CMSC498D Homework 2 Report

William Lin

February 28th 2025

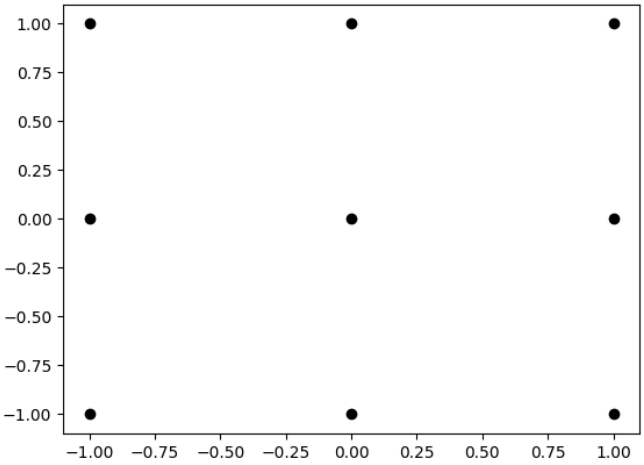
Introduction

In this homework: I explored various different image filters, the inner workings of convolution by pixel-based filtering and through the fourier transform, extracting phase and magnitude of frequencies, and creating hybrid images by combining frequencies of images. I understood the theories quite well, however the main struggles I found were working with the numpy functions. I've never worked with numpy's fft functions, there were lots of documentation reading. But, I got comfortable with them at the end of the homework.

Filtering

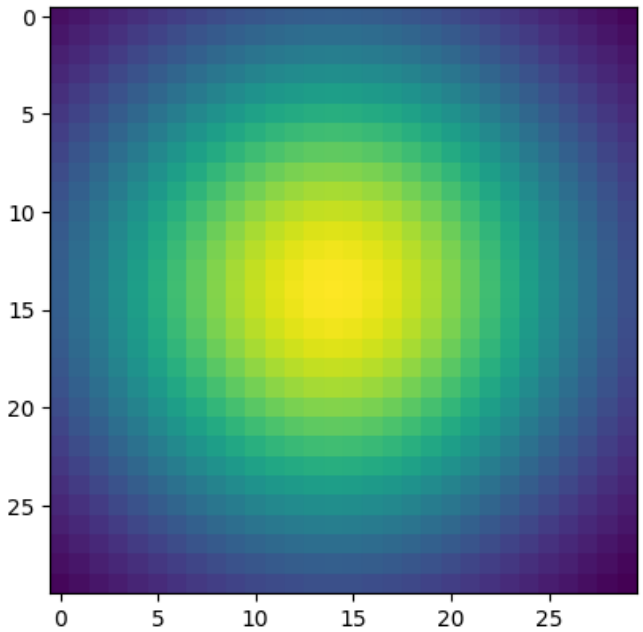
Gaussian Kernel

In this section, I understood the idea was to take advantange of the Gaussian distribution given by $f(i,j) = \frac{1}{2\pi\sigma^2}e^{-\frac{i^2+j^2}{2\sigma^2}}$. Where the highest weighted values would be the center pixel and weighting decreases the futher it is away from the center. At first, I wasn't sure what `np.meshgrid()` did and why it would help me. However, after reading the documentation and also playing around with an example:



This is perfect for the gaussian function since we can center at (0, 0) and get an appropriate coordinate for all the values around the center to plug into the gaussian function. On top of that, the output given by `np.meshgrid()` perfectly allows me to compute the full kernel in just 2 lines.

After successfuly implementing `gausskernel()`, the visualization for $\sigma = 10$ is shown here:



Convolution

Regarding `myfilter()`, I performed a raster scan starting from the top left pixel of the image and ended on the bottom right pixel. Then, I performed an elementwise multiplication with the filter. I didn't apply a physical padding to the image since simply checking if my elementwise multiplication was going to out of bounds of the image is enough to achieve that zero value effect which also saves some memory. Although I do want to note that this pixel-based implementation is very slow. One problem I did have at first with the convolution was not flipping my filter and so I performed a correlation instead of a convolution. This was fixed after realizing my shifts for filter 6 and 7 were off.

Filters

Using `myfilter()` to convolve "Iribe.jpg" with various filters results in the following. I've also included the time module to time each convolution which will be useful later.

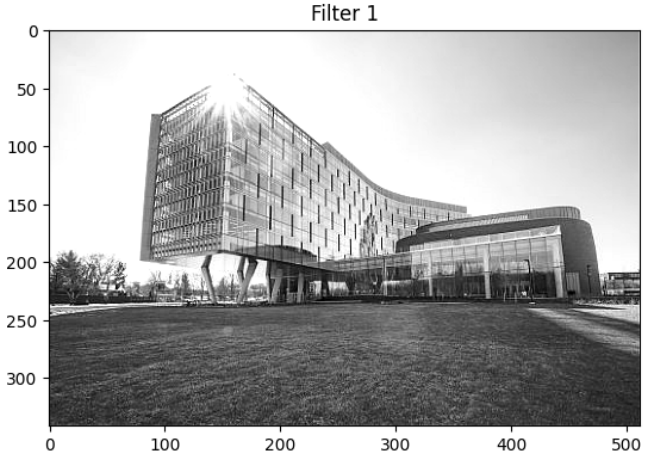
Filter 1

Filter one is given by:

$$\begin{bmatrix} -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \\ -\frac{1}{9} & 2 & -\frac{1}{9} \\ -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \end{bmatrix}$$

This filters is very similar to a laplacian filter. Although I'm not quite sure if it exactly is. However, the filter works by weighing the center pixel higher and puts a negative weight on the surrounding pixel values. This achieves a slightly sharpening effect on the image.

`Time taken: 1.27 s`



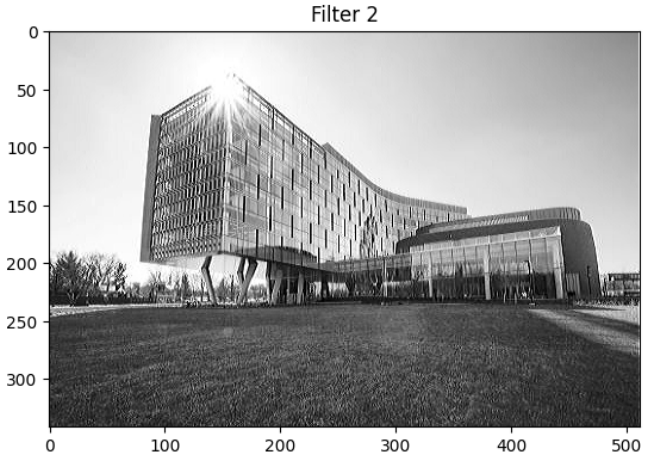
Filter 2

Filter two is given by:

$$\begin{bmatrix} -1 & 3 & -1 \end{bmatrix}$$

This filter also sharpens the image as well. The center pixel is weighted higher while the left and right values have negative weights. So again, we see the sharpening effect.

`Time taken: 0.48 s`

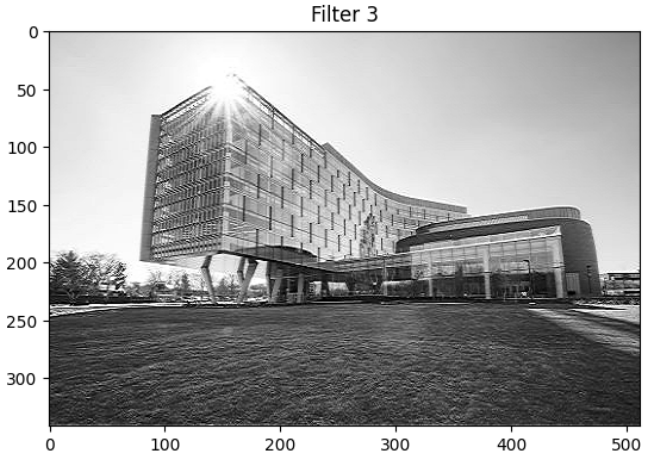


Filter 3

Filter three is given by:

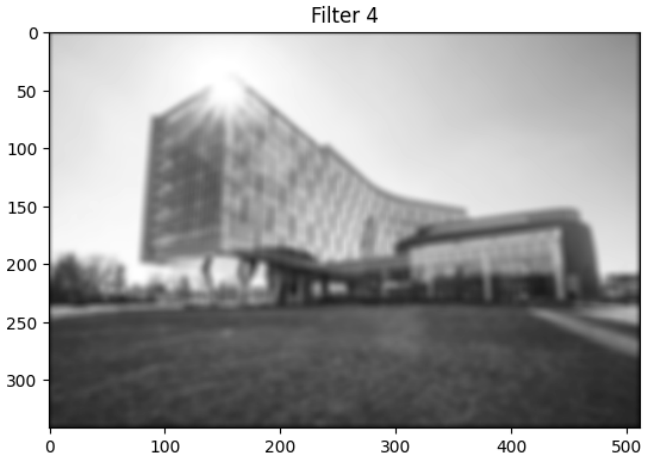
$$\begin{bmatrix} -1 \\ 3 \\ -1 \end{bmatrix}$$

This filter is similar to the above in that it weighs the center pixel higher. But this time, the upper and lower values have negative weights. Also, horizontal edges are now sharpened. Acting like a horizontal edge detector.
`Time taken: 0.52 s`



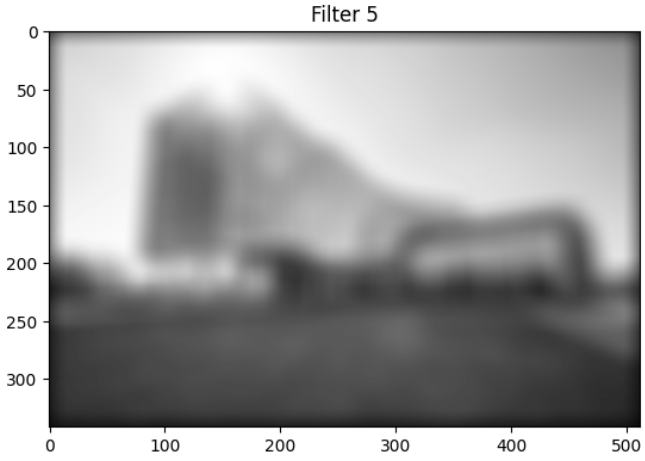
Filter 4

Filter four is given by `gausskernel(3)`. This filter is a gaussian blur filter. Since σ has a fairly lower value of 3, the blur isn't as intense compared to what we will see in filter 5. I noticed that the convolution for this filter started taking a little longer given that the size of the kernel grew a bit (9 x 9).
`Time taken: 10.03 s`



Filter 5

Filter five is given by `gausskernel(10)`. Notice that this gaussian blur kernel is way for intense with $\sigma = 10$. The kernel's size 30 x 30 significantly slowed down the convolution. I would like to note that although this was fairly slow, the gaussian kernel is separable which if used, would make this convolution method faster when convolving images with gaussian kernels.
`Time taken: 106.47 s`

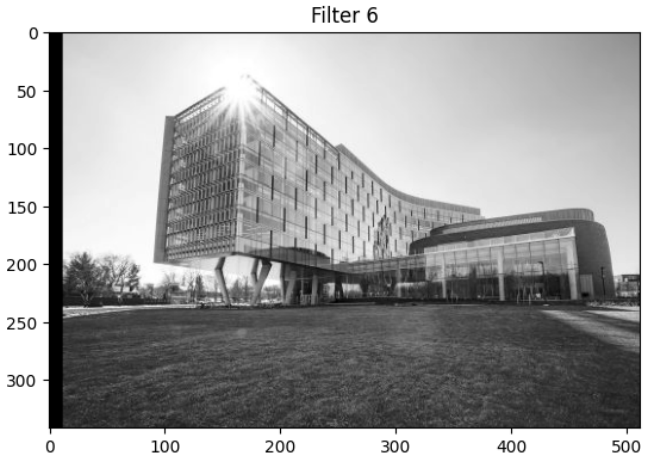


Filter 6

Filter six is given by:

$$\begin{bmatrix} 0 & 0 & \dots & 0 & 1 \end{bmatrix}$$

(contains 24 zeros followed by a 1)
This filter shifts the image to the right by 12 pixels. The reason for this is because for a given pixel on the image, the center value will be aligned with index 12 of the filter. The 1 value in the filter will be 12 pixels away. Also, the filter is flipped. So the current pixel value will capture the pixel value that is 12 pixels left. Thus shifting everything to the right
`Time taken: 3.40 s`

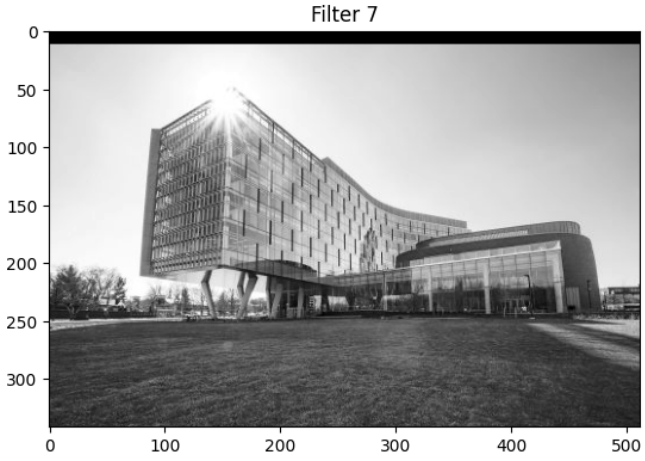


Filter 7

Filter seven is given by:

$$\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

(contains 24 zeros followed by a 1)
Similar to filter six, this filter instead shifts the image down by 12 pixels.
`Time taken: 3.91 s`

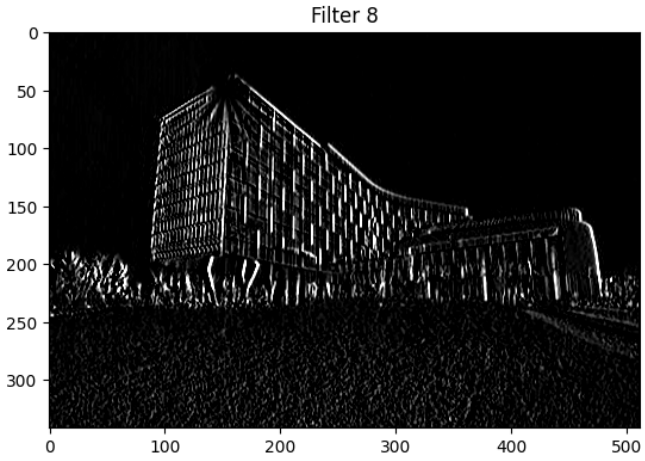


Filter 8

Filter eight is given by:

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

This filter is the sobel x filter which is used for horizontal edge detection which is reflected in the image.
`Time taken: 1.36 s`

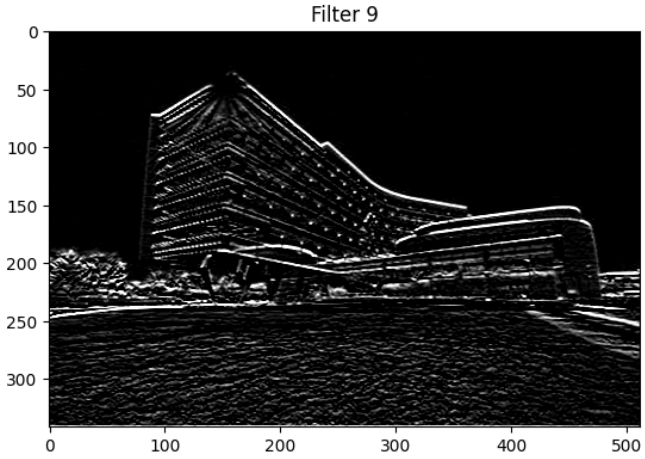


Filter 9

Filter nine is given by:

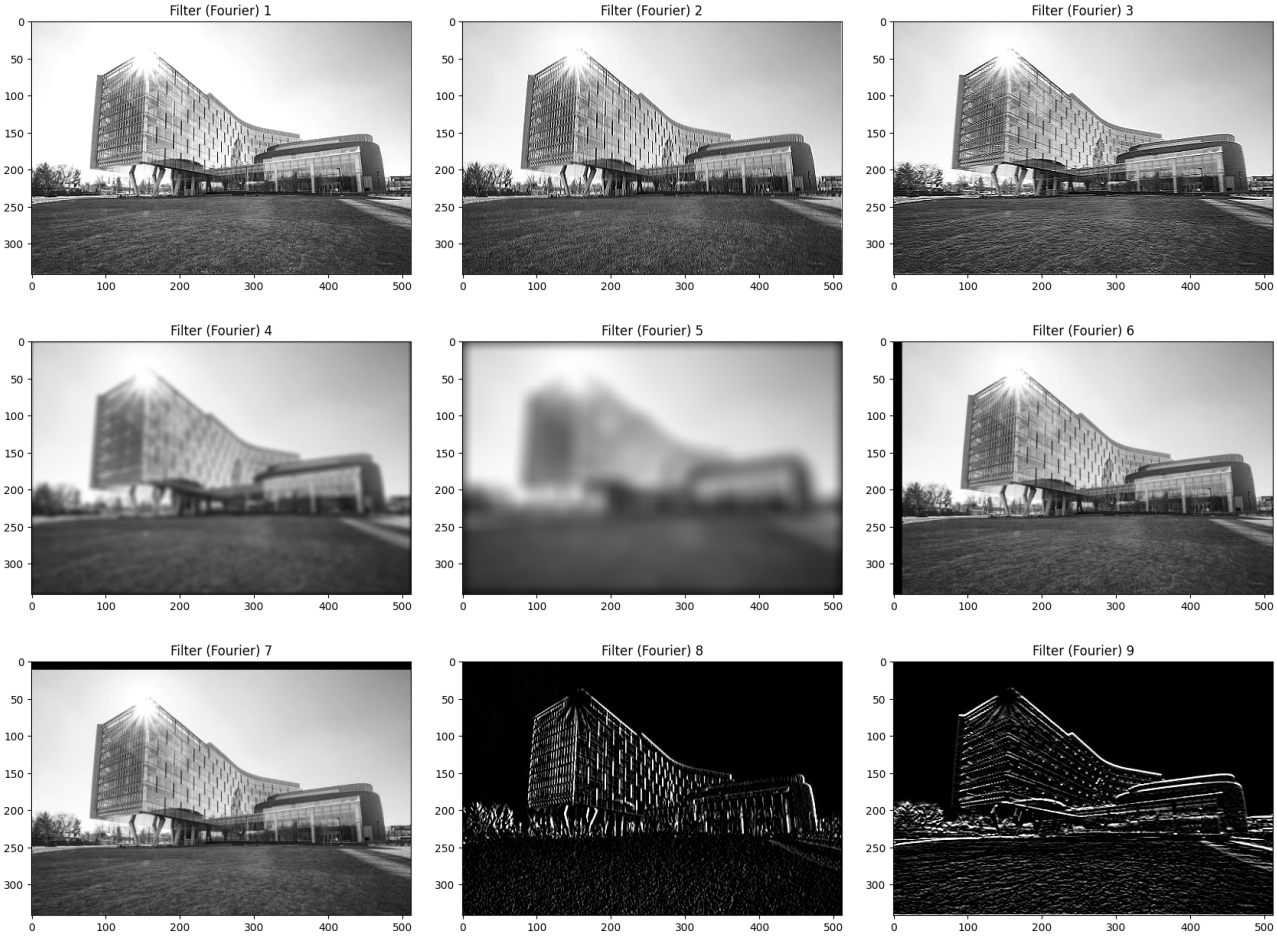
$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

This filter is the sobel x filter which is used for horizontal edge detection which is reflected in the image.
`Time taken: 1.35 s`



Fourier Domain Filtering

For this section, I struggled a little bit with what `np.fft.fft2` and `np.fft.ifft2` did. However, it was actually really easy after referring to the code given in class and reading documentation. I added extra padding so that the sizes are the next power of two. The process simply involves taking the fourier transform of both the image and filter, then performing an elementwise multiplication between the two fourier transformations. Afterwards, we look at the real part of the result and also make to crop everything properly. This resulted in the same convolution we did in the previous parts.



The following is the time it took for each filter between the `myfilter()` and `myfilterFFT()`.

Filter #	Time taken with `myfilter()`	Time taken with `myfilterFFT()`	Δ
1	1.27	0.062	1.208
2	0.48	0.057	0.423
3	0.52	0.099	0.421
4	10.03	0.048	9.982
5	106.47	0.048	106.422
6	3.40	0.048	3.352
7	3.91	0.085	3.825
8	1.36	0.047	1.313
9	1.35	0.056	1.294

It is evident that `myfilterFFT()` accelerated the speed of filter 6 by a great amount compared to the pixel-based filtering.

Fourier Phase and Magnitude

For this part, I used both the textbook to get a general idea of how combining the phase and magnitude of an image would work. However, I also read chapter 4.6 of Digital Image Processing (4e) by Rafael C. Gonzalez and Richard E. Woods. This was really helpful in seeing everything broken down and the exact formula that can be used. I wasn't fully aware that the Fourier transform of the images could be rewritten in polar form in terms of the phase and magnitude of the image:

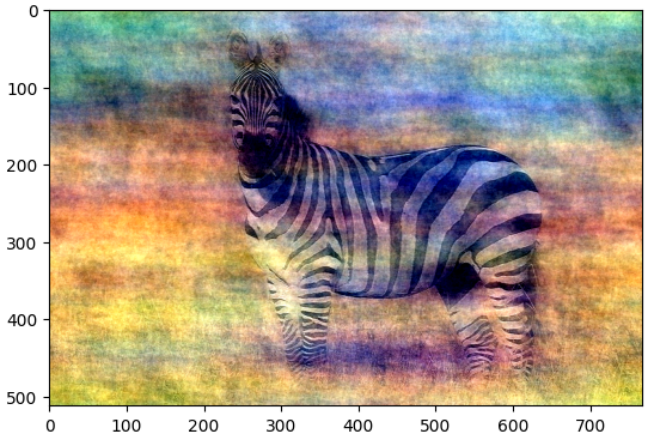
$$F(u,v) = R(u,v) + jI(u,v) = |F(u,v)|e^{j\phi(u,v)} \text{ where}$$

$$\text{magnitude is } |F(u,v)| = [R^2(u,v) + I^2(u,v)]^{\frac{1}{2}}$$

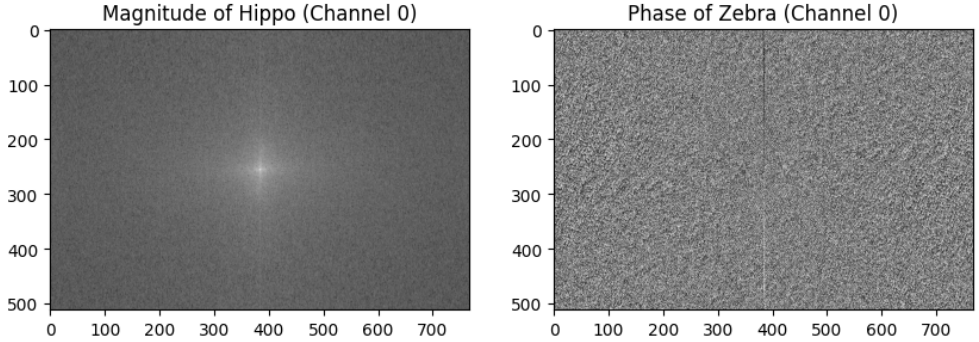
$$\text{phase is } \phi(u,v) = \arctan[\frac{I(u,v)}{R(u,v)}]$$

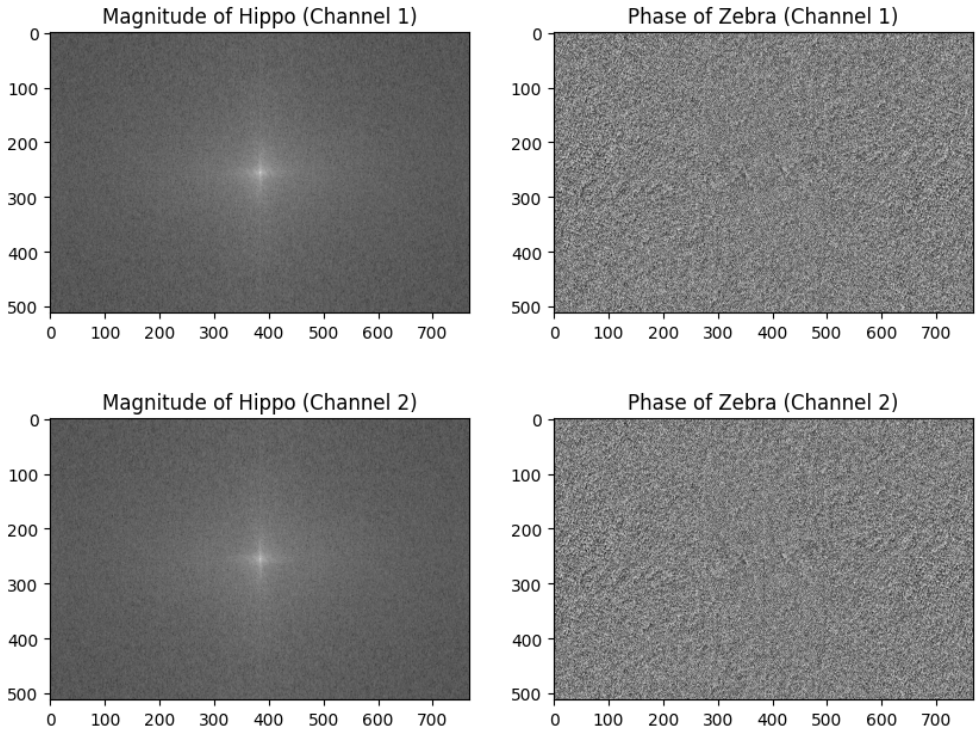
Now if we calculate the magnitude of the hippor image and phase of the zebra image. Then writing the resulting fourier transform in polar form, we are able to call `np.fft.ifft2()` to convert back to an image.

Here was my final product:



The visualizations for the magnitude and phases were also fairly easy after looking at the class slides. Here were the visualizations of the magnitudes and phases for each channel.





Hybrid Images

I spent the most amount of time on this part. After reading the paper on hybrid images, I looked into how I would be able to get a low pass and high pass filter for the hippo and zebra images. However, I couldn't get an image that was not just nonsense. After a few more hours of thinking reading through the paper and textbooks, I saw the hint posted on piazza. I also realized that I didn't fully interpret the directions for this part correctly. I simply had to replace the frequencies of the hippo image with the zebra image. First to get the frequencies, I knew that I would have to take the fourier transform of both images. Throught the piazza post, I learned about `np.fft.fftshift()` and `np.fft.ifftshift` which were so helpful since these it places the lowest frequencies towards the center. Which is exactly what I wanted. What was even interesting was that it also was able to shift everything back.

My final product was this: