DIYA REN
53483153

Assignment 1 Image Filtering and Hybrid Images

Part 1

1.1

In [1]:
```
import numpy as np

def boxfilter(n):
    #assertion statement for which n is not odd
    assert n%2!=0,"Dimension must be odd"
    #full n*n array with 0.04
    box = np.full((n,n),0.04)
    #sum up every element in the array
    sum = np.sum(box)
    #use the ratio of 0.04 and sum of the array elements
    #as the new array numer
    result = box/sum
    return result

#tests
print boxfilter(3)
print boxfilter(5)
print boxfilter(4)
```

```
[[0.11111111 0.11111111 0.11111111]
 [0.11111111 0.11111111 0.11111111]
 [0.11111111 0.11111111 0.11111111]]
[[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]]

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-1-bfca8b26f6c4> in <module>()
     16 print boxfilter(3)
     17 print boxfilter(5)
---> 18 print boxfilter(4)
     19
     20

<ipython-input-1-bfca8b26f6c4> in boxfilter(n)
      3 def boxfilter(n):
      4     #assertion statement for which n is not odd
----> 5     assert n%2!=0,"Dimension must be odd"
      6     #full n*n array with 0.04
      7     box = np.full((n,n),0.04)

AssertionError: Dimension must be odd
```

DIYA REN
53483153

1.2

jupyter gauss1d Last Checkpoint: 7 minutes ago (unsaved changes)

```
In [28]: import numpy as np
         import math

         def gauss1d(sigma):
             #get length of the array
             len = round(6*sigma)
             if len%2==0:
                 len = len+1
             else:
                 len = len

             #set the arrage of the array base on the size limit
             initArr = np.arange(-(int)(len/2),(int)(len/2+1))
             #get Guassian value bases on the formula
             initArr = (1/(2*math.pi*math.pow(sigma,2)))*(np.exp(-np.power(initArr,2)/(2*math.pow(sigma,2))))
             sumArr = np.sum(initArr)
             #normalization
             result = initArr/sumArr
             return result
         #tests
         print gauss1d(0.3)
         print gauss1d(0.5)
         print gauss1d(1)
         print gauss1d(2)
```

```
[0.00383626 0.99232748 0.00383626]
[0.10650698 0.78698604 0.10650698]
[0.00443305 0.05400558 0.24203623 0.39905028 0.24203623 0.05400558
 0.00443305]
[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 ]
```

DIYA REN
53483153

1.3

jupyter gauss2d Last Checkpoint: 8 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted | Python 2

Run Code

In [29]:

```
import numpy as np
import math
from scipy import signal
import matplotlib.pyplot as plt

def gauss1d(sigma):
    len = round(6*sigma)
    if len%2==0:
        len = len+1
    else:
        len = len

    initArr = np.arange(-(int)(len/2),(int)(len/2+1))
    initArr = (1/(2*math.pi*math.pow(sigma,2)))*(np.exp(-np.power(initArr,2)/(2*math.pow(sigma,2))))
    sumArr = np.sum(initArr)
    result = initArr/sumArr
    return result

def gauss2d(sigma):
    #get the 1D Gaussian
    gau1d = gauss1d(sigma)
    #convert 1D array into a 2D array
    gau1d = gau1d[np.newaxis]
    #get the transpose of the 2D array
    gauTrans = np.transpose(gau1d)
    #convolve two 2D arrays generate above
    ans = signal.convolve2d(gau1d,gauTrans)
    print "result for ",sigma," is:"
    return ans
#tests
print gauss2d(0.5)
print gauss2d(1)
```

```
result for  0.5  is:
[[0.01134374 0.08381951 0.01134374]
 [0.08381951 0.61934703 0.08381951]
 [0.01134374 0.08381951 0.01134374]]
result for  1  is:
[[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]]
```

DIYA REN
53483153

1.4

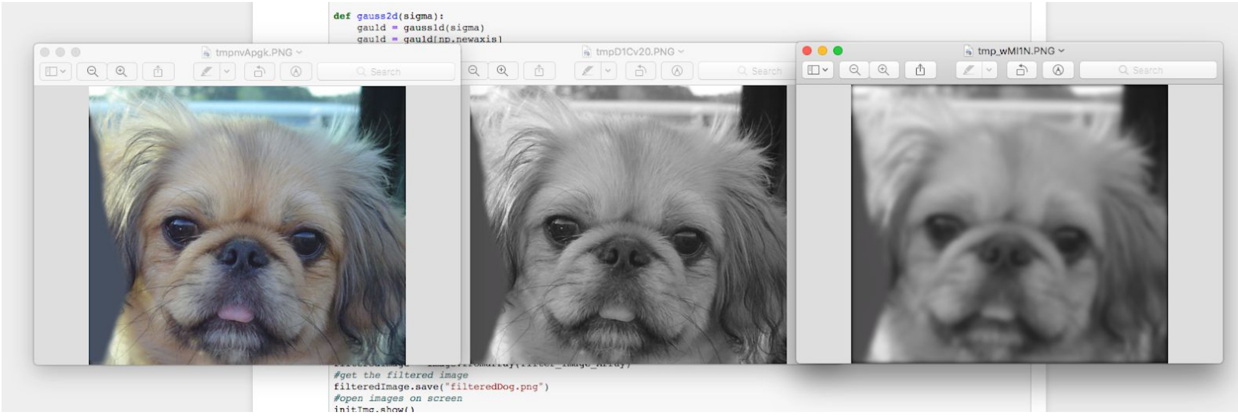
Part 1:

jupyter gaussconvolve2d Last Checkpoint: 17 minutes ago (unsaved changes)

In [12]:

```python
from PIL import Image
import numpy as np
import math
from scipy import signal


def gauss1d(sigma):
    len = round(6*sigma)
    if len%2==0:
        len = len+1
    else:
        len = len

    initArr = np.arange(-(int)(len/2),(int)(len/2+1))
    initArr = (1/(2*math.pi*math.pow(sigma,2)))*(np.exp(-np.power(initArr,2)/(2*math.pow(sigma,2))))
    sumArr = np.sum(initArr)
    result = initArr/sumArr
    return result

def gauss2d(sigma):
    gau1d = gauss1d(sigma)
    gau1d = gau1d[np.newaxis]
    gauTrans = np.transpose(gau1d)
    ans = signal.convolve2d(gau1d,gauTrans)
    print "result for ",sigma," is:"
    return ans

def gaussconvolve2d(array,sigma):
    gau2d = gauss2d(sigma)
    #applied Gaussian convolution to a 2D array for the given value of sigma
    ans = signal.convolve2d(array,gau2d,'same')
    return ans

#open and load the image
img = Image.open("/Users/diyaren/Desktop/CPSC425/Assignments/Assign1/dog.jpg")
img.load()
#copy the initial pic
initImg = img
#convert the RGB pic into greyscale
img = img.convert("L")
#save the greyscale pic
img.save("greyscale.png")
#convert the greyscale pic into Numpy array
numpyArr = np.asarray(img)
#apply Gaussian filter with sigma 3
filter_Image_Array = gaussconvolve2d(numpyArr,3)
#set data type to unit8 for 8 bit image
filter_Image_Array = filter_Image_Array.astype('uint8')
#convert from numpy array to image
filteredImage = Image.fromarray(filter_Image_Array)
#get the filtered image
filteredImage.save("filteredDog.png")
#open images on screen
initImg.show()
img.show()
filteredImage.show()
```

DIYA REN
53483153



Part 2:

Why does Scipy have seperate functions ‘signal.convolve2d’ and ‘signal.correlate2d’?

Ans:

Scipy.signal.correlate2d does a matched filtering(cross-correlation) of an N-dimensional signal and a N-dimensional template

Scipy.signal.convolve2d does a convolution of an N-dimensional signal and an N-dimensional kernel.

The matched filtering is done by correlating a known signal, or template, ,with an unknown signal to detect the presence of the template in the unknown signal. Scipy.signal.convolve2d and Scipy.signal.correlate2d will produce the same result only when convolve the unknown signal with a conjugated time-reversed version of the template. So when dealing with other cases, we still need to process them separately using different Scipy functions.

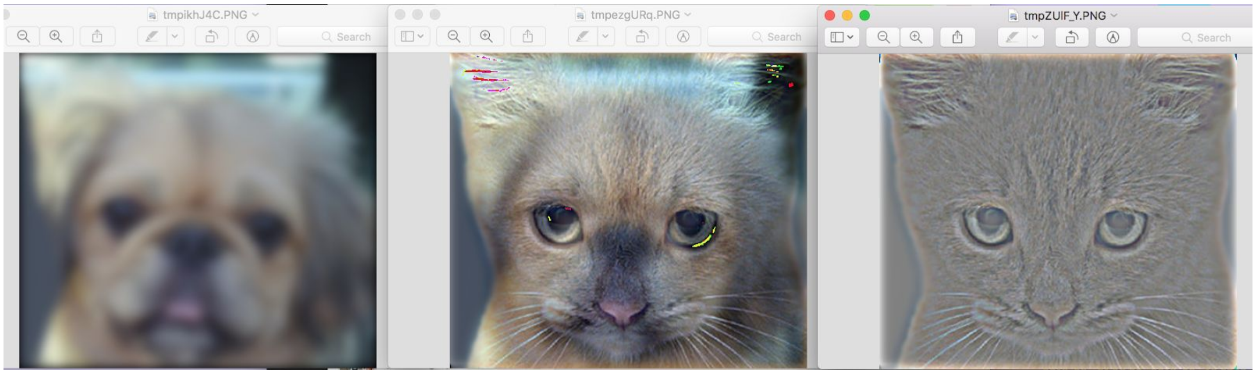
1.5 Convolution with a 2D Gaussian filter is not the most efficient way to perform Gaussian convolution on an image. In a few sentences, explain how this could be implemented more efficiently taking advantage of separability and why, indeed, this would be faster. NOTE: It is not necessary to implement this. Just the explanation is required. Your answer will be graded for clarity.

Ans: It is best to take advantage of the Gaussian separable property by dividing the process into two passes. In the first pass, a one-dimensional kernel is used to blur the image in only the horizontal or vertical direction. In the second pass, the same one-dimensional kernel is used to blur in the remaining direction. The resulting effect is the same as convolving with a two-dimensional kernel in a single pass, but requires fewer calculations.

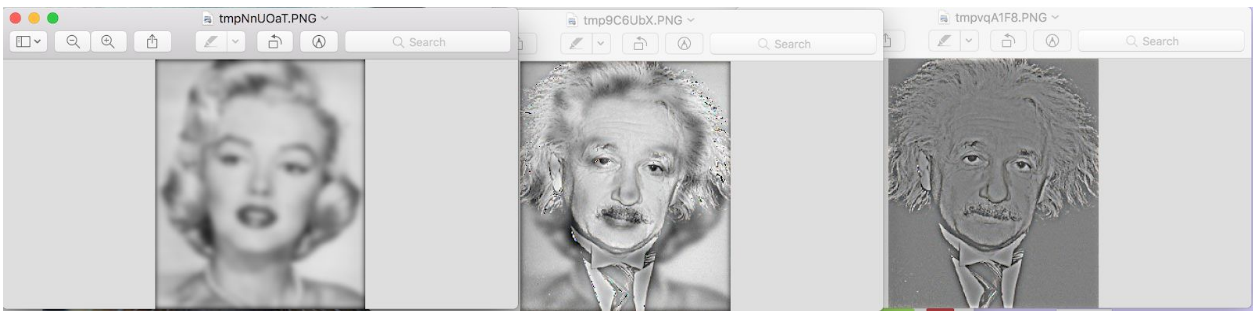
DIYA REN
53483153

Part 2

Set 1: use 6 as sigma



Set 2: use 3 as sigma



Set 3: use 1 as sigma