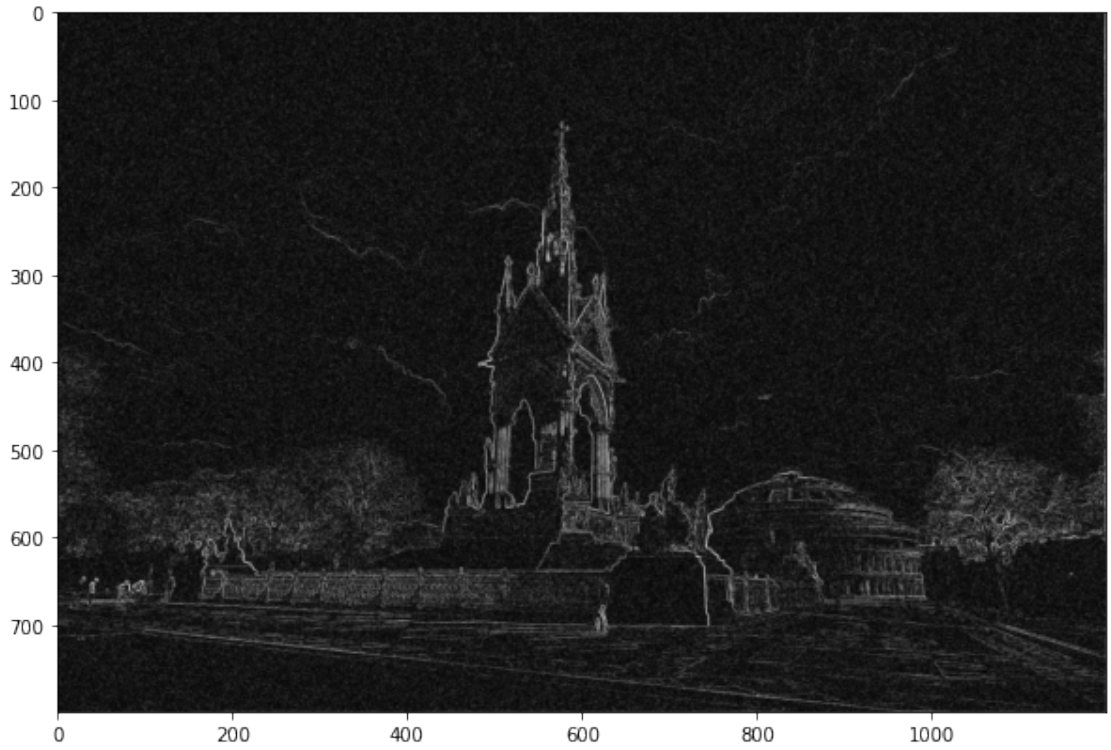
1.4 2. Edge detection.

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

1.4.1 2.1 Implement 3x3 Prewitt filters and convolve with the noisy image. (10 points)

[ ]:
```
# Design the Prewitt filters
prewitt_x = np.array([[1, 0, -1],
                      [1, 0, -1],
                      [1, 0, -1]])
prewitt_y = np.array([[1, 1, 1],
                      [0, 0, 0],
                      [-1, -1, -1]])

# Prewitt filtering
# 'mode': A string indicating the size of the output
# 'mode' attribute set to 'same' to ensure uniformity across results
image_sx = scipy.signal.convolve2d(image_noisy, prewitt_x, mode='same')
image_sy = scipy.signal.convolve2d(image_noisy, prewitt_y, mode='same')

# Calculate the gradient magnitude
grad_mag = np.sqrt(image_sx**2 + image_sy**2)

# Print the filters (provided)
print('prewitt_x:')
print(prewitt_x)
print('prewitt_y:')
print(prewitt_y)

# Display the gradient magnitude image (provided)
plt.imshow(grad_mag, cmap='gray')
plt.gcf().set_size_inches(10, 8)
```

```
prewitt_x:
[[ 1  0 -1]
 [ 1  0 -1]
 [ 1  0 -1]]
prewitt_y:
[[ 1  1  1]
 [ 0  0  0]
 [-1 -1 -1]]
```

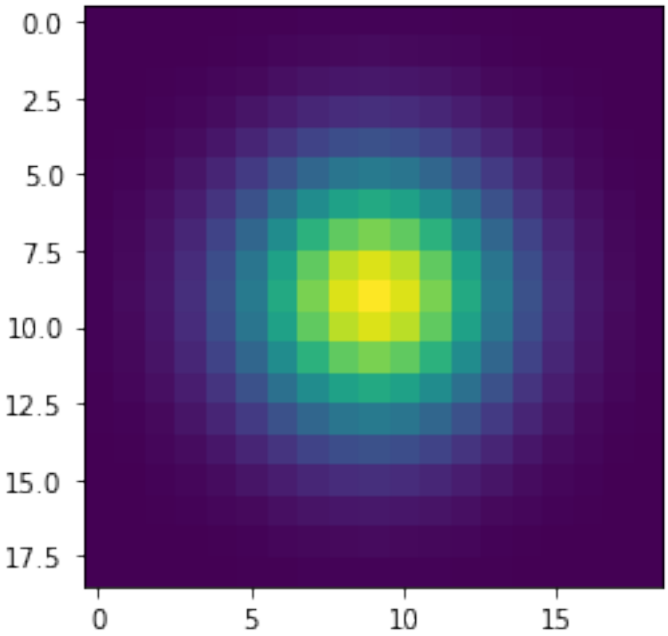


1.4.2 2.2 Implement a function that generates a 2D Gaussian filter given the parameter σ. (10 points)

[ ]:
```
# 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
    # k as 3 or 4

    k = 3

    radius = int((k * sigma))
    size = 2 * radius + 1
    h = []

    for i in range(size):
        h.append([])
        for j in range(size):
            num = math.exp((-(i - radius) ** 2 - (j - radius) ** 2) / (2 *␣
↪sigma ** 2))
            den = 2 * math.pi * sigma ** 2
```

```
            h[i].append(num / den)

    return h

# Visualise the Gaussian filter when sigma = 3 pixel (provided)
sigma = 3

h = gaussian_filter_2d(sigma)

plt.imshow(h)
```

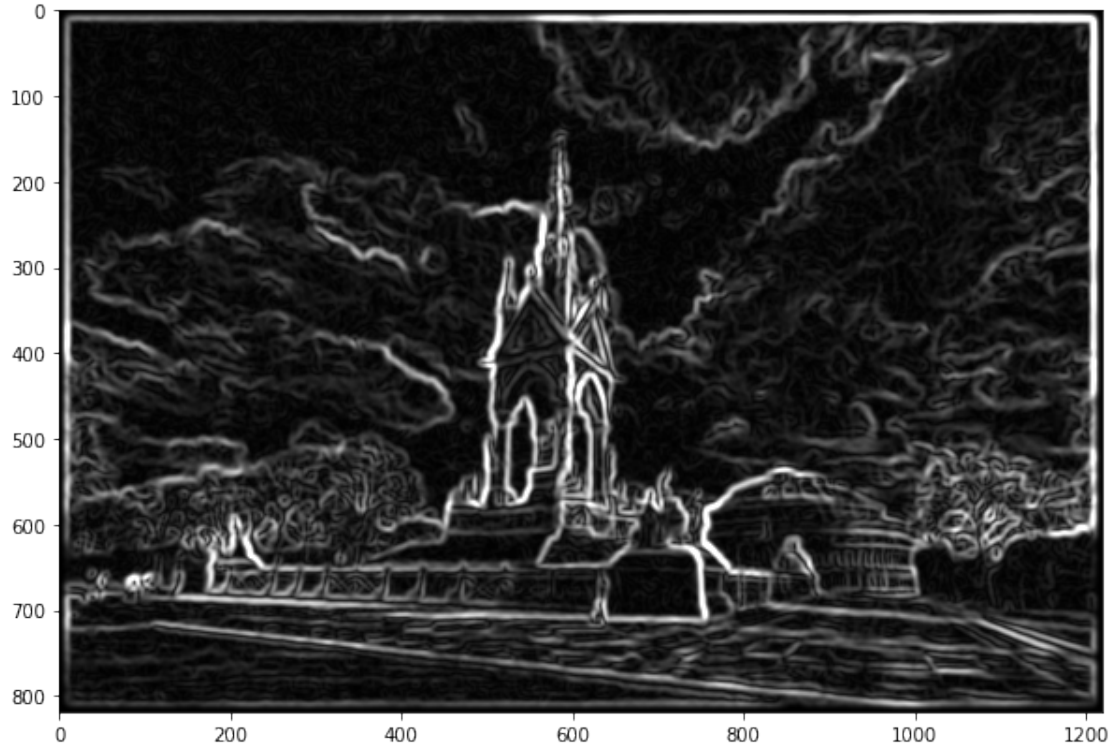
[ ]: <matplotlib.image.AxesImage at 0x2216adda5b0>



1.4.3 2.3 Perform Gaussian smoothing ($\sigma = 3$ pixels), followed by Prewitt filtering, show the gradient magnitude image. (5 points)

[ ]:
```
# Perform Gaussian smoothing before Prewitt filtering
gauss_kernal = gaussian_filter_2d(3)

image_gauss_smoothed = scipy.signal.convolve2d(image_noisy, gauss_kernal)
# image_gauss_smoothed = scipy.ndimage.gaussian_filter(image_noisy, 3)

# Prewitt filtering
image_sx = scipy.signal.convolve2d(image_gauss_smoothed, prewitt_x)
image_sy = scipy.signal.convolve2d(image_gauss_smoothed, prewitt_y)
```

```
# Calculate the gradient magnitude
grad_mag = np.sqrt(image_sx**2 + image_sy**2)

# Display the gradient magnitude image (provided)
plt.imshow(grad_mag, cmap='gray', vmin=0, vmax=100)
plt.gcf().set_size_inches(10, 8)
```

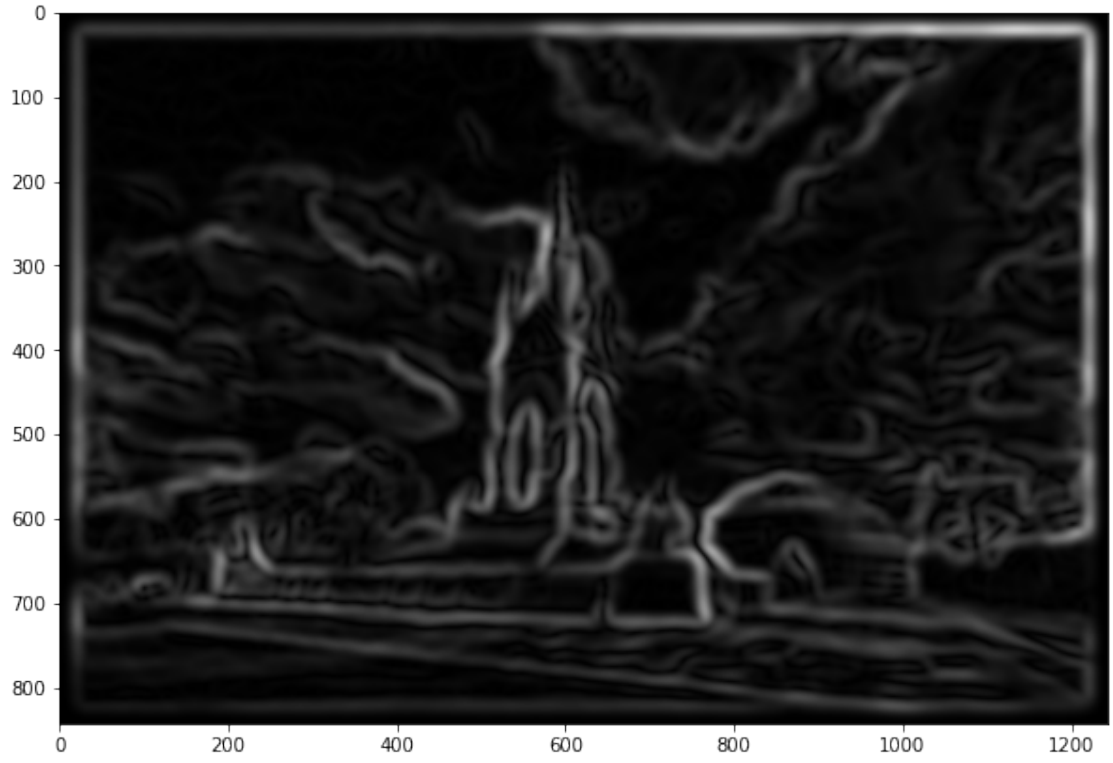


1.4.4 2.4 Perform Gaussian smoothing ($\sigma = 7$ pixels) and evaluate the computational time for Gaussian smoothing. After that, perform Prewitt filtering. (7 points)

[ ]:
```
# Construct the Gaussian filter
sigma = 7
gaussian_filter = gaussian_filter_2d(sigma)

# Perform Gaussian smoothing and count time
start_time = time.time()
image_gauss_smoothed = scipy.signal.convolve2d(image_noisy, gaussian_filter,␣
 ↪mode = 'same')
end_time = time.time() - start_time

print("Gaussian smoothing time taken: ", end_time)
```

```
# Prewitt filtering
image_sx = scipy.signal.convolve2d(image_gauss_smoothed, prewitt_x, mode='same')
image_sy = scipy.signal.convolve2d(image_gauss_smoothed, prewitt_y, mode='same')

# Calculate the gradient magnitude
grad_mag = np.sqrt(image_sx**2 + image_sy**2)

# Display the gradient magnitude image (provided)
plt.imshow(grad_mag, cmap='gray', vmin=0, vmax=100)
plt.gcf().set_size_inches(10, 8)
```

```
Gaussian smoothing time taken:  5.420028448104858
```

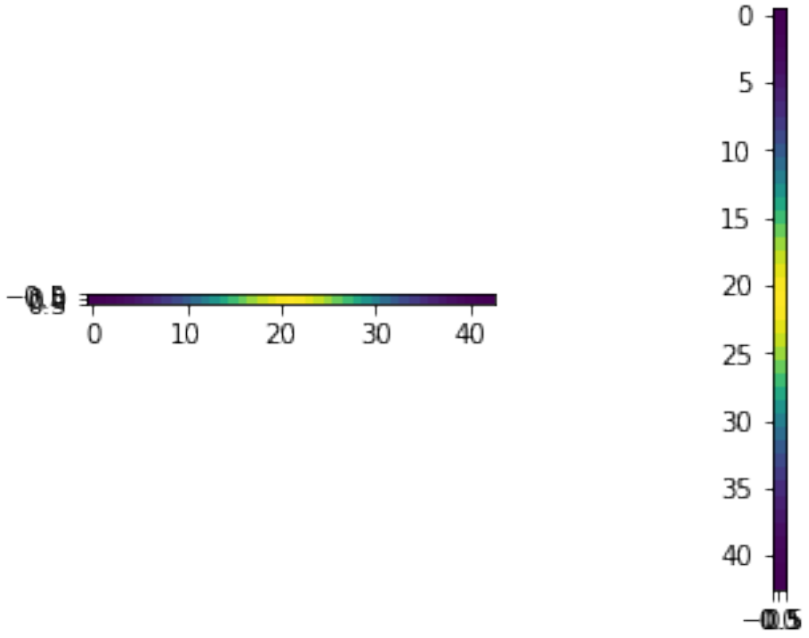


1.4.5 2.5 Implement a function that generates a 1D Gaussian filter given the parameter σ. Generate 1D Gaussian filters along x-axis and y-axis respectively. (10 points)

[ ]:
```
# 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
    k = 3
    radius = int(k * sigma)
    h = []

    for i in range(2 * radius + 1):
        exp = math.exp((-(i - radius) ** 2) / (2 * sigma ** 2))
        denominator = 2 * math.pi * sigma ** 2
        h.append(exp / denominator)

    return h

# sigma = 7 pixel (provided)
sigma = 7

# The Gaussian filter along x-axis. Its shape is (1, sz).
h_x = np.expand_dims(gaussian_filter_1d(sigma), axis = 0)

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

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

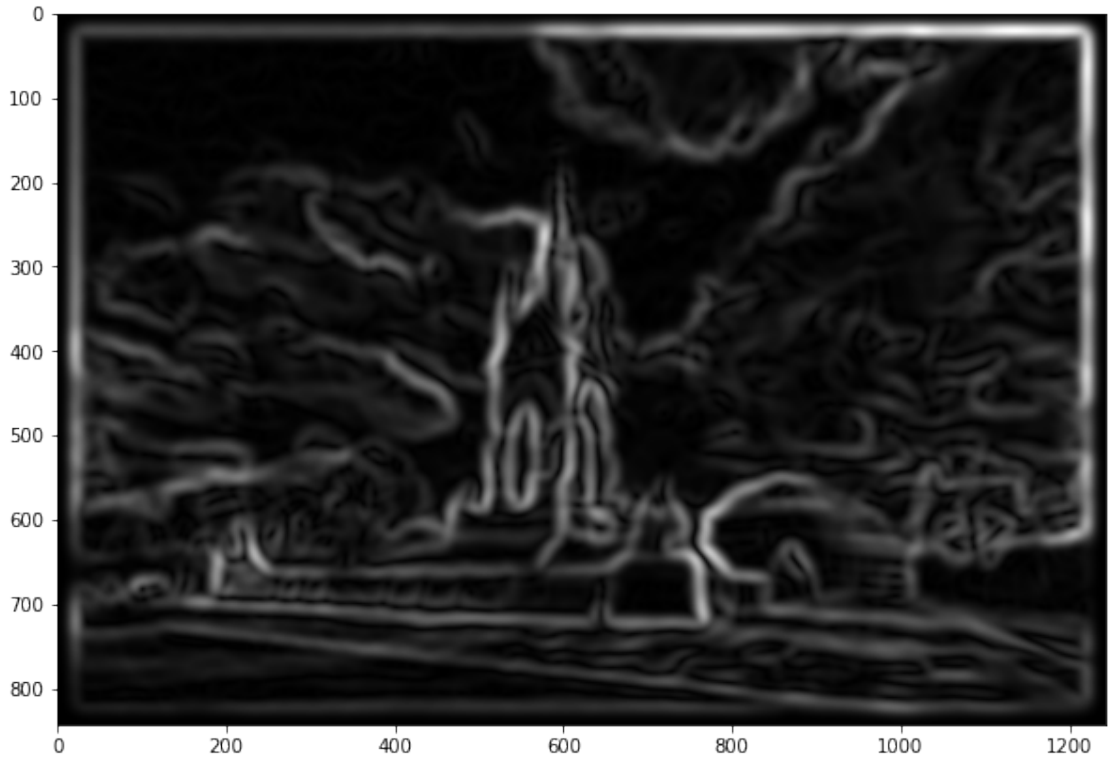
[ ]: <matplotlib.image.AxesImage at 0x221694b2e50>



1.4.6 2.6 Perform Gaussian smoothing ($\sigma = 7$ pixels) using two separable filters and evaluate the computational time for separable Gaussian filtering. After that, perform Prewitt filtering, show results and check whether the results are the same as the previous one without separable filtering. (9 points)

[ ]:
```
# Perform separable Gaussian smoothing and count time
start_time = time.time()
image_gauss_x_smoothed = scipy.signal.convolve2d(image_noisy, h_x)
image_gauss_smoothed = scipy.signal.convolve2d(image_gauss_x_smoothed, h_y)
end_time = time.time() - start_time
print("Separated Gaussian smoothing time taken", end_time)

# Prewitt filtering
image_sx = scipy.signal.convolve2d(image_gauss_smoothed, prewitt_x)
image_sy = scipy.signal.convolve2d(image_gauss_smoothed, prewitt_y)

# Calculate the gradient magnitude
grad_mag2 = np.sqrt(image_sx**2 + image_sy**2)

# Display the gradient magnitude image (provided)
plt.imshow(grad_mag2, cmap='gray')
plt.gcf().set_size_inches(10, 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.

# Mentioned in section 2.7
```

```
Separated Gaussian smoothing time taken 0.4664003849029541
```



1.4.7 2.7 Comment on the Gaussian + Prewitt filtering results and the computational time. (9 points)

By applying the Prewitt filter on images that have already been filtered with a Gaussian filter, there are several aspects in the filtering results to note:

1. The final Prewitt filtered image, with higher values of σ i.e. $\sigma = 7$, has thicker edges with reduced intensities as indicated with the softer color intensity. This is due to the fact that Gaussian smoothing with a higher σ value has a greater smoothing effect. The Gaussian filter is essentially a weighted moving average filter that spreads out the intensity i.e. smoothing out intensities. This results in the thicker lines but reduced intensity as seen in the image.
2. There are less edges detected by the Prewitt image with higher values of σ. A Gaussian filter with a higher σ value would have a greater smoothing effect and would "blur" some edges that are already low intensity to an intensity that the Prewitt filter cannot detect.

In computational times, there are several aspects in the times to note:

1. Computational time for convolution with separated 1D Gaussian filters ($t = 0.4664003849029541$) is faster compared to convolution with a 2D Gaussian filter ($t = 5.420028448104858$). It is approximately $11.621\times$ faster.
2. The resultant image for both separated 1D Gaussian filters and 2D Gaussian filter are the same.

1.5 3. Challenge: Implement a 2D Gaussian filter using Pytorch.

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.

[ ]:
```
# 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. (7 points)

[ ]:
```
# Expand the dimension of the numpy array
image_noisy_expanded = np.expand_dims(np.expand_dims(image_noisy, axis=0),␣
↪axis=0)

# Convert to a Pytorch tensor using torch.from_numpy
image_noisy_expanded_torch = torch.from_numpy(image_noisy_expanded).float()
```

1.5.2 3.2 Create a Pytorch Conv2D filter, set its kernel to be a 2D Gaussian filter. (7 points)

[ ]:
```
# A 2D Gaussian filter when sigma = 3 pixel (provided)
sigma = 3
h = gaussian_filter_2d(sigma)

# Create the Conv2D filter (provided)
conv = torch.nn.Conv2d(in_channels=1, out_channels=1, kernel_size=3, padding=1,␣
↪bias=False)

# Set the kernel weight
conv.weight.data = torch.nn.Parameter(torch.ones((1,1,3,3))/9)
```

1.5.3 3.3 Apply the filter to the noisy image tensor and display the output image. (6 points)

[ ]:
```
# Filtering
image_filtered = conv(image_noisy_expanded_torch)

# Display the filtering result (provided)
plt.imshow(image_filtered.detach().numpy().reshape((image_filtered.shape[2],␣
↪image_filtered.shape[3])), cmap='gray')
plt.gcf().set_size_inches(10, 8)
```

1.6 4. Survey: How long does it take you to complete the coursework?

5 Hours