Homework 2 Writeup

Instructions

- Describe any interesting decisions you made to write your algorithm.
- Show and discuss the results of your algorithm.
- Feel free to include code snippets, images, and equations.
- Use as many pages as you need, but err on the short side If you feel you only need to write a short amount to meet the brief, th
- **Please make this document anonymous.**

In the beginning...

In mathematical view, image convolution uses 2D matrix to effect the image. To achieve image convolution, we should receive an image, and the filter, usually has smaller size compared to the image. In transformation part, flipping the filter horizontally and vertically at first is necessary. Then locate the filter's first element at the pixel of the image. Do the elementwise multiplication, and sum up the result of the calculation, the output image's pixel at the same position of the center element of the filter would have that value. Do this for every pixel of the image. When filter exceeds the image, we need paddings for image to calculate the result properly. The padding could be 0, or the reflection of the image. After all the jobs are completed, the output image will come out, the image applied filter to the original image.

Interesting Implementation Detail

My code uses reflection padding for getting better filter applied results.

```
[m, n] = size(image(:,:,1));
[m_, n_] = size(filter);

pad_m = (m_ - 1) ./ 2;
pad_n = (n_ - 1) ./ 2;

M = m + pad_m * 2;
N = n + pad_n * 2;

reflected_image = padarray(image, [pad_m pad_n 0], '
    symmetric', 'both');
```

Then, we calculate elementwise multiplication per pixel, and sum them, and make it output's pixel value. filter should be flipped in two direction before the processing.

```
filter = flipud(fliplr(filter));
temp = zeros([M N 3]);
for i = 1:3
    for x = pad_m + 1 : (pad_m + m)
        for y = pad_n + 1 : (pad_n + n)
            for k = -1 * pad_m : pad_m
                for l = -1 * pad_n : pad_n
                    reflected_image(x + k, y + l, i);
                    filter(k + pad_m + 1, l + pad_n + 1);
                    temp(x, y, i) = temp(x, y, i) +
                       reflected_image(x + k, y + l, i) *
                        filter(k + pad_m + 1, l + pad_n +
                        1);
                end
            end
        end
    end
end

output = real(temp(pad_m + 1: (m + pad_m), pad_n + 1 : (n
    + pad_n), :));
```

Also, we can call the function with the new argument: isftt. If it is 1, we use fft algorithm to generate the image. This is much faster.

```
if ~exist('isfft', 'var')
    isfft = 0;
end
```

```
if (isfft == 1)
    low_pass_image = ifft2(fft2(reflected_image) .* fft2(
       filter, M, N));

    output = real(low_pass_image(m_ : (m + m_ - 1), n_ :
       (n + n_ - 1), :));
else
    % normal convolution code...
end
```

A Result

We could extrace different feature from the two images, and make hybrid images with the code. But it takes some time to do the convolution; Maybe there is better algorithm

or function to calculate sum of the elementwise multiplication.