Project 2

Maryam Darei **Computational Photography**
Dr. Pless

Hybrid Images

Background

A hybrid image is an image that is perceived in one of two different ways, depending on viewing distance, based on the way humans process visual input. These images are composite images that blend two pictures; one picture is filtered using a high pass, and the other is filtered with a low pass. When the two filtered images are layered over each other, the human eye will see one or the other, depending on the viewing distance.

Data Description:

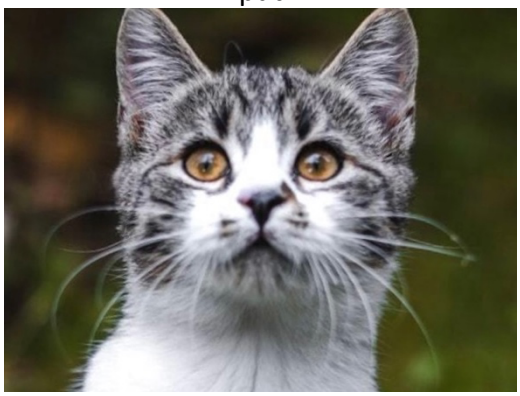
Data Type: Both RGB and grayscale images of any resolution are valid inputs

Data Characteristics:

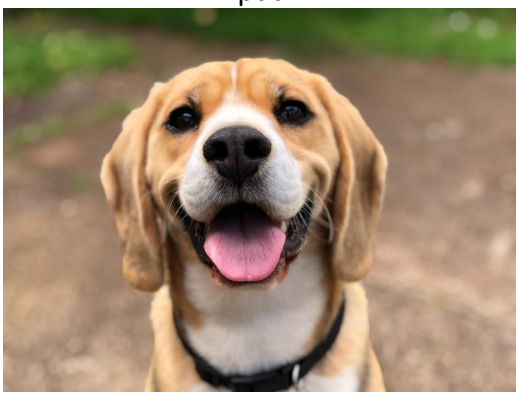
- We have two RGB images as an input which would have the potential to hybrid with each other
- Attribute breakdown: 2 quantitative input variables, and 1 quantitative output variable
- Missing Attribute Values: None
- The output after filtering should be the same size as the input image.

Sample of input and output:

Input 1



Input 2



Data pre-processing and Feature Engineering:

1- **Match the image's size:** We should match two input images' sizes before hybrid and create output.

2- **Alignment images:** The alignment of these images is very important for a better final result. If we want to use the faces photos, we can align them based on eyes location. It means we should their eyes are in the same place.

Filters characteristic:

We should have two filters: a lowpass filter and a high pass filter. Be careful that the image filter must be an odd-number width and height, to have an explicit center pixel.

For the low pass filter, we can use a standard 2D Gaussian filter or Blur filter and we make the high pass filter by subtracting the low pass filter from the original image.

A simple blur can be done using this kernel:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

And below matrix is the kernel for the Gaussian Blur:

$$\frac{1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix}$$

Code description:

I have written it in **MATLAB**.

As it is mentioned, we need two kernels; one lowpass filter and another one is high pass filter.

First attempt: I wrote functions with small differences for each of the low-pass and high-pass filters and tested them in separate codes. In program test1 I have used the functions lowpassfilter1 and lowpassfilter2 as a low pass filter for image smoothing(blurring) and highpassfilter1 and highpassfilter2 as a high pass filter.

lowpassfilter1: In this function, I used a simple 3X3 box blur kernel

lowpassfilter2: In this function, I used a 5X5 Gaussian blur kernel.

highpassfilter1: I use the Laplacian Filter which is a smoothing technique that is mainly used to detect image edges. For better output, I doubled the filtered image and added it to itself.

highpassfilter2: In this function, I created a high pass filter by subtracting the lowpass features from an original image.

Low-pass filters:

```
function [Blurimage] = lowpassfilter1(input_image)
% input_image : input mage that we want to extract lowpass
% BlurFilter is the 3X3 Box Blur kernel
% Blurimage : the function output

%lowpass filter // Box Blur kernel:
BlurFilter =  [[1 / 9, 1 / 9, 1 / 9],
               [1 / 9, 1 / 9, 1 / 9],
               [1 / 9, 1 / 9, 1 / 9]];

Blurimage = imfilter(input_image,BlurFilter);


end
```

```
function [GaussianBlurimage] = lowpassfilter2(input_image)
% input_image : input mage that we want to extract its lowpass
% BlurFilter is the 5X5 Gaussian kernel
% GaussianBlurimage : the function output

%lowpass filter //Gaussian Blur:
GaussianBlurfilter=[[1 / 256, 4  / 256,  6 / 256,  4 / 256, 1 / 256],
                    [4 / 256, 16 / 256, 24 / 256, 16 / 256, 4 / 256],
                    [6 / 256, 24 / 256, 36 / 256, 24 / 256, 6 / 256],
                    [4 / 256, 16 / 256, 24 / 256, 16 / 256, 4 / 256],
                    [1 / 256, 4  / 256,  6 / 256,  4 / 256, 1 / 256]];

GaussianBlurimage = imfilter(input_image,GaussianBlurfilter);
```

High-pass filters:

```
function [Highpassimage] = highpassfilter1(input_image)
% input_image : input mage that we want to extract its highpass features
% Highpass filter is the 3X3 Laplace edge detector kernel
% Highpassimage : the function output

%highpass filter // Laplacian Filter
IHigh_filtered = [-1 -1 -1 ,-1 8 -1, -1 -1 -1];
Highpassimage1 = imfilter(input_image,IHigh_filtered);

% For a better output, I doubled the filtered image and added it to itself
for colorI = 1:3
      Highpassimage(:,:,colorI) = 2*Highpassimage1(:,:,colorI) + 2*Highpassimage1(:,:,colorI);
end

end
```

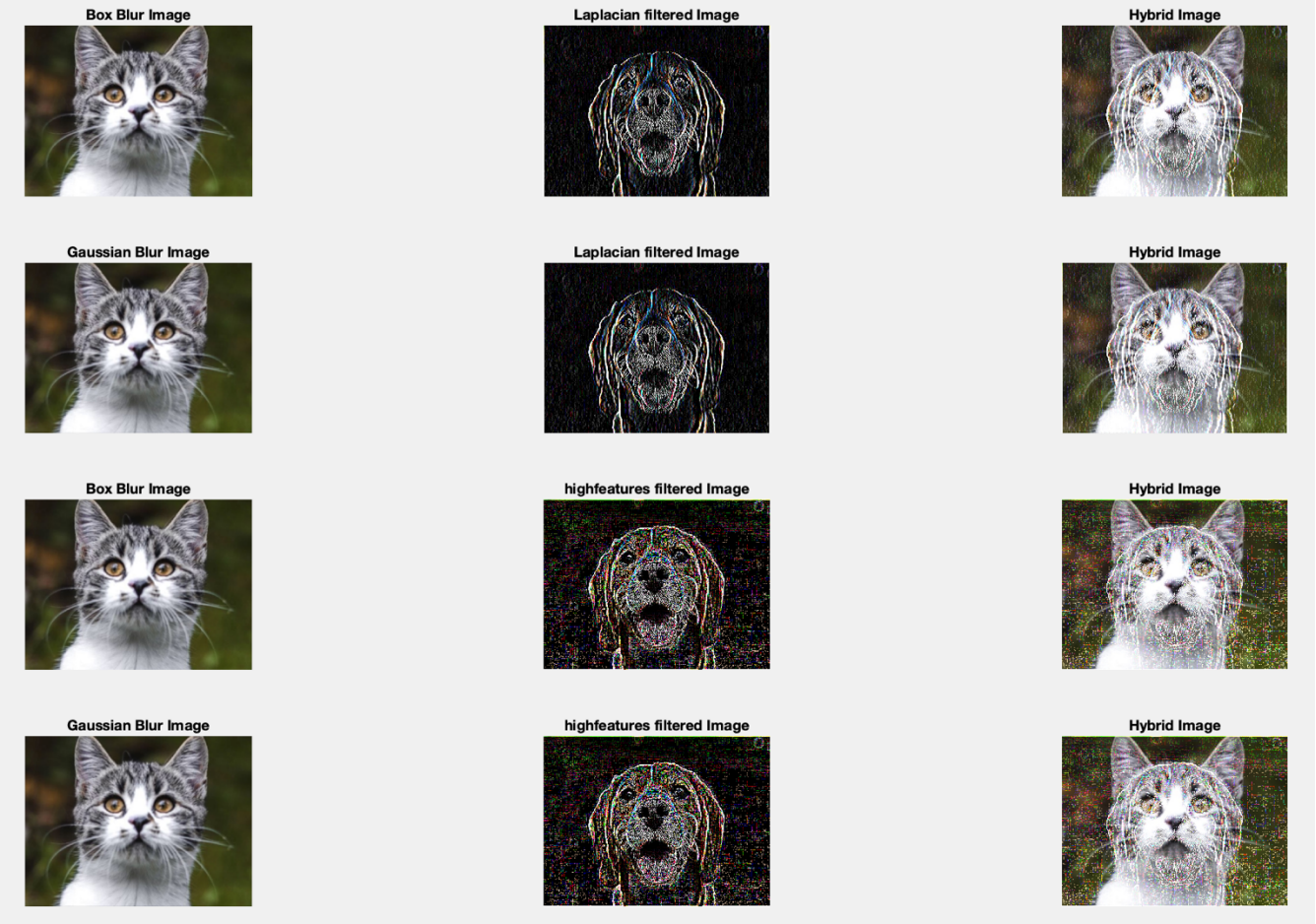
```
function [Highpassimage] = highpassfilter2(input_image)
% input_image : input mage that we want to extract its highpass features
% Highpass filter created by subtract the lowpass features form orginal image
% Highpassimage : the function output

%highpass filter
Ihigh_filtered = input_image - lowpassfilter2(input_image);

Highpassimage = Ihigh_filtered .* input_image;

end
```

I developed a function to hybrid 3 channels of two filtered images and created a hybrid image. In this example, all images are RGB format.

Hybrid function:

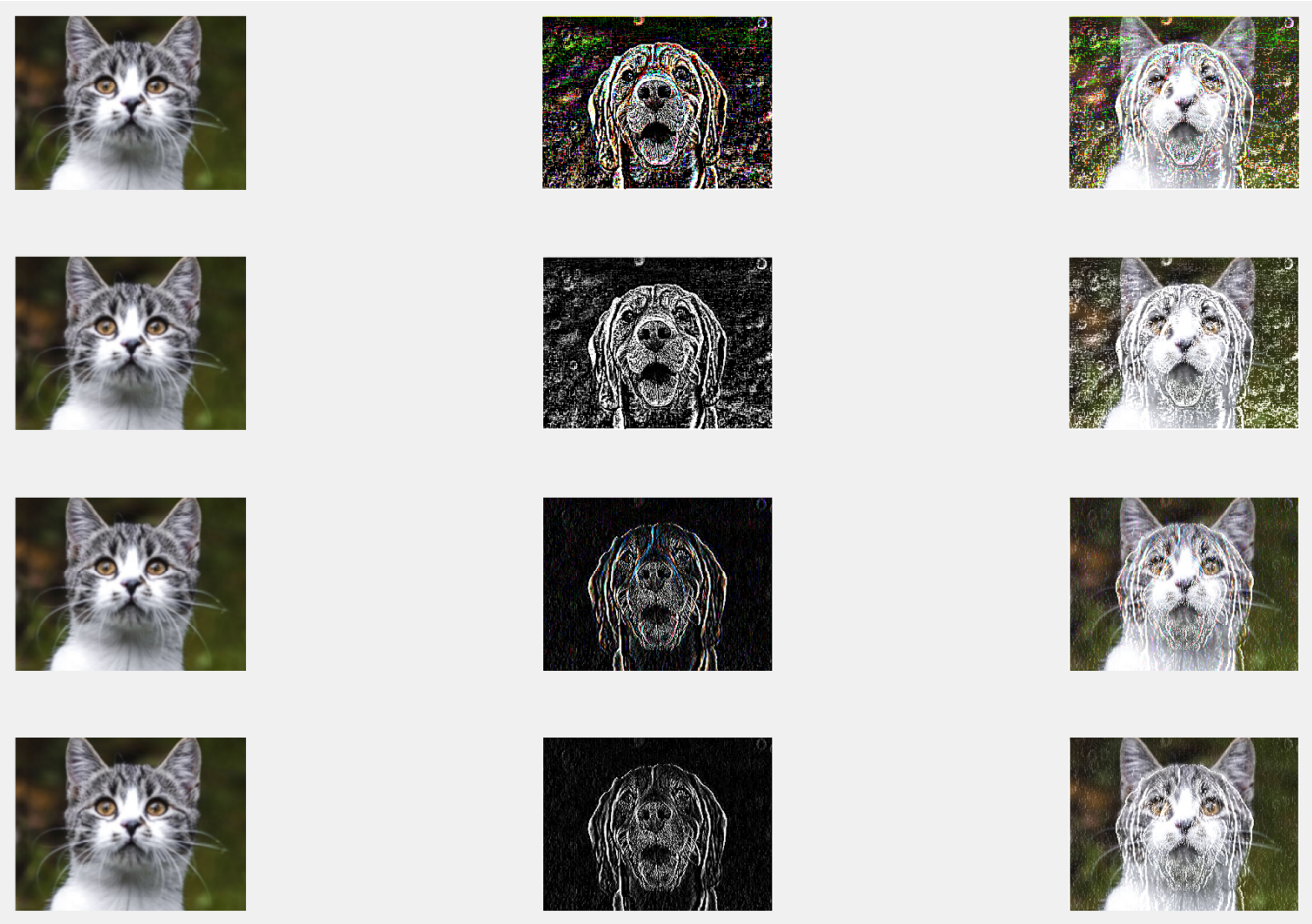
```
function [output_image] = hybridfunc(image1, image2)
% image1, image2 : input mages that we want to merge with
% output_image : the function output

% merge two images
 for colorI = 1:3
      output_image(:,:,colorI) = image1(:,:,colorI) + image2(:,:,colorI);
 end

end
```

Here is the result of test1:



Second attempt: In the second approach, test2, I used a built-in low pass filter, imgaussfilt(A, sigma) that filters image A with a 2-D Gaussian smoothing kernel with standard deviation specified by sigma. For the high pass filter, I used lowpass filter1 and also another function that calculated a high pass filter by subtracting the lowpass features(imgaussfilt(A, sigma)) from an original image. There are two differences with the first approach. First, it is possible to control the amount of blur with the possibility of changing the sigma. I made the second difference by converting the RGB high pass filtered images to Gray and then merging with the blur images.

Here is the result of test2:



3rd attempt: It has the best result. In this approach, I wrote a low pass filter that converts the input image to the frequency domain by using Fourier transform and then using a 2-D Gaussian smoothing kernel to extract the low-frequency features. For the high pass filter, I also used the Fourier transform. In the main program, I merged both low pass and high pass filtered images in the frequency domain and created a hybrid image in frequency format. Then for displaying the result I convert it to RGB image format.

Low-pass filters:

```
function [output_image, IL] = lowpassfilter3(input_image, sigma)
% input_image : input mage that we want to extract lowpass features
% Sigma: standard deviation
% output_image : the function output in image format
% IL : the function output(image) in frequency domain

   IF = fftshift(fft2(double(input_image)));
   [m,n,z] = size(input_image);
   h = fspecial('gaussian', [m n], sigma);
   h = h./max(max(h));
   IL=IF.*h;
   output_image = uint8(real(ifft2(ifftshift(IL))));

end
```

High-pass filters:

```
function [output_image, IH] = highpassfilter3(input_image, sigma)
% input_image : input mage that we want to extract highpass features
% Sigma: standard deviation
% output_image : the function output in image format
% IH : the function output(image) in frequency domain

   IF = fftshift(fft2(double(input_image)));
   [m,n,z] = size(input_image);
   h = fspecial('gaussian', [m n], sigma);
   h = h./max(max(h));
   IH=IF.*(1-h);
   output_image = uint8(real(ifft2(ifftshift(IH))));

end
```

Here is the result of test3:

Result Analyze:

It appears the 3rd approach is better than the others. In this approach, I used the frequency concept.

Bells & Whistles (Extra Points):

- **using color to enhance the effect:** In the first approach, I try to enhance the high pass filtered image by adding it to itself. In the second approach, I used the grayscale image for the high pass filtered image. It seems we have better quality in the background. In 3rd approach, I had to transform the result to the RGB image from the frequency domain before converting the high pass filtered image to grayscale and not really good result.

- **three-part hybrid image:** Another kind of hybrid image can be presented differently from three different distances; far, middle, and near. For this concept, we need to consider the image in the middle distance that is not viewed clear from near and far. For reaching out to this matter, we use three different frequency filters, each one allows different frequency bands (the low, middle, and high) to pass. We proposed low and high pass filters previously. we need a special bandpass filter (MF filter) for extracting frequencies to be seen from the middle distance. We can use lower and upper bound for the MF filters.

$$M_{MF}(\mathbf{p})=(k_L(1-l(\mathbf{p}))+k_U l(\mathbf{p}))(1-m(\mathbf{p}))+m(\mathbf{p})$$

Here, l(p) and m(p) are the pixel values of the LF map and MF map accordingly. K_L and k_U give the lower and upper bound of the local frequency map K_L, $k_U \in [0, 1]$ when m(p) is zero (provided that $l(p) \in [0, 1]$ and $m(p) \in [0, 1]$).

Another Bells & Whistles:

1- One of the things that can be done in this concept is displaying different emotions in faces. One of the high or low spatial frequencies corresponds to faces with ”sad” expressions, and another spatial frequency corresponds to the same faces with ”happy” or ”surprise” emotions. It means we can have a hybrid face that shows different emotions at different distances.

2- Another thing is related to color. Color offers a very strong grouping sign that can be used to create more convincing illusions