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

In the beginning...

first of all I took a look a the lecture slides and recap the formulas and differences between convolution and correlation. To get a even better understanding I extended my research by some good sources on the internet.
The convolution formula is:

$$h[m,n] = \sum_{k,l} f[k,l] * I[m-k, n-l]$$

flipping the filter on x and y axis results in the same as the correlation and your formula changes to:

$$h[m,n] = \sum_{k,l} f[k,l] * I[m+k, n+l]$$

Since flipping is not the big deal, it might help to reduce the difficulty of implementing the filter.

Interesting Implementation Detail

One interesting implementation detail might be the way I handle the kernel 'image-fragment' multiplication.

```
1 subimages = view_as_windows(image_adj, kernel_fliped.
     shape)
2 for i in range(subimages.shape[0]):
3     for j in range(subimages.shape[1]):
4         #Apply filter and sum up
5         result = np.multiply(subimages[i][j],
              kernel_fliped).sum()
6         h[i][j][layer] = result
```

I used the '*view_as_windows*' to generate all possible 'image-fragments' using the current kernel. This reduces my code massively and leads to a good general overview. Using this function, i iterate over all fragments and apply the (flipped) filter and sum everything up. Inspecting the '*view_as_windows*' function reveals how the image-fragments' are generated. Knowing this it is easy to assign the results to the right cell of h.

A Result

Since I have tested the individual code sections one by one, i had no really bad results (just some black pictures).
After finishing the coding part, I start trying out different combinations of hybrid images. I figured out that it is important which image is high/low frequency. Figure 1 shows an example illustration where left Einstein has low frequency and on the right Einstein has high frequency.

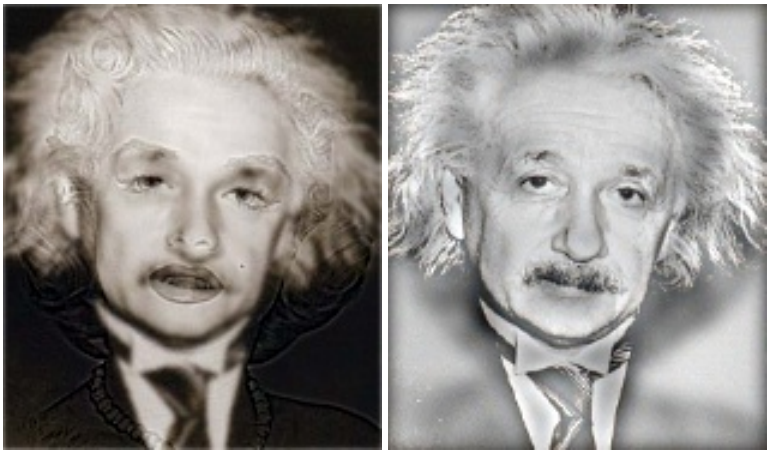

Figure 1: *Left:* Einstein is low fr *Right:* Einstein is high fr

An other try of creating a hybrid picture of kaist and postech mascot can be seen in figure 2. The result is not a perfect image, since i couldn't find really good matching images.

Figure 2: Try of creating a mascot hybrid image

After finishing the tasks, I start working on the second padding method. I extended my

filter function by a flag, but still function is backward compatible, because of the default value as seen in the following:

```
1 def my_filter2D(image, kernel, bordertype='ZEROS'):
```

The second padding is outsourced into a own function:

```
1 def reflected_pad(image, lr_extend, ud_extend):
2     pad_left = np.fliplr(image[:,1:lr_extend+1,:])
3     pad_right = np.fliplr(image[:,-lr_extend-1:-1,:])
4     image = np.concatenate((pad_left, image, pad_right),
          axis=1)
5     pad_top = np.flipud(image[1:ud_extend+1,:,:])
6     pad_down = np.flipud(image[-ud_extend-1:-1,:,:])
7     image = np.concatenate((pad_top, image, pad_down),
          axis=0)
8     return image
```

This code takes for every direction (left, right, up, down) the reflected part of the array (but ignores the first row/col) and flips it to fulfill the requirement. Since I first added the left/right padding, the top/down padding also reflects the added left/right padding. Using the new padding method, we are able to blur pictures without a black border as seen in figure 3.

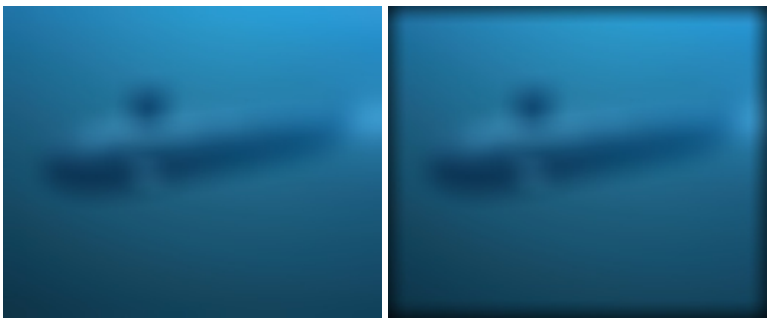

Figure 3: *Left:* reflected pad *Right:* zero pad