Lab Assignment 1 – The Fourier Transform

Computer Vision - Term 5, 2022

Instructor: Dr. Saumya Jetly
TA: Ribhu Lahiri

Deadline: Monday, 7 March 2022 11:59 pm

Submission form link: https://forms.gle/UowanRjZ3f8m5skr7

Total points: 5 (+1 extra credit)

Part 1: Fast Fourier transformation of images (2 points)

In the first part of the assignment, you will create a backbone for the rest of it. Here, you need to create functions for the fast fourier transform, the inverse of that, a function to shift the zero-frequency componenet to the center of the spectrum, and an inverse for that as well.

Note: If you use pre-built functions there will be a slight deduction in grades

```
# Imports
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
# Reading in a sample image
from PIL import Image

im = Image.open("Pikachu.jpg")
```

```
img = cv2.imread("Pikachu.jpg", 0)
print(img.shape)
img_2 = cv2.imread('pup.jpg', 0)
img_2.shape
```

```
(924, 820)
(545, 800)
```

```
center = (img.shape[0]/2, img.shape[1]/2)
arr = np.zeros(img.shape)
arr
```

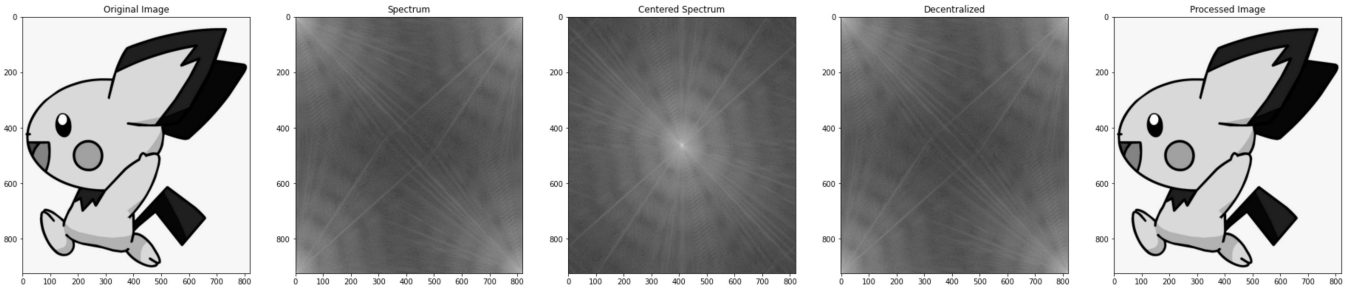
```
 array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]])
```

```
# Task 1.a:

def fft(image):
    '''
    Applies a fast fourier transform to the given image

    Parameters
    ----------
    image: PIL.Image or np.ndarray
        A PIL image object, or a numpy array representing the image

    Returns
    -------
    fft_image: np.ndarray
        A numpy array of the fft-transformed image

    '''
    # YOUR CODE HERE
    fft_image = np.fft.fft2(image)
    return fft_image


# Task 1.b:
def inv_fft(image):
    '''
    Inverts a fourier transformed image back to an image

    Parameters
    -----------
    image: np.ndarray
        An ff transformed image array

    Returns:
    ----------------------
    ifft_image: np.ndarray
        A numpy array of reconstructed image

    '''
    #TODO: YOUR CODE HERE
    image_inverse = np.fft.ifft2(image)
    return image_inverse
```

```
# Task 1.c:
def fft_shift(image):
    '''
    Shift the zero-frequency component to the center of the spectrum

    Parameters
    ----------
    image: np.ndarray
        An ff transformed image array

    Returns:
    ------------------------
    fft_shifted: np.ndarray
        A numpy array of shifted spectrum

    '''
    #TODO: YOUR CODE HERE
    image_shift = np.fft.fftshift(image)
    return image_shift

# Task 1.d
def inv_fft_shift(image):
    '''
    Inverse the shift to decentralise origin

    Parameters
    ----------
    image: np.ndarray
        A shifted ff transformed image array

    Returns:
    ------------------------
    fft_shifted: np.ndarray
        A numpy array of decentralised spectrum

    '''
    #TODO: YOUR CODE HERE
    image_ishift = np.fft.ifftshift(image)
    return image_ishift


plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False)

img = cv2.imread("Pikachu.jpg", 0) #TODO: Replace image here
img_fft = fft(img)
img_shift = fft_shift(img_fft)
img_inv_shift = inv_fft_shift(img_shift)
img_inv = inv_fft(img_inv_shift)

plt.subplot(151), plt.imshow(img, "gray"), plt.title("Original Image")
plt.subplot(152), plt.imshow(np.log(1+np.abs(img_fft)), "gray"), plt.title("Spectrum")
plt.subplot(153), plt.imshow(np.log(1+np.abs(img_shift)), "gray"), plt.title("Centered Spectr
```

```
plt.subplot(154), plt.imshow(np.log(1+np.abs(img_inv_shift)), "gray"), plt.title("Decentraliz
plt.subplot(155), plt.imshow(np.abs(img_inv), "gray"), plt.title("Processed Image")
```

```
(<AxesSubplot:title={'center':'Processed Image'}>,
 <matplotlib.image.AxesImage at 0x243a7e4bfd0>,
 Text(0.5, 1.0, 'Processed Image'))
```



Part 2: Low pass and High pass filters (2 points)

In this second part you will create two basic filters, the Gaussian low pass and high pass filters. These will give you an intuition behind how filters can be applied in the FFT domain to modify images and extract certain features that can be useful.

Equations for reference:

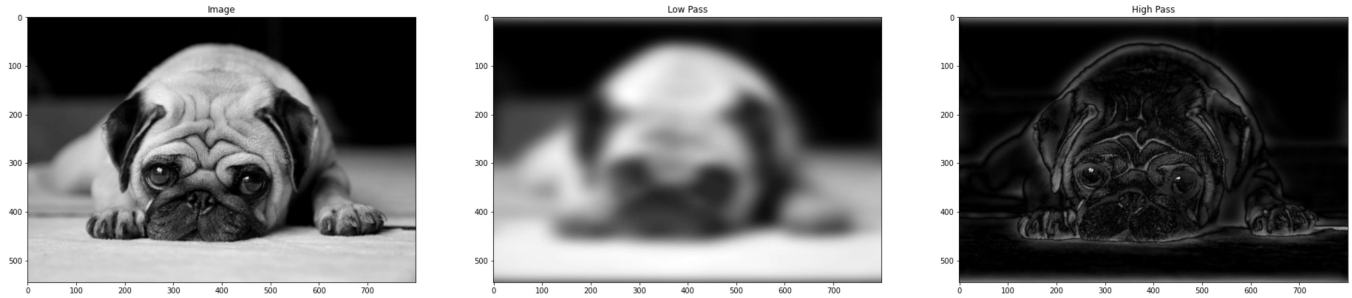
1. Low Pass filter => $H(u,v) = e^{-D^2(u,v)/2D_0}$
2. High Pass filter => $H(u,v) = 1 - e^{-D^2(u,v)/2D_0}$

```
# Feel free to create any other helper functions

def distance(p1, p2):
    return np.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

# Task 2.a:
def gaussian_low_pass(D, shape):
    '''
    Creates gaussian low pass filter for a given image

    Parameters
    ----------
    D: int
```

```
        A constant

    shape: list or tuple
        The dimensions of the image (in 2D)

    Returns:
    ----------------------
    filter: np.ndarray
        A numpy array of the low pass filter

    '''
    LPF = np.zeros(shape)
    center = (shape[0]/2, shape[1]/2)
    for i in range(shape[0]):
        for j in range(shape[1]):
            point = (i,j)
            Di = distance(point, center)
            LPF[i][j] = np.exp(-(Di**2)/(2*D))
    return LPF



# Task 2.b:
def gaussian_high_pass(D, shape):
    '''Creates gaussian high pass filter for a given image

    Parameters
    ----------
    D: int
      A constant

    shape: list or tuple
        The dimensions of the image (in 2D)

    Returns:
    ----------------------
    filter: np.ndarray
        A numpy array of the high pass filter

    '''
    LPF = gaussian_low_pass(D, shape)
    HPF = 1 - LPF
    return HPF


img = cv2.imread('pup.jpg', 0) #TODO: Replace image here
original = fft(img)
shifted = fft_shift(original)

plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False)

plt.subplot(131), plt.imshow(img, "gray"), plt.title("Image")
```

```
LPcenter = shifted * gaussian_low_pass(50, img.shape)
low_pass = inv_fft_shift(LPcenter)
inverse_low_pass = inv_fft(low_pass)
plt.subplot(132), plt.imshow(np.abs(inverse_low_pass), "gray"), plt.title("Low Pass")

HPcenter = shifted * gaussian_high_pass(50, img.shape)
high_pass = inv_fft_shift(HPcenter)
inverse_high_pass = inv_fft(high_pass)
plt.subplot(133), plt.imshow(np.abs(inverse_high_pass), "gray"), plt.title("High Pass")
```

```
(<AxesSubplot:title={'center':'High Pass'}>,
 <matplotlib.image.AxesImage at 0x243a8081670>,
 Text(0.5, 1.0, 'High Pass'))
```




```
print(shifted)
```

```
[[ 190.88676615-186.71943801j  -63.6420343 +483.56888597j
  -290.14167193-267.5974098j  ... -727.05012102+477.69441978j
   482.40975424-752.33004318j  112.58761194+489.50293884j]
 [-234.57581549+243.74007148j  107.80112613-548.58076452j
  -298.17673232+148.05878122j ...  699.25386643-415.58287314j
  -372.38430142+187.42947586j  135.71333875  -5.49584948j]
 [  40.2038553 -267.58919717j -268.2892852 +479.36438675j
   367.30752434-477.23016763j ... -550.84430584+229.69928259j
   361.27289608 -73.73016767j -315.23852254 +43.46340934j]
 ...
 [  40.2038553 +267.58919717j -315.23852254 -43.46340934j
   361.27289608 +73.73016767j ... -236.53281401-160.80594724j
   367.30752434+477.23016763j -268.2892852 -479.36438675j]
 [-234.57581549-243.74007148j  135.71333875  +5.49584948j
  -372.38430142-187.42947586j ... -178.8001045  +55.37942775j
  -298.17673232-148.05878122j  107.80112613+548.58076452j]
 [ 190.88676615+186.71943801j  112.58761194-489.50293884j
```

```
        482.40975424+752.33004318j ...  531.06000427-231.09656669j
       -290.14167193+267.5974098j   -63.6420343 -483.56888597j]]
```

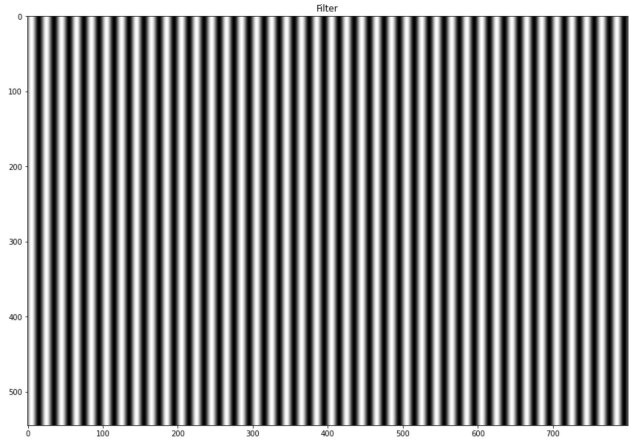
Part 3: Filters in the FFT domain (1 point)

In the third part, we apply more filters but this time, we compare them to filters applied in the Image's original spatial domain itself. This should help you to understand the connection between information in the spatial domain and in the frequency domain, and how FFT helps to work between the two.

The task is to play around with the given filters and come up with 2 novel filters, 1 in each domain.

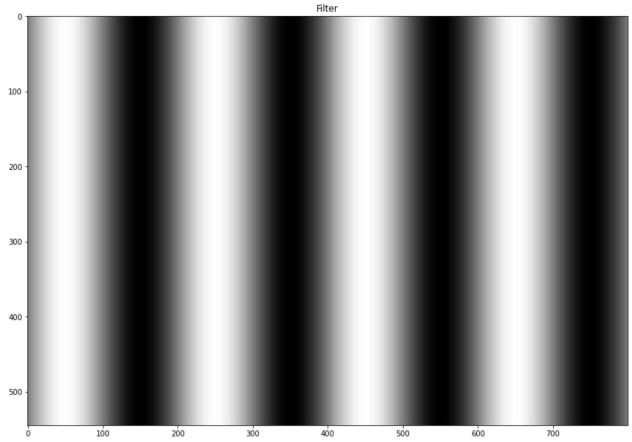
```
def mask_filter(mask, image):
    '''
    Filters the image using the mask and returns it

    Parameters
    -----------
    mask: np.ndarray
      The mask

    image: np.ndarray
        The image

    Returns:
    ------------------------
    filtered_image: np.ndarray
        The image with the filter applied

    '''
    return mask * image


x = np.arange(shifted.shape[1])  # generate 1-D sine wave of required period
y = np.sin(2 * np.pi * x / 20)

y += max(y)

mask = np.array([[y[j]*127 for j in range(shifted.shape[1])] for i in range(shifted.shape[0])

masked = mask_filter(mask, shifted)
inv_mask = inv_fft_shift(masked)
invert = inv_fft(inv_mask)

plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False)

plt.subplot(121), plt.imshow(mask, "gray"), plt.title("Filter")
plt.subplot(122), plt.imshow(np.abs(invert), "gray"), plt.title("Transformed Image")
```

```
(<AxesSubplot:title={'center':'Transformed Image'}>,
 <matplotlib.image.AxesImage at 0x243b9002400>,
 Text(0.5, 1.0, 'Transformed Image'))
```



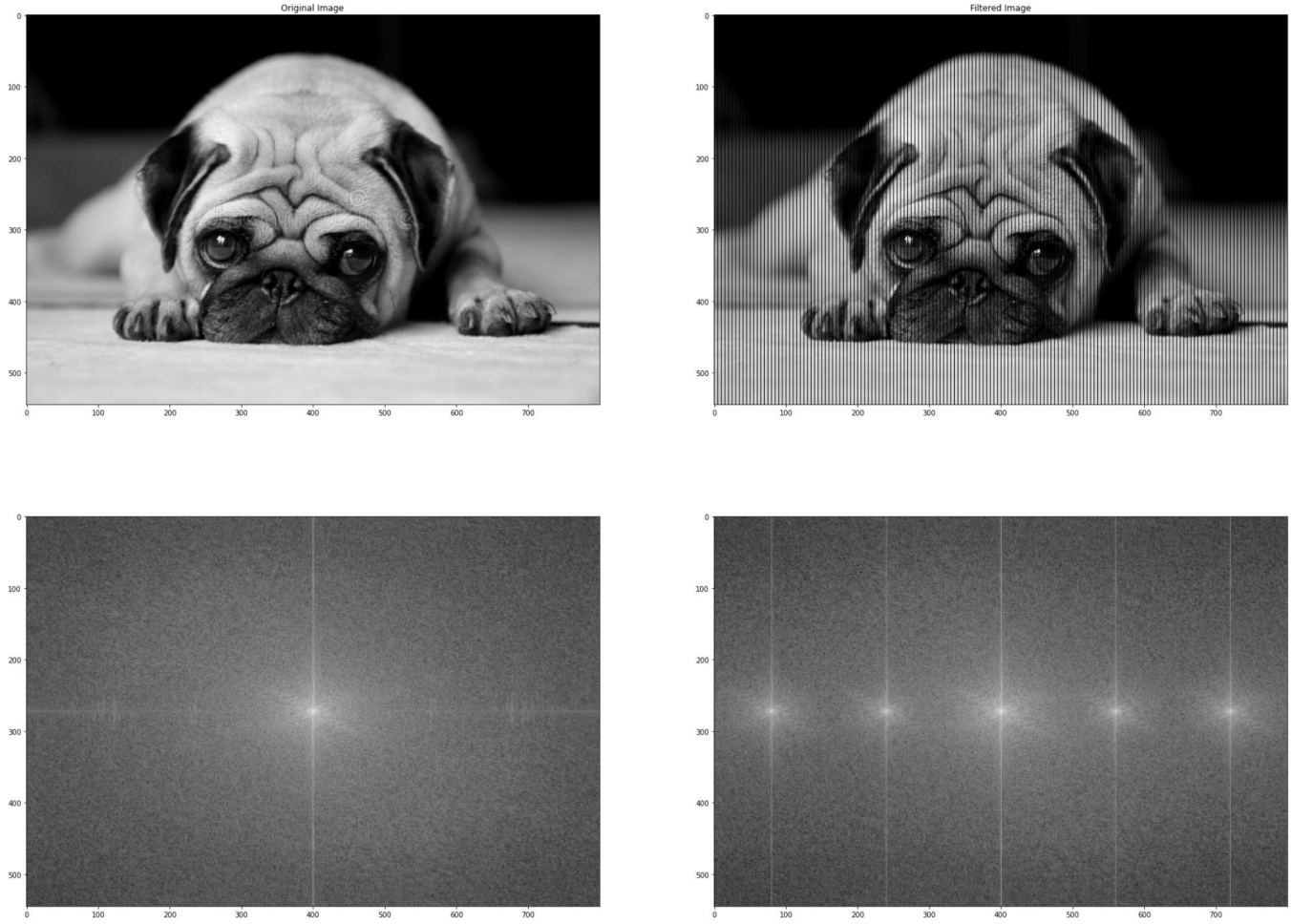

```
x = np.arange(shifted.shape[1])  # generate 1-D sine wave of required period
y = np.sin(np.pi * x / 100)

y += max(y)

mask = np.array([[y[j]*127 for j in range(shifted.shape[1])] for i in range(shifted.shape[0])

masked = mask_filter(mask, shifted)
inv_mask = inv_fft_shift(masked)
invert = inv_fft(inv_mask)
plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False)

plt.subplot(121), plt.imshow(mask, "gray"), plt.title("Filter")
plt.subplot(122), plt.imshow(np.abs(invert), "gray"), plt.title("Transformed Image")
```

```
(<AxesSubplot:title={'center':'Transformed Image'}>,
 <matplotlib.image.AxesImage at 0x243c1fb7040>,
 Text(0.5, 1.0, 'Transformed Image'))
```



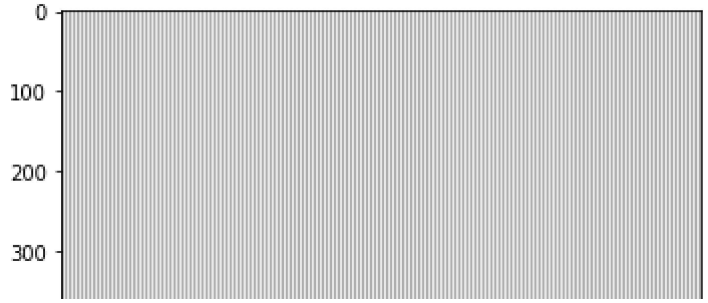

```
mask = np.ones(img.shape)

for i in range(img.shape[0]):
  for j in range(0,img.shape[1],5):
    mask[i][j] = 0

filter_img = mask_filter(mask, img)
fft_filtered_img = fft(filter_img)
fft_filtered = fft_shift(fft_filtered_img)

fft_img = fft(img)
fft_center = fft_shift(fft_img)

plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False)

plt.subplot(221), plt.imshow(np.abs(img), 'gray'), plt.title('Original Image')
plt.subplot(222), plt.imshow(np.abs(filter_img), 'gray'), plt.title('Filtered Image')
plt.subplot(223), plt.imshow(np.log(1+np.abs(fft_center)), 'gray')
plt.subplot(224), plt.imshow(np.log(1+np.abs(fft_filtered)), 'gray')
```

```
(<AxesSubplot:>, <matplotlib.image.AxesImage at 0x243ba3b5520>)
```



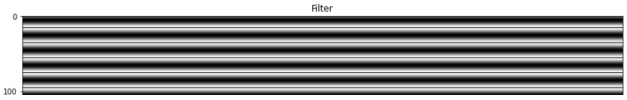

```
plt.imshow(mask, 'gray')
```

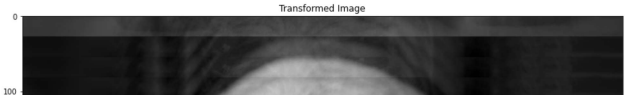
```
<matplotlib.image.AxesImage at 0x243b93e3eb0>
```




```
# TODO: YOUR CODE HERE
###Frequency domain filter
y = np.arange(shifted.shape[1])  # generate 1-D sine wave of required period
x = np.sin(2*np.pi * y / 20)

x += max(x)

mask = np.array([[x[i]*(-127) for j in range(shifted.shape[1])] for i in range(shifted.shape[

masked = mask_filter(mask, shifted)
inv_mask = inv_fft_shift(masked)
invert = inv_fft(inv_mask)

plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False)

plt.subplot(121), plt.imshow(mask, "gray"), plt.title("Filter")
plt.subplot(122), plt.imshow(np.abs(invert), "gray"), plt.title("Transformed Image")
```

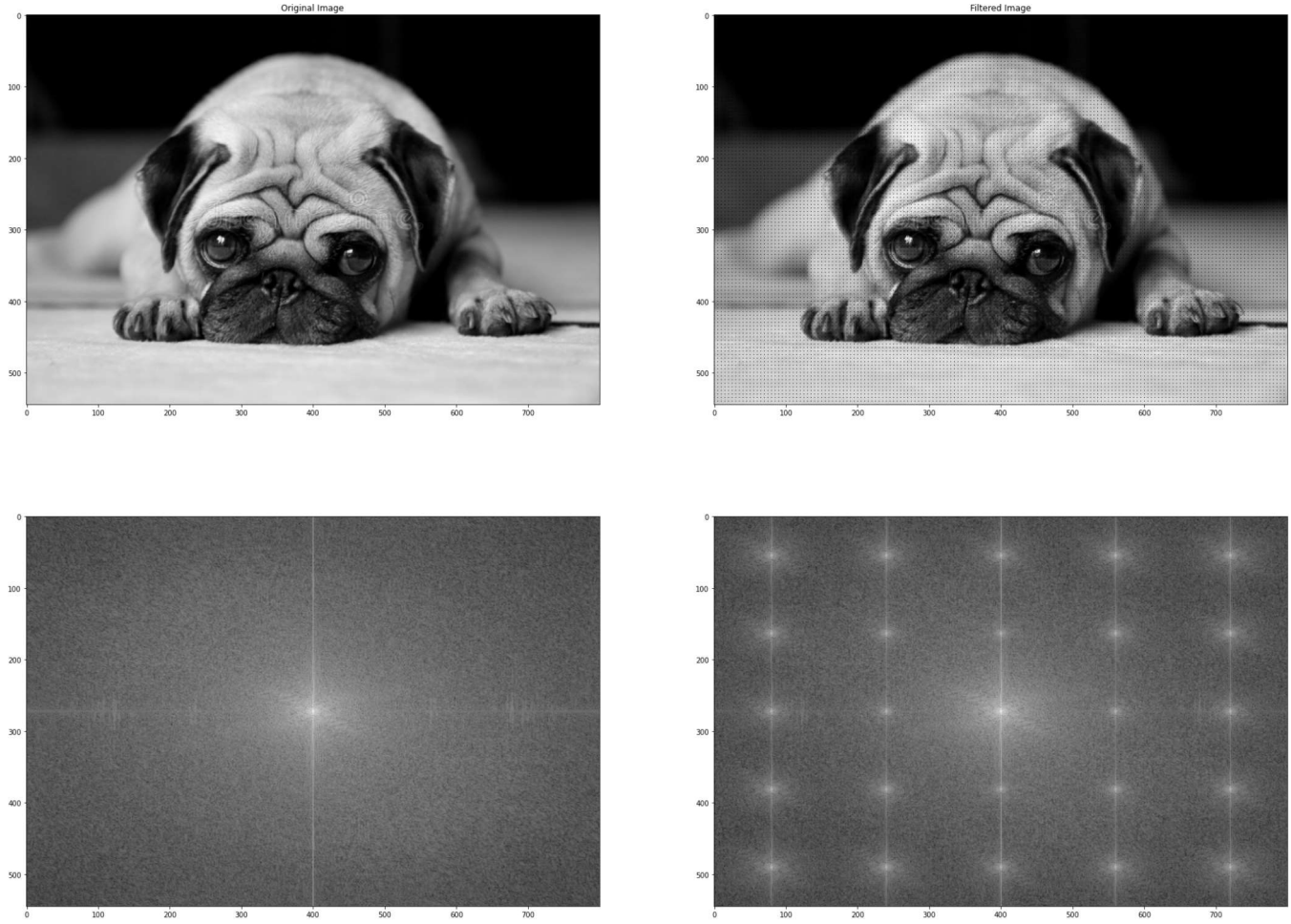
```
(<AxesSubplot:title={'center':'Transformed Image'}>,
 <matplotlib.image.AxesImage at 0x243c2312f10>,
 Text(0.5, 1.0, 'Transformed Image'))
```





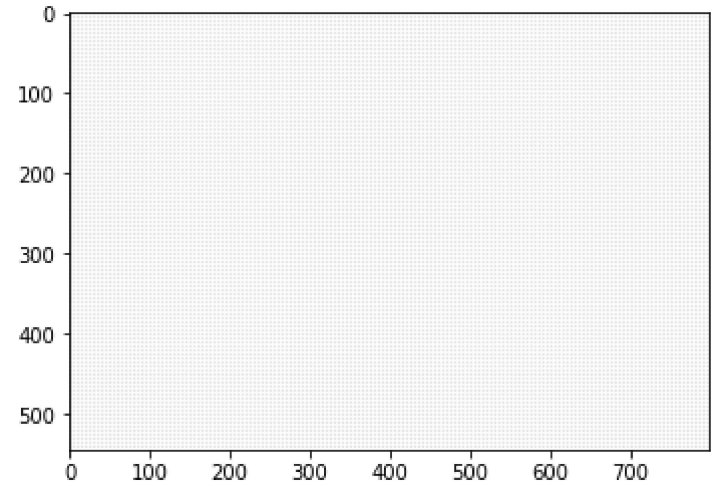

```
#spatial domain filter
mask = np.ones(img.shape)

for i in range(0,img.shape[0],5):
  for j in range(0,img.shape[1],5):
    mask[i][j] = 0

filter_img = mask_filter(mask, img)
fft_filtered_img = fft(filter_img)
fft_filtered = fft_shift(fft_filtered_img)

fft_img = fft(img)
fft_center = fft_shift(fft_img)

plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False)

plt.subplot(221), plt.imshow(np.abs(img), 'gray'), plt.title('Original Image')
plt.subplot(222), plt.imshow(np.abs(filter_img), 'gray'), plt.title('Filtered Image')
plt.subplot(223), plt.imshow(np.log(1+np.abs(fft_center)), 'gray')
plt.subplot(224), plt.imshow(np.log(1+np.abs(fft_filtered)), 'gray')
```

```
(<AxesSubplot:>, <matplotlib.image.AxesImage at 0x243b9451520>)
```




```
plt.imshow(mask, 'gray')
```

```
<matplotlib.image.AxesImage at 0x243ba3702e0>
```



```
print(mask.shape)
print(y.shape)
```

```
print(x.shape)
print(mask)
```

```
(545, 800)
(800,)
(800,)
[[0. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]
 ...
 [1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]]
```

Part 4: Construction of Hybrid Images [Optional] (1 Point)

As an optional part, you can try to replicate the results from Oliva et al's Hybrid Images paper (https://stanford.edu/class/ee367/reading/OlivaTorralb_Hybrid_Siggraph06.pdf), as discussed in the class.

HINT: You already have the building blocks for it in the functions above.

```
# Task 3:
def create_hybrid_image(lp_image, hp_image):
    '''
    Creates a hybrid image combining low pass features from one image
    with high pass features from the other

    Parameters
    -----------
    lp_image: np.ndarray
      FF transformed image whose low pass features are to be combined

    hp_image: np.ndarray
      FF transformed image whose high pass features are to be combined

    Returns:
    ------------------------
    hybrid_image: np.ndarray
        A numpy array of the combined high pass and low pass image

    '''
    # TODO: YOUR CODE HERE
    shifted = fft_shift(lp_image)
    LPcenter = shifted * gaussian_low_pass(50, lp_image.shape)
    low_pass = inv_fft_shift(LPcenter)
    inverse_low_pass = inv_fft(low_pass)

    shifted = fft_shift(hp_image)
```

```
    HPcenter = shifted * gaussian_high_pass(50, hp_image.shape)
    high_pass = inv_fft_shift(HPcenter)
    inverse_high_pass = inv_fft(high_pass)
    hybrid_image = np.abs(inverse_low_pass) + np.abs(inverse_high_pass)
    return hybrid_image



img1 = cv2.imread('pup.jpg', 0)
img2 = cv2.imread('Pikachu.jpg', 0)

img1 = img1[0:544, 0:800]
img2 = img2[0:544, 0:800]

orig1 = np.fft.fft2(img1)
orig2 = np.fft.fft2(img2)

plt.figure(figsize=(6.4*5, 4.8*5), constrained_layout=False)

plt.subplot(131), plt.imshow(img1, "gray"), plt.title("Image 1")
plt.subplot(132), plt.imshow(img2, "gray"), plt.title("Image 2")

plt.subplot(133), plt.imshow(create_hybrid_image(orig1, orig2), "gray"), plt.title("Hybrid Im
```

```
(<AxesSubplot:title={'center':'Hybrid Image'}>,
 <matplotlib.image.AxesImage at 0x243ba3702b0>,
 Text(0.5, 1.0, 'Hybrid Image'))
```