Report

Part 1: Gaussian Filtering

1)

Code:

```
 9 # Question 1
10 def boxfilter(n):
11
12     # Checks that n is odd
13     assert n % 2 == 1, "Dimension must be odd"
14
15     # Ensures filter sums to 1
16     return np.ones((n, n))/(n*n)
17
```

Output:

```
In [3]: boxfilter(3)
Out[3]:
array([[ 0.11111111,  0.11111111,  0.11111111],
       [ 0.11111111,  0.11111111,  0.11111111],
       [ 0.11111111,  0.11111111,  0.11111111]])

In [4]: boxfilter(4)

AssertionErrorTraceback (most recent call last)
<ipython-input-4-65de769d965e> in <module>()
----> 1 boxfilter(4)

/Users/Terence/Desktop/Assignment 1.py in boxfilter(n)
     11
     12     # Checks that n is odd
---> 13     assert n % 2 == 1, "Dimension must be odd"
     14
     15     # Ensures filter sums to 1
AssertionError: Dimension must be odd

In [5]: boxfilter(5)
Out[5]:
array([[ 0.04,  0.04,  0.04,  0.04,  0.04],
       [ 0.04,  0.04,  0.04,  0.04,  0.04],
       [ 0.04,  0.04,  0.04,  0.04,  0.04],
       [ 0.04,  0.04,  0.04,  0.04,  0.04],
       [ 0.04,  0.04,  0.04,  0.04,  0.04]])
```

2)

Code:

```
def gauss1d(sigma):

    # Calulates the length and ensures it is a double
    filter_length = (math.ceil(6 * float(sigma)))

    # Add one if even to ensure it is odd
    if filter_length % 2 == 0:
        filter_length += 1

    # Calculate mid point and edges,sorted into array
    k = math.floor(filter_length/2)
    d = np.arange((-1 * k), k + 1)

    #apply gaussian function
    gauss_fun = np.exp( -(d ** 2)/(2 * sigma ** 2))

    # Normalize values and return results
    return (gauss_fun / np.sum(gauss_fun))
```

Output:

```
In [2]: gauss1d(0.3)
Out[2]: array([ 0.00383626,  0.99232748,  0.00383626])

In [3]: gauss1d(0.5)
Out[3]: array([ 0.10650698,  0.78698604,  0.10650698])

In [4]: gauss1d(1)
Out[4]:
array([ 0.00443305,  0.05400558,  0.24203623,  0.39905028,  0.24203623,
        0.05400558,  0.00443305])

In [5]: gauss1d(2)
Out[5]:
array([ 0.0022182 ,  0.00877313,  0.02702316,  0.06482519,  0.12110939,
        0.17621312,  0.19967563,  0.17621312,  0.12110939,  0.06482519,
        0.02702316,  0.00877313,  0.0022182 ])
```

3)

Code:

```
# Question 3
def gauss2d(sigma):
    two_dim = gauss1d(sigma)[np.newaxis]
    two_dim_transpose = np.transpose(two_dim)
    return signal.convolve2d(two_dim, two_dim_transpose)
```

Output:

```
In [6]: gauss2d(0.5)
Out[6]:
array([[ 0.01134374,  0.08381951,  0.01134374],
       [ 0.08381951,  0.61934703,  0.08381951],
       [ 0.01134374,  0.08381951,  0.01134374]])

In [7]: gauss2d(1)
Out[7]:
array([[  1.96519161e-05,   2.39409349e-04,   1.07295826e-03,
          1.76900911e-03,   1.07295826e-03,   2.39409349e-04,
          1.96519161e-05],
       [  2.39409349e-04,   2.91660295e-03,   1.30713076e-02,
          2.15509428e-02,   1.30713076e-02,   2.91660295e-03,
          2.39409349e-04],
       [  1.07295826e-03,   1.30713076e-02,   5.85815363e-02,
          9.65846250e-02,   5.85815363e-02,   1.30713076e-02,
          1.07295826e-03],
       [  1.76900911e-03,   2.15509428e-02,   9.65846250e-02,
          1.59241126e-01,   9.65846250e-02,   2.15509428e-02,
          1.76900911e-03],
       [  1.07295826e-03,   1.30713076e-02,   5.85815363e-02,
          9.65846250e-02,   5.85815363e-02,   1.30713076e-02,
          1.07295826e-03],
       [  2.39409349e-04,   2.91660295e-03,   1.30713076e-02,
          2.15509428e-02,   1.30713076e-02,   2.91660295e-03,
          2.39409349e-04],
       [  1.96519161e-05,   2.39409349e-04,   1.07295826e-03,
          1.76900911e-03,   1.07295826e-03,   2.39409349e-04,
          1.96519161e-05]])
```

4 a and b)

Though convolution and correlation may seem similar for some certain examples, they ultimately have different applications. Convolution is associative whereas correlation is not. Thus we need two different functions ‘signal.convolve2d’ and ‘signal.correlate2d’ in Scipy.

Code for question 4 (all parts including 4a, 4b and 4c):

```
# Question 4
#Part a
def gaussconvolve2d(array,sigma):
    gc2d_filter = gauss2d(sigma);
    result = signal.convolve2d(array, gc2d_filter, 'same')
    return result

# Though convolution and correlation may seem similar for some certain examples, they ultimately
# have different applications. Convolution is associative whereas correlation is not. Thus we need
# two different functions ‘signal.convolve2d’ and ‘signal.correlate2d’ in Scipy.

#Part b
image_file = "/Users/Terence/Desktop/dog.jpg"

#convert image to grey scale, apply numpy array and calls gaussconvolve2d function
image = Image.open(image_file).convert('L')
array = np.asarray(image)
convolve = gaussconvolve2d(array, 3)

#Part c

# Original image converted to greyscale
image.show()

# Filtered image

# Filters images using the gaussconvolve2d function
filtered_img = Image.fromarray(np.uint8(convolve))
filtered_img.show()
```

4c)

Original greyscale:

Filtered:

5)

Given that we are aware of the separability of the Gaussian blur, we may effectively simplify the process by multiplying the functions using Fourier transforms. For clarification, by dividing the Gaussian blur function into the two functions f(x) and f (y). The convolution process is then effectively reduced to multiplication by using the Fourier transform on the functions. Consequently, the entire process becomes more effective.

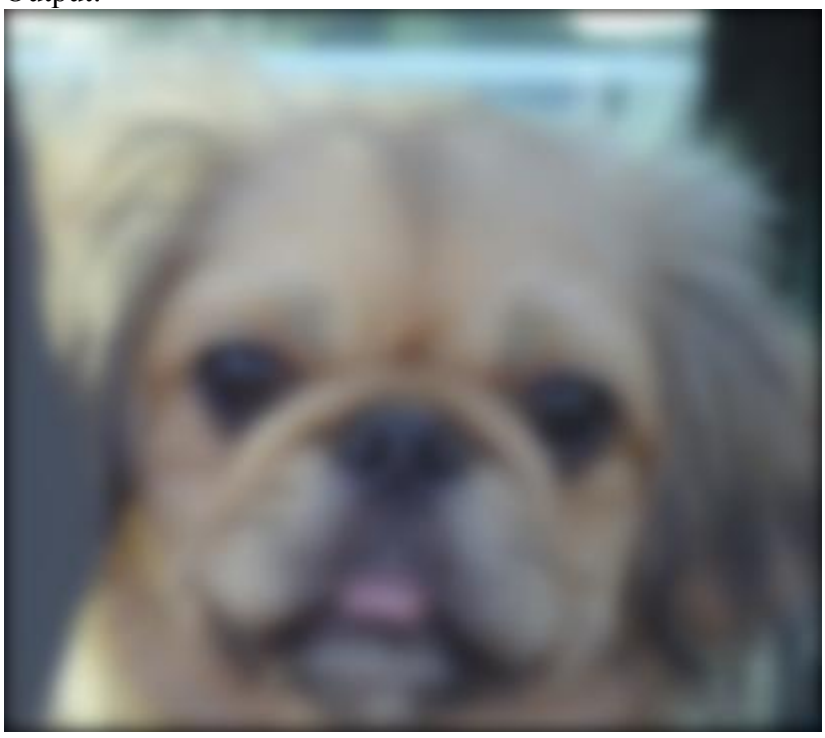
Part 2: Hybrid Images

1)

Code:

```
#Question 1
dog_image_file = "/Users/Terence/Desktop/hw1/0b_dog.bmp"
dog = Image.open(dog_image_file)
d_array = np.asarray(dog)
red = gaussconvolve2d(d_array[:,:,0], 5)
green = gaussconvolve2d(d_array[:,:,1], 5)
blue = gaussconvolve2d(d_array[:,:,2], 5)
red=red[:,:,np.newaxis]
green=green[:,:,np.newaxis]
blue=blue[:,:,np.newaxis]
filtered_dog = Image.fromarray(np.uint8(np.concatenate((red, green, blue ), axis=2)))
filtered_dog.show()
```

Output:



2)

Code:

```
#Question 2
cat_image_file = "/Users/Terence/Desktop/hw1/0a_cat.bmp"
cat = Image.open(cat_image_file)
c_array = np.asarray(cat)
c_red =  c_array[:,:,0] - gaussconvolve2d(c_array[:,:,0], 5)+128
c_green = c_array[:,:,1] - gaussconvolve2d(c_array[:,:,1], 5)+128
c_blue =  c_array[:,:,2] - gaussconvolve2d(c_array[:,:,2], 5)+128
c_red = c_red[:,:,np.newaxis]
c_green = c_green[:,:,np.newaxis]
c_blue = c_blue[:,:,np.newaxis]
filtered_cat = Image.fromarray(np.uint8(np.concatenate((c_red, c_green, c_blue ), axis=2)))
filtered_cat.show()
```

Output:

3)

Code:

```
#Question 3
# new arrays without the +128 for high frequency
c_red1 =  c_array[:,:,0] - gaussconvolve2d(c_array[:,:,0], 5)
c_green1 = c_array[:,:,1] - gaussconvolve2d(c_array[:,:,1], 5)
c_blue1 =  c_array[:,:,2] - gaussconvolve2d(c_array[:,:,2], 5)
c_red1 = c_red1[:,:,np.newaxis]
c_green1 = c_green1[:,:,np.newaxis]
c_blue1 = c_blue1[:,:,np.newaxis]

#adding the low and high frequency images and clipping
filtered_combined = Image.fromarray(np.uint8(np.concatenate((np.clip((c_red1 + red),0,255), np.clip(c_green1 + green, 0 ,255), np.clip(c_blue1 + blue,0,255)), axis=2)))
filtered_combined.show()
```

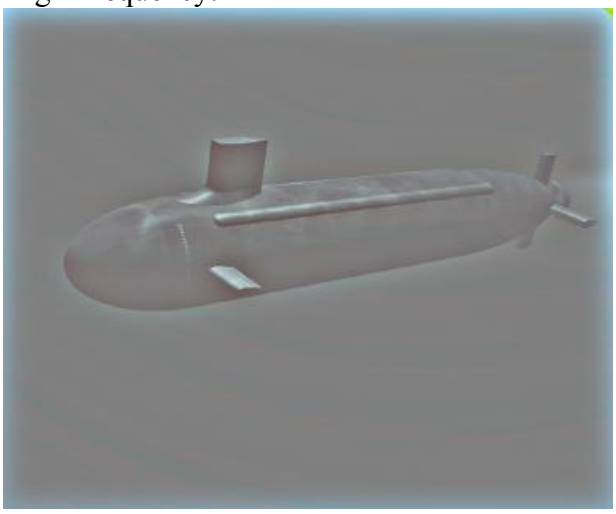
Output:

Note: Due to the similarity of the code to the code above and the requirement to repeat the process above for 2 more pairs of images, I will not repeat the code on the pdf in order to save space (code not required as specified in piazza). Below shown are the low, high, and hybrid image.

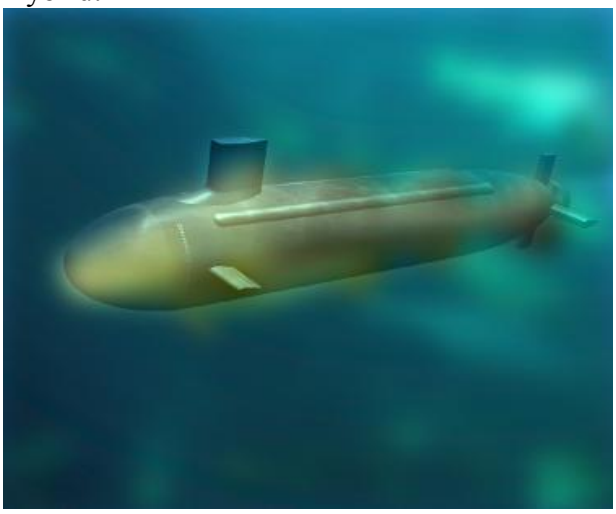
Images used: fish and submarine with sigma = 8

Low frequency:

High Frequency:



Hybrid:



Images used: bird and plane with sigma = 12

Low frequency:

High frequency:

Hybrid: