In [1]:
```
#Part 1: Gaussian Filtering
```

In [29]:
```
from PIL import Image
import numpy as np
import math
from scipy import signal
import PIL
```

In [3]:
```
#Question 1. A box filter that gives a number of deminsion as input and outpu
#(it checks if  the number of deminsion is even assert "Dimension must be odd
def boxfilter(n):

    assert (n%2 != 0),"Dimension must be odd"
    return 0.1*np.ones((n,n))
```

In [4]:
```
boxfilter(3)
```

Out[4]:
```
array([[0.1, 0.1, 0.1],
       [0.1, 0.1, 0.1],
       [0.1, 0.1, 0.1]])
```

In [5]:
```
boxfilter(4)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-5-5870f78beb34> in <module>()
----> 1 boxfilter(4)

<ipython-input-3-da69ecea03d2> in boxfilter(n)
      3 def boxfilter(n):
      4
----> 5     assert (n%2 != 0),"Dimension must be odd"
      6     return 0.1*np.ones((n,n))
      7

AssertionError: Dimension must be odd
```

In [6]:
```
boxfilter(5)
```

Out[6]:
```
array([[0.1, 0.1, 0.1, 0.1, 0.1],
       [0.1, 0.1, 0.1, 0.1, 0.1],
       [0.1, 0.1, 0.1, 0.1, 0.1],
       [0.1, 0.1, 0.1, 0.1, 0.1],
       [0.1, 0.1, 0.1, 0.1, 0.1]])
```

In [7]:

```
#Question 2


# A helper function, normilize a given array between 0 to 1
def normalizer(array):
    n = array/array.sum()
    return n

# A helper function, it gets sigma as an input and determine the size of the
def size_filter(sigma):

    #a value for deteimining the size of filter
    n = 6*sigma
    size_filter = 0

    #######
    #check the value of n, in order to determining the size of filter

    # if n is an integer odd number, then n is the size of filter
    if(n % 2 !=0 and n % 10 ==0):

        size_filter = n

     # if n is a decimal then we need to see what is the absolute value of n
    if(n % 2 !=0 and n % 10 !=0):

        # if absolute value of n is an even number like 2, then size of filte
        if(math.floor(n) % 2  == 0):
            size_filter = n + 1

        # if absolute value of n is an odd number like 3, then size of filter
        # because we should round up n to the nearest odd number
        else:
            size_filter = n + 2
    # if n is a integer number,then size of filter will be n+1
    else:
        size_filter = n + 1
   #########

    return size_filter

# it gets sigma as an input and outputs a 1-D Guassian filter
def gauss1d(sigma):

    size = size_filter(sigma)

    # making x array without using for loop,
    #using arrange method in numpy library and then ceiling the values to get
    a = math.ceil(3*sigma - 0.5)
    x = np.arange(-a,a+1)

    # Guassian filter = C exp(-X^2/ 2*sigma^). not that the calculation shoul
    guassian = np.exp(-x**2 /float(2*(sigma**2))).astype(float)

    #Normalize the resualt filter between 0 and 1 and retuen it
    return normalizer(guassian)
```

In [8]:
```
#Show the filter values produced for sigma values of 0.3, 0.5, 1, and 2.
gauss1d(0.3)
```

Out[8]:
```
array([0.00383626, 0.99232748, 0.00383626])
```

In [9]:
```
gauss1d(0.5)
```

Out[9]:
```
array([0.10650698, 0.78698604, 0.10650698])
```

In [10]:
```
gauss1d(1)
```

Out[10]:
```
array([0.00443305, 0.05400558, 0.24203623, 0.39905028, 0.24203623,
       0.05400558, 0.00443305])
```

In [11]:
```
gauss1d(2)
```

Out[11]:
```
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 ])
```

In [12]:
```
#Question 3

# it gets sigma as an input and outputs a 2-D Guassian filter
def gauss2d(sigma):
    #form a 1D filter and represent the 1D filter in the form of a 2D filter
    gauss1 = gauss1d(sigma)[np.newaxis]

    # Transpose Matrix of the gauss1 filter
    Tgauss1 = np.transpose(gauss1)

    # Convolve the 1D gaussian filter into its transpose to get the 2D gaussi
    gauss2 = signal.convolve2d(gauss1,Tgauss1)

    return gauss2
```

In [13]:
```
#Show the 2D Gaussian filter for sigma values of 0.5 and 1

gauss2d(0.5)
```

Out[13]:
```
array([[0.01134374, 0.08381951, 0.01134374],
       [0.08381951, 0.61934703, 0.08381951],
       [0.01134374, 0.08381951, 0.01134374]])
```

In [14]:
```
gauss2d(1)
```

Out[14]:
```
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]])
```

In [15]:
```
import matplotlib.pylab as plt
plt.imshow(gauss2d(10))
plt.colorbar()
plt.show()
a = gauss2d(100)
a
```

```
<Figure size 640x480 with 2 Axes>
```

Out[15]:
```
array([[1.97460201e-09, 2.03463586e-09, 2.09628528e-09, ...,
        2.09628528e-09, 2.03463586e-09, 1.97460201e-09],
       [2.03463586e-09, 2.09649491e-09, 2.16001866e-09, ...,
        2.16001866e-09, 2.09649491e-09, 2.03463586e-09],
       [2.09628528e-09, 2.16001866e-09, 2.22546718e-09, ...,
        2.22546718e-09, 2.16001866e-09, 2.09628528e-09],
       ...,
       [2.09628528e-09, 2.16001866e-09, 2.22546718e-09, ...,
        2.22546718e-09, 2.16001866e-09, 2.09628528e-09],
       [2.03463586e-09, 2.09649491e-09, 2.16001866e-09, ...,
        2.16001866e-09, 2.09649491e-09, 2.03463586e-09],
       [1.97460201e-09, 2.03463586e-09, 2.09628528e-09, ...,
        2.09628528e-09, 2.03463586e-09, 1.97460201e-09]])
```

In [16]:

```
#Question 4
#a)

# A gaussian 2D filter, convolve the given array with a 2D gaussian filter ba
def gaussconvolve2d(array,sigma):

    # making a 2D gaussian filter using gauss2d method
    filterr = gauss2d(sigma)

    #convolve the 2D gaussian filter with the given array
    g2d = signal.convolve2d(array, gauss2d(sigma),'same')

    #normilize the resault and return the value
    return g2d
```

a) continue, Why does Scipy have separate functions ‘signal.convolve2d’ and ‘signal.correlate2d’?

1)convolution is linear operations on the signal or signal modifier
2)correlation is a measure of similarity between two signals.

the basic difference between convolution and correlation is that the convolution process rotates the matrix by 180 degrees. Most of the time the choice of using the convolution and correlation is up to the preference of the users, and it is identical when the kernel is symmetrical.

As I mentioned above, in the case that the kernel is asymmetrical we get different resualts, and also they usully have a different ussage. so Scipy as a professional library should define two different functions for these two different methods.

In [17]:

```
#Question 4
#b)

#1)first we load the image of dog into python
from PIL import Image, ImageFilter
#Read image
im = Image.open( 'Dog.jpg' )


#2)convert it to grey scale and save the grey scale image
img = Image.open('Dog.jpg' ).convert('L')
img.save('GrayDog.jpg' )

#3)convert image to an numpy array
img_as_np = np.asarray(img).astype('float')

#4) run ‘gaussconvolve2d’ (with a sigma of 3).
filtered_img = gaussconvolve2d(img_as_np,3)
```

In [18]:

```
#Question 4
#c)
#1)convert the normalized filtered image array into the 0 to 255 scale
#and change the typr to unit 8 in order to visiulize the array as an image
filtered_img1 = ((filtered_img - filtered_img.min()) * (1/(filtered_img.max()

#2) save the array as an image using fromarray method in PIL library and show
GuassianfilteredImage = Image.fromarray(filtered_img1)
GuassianfilteredImage.save('filteredDog.jpeg')
GuassianfilteredImage
```

Out[18]:

In [19]:

```
#original image
im
```

Out[19]:

```
Question 5

it is best to take advantage of the Gaussian filter’s 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.
complexity of filtering an n×n image with an m×m kernel is O(n^2.m^2)
while for separabale kernel is O(n.m^2).
```

In [20]:

```
#Part 2: Hybrid Images
```

In [21]:

```
#Question 1

#open the dog image and covert the image to a numpy array
img =  Image.open( 'Dog.jpg' )
doggie = np.asarray(img).astype('float')

#separate the colored image array's channel
r =doggie[:,:,0]
g =doggie[:,:,1]
b = doggie[:,:,2]

#filter each channel diferently using "gaussconvolve2d" method(sigma = 5)
filtered_r = gaussconvolve2d(r,5)
filtered_g = gaussconvolve2d(g,5)
filtered_b = gaussconvolve2d(b,5)

#convert the normalized filtered channels' array into the 0 to 255 scale
#and change the typr to unit 8 in order to visiulize the array as an image
filtered_r1 = ((filtered_r - filtered_r.min()) * (1/(filtered_r.max() - filte
filtered_g1 = ((filtered_g - filtered_g.min()) * (1/(filtered_g.max() - filte
filtered_b1 = ((filtered_b - filtered_b.min()) * (1/(filtered_b.max() - filte

#convert each channels' array to image using formarray method
imr=Image.fromarray(filtered_r1)
imb=Image.fromarray(filtered_b1)
img=Image.fromarray(filtered_g1)

#merge 3 channels to get a colored filtered image of the dog
coloredDog=Image.merge("RGB",(imr,img,imb))
coloredDog
```

Out[21]:

In [ ]:

In [22]:

```
#Question 1

#open the cat image and covert the image to a numpy array
cat1 =  Image.open( 'cat.jpg' )
acat = np.asarray(cat1).astype('float')

#separate the colored image array's channel
cr = acat[:,:,0]
cg = acat[:,:,1]
cb = acat[:,:,2]


#filter each channel diferently using "gaussconvolve2d" method(sigma = 5)
filtered_cr = gaussconvolve2d(cr,5)
filtered_cg = gaussconvolve2d(cg,5)
filtered_cb = gaussconvolve2d(cb,5)


#convert the normalized filtered channels' array into the 0 to 255 scale
#and change the typr to unit 8 in order to visiulize the array as an image
filtered_cr1 = ((filtered_cr - filtered_cr.min()) * (1/(filtered_cr.max() - f
filtered_cg1 = ((filtered_cg - filtered_cg.min()) * (1/(filtered_cg.max() - f
filtered_cb1 = ((filtered_cb - filtered_cb.min()) * (1/(filtered_cb.max() - f


#to get the high frequency image of the cat we should subctract the normalize
#from their normalized low frequency image of the cat
#frequency image is actually zero-mean with negative values so it is visualiz
h_cr = normalizer(cr) - normalizer(filtered_cr1) + 0.5
h_cg = normalizer(cg) - normalizer(filtered_cg1)+ 0.5
h_cb = normalizer(cb) -normalizer(filtered_cb1) + 0.5

#convert the normalized filtered channels' array into the 0 to 255 scale
#and change the typr to unit 8 in order to visiulize the array as an image
h_cr = ((h_cr - h_cr.min()) * (1/(h_cr.max() - h_cr.min()) * 255)).astype('ui
h_cg = ((h_cg - h_cg.min()) * (1/(h_cg.max() - h_cg.min()) * 255)).astype('ui
h_cb = ((h_cb - h_cb.min()) * (1/(h_cb.max() - h_cb.min()) * 255)).astype('ui

#convert each channels' array to image using formarray method
imr1=Image.fromarray(h_cr)
img1=Image.fromarray(h_cg)
imb1=Image.fromarray(h_cb)


#merge 3 channels to get a colored filtered image of the cat
cat=Image.merge("RGB",(imr1,img1,imb1))
cat
```

Out[22]:

In [23]:
```
#Question 2
#covert the low frequency image of dog and high frequency image of cat to num
catt = np.asarray(cat).astype('float')
dogg = np.asarray(coloredDog).astype('float')

#separate the colored images array's channel

cr = catt[:,:,0]
cg = catt[:,:,1]
cb = catt[:,:,2]

dr = dogg[:,:,0]
dg = dogg[:,:,1]
db = dogg[:,:,2]

# add the normilized low frequency image of dog and normilized high frequency
hr = normalizer(cr) + normalizer(dr)
hg = normalizer(cg) + normalizer(dg)
hb = normalizer(cb) + normalizer(db)


#convert the normalized filtered channels' array into the 0 to 255 scale
#and change the typr to unit 8 in order to visiulize the array as an image
hr11 = ((hr - hr.min()) * (1/(hr.max() - hr.min()) * 255)).astype('uint8')
hg11 = ((hg - hg.min()) * (1/(hg.max() - hg.min()) * 255)).astype('uint8')
hb11 = ((hb - hb.min()) * (1/(hb.max() - hb.min()) * 255)).astype('uint8')

#convert each channels' array to image using formarray method
hr1=Image.fromarray(hr11)
hg1=Image.fromarray(hg11)
hb1=Image.fromarray(hb11)

#merge 3 channels to get the hybrid image
hybrid=Image.merge("RGB",(hr1,hg1,hb1))
hybrid
```

Out[23]:

In [24]:

```
#Question 2

def imageresizer(image, basewidth = 500, baseheight = 400):
   # img = Image.open(image)
    img1 = image.resize((basewidth, baseheight), PIL.Image.ANTIALIAS)
    return img1

def lowfrequencyimage(image,sigma):

    image =  Image.open( image )

    #open the image and covert the image to a numpy array
    image = np.asarray(image).astype('float')

    #separate the colored image array's channel
    r =image[:,:,0]
    g =image[:,:,1]
    b =image[:,:,2]

    #filter each channel diferently using "gaussconvolve2d" method
    filtered_r = gaussconvolve2d(r,sigma)
    filtered_g = gaussconvolve2d(g,sigma)
    filtered_b = gaussconvolve2d(b,sigma)

    #convert the normalized filtered channels' array into the 0 to 255 scale
    #and change the typr to unit 8 in order to visiulize the array as an imag
    filtered_r1 = ((filtered_r - filtered_r.min()) * (1/(filtered_r.max() - f
    filtered_g1 = ((filtered_g - filtered_g.min()) * (1/(filtered_g.max() - f
    filtered_b1 = ((filtered_b - filtered_b.min()) * (1/(filtered_b.max() - f

    #convert each channels' array to image using formarray method
    imr=Image.fromarray(filtered_r1)
    imb=Image.fromarray(filtered_b1)
    img=Image.fromarray(filtered_g1)

    #merge 3 channels to get a colored filtered image of the dog
    fimage=Image.merge("RGB",(imr,img,imb))
    return fimage

def highfrequencyimage(image, sigma):

    image =  Image.open( image )

    #covert the image to a numpy array
    acat = np.asarray(image).astype('float')

    #separate the colored image array's channel
    cr = acat[:,:,0]
    cg = acat[:,:,1]
    cb = acat[:,:,2]


    #filter each channel diferently using "gaussconvolve2d" method
    filtered_cr = gaussconvolve2d(cr,sigma)
    filtered_cg = gaussconvolve2d(cg,sigma)
    filtered_cb = gaussconvolve2d(cb,sigma)
```

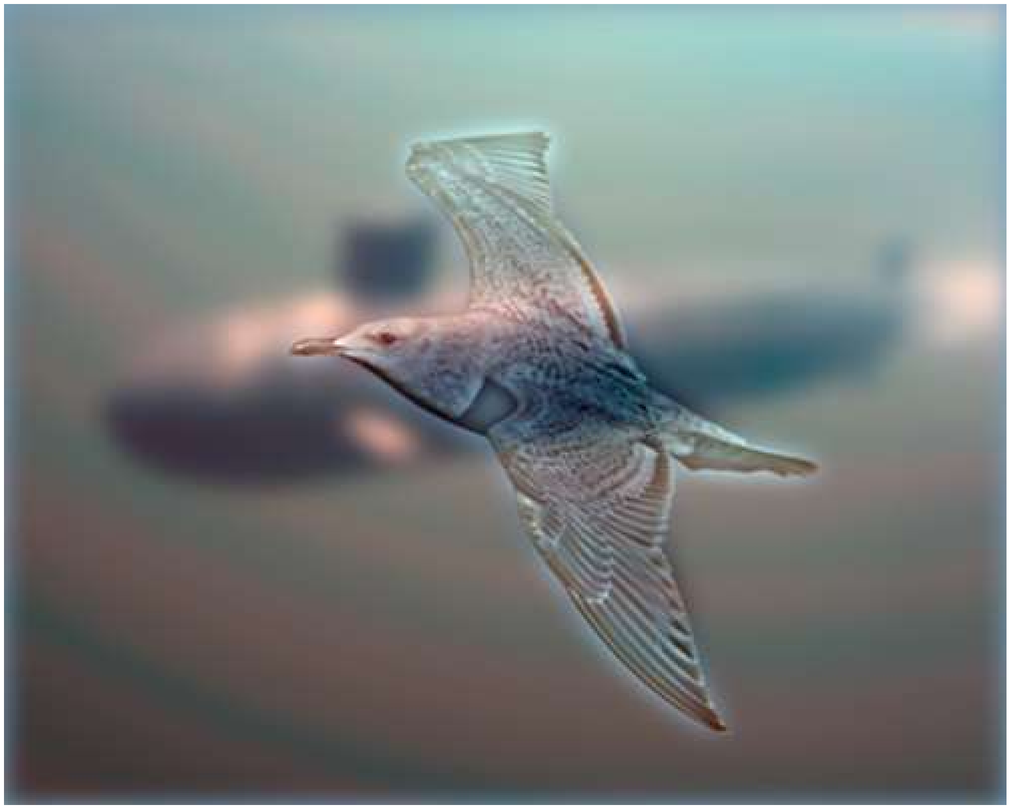
```
    #convert the normalized filtered channels' array into the 0 to 255 scale
    #and change the typr to unit 8 in order to visiulize the array as an imag
    filtered_cr1 = ((filtered_cr - filtered_cr.min()) * (1/(filtered_cr.max()
    filtered_cg1 = ((filtered_cg - filtered_cg.min()) * (1/(filtered_cg.max()
    filtered_cb1 = ((filtered_cb - filtered_cb.min()) * (1/(filtered_cb.max()


    #to get the high frequency image of the cat we should subctract the norma
    #from their normalized low frequency image of the cat
    #frequency image is actually zero-mean with negative values so it is visu
    h_cr = normalizer(cr) - normalizer(filtered_cr1) + 0.5
    h_cg = normalizer(cg) - normalizer(filtered_cg1)+ 0.5
    h_cb = normalizer(cb) -normalizer(filtered_cb1) + 0.5

    #convert the normalized filtered channels' array into the 0 to 255 scale
    #and change the typr to unit 8 in order to visiulize the array as an imag
    h_cr = ((h_cr - h_cr.min()) * (1/(h_cr.max() - h_cr.min()) * 255)).astype
    h_cg = ((h_cg - h_cg.min()) * (1/(h_cg.max() - h_cg.min()) * 255)).astype
    h_cb = ((h_cb - h_cb.min()) * (1/(h_cb.max() - h_cb.min()) * 255)).astype

    #convert each channels' array to image using formarray method
    imr1=Image.fromarray(h_cr)
    img1=Image.fromarray(h_cg)
    imb1=Image.fromarray(h_cb)


    #merge 3 channels to get a colored filtered image of the cat
    hfimage=Image.merge("RGB",(imr1,img1,imb1))
    return hfimage









# making hybrid of different pictures, low frequncy of image 1 + high frequen
def hybrid(image1,image2,sigma):


    image1 = lowfrequencyimage(image1,sigma)
    image2= highfrequencyimage(image2, sigma)
    print(image1.size)
    print(image2.size)
    image1 = imageresizer(image1, basewidth = 500, baseheight = 400)
    image2 = imageresizer(image2, basewidth = 500, baseheight = 400)
    #covert the low frequency image of image1 and high frequency image of ima
    #image1 =  Image.open( image )
    #image2 =  Image.open( image )

    image1 = np.asarray(image1).astype('float')
```

```
    image2 = np.asarray(image2).astype('float')

    #separate the colored images array's channel

    cr11 = image1[:,:,0]
    cg11 = image1[:,:,1]
    cb11 = image1[:,:,2]

    dr11 = image2[:,:,0]
    dg11 = image2[:,:,1]
    db11 = image2[:,:,2]

    # add the normilized low frequency image1 and normilized high frequency i
    hr = normalizer(cr11) + normalizer(dr11)
    hg = normalizer(cg11) + normalizer(dg11)
    hb = normalizer(cb11) + normalizer(db11)


    #convert the normalized filtered channels' array into the 0 to 255 scale
    #and change the typr to unit 8 in order to visiulize the array as an imag
    hr11 = ((hr - hr.min()) * (1/(hr.max() - hr.min()) * 255)).astype('uint8'
    hg11 = ((hg - hg.min()) * (1/(hg.max() - hg.min()) * 255)).astype('uint8'
    hb11 = ((hb - hb.min()) * (1/(hb.max() - hb.min()) * 255)).astype('uint8'

    #convert each channels' array to image using formarray method
    hr1=Image.fromarray(hr11)
    hg1=Image.fromarray(hg11)
    hb1=Image.fromarray(hb11)

    #merge 3 channels to get the hybrid image
    hybrid=Image.merge("RGB",(hr1,hg1,hb1))
    return hybrid
```

In [30]:
```
hybrid('3b_submarine.bmp', '4a_bird.bmp',5)
```

```
(375, 307)
(375, 331)
```

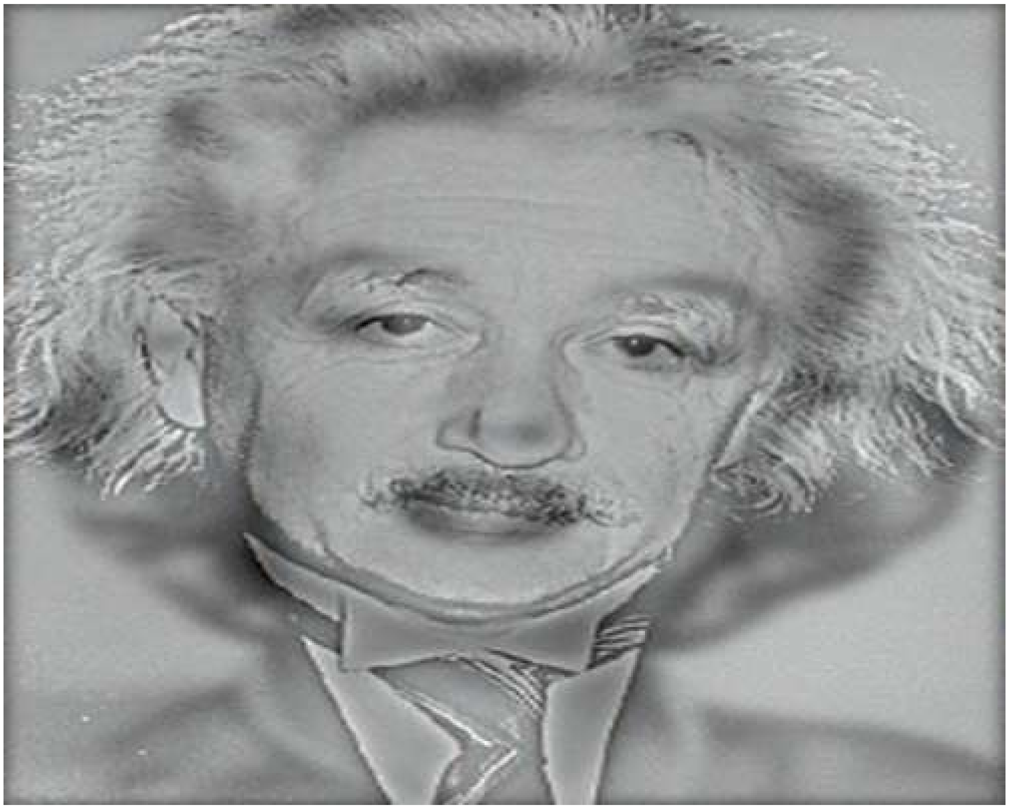
Out[30]:



In [31]:
```
hybrid('2b_marilyn.bmp', '2a_einstein.bmp',2.5)
```

```
(225, 265)
(225, 265)
```

Out[31]:



In [32]:
```
hybrid('2a_einstein.bmp', '2b_marilyn.bmp',5)
```

```
(225, 265)
(225, 265)
```

Out[32]:

In [33]:
```
hybrid('4b_plane.bmp', '4a_bird.bmp',5)
```

```
(375, 331)
(375, 331)
```

Out[33]:

In [34]:
```
hybrid('4a_bird.bmp', '4b_plane.bmp',5)
```

```
(375, 331)
(375, 331)
```

Out[34]:

In [ ]:

In [ ]:

In [ ]: