

CV2_HW1_CS4090

February 17, 2022

Submitted By: Chandan Suri, CS4090

In Collaboration with: Gursifath Bhasin, GB2760

1 COMS 4731 Computer Vision – Homework 2

- This homework contains the following components:
 - **Problem 1: Image Denoising (40 points)**
 - * Implement a mean filter using “for” loop.
 - * Implement the `convolve_image` function.
 - * Implement a mean filter using a filter matrix.
 - * Implement a Gaussian filter.
 - **Problem 2: Edge Detection (30 points)**
 - * Implement a Delta (Gradient) filter.
 - * Implement a Laplacian filter.
 - **Problem 3: Hybrid Images (30 points)**
 - * Fourier transform.
 - * Implement low and high pass filters and apply them to images.
 - * Create a hybrid image using high-pass and low-pass filtered images.
- Your job is to implement the sections marked with TODO to complete the tasks. Please read through all the questions before starting. Good luck.
- Submission.
 - Please submit the notebook (ipynb and pdf) including the output of all cells. Please note that you should export your completed notebook as a PDF (CV2_HW1_UNI.pdf) and upload to GradeScope. Then, please submit the executable completed notebook (CV2_HW1_UNI.ipynb) to Cousework. For both files, 1) remember to replace with your own uni; 2) the completed notebook should show your code, as well as the final combined image you created.

2 Problem 1: Image Denoising

Taking pictures at night is challenging because there is less light that hits the film or camera sensor. To still capture an image in low light, we need to change our camera settings to capture more light. One way is to increase the exposure time, but if there is motion in the scene, this leads to blur. Another way is to use sensitive film that still responds to low intensity light. However, the trade-off

is that this higher sensitivity increases the amount of noise captured, which often shows up as grain on photos. In this problem, your task is to clean up the noise with signal processing.

2.1 Visualizing the Grain

To start off, let's load up the image and visualize the image we want to denoise.

```
[ ]: import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from IPython import display
from scipy.signal import convolve2d
from math import *
import time
%matplotlib inline

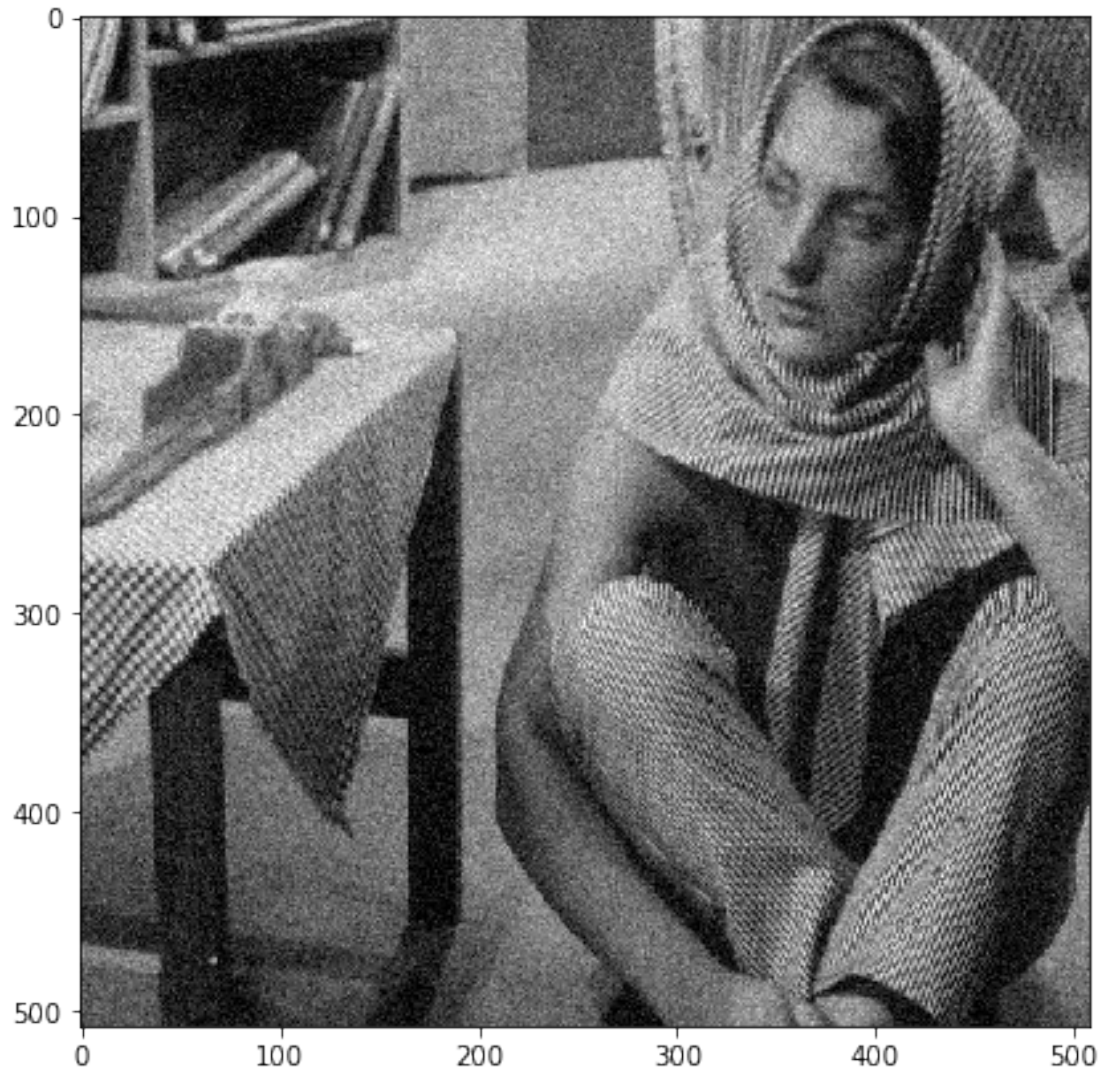
plt.rcParams['figure.figsize'] = [7, 7]

def load_image(filename):
    img = np.asarray(Image.open(filename))
    img = img.astype("float32") / 255.
    return img

def gray2rgb(image):
    return np.repeat(np.expand_dims(image, 2), 3, axis=2)

def show_image(img):
    if len(img.shape) == 2:
        img = gray2rgb(img)
    plt.imshow(img, interpolation='nearest')

# load the image
im = load_image('noisy_image.jpg')
im = im.mean(axis=2) # convert to grayscale
show_image(im)
```



2.2 Mean Filter using “for” loop

Let's try to remove this grain with a mean filter. For every pixel in the image, we want to take an average (mean) of the neighboring pictures. Implement this operation using “for” loops and visualize the result:

```
[ ]: pad_size = 2
im_pad = np.pad(im, pad_size, mode='constant') # pad the border of the
↳ original image

def apply_mean_filter():
    im_out = np.zeros_like(im) # initialize the output image array
    num_rows, num_cols = im.shape
```

```

    for row_idx in range(num_rows):
        for col_idx in range(num_cols):
            im_out[row_idx][col_idx] = np.mean(im_pad[row_idx - pad_size:
↪row_idx + pad_size + 1,\
                                                    col_idx - pad_size:pad_size+
↪col_idx + pad_size + 1])

    return im_out

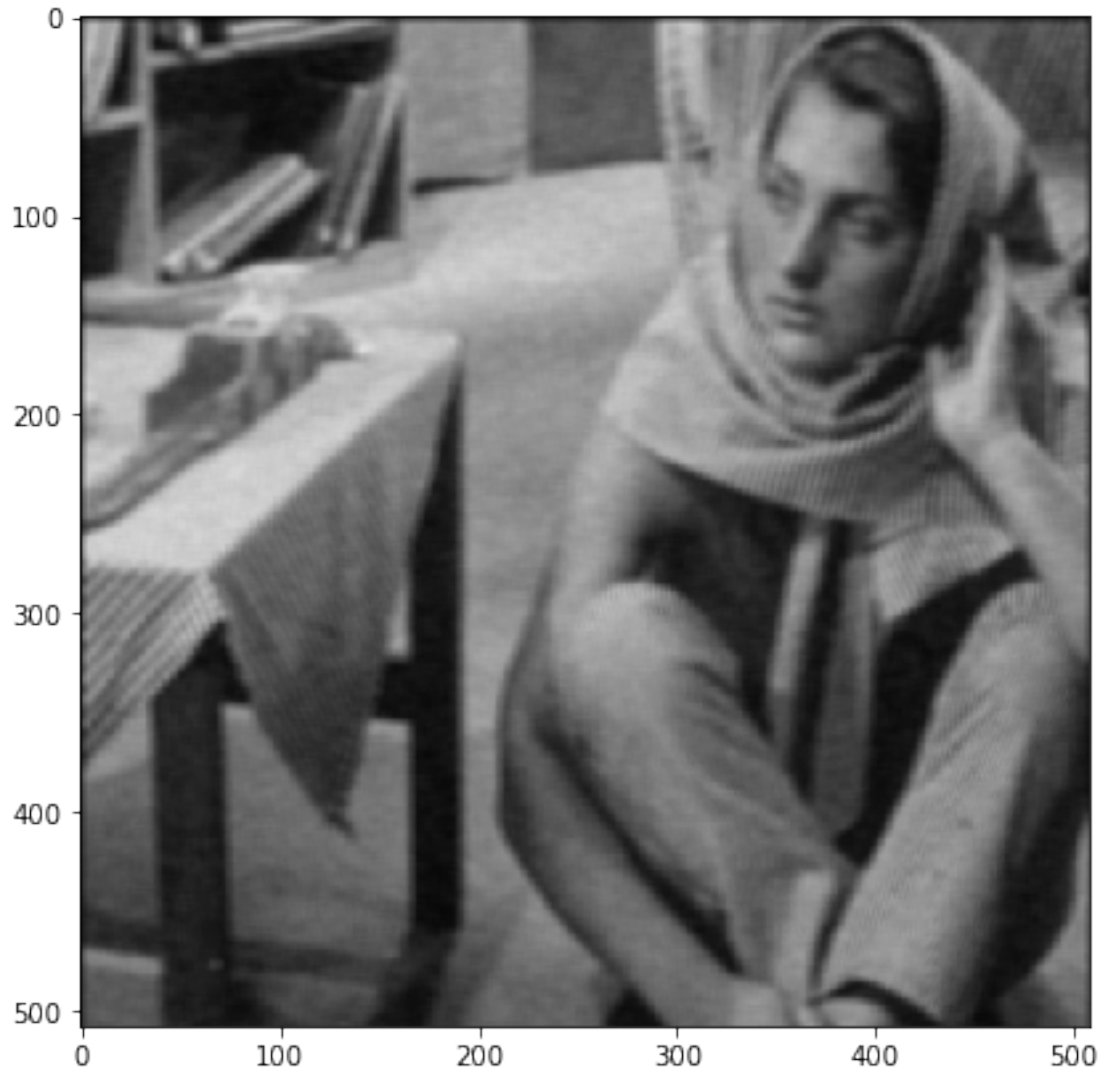
im_out = apply_mean_filter()
show_image(im_out)

```

```

/Users/chandansuri/opt/anaconda3/lib/python3.8/site-
packages/numpy/core/fromnumeric.py:3372: RuntimeWarning: Mean of empty slice.
    return _methods._mean(a, axis=axis, dtype=dtype,
/Users/chandansuri/opt/anaconda3/lib/python3.8/site-
packages/numpy/core/_methods.py:170: RuntimeWarning: invalid value encountered
in true_divide
    ret = ret.dtype.type(ret / rcount)

```



2.3 Implement the `convolve_image` function.

In practice, applying filters to images can be more efficient by using convolution, which is a function that takes as input the raw image and a filter matrix, and outputs the convolved image. Implement your `convolve_image` function below.

```
[ ]: def convolve_image(image, filter_matrix):  
    ''' Convolve a 2D image using the filter matrix.  
    Args:  
        image: a 2D numpy array.  
        filter_matrix: a 2D numpy array.  
    Returns:  
        the convolved image, which is a 2D numpy array same size as the input_  
        ↪ image.
```

```

TODO: Implement the convolve_image function here.
'''

num_rows, num_cols = image.shape
filter_size = filter_matrix.shape[0]
im_output = np.zeros_like(image)

# I am doing padding here rather than removing some of the rows or columns
↳when applying the kernel.
im_pad = np.pad(im, filter_size//2, mode='constant')

for row_idx in range(num_rows):
    for col_idx in range(num_cols):
        spliced_image_subset = im_pad[row_idx: row_idx + filter_size,
↳col_idx: col_idx + filter_size]
        im_output[row_idx][col_idx] = np.sum(filter_matrix *
↳spliced_image_subset)

return im_output

```

2.4 Mean Filter with Convolution

Implement this same operation with a convolution instead. Fill in the mean filter matrix here, and visualize the convolution result.

```

[ ]: ''' TODO: Create a mean filter matrix here. '''
mean_filt = np.full((5, 5), 1/25)

convolve_image(im, mean_filt)

```

```

[ ]: array([[0.24705882, 0.33505884, 0.42180392, ..., 0.2012549 , 0.16407843,
           0.12047059],
          [0.32611766, 0.44015688, 0.55403924, ..., 0.26384315, 0.21019608,
           0.15450981],
          [0.40439218, 0.5381961 , 0.6787451 , ..., 0.34839216, 0.27482355,
           0.1970196 ],
          ...,
          [0.18164706, 0.25286275, 0.32501963, ..., 0.39937255, 0.3217255 ,
           0.24329412],
          [0.1427451 , 0.20235294, 0.25552943, ..., 0.32          , 0.25082353,
           0.19215687],
          [0.10368628, 0.14964706, 0.18933333, ..., 0.24674511, 0.19639216,
           0.15137255]], dtype=float32)

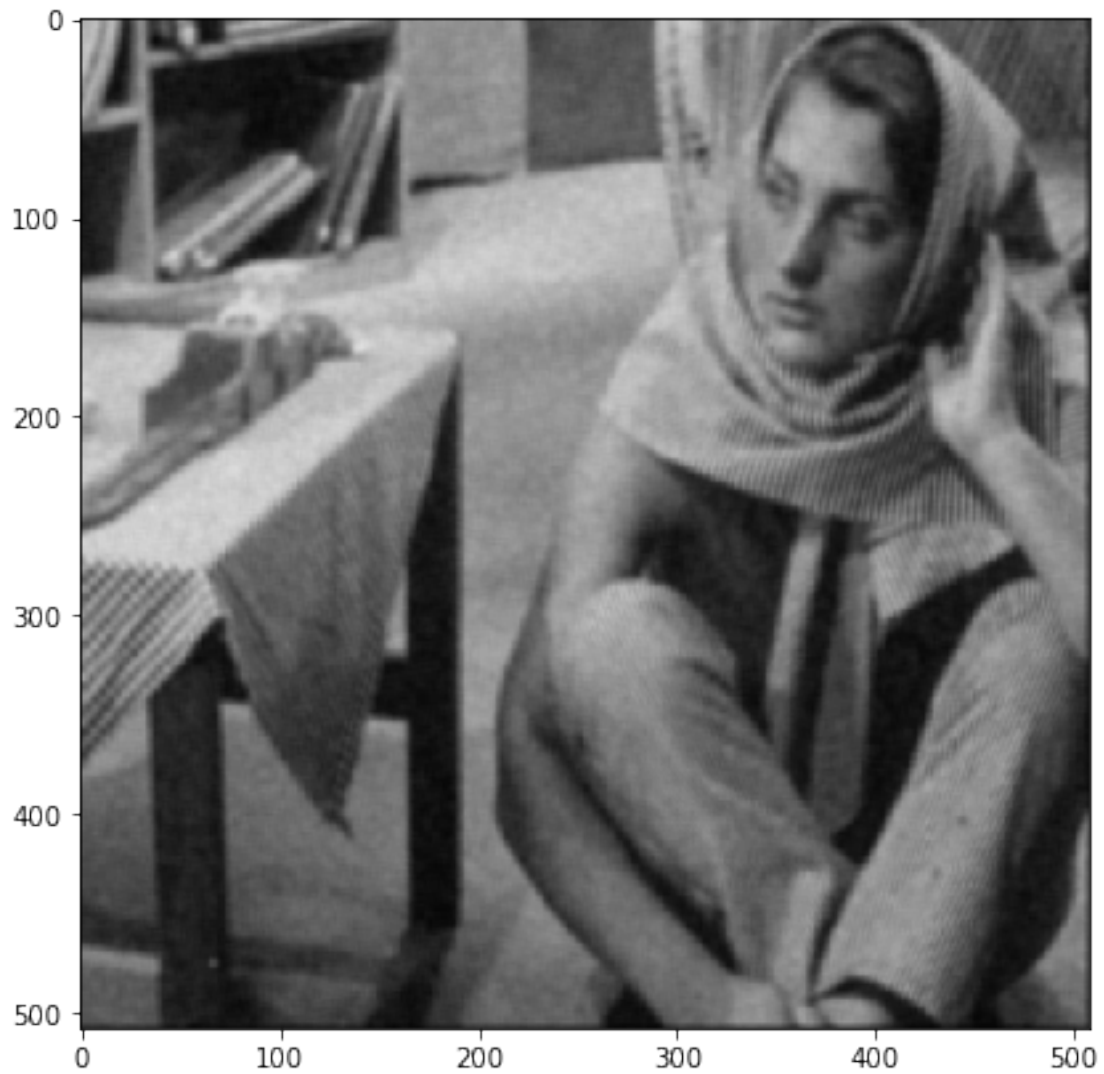
```

Apply mean filter convolution using your `convolve_image` function and the `mean_filt` matrix.

```

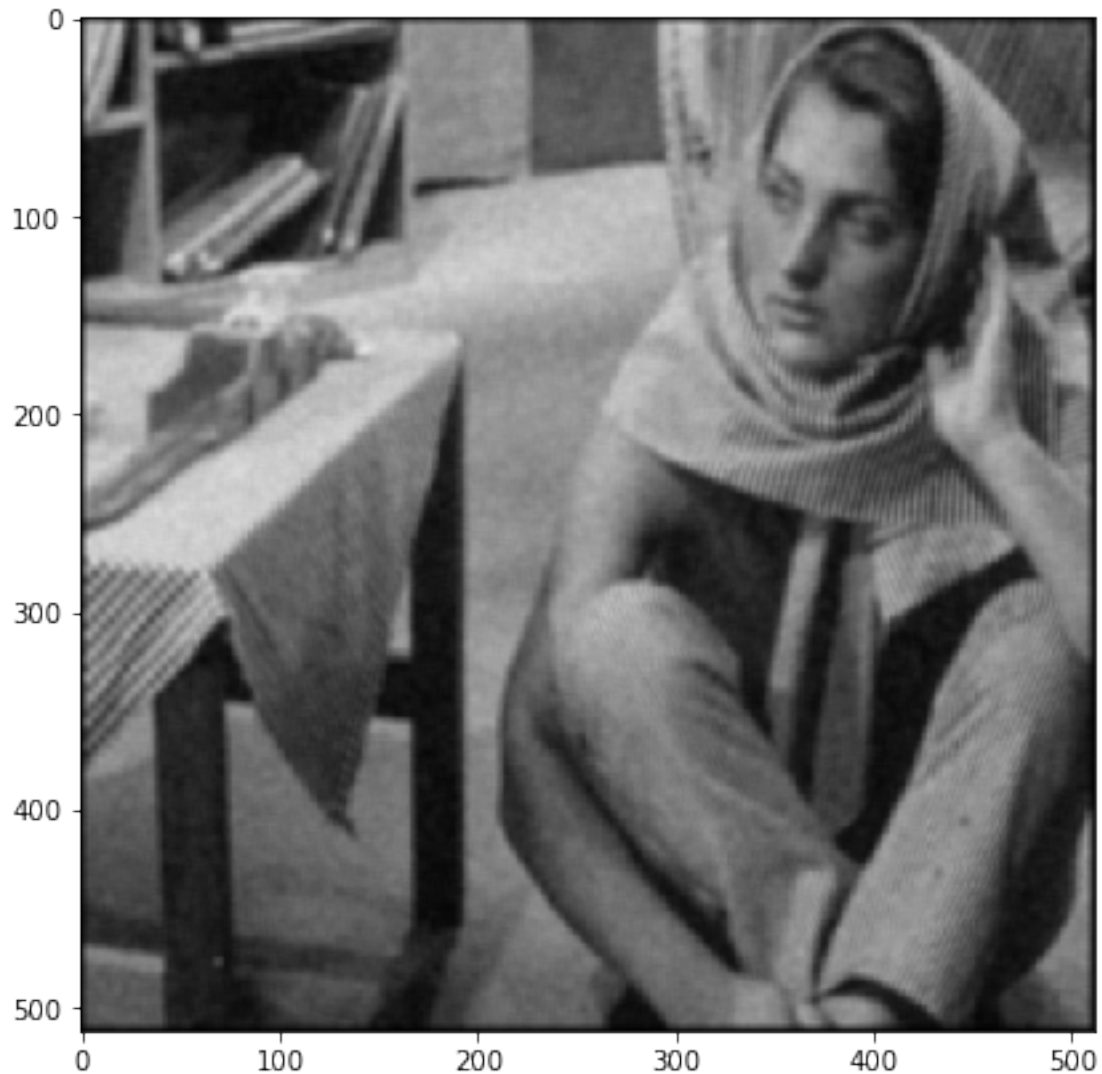
[ ]: show_image(convolve_image(im, mean_filt))

```



Compare your convolution result with the `scipy.signal.convolve2d` function (they should look the same).

```
[ ]: show_image(convolve2d(im, mean_filt))
```



Note: In the sections below, we will use the `scipy.signal.convolve2d` function for grading. But feel free to test your `convolve_image` function on other filters as well.

2.5 Gaussian Filter

Instead of using a mean filter, let's use a Gaussian filter. Create a 2D Gaussian filter, and plot the result of the convolution.

Hint: You can first construct a one dimensional Gaussian, then use it to create a 2D dimensional Gaussian.

```
[ ]: def gaussian_filter(sigma, k=20):  
    '''  
    Args:
```



```

    sigma: the standard deviation of Gaussian kernel.
    k: controls size of the filter matrix.
    Returns:
        a 2D Gaussian filter matrix of the size (2k+1, 2k+1).

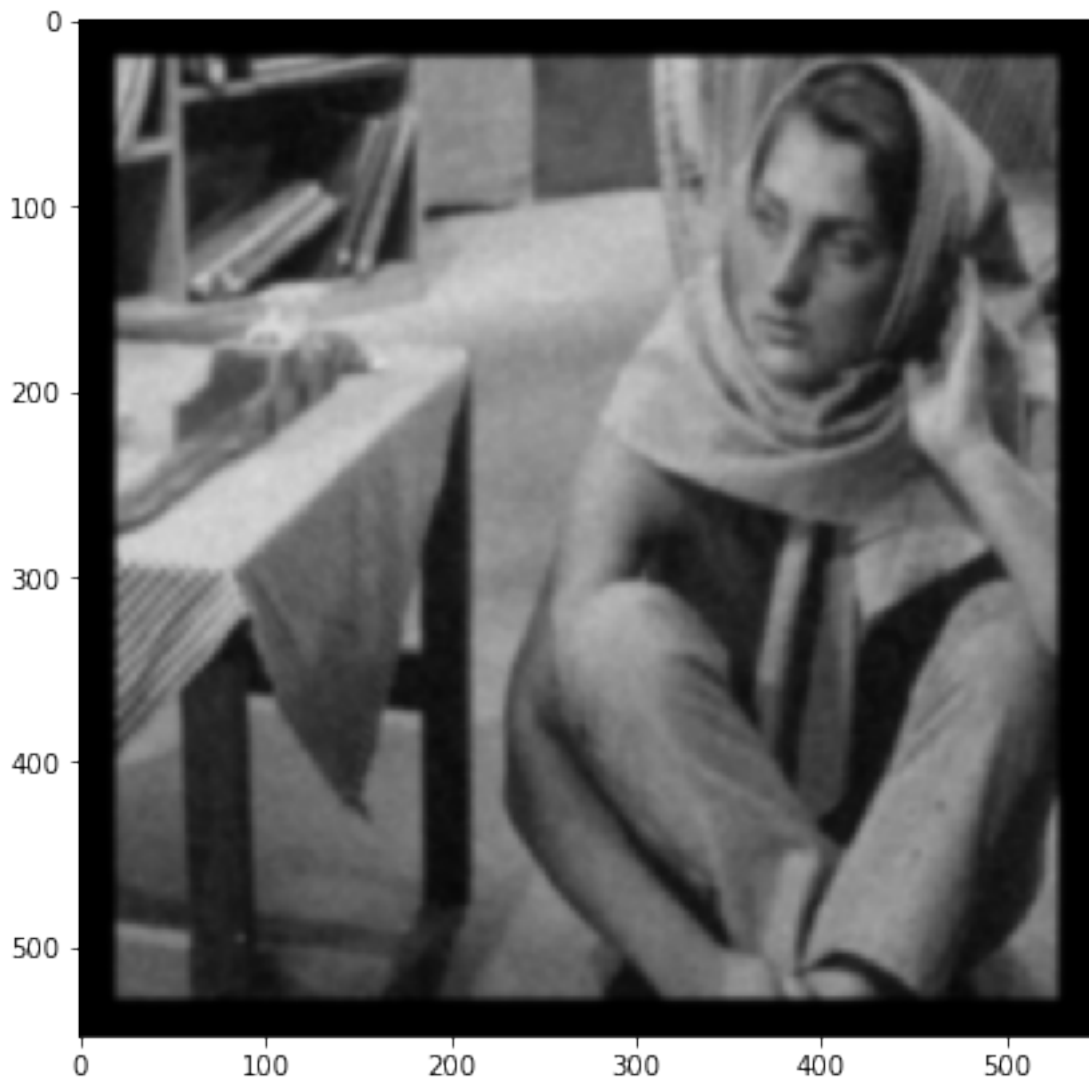
    TODO: Implement a Gaussian filter here.
    '''

    gaussian_values = np.array([exp(-x**2/(2*sigma**2)) / (sqrt(2*pi)*sigma)
    ↪ for x in range(-k, k + 1)])
    gaussian_filter_vals = np.outer(gaussian_values, gaussian_values)

    return gaussian_filter_vals

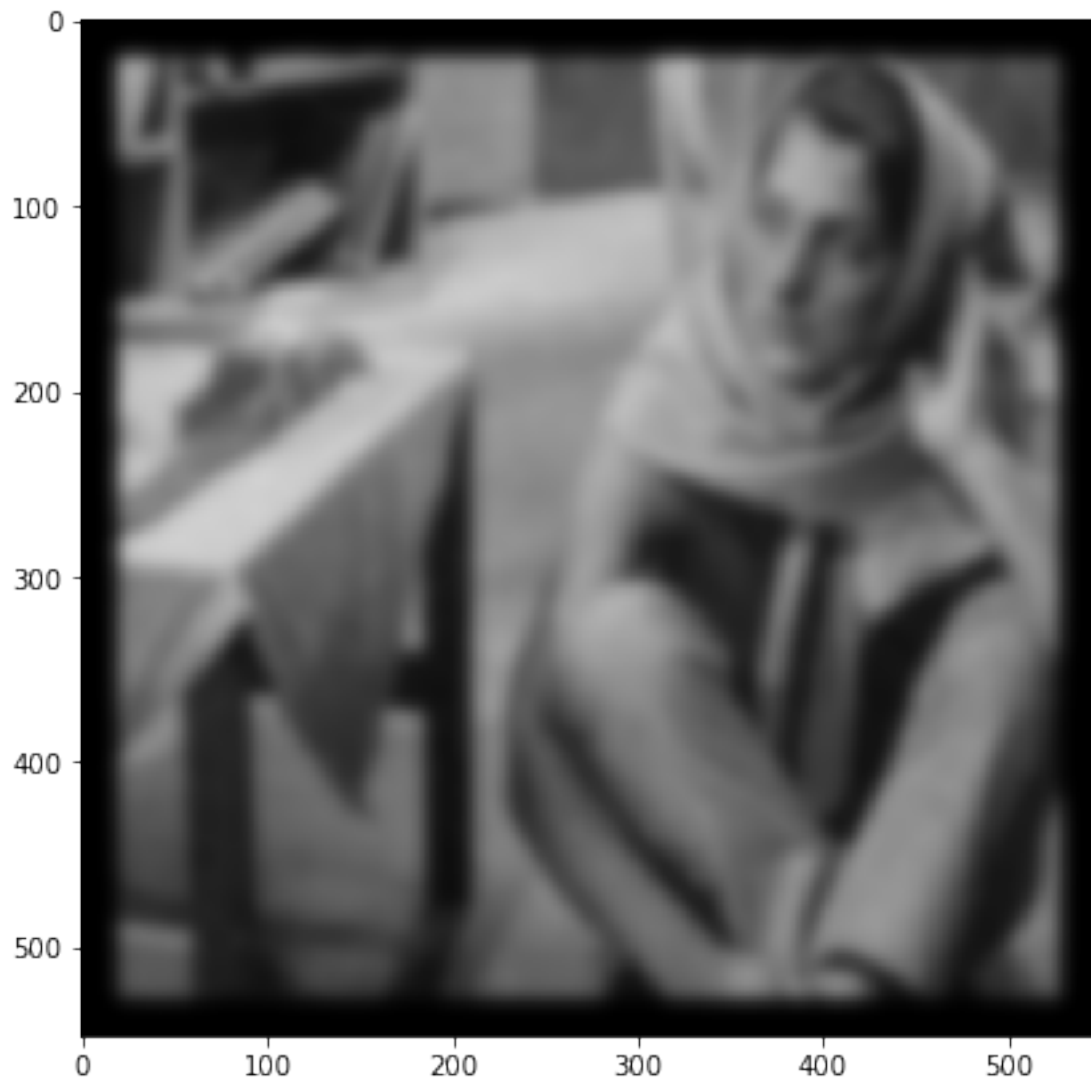
show_image(convolve2d(im, gaussian_filter(2)))

```

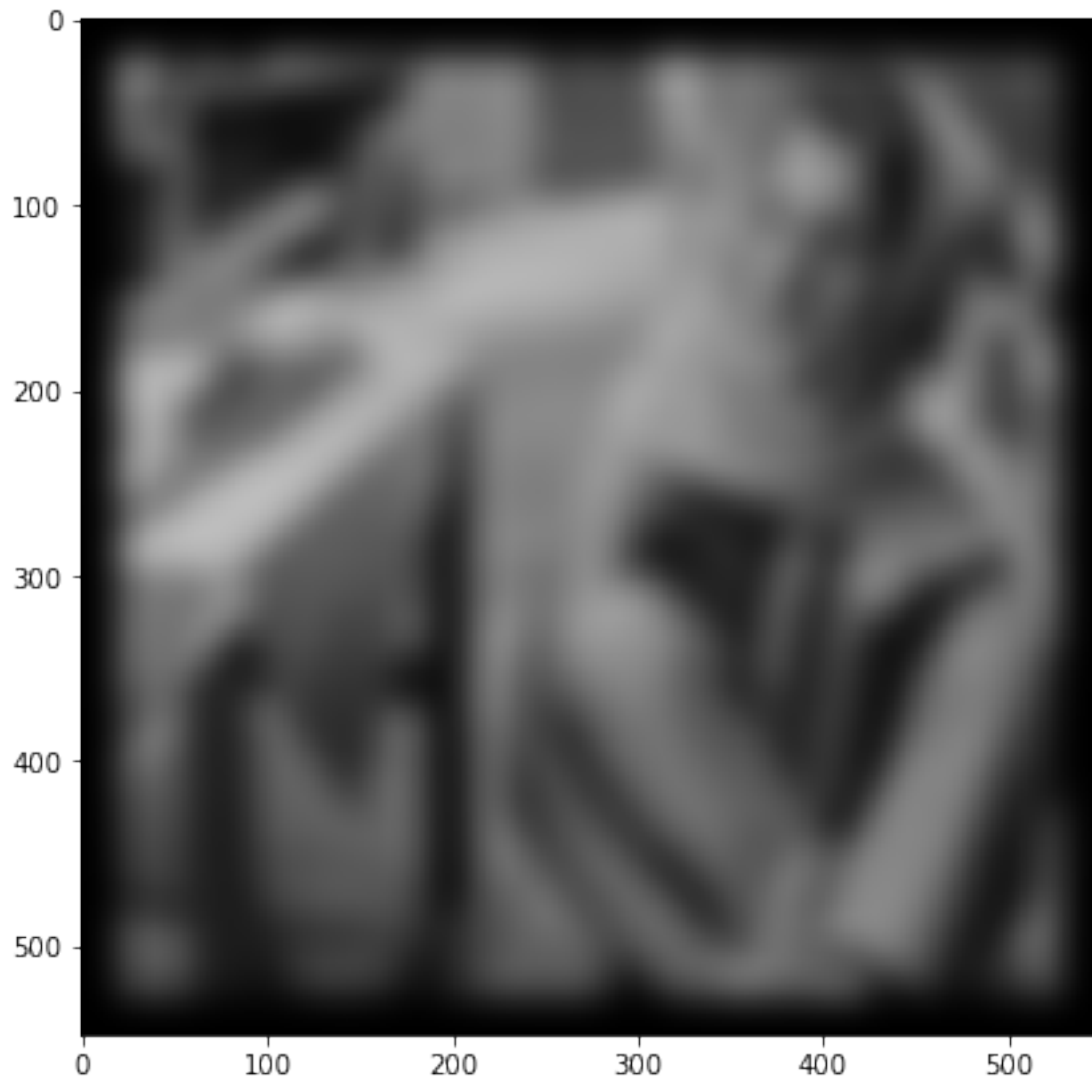


The amount the image is blurred changes depending on the sigma parameter. Change the sigma parameter to see what happens. Try a few different values.

```
[ ]: show_image(convolve2d(im, gaussian_filter(5)))
```



```
[ ]: show_image(convolve2d(im, gaussian_filter(10)))
```

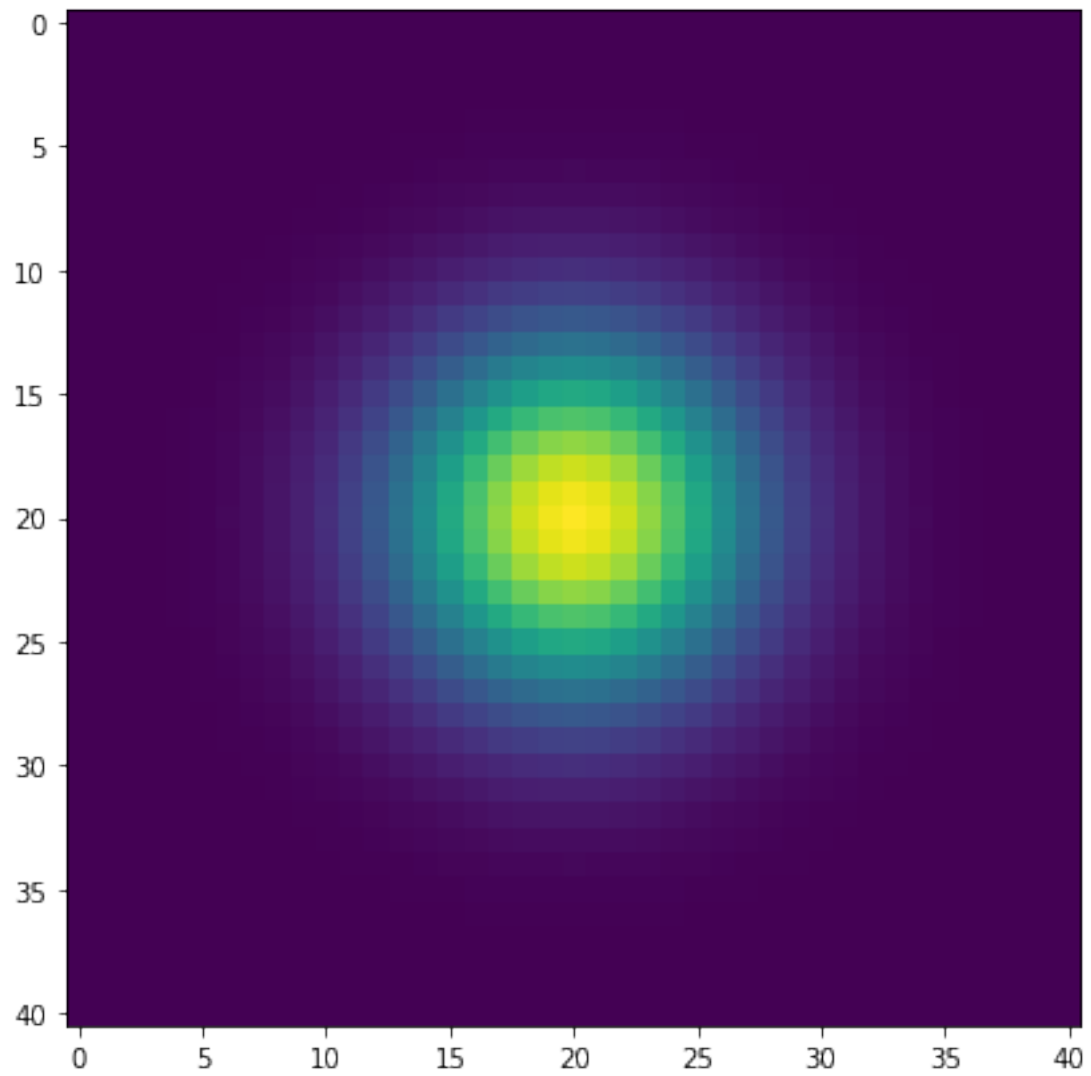


2.6 Visualizing Gaussian Filter

Try changing the sigma parameter below to visualize the Gaussian filter directly. This gives you an idea of how different sigma values create different convolved images.

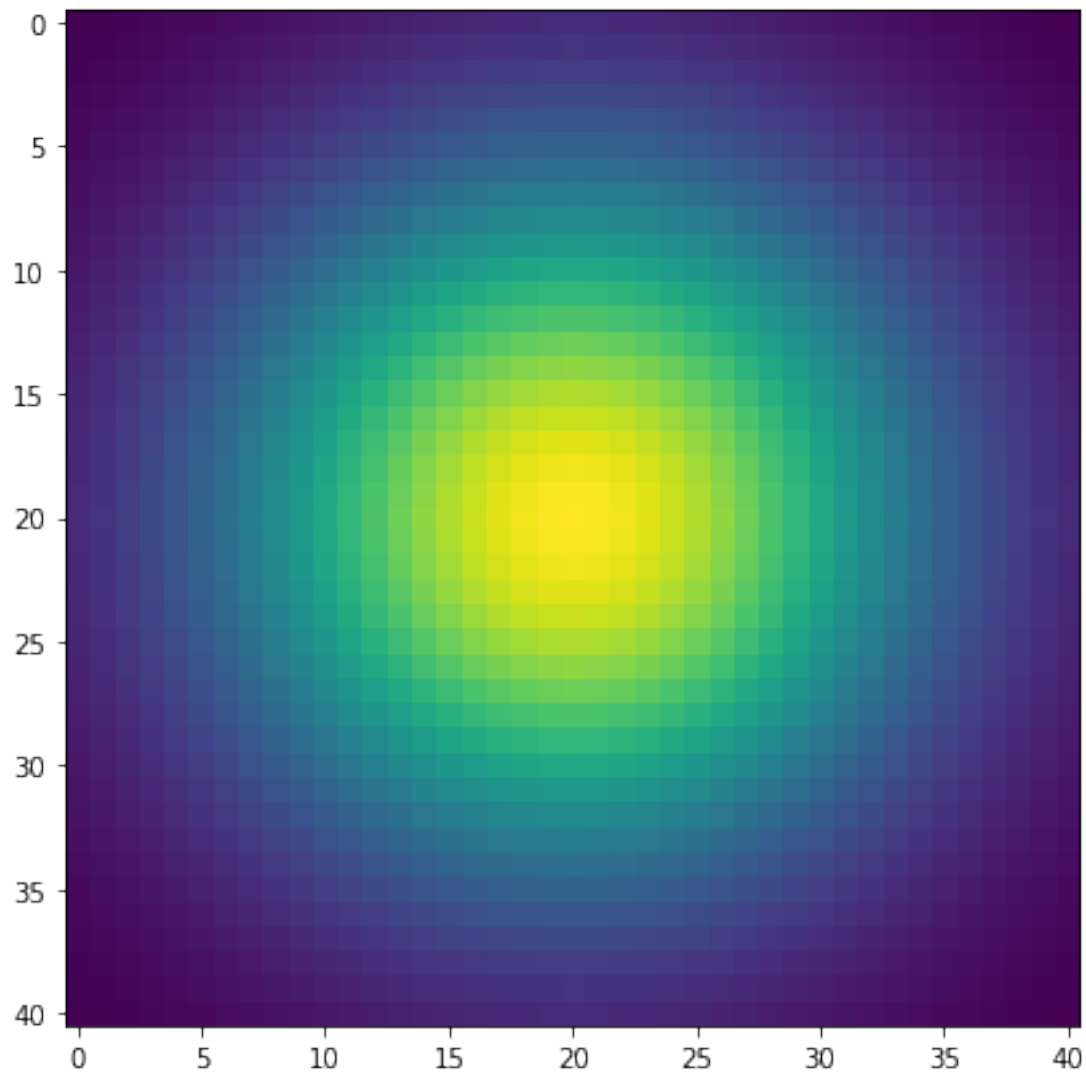
```
[ ]: plt.imshow(gaussian_filter(sigma=5))
```

```
[ ]: <matplotlib.image.AxesImage at 0x7fca40f07b80>
```



```
[ ]: plt.imshow(gaussian_filter(sigma=10))
```

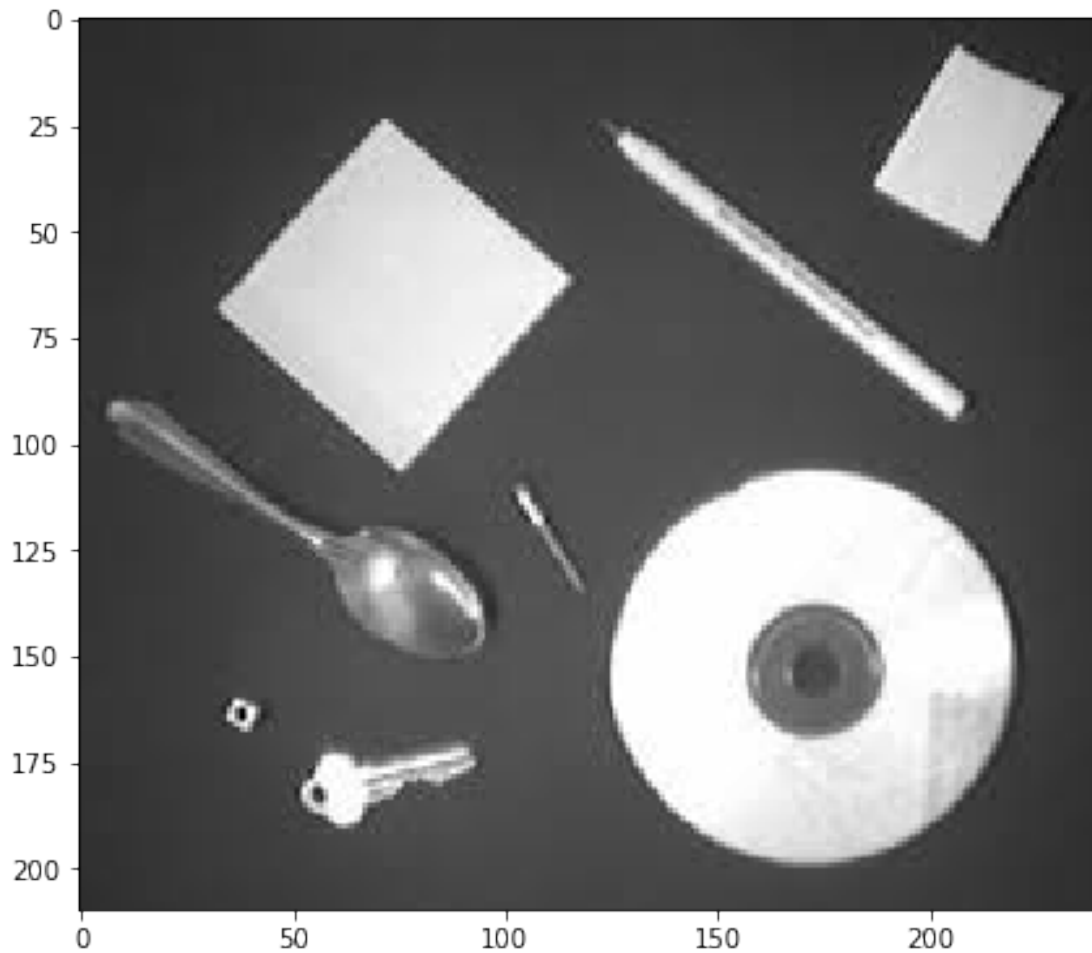
```
[ ]: <matplotlib.image.AxesImage at 0x7fca14204340>
```



3 Problem 2: Edge Detection

There are a variety of filters that we can use for different tasks. One such task is edge detection, which is useful for finding the boundaries regions in an image. In this part, your task is to use convolutions to find edges in images. Let's first load up an edgy image.

```
[ ]: im = load_image('edge_detection_image.jpg')
      im = im.mean(axis=2) # convert to grayscale
      show_image(im)
```

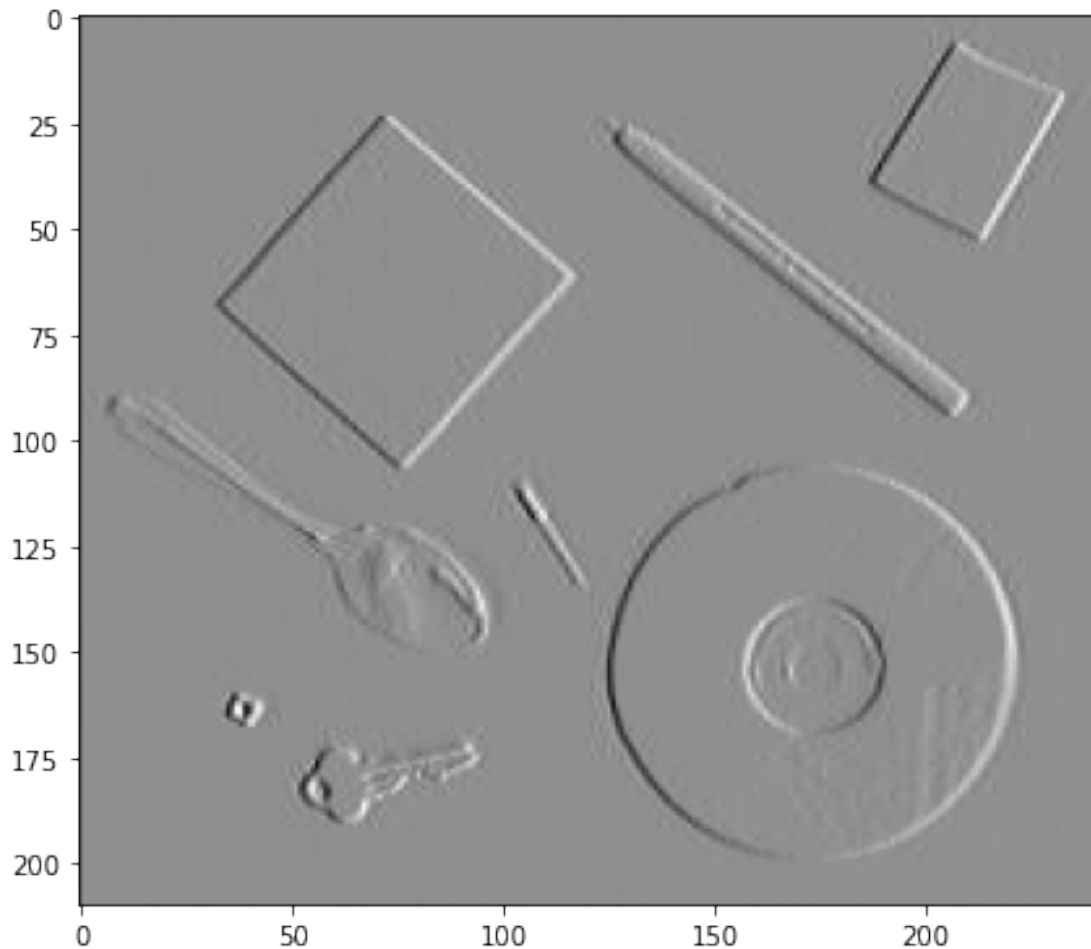


3.1 Delta (Gradient) Filters

The simplest edge detector is a Delta (Gradient) filter. Implement a Delta (Gradient) filter below, and convolve it with the image. Show the result.

```
[ ]: ''' TODO: Implement a delta filter here. '''  
delta_filt = np.array([[-1, 1]])  
  
plt.imshow(convolve2d(im, delta_filt), cmap='gray')
```

```
[ ]: <matplotlib.image.AxesImage at 0x7fca30a1bf10>
```



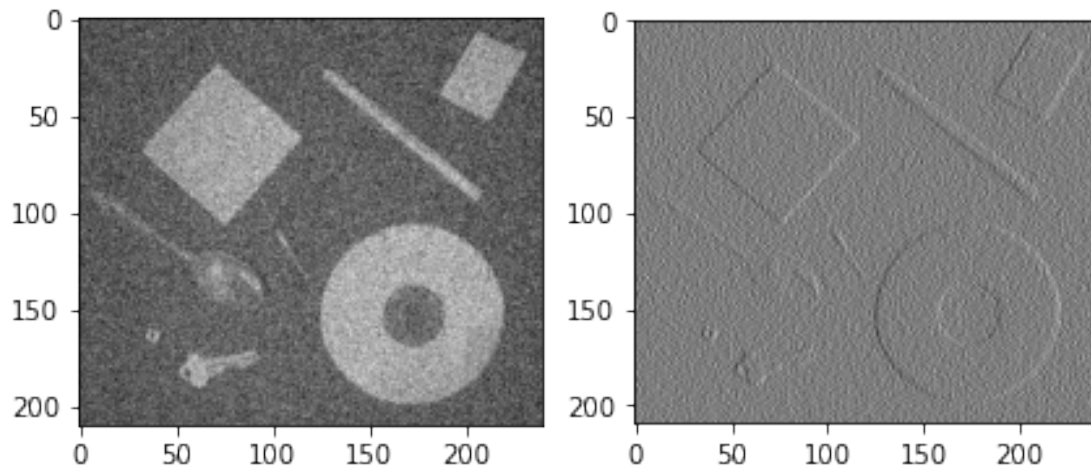
3.2 Noise

The issue with the delta filter is that it is sensitive to noise in the image. Let's add some Gaussian noise to the image below, and visualize what happens. The edges should be hard to see.

```
[ ]: im = load_image('edge_detection_image.jpg')
im = im.mean(axis=2)
im = im + 0.2*np.random.randn(*im.shape)

f, axarr = plt.subplots(1,2)
axarr[0].imshow(im, cmap='gray')
axarr[1].imshow(convolve2d(im, delta_filt), cmap='gray')
```

```
[ ]: <matplotlib.image.AxesImage at 0x7fca141161f0>
```



3.3 Laplacian Filters

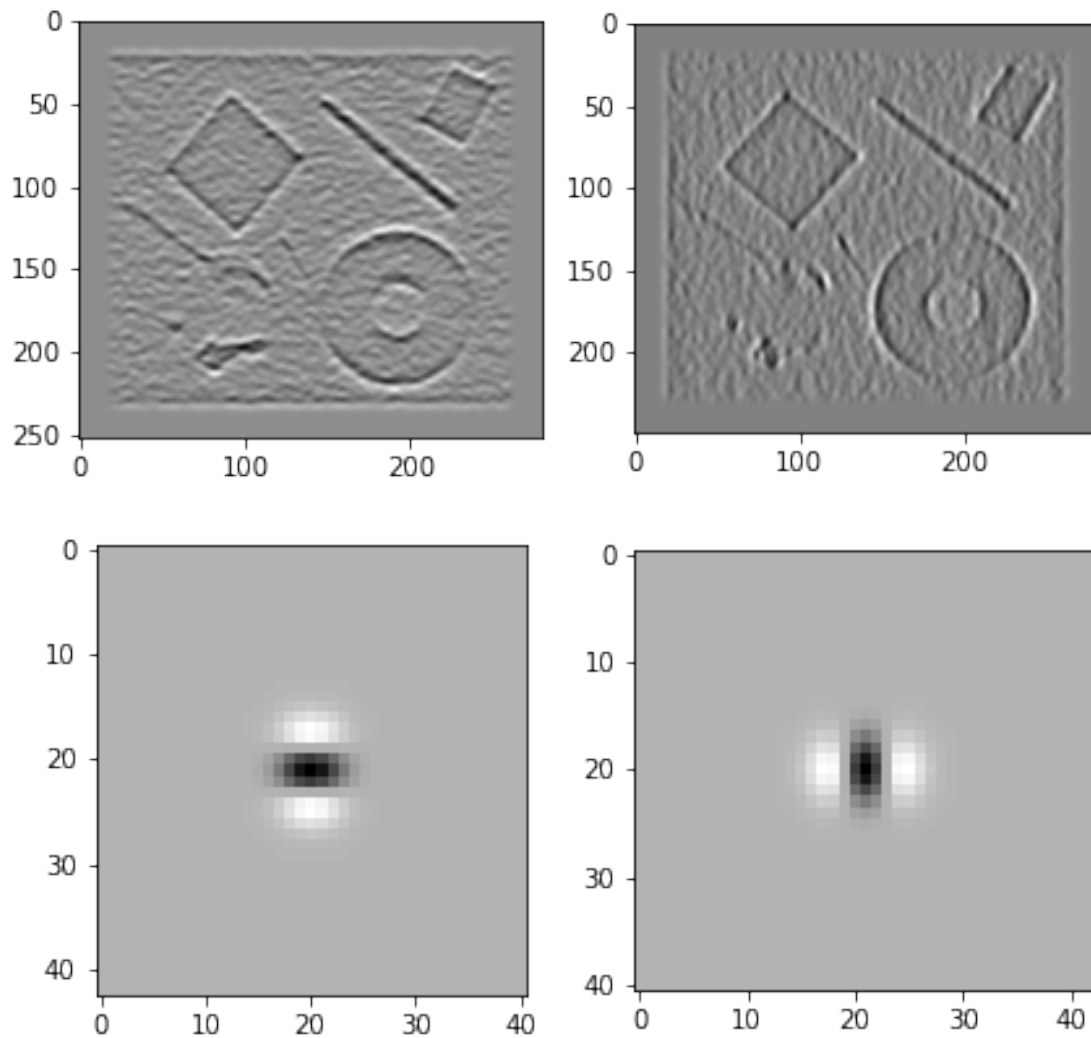
Laplacian filters are edge detectors that are robust to noise (Why is this? Think about how the filter is constructed.). Implement a Laplacian filter below for both horizontal and vertical edges.

```
[ ]: delta_filt = np.array([[ -1, 1]])

# As for Laplacian we need double derivatives, we have to convolve with the
# delta filter twice
# which is equivalent to taking derivative twice.
''' TODO: Implement a Laplacian filter for horizontal edges. '''
lap_x_filt = convolve2d(convolve2d(gaussian_filter(2), delta_filt), delta_filt)
''' TODO: Implement a Laplacian filter for vertical edges. '''
lap_y_filt = convolve2d(convolve2d(gaussian_filter(2), np.
    transpose(delta_filt)), np.transpose(delta_filt))

f, axarr = plt.subplots(2,2)
axarr[0,0].imshow(convolve2d(im, lap_y_filt), cmap='gray')
axarr[0,1].imshow(convolve2d(im, lap_x_filt), cmap='gray')
axarr[1,0].imshow(lap_y_filt, cmap='gray')
axarr[1,1].imshow(lap_x_filt, cmap='gray')
```

```
[ ]: <matplotlib.image.AxesImage at 0x7fca4124cca0>
```

4 Problem 3: Hybrid Images

Hybrid images is a technique to combine two images in one. Depending on the distance you view the image, you will see a different image. This is done by merging the high-frequency components of one image with the low-frequency components of a second image. In this problem, you are going to use the Fourier transform to make these images. But first, let's visualize the two images we will merge together.

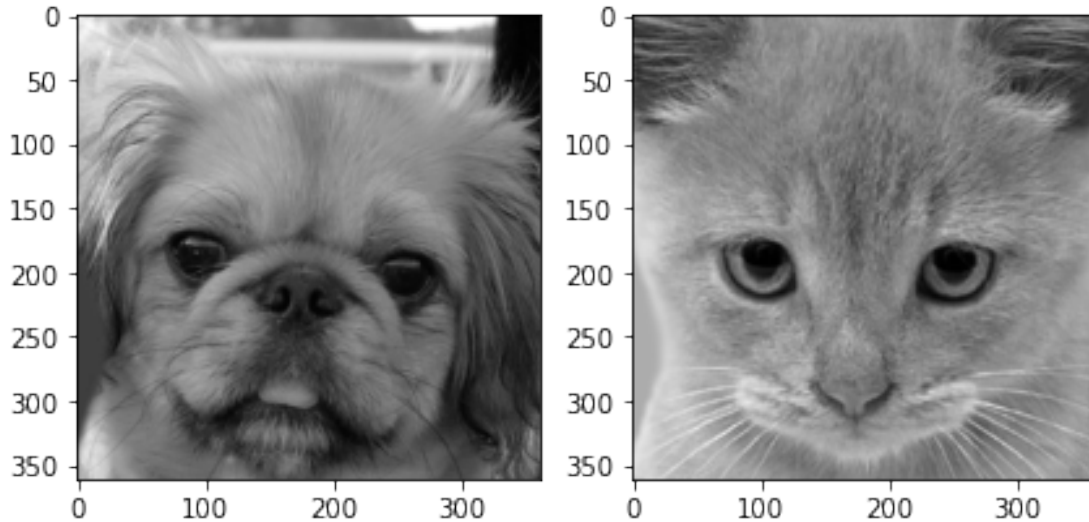
```
[ ]: from numpy.fft import fft2, fftshift, ifftshift, ifft2

dog = load_image('dog.jpg').mean(axis=-1)[: , 25:-24]
cat = load_image('cat.jpg').mean(axis=-1)[: , 25:-24]

f, axarr = plt.subplots(1,2)
```

```
axarr[0].imshow(dog, cmap='gray')
axarr[1].imshow(cat, cmap='gray')
```

```
[ ]: <matplotlib.image.AxesImage at 0x7fca1411bbe0>
```



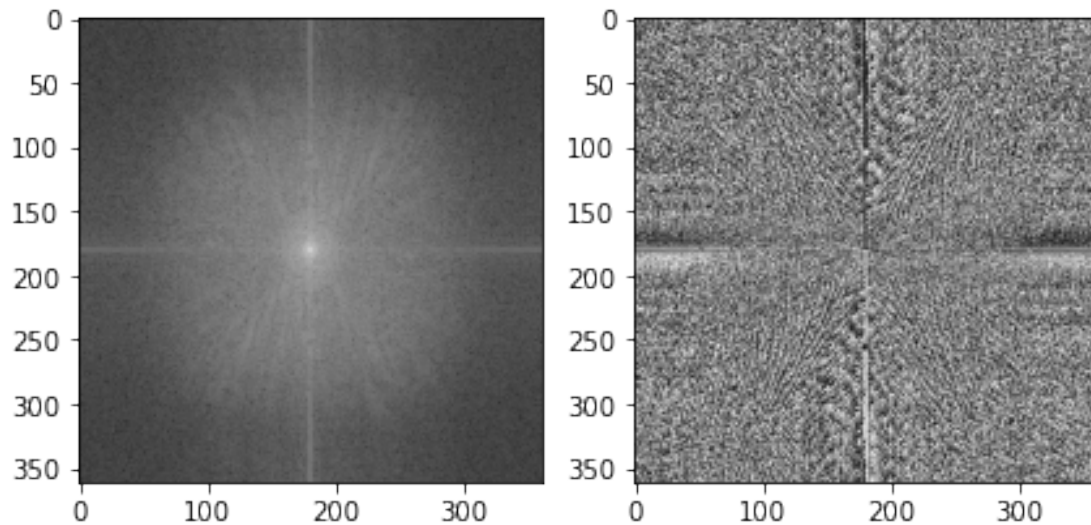
4.1 Fourier Transform

In the code box below, compute the Fourier transform of the two images. You can use the `fft2` function. You can also use the `fftshift` function, which may help in the next section.

```
[ ]: cat_fft = fftshift(fft2(cat))
dog_fft = fftshift(fft2(dog))

# Visualize the magnitude and phase of cat_fft. This is a complex number, so we
↳ visualize
# the magnitude and angle of the complex number.
# Curious fact: most of the information for natural images is stored in the
↳ phase (angle).
f, axarr = plt.subplots(1,2)
axarr[0].imshow(np.log(np.abs(cat_fft)), cmap='gray')
axarr[1].imshow(np.angle(cat_fft), cmap='gray')
```

```
[ ]: <matplotlib.image.AxesImage at 0x7fca305bce50>
```



4.2 Low and High Pass Filters

By masking the Fourier transform, you can compute both low and high pass of the images. In Fourier space, write code below to create the mask for a high pass filter of the cat, and the mask for a low pass filter of the dog. Then, convert back to image space and visualize these images.

You may need to use the functions `ifft2` and `ifftshift`.

```
[ ]: def generate_masks(img_fft):
    mid_pixel_idx = img_fft.shape[0]//2
    num_values_to_mask = 10

    high_mask = np.ones(img_fft.shape)
    high_mask[mid_pixel_idx - num_values_to_mask: mid_pixel_idx +
    ↪num_values_to_mask, \
               mid_pixel_idx - num_values_to_mask: mid_pixel_idx +
    ↪num_values_to_mask] = 0
    low_mask = 1 - high_mask

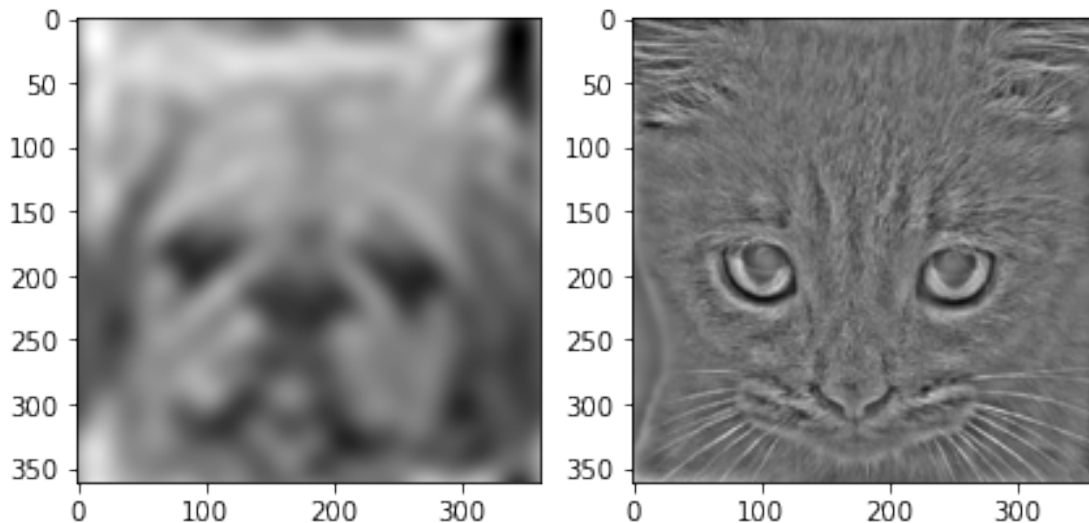
    return low_mask, high_mask

cat_low_pass_filter, cat_high_pass_filter = generate_masks(cat_fft)
dog_low_pass_filter, dog_high_pass_filter = generate_masks(dog_fft)

cat_filtered = np.multiply(cat_fft, cat_high_pass_filter)
cat_filtered = np.real(ifft2(ifftshift(cat_filtered)))
dog_filtered = np.multiply(dog_fft, dog_low_pass_filter)
dog_filtered = np.real(ifft2(ifftshift(dog_filtered)))
```

```
f, axarr = plt.subplots(1,2)
axarr[0].imshow(dog_filtered, cmap='gray')
axarr[1].imshow(cat_filtered, cmap='gray')
```

```
[ ]: <matplotlib.image.AxesImage at 0x7fca305c9af0>
```



4.3 Hybrid Image Results

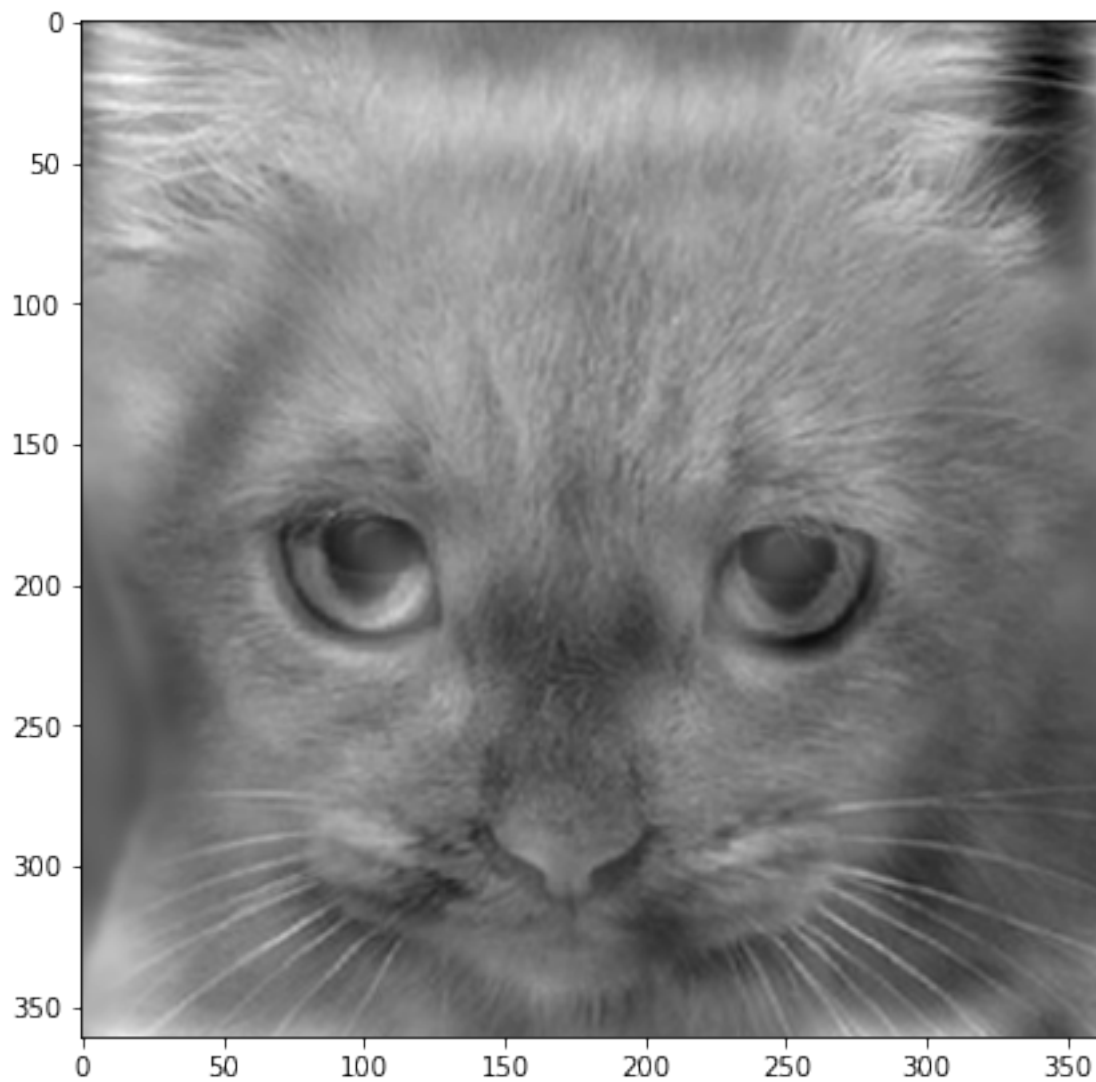
Now that we have the high pass and low pass filtered images, we can create a hybrid image by adding them. Write the code to combine the images below, and visualize the hybrid image.

Depending on whether you are close or far away from your monitor, you should see either a cat or a dog. Try creating a few different hybrid images from your own photos or photos you found. Submit them, and we will show the coolest ones in class.

```
[ ]: hybrid = np.add(cat_filtered, dog_filtered)

plt.imshow(hybrid, cmap='gray')
```

```
[ ]: <matplotlib.image.AxesImage at 0x7fca309da400>
```



4.4 Acknowledgements

This homework is based on assignments from Aude Oliva at MIT, and James Hays at Georgia Tech.