Nikolaos Chrysovalantis Tsagkopoulos
Student ID: 30357918

Hybrid Images (1st coursework).

In this report, I will demonstrate how I created a function that performs a convolution on an image with a template (a gaussian matrix) and acts as a lowpass (and a highpass as an extension) filter. The final goal is to obtain the low frequencies from one picture and the high frequencies of an other so adding them together will result in a hybrid image. The project follows the procedure as A. Oliva suggested in *"Hybrid images"* (2006)[1].

Convolution function:

For the convolution I made a function that takes as inputs a matrix A, and a matrix b. During the convolution, the template b starts from the left top corner of matrix A and strides from left to right, top to bottom until it reaches the bottom right corner of A and does an element wise multiplication storing the summation of the products before it moves to the next pixel.

The strategy is first to initialize a matrix that corresponds as output of the operation with dimensions equal to matrix's A. Then, we should find the starting and ending indices of C when the computation begins and stops (begins when the top left of b is on the top left corner of C) and refer to the center of b every time.

The template has always odd dimensions so the indices for the rows and columns go as:

$$i = \left(\frac{rows(b)}{2} + \frac{1}{2}\right) : \left[rows(A) - \left(\frac{rows(b)}{2} - \frac{1}{2}\right)\right]$$

$$j = \left(\frac{col(b)}{2} + \frac{1}{2}\right) : \left[\ col(A) - \left(\frac{col(b)}{2} - \frac{1}{2}\right)\right]$$

$$where\ step\ is\ 1.$$

So, the computations are stored in a matrix with dimensions equal as A and derives from the following equation:

$$C_{i,j} = \sum_i \sum_j A_{\left(i-\frac{\mathrm{rows}(b)}{2}:i+\frac{\mathrm{rows}(b)}{2},\,j-\frac{col(b)}{2}:j+\frac{col(b)}{2}\right)} .* \; b_{(1:end,1:end)}$$

$where\ b, rotated\ by\ 180°$. Intuitively, (shown by the index numbering) we see that the resulting image has black borders proportional to the kernel size.

Function: my_conv()

```
function [im_res,b] = my_conv(im,g)
%This function performs the conv. operator similar to the
%prebuilt matlab function: conv2d(A,b,'same').
%Commented section compares my_conv() to conv2().
%Author: Nikolaos Tsagkopoulos

imdim = size(im); %Image dim.
if size(imdim) < 3
    imdim(3) = 1;
end

temp_dim = size(g); %Template dim.
im_res = zeros(imdim); %Output init.

leftB = temp_dim(2)/2 + .5;   %Indices modification
rightB = temp_dim(2)/2 - .5;  %Works only for odd temps.
upperB = temp_dim(1)/2 + .5;
downB = temp_dim(1)/2 - .5;
%% CONVOLUTION %%
g_flp = flip(flip(g,2),1); %filter rotation 180

for k = 1 : imdim(3) %for every channel
    for i = upperB : imdim(1) - downB
        for j = leftB : imdim(2) - rightB
            im_res(i,j,k) = sum(sum(im(i-upperB+1:i+downB,...
                                    j-leftB+1:j+rightB,k).*g_flp));
        end
    end
end

%% conv2 Comparison
%The section belowcompares the my_conv()
%result with the prebuild function conv2()

% real_sum = zeros(imdim(1),imdim(2),imdim(3));
% for i=1:imdim(3)
%     real_sum(:,:,i) = conv2(im(:,:,i),g,'same');
% end
% b = real_sum - im_res;
% figure(); image(real_sum - im_res);

end
```

In the comments, right after the convolution function, I added some lines of code that checks the output with the prebuilt function of MATLAB. The only difference is that MATLAB smooths also the borders because the output size is the same as that of the input.

To reach the desired result I tried two approaches, one in the spatial domain and one in the frequency domain. Their basic difference is that in the first, image filtering is done through convolutions and in the second one through element by element multiplications. It is worth to mention that I tried several ways to implement the highpass filter and it was challenging.

Approach 1: Spatial Domain

After building the *my_conv* function it is was quite easy to build the hybrid images, so I created the *main_spatial* script which consists of the following steps:

1) Image reading
2) Setting *sigmas* and implement filtering
3) Obtaining the FFT for representation purposes in the freq. domain
4) Hybrid Images
5) Visualizing:
    a) Figure(1); Spatial representation of Original, filtered images.
    b) Figure(2); Representation of Original, filtered images in the frequency domain.
    c) Figure(3); Image pyramid of the hybrid image

Filtering:

In this approach the filters are just templates in the spatial domain and created in a function which takes as inputs *im* and *sigma*, creates an odd square matrix 4σ, and values of the template comes through the equation:

$$f(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-x_0)^2+(y-y_0)^2}{2\sigma^2}}$$

Then, inside in the *lowpass* function, performs the convolution and returns the smoother image.

Highpass filter is implemented through the *highpass* function which is basically a lowpass filter. The high frequencies of the image are obtained after subtraction of the convolved image from the original. After the subtraction the borders of the resulting image are corrected with a silly trick.

Function: lowpass()

```
function im_res = lowpass(im,sigma)
%This function create a gaussian template
%as the prebuilt:
%filter = fspecial('gaussian',ker_size,sigma);
%Author: Nikolaos Tsagkopoulos

ker_size = ceil(4*sigma + 1); %kernel dim Correction
if ~mod(ker_size,2)
    ker_size = ker_size + 1;
end

filter = zeros(ker_size,ker_size);
x0 = ceil(ker_size/2);
y0 = x0;
x = 1:ker_size;
y = 1:ker_size;
scalar = 1/(2*pi*sigma^2);
exponent = -1/(2*sigma^2);
for i =1:ker_size
    for j = 1:ker_size
        vars = (x(i) - x0).^2 + (y(j) - y0).^2;
```

```
        filter(i,j) = scalar.*exp(vars.*exponent);
    end
end

im_res = my_conv(im,filter);
end
```

Function: highpass()

```
function im_res = highpass(im,sigma)
%This function acts exactly as lowpass() but
%in order to have a highpass filter we remove
%the resulting image from the original and
%and correct the borders afterwards.

ker_size = ceil(4*sigma + 1); %kernel dim Correction
if ~mod(ker_size,2)
    ker_size = ker_size + 1;
end

filter = zeros(ker_size,ker_size);
x0 = ceil(ker_size/2);
y0 = x0;
x = 1:ker_size;
y = 1:ker_size;
scalar = 1/(2*pi*sigma^2);
exponent = -1/(2*sigma^2);
for i =1:ker_size
    for j = 1:ker_size
        vars = (x(i) - x0).^2 + (y(j) - y0).^2;
        filter(i,j) = scalar.*exp(vars.*exponent);
    end
end

im_res = my_conv(im,filter);
im_res = im - im_res;%Obtaining the High freq
im_res(im_res == im) = 0; %correcting the borders <-( the silly trick)

end
```

The next step in the *main_spatial* script is the *FFT* implementation, the representation functions and the hybrid image generator. They do not deserve more discussion about the implementation rather than just posting the codes and discussing the results in the end. So, let's proceed in the 2nd approach.

Approach 2: Frequency Domain

In this method I tried to apply the steps A. Oliva suggested, doing operations in the frequency domain. The sequence of the steps is:

1) Image reading
2) Setting *sigmas* and creating gaussian templates in the Freq. Domain
3) FFT in the original images
4) Convolution based on the equation:

$$\mathcal{F}(I_{convolved}) = \mathcal{F}(image).* \mathcal{F}(template)$$

5) IFFT in the convolved images for representation in the spatial domain

$$I_{convolved} = \mathcal{F}^{-1}\{\mathcal{F}(image).* \mathcal{F}(template)\}$$

6) Hybrid images obtained by the equation:

$$Hybrid\ image = \mathcal{F}^{-1}\left\{\mathcal{F}\left(I_{convolved,Low}\right) + \mathcal{F}\left(I_{convolved,High}\right)\right\}$$

Filtering:

In this approach the filter should be implemented in the frequency domain, but it was challenging. The *FFT* of the filter gives the same dimensions as in the spatial domain resulting in a template and an image with unequal dimensions. I kept images' dimensions as a reference and zero padded the rest of the template before the *FFT*.

The challenge was that the template is always odd, but some images have even dimensions. My solution was to implement an if condition that could check whether the filter could be zero padded, otherwise I cropped the excess row or column.

I tried to obtain the highpass filter from the lowpass after the *FFT*, so it would be in the frequency domain. I tried both:

$$\mathrm{H}(\omega_h) = 1 - \mathrm{H}(\omega_l), and$$

$$\mathrm{H}(\omega_h) = e^{\varphi(\omega_l)}\,(1 - \|\mathrm{H}(\omega_l)\|), \qquad where\ \varphi(\omega_l) = \arctan\left(\frac{Im(\omega_l)}{Real(\omega_l)}\right)$$

But the results were poor comparing with the first approach.

Function: LP_fourier()

```
function [f, im] = LP_fourier(sigma,dim,im)
%This function transforms a gaussian filter
%in the freq. domain and zero padding it be-
%fore, so it can do element wise multiplica-
%tion with an image.
%Author: Nikolaos Tsagkopoulos

ker_size = ceil(4*sigma + 1); %kernel dim Correction
if ~mod(ker_size,2)
    ker_size = ker_size + 1;
end

for i = 1:dim(3) %Gaussian filter
    f(:,:,i) = fspecial('gaussian',ker_size,sigma);
end

if mod((dim(1)-size(f,1))/2,2) %Check conditions for
    im = im(2:end,:,:);         %dimension reduction
end
if mod((dim(2)-size(f,2))/2,2)
    im = im(:,2:end,:);
end

dim=size(im); %zero pad and FFT
f = padarray(f,[(dim(1)-size(f,1))/2 (dim(2)-size(f,2))/2],'both');
f = fftshift(fft2(f));
end
```

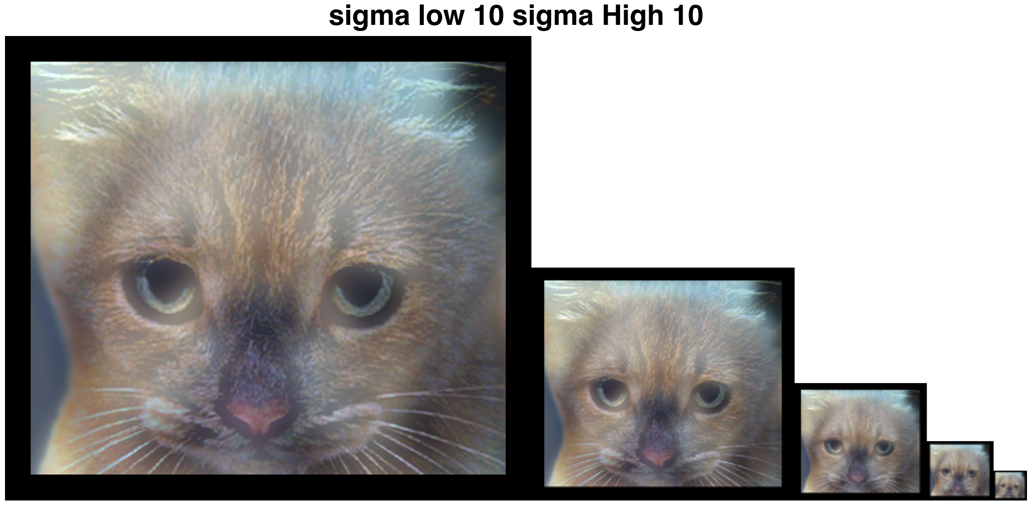
Function: HP_fourier()

```
function [fH, im] = HP_fourier(sigma,dim,im)
%This function based on the LP_fourier()
%transforms an lowpass filter to highpass.
%Author: Nikolaos Tsagkopoulos

ker_size = ceil(4*sigma + 1); %kernel dim Correction
if ~mod(ker_size,2)
    ker_size = ker_size + 1;
end
for i = 1:dim(3) %Gaussian filter
    f(:,:,i) = fspecial('gaussian',ker_size,sigma);
end

if mod((dim(1)-size(f,1))/2,2) %Check conditions for
    im = im(2:end,:,:);         %dimension reduction
end
if mod((dim(2)-size(f,2))/2,2)
    im = im(:,2:end,:);
end

dim=size(im); %zero pad
f = padarray(f,[(dim(1)-size(f,1))/2 (dim(2)-size(f,2))/2],'both');
f = fftshift(fft2(f));

%getting the Highpass filter
fH =   exp(complex(0,angle(f))).*(ones(dim) - abs(f));
end
```

Results

From the database you provided us with, I found interesting combinations between the: $dog, bird, motorbike, marilyn, submarine$ as low frequency images and $cat, plane, bicycle, einstein, fish$ as high frequency pictures.

I present you some examples for various sigmas with both approaches. Using the same cut-off frequency[1] for both low and high feature images is a poor technique as you see from the following figure.



A. Spatial approach:

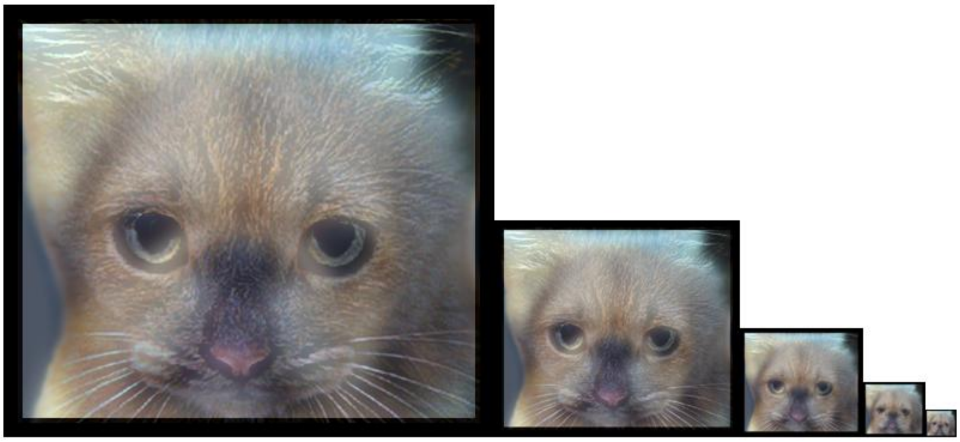


Figure 1 sigma low = 8, sigma high = 6

[1] Where: $Cutoff\ Frequency\ F_c = \frac{F_s}{2\pi\sigma}$, and $F_s = size(im)\ in\ spatial\ domain.$

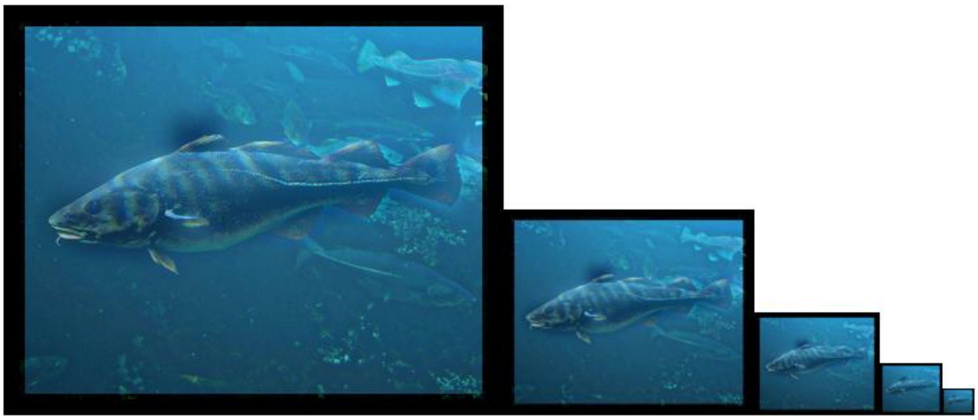


Figure 2 sigma low = 8, sigma high = 6

Figure 3 sigma low = 6, sigma high = 4

Figure 4 sigma low = 8, sigma high = 4

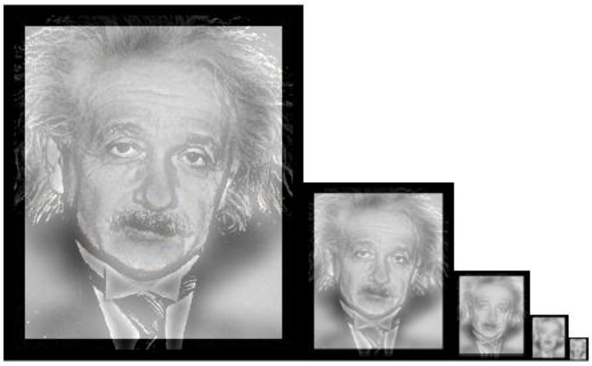


Figure 5 sigma low = 8, sigma high = 4

B. Frequency approach:

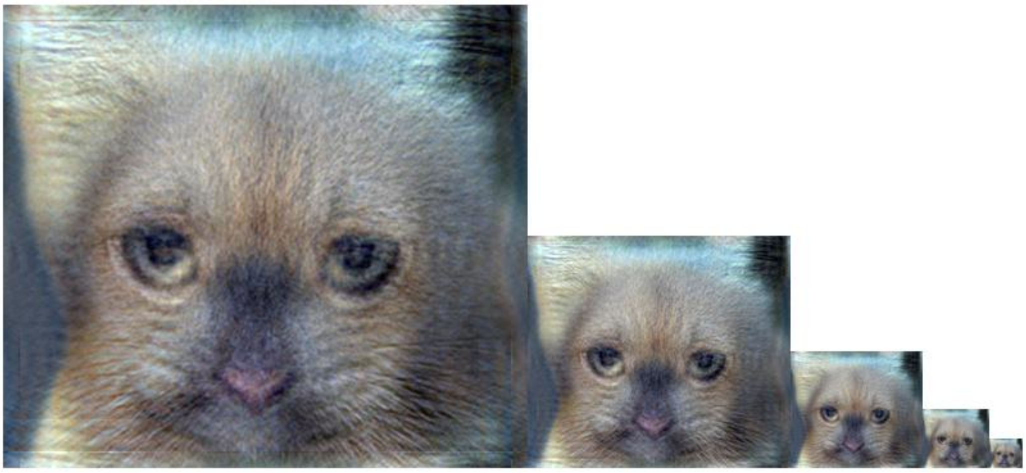


Figure 6 sigma low = 8, sigma high = 6

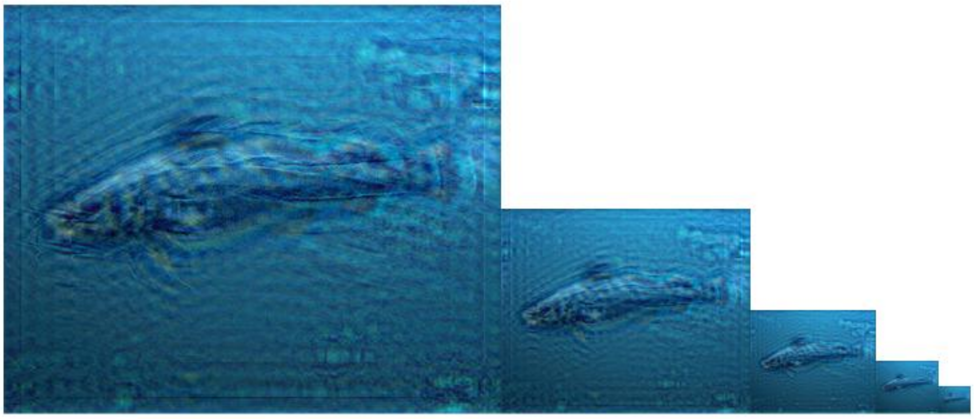


Figure 7 sigma low = 8, sigma high = 4

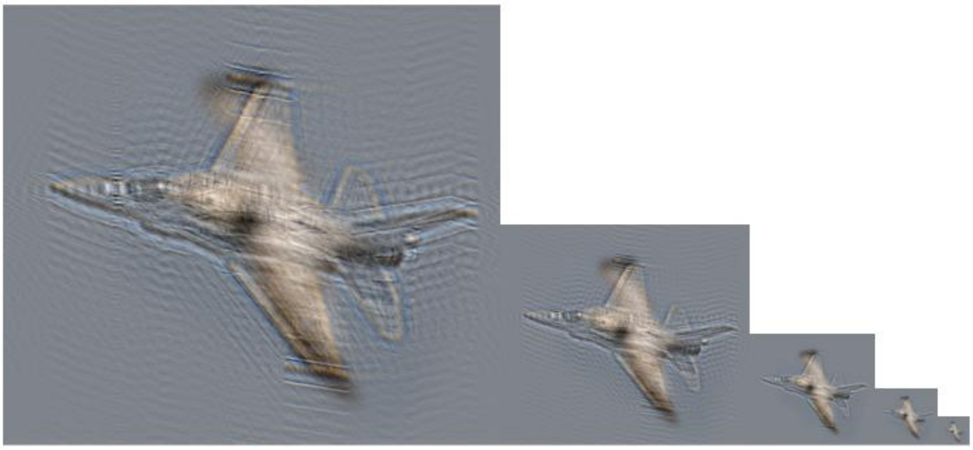


Figure 8 sigma low = 6, sigma high = 4

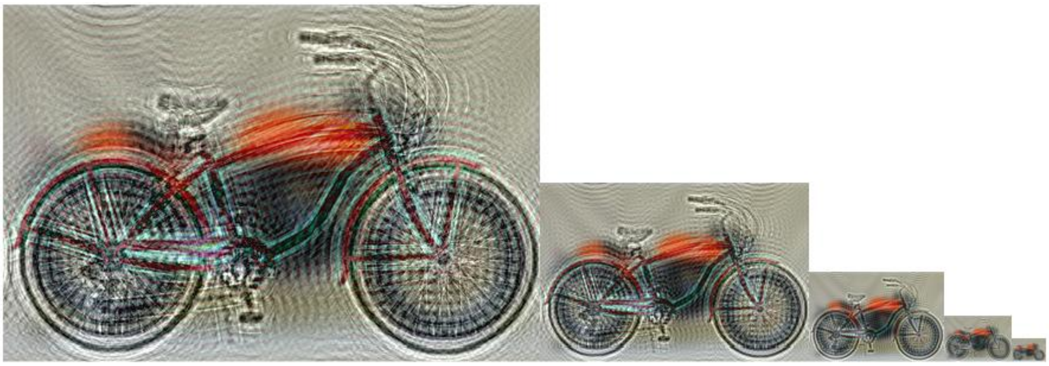


Figure 9 sigma low = 8, sigma high = 4

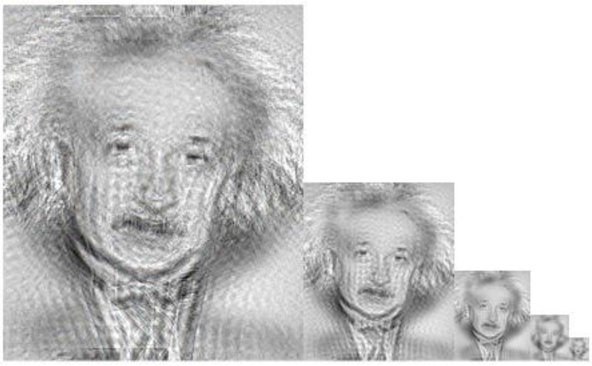


Figure 10 sigma low = 8, sigma high = 4

Anyone can see that in the 2nd approach, we do not lose dimensions through the convolution, but the high frequencies are distorted. My conclusion is that is due to either wrong implementation of the highpass filter, either something awkward happens after the *fftshift* implementation, or it is due to the phase.

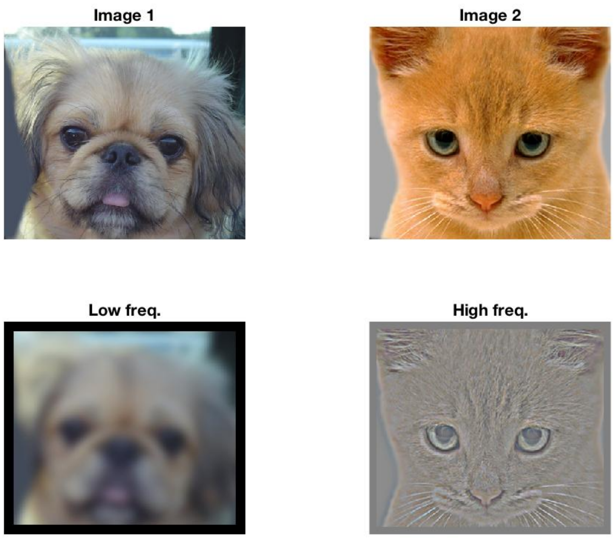
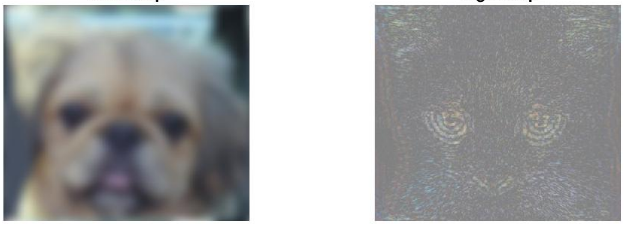


Figure 11 Is it obvious that the highpass filters are different

Low freq. High freq.



Next, I will post the rest of the codes.

Function: main_spatial()

```
%=============== METHOD 1. *SPATIAL DOMAIN* ==================
%In this method we apply the filters through a convolutional
%operation in the spatial domain. Then we represent the images
%in the spatial and the frequency domain. In the end, we produce
%the hybrid images with addition in the spatial domain.
%Author: Nikolaos Tsgkopoulos

close all; clear all; clc;
addpath('data'); %add image path to directory

im1 = imread('submarine.bmp'); %image 1
im1 = im2double(im1);
im2 = imread('fish.bmp'); %image 2
im2 = im2double(im2);
imdim = size(im1);

%% Filtering the images
sigmaL = 8;
sigmaH = 6;

im_1_filtrd = lowpass(im1,sigmaL);
im_2_filtrd = highpass(im2,sigmaH);

%% FFT for RGB
Fou_im1 = zeros(imdim); %dim initialization
Fou_im2 = zeros(imdim);
Fou_im1_filtrd = zeros(imdim);
Fou_im2_filtrd = zeros(imdim);

for i = 1:size(im1,3) %Calculated for every channel
    Fou_im1(:,:,i) = fftshift(fft2(im1(:,:,i)));
    Fou_im2(:,:,i) = fftshift(fft2(im2(:,:,i)));
    Fou_im1_filtrd(:,:,i) = fftshift(fft2(im_1_filtrd(:,:,i)));
    Fou_im2_filtrd(:,:,i) = fftshift(fft2(im_2_filtrd(:,:,i)));
end

%% Visualizing the features

spatial_representation(im1,im2,im_1_filtrd,im_2_filtrd);
freq_representation(Fou_im1,Fou_im2,Fou_im1_filtrd,Fou_im2_filtrd);

%% Hybrid Images

im_1_filtrd = im2uint8(im_1_filtrd);
im_2_filtrd = im2uint8(im_2_filtrd);
```

```
Hybrid = imadd(im_1_filtrd,im_2_filtrd);


%Image pyramid
N = 5;
im_pyramid(Hybrid,N,sigmaL,sigmaH)
```

Function: main_frequency()

```
%=============== METHOD 2. *FREQUENCY DOMAIN* ==================
%In this method we apply the filters through a element wise multi-
%plication between the freq representation of the filter and the
%freq. representation of the image. [...]
%operation in the spatial domain. Then we represent the images
%in the spatial and the frequency domain. In the end, we produce
%the hybrid images with addition in the spatial domain.
%Author: Nikolaos Tsgkopoulos

close all; clear all; clc;
addpath('data'); %add image path to directory

im1 = imread('dog.bmp'); %image 1
im1 = im2double(im1);
im2 = imread('cat.bmp'); %image 2
im2 = im2double(im2);
imdim = size(im1);

%% Getting Fourier(filter) padded & Fourier(im)
sigmaL = 8;
sigmaH = 6;

[Fou_filterL, im1] = LP_fourier(sigmaL,imdim,im1);
[Fou_filterH, im2] = HP_fourier(sigmaH,imdim,im2);
imdim=size(im1);

Fou_im1 = fftshift(fft2(im1));
Fou_im2 = fftshift(fft2(im2));


%% Convolving in the freq. domain
FLow_conv1 = Fou_im1.*Fou_filterL; %shifted for reasons(?)
FHigh_conv2 = Fou_im2.*Fou_filterH; % -"-
spatial_low = zeros(imdim);
spatial_high = zeros(imdim);

%Deconvolving (spatial domain)

spatial_low = abs(ifftshift(ifft2(FLow_conv1)));
spatial_high = abs(ifftshift(ifft2(FHigh_conv2)));

%% Visualizing the features

spatial_representation(im1,im2,spatial_low,spatial_high);
freq_representation(Fou_im1,Fou_im2,FLow_conv1,FHigh_conv2);
```

```
%% Hybrid Images

Hybrid = ifftshift(ifft2(FLow_conv1 + FHigh_conv2));

%Image pyramid
N = 5;
im_pyramid(abs(Hybrid),N,sigmaL,sigmaH)
```

Function: spatial_representation()

```
function spatial_representation(A,B,Af,Bf)

figure(1);
subplot(2,2,1); imshow(A); title('Image 1'); ylabel('Original');
subplot(2,2,2); imshow(B); title('Image 2');
subplot(2,2,3); imshow(Af); title('Low freq.');
subplot(2,2,4); imshow(Bf + 0.5); title('High freq.'); %Visual reasons

end
```

Function: freq_representation()

```
function freq_representation(A,B,AFourier,BFourier)

figure(2);
subplot(2,2,1); image(abs(A(:,:,1))); title('Image 1'); ylabel('Original');
subplot(2,2,2); image(abs(B(:,:,1))); title('Image 2');
subplot(2,2,3); image(abs(AFourier(:,:,1))); title('Low freq.');
subplot(2,2,4); image(abs(BFourier(:,:,1))); title('High freq.');

end
```

Function: im_pyramid()

```
function im_pyramid(im,n,sL,sH)
%This function performs a pyramid of the image "im"
%in "n" samples
%Author: Nikolaos Tsagkopoulos

image = im;
hybrid_pyr = im;
for i=2:n
    image{i} = imresize(image{i-1},.5);
    %changing subsampled images dimensions, so we can concatenate
    %and show the images as one
    hybrid_pyr{i} = padarray(image{i},size(im,1)-size(image{i},1),255,'pre');
end
image = cat(2,hybrid_pyr{:});
figure(3);
imshow(image)
title(sprintf('sigma low %s sigma High %s',num2str(sL), num2str(sH)))
end
```

Works Cited:

[1] Oliva, A., Torralba, A. and Schyns, P. G. Hybrid images. In *Proceedings of the ACM SIGGRAPH 2006 Papers* (Boston, Massachusetts, 2006). ACM, [insert City of Publication],[insert 2006 of Publication].

[2] Nixon, M. and Aguado, A. S. *Feature Extraction \& Image Processing for Computer Vision, Third Edition*. Academic Press, Inc., 2012.