Exercise Round 1 Daniel Kusnetsoff

Task 1

a) Convert the four points below (cartesian x,y-coordinates) into their corresponding homogeneous coordinate form.

```
x1 =[2;1]
```

```
x1 = 2×1
     2
     1
```

```
x1 =[2;1;1]
```

```
x1 = 3×1
     2
     1
     1
```

```
x2 =[1;-2]
```

```
x2 = 2×1
     1
    -2
```

```
x2 =[1;-2;1]
```

```
x2 = 3×1
     1
    -2
     1
```

```
x3 =[1;1]
```

```
x3 = 2×1
     1
     1
```

```
x3 =[1;1;1]
```

```
x3 = 3×1
     1
     1
     1
```

```
x4 =[-1;0]
```

```
x4 = 2×1
    -1
     0
```

```
x4 =[-1;0;1]
```

```
x4 = 3×1
    -1
     0
     1
```

b) The line **l** through two points **x** and **x′** is **l** = **x** × **x′**. Use this to form two lines, line **l**1 through homogeneous points **x**1 and **x**2, and **l**2 through **x**3 and **x**4.

As **l** = **x** × **x′**,

*x(transpose)*l = x(transpose)*x cross(x') = 0*

```
A=cross(x1,x2)
```

```
A = 3×1
     3
    -1
    -5
```

```
B=cross(x3,x4)
```

```
B = 3×1
     1
    -2
     1
```

c.The intersection of two lines **l** and **l′** is the point **x** = **l** × **l′**. Use lines **l**1 and **l**2 to

calculate their point of intersection and convert this back into cartesian coordinates.

```
C=cross(A,B)
```

```
C = 3×1
   -11
    -8
    -5
```

```
-11/5
```

```
ans = -2.2000
```

```
-8/5
```

```
ans = -1.6000
```

Point of intersection [-2.2;-1.6]

Untitled17

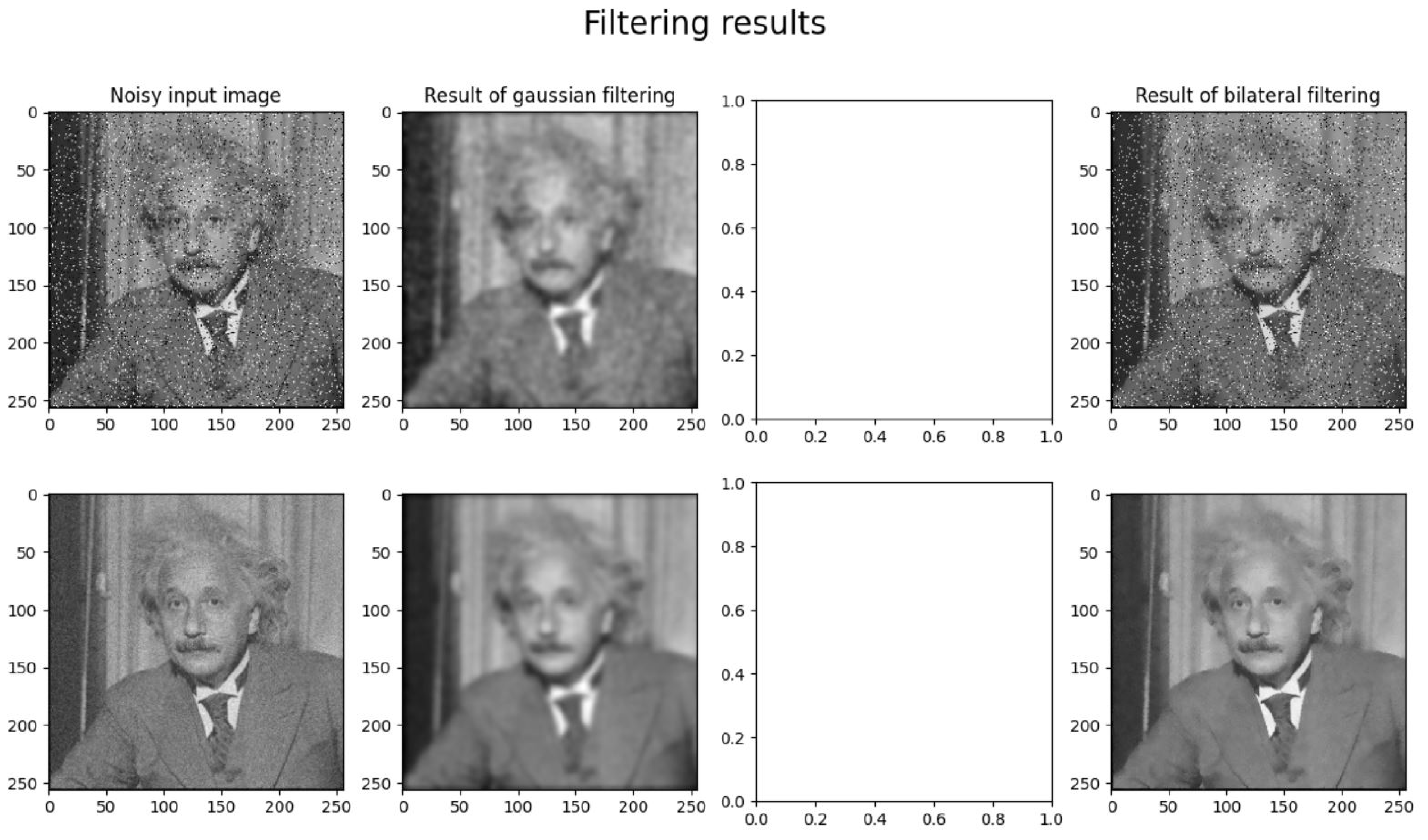
January 16, 2022

[ ]:
```
# -*- coding: utf-8 -*-
"""
Created on Wed Jan 13 08:38:35 2021

@author: tiitu
"""



import os
import sys
sys.path.append(os.getcwd())

from matplotlib.pyplot import imread
from skimage.transform import resize as imresize
#from scipy.misc import imresize  # deprecated, may work with older versions of␣
↪scipy
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage.filters import convolve as conv2
from scipy.ndimage.filters import convolve1d as conv1
from utils import imnoise, gaussian2, bilateral_filter


# Load test images and convert to double precision in the interval [0,1].
im = imread('einsteinpic.jpg') / 255.
im = imresize(im, (256, 256))

# Generate noise
imns = imnoise(im, 'salt & pepper', 0.05) * 1.            # "salt and pepper"␣
↪noise
imng = im + 0.05*np.random.randn(im.shape[0],im.shape[1])  # zero-mean Gaussian␣
↪noise


# Apply a Gaussian filter with a standard deviation of 2.5
sigmad = 2.5
g, _, _, _, _, _, = gaussian2(sigmad)
```

```python
gflt_imns = conv2(imns, g, mode='reflect')
gflt_imng = conv2(imng, g, mode='reflect')


# Instead of directly filtering with g, make a separable implementation
# where you use horizontal and vertical 1D convolutions.
# Store the results again to gflt_imns and gflt_imng, use conv1 instead.
# The result should not change.
# See Szeliski's Book chapter 3.2.1 Separable filtering, numpy.linalg.svd and␣
↪scipy.ndimage.filters convolve1d

##--your-code-starts-here--##
## 1d-gaussian

def gaussian1(sigma, N=None):
    if N is None:
        N = 2  * np.maximum(4, np.ceil(6*sigma)) + 1
    k = (N - 1) / 2.
    x = np.arange(-k, k+1)
    g = 1/(np.sqrt(2 * np.pi * sigma**2)) * np.exp(-(x**2) / (2 * sigma ** 2))
    return g

g1d=gaussian1(sigmad)

gflt_imns_x = conv1(imns, g1d, mode="reflect", axis=0)
gflt_imns_xy = conv1(gflt_imns_x, g1d, mode='reflect', axis=1)

gflt_imns_y = conv1(imns, g1d, mode="reflect", axis=1)
gflt_imns_yx = conv1(gflt_imns_y, g1d, mode='reflect', axis=0)

#gflt_imns = conv1(imns, g, mode='reflect')
#gflt_imng = conv1(imng, g, mode='reflect')
##--your-code-ends-here--##


# Median filtering is done by extracting a local patch from the input image
# and calculating its median
def median_filter(img, wsize):
    nrows, ncols = img.shape
    output = np.zeros([nrows, ncols])
    k = (wsize - 1) / 2

    for i in range(nrows):
        for j in range(ncols):
            # Calculate local region limits
            iMin = int(max(i - k, 0))
            iMax = int(min(i + k, nrows - 1))
```

```
            jMin = int(max(j - k, 0))
            jMax = int(min(j + k, ncols - 1))

            # Use the region limits to extract a patch from the image,
            # calculate the median value (e.g using numpy) from the extracted
            # local region and store it to output using correct indexing.

            ##--your-code-starts-here--##

            ##--your-code-ends-here--##

    return output

# Apply median filtering, use neighborhood size 5x5
# Store the results in medflt_imns and medflt_imng
# Use the median_filter function above

##--your-code-starts-here--##

##--your-code-ends-here--##


# Apply bilateral filter to each image with window size 11.
# See section 3.3.1 of Szeliski's book
# Use sigma value 2.5 for the domain kernel and 0.1 for range kernel.

wsize = 11
sigma_d = 2.5
sigma_r = 0.1

bflt_imns = bilateral_filter(imns, wsize, sigma_d, sigma_r)
bflt_imng = bilateral_filter(imng, wsize, sigma_d, sigma_r)

# Display filtering results
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(16,8))
ax = axes.ravel()
ax[0].imshow(imns, cmap='gray')
ax[0].set_title("Noisy input image")
ax[1].imshow(gflt_imns, cmap='gray')
ax[1].set_title("Result of gaussian filtering")
#ax[2].imshow(medflt_imns, cmap='gray')
#ax[2].set_title("Result of median filtering")
ax[3].imshow(bflt_imns, cmap='gray')
ax[3].set_title("Result of bilateral filtering")
ax[4].imshow(imng, cmap='gray')
ax[5].imshow(gflt_imng, cmap='gray')
#ax[6].imshow(medflt_imng, cmap='gray')
```

```
ax[7].imshow(bflt_imng, cmap='gray')
plt.suptitle("Filtering results", fontsize=20)
plt.show()
```


Filtering results
Noisy input image
Result of gaussian filtering
Result of bilateral filtering
0
50
100
150
200
250
0.0
0.2
0.4
0.6
0.8
1.0

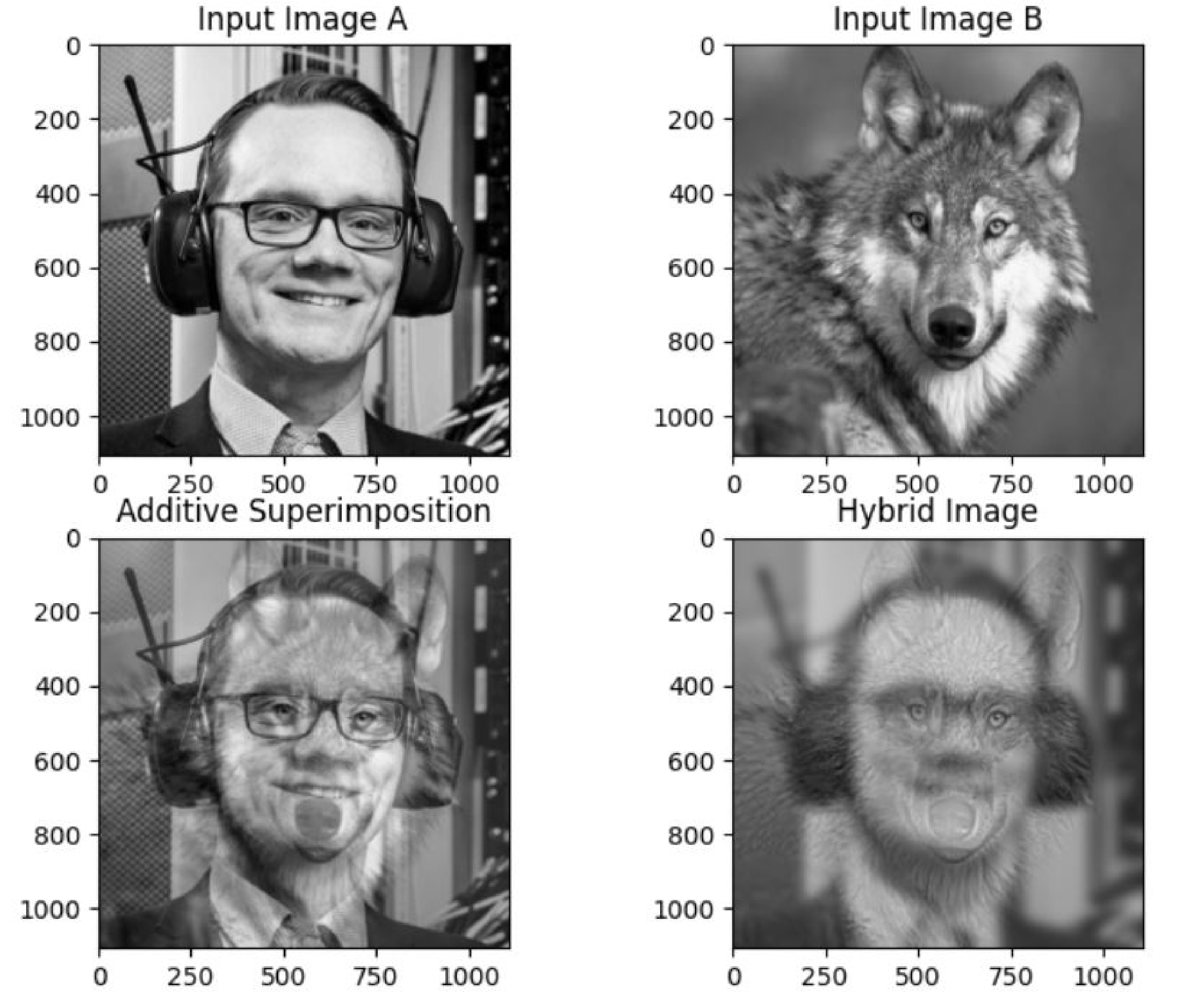
Untitled18

January 16, 2022

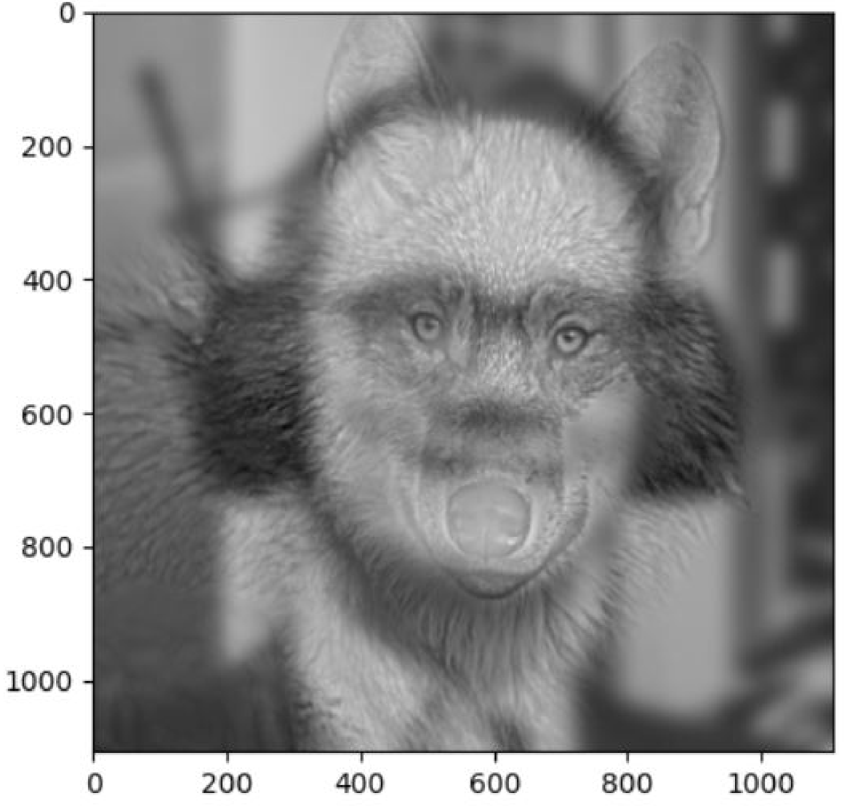
[ ]:
```
import os
import sys
sys.path.append(os.getcwd())

from matplotlib.pyplot import imread
import numpy as np
from numpy.fft import fftshift, fft2
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter, map_coordinates
from utils import affinefit


# Load test images
man = imread('man.jpg') / 255.
wolf = imread('wolf.jpg') / 255.

# The pixel coordinates of eyes and chin have been manually found
# from both images in order to perform affine alignment
man_eyes_chin = np.array([[502, 465],   # left eye
                          [714, 485],   # right eye
                          [594, 875]])  # chin
wolf_eyes_chin = np.array([[851,  919],    # left eye
                           [1159, 947],    # right eye
                           [975,  1451]])  # chin

# Warp wolf to man using an affine transformation and the coordinates above
A, b = affinefit(man_eyes_chin, wolf_eyes_chin)
xv, yv = np.meshgrid(np.arange(0, man.shape[1]), np.arange(0, man.shape[0]))
pt = np.dot(A, np.vstack([xv.flatten(), yv.flatten()])) + np.tile(b, (xv.
 ↪size,1)).T
wolft = map_coordinates(wolf, (pt[1,:].reshape(man.shape), pt[0,:].reshape(man.
 ↪shape)))

# We'll start by simply blending the aligned images using additive␣
 ↪superimposition
additive_superimposition = man + wolft
```

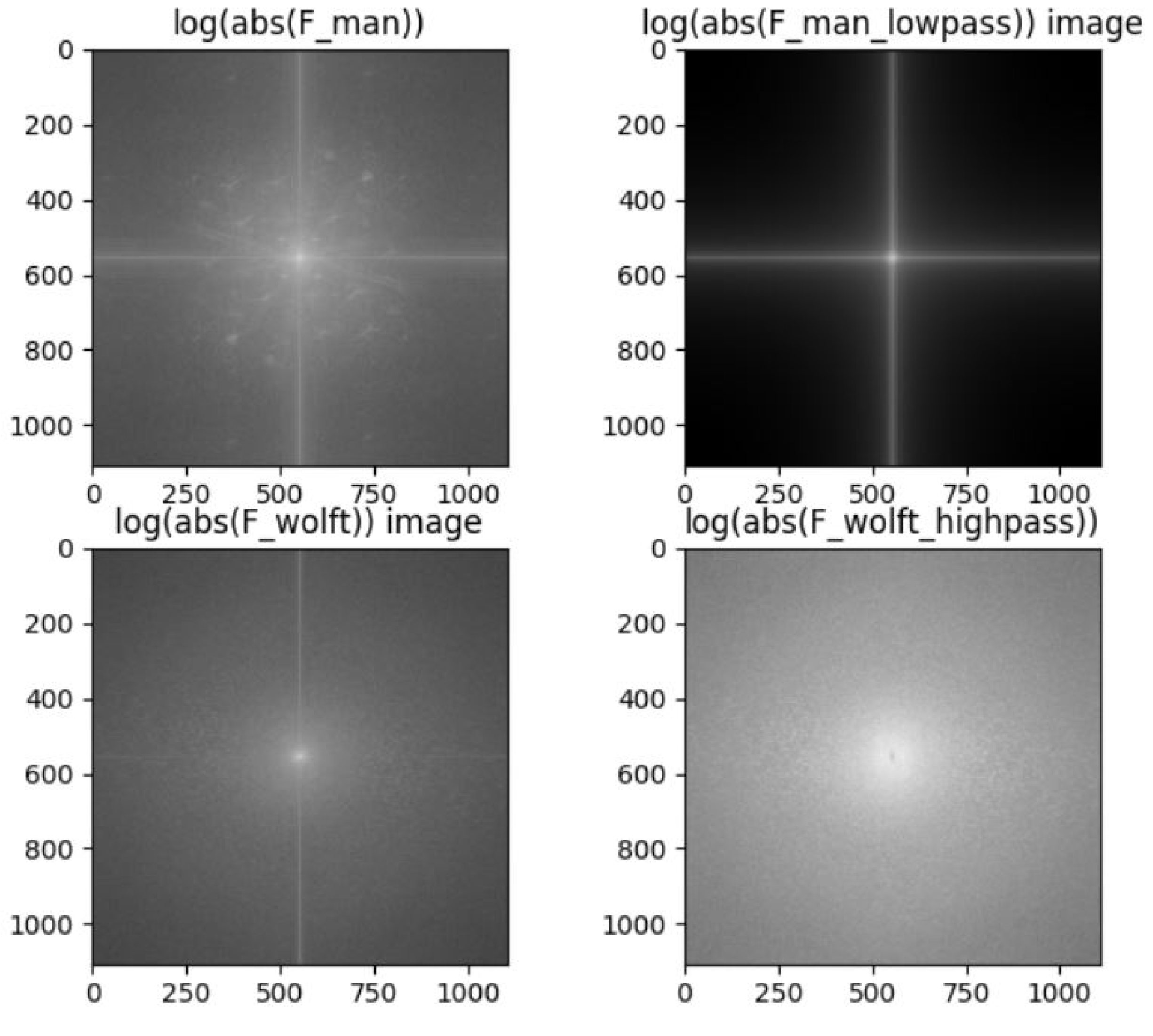
```
# Next we create two different Gaussian kernels for low-pass filtering the two␣
↪images
sigmaA = 16
sigmaB = 8
man_lowpass = gaussian_filter(man, sigmaA, mode='nearest')
wolft_lowpass = gaussian_filter(wolft, sigmaB, mode='nearest')

# Your task is to create a hybrid image by combining a low-pass filtered
# version of the human face with a high-pass filtered wolf face
# HINT: A high-passed image is equal to the low-pass filtered result removed␣
↪from the original.
# Experiment also by trying different values for 'sigmaA' and 'sigmaB' above.

# Replace the zero image below with a high-pass filtered version of 'wolft'
##--your-code-starts-here--##
#wolft_highpass = np.zeros(wolft.shape)
wolft_highpass = wolft - wolft_lowpass
#plt(wolft_highpass)

#plt.show()
##--your-code-ends-here--##

# Replace also the zero image below with the correct hybrid image using your␣
↪filtered results
##--your-code-starts-here--##
#hybrid_image = np.zeros(man_lowpass.shape)
hybrid_image = man_lowpass + wolft_highpass
##--your-code-ends-here--##


# Try looking at the results from different distances.
# Notice how strongly the interpretation of the hybrid image is affected
# by the viewing distance
plt.figure(1)
plt.imshow(hybrid_image, cmap='gray')

# Display input images and both output images.
plt.figure(2)
plt.subplot(2,2,1)
plt.imshow(man, cmap='gray')
plt.title("Input Image A")
plt.subplot(2,2,2)
plt.imshow(wolft, cmap='gray')
plt.title("Input Image B")
plt.subplot(2,2,3)
plt.imshow(additive_superimposition, cmap='gray')
plt.title("Additive Superimposition")
```

```python
plt.subplot(2,2,4)
plt.imshow(hybrid_image, cmap='gray')
plt.title("Hybrid Image")


# Visualize the log magnitudes of the Fourier transforms of the original images.
# Your task is to calculate 2D fourier transform for wolf/man and their␣
↪filtered results using fft2 and fftshift
##--your-code-starts-here--##
#F_man = np.zeros(man.shape)
#F_man_lowpass = np.zeros(man_lowpass.shape)
#F_wolft = np.zeros(wolft.shape)
#F_wolft_highpass = np.zeros(wolft_highpass.shape)
from numpy import fft
## magnitudes
def shift_tf(image):
    return fft.fftshift(fft.fft2(image))

F_man = shift_tf(man)
F_man_lowpass = shift_tf(man_lowpass)
F_wolft = shift_tf(wolft)
F_wolft_highpass = shift_tf(wolft_highpass)
##--your-code-ends-here--##


# Display the Fourier transform results
plt.figure(3)
plt.subplot(2,2,1)
plt.imshow(np.log(np.abs(F_man)), cmap='gray')
plt.title("log(abs(F_man))")
plt.subplot(2,2,2)
plt.imshow(np.log(np.abs(F_man_lowpass)), cmap='gray')
plt.title("log(abs(F_man_lowpass)) image")
plt.subplot(2,2,3)
plt.imshow(np.log(np.abs(F_wolft)), cmap='gray')
plt.title("log(abs(F_wolft)) image")
plt.subplot(2,2,4)
plt.imshow(np.log(np.abs(F_wolft_highpass)), cmap='gray')
plt.title("log(abs(F_wolft_highpass))")

plt.show()
```



Input Image A
Input Image B
Additive Superimposition
Hybrid Image
0
200
400
600
800
1000
0
250
500
750
1000


0
200
400
600
800
1000
0
200
400
600
800
1000



log(abs(F_man))
log(abs(F_man_lowpass)) image
log(abs(F_wolft)) image
log(abs(F_wolft_highpass))
0
200
400
600
800
1000
0
250
500
750
1000