ENSE 885AY

Application of Deep Learning in Computer Vision

Assignment A01

Image Filtering and Hybrid Images

Instructed by

Dr. Kin-Choong Yow

Student:

Marzieh Zamani

List of Sections and Sub-sections

1. Introduction

1.1. Overview (Key points from the assignment description) [1]

Assignment Subject:

Image Filtering and Hybrid Images

Assignment objectives:

Writing an image filtering function and using it to create hybrid images.

Definition of Hybrid Images:

Hybrid images are static images that are perceived differently when viewing from close or far distance.

Basic Idea for Creation of Hybrid Images:

Looking from close distance, we see the higher frequencies of an image;

Looking from far distance, we see the lower frequencies of an image;

Therefore, if we extract the higher frequencies of image 1 and the lower frequencies of image 2 and blend them together into a hybrid image, we would perceive image 1 (HF*) when looking closely and perceive image 2 (LF*) when looking distantly.

HF*: High Frequency

LF*: Low Frequency

Steps to creating a hybrid image:

1. Create a 2D Gaussian filter to extract the lower frequencies of an input image
2. Load image1 and image2
3. Apply filter to image1 to obtain low-frequency image1 => image1_LF
4. Apply filter to image2 to obtain low-frequency image2 => image2_LF

5. Subtract low-frequency image2 from original image to obtain high-frequency image2
   ⇨ image2 _HF = image2 - image2_LF
6. Add low-frequency image1 and high-frequency image2 to obtain the hybrid image
   ⇨ hybrid image = image1_LF + image2_HF

1.2. Filter Review: Gaussian Blurring

Gaussian blurring (or Gaussian smoothing) is an image is blurred through convolution of a Gaussian function over the image [2].

A 1D Gaussian filter can be obtained using cv.getGaussianKernel function from OpenCV library [3]:

Gaussian_1D = cv.getGaussianKernel(ksize, sigma)

The output of this function is a ksize×1 matrix of Gaussian filter coefficients (G(i)):

$G(i) = \alpha * e^{[-(i-(ksize-1)/2)2/(2 * sigma2)]},$

where

i = 0 … ksize−1

α = the normalizing scale factor (so that $\sum_i G(i) = 1$)

sigma = Gaussian standard deviation

Then, a 2D Gaussian filter can be obtained by multiplication of two 1D Gaussian filters [4].

$Gaussian\_2D = Gaussian\_1D * Gaussian\_1D^T$

The 2D Gaussian filter used in this assignment (figure 1) has can be described by following parameters:

- ksize = 29
- sigma = cut-off frequency = 7

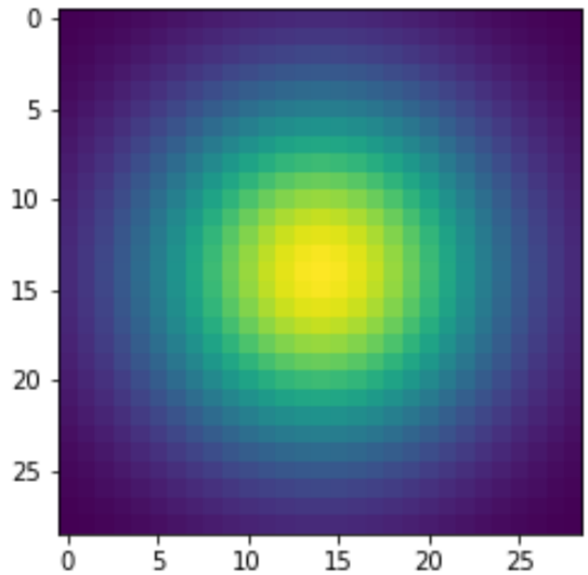


Figure 1: 2D Gaussian filter with size = 29 and sigma = 7

2. Student Code

2.1. Implantation of the Filter (my_imfilter function)

filtered_image = my_imfilter(image, filter)

Step 1) Pre-processing: Padding Image (with mirrored edges)

In order to convolve the filter over all image pixels including edge pixels, we need to pad image edges. The pad size is equal to half of (filter size – 1). The padding could be simply done using np.pad function:

```
# Pad the input image with mirrored edges
pad_h = int((filter.shape[0]-1)/2)
pad_w = int((filter.shape[1]-1)/2)
padded_image = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w), (0, 0)), 'symmetric')
```

Step 2) Applying the filter over the input image

The approach of filter implementation would be as follows:

For every layer of the input image:

For every pixel (i, j) of the image layer:

- Extract the neighborhood window with the same size of filter and centered on the image pixel (i,j) which corresponds to pixel (i – pad_h, j – pad_w, layer) of padded_image (padded_image[row:row+filter.shape[0], col:col+filter.shape[1], layer])

- Multiply the neighborhood matrix by the filter matrix (dot product);
- Sum all entries of the resulting matrix to obtain the output value for pixel (i,j);

These algorithm was accomplished with following code:

```
# Apply the filter to the input image
filtered_image = image.copy()
for layer in range(image.shape[2]):
    for row in range(image.shape[0]):
        for col in range(image.shape[1]):
            filtered_image[row][col][layer] = np.sum(np.multiply(padded_image[row:row + filter.shape[0], col:col + filter.shape[1], layer], filter))
```

2.2. Creating Hybrid Image (create_hybrid_image function)

low_frequencies, high_frequencies, hybrid_image = create_hybrid_image(image1, image2, filter)

Step 1) Obtain low-frequency image1 and high-frequency image2

- Apply filter to image1 to obtain low-frequency image1;
- Apply filter to image2 to obtain low-frequency image2;
- Subtract low-frequency image2 from original image to obtain high-frequency image2;

```
# Apply filter to obtain low_frequency image1
low_frequencies = my_imfilter(image1, filter)

# Apply filter to obtain low_frequency image2
low_freq_image2 = my_imfilter(image2, filter)

# Subtract low_frequency image2 from original image to obtain high_frequencies image2
high_frequencies = image2 - low_freq_image2
```

Step 2) Add low-frequney image1 and high-frequency image2 to obtain the hybrid image

```
# Sum  low_frequencies & high_frequencies to obtain hybrid image
```

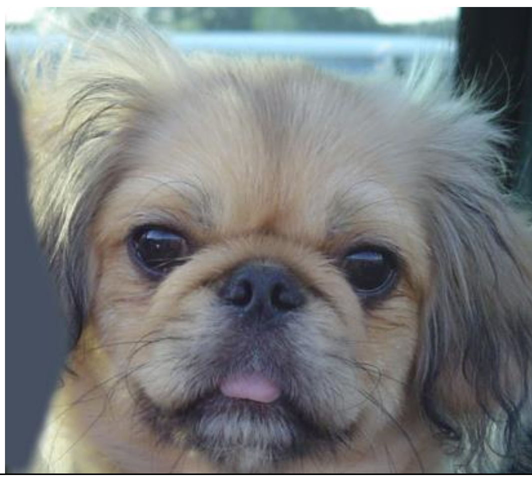
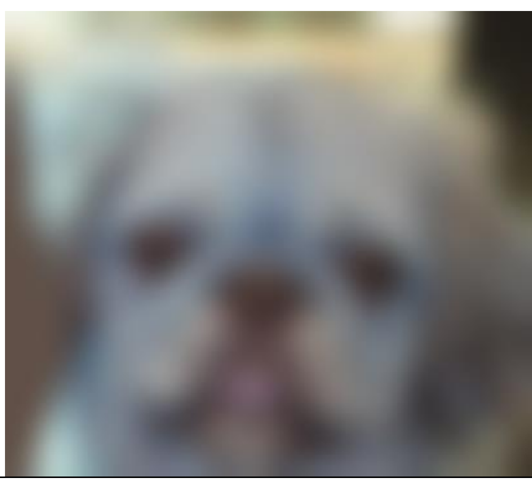
```
hybrid_image = low_frequencies + high_frequencies
```

3. Results and Discussion

3.1. Hybrid Images Using Provided Data & Tuned Cut-off Frequency

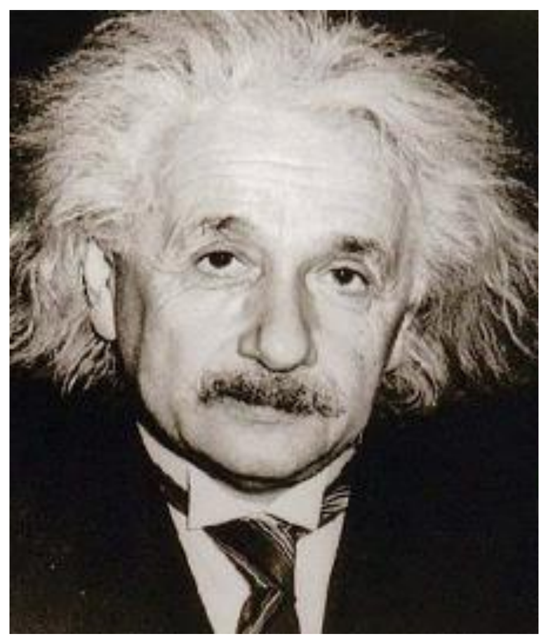
This section presents the hybrid images for the image pairs provided as input data. Cut-off frequency is tuned for each pair of images to obtain balanced view of both low-frequency and high frequency images. Results are presented in tables 1 to 5.

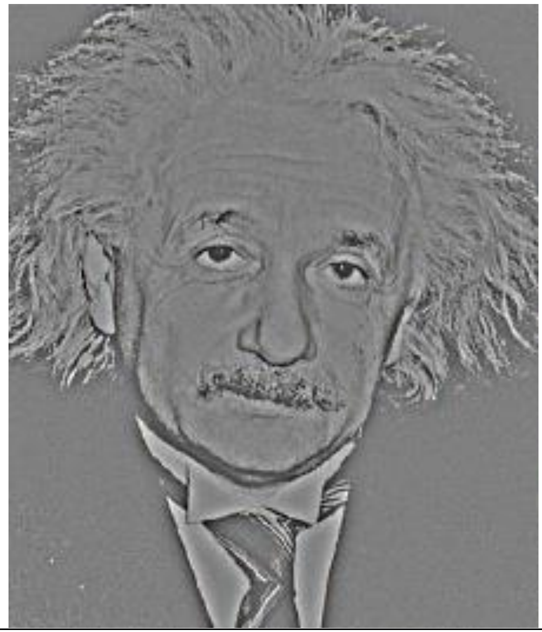
Table 1. Hybrid Image of Cat (LF) and Dog (HF) | cut-off = 7 & filter size = 29

Original image1	**Original image2**
	
Low-frequency image1	**High-frequency image2**
	

Hybrid_image = low-frequency image1 + high-frequency image2

Table 2. Hybrid Image of Marilyn (LF) and Einstein (HF) | cut-off = 3 & filter size = 29

Original image1	**Original image2**
	
Low-frequency image1	**High-frequency image2**

Hybrid_image = low-frequency image1 + high-frequency image2

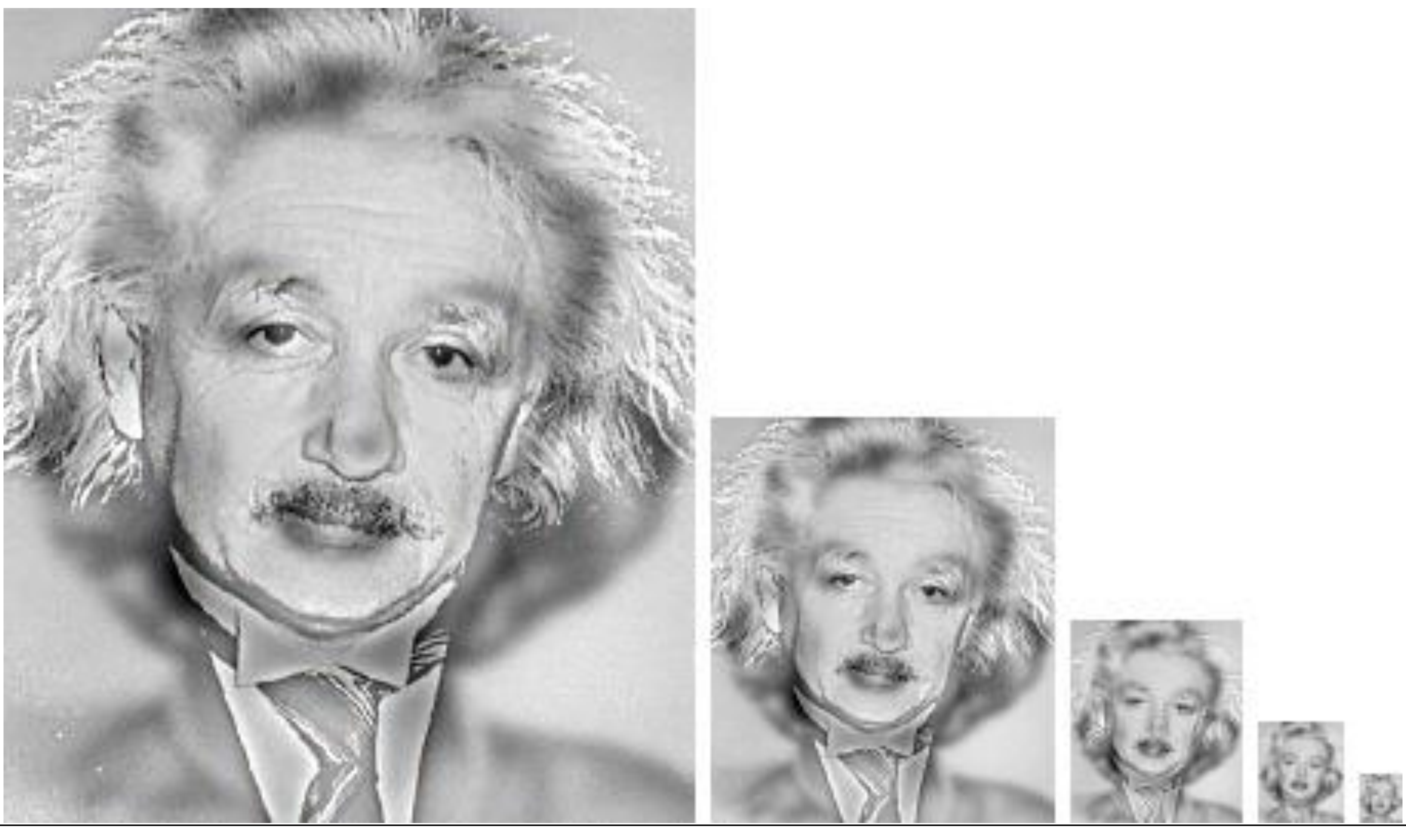


Table 3. Hybrid Image of Submarine (LF) and Fish (HF) | cut-off = 2 & filter size = 29

Original image1	Original image2

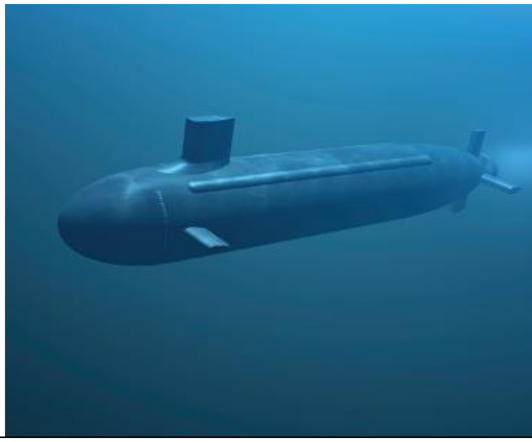
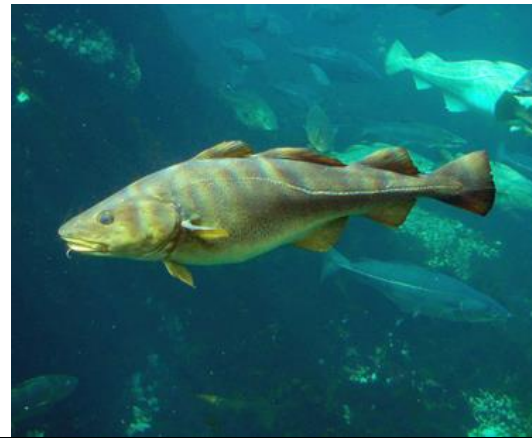
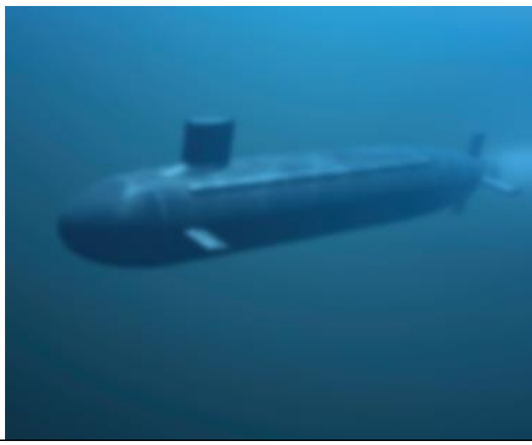
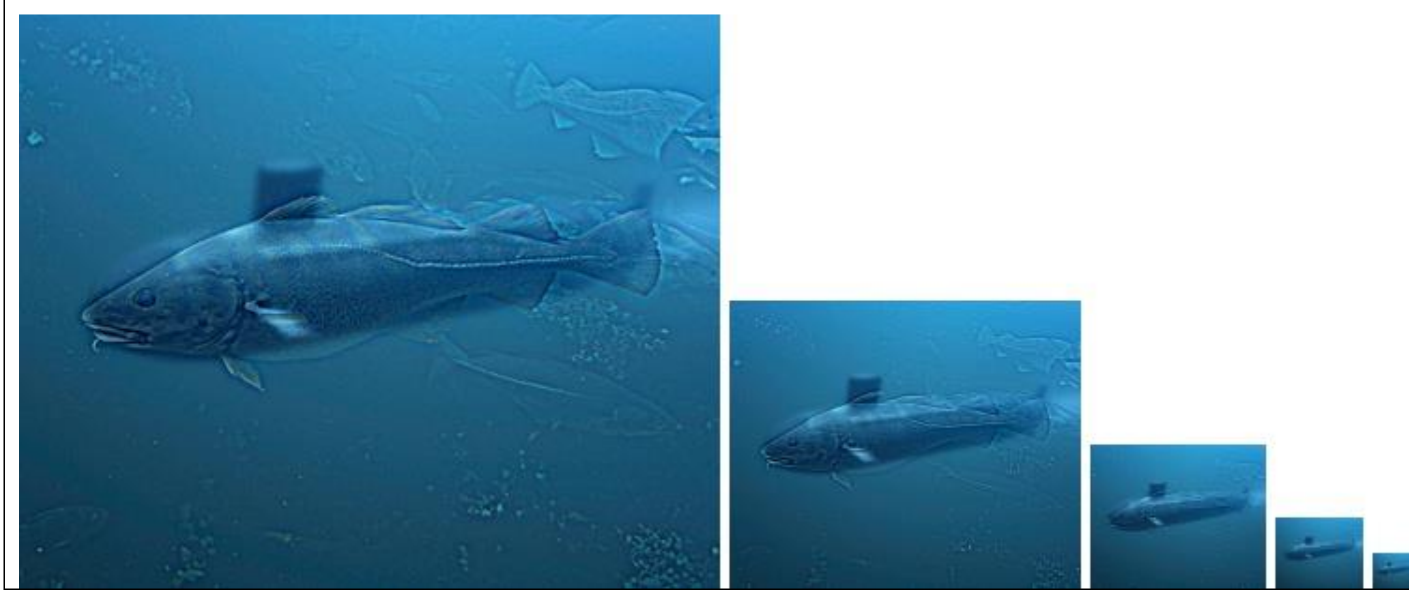
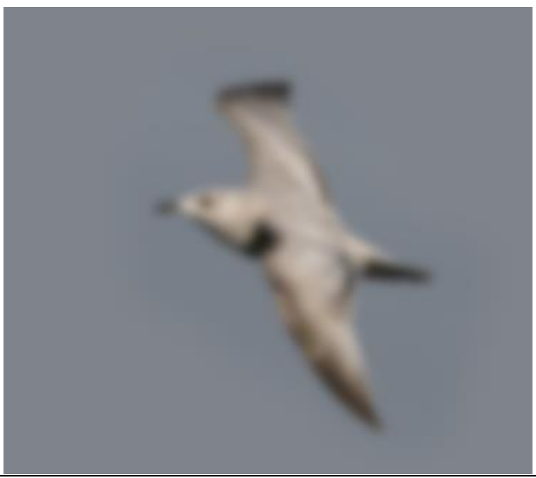
	
Low-frequency image1	**High-frequency image2**
	
Hybrid_image = low-frequency image1 + high-frequency image2	
	

Table 4. Hybrid Image of Bird (LF) and Plane (HF) | cut-off = 4 & filter size = 29

Original image1	**Original image2**
	
Low-frequency image1	**High-frequency image2**
	

Hybrid_image = low-frequency image1 + high-frequency image2

Table 5. Hybrid Image of Motorcycle (LF) and Bicycle (HF) | cut-off = 5 & filter size = 29

Original image1	Original image2
	
Low-frequency image1	**High-frequency image2**
	
Hybrid_image = low-frequency image1 + high-frequency image2	
	

Remarks on Hybrid Images Using Provided Data and Tuned Parameters

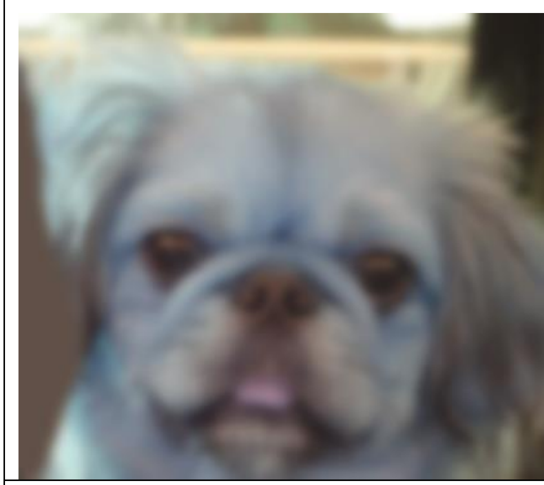
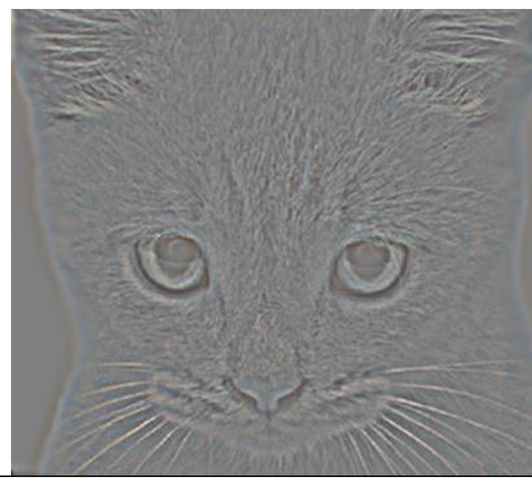
- Hybrid images are obtained by blending the low-frequency and high frequency images;

- Both combinations were tried and the more interesting one was chosen;
- While keeping the filter size constantly at 29, cut-off frequencies were tuned to balance the dominance of low-frequency and high-frequency images.

3.2. Effect of Cut-off Frequency on Hybrid Images

In order to analyse the effect of cut-off frequency on the resulting hybrid image, 5 different cut-off frequency were compared (3, 4, 7, 12, 14). Among them, the default value 7, and values 4 and 12 were chosen for illustration. The resulting low-frequency & high-frequency images as well as hybrid images are presented in tables 6 to 8.

Table 6. Effect of cut-off frequency on low-frequency & high-frequency images (Constant filter size = 29)

Low-frequency image1	High-frequency image2
Cut-off frequency = 4	
	
Cut-off frequency = 7	

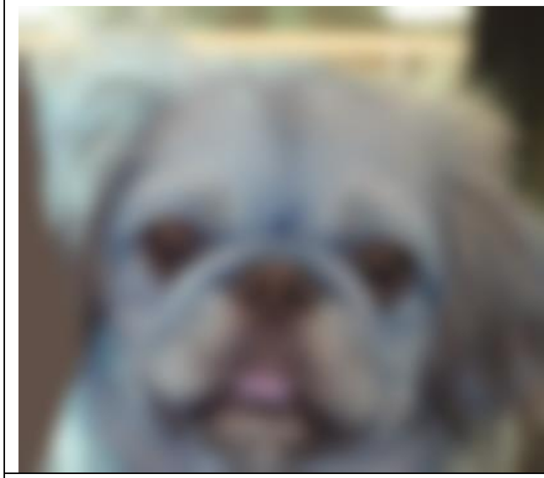
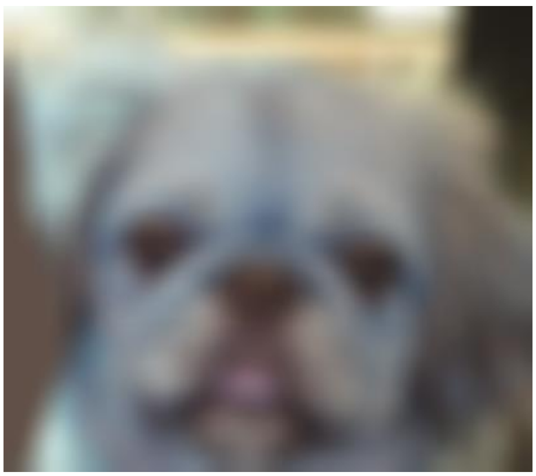
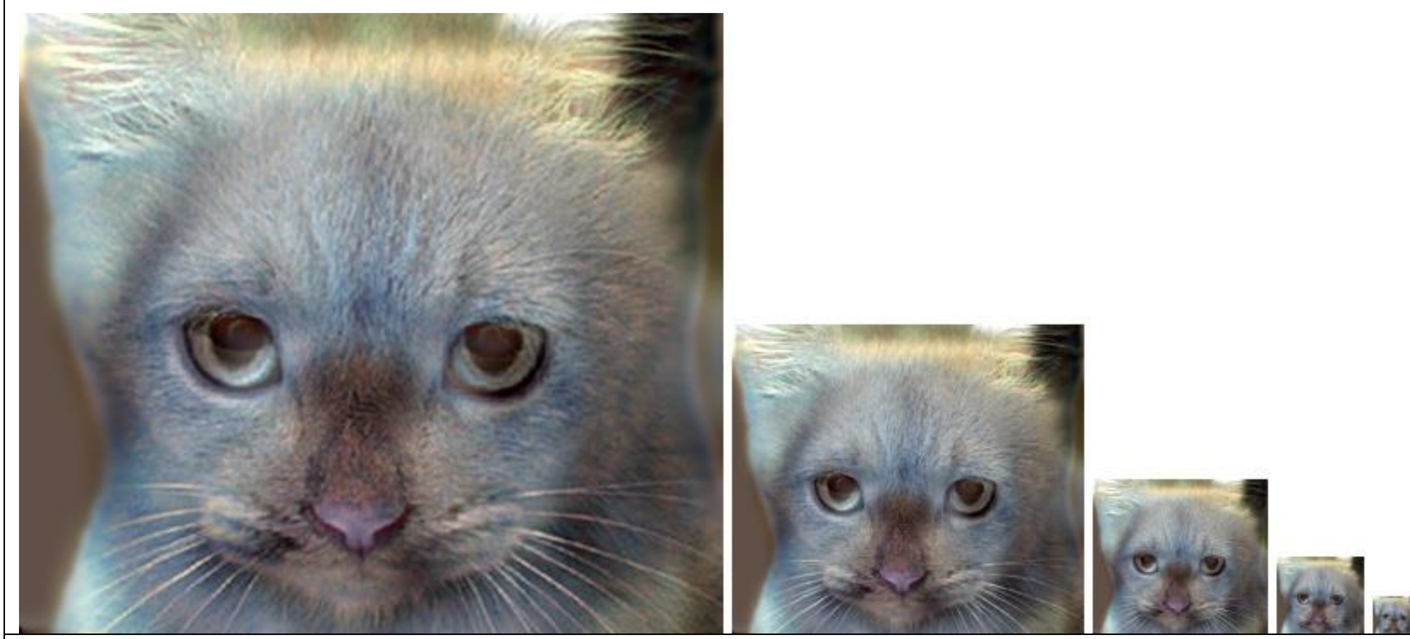
	
Cut-off frequency = 12	
	

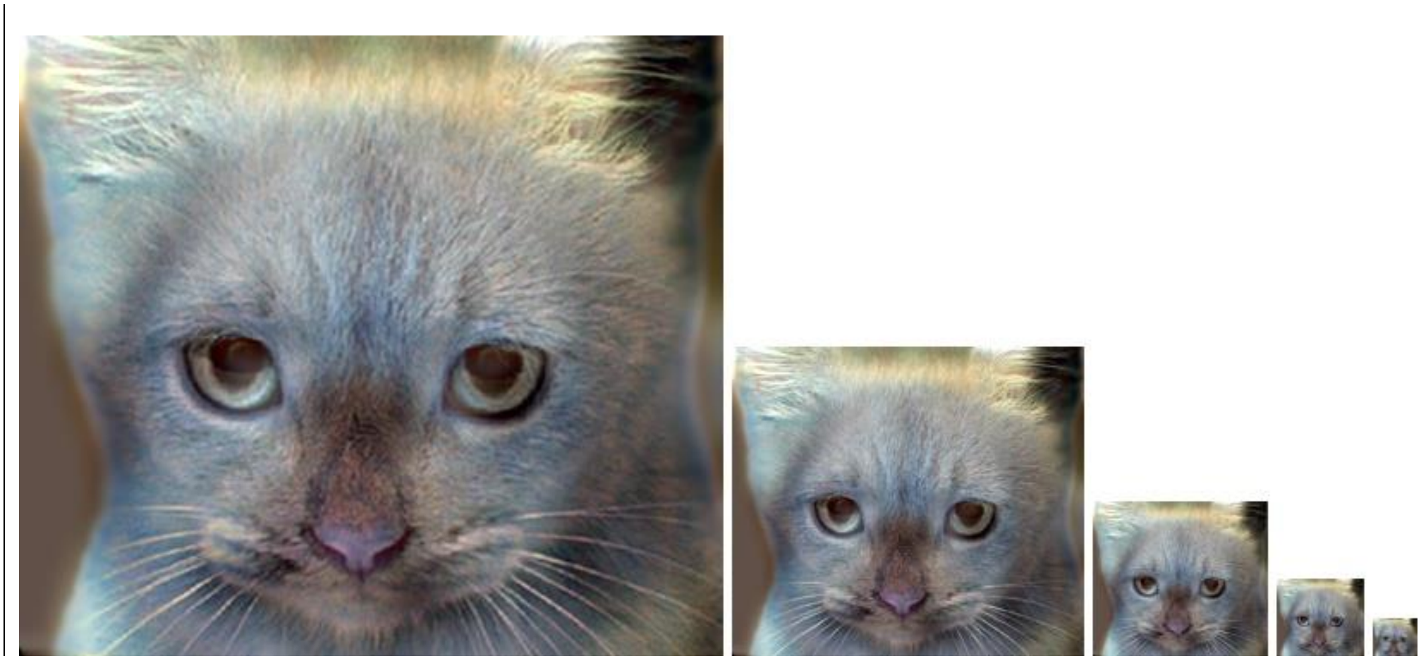
Table 7. Effect of cut-off frequency on hybrid images (Constant filter size = 29)

Hybrid_image = low-frequency image1 + high-frequency image2 **Image distance code from left to right:** **A (the closest), B, C, D, E (the most far)**
Cut-off frequency = 4

Cut-off frequency = 7



Cut-off frequency = 12



Remarks on the effect of cut-off frequency:

- As we reduce cut-off frequency from 7 to 4, the blurring effect on the low-frequency image (dog) is reduced as well. In other words, the low- frequency image has higher frequencies and will be more dominant in the hybrid image. As a result of this, by reducing the cut-off frequency, the low-frequency image (dog) will become visible from closer distance
- Similarly, increasing cut-off frequency from 7 to 12 leads to more blurring effect (lower frequencies) on the low-frequency image (dog). Therefore, the high-frequency image will become more dominant in the hybrid image and it takes more distance to perceive the low-frequency image;
- In brief, the lower the cut-off frequency, the more dominant the low-frequency image (dog) would be in the final hybrid image;
- The effect of cut-off on the low/high frequency perception is also summarized in table 8.

Table 8. Effect of cut-off frequency on the low/high frequency perception

	Image perceived at distances A (the closest), B, C, D, E (the most far)				
Cut-off freq.	**Distance A**	**Distance B**	**Distance C**	**Distance D**	**Distance E**
4	High freq. Cat	High & Low freq. Cat & Dog	Low freq. Dog	Low freq. Dog	Low freq. Dog

7	High freq. Cat	High freq. Cat	High & Low freq. Cat & Dog	Low freq. Dog	Low freq. Dog
12	High freq. Cat	High freq. Cat	High freq. Cat	Low freq. Dog	Low freq. Dog

Extra Works

Following tasks were done outside assignment requirements:

- Tuning cut-off frequency presented along the results in section "3.1. Hybrid Images Using Provided Data & Tuned Cut-off Frequency".
- Investigating the effect of cut-off frequency on hybrid images and discussion of results are presented in section "3.2. Effect of Cut-off Frequency on Hybrid Images"

Additional Notes

A few points might be worth mentioning:

- Results of section "3.1. Hybrid Images Using Provided Data & Tuned Cut-off Frequency" are generated using the notebook "proj1_5RegularImages.ipynb";
- Results of section "3.2. Effect of Cut-off Frequency on Hybrid Images" are generated using the notebook "proj1_CutoffEffect.ipynb".
- Images saved by the provided "save_image" function had some noises so I modified the function. The original "utils.py" is not change and the used code is saved as "utils_R1.py".

References

[1] Assignment 01 description by Dr. Kin-Choong Yow
[2] https://en.wikipedia.org/wiki/Gaussian_blur
[3] https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html#gac05a120c1ae92a6060dd0db190a61afa
[4] https://theailearner.com/tag/cv2-getgaussiankernel/