coursework_01

February 3, 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.

[1]:
```
# 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()`.

[2]:
```
# Read the image (provided)
image = imageio.imread('hyde_park.jpg')
plt.imshow(image, cmap='gray')
plt.gcf().set_size_inches(10, 8)
```

[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(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)

[4]:
```
def create_average_filter(size):
    filter = []
    for i in range(size):
        row = []
        for j in range(size):
            row.append(1)
        filter.append(row)
    return np.array(filter)/size

start_time = time.time()
# Design the filter h
h = create_average_filter(3)

# Convolve the corrupted image with h using scipy.signal.convolve2d function
image_filtered = scipy.signal.convolve2d(image_noisy,h,boundary='wrap')

# 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)
print("This operation took --- %s seconds ---" % (time.time()-start_time))
```

```
Filter h:
[[0.33333333 0.33333333 0.33333333]
 [0.33333333 0.33333333 0.33333333]
 [0.33333333 0.33333333 0.33333333]]
This operation took --- 0.057874441146850586 seconds ---
```

1.3.3 1.2 Filter the noisy image with a 11x11 moving average filter. (5 points)

[5]:
```
start_time = time.time()
# Design the filter h
h= create_average_filter(11)

# Convolve the corrupted image with h using scipy.signal.convolve2d function
image_filtered = scipy.signal.convolve2d(image_noisy,h,boundary='symm') # uses␣
 ↪the symmetrical padding
```

```python
# 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)
print("This operation took --- %s seconds ---" % (time.time()-start_time))
```

```
Filter h:
[[0.09090909 0.09090909 0.09090909 0.09090909 0.09090909 0.09090909
  0.09090909 0.09090909 0.09090909 0.09090909 0.09090909]
 [0.09090909 0.09090909 0.09090909 0.09090909 0.09090909 0.09090909
  0.09090909 0.09090909 0.09090909 0.09090909 0.09090909]
 [0.09090909 0.09090909 0.09090909 0.09090909 0.09090909 0.09090909
  0.09090909 0.09090909 0.09090909 0.09090909 0.09090909]
 [0.09090909 0.09090909 0.09090909 0.09090909 0.09090909 0.09090909
  0.09090909 0.09090909 0.09090909 0.09090909 0.09090909]
 [0.09090909 0.09090909 0.09090909 0.09090909 0.09090909 0.09090909
  0.09090909 0.09090909 0.09090909 0.09090909 0.09090909]
 [0.09090909 0.09090909 0.09090909 0.09090909 0.09090909 0.09090909
  0.09090909 0.09090909 0.09090909 0.09090909 0.09090909]
 [0.09090909 0.09090909 0.09090909 0.09090909 0.09090909 0.09090909
  0.09090909 0.09090909 0.09090909 0.09090909 0.09090909]
 [0.09090909 0.09090909 0.09090909 0.09090909 0.09090909 0.09090909
  0.09090909 0.09090909 0.09090909 0.09090909 0.09090909]
 [0.09090909 0.09090909 0.09090909 0.09090909 0.09090909 0.09090909
  0.09090909 0.09090909 0.09090909 0.09090909 0.09090909]
 [0.09090909 0.09090909 0.09090909 0.09090909 0.09090909 0.09090909
  0.09090909 0.09090909 0.09090909 0.09090909 0.09090909]
 [0.09090909 0.09090909 0.09090909 0.09090909 0.09090909 0.09090909
  0.09090909 0.09090909 0.09090909 0.09090909 0.09090909]]
This operation took --- 0.26631903648376465 seconds ---
```

1.3.4 1.3 Comment on the filtering results. How do different kernel sizes influence the filtering results? (10 points)

With respect to the original noisy image, both the 3x3 filter and the 11x11 filter blur/smooth the image. This is done by setting the value of each pixel to the average of the NxN pixels around it. - The smaller 3x3 kernel was effective in reducing the noise while blurring out the image. The bigger 11x11 kernal reduced the noise by a larger amount, but resulted in a more blurred image. - In terms of boundary pixels, the boundary for a 3x3 would be small (1 pixel), while the edge for the 11x11 is much larger (5 pixels). This would be set to black with the default padding, but if we use "boundary='symm' ", we can set it to mirror values on the opposite side so we do not have a black edge. - In terms of operation time to filter the image, as seen in the print message added, the 3x3 kernel took roughly 0.045 seconds, while the 11x11 kernel took roughly 0.275 seconds. This difference in processing time is due to the computational complexity (both scale with image size N^2, but K^2 is larger for the 11x11). The complexity for both is $O(N^2K^2)$

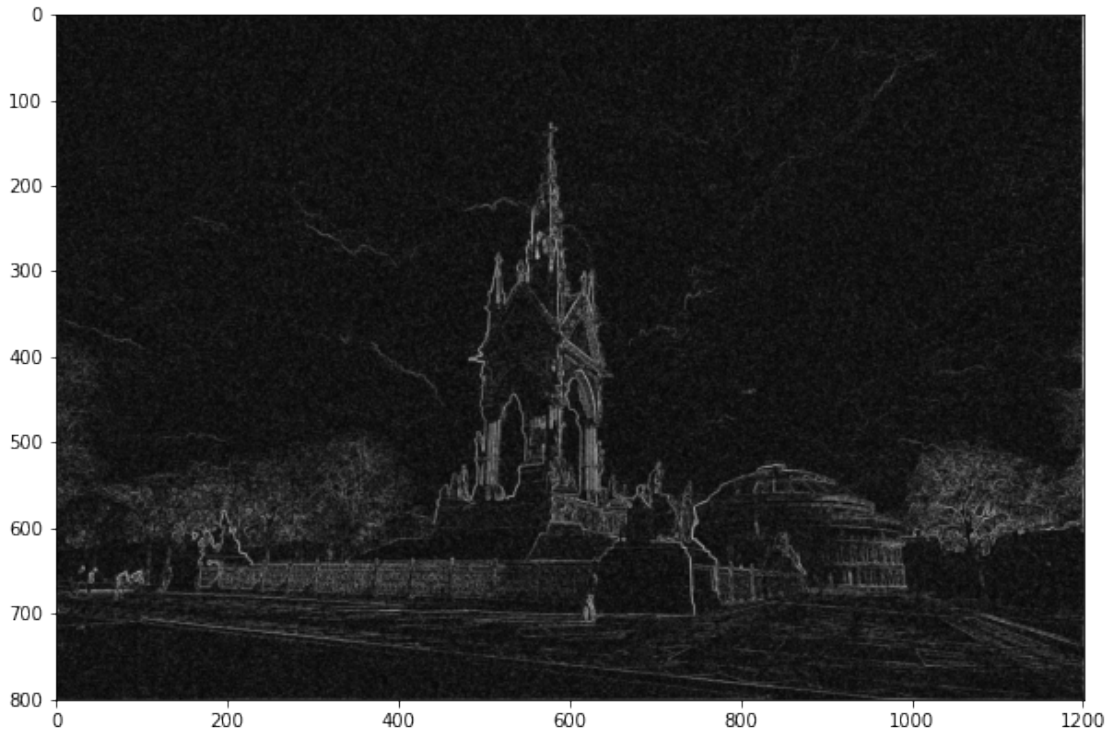
1.4 2. Edge detection.

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)

[6]:
```
# 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
prewitt_x_image = scipy.signal.convolve2d(image_noisy,prewitt_x)
prewitt_y_image = scipy.signal.convolve2d(image_noisy,prewitt_y)

# Calculate the gradient magnitude
grad_mag = np.hypot(prewitt_x_image,prewitt_y_image) # hypot calculates the␣
 ↪gradient magnitude

# 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]]
```

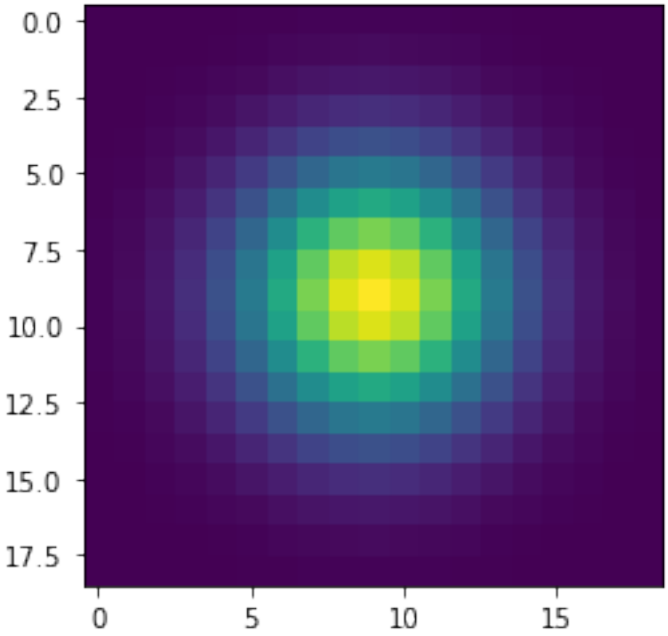


1.4.2 2.2 Implement a function that generates a 2D Gaussian filter given the parameter σ. (10 points)

[7]:
```
# Design the Gaussian filter
def gaussian_filter_2d(sigma):
    # ======== Previous answer using SciPy =========
    # identity = np.zeros((6*sigma+1,6*sigma+1)) # set size of kernel
    # identity[3*sigma][3*sigma] = 1
    # h = scipy.ndimage.gaussian_filter(np.float_(identity), sigma=sigma)
    # return h
    size = sigma*6+1
    x, y = np.mgrid[-size//2 + 1:size//2 + 1, -size//2 + 1:size//2 + 1]
    g = np.exp(-((x**2 + y**2)/(2.0*sigma**2)))
    return g/g.sum()

# Visualise the Gaussian filter when sigma = 3 pixel (provided)
sigma = 3
h = gaussian_filter_2d(sigma)
print(h.shape)
plt.imshow(h)
```

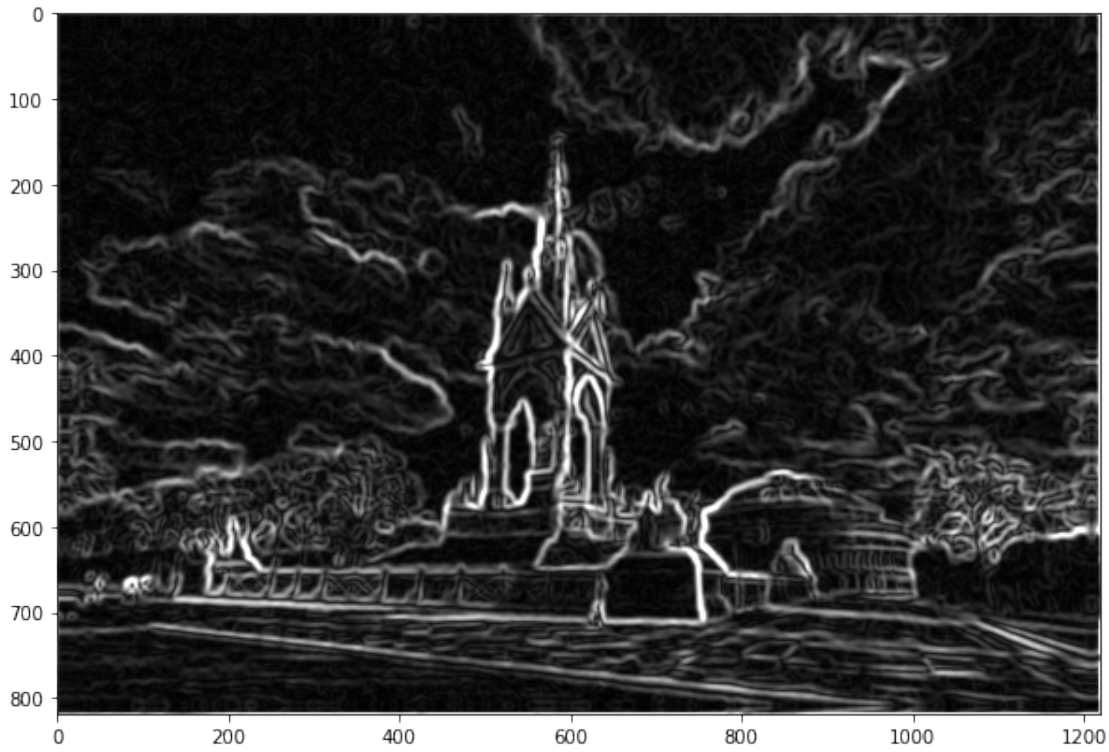
```
(19, 19)
```

[7]: <matplotlib.image.AxesImage at 0x1cbc1186790>



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

[8]:
```
# Perform Gaussian smoothing before Prewitt filtering
h = gaussian_filter_2d(3)
gauss_image = scipy.signal.convolve2d(image_noisy,h,boundary='symm')

# Prewitt filtering
prewitt_x_image = scipy.signal.convolve2d(gauss_image,prewitt_x)
prewitt_y_image = scipy.signal.convolve2d(gauss_image,prewitt_y)

# Calculate the gradient magnitude
grad_mag = np.hypot(prewitt_x_image,prewitt_y_image)

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

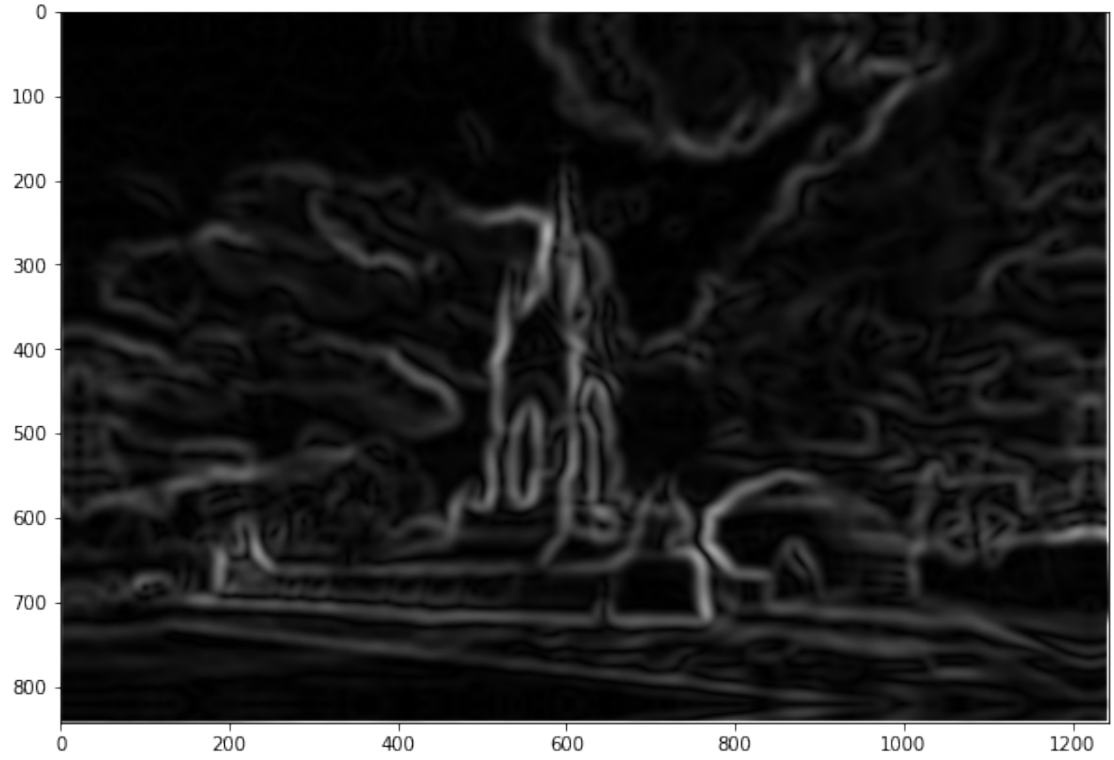


1.4.4 2.4 Perform Gaussian smoothing ($\sigma = 7$ pixels) and evaluate the computational time for Gaussian smoothing. After that, perform Prewitt filtering. (7 points)

[9]:
```
# Construct the Gaussian filter
h = gaussian_filter_2d(7)
print(h)

# Perform Gaussian smoothing and count time
start_time = time.time()
gauss_image = scipy.signal.convolve2d(image_noisy,h,boundary='symm')
print("The gaussian smoothing took --- %s seconds ---" % (time.
  ↪time()-start_time))

# Prewitt filtering
prewitt_x_image = scipy.signal.convolve2d(gauss_image,prewitt_x)
prewitt_y_image = scipy.signal.convolve2d(gauss_image,prewitt_y)

# Calculate the gradient magnitude
grad_mag = np.hypot(prewitt_x_image,prewitt_y_image)

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

```
plt.gcf().set_size_inches(10, 8)
```

```
[[4.02540837e-07 6.11652247e-07 9.10617627e-07 … 9.10617627e-07
  6.11652247e-07 4.02540837e-07]
 [6.11652247e-07 9.29392590e-07 1.38366413e-06 … 1.38366413e-06
  9.29392590e-07 6.11652247e-07]
 [9.10617627e-07 1.38366413e-06 2.05997600e-06 … 2.05997600e-06
  1.38366413e-06 9.10617627e-07]
 …
 [9.10617627e-07 1.38366413e-06 2.05997600e-06 … 2.05997600e-06
  1.38366413e-06 9.10617627e-07]
 [6.11652247e-07 9.29392590e-07 1.38366413e-06 … 1.38366413e-06
  9.29392590e-07 6.11652247e-07]
 [4.02540837e-07 6.11652247e-07 9.10617627e-07 … 9.10617627e-07
  6.11652247e-07 4.02540837e-07]]
The gaussian smoothing took --- 4.089043378829956 seconds ---
```

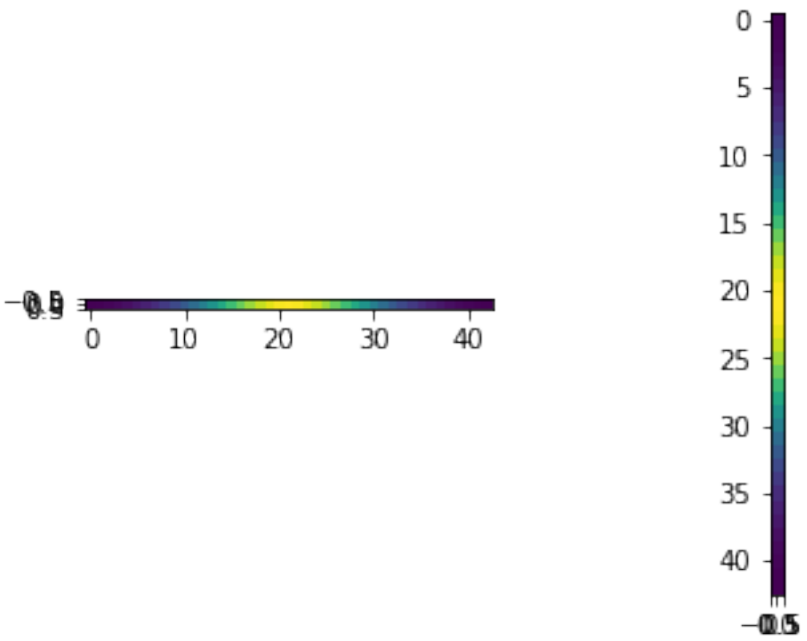


1.4.5 2.5 Implement a function that generates a 1D Gaussian filter given the parameter σ. Generate 1D Gaussian filters along x-axis and y-axis respectively. (10 points)

[10]:
```
# Design the Gaussian filter
def gaussian_filter_1d(sigma):
    # identity = np.zeros(6*sigma+1) # set size of kernel
    # identity[3*sigma] = 1
    # h = scipy.ndimage.gaussian_filter1d(np.float_(identity), sigma=sigma)
    # return h
    length = sigma*6+1
    x = np.linspace(-(length-1)/2., (length-1)/2., length)
    x = np.exp(-((x**2)/(2.0*sigma**2)))
    return x/np.sum(x)
print(gaussian_filter_1d(3))

# sigma = 7 pixel (provided)
sigma = 7

# The Gaussian filter along x-axis. Its shape is (1, sz).
h_x = gaussian_filter_1d(sigma)
h_x = np.resize(h_x,(1,6*sigma+1))

# The Gaussian filter along y-axis. Its shape is (sz, 1).
h_y = gaussian_filter_1d(sigma)
h_y = np.resize(h_y,(6*sigma+1,1))

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

```
[0.00147945 0.00380424 0.00875346 0.01802341 0.03320773 0.05475029
 0.08077532 0.106639   0.12597909 0.133176   0.12597909 0.106639
 0.08077532 0.05475029 0.03320773 0.01802341 0.00875346 0.00380424
 0.00147945]
```

[10]:
```
<matplotlib.image.AxesImage at 0x1cbbcd35f40>
```

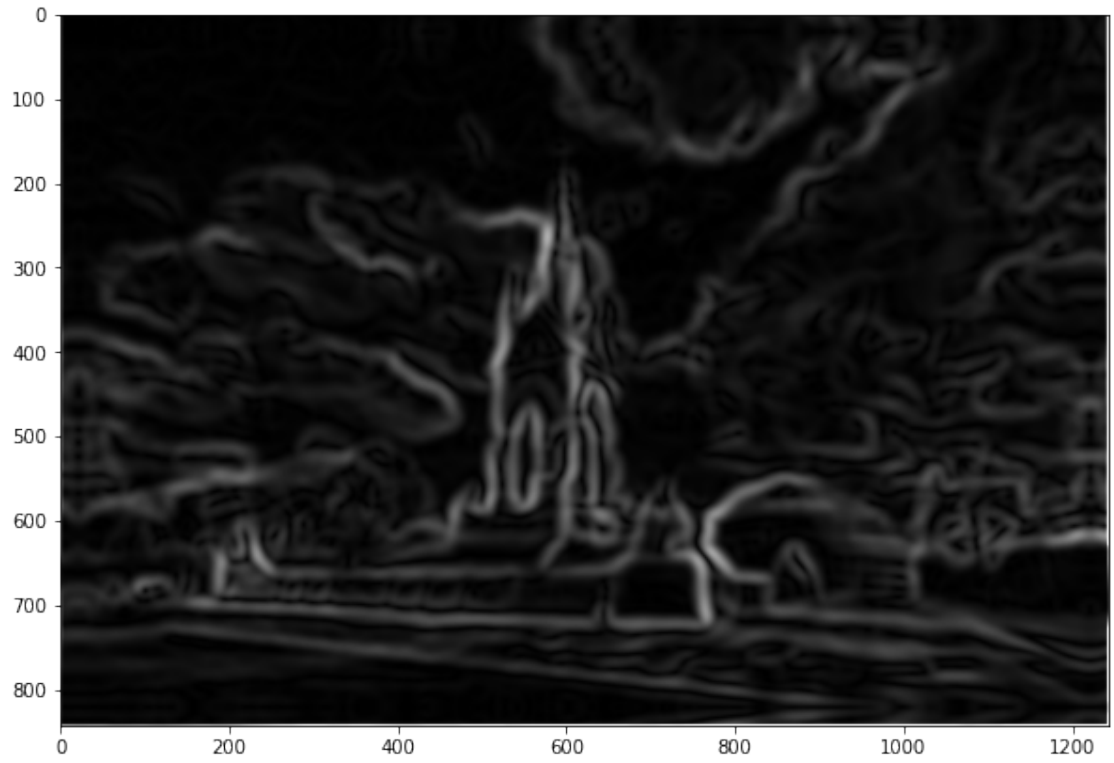


1.4.6 2.6 Perform Gaussian smoothing ($\sigma = 7$ pixels) using two separable filters and evaluate the computational time for separable Gaussian filtering. After that, perform Prewitt filtering, show results and check whether the results are the same as the previous one without separable filtering.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

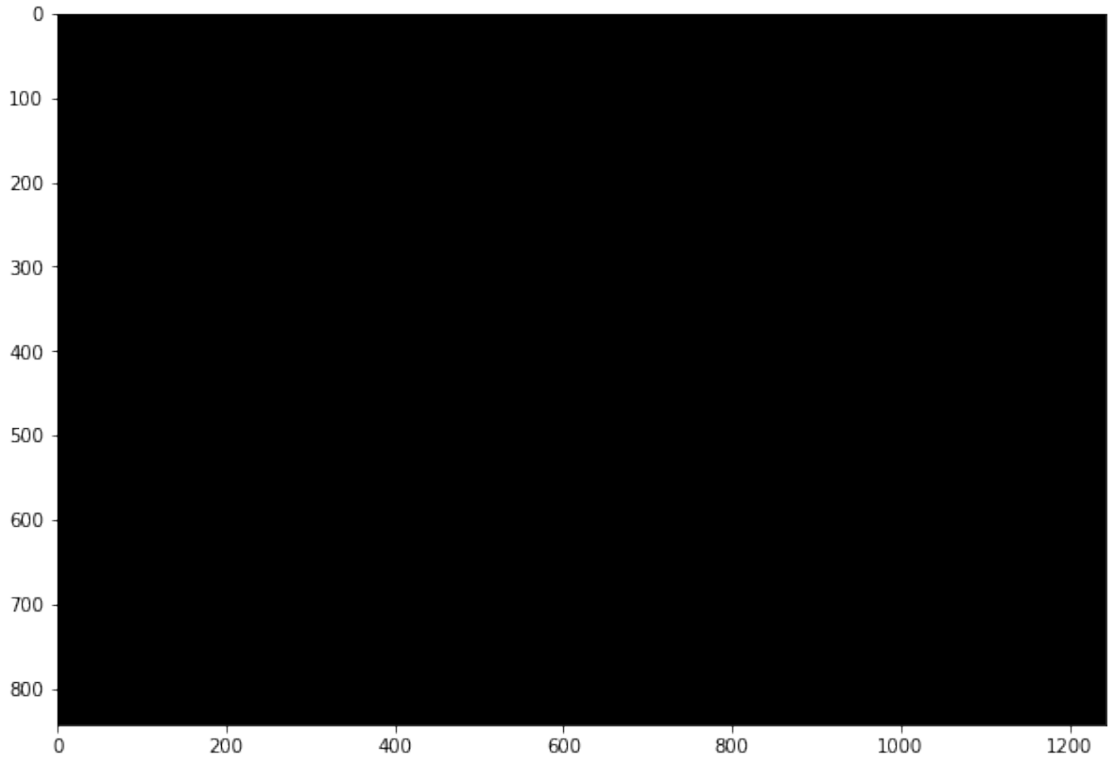
[11]:
```
# Perform separable Gaussian smoothing and count time
sigma = 7
h_x = gaussian_filter_1d(sigma)
h_x = np.resize(h_x,(1,6*sigma+1))
h_y = gaussian_filter_1d(sigma)
h_y = np.resize(h_y,(6*sigma+1,1))
print((h_x*h_y).shape)

start_time = time.time()
gauss_image_sep = scipy.signal.convolve2d(image_noisy,h_x,boundary='symm')
gauss_image_sep = scipy.signal.convolve2d(gauss_image_sep,h_y,boundary='symm')␣
 ↪# combining separable filters here
print("The gaussian smoothing took --- %s seconds ---" % (time.
 ↪time()-start_time))

# Prewitt filtering
prewitt_x_image = scipy.signal.convolve2d(gauss_image_sep,prewitt_x)
prewitt_y_image = scipy.signal.convolve2d(gauss_image_sep,prewitt_y)

# Calculate the gradient magnitude
```

```
grad_mag2 = np.hypot(prewitt_x_image,prewitt_y_image)

# Display the gradient magnitude image (provided)
plt.imshow(grad_mag2, cmap='gray', vmin=0, vmax=100)
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.
```

```
(43, 43)
The gaussian smoothing took --- 0.3341386318206787 seconds ---
```



[12]:
```
grad_mag_diff = grad_mag-grad_mag2
plt.imshow(grad_mag_diff, cmap='gray', vmin=0, vmax=100)
plt.gcf().set_size_inches(10, 8)
```



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

- Using a 2D convoulution with a sigma size of 7 results in a 42x42 kernel (assuming k is 3). Using such a size of kernel would result in high computational power and requires much more time. In this case, the filtering took ~4.48 seconds.
- On the other hand, using separable 1D convolutions of the same gaussian filter, we can see that h_x*h_y still has a shape of 42x42, but convolving them separately resulted in the gaussian filtering taking significantly shorter time ~0.36 seconds.
- By comparing the Prewitt filtering results from the two typed of Gaussian smoothing, we can see they result in the same image, but the separable convolultions take significantly less time and computational power. In the second cell, we calculate the difference between the gradient magnitude using the 2D filter vs. the separable 1D filters, in which case we see all black, indicating that they are the same gradient magnitude.

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.

[13]:
```
# 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)

[14]:
```
# Expand the dimension of the numpy array
noisy_expanded = np.array([[image_noisy]])
print(noisy_expanded.shape)

# Convert to a Pytorch tensor using torch.from_numpy
noisy_tensor = torch.from_numpy(noisy_expanded)
```

```
(1, 1, 799, 1200)
```

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

[15]:
```
# A 2D Gaussian filter when sigma = 3 pixel (provided)
sigma = 3
h = gaussian_filter_2d(sigma)
print(h.shape)

# Create the Conv2D filter (provided)
conv = torch.nn.Conv2d(1,1,3*sigma+1, stride=1, padding=1)
print(conv)

h_arr = torch.from_numpy(h)
h_arr = torch.unsqueeze(h_arr, 0)
h_arr = torch.unsqueeze(h_arr, 0)

# Set the kernel weight
conv.weight.data = torch.nn.Parameter(h_arr)
```

```
(19, 19)
Conv2d(1, 1, kernel_size=(10, 10), stride=(1, 1), padding=(1, 1))
```

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

[16]:
```
# Filtering
image_filtered = conv(noisy_tensor)
print(image_filtered)

# Display the filtering result (provided)
plt.imshow(image_filtered[0][0].data.numpy(), cmap='gray')
plt.gcf().set_size_inches(10, 8)
```

```
tensor([[[[ 66.5912,  66.9516,  67.0214,  …, 205.5237, 203.5880, 201.5096],
          [ 66.1429,  66.5218,  66.6689,  …, 204.6073, 203.0256, 201.3191],
          [ 65.3857,  65.7350,  65.9375,  …, 203.1621, 201.9139, 200.5629],
          …,
          [ 18.7689,  19.1651,  19.7867,  …,  47.9766,  49.2866,  50.5811],
          [ 20.3869,  20.6846,  21.1841,  …,  41.6389,  43.0169,  44.4275],
          [ 21.9791,  22.2041,  22.5848,  …,  36.0889,  37.3821,  38.7675]]]],
       dtype=torch.float64, grad_fn=<ThnnConv2DBackward>)
```

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

Maybe around 8 hours