Applied Computer Vision (CS-696)
Homework Assignment 2

Name – Dhaval Harish Sharma
Red ID – 824654344

Solution:

Convolution

For more useful and interesting operations, we must consider not only the pixel but also the neighboring pixels. The output of the operation should depend on the pixel as well as its surrounding region. The convolution operation is the basis of various neighborhood processing techniques. Convolution requires defining a matrix whose entries determine the type of neighborhood operation. This matrix is also called by various names such as filter, window, mask or kernel.

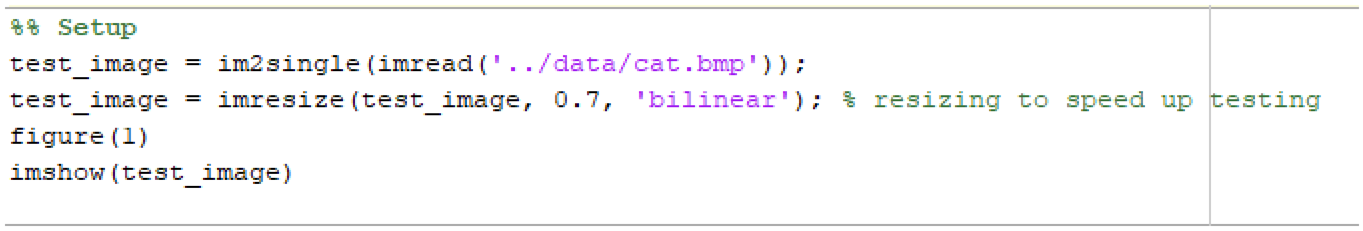
[my_imfilter.m]

Now, talking about the my_imfilter function in the file my_imfilter.m, it takes two parameters as input. One is the image itself and the second one is the filter. It provides an output in the form of image with the filter applied on the input image. The code begins by getting the dimensions of the input image. It initializes the output image in the form of a matrix. If the image is grayscale, then we only need to apply the filter once otherwise we need to apply the filter thrice in case of RGB images having three dimensions.

There is a out_dim_filter function which takes the image and filter as input and applies the filter on the image. It begins by getting the number of pixels in the x and y direction. Then, it finds the number of pixels to pad in the input image to prevent the convolution failure on the edges of the image. It pads the input image with the necessary black pixels. The process of convolution begins by getting the equal number of pixels as filter from the image, multiplying the corresponding pixels with the filter pixels, adding the result and putting it in the output image.

[proj2_test_filtering.m]

There's a file called proj2_test_filtering.m which contains test cases to test the working of the my_imfilter function. The first thing to check the my_imfilter is to import images. The following code inputs the image and converts it to corresponding matrix. It then, plots the image in the figure window.



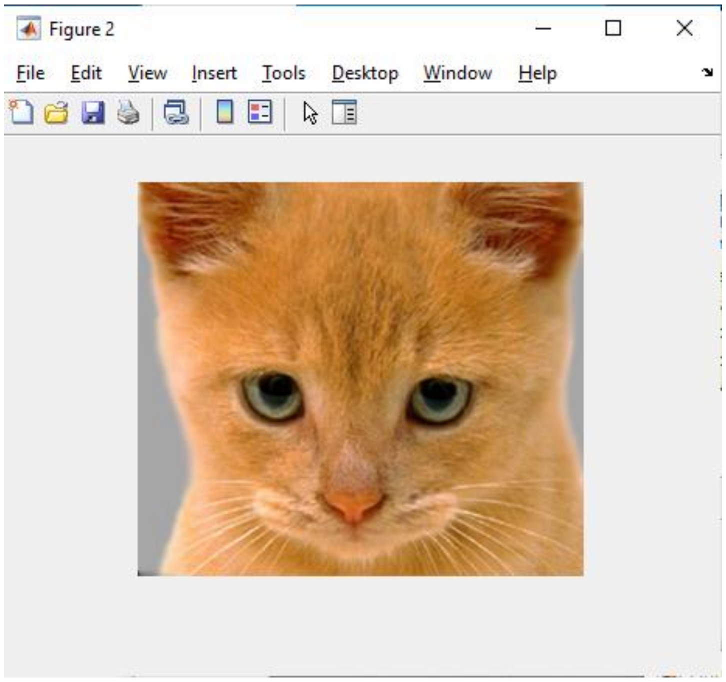
```
%% Setup
test_image = im2single(imread('../data/cat.bmp'));
test_image = imresize(test_image, 0.7, 'bilinear'); % resizing to speed up testing
figure(1)
imshow(test_image)
```

The next step is to check different filters on this image. The following are the respective codes and outputs for different filters applied on the cat image with the help of my_imfilter function:

(1) Identity Filter:

```
%% Identify filter
% This filter should do nothing regardless of the padding method you use.
identity_filter = [0 0 0; 0 1 0; 0 0 0];

identity_image  = my_imfilter(test_image, identity_filter);

figure(2); imshow(identity_image);
imwrite(identity_image, 'identity_image.jpg', 'quality', 95);
```

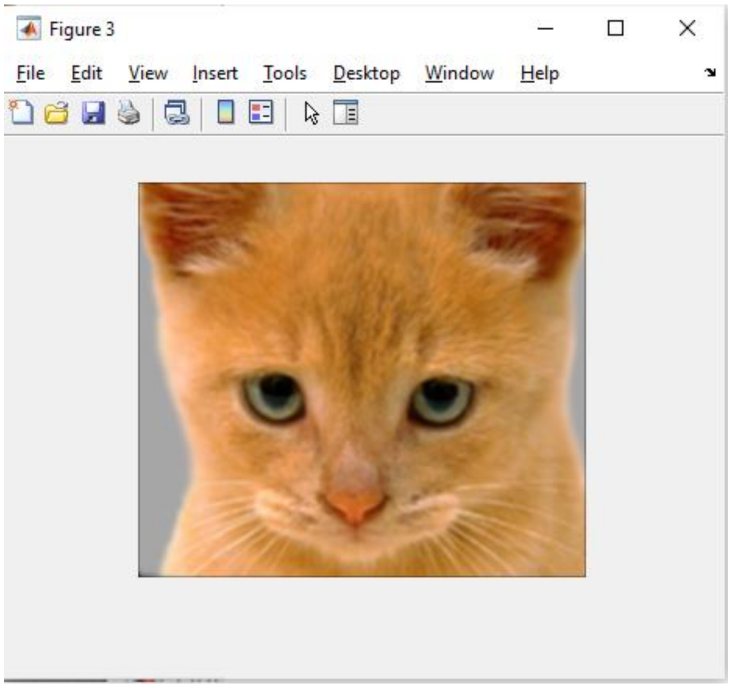
Output:



(2) Box Filter:

```
%% Small blur with a box filter
% This filter should remove some high frequencies
blur_filter = [1 1 1; 1 1 1; 1 1 1];
blur_filter = blur_filter / sum(sum(blur_filter)); % making the filter sum to 1

blur_image = my_imfilter(test_image, blur_filter);

figure(3); imshow(blur_image);
imwrite(blur_image, 'blur_image.jpg', 'quality', 95);
```

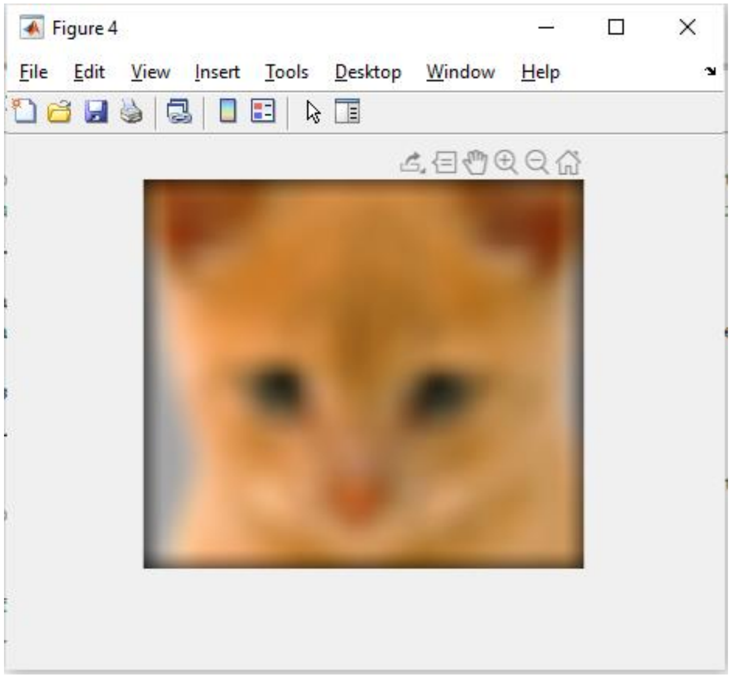
Output:



(3) Gaussian Blur:

```
%% Large blur
% This blur would be slow to do directly, so we instead use the fact that
% Gaussian blurs are separable and blur sequentially in each direction.
large_1d_blur_filter = fspecial('Gaussian', [25 1], 10);

large_blur_image = my_imfilter(test_image, large_1d_blur_filter);
large_blur_image = my_imfilter(large_blur_image, large_1d_blur_filter'); % notice the ' operator which transposes the filte

figure(4); imshow(large_blur_image);
imwrite(large_blur_image, 'large_blur_image.jpg', 'quality', 95);

% If you want to see how slow this would be to do naively, try out this
% equivalent operation:
% tic
% tic and toc run a timer and then print the elapsed time
% large_blur_filter = fspecial('Gaussian', [25 25], 10);
% large_blur_image = my_filter(test_image, large_blur_filter);
% toc
```

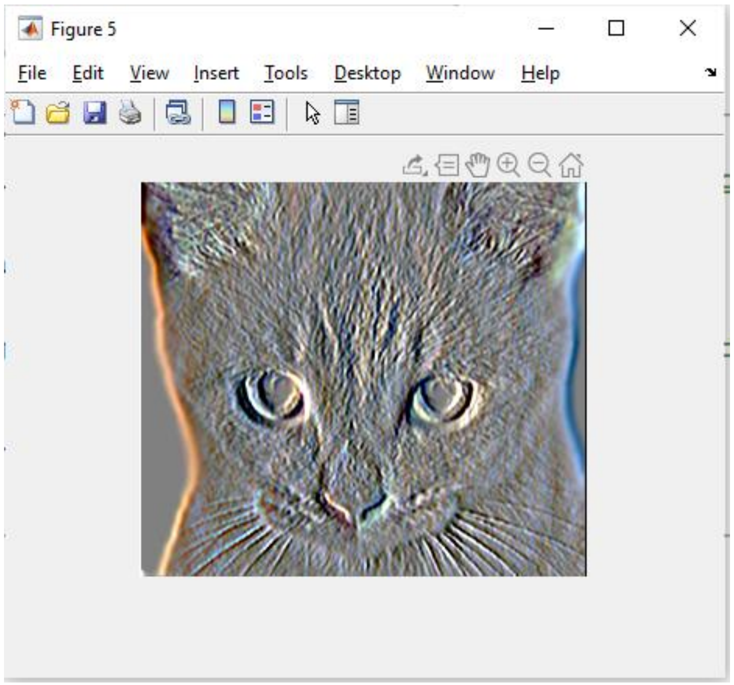
Output:



(4) Sobel Operator:

```
%% Oriented filter (Sobel Operator)
sobel_filter = [-1 0 1; -2 0 2; -1 0 1]; % should respond to horizontal gradients

sobel_image = my_imfilter(test_image, sobel_filter);

%0.5 added because the output image is centered around zero otherwise and mostly black
figure(5); imshow(sobel_image + 0.5);
imwrite(sobel_image + 0.5, 'sobel_image.jpg', 'quality', 95);
```

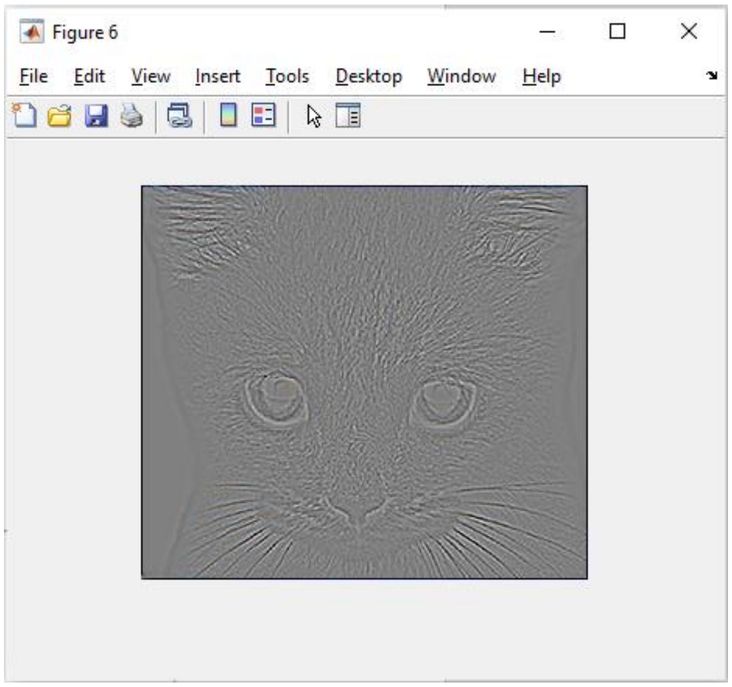
Output:



(5) Discrete Laplacian:

```
%% High pass filter (Discrete Laplacian)
laplacian_filter = [0 1 0; 1 -4 1; 0 1 0];

laplacian_image = my_imfilter(test_image, laplacian_filter);

% 0.5 added because the output image is centered around zero otherwise and mostly black
figure(6); imshow(laplacian_image + 0.5);
imwrite(laplacian_image + 0.5, 'laplacian_image.jpg', 'quality', 95);
```

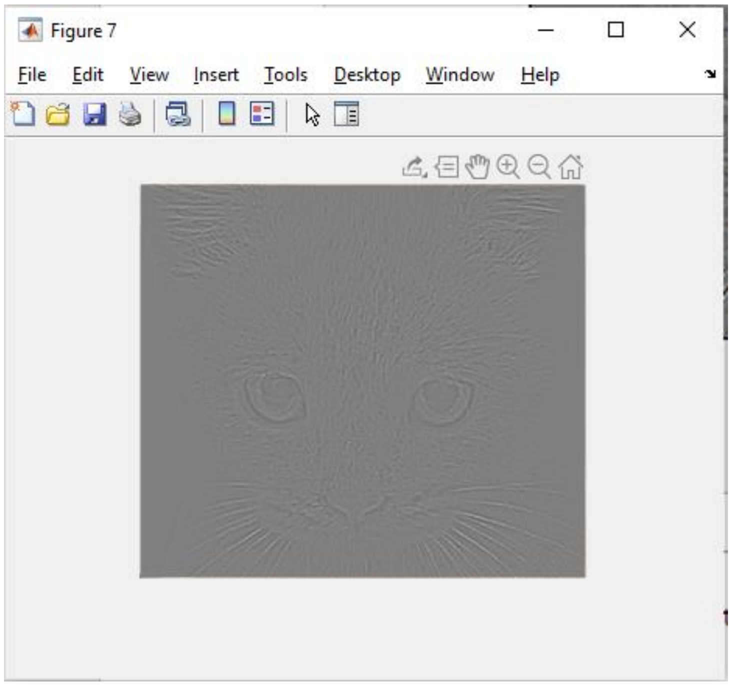
Output:



(6) High Pass Filter:

```
%% High pass "filter" alternative
high_pass_image = test_image - blur_image; % simply subtract the low frequency content

figure(7); imshow(high_pass_image + 0.5);
imwrite(high_pass_image + 0.5, 'high_pass_image.jpg', 'quality', 95);
```

Output:



The algorithm is a pretty straightforward algorithm for performing convolution operations. To figure out this algorithm, I learnt the basics of the neighborhood operations. Then, I followed the exact same steps as mentioned in the slides to figure out the application of the algorithm. The challenging part for this algorithm was to multiply the corresponding filter values and the image values.

Hybrid Images:
A hybrid image is the sum of a low-pass filtered version of the one image and a high-pass filtered version of a second image. There is a free parameter, which can be tuned for each image pair, which controls how much high frequency to remove from the first image and how much low frequency to leave in the second image. This is called the "cutoff frequency".

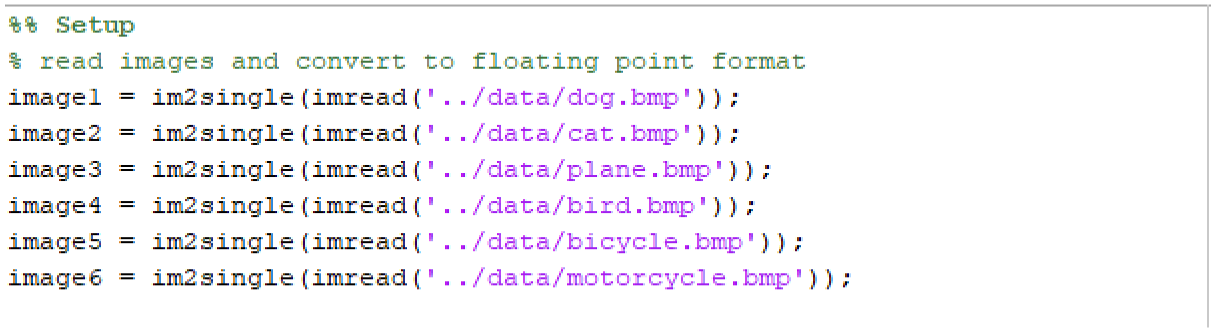
[proj2.m]
The proj2.m file takes multiple images as input for its operation. It then, converts them into the corresponding matrices containing the pixel values. The filtering and hybrid image generation step follows next. It initializes the cutoff frequency variable for the hybrid image construction. There is a filter declared using in-built fspecial function which initializes a gaussian filter with the necessary cutoff frequency. We then form the low frequency images using the my_imfilter function and the necessary images.

There are three methods defined for forming high frequency images. The first method generates low frequency images using the same process as before and then subtracts the result from the original image to get the high frequency images. The second method performs the same process but with two different cutoff frequencies for the high as well as low frequency images. The third method is a bit different from the first and the second method. It uses Laplacian filter for generating the high frequency images.

We then form the hybrid images by adding the corresponding low frequency and high frequency images. The hybrid images along with the low and high frequency images are then displayed on the screen using figures.

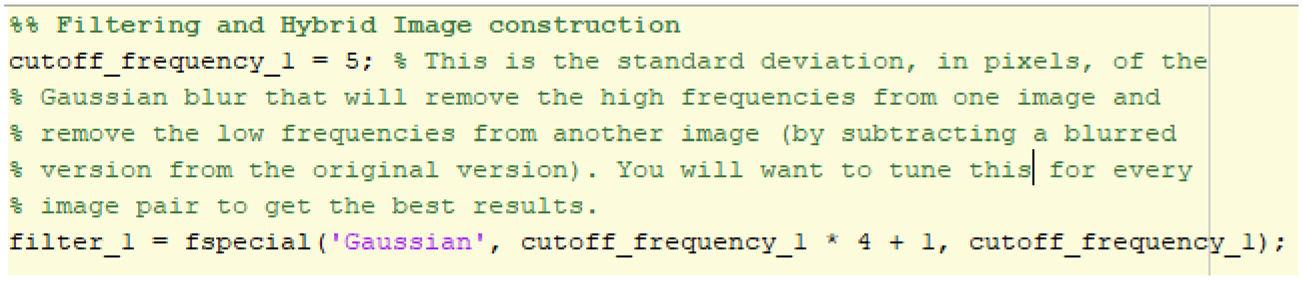
The respective codes and the results are as shown below:

This code inputs the image and converts it to corresponding matrix containing pixel values.



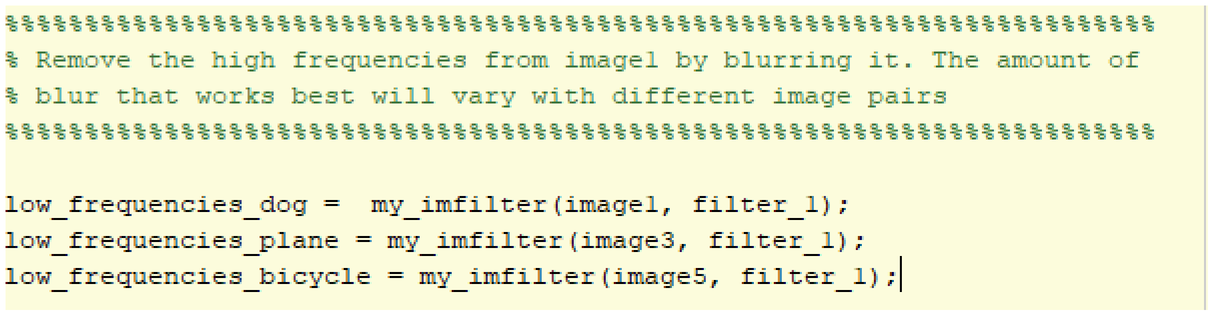
```
%% Setup
% read images and convert to floating point format
image1 = im2single(imread('../data/dog.bmp'));
image2 = im2single(imread('../data/cat.bmp'));
image3 = im2single(imread('../data/plane.bmp'));
image4 = im2single(imread('../data/bird.bmp'));
image5 = im2single(imread('../data/bicycle.bmp'));
image6 = im2single(imread('../data/motorcycle.bmp'));
```

The filter is generated with the help of fspecial and the cutoff frequency is decided.



```
%% Filtering and Hybrid Image construction
cutoff_frequency_1 = 5; % This is the standard deviation, in pixels, of the
% Gaussian blur that will remove the high frequencies from one image and
% remove the low frequencies from another image (by subtracting a blurred
% version from the original version). You will want to tune this for every
% image pair to get the best results.
filter_1 = fspecial('Gaussian', cutoff_frequency_1 * 4 + 1, cutoff_frequency_1);
```

The low frequency images are generated.



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Remove the high frequencies from image1 by blurring it. The amount of
% blur that works best will vary with different image pairs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

low_frequencies_dog =  my_imfilter(image1, filter_1);
low_frequencies_plane = my_imfilter(image3, filter_1);
low_frequencies_bicycle = my_imfilter(image5, filter_1);
```

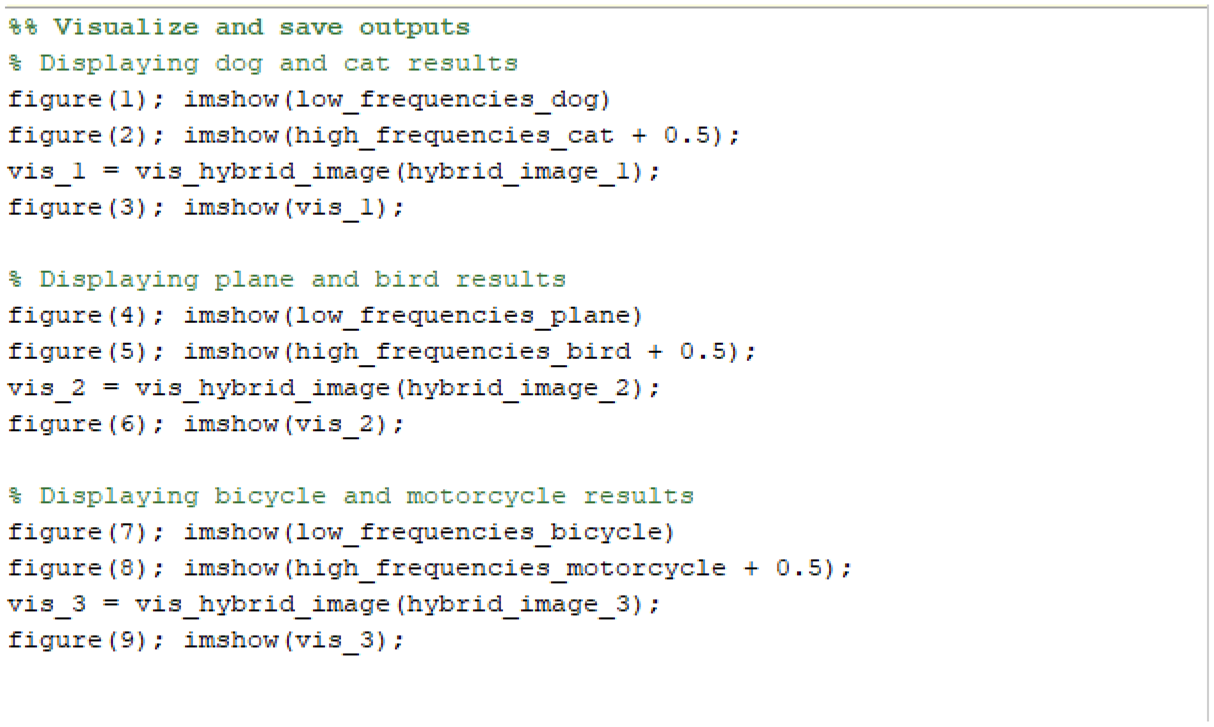
The high frequency images are generated.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Remove the low frequencies from image2. The easiest way to do this is to
% subtract a blurred version of image2 from the original version of image2.
% This will give you an image centered at zero with negative values.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Method 1 (Using single cutoff frequency)
low_pass_image_cat =  my_imfilter(image2, filter_1);
high_frequencies_cat = image2 - low_pass_image_cat;
low_pass_image_bird = my_imfilter(image4, filter_1);
high_frequencies_bird = image4 - low_pass_image_bird;
low_pass_image_motorcycle = my_imfilter(image6, filter_1);
high_frequencies_motorcycle = image6 - low_pass_image_motorcycle;


% Method 2 (Using two cutoff frequencies)
% cutoff_frequency_2 = 5;
% filter_2 = fspecial('Gaussian', cutoff_frequency_2 * 4 + 1, cutoff_frequency_2);
% low_pass_image =  my_imfilter(image2, filter_2);
% high_frequencies = image2 - low_pass_image;

% Method 3 (Using laplacian filter instead of gaussian)
% laplacian_filter = [0 1 0; 1 -4 1; 0 1 0];
% high_frequencies = my_imfilter(image2, laplacian_filter);
```

The hybrid images are formed.

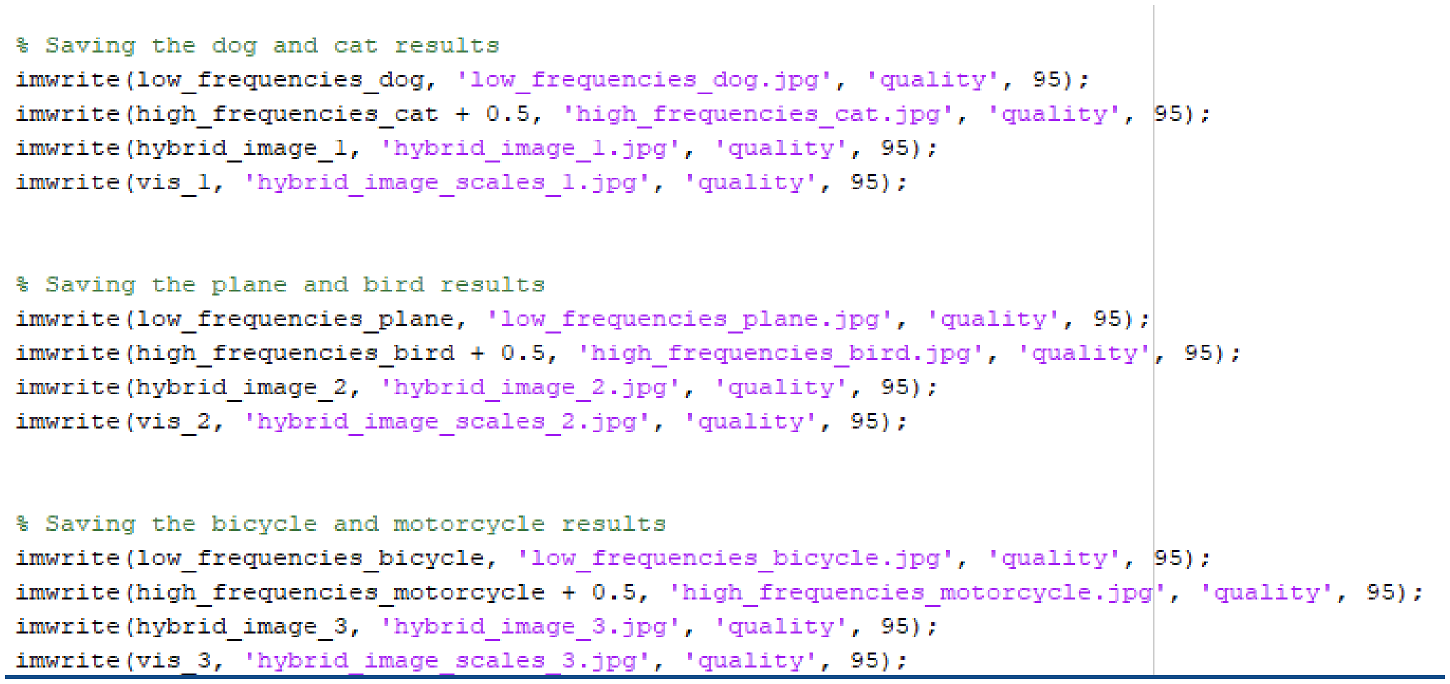
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Combine the high frequencies and low frequencies
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

hybrid_image_1 = low_frequencies_dog + high_frequencies_cat;
hybrid_image_2 = low_frequencies_plane + high_frequencies_bird;
hybrid_image_3 = low_frequencies_bicycle + high_frequencies_motorcycle;
```

Displaying the results.



```
%% Visualize and save outputs
% Displaying dog and cat results
figure(1); imshow(low_frequencies_dog)
figure(2); imshow(high_frequencies_cat + 0.5);
vis_1 = vis_hybrid_image(hybrid_image_1);
figure(3); imshow(vis_1);

% Displaying plane and bird results
figure(4); imshow(low_frequencies_plane)
figure(5); imshow(high_frequencies_bird + 0.5);
vis_2 = vis_hybrid_image(hybrid_image_2);
figure(6); imshow(vis_2);

% Displaying bicycle and motorcycle results
figure(7); imshow(low_frequencies_bicycle)
figure(8); imshow(high_frequencies_motorcycle + 0.5);
vis_3 = vis_hybrid_image(hybrid_image_3);
figure(9); imshow(vis_3);
```

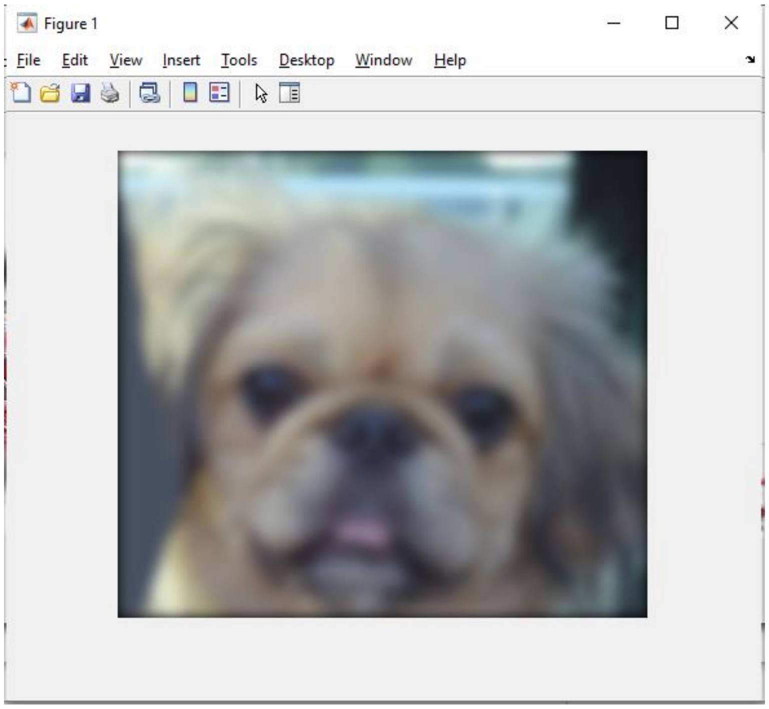
Saving the results.



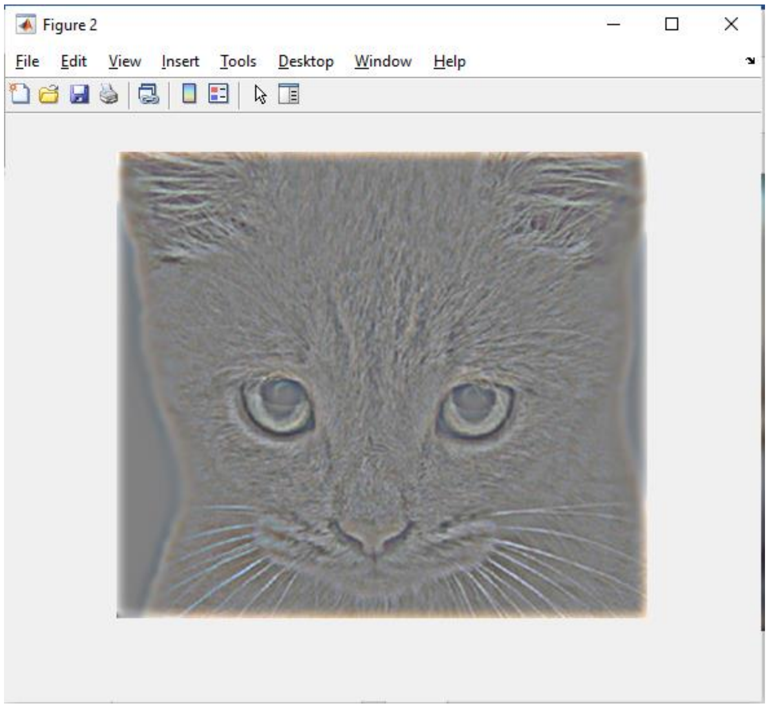
```
% Saving the dog and cat results
imwrite(low_frequencies_dog, 'low_frequencies_dog.jpg', 'quality', 95);
imwrite(high_frequencies_cat + 0.5, 'high_frequencies_cat.jpg', 'quality', 95);
imwrite(hybrid_image_1, 'hybrid_image_1.jpg', 'quality', 95);
imwrite(vis_1, 'hybrid_image_scales_1.jpg', 'quality', 95);


% Saving the plane and bird results
imwrite(low_frequencies_plane, 'low_frequencies_plane.jpg', 'quality', 95);
imwrite(high_frequencies_bird + 0.5, 'high_frequencies_bird.jpg', 'quality', 95);
imwrite(hybrid_image_2, 'hybrid_image_2.jpg', 'quality', 95);
imwrite(vis_2, 'hybrid_image_scales_2.jpg', 'quality', 95);


% Saving the bicycle and motorcycle results
imwrite(low_frequencies_bicycle, 'low_frequencies_bicycle.jpg', 'quality', 95);
imwrite(high_frequencies_motorcycle + 0.5, 'high_frequencies_motorcycle.jpg', 'quality', 95);
imwrite(hybrid_image_3, 'hybrid_image_3.jpg', 'quality', 95);
imwrite(vis_3, 'hybrid_image_scales_3.jpg', 'quality', 95);
```

The output for the above procedure is as follows:

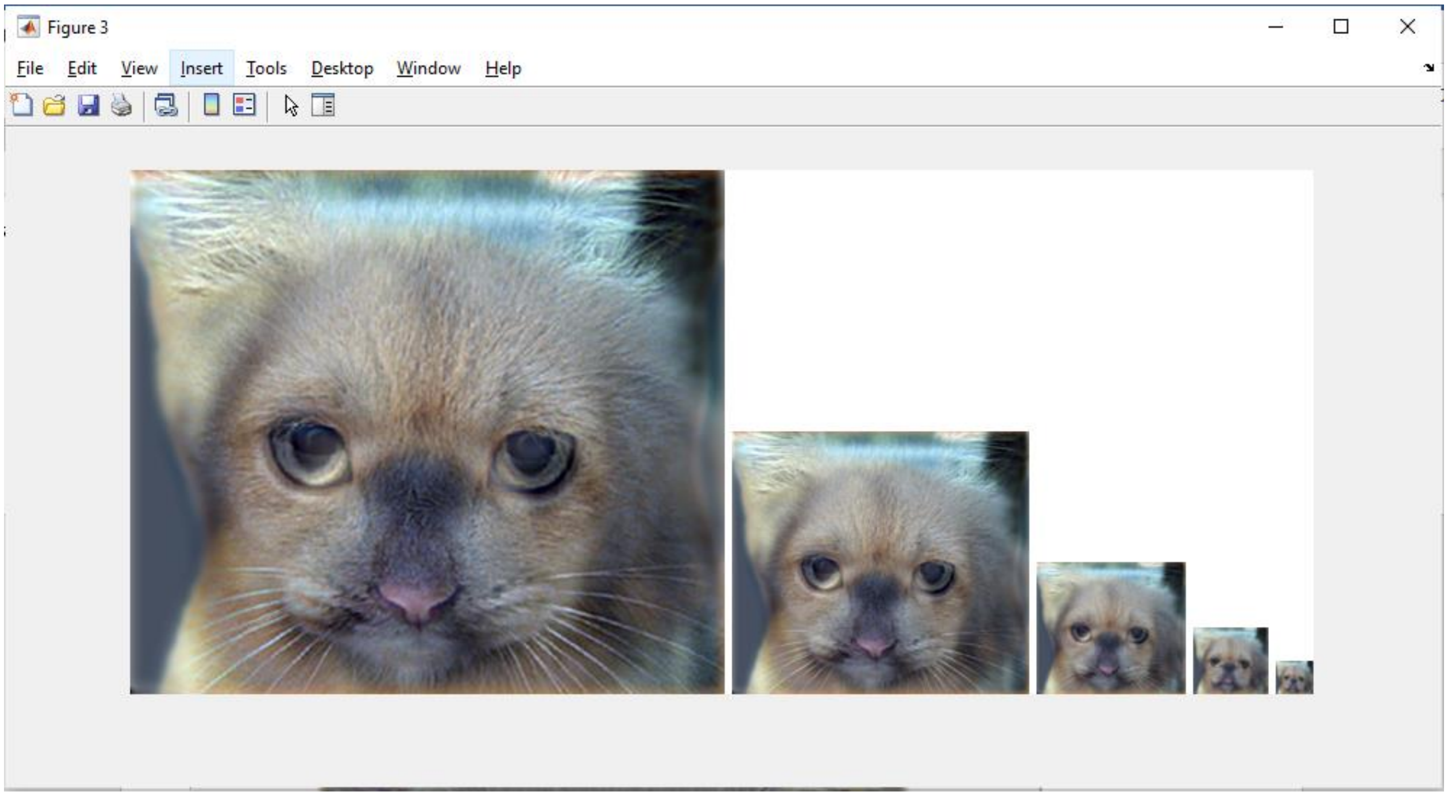
The low pass filtered image of dog:



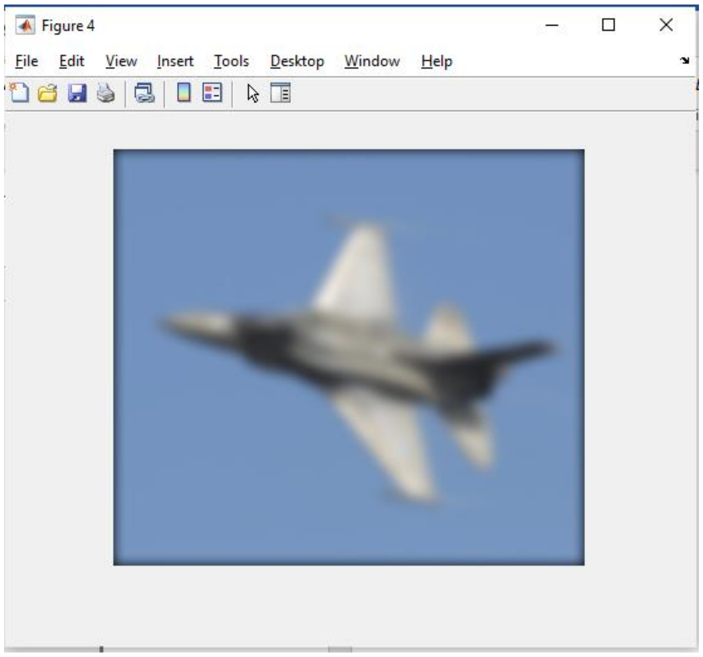
The high pass filtered image of cat:



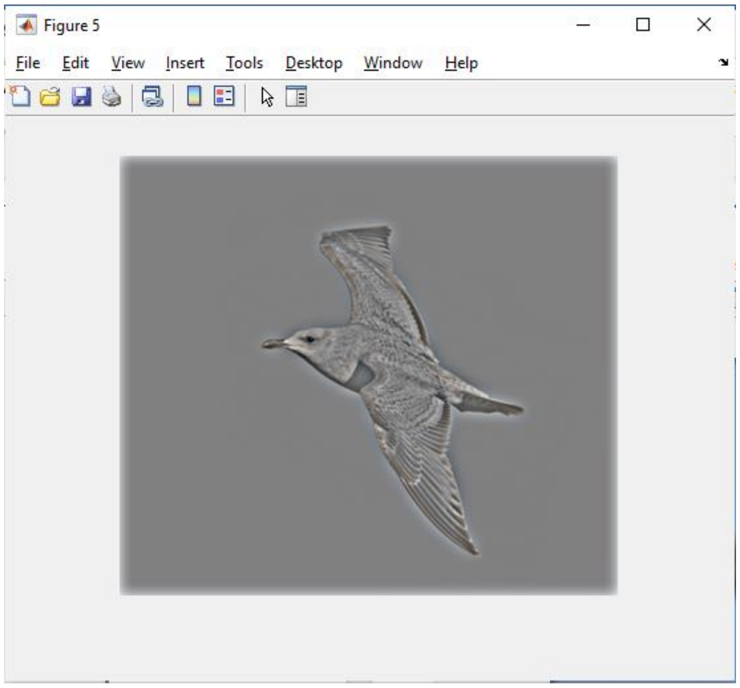
The hybrid image of dog and cat:



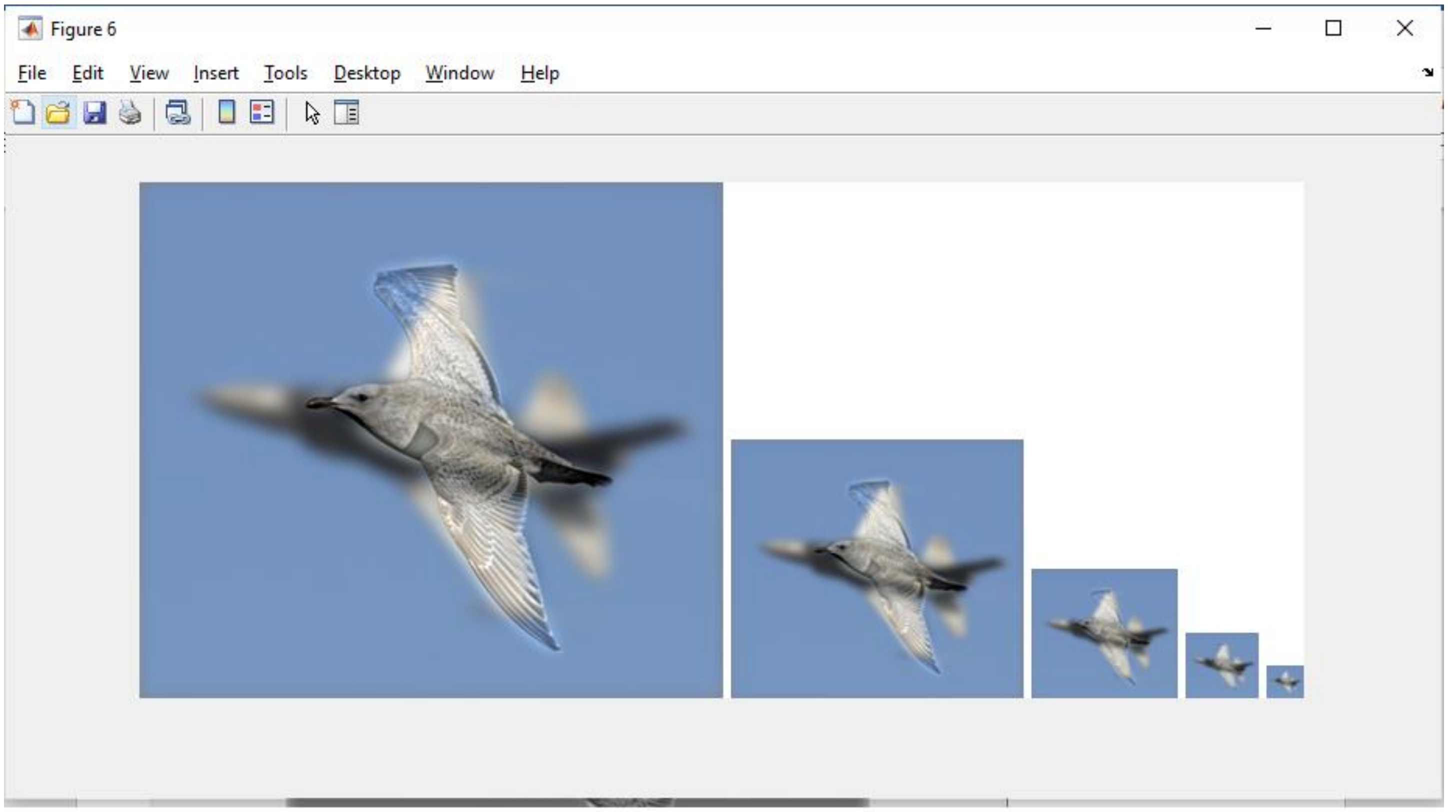
The low pass filtered image of plane:



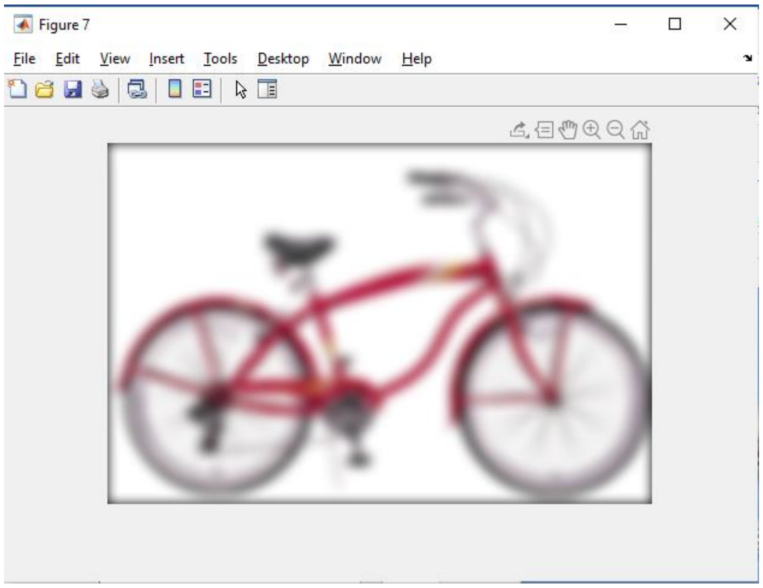
The high pass filtered image of bird:



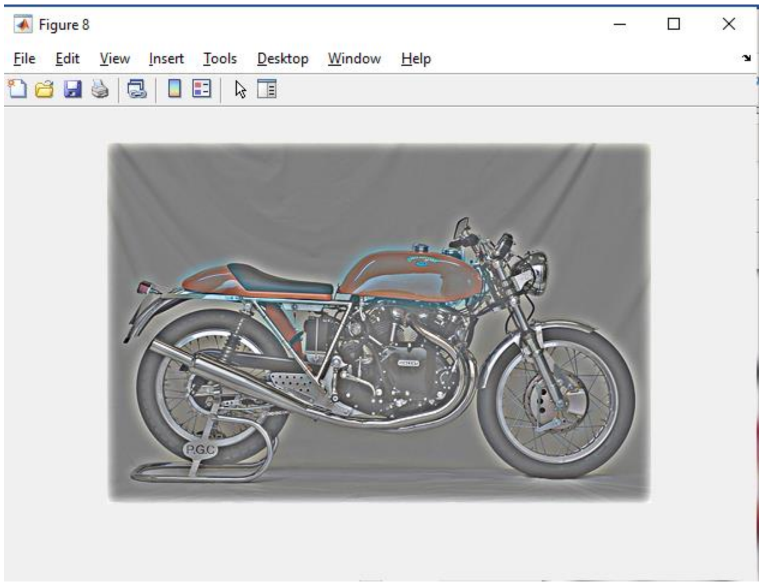
The hybrid image of bird and plane:



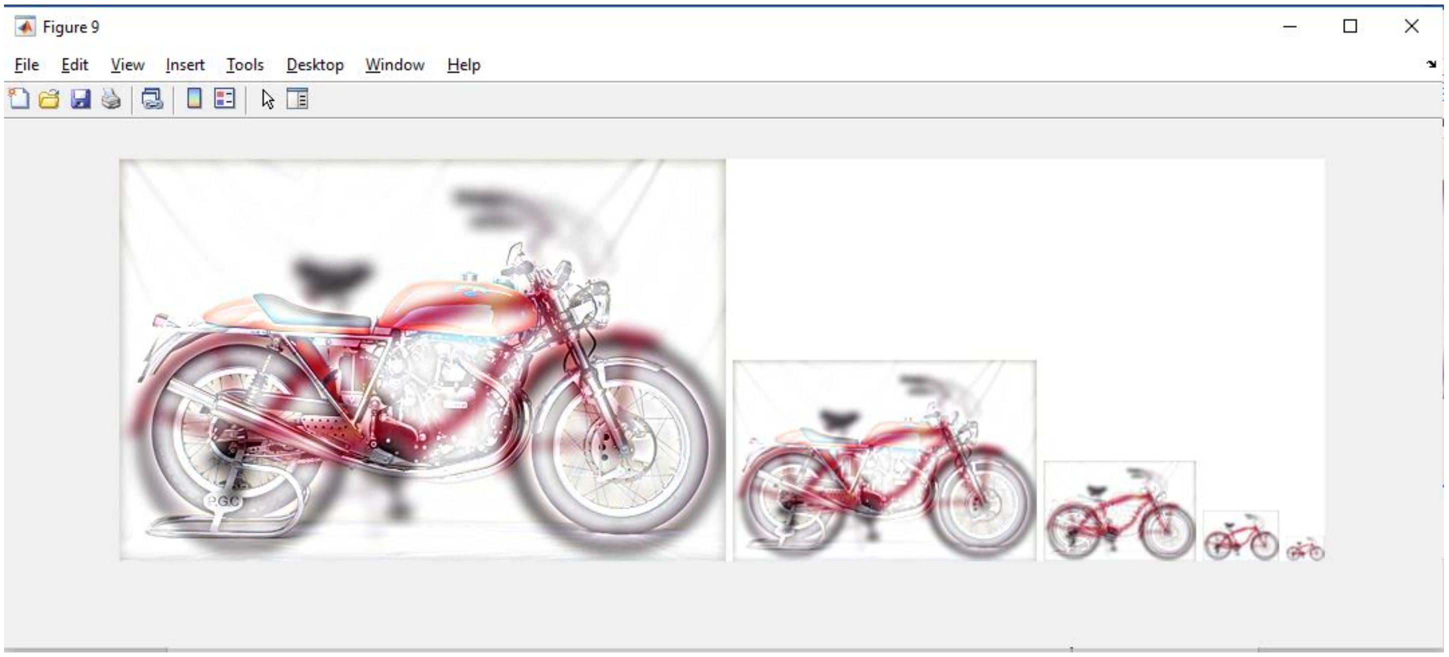
The low pass filtered image of bicycle:



The high pass filtered image of motorcycle:



The hybrid image of bicycle and motorcycle:



The algorithm for finding hybrid images was a pretty simple process. I had to think of a way to find the filtered image given the image filters and the my_imfilter did just that part. I used my_imfilter to generate filtered copies of the image. Then, adding the images is not a big deal in MATLAB. You just need to put a plus sign between both the images. The second cutoff frequency gave some different results than the single one. It produced a slightly different textured image. The Laplacian filter is also a great tool to find the high frequency components of an image. It can help to find the second derivative of the image but in our case, we don't need to find the lighter edges as they can prove to be obstructing the hybrid image. The vis_hybrid_image function did a pretty great job for visualizing the results correctly without going to have to see the image from different distances.

In conclusion, the purpose of this assignment was to get familiar with the filtering functions properly. I think this assignment did a pretty decent job for familiarizing us with the concept of convolution. The formation of my_imfilter function helped me understand the basics of filters well. Also, the concept of hybrid images was new to me and it was fun to learn about the insides of the hybrid images.