

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, then
- **Please make this document anonymous.**

Algorithm Implementation

In the filtering of an image using convolution, a filter called the kernel is applied to each pixel of the image. The given `my_imfilter` takes the image and the filter as parameters and returns the output.

After studying the convolution filtering algorithm, I learned that the filter should be flipped both horizontally and vertically before applying. The flipping of the kernel is done in line (1) of the code below.

When applying the kernel on elements at the edge, there should be padding around it so that the size of the kernel and the size of the element and its neighbor match. The width of the padding is half the size of the kernel at respective dimensions. The calculation of the padding size and padding the image is done in lines (2) and (3).

After applying filter to the padded image, the resulting matrix is the same size as the padded image. Thus, a zero matrix of the same size as padded image is created in line (4).

The given image is in RGB space, so the for loop is iterated through the third dimension of the image matrix in line (5). Only the pixels that are not paddings are iterated in the double for loop in lines (6) and (7). The target area centered at the pixel (i,j) is accessed in line (8).

The corresponding element in the resulting filtered image is updated by the sum of elements of the element-wise multiplication between the target matrix and the kernel. This is the main idea of convolution filtering and shown in line (9). After the triple for loops, the filtered image should be cropped back to match the original image size. The crop is done in line (13).

The full code is in file `my_imfilter.m`.

```
1 flipped_kernel = flip(flip(filter, 1), 2);
2 pad_size = floor(size(filter)/2);
3 padded_image = padarray(image, [pad_size, 0]);
4 filtered_image = zeros(size(padded_image));
```

```

5 for d=1:size(padded_image, 3)
6     for i=pad_size(1)+1:size(image, 1)+pad_size(1)
7         for j=pad_size(2)+1:size(image, 2)+pad_size(2)
8             target = padded_image(i-pad_size(1):i+
                                   pad_size(1), j-pad_size(2):j+pad_size(2),
                                   d);
9             filtered_image(i, j, d) = sum(sum(target.*
                                   flipped_kernel));
10        end
11    end
12 end
13 output=filtered_image(pad_size(1)+1:size(image,1)+
    pad_size(1), pad_size(2)+1:size(image,2)+pad_size(2),
    :);

```

Also, an hybrid image is created using the `gen_hybrid_image` function which uses the `my_imfilter` function. Implementation is as below (*gen_hybrid_image.m*).

```

1 function [hybrid_image, low_frequencies, high_frequencies]
   = gen_hybrid_image( image1, image2, cutoff_frequency )
2
3 filter = fspecial('Gaussian', cutoff_frequency*4+1,
   cutoff_frequency);
4 low_frequencies = my_imfilter(image1, filter);
5
6 high_frequencies = image2-my_imfilter(image2, filter);
7
8 hybrid_image = low_frequencies + high_frequencies;

```

Attempts

While implementing the algorithm, there were several bugs and errors that I had to resolve. The most difficult test case to pass was the large blur. Initially, I had an incorrect padding calculation and ended up with the following image. Shown in Figure 1.

Also, I attempted creating a function for element-wise multiplication between matrices. However, it made the code to be longer and difficult to debug. Later, I found that the same arithmetic can be performed by the `.*` operator in front of the multiplication operator `*`, and replaced the corresponding code with a simpler version.

Most of other errors were about calculation between matrices of different dimensions.

Results

The results of the test cases are shown in Table 1. The difference between the results of original function and `my_imfilter` is unnoticeable to the eye.

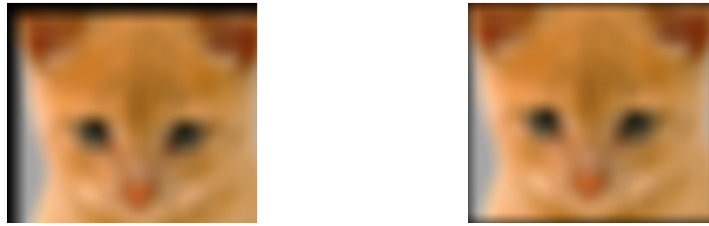


Figure 1: *Left*: Incorrect padding calculation. *Right*: After fix..

Table 1: Comparison between my_imfilter and original imfilter functions.





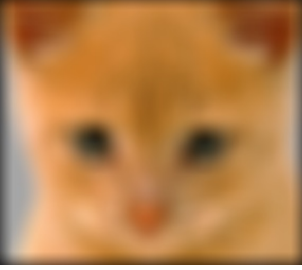
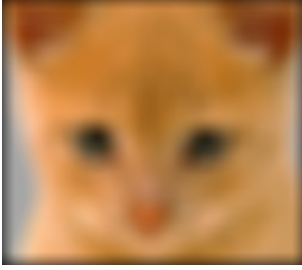






| Test case | my_imfilter | imfilter |
|------------|---|--|
| Identity |  |  |
| Small Blur |  |  |
| Large Blur |  |  |

Table 1: Comparison between my_imfilter and original imfilter functions.

| Test case | my_imfilter | imfilter |
|------------------|--|---|
| Sobel Filter |  |  |
| Laplacian Filter |  |  |
| High Pass |  |  |

The results of the hybrid image is shown in Figure 2. High frequencies are removed from the cat image and the low frequencies are removed from the dog image. The low passed image tends to be blurry because the difference from the neighboring pixels is reduced. On the other hand, high passed filter tends to be sharpened. The resulting images are combined to create the hybrid image.



Figure 2: *Left*: Low frequency cat image. *Middle*: High frequency dog image. *Right*: Hybrid image