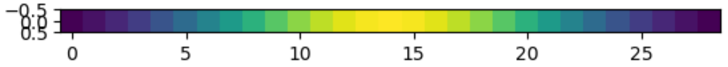
Warning: Do not delete slides.
This includes extra credit slides and any problems you do not complete.
All problems, including extra credit, must be assigned to a slide on Gradescope. Failure to follow this will result in a penalty

CS 6476 Project 1

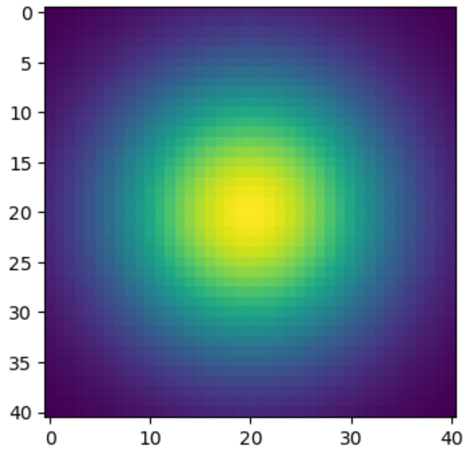
Auryn Yamamura
ayamamura6@gatech.edu
ayamamura6
904154249

Part 1: Image filtering

1D:



2D:



[Describe your implementation of my_conv2d_numpy() in words. Make sure to discuss padding, and the operations used between the filter and image.]

I used np.pad to pad the edges of the image with zeros (to be precise, I added k-1 zeros to the height, and j-1 zeros to the width).

Afterwards I used a triple for-loop to slide my filter across the image, multiplying elementwise and storing the sum to a new image called ”filtered_img”.

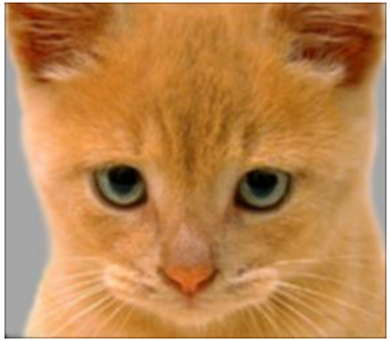
Part 1: Image filtering

Identity filter

Briefly Explain what this filter did:

This filter return the identity of the image (aka the input image).

Small blur with a box filter

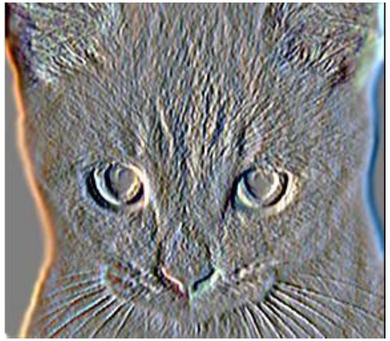


Briefly Explain what this filter did:

This filter slightly blurred the image by removing some high frequencies present in the image.

Part 1: Image filtering

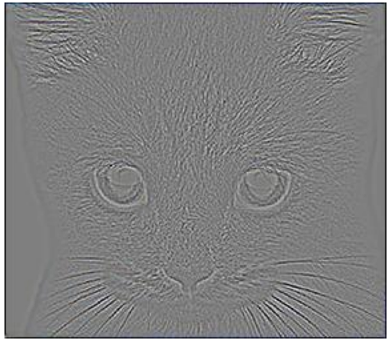
Sobel filter



Briefly Explain what this filter did:

This filter takes a 2D gradient in the x-direction, thus performing vertical edge detection.

Discrete Laplacian filter



Briefly Explain what this filter did:

This filter also performs a 2D gradient with no bias to direction, resulting in general edge detection.

Part 1: Hybrid images

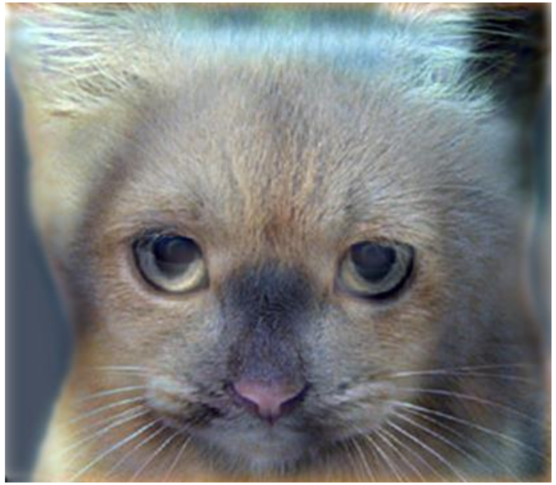
[Describe the three main steps of create_hybrid_image() here. Explain how to ensure the output values are within the appropriate range for matplotlib visualizations.]

1. Extract low frequencies from image a by convolving with Gaussian filter (i.e. blurring).

2. Extract high frequencies from image b by subtracting the result of our Gaussian blur from the input image

3. Add the low and high frequencies of images a and b together and clip the value between 0 and 1 using `np.clip`.

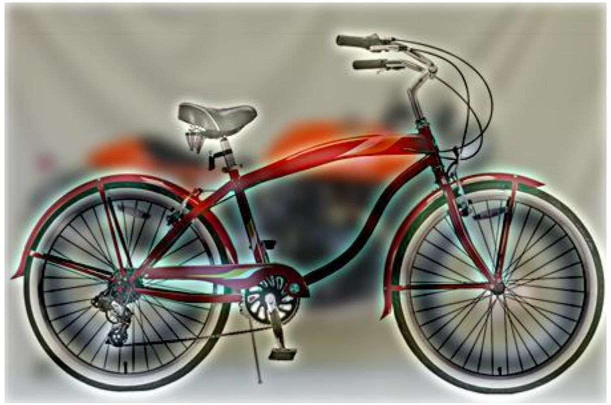
Cat + Dog



Cutoff frequency: 6

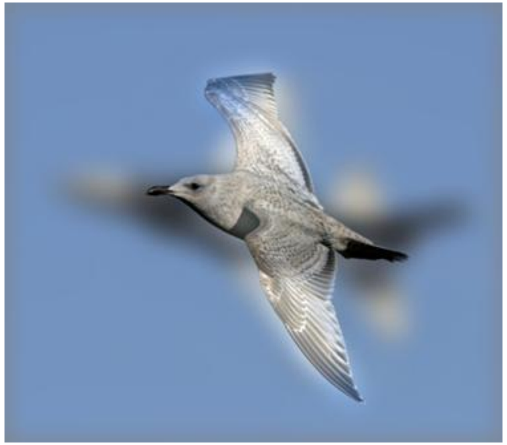
Part 1: Hybrid images

Motorcycle + Bicycle



Cutoff frequency: 5

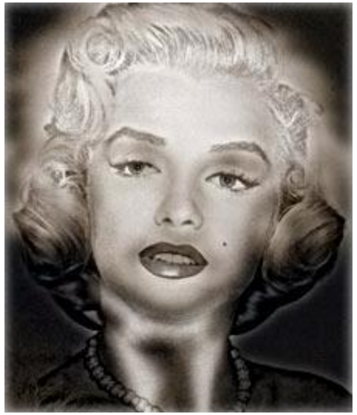
Plane + Bird



Cutoff frequency: 10

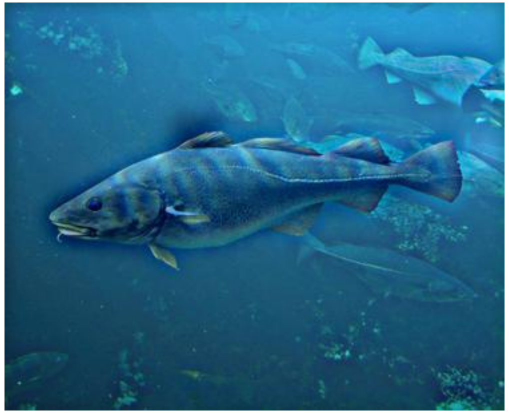
Part 1: Hybrid images

Einstein + Marilyn



Cutoff frequency: 4

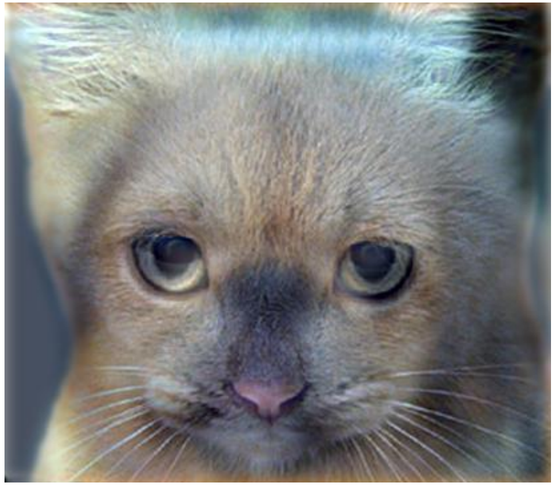
Submarine + Fish



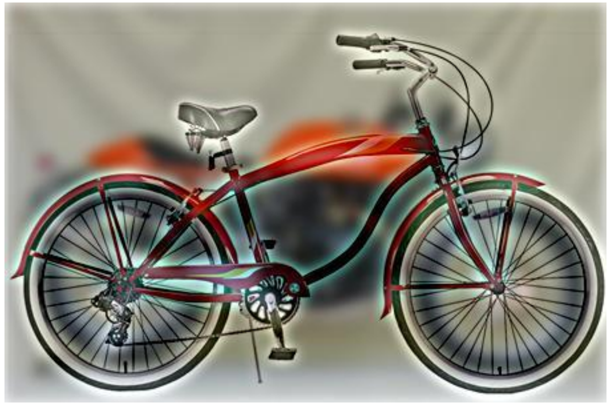
Cutoff frequency: 7

Part 2: Hybrid images with PyTorch

Cat + Dog

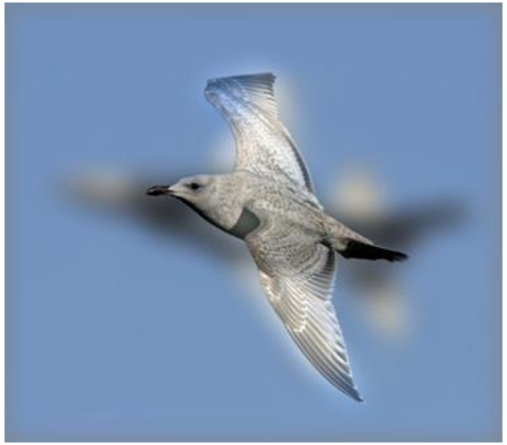


Motorcycle + Bicycle

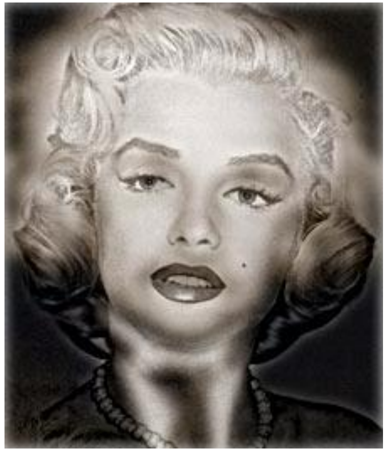


Part 2: Hybrid images with PyTorch

Plane + Bird

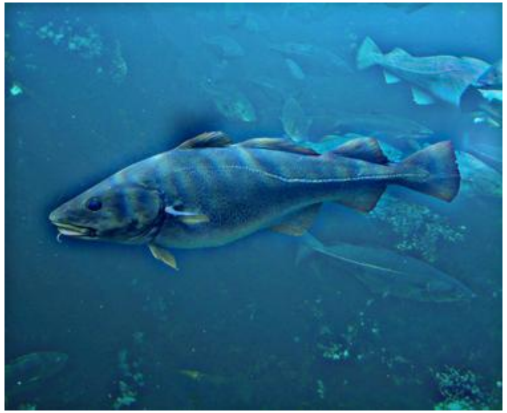


Einstein + Marilyn



Part 2: Hybrid images with PyTorch

Submarine + Fish



Part 1 vs. Part 2

[Compare the run-times of Parts 1 and 2 here, as calculated in project-1.ipynb. Which method is faster?]

Part 1 Runtime: 6.744 seconds

Part 2 Runtime: 2.313 seconds

It is clear that the Pytorch implementation is noticeably faster than the naive NumPy implementation.

Part 3: Understanding input/output shapes in PyTorch

[Consider a 1-channel 5x5 image and a 3x3 filter. What are the output dimensions of a convolution with the following parameters?

Stride	Padding	Output Shape
s = 1	p = 0	(1, 3, 3)
s = 2	p = 0	(1, 2, 2)
s = 1	p = 1	(1, 5, 5)
s = 2	p = 1	(1, 3, 3)

We know that $d_2 = N$ (batch size) and

$$h_2 = \frac{h_1 - k + 2p}{s} + 1, \qquad w_2 = \frac{w_1 - k + 2p}{s} + 1$$

so using these formulas we get the results above.

[What are the input & output dimensions of the convolutions of the dog image and a 3x3 filter with the following parameters:

Stride	Padding	Output Shape
s = 1	p = 0	(1, 359, 408)
s = 2	p = 0	(1, 180, 204)
s = 1	p = 1	(1, 361, 410)
s = 2	p = 1	(1, 181, 205)

There is only one filter applied to the dog image so $N = 3$ (since batch size = # of filters * # of channels). The dog image has shape ($d_1 = 3$, $h_1 = 361,\ w_1 = 410$) and since the filter is 3x3, $k = 3$. Using the formulas provided we get the above results.

Part 3: Understanding input/output shapes in PyTorch

[How many filters did we apply to the dog image?]

Arguably this question be answered in two ways: Technically we applied 12 filters to the dog image, resulting in 4 altered images.

However, one could argue we applied 4 filters to the image since we needed 3 filters each due to the number of input channels.

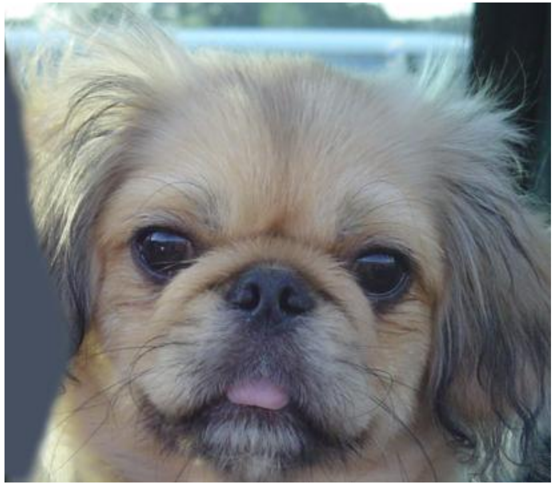
[Section 3 of the handout gives equations to calculate output dimensions given filter size, stride, and padding. What is the intuition behind this equation?]

The intuition is the same across each dimension, so we shall only pick one to demonstrate (h): Assuming $s = 1$ and $p = 0$, we can slide our filter at most $h_1 - (h_k - 1) = h_1 - h_k + 1$ times across the image so that would be the value of h_2.

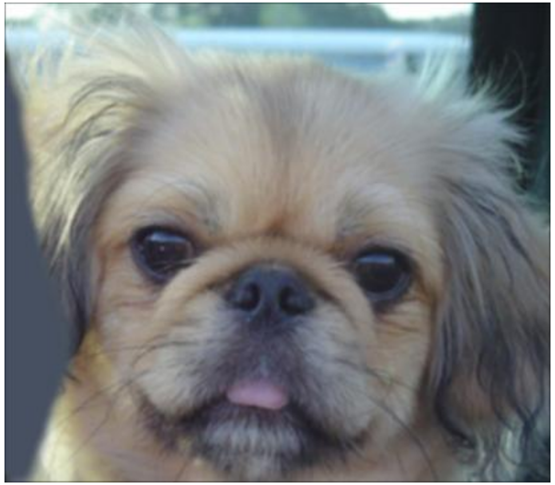
If $p > 1$, then we add p pixels to both side, resulting in $h_2 = h_1 + 2p - (h_k - 1)$.

If $s > 1$, then we need to re-adjust by dividing both the height of our padded image ($h_1 + 2p$) and the height of our filter (h_k) by s, resulting in new heights $h_1' = \frac{h_1+2p}{s}$, $h_k' = \frac{h_k}{s}$. Now we can use our formula from before to get $h_2 = h_1' - h_k' + 1$. If we plug in the values for h_1' and h_k', we get the formula from the handout.

Part 3: Understanding input/output shapes in PyTorch

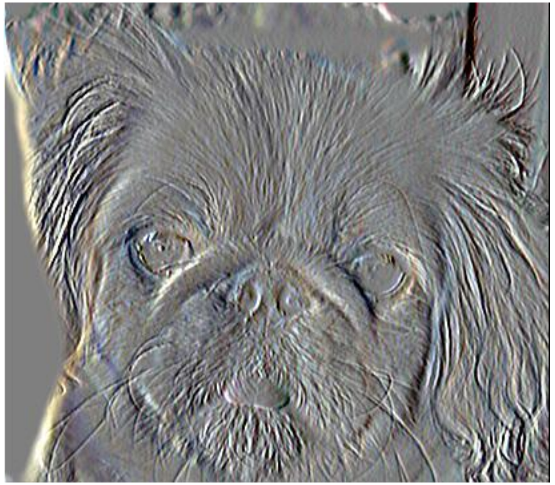


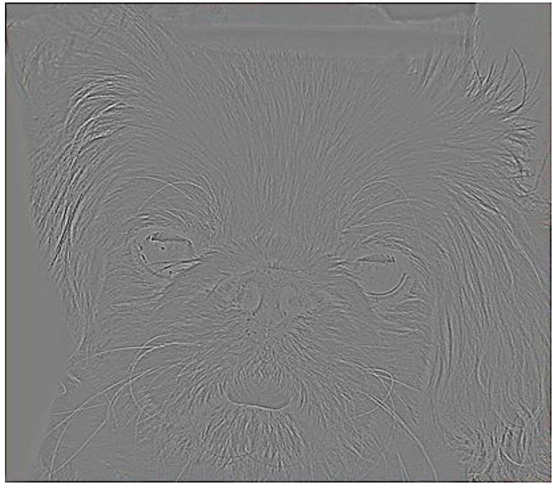
Identity Filter



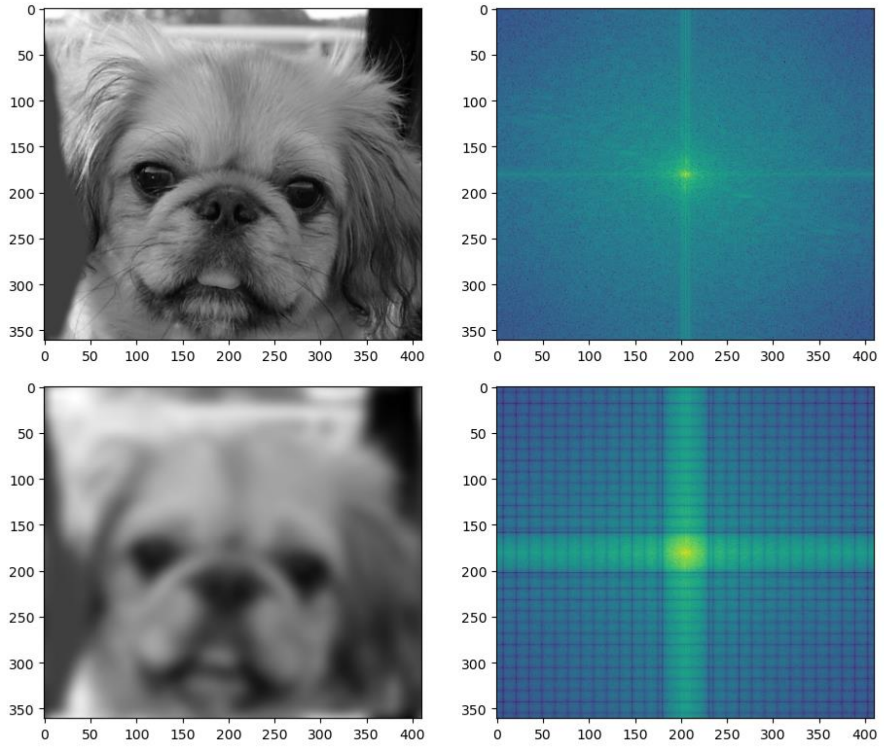
Blur Filter

Part 3: Understanding input/output shapes in PyTorch


Sobel Filter


Laplacian Filter

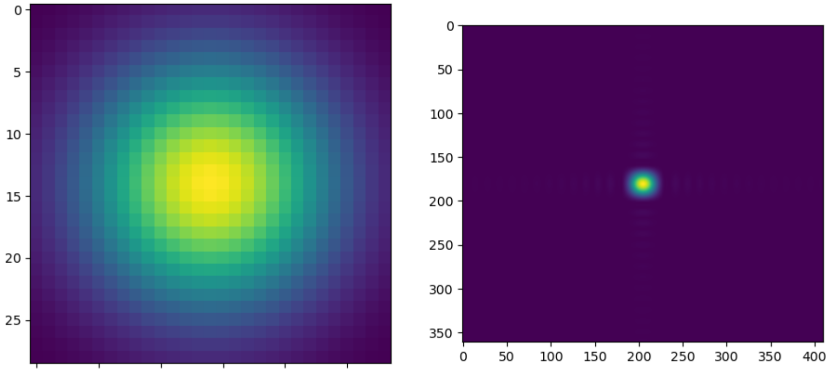
Part 4: Frequency Domain Convolutions (extra credit)

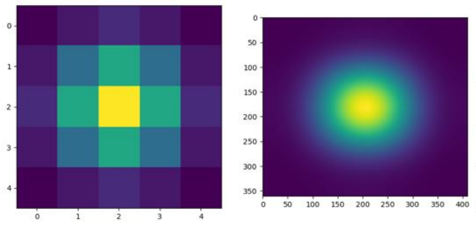


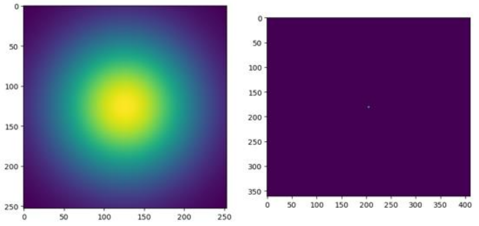
For the original dog image, you can see that a lot of high frequencies are used, but also the frequencies along the x-/y-axes are stronger because of sharp vertical/horizontal edges.

For the blurred image, we lose a lot of edges and as well as our higher frequencies, which causes the grid-like structure of zero-values for high frequencies. Additionally, since the vertical/horizontal edges have become a lot smoother, we need more high frequencies than before along the x-/y-axes to recreate the now smoothed edges.

Part 4: Frequency Domain Convolutions (extra credit)


Cutoff Frequency = 7


Cutoff frequency = 1


Cutoff frequency = 63

Strangely, I found that lowering the cutoff frequency allowed for more high frequencies to be preserved in the dog image (whereas raising the cutoff frequency restricted the amount of high frequencies). Based off what I've heard in lecture + read online, I would expect the opposite to occur.

I wonder if this is because of how we calculated k and sigma for creating our Gaussian kernels or maybe how the filter size changed with cutoff frequency.

If you have feedback, do let me know. My code passed all tests on Gradescope though.

Part 4: Frequency Domain Convolutions (extra credit)

The Convolution Theorem can mathematically expressed as the following:

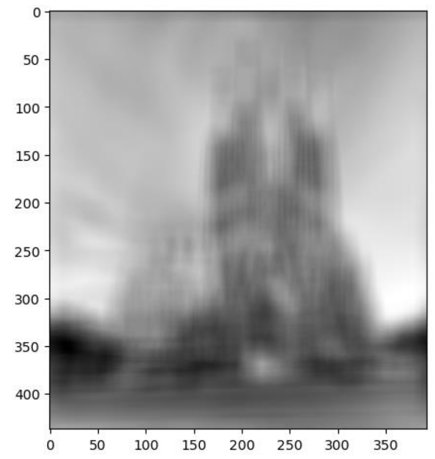
$$F[g * h] = F[g]F[h]$$
$$g * h = F^{-1}[F[g]F[h]]$$

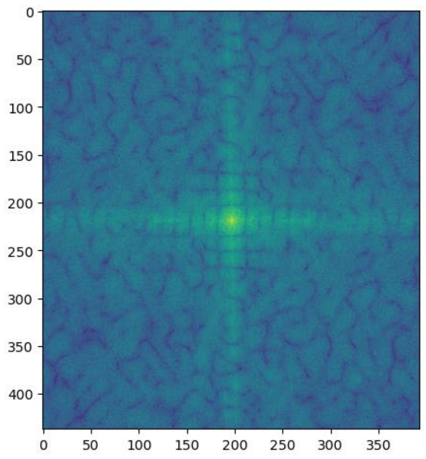
Supposing that g represents our filter and h our image, then the Convolution Theorem states that we can convolve our filter and image by taking the inverse FT of the product of Fourier transformed g/h (i.e the elementwise product of their frequency representations).

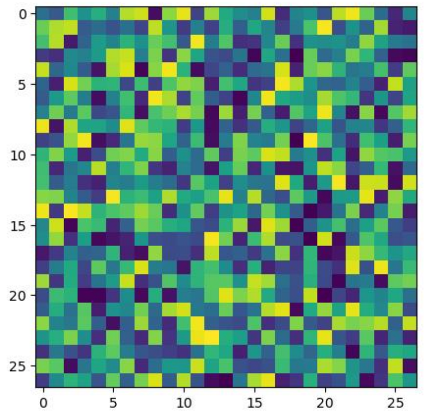
We can use the Convolution Theorem to also perform deconvolutions – instead of multiplying our frequency representations, we instead element-wise divide the frequencies our convolved image by the frequencies of our filter:

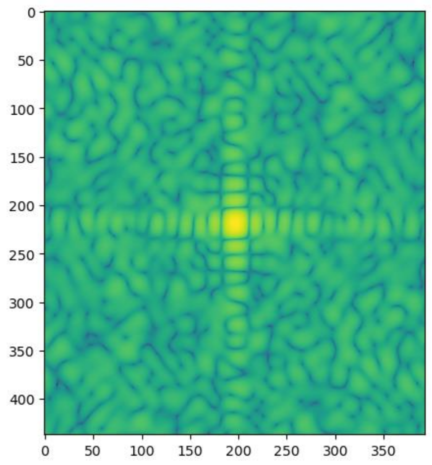
$$h = F^{-1}[\frac{F[g * h]}{F[g]}]$$

Part 4: Frequency Domain Convolutions (extra credit)

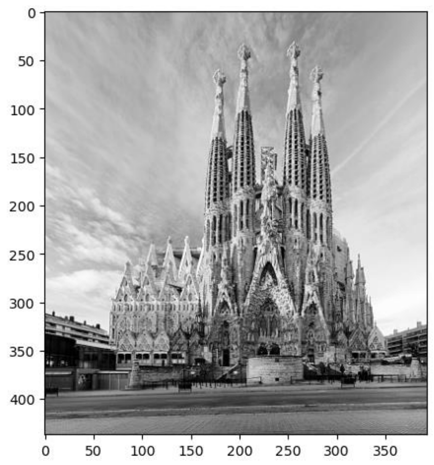


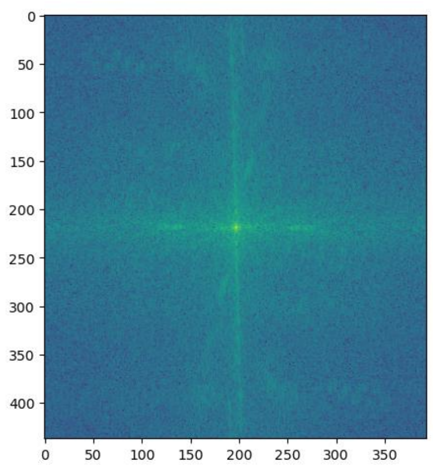


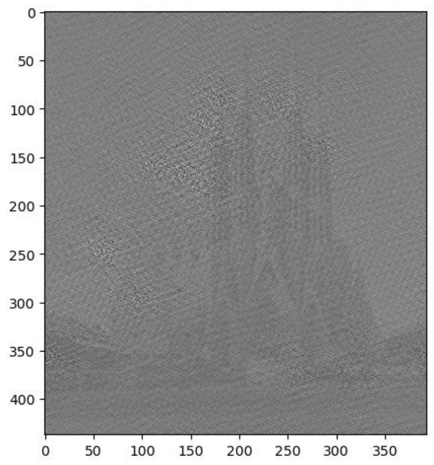


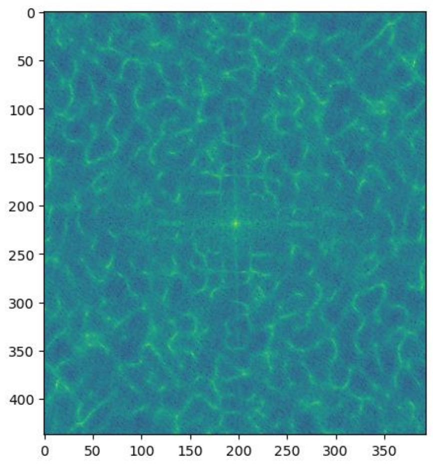


Part 4: Frequency Domain Convolutions (extra credit)









Part 4: Frequency Domain Convolutions (extra credit)

[What factors limit the potential uses of deconvolution in the real world? Give two possible factors]

1. The presence of noise (perhaps due to image compression) leaves us with high frequency artifacts that remain in the deconvolved image. This requires strong regularization (i.e. de-noising).
2. We often don’t know what filter was used to create our image.

[Describe any structures found in the frequency domain of the mystery image and explain what it’s caused by.]

Assuming this question is asking about the mystery image prior to deconvolution – similar to the blurred dog image, we can see that the frequencies along the x/y-axes are stronger than rest (though not significantly so) indicating the presence of vertical/horizontal “edges.”

We can also see a more organic pattern with dips, alluding to the noise added by the random filter used in convolution.

Conclusion

Assume that we extract the low frequencies from image a and the high frequencies from image b.

Lowering the cutoff frequency results more image a being more present, whereas raising the cutoff frequency preserves more of image b. I’m not sure if this is meant to be intended result though.

Image b will be more prominent in a larger/closer image, whereas image a is prominent in a smaller/farther image. Swapping the images will swap these effects.