CPSC 425 - Assignment 1

Eloise Peng
13812169

Sept. 24, 2019

Part 1

Q1:

functions:

```
# Help function to check if the parameter n is odd
is_odd = lambda n: n%2 == 1
```

```
# Part 1
# Q1
# Define function boxfilter(n) returns a box filter of size n by n
# Note: n is odd
def boxfilter( n ):
    "This returns a boxfilter with size n by n where n is odd"
    assert ( is_odd(n) ), "Dimension must be odd"

    # calculate value for each box
    value = filter_sum / (n * n)

    # insert values to form a numpy array of n by n
    return np.asfarray([[value] * n] * n)
```

outputs

```
>>> import a1
>>>
>>> a1.boxfilter(3)
array([[0.11111111, 0.11111111, 0.11111111],
       [0.11111111, 0.11111111, 0.11111111],
       [0.11111111, 0.11111111, 0.11111111]])
>>>
>>> a1.boxfilter(4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "a1.py", line 24, in boxfilter
    assert ( is_odd(n) ), "Dimension must be odd"
AssertionError: Dimension must be odd
>>>
>>> a1.boxfilter(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]])
>>>
```

Q2:

functions:

```
# Q2
# Define function gauss1d(sigma) returns a 1D Gaussian filter for a given value of sigma
def gauss1d(sigma):
    "This returns a 1D Gaussian filter for a given value of sigma"

    # Convert the length of Gaussian 1D array to each side of 0 as k
    # Length: 6 times sigma rounded up to the next odd integer
    # Generate a 1-D array arr of values x where the x is the distance from the center
    k = (math.ceil(6 * sigma)) // 2
    print (k)
    arr = np.arange(-k, k + 1)

    # Calculate the Gaussian value corresponding to each element
    # Gaussian function: exp(- x^2 / (2*sigma^2))
    filter = np.exp((- np.square(arr) / (2 * sigma ** 2)))

    # normalize values in  the filter so that they sum to 1
    # return np.asfarray(filter / sum(filter))
    return np.array(filter) / np.sum(filter)
```

output:

```
>>> a1.gauss1d(0.3)
1.0
array([0.00383626, 0.99232748, 0.00383626])
>>>
>>> a1.gauss1d(0.5)
1.0
array([0.10650698, 0.78698604, 0.10650698])
>>>
>>> a1.gauss1d(1)
3.0
array([0.00443305, 0.05400558, 0.24203623, 0.39905028, 0.24203623,
       0.05400558, 0.00443305])
>>>
>>> a1.gauss1d(2)
6.0
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 ])
>>>
```

Q3

functions:

```
# Q3
def gauss2d(sigma):
    "This returns a 2D Gaussian filter for a given value of sigma"

    f = gauss1d(sigma)
    # make 2d
    f = f[np.newaxis]
    ft = f.transpose()

    return np.asfarray(signal.convolve(f, ft))
```

output:

```
>>>
>>> a1.gauss2d(0.5)
1.0
array([[0.01134374, 0.08381951, 0.01134374],
       [0.08381951, 0.61934703, 0.08381951],
       [0.01134374, 0.08381951, 0.01134374]])
>>>
>>> a1.gauss2d(1)
3.0
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]])
>>>
```

Q4:

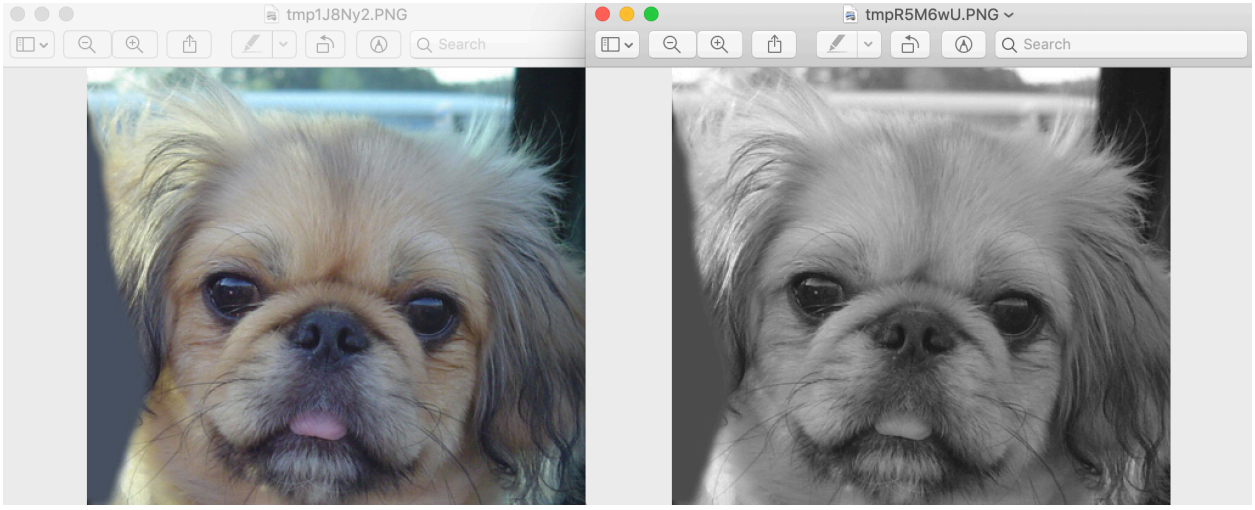
a) 'signal.convolve2d' and 'signal.correlate2d' will produce a different result when the filter is non-symmetric.

b)

```
>>> import numpy as np
>>> from PIL import Image
>>>
>>> im = Image.open('dog.jpg')
>>>
>>> # convert the image to a greyscale image
... im_grey = im.convert('L')
>>> # convert to Numpy array for (for subsequent processing)
... im_grey_array = np.asarray(im_grey)
>>> a1.gaussconvolve2d(im_grey_array, 3)
array([[ 93.9642061 , 106.26589855, 118.58846098, ...,  12.76419295,
         11.80531079,  10.88495968],
       [106.11828337, 120.0949488 , 134.015032  , ...,  14.37331242,
         13.31167404,  12.27395013],
       [118.21463839, 133.90452546, 149.45533022, ...,  16.15543263,
         14.9286089 ,  13.77406986],
       ...,
       [ 33.0364877 ,  39.22645384,  45.04542533, ...,  29.61356199,
         26.1585508 ,  23.06867143],
       [ 29.2608715 ,  35.01481777,  40.36645145, ...,  26.0770865 ,
         22.9842663 ,  20.08533743],
       [ 25.60125118,  30.91681269,  35.75437508, ...,  22.76356101,
         20.17946127,  17.79266471]])
>>>
>>> # convert the image to a numpy array (for subsequent processing)
... # convert the result back to a PIL image and save
... im_grey = Image.fromarray(im_grey_array)
>>>
```

c)

```
>>> im.show()
>>> im_grey.show()
>>>
```



Q5:

Since Gaussian kernels are separable, we can separate the 2D array into 2 vectors, and convolve each row and column accordingly. The new operation will reduce the runtime from O(m^2 + n ^2) to O(m^2 + 2n).

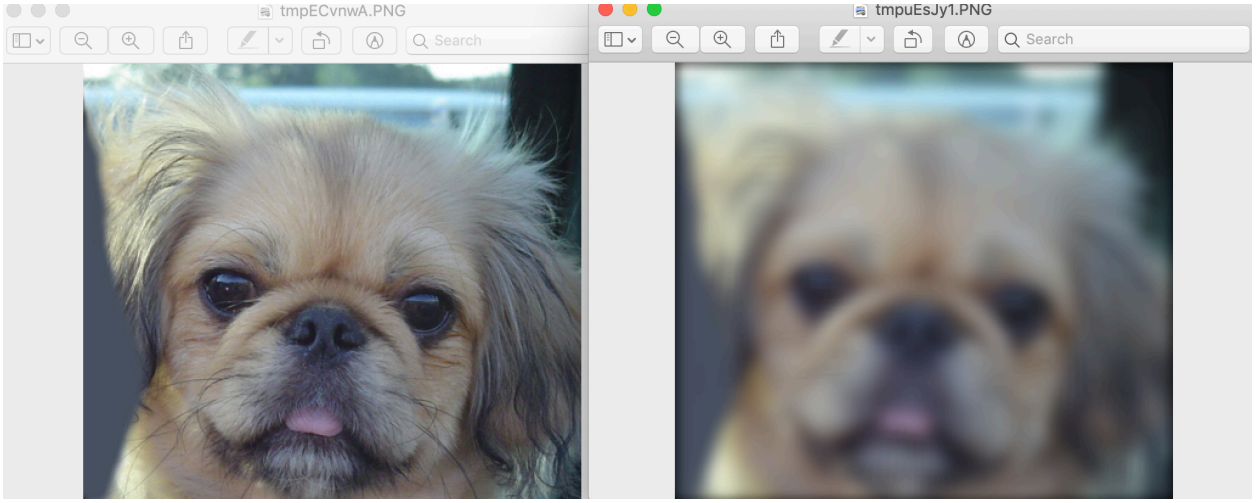
Part 2

function:

```python
# Part 2
# Q1-Q2
def lohipass_filter(path, sigma, is_high_pass):
    "Filter a image by its color channels separately and turn it into its blurred version"

    im = Image.open(path)
    im_array = np.asfarray(im)

    img_rgb = im.convert('RGB')
    r, g, b = img_rgb.split();
    r, g, b = np.asfarray(r), np.asfarray(g), np.asfarray(b)

    r = gaussconvolve2d(r, sigma)
    g = gaussconvolve2d(g, sigma)
    b = gaussconvolve2d(b, sigma)

    img_rgb_array = np.dstack((r, g, b))

    if is_high_pass: img_rgb_array = np.subtract(im_array, img_rgb_array) + scaler

    img_rgb = Image.fromarray(np.uint8(img_rgb_array))
    img_rgb.show()

    return img_rgb_array
```

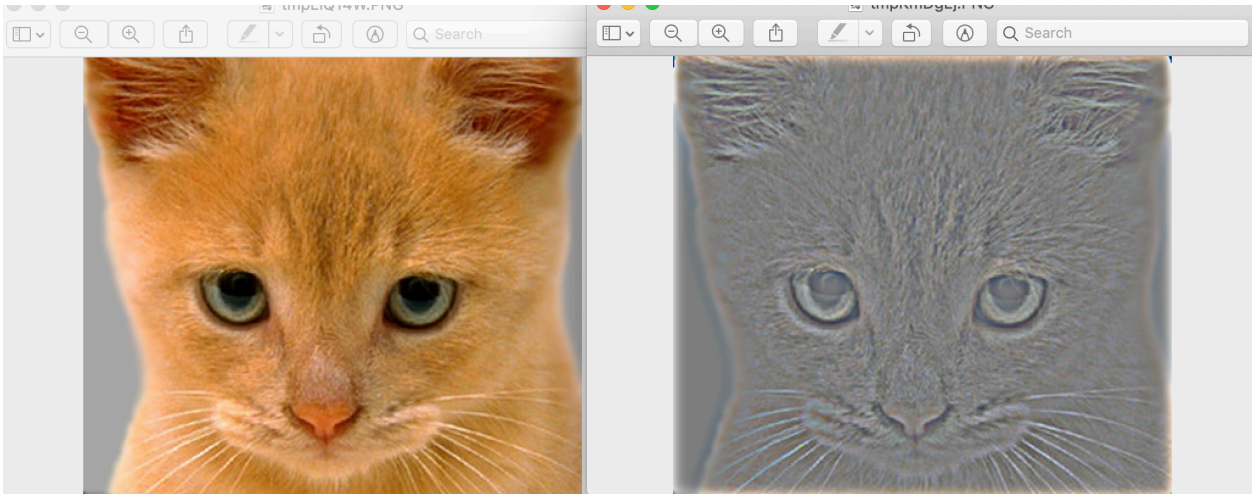
Q1: Picked sigma = 5

```
>>> a1.lohipass_filter('dog.jpg', 5, False)
```



Q2: Picked sigma = 5

```
>>> a1.lohipass_filter(cat.jpg', 5, True)
```



Q3:

```python
# Q3
def hybrid_filter(low_path, high_path, low_sigma, high_sigma):
    "Convert two images (1 high and 1 low frequency image) into one blurry hybrid image"
    "using lohipass_filter()"

    low = lohipass_filter(low_path, low_sigma, False)
    high = lohipass_filter(high_path, high_sigma, True)
    im = Image.fromarray(np.uint8(np.clip(np.add(low, high - scaler), lower_range, upper_range)))
    im.show()

    return im
```

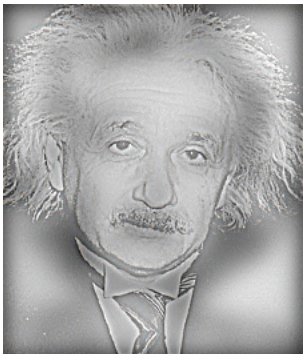
demos:

#1

```
>>> a1.hybrid_filter('dog.jpg','cat.jpg',5,8)
```

#2

```
>>> a1.hybrid_filter('2b_marilyn.bmp','2a_einstein.bmp',9, 2)
```



#3

```
>>> a1.hybrid_filter('hp_lego1.jpeg', ‘hp_lego1.png',2,8)
```

#4

```
>>> a1.hybrid_filter('4a_bird.bmp','4b_plane.bmp',3, 20)
```