Hybrid Images

Introduction

Hybrid images was proposed in the SIGGRAPH 2006 paper by Oliva, Torralba, and Schyns. Here is the webpage of their work.

Hybrid images are static images that change in interpretation as a function of the viewing distance. The basic idea is that high frequency tends to dominate perception when it is available, but, at a distance, only the low frequency (smooth) part of the signal can be seen. By blending the high frequency portion of one image with the low-frequency portion of another, you get a hybrid image that leads to different interpretations at different distances.

Low-frequency parts of the image can be understood as "contours", such as the shape of the face. "high attached microdermabrasion" 微晶磨皮

The high-frequency parts of the image can be understood as "details", such as wrinkles and spots on the face.

Therefore, we often say that the low-frequency part of the image is obtained after the image is blurred, and the image is sharpened to make the high-frequency information of the image more.

Basic steps

The overall idea of the implementation is simple-superimpose a picture with only low-frequency information and an image with only high-frequency information. Specific steps are as follows:

1. Prepare low-frequency filters (usually Gaussian blur filter).
2. Convolution of each dimension of the first image with Gaussian filter.
3. Convolution of each dimension of the second image with Gaussian filter, and subtract the filtered image from the original image to obtain the high-frequency image. ! NOTE that the Gaussian filter in this step may have different standard deviation from the above one. You need to try different values to produce satisfatory results.
4. Add the two processed images to get a hybrid image.

Implementation details

1. Read a pair of images and alignment

Read a pair of images. You can use the the given images for test, but for submission, you have to use your own images. The image pair should be well aligned for satisfatory results. You need to mannualy select two points on each image and calculate the rotation and scaling parameters.


For example, the above cat and dog are not aligned. We need to rotate and scale the dog image to align with the cat. We select the centers of the two eyes on the dog image, then we can caculate the rotation angle and the scaling factor.

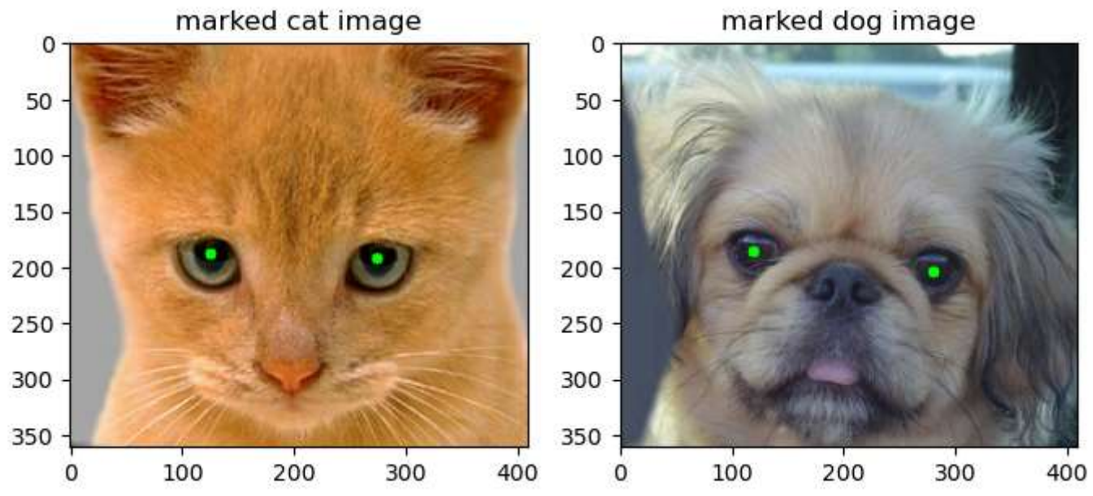
This method exploys two points on each image. You need to think of appropriate steps to do the alignment. The following steps are just one possible way:

1. Select two points A1 and A2 on image A for alignement; and selcet two points B1 and B2 on image B;
2. Translate image B so that B1 is aligned with A1;
3. Rotate image B to make the line B1B2 have the same direction with line A1A2;
4. Scale image B, so that the distance between B1 and B2 is equal to the distance between A1 and A2;
5. Crop the images to make the two image have the same size.

Please write your code below to align your image pair. Note that, you can use OpenCV functions for rotation and scaling.

In [ ]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt

# for example, the two points on the first image are:
p11 = [100, 200]
p12 = [200, 200]
# and the corresponding two points on the second image are:
p21 = [120, 190]
p22 = [190, 220]
# TODO: write your own code to calculate the rotation and scaling parameter

# read images
img_cat = cv2.cvtColor(cv2.imread('cat.jpg'), cv2.COLOR_BGR2RGB)
img_dog = cv2.cvtColor(cv2.imread('dog.jpg'), cv2.COLOR_BGR2RGB)

# calculate the move between two points
def calcMove(psrc, pdst):
    return [pdst[0]-psrc[0], pdst[1]-psrc[1]]

# calculate the angle between two vectors
def calcAngle(v1, v2):
    norm1 = np.linalg.norm(v1)
    norm2 = np.linalg.norm(v2)
    r_theta = np.arccos(np.dot(v1, v2)/ (norm1 * norm2))
    theta = r_theta * 180 / np.pi
    return theta

# calculate the scale rate between two vectors
```

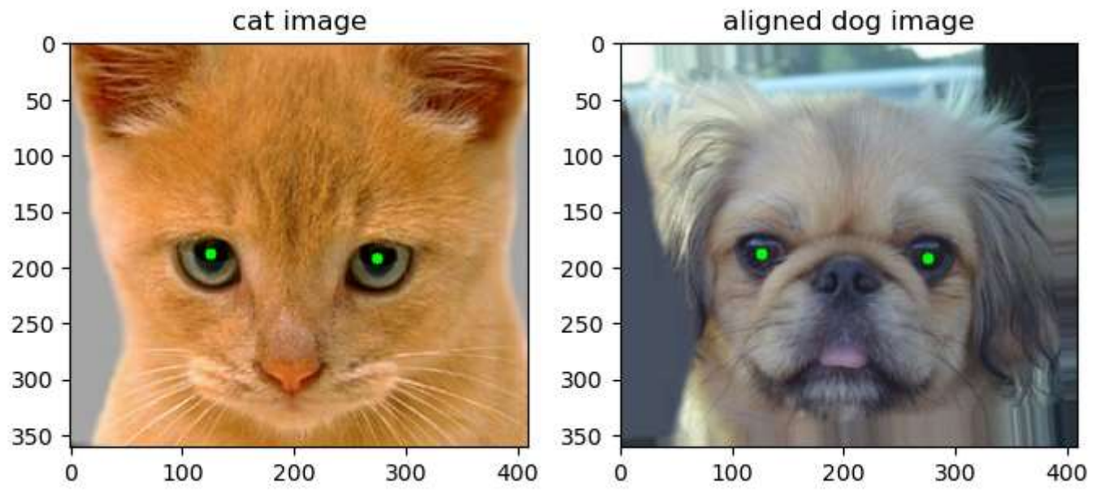
```
def calcRate(v1, v2):
    norm1 = np.linalg.norm(v1)
    norm2 = np.linalg.norm(v2)
    f = norm1 / norm2
    return f

# align the images
def Align(img, trans, center, angle, scale):
    rows, cols, _ = img.shape
    # translate
    Mat = np.array([[1, 0, trans[0]], [0, 1, trans[1]]], np.float32)
    img = cv2.warpAffine(img, Mat, (cols, rows), borderMode=cv2.BORDER_REPLICATE)
    # rotate and scale
    Mat = cv2.getRotationMatrix2D(center, angle, scale)
    img = cv2.warpAffine(img, Mat, (cols, rows), borderMode=cv2.BORDER_REPLICATE)
    return img

# mark the points
def Mark(img, points, radius=5, color=[0, 0, 0]):
    res = np.copy(img)
    for point in points:
        cv2.circle(res, point, radius, color, thickness=-1)
    return res

# draw multiple images
def ImageListPlot(img_list, title_list=None):
    length = len(img_list)
    plt.figure(figsize=(4*length, 4))
    for i in range(length):
        plt.subplot(1, length, i + 1)
        plt.imshow(img_list[i])
        if title_list:
            plt.title(title_list[i])

p11, p12 = [126, 189], [275, 193]
p21, p22 = [119, 187], [280, 205]

marked_img_cat = Mark(img_cat, [p11, p12], color=[0, 255, 0])
marked_img_dog = Mark(img_dog, [p21, p22], color=[0, 255, 0])

ImageListPlot([marked_img_cat, marked_img_dog], ['marked cat image', 'marked dog image'])
```



In [ ]:
```
# TODO: apply the rotation and scaling on the image
# translate, rotate and scale
```

```
trans = calcMove(p21, p11)
v1 = [p12[0] - p11[0], p12[1] - p11[1]]
v2 = [p22[0] - p21[0], p22[1] - p21[1]]
angle = calcAngle(v1, v2)
f = calcRate(v1, v2)
img_dog = Align(img_dog, trans, p21, angle, f)

# use points from cat image to validate the align effect of dog image
marked_img_dog = Mark(img_dog, [p11, p12], color=[0, 255, 0])
print(f"img_cat.shape: {img_cat.shape}, img_dog.shape: {img_dog.shape}")
ImageListPlot([marked_img_cat, marked_img_dog],["cat image", "aligned dog image"])
```

```
img_cat.shape: (361, 410, 3), img_dog.shape: (361, 410, 3)
```



2. Make Gaussian filters

You need to make your own Gaussian filters even the OpenCV has provided you with some easy to use functions as follows.

In [ ]:
```
# import cv2
# out = cv2.GaussianBlur(in, 0, 4)
```

When you make your own Gaussian filter, you have to specify the filter size (usually odd number) and the standard deviation. In OpenCV, if the kernerl size is given but the standard deviation σ is not given, then σ can be determined according to:

$$\sigma = 0.3 * ((ksize - 1) * 0.5 - 1) + 0.8$$

If the σ is given but the kernel size is not given, then the kernel size can be se to 6σ or 8σ (usually an odd number), as we have discussed in the lecture that $[-3\sigma, 3\sigma]$ of a Gaussian function captures 99.7% of the energy.

This blog gives a detailed discussion on the selection of kernel size and σ.

Please finish the following function to generate your Gaussian filter:

NOTE!!!! The sum of the kernel weights should be 1.

In [ ]:
```
# Given the filter size and the standard deviation, a Gaussian filter can be computed.
def MakeGaussianFilter(ksize=None, sigma=None):
    if ksize is None and sigma is None:
        ksize = 3
        sigma = 0.8
    elif ksize is None:
        ksize = int(6 * sigma)
```

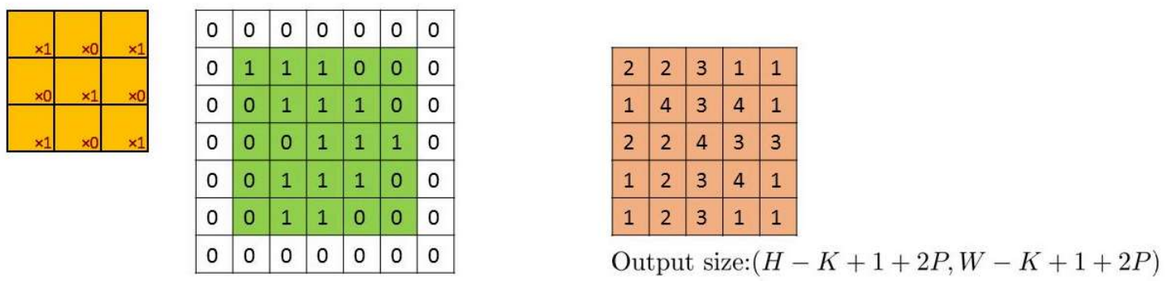
```
    else:
        sigma = 0.3 * ((ksize - 1) * 0.5 - 1) + 0.8
    if ksize % 2 == 0:
        raise Exception(f'ksize is equal to {ksize}, which should be odd!!!')
    kernel = np.zeros((ksize, ksize))
    # calculate the center's position
    center = ksize // 2
    sum = 0.0
    for i in range(ksize):
        x = i - center
        for j in range(ksize):
            y = j - center
            kernel[i][j] = np.exp(-(x**2+y**2)/(2*sigma**2))/(2*np.pi*sigma**2)
            sum += kernel[i][j]
    # The sum of the kernel weights should be 1.
    kernel /= sum
    return kernel

kernel1 = MakeGaussianFilter(3, 0.8)

kernel2 = cv2.getGaussianKernel(3, 0.8)
kernel2 = kernel2 @ kernel2.T

print("my Gaussian kernel:\n", kernel1)
print("opencv's Gaussian kernel:\n",kernel2)
```

```
my Gaussian kernel:
 [[0.05711826 0.12475775 0.05711826]
 [0.12475775 0.27249597 0.12475775]
 [0.05711826 0.12475775 0.05711826]]
opencv's Gaussian kernel:
 [[0.05711826 0.12475775 0.05711826]
 [0.12475775 0.27249597 0.12475775]
 [0.05711826 0.12475775 0.05711826]]
```

3. Padding your images

We will use zero-padding to keep the filtered image having the same size with the input image. The padding should be half size of your filter kernel. For example, the following figure shows a 3x3 filter and a 5x5 input image. With padding on the top, bottom, left and right with 1 row/column zeros (note, the filter size 3 divided by 2, therefore the padding is 1, 3//2=1), the filtered image has the same size of the input (5x5). If your filter size is, for example 21x21, then the padding should be 10 rows/columns of zeros on each of the 4 boarders.

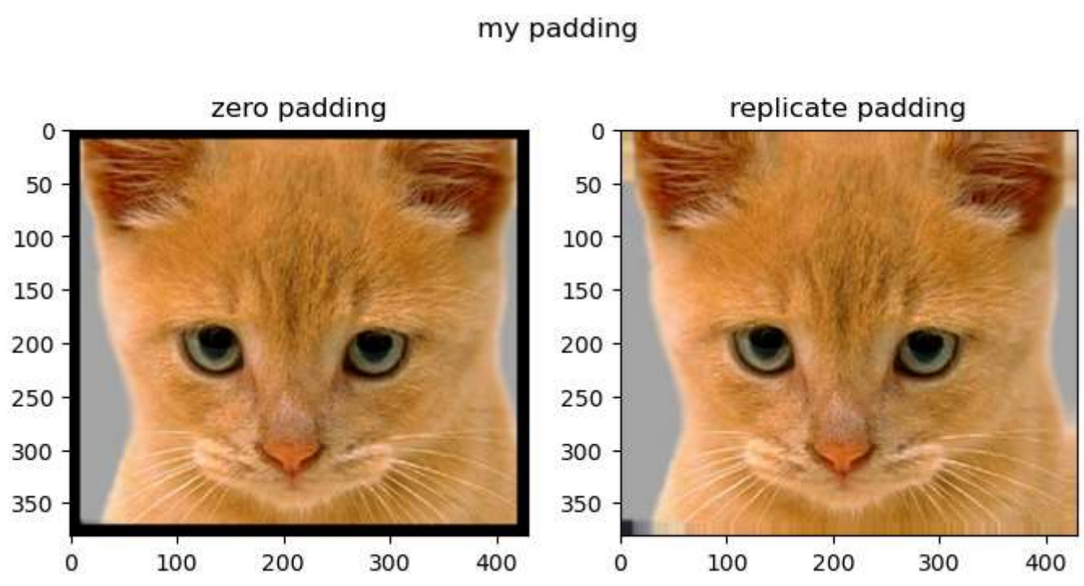
- Convolution with zero-padding of P pixels

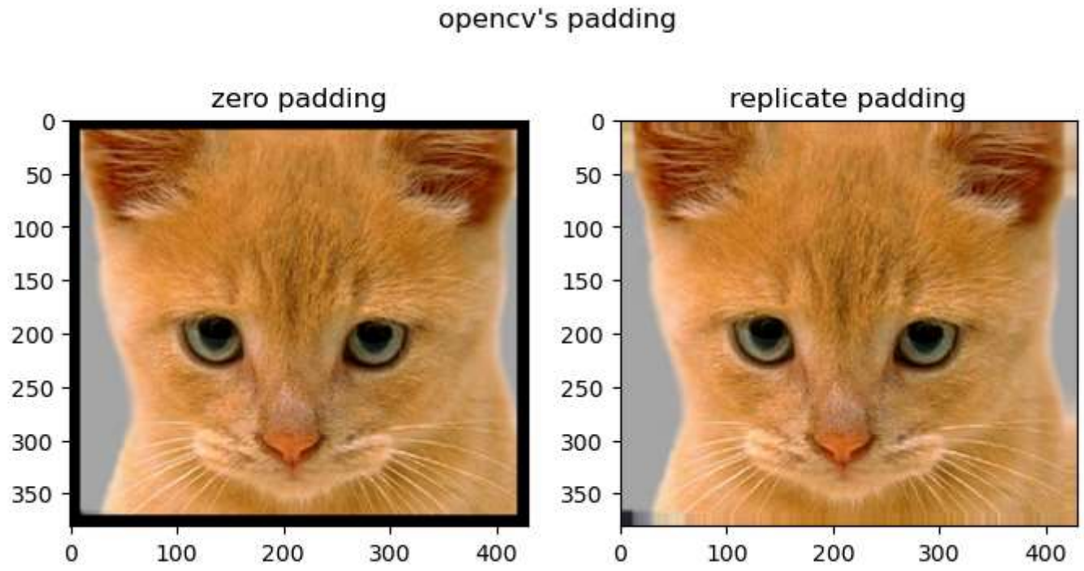


In [ ]:
```
# Given image, padding size and mode, a padded image will return.
def Padding(im, pad, mode='zero'):
    rows, cols, channels = im.shape
    im_padded = np.zeros((rows+2*pad, cols+2*pad, channels), dtype=im.dtype)
    # default zero padding
    im_padded[pad:pad+rows, pad:pad+cols,:] = im
```

```
    # replicate padding
    if mode == 'replicate':
        # fill four edges
        im_padded[0:pad, pad:cols+pad, :] += im_padded[pad, pad:cols+pad, :]
        im_padded[rows+pad:, pad:cols+pad, :] += im_padded[rows+pad-1, pad:cols+pad, :]
        im_padded[pad:rows+pad, 0:pad, :] += im_padded[pad:rows+pad, pad, None, :]
        im_padded[pad:rows+pad, cols+pad:, :] += im_padded[pad:rows+pad, cols+pad-1, None, :]
        # fill four cornors
        im_padded[0:pad, 0:pad, :] += im_padded[pad, None, pad, None, :]
        im_padded[rows+pad:, 0:pad, :] += im_padded[rows+pad-1, None, pad, None, :]
        im_padded[0:pad, cols+pad:, :] += im_padded[pad, None, cols+pad-1, None, :]
        im_padded[rows+pad:, cols+pad:, :] += im_padded[rows+pad-1, None, cols+pad-1, None, :]
    return im_padded

padded_img_cat1 = Padding(img_cat, 10)
padded_img_cat2 = Padding(img_cat, 10, mode='replicate')
ImageListPlot([padded_img_cat1, padded_img_cat2], ["zero padding", "replicate padding"])
plt.suptitle("my padding")

padded_img_cat3 = cv2.copyMakeBorder(img_cat, 10, 10, 10, 10, cv2.BORDER_CONSTANT)
padded_img_cat4 = cv2.copyMakeBorder(img_cat, 10, 10, 10, 10, cv2.BORDER_REPLICATE)
ImageListPlot([padded_img_cat3, padded_img_cat4], ["zero padding", "replicate padding"])
plt.suptitle("opencv's padding")
```

Out[ ]: Text(0.5, 0.98, "opencv's padding")



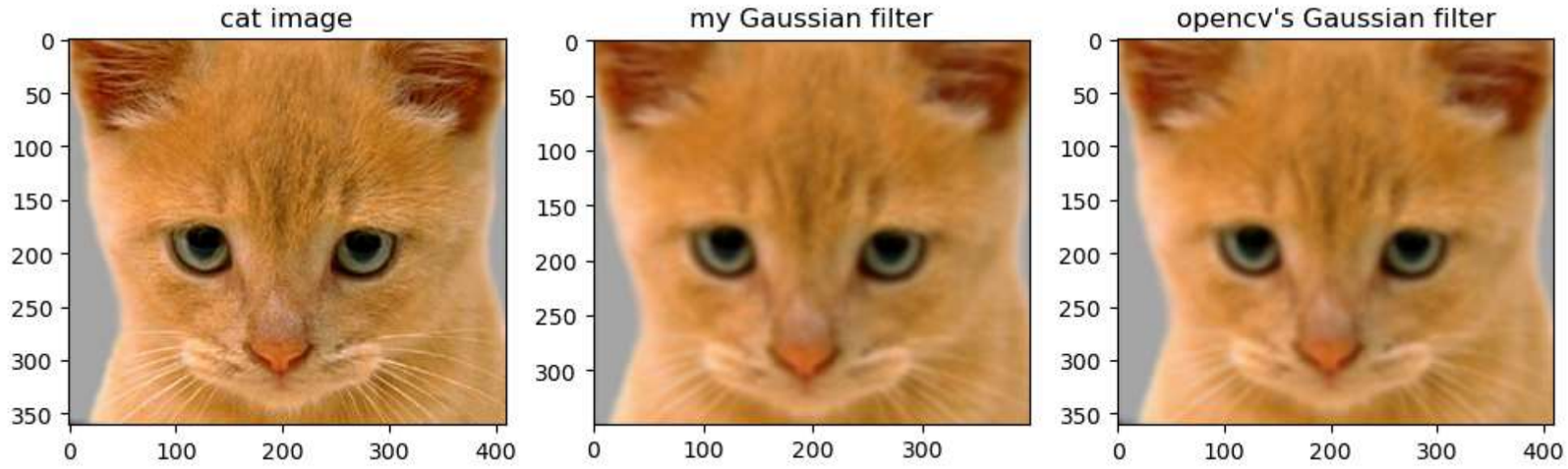


4. Convolution

Write your own code for the 2D convolution function. Use a filter to convolve on an image (RGB image).

For just this assignment, you are forbidden from using any Numpy, Scipy, OpenCV, or other preimplemented functions for filtering. You are allowed to use basic matrix operations like np.shape, np.zeros, and np.transpose. This limitation will be lifted in future assignments, but for now, you should use for loops or Numpy vectorization to apply a kernel to each pixel in the image.

In [ ]:
```
def Convolution2D(im, filter):
    fsize = filter.shape[0]
    rows, cols = im.shape
    Rows = rows - fsize + 1
    Cols = cols - fsize + 1
    result = np.zeros((Rows, Cols))
    for i in range(Rows):
        for j in range(Cols):
            result[i, j] = np.sum(np.multiply(im[i:i+fsize, j:j+fsize], filter))
    return result

def Convolution3D(im, filter):
    fsize = filter.shape[0]
    rows, cols, channels = im.shape
    Rows = rows - fsize + 1
    Cols = cols - fsize + 1
    result = np.zeros((Rows, Cols, channels))
    for k in range(channels):
        result[:, :, k] = Convolution2D(im[:, :, k], filter)
    result = result.astype(np.uint8)
    return result

filter = MakeGaussianFilter(13)
filtered_img_cat1 = Convolution3D(img_cat, filter)
sigma = 0.3*((13-1)*0.5-1) + 0.8
filtered_img_cat2 = cv2.GaussianBlur(img_cat, (13, 13), sigma)
ImageListPlot([img_cat, filtered_img_cat1, filtered_img_cat2], ["cat image", "my Gaussian filter", "opencv's Gaussian filter"])
```

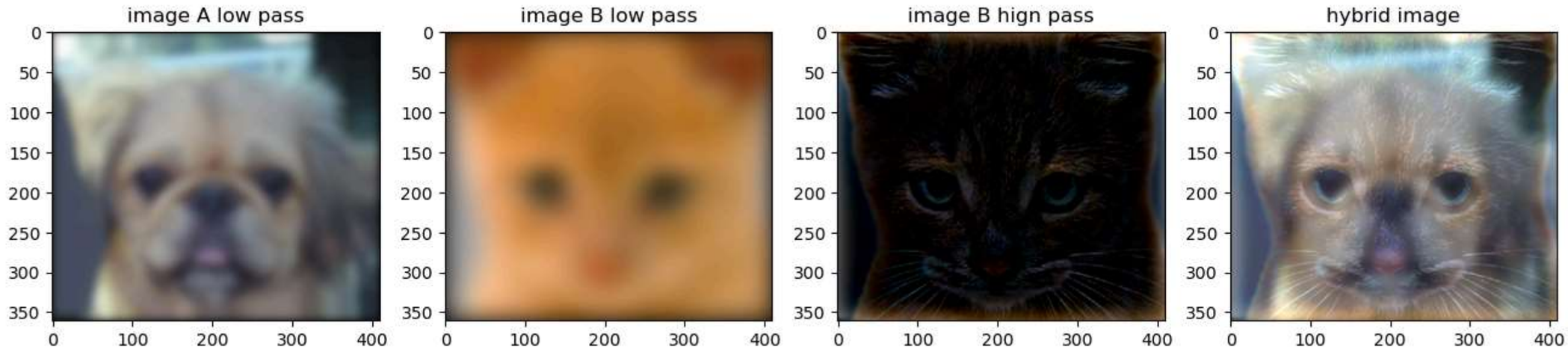


5. Filter your images and make hybrid image

Make a Gaussian filter to convolve on image A, get the blured image A'. Make another Gaussian filter to convolve on image B, and substract the result from the original B, get the detail image B'. Add A' and B' to get the hybrid image.

NOTE! You may have to try different values for the standard deviation in Gaussian filters to get satifactory results.

In [ ]:
```
# TODO:
def GaussianBlur(im, ksize, sigma=None):
    filter = MakeGaussianFilter(ksize, sigma)
    # use padding to keep the filtered image having the same size with the input image
    im_padded = Padding(im, (ksize-1)//2)
    im_filtered = Convolution3D(im_padded, filter)
    return im_filtered

def Hybrid(im_A, im_B, ksize, sigma):
    # get low pass of two images
    low_pass_A = GaussianBlur(im_A, ksize[0], sigma[0])
    low_pass_B = GaussianBlur(im_B, ksize[1], sigma[1])
    # get high pass of B image
    high_pass_B = cv2.subtract(im_B, low_pass_B)
    # get hybrid image
    hybrid = cv2.add(low_pass_A, high_pass_B)
    image_list = [low_pass_A, low_pass_B, high_pass_B, hybrid]
    title_list = ["image A low pass","image B low pass","image B hign pass","hybrid image"]
    ImageListPlot(image_list, title_list)
    return hybrid

img_hybrid = Hybrid(img_dog, img_cat, ksize=[35, 101], sigma=[6, 15])
```



6. Make Gaussian pyramid to show your hybrid image

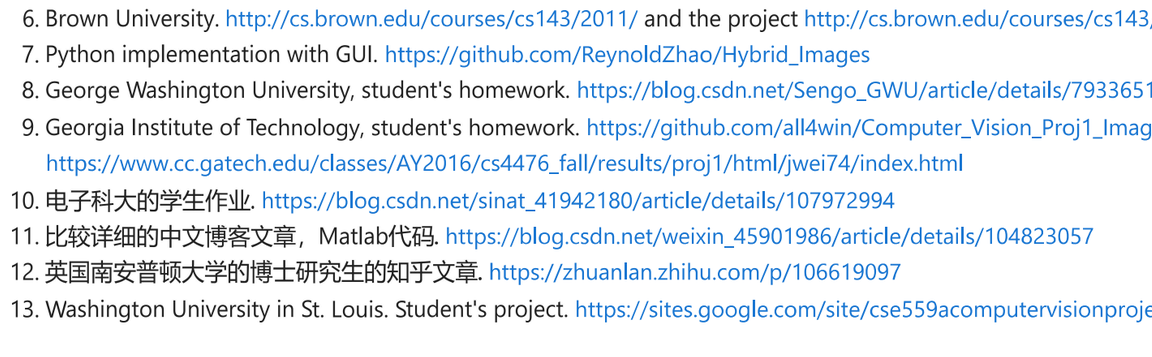
Use Gaussian filter to build up a Gaussian pyramid. The lower level image in the pyramid should exhibit high frequency information (details); while the higher level image in the pyramid should exhibit low frequency information.

In [ ]:

```python
# TODO:
# subsample the image
def subSample(img, step=2):
    # use the Gaussian filter to blur the image
    img = GaussianBlur(img, ksize=3)
    rows, cols, channels = img.shape
    Rows, Cols = (rows - 1 + step) // step, (cols - 1 + step) // step
    result = np.zeros((Rows, Cols, channels), dtype=np.uint8)
    # subsample from original image
    for k in range(channels):
        for i in range(Rows):
            for j in range(Cols):
                result[i, j, k] = img[i * step - step + 1, j * step - step + 1, k]
    return result

# draw the Gaussian Pyramid
def GaussianPyramid(img, n=4, sep=5):
    image_list = [img]
    rows, cols, channels = img.shape
    # subsample n images, set seperation between them.
    for i in range(n):
        img = subSample(img)
        image_list.append(img)
```

```
        cols += sep + img.shape[1]
    result = np.ones((rows, cols, channels), dtype=np.uint8) * 255
    left = 0
    # draw the Gaussian Pyramid
    for im in image_list:
        row, col = im.shape[0], im.shape[1]
        result[rows-row:rows, left:left+col, :] = im
        left += sep + col
    return result

plt.figure(figsize=(12, 5))
# drop the axis
plt.xticks([])
plt.yticks([])
plt.axis('off')
# draw the Gaussian Pyramid
img_pyramid = GaussianPyramid(img_hybrid)
plt.imshow(img_pyramid)
```

Out[ ]: <matplotlib.image.AxesImage at 0x1d7e1cbafe0>



References

1. The origianl authors' project page. http://olivalab.mit.edu/hybridimage.htm
2. A Tutorial. https://jeremykun.com/2014/09/29/hybrid-images/
3. Chinese blog，matlab code, with alignment。 https://blog.csdn.net/breeze_blows/article/details/102962559
4. Matlab code. https://github.com/coldmanck/Image-Filtering-and-Hybrid-Images
5. Soton University. http://comp3204.ecs.soton.ac.uk/cw/coursework2.html
6. Brown University. http://cs.brown.edu/courses/cs143/2011/ and the project http://cs.brown.edu/courses/cs143/2011/proj1/
7. Python implementation with GUI. https://github.com/ReynoldZhao/Hybrid_Images
8. George Washington University, student's homework. https://blog.csdn.net/Sengo_GWU/article/details/79336511
9. Georgia Institute of Technology, student's homework. https://github.com/all4win/Computer_Vision_Proj1_Image_Filtering_and_Hybrid_Images and https://www.cc.gatech.edu/classes/AY2016/cs4476_fall/results/proj1/html/jwei74/index.html
10. 电子科大的学生作业. https://blog.csdn.net/sinat_41942180/article/details/107972994
11. 比较详细的中文博客文章，Matlab代码. https://blog.csdn.net/weixin_45901986/article/details/104823057
12. 英国南安普顿大学的博士研究生的知乎文章. https://zhuanlan.zhihu.com/p/106619097
13. Washington University in St. Louis. Student's project. https://sites.google.com/site/cse559acomputervisionprojects/home/project-1-hybrid-images