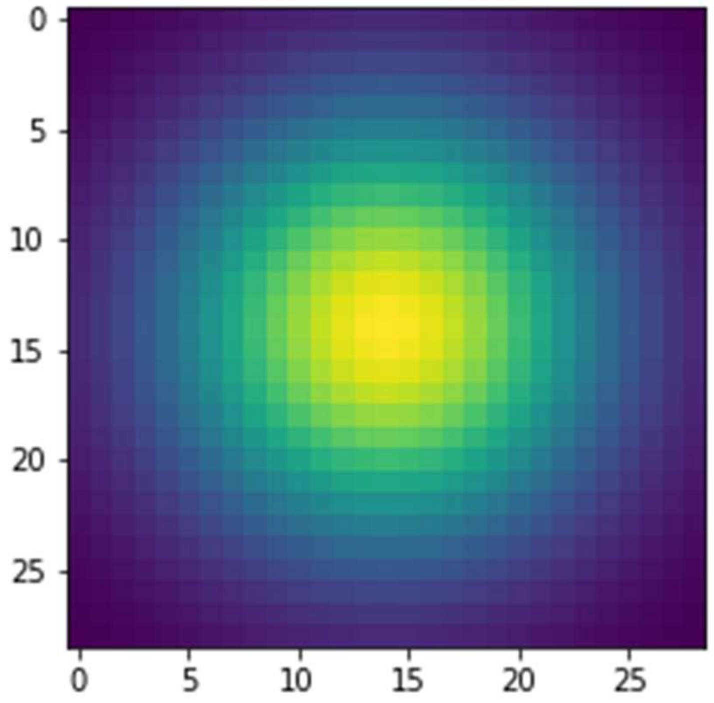
CS 6476 Project 1

Cameron Potter
cgpotter@gatech.edu
cpotter8
903465425

Part 1: Image filtering



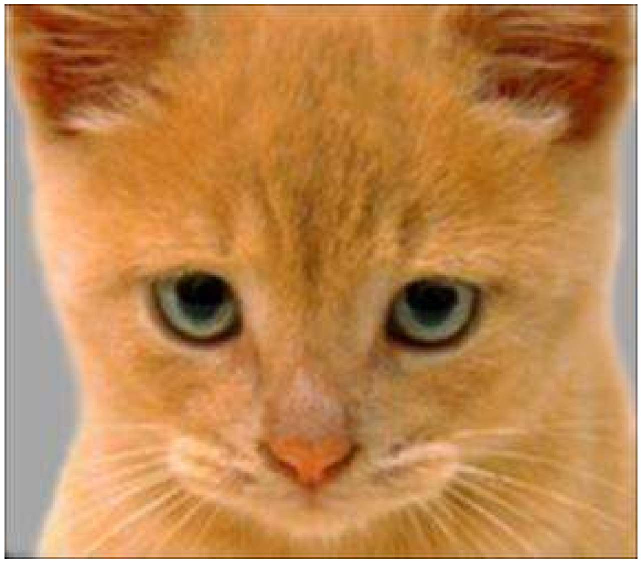
First, we put the dimensions of the image (m, n, c) and filter (k, j) into variables. Then, we need to pad the input image so that the filter does not reduce resolution. We are given that the filter dimensions are odd, so that means the image must be padded with floor(k/2) and floor(j/2) zeros to get new dimensions (m + k – 1, n + k – 1, c). We then can stack the filter on top of itself c times to apply to each layer of the image. We then simply iterate over the pixels and sum of the element-wise multiples of the filter and the padded image with the equation:

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

Part 1: Image filtering

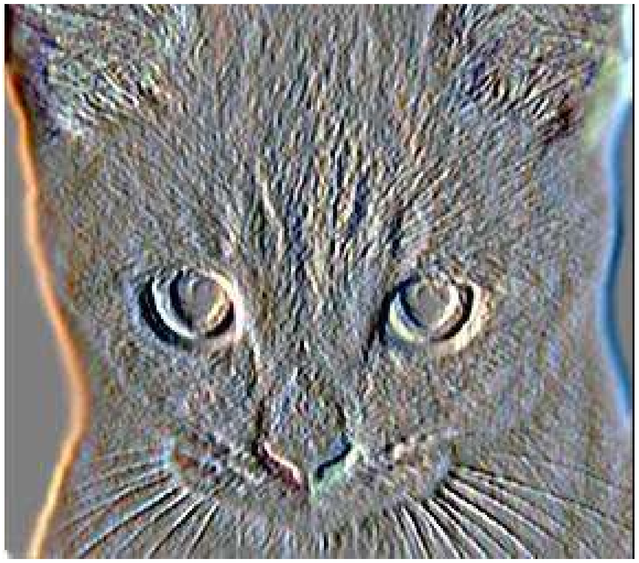
Identity filter

Small blur with a box filter

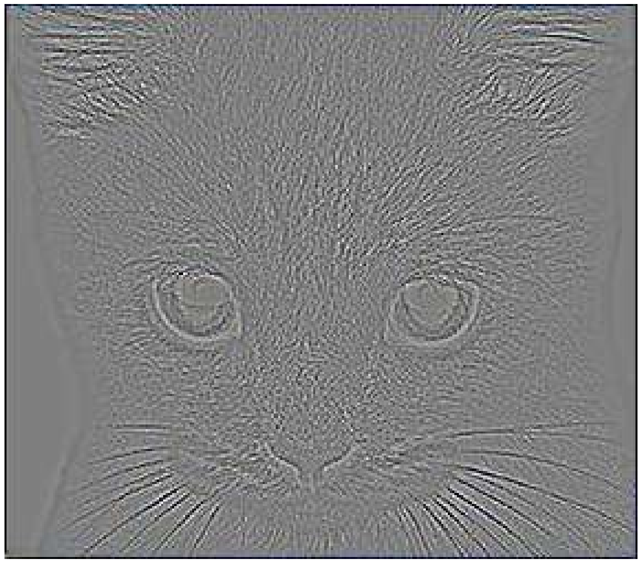


Part 1: Image filtering

Sobel filter



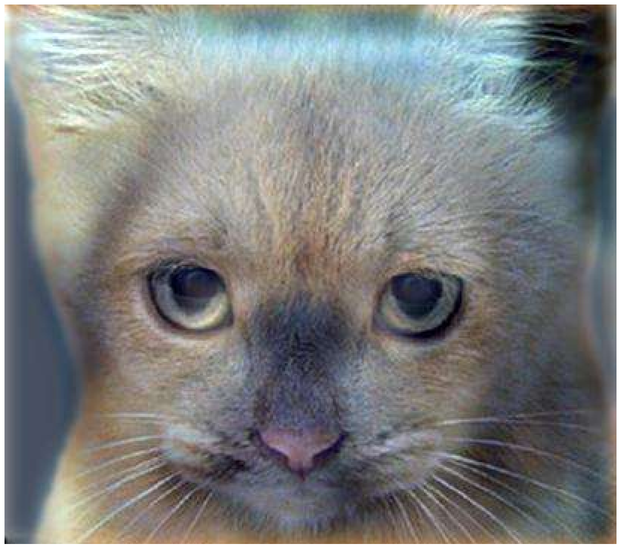
Discrete Laplacian filter



Part 1: Hybrid images

First, we must get the low frequencies from image1 with the given lowpass filter using my_conv2d_numpy. Next, we get the high frequencies from image2 by subtracting its low frequencies from the original image. Finally, we add the low frequencies from image1 and the high frequencies of image2 to get our hybrid image. We can utilize np.clip to ensure our hybrid image has values in [0, 1].

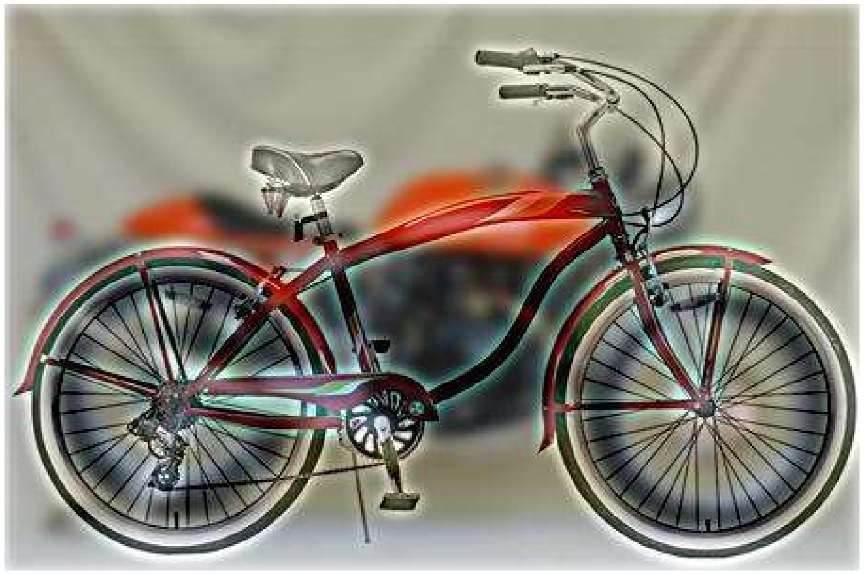
Cat + Dog



Cutoff frequency: 7

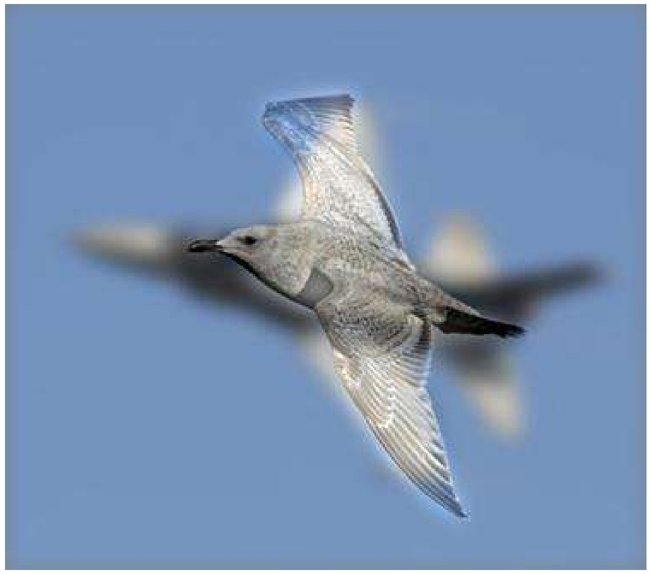
Part 1: Hybrid images

Motorcycle + Bicycle



Cutoff frequency: 4

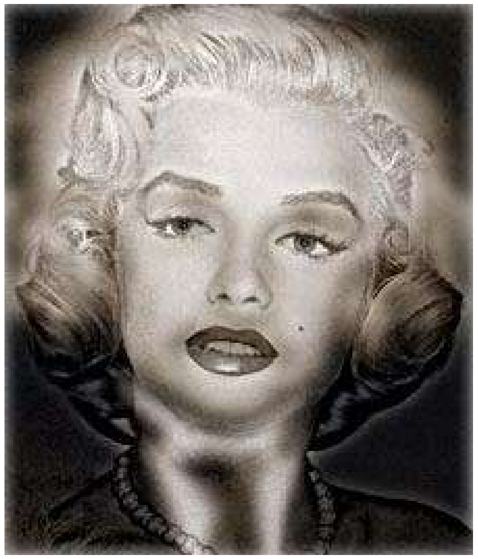
Plane + Bird



Cutoff frequency: 6

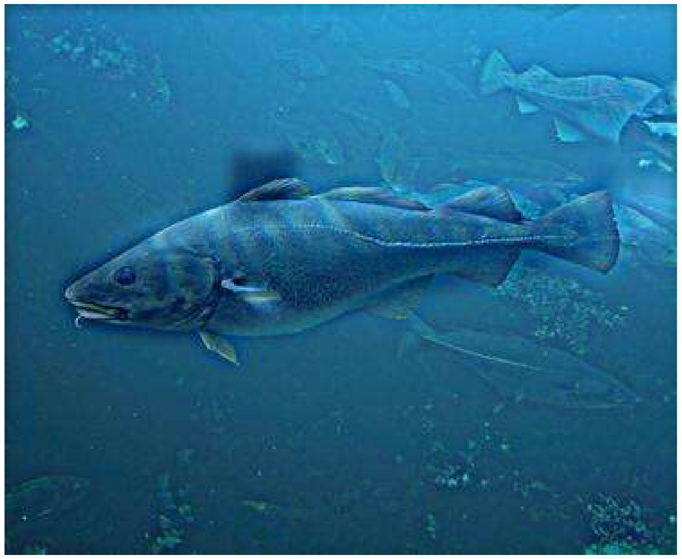
Part 1: Hybrid images

Einstein + Marilyn



Cutoff frequency: 4

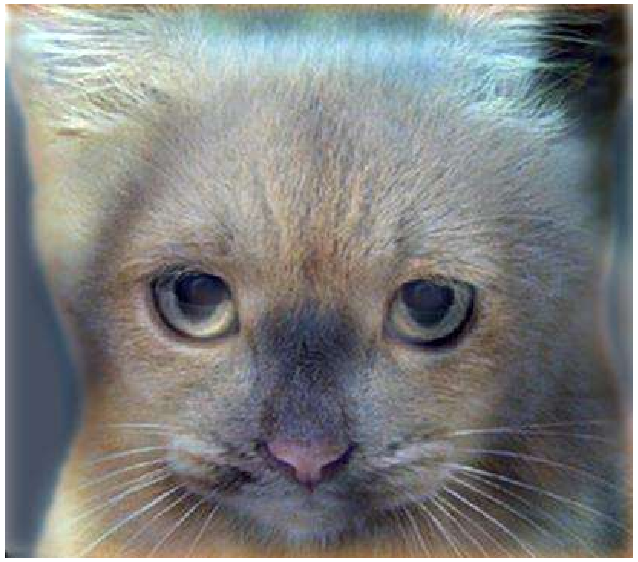
Submarine + Fish



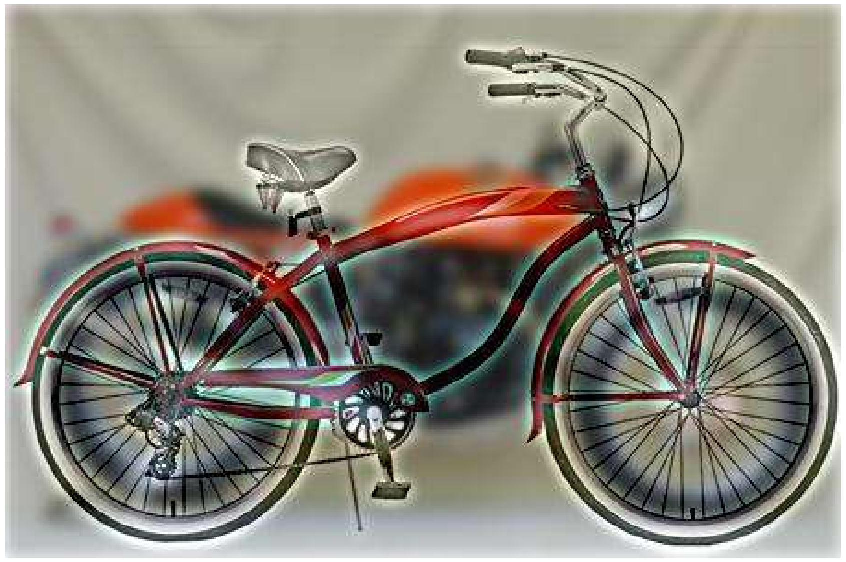
Cutoff frequency: 4

Part 2: Hybrid images with PyTorch

Cat + Dog

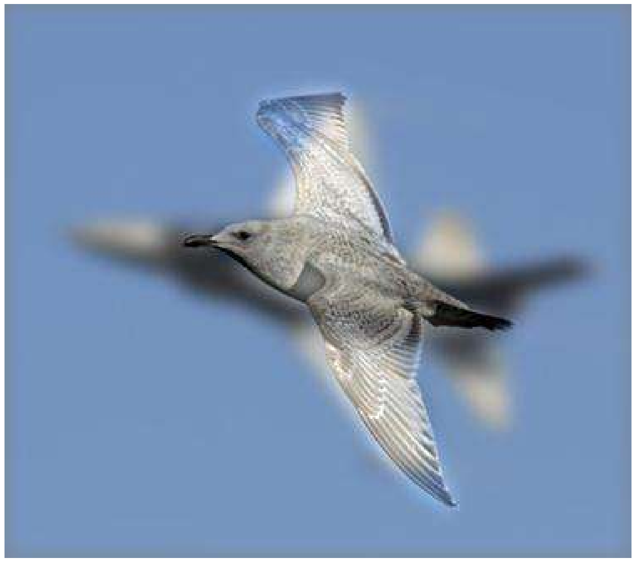


Motorcycle + Bicycle

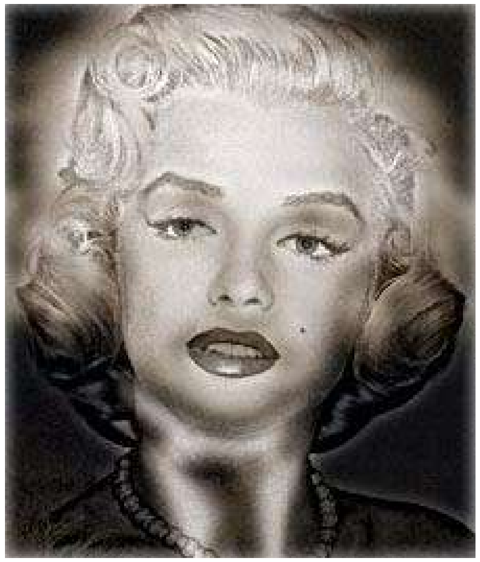


Part 2: Hybrid images with PyTorch

Plane + Bird

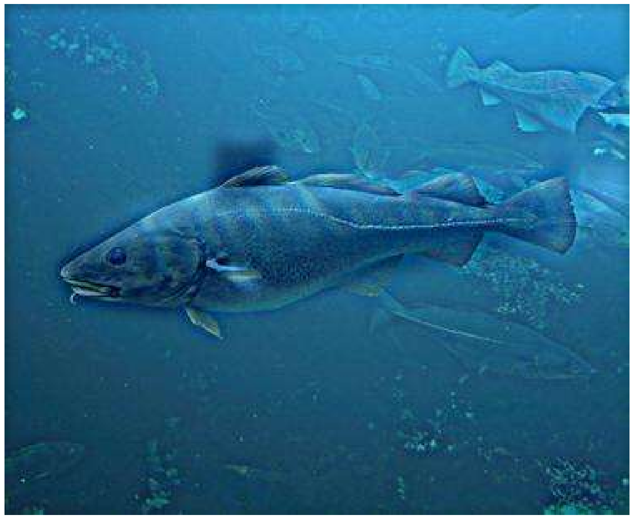


Einstein + Marilyn



Part 2: Hybrid images with PyTorch

Submarine + Fish



Part 1 vs. Part 2

Part 1: 7.817 seconds

Part 2: 1.109 seconds

Clearly, Part 2 using PyTorch is faster

Part 3: Understanding input/output shapes in PyTorch

An image input into my_conv2d_pytorch is a tensor of shape (1, d1, h1, w1), and a kernel input into my_conv2d_pytorch is a tensor of shape (N, d1/groups, k, k). The function will then output an image in the form of a tensor of shape (1, d2, h2, w2) where d2 = N, h2 = (h1 - k + 2 * padding) / stride + 1, and w2 = (w1 - k + 2 * padding) / stride + 1. Hence, if d1=1, h1=w1=5, N=1, and k=3, then the output shapes will be:

- Stride = 1, padding = 0 → output= (1, 1, 3, 3)
- Stride = 2, padding = 0 → output= (1, 1, 2, 2)
- Stride = 1, padding = 1 → output= (1, 1, 5, 5)
- Stride = 2, padding = 1 → output= (1, 1, 3, 3)

If d1=3, h1=361, w1=410, N=1, and k=3, then the output shapes will be:

- Stride = 1, padding = 0
  - → output = (1, 1, 359, 408)
- Stride = 2, padding = 0
  - → output = (1, 1, 180, 204)
- Stride = 1, padding = 1
  - → output = (1, 1, 361, 410)
- Stride = 2, padding = 1
  - → output = (1, 1, 181, 205)

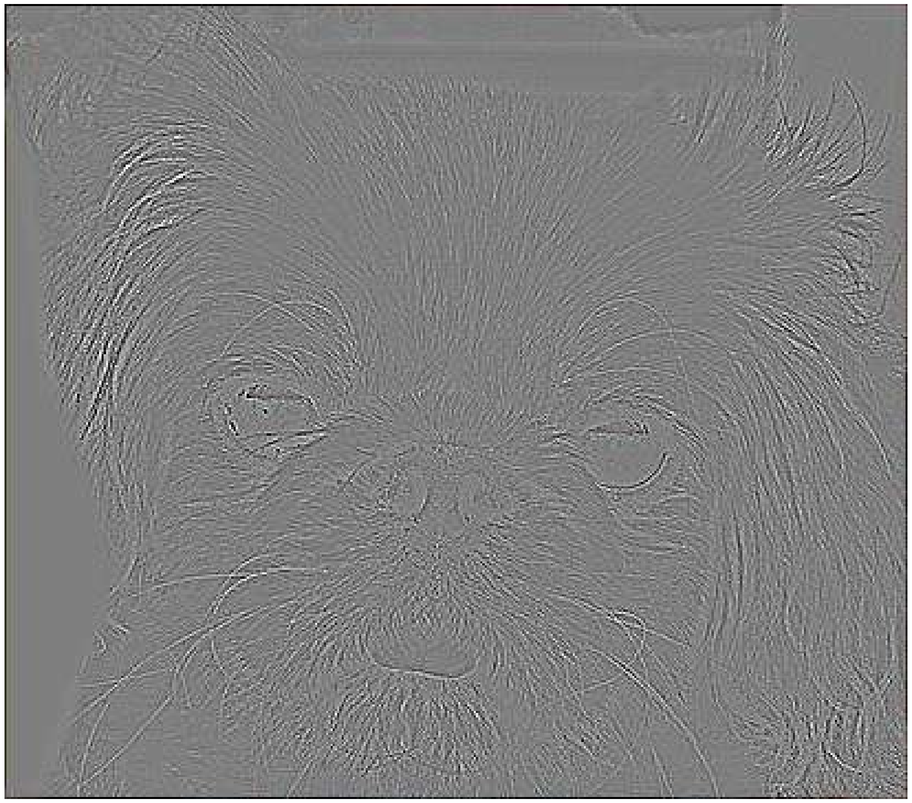
Part 3: Understanding input/output shapes in PyTorch

We applied 4 filters to the dog image.

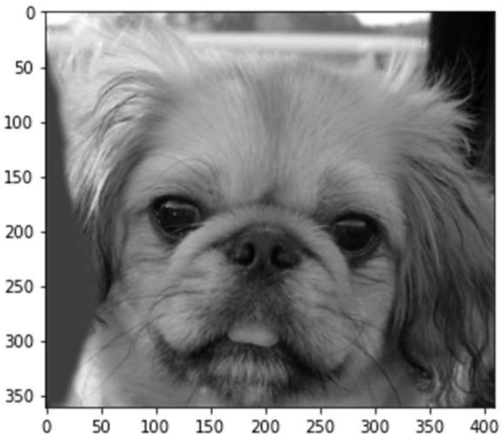
The output has shape (1, d2, h2, w2) where d2 = N, h2 = (h1 - k + 2 * padding) / stride + 1, and w2 = (w1 - k + 2 * padding) / stride + 1. For h2, we realize that the filter of size k will eliminate floor(k/2) pixels on each side, so we subtract k from the original size. Each increment in padding with increase the number of pixels on each side by 2*padding, and thus we add that to our expression. The stride shows how many pixels we will skip over, so we must divide by the proportion of pixels skipped. The plus one outside the fraction is there because k is odd, and thus we will always maintain the middle pixel of the filter, but we subtracted it in the fraction, so we must add it back.

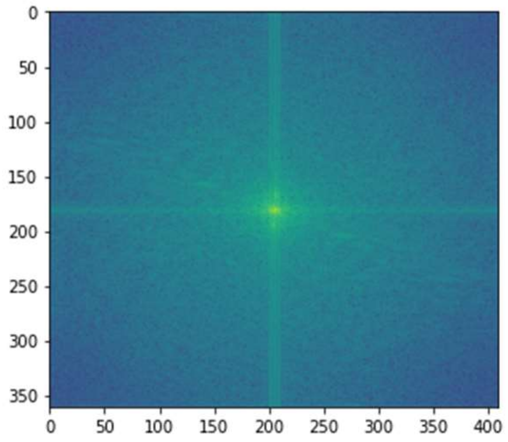
Part 3: Understanding input/output shapes in PyTorch

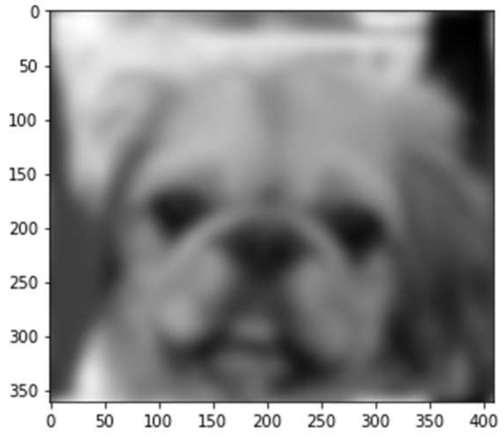
Part 3: Understanding input/output shapes in PyTorch

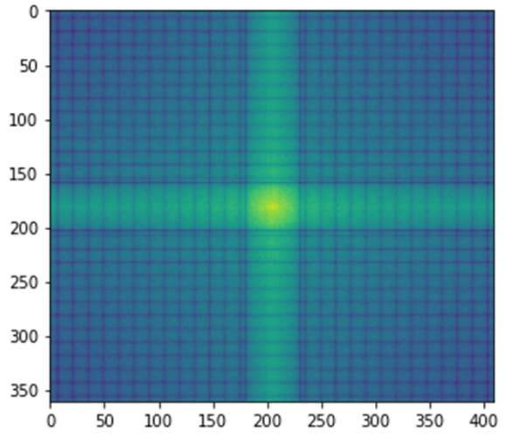


Part 4: Frequency Domain Convolutions

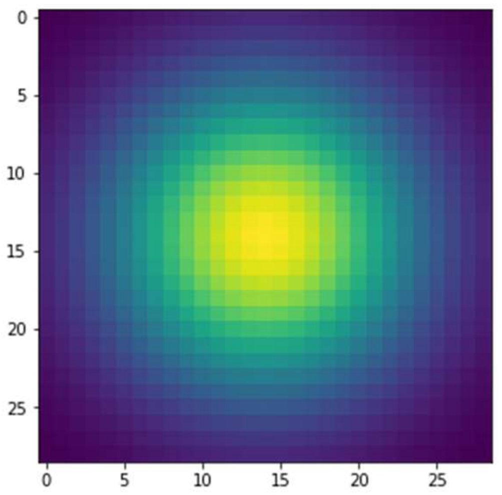


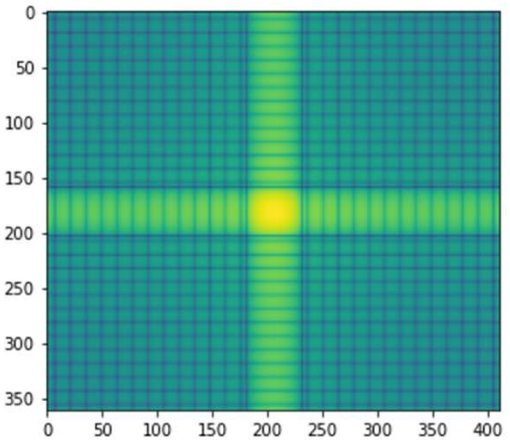






Part 4: Frequency Domain Convolutions





The frequency domain represents the change in intensity from one pixel to the next. Hence, the frequency domain will essentially visualize the gradient of the Gaussian kernel, which is not identical to the Gaussian.

If the goal was to make the images more similar, a possible adjustment would be to move closer to the continuous domain in the original spatial image. This could be done b by taking more samples, as we currently have a few discrete samples of a continuous function. Further padding would also help, as the “zooming out” would have the frequencies change less and model the Gaussian closer.

Part 4: Frequency Domain Convolutions

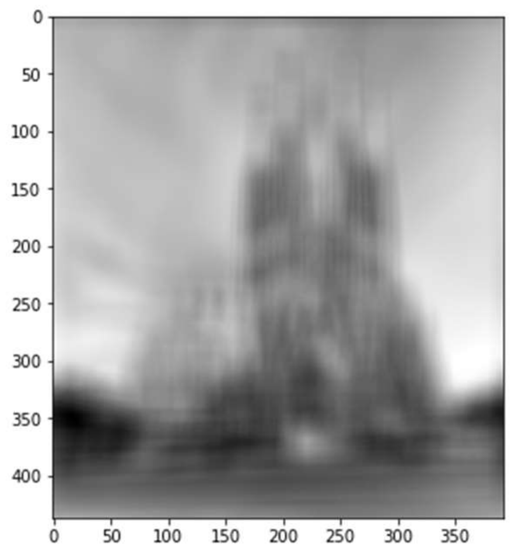
The convolution theorem states that the Fourier transform of the convolution of two functions is the product of their Fourier transforms, and convolution in spatial domain is equivalent to multiplication in frequency domain, typically shows by the equations:

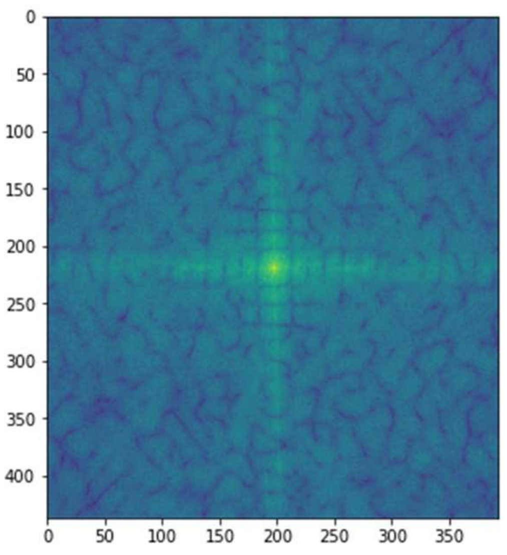
$$F[g * h] = F[g]F[h]$$

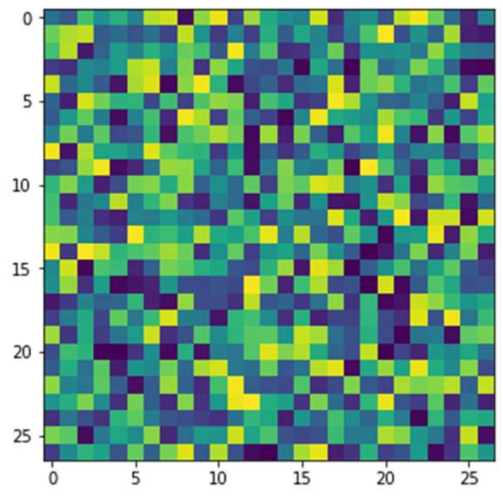
$$g * h = F^{-1}[F[g]F[h]]$$

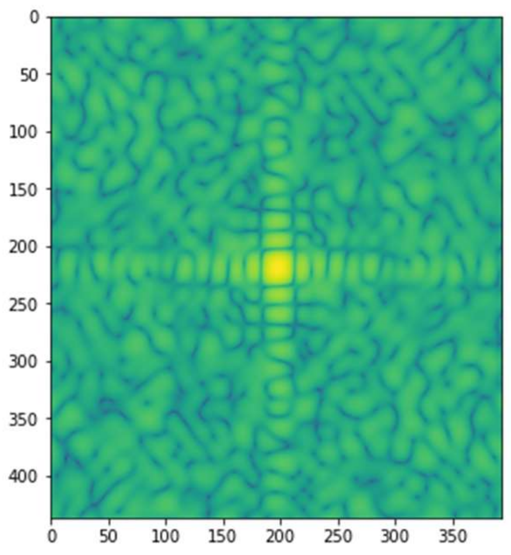
This helps with deconvolution, as if g is the image and h is the filter, if we are given g*h and h, we can use these equations to show g = F^{-1}[(F[g*h]/[F[h]) to get the deconvolution.

Part 4: Frequency Domain Convolutions

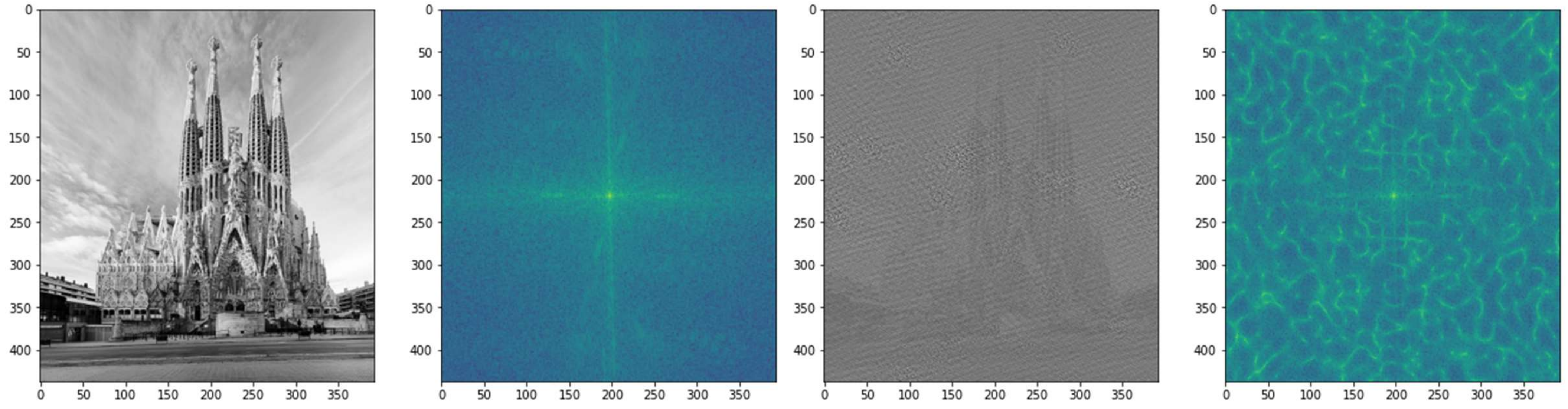








Part 4: Frequency Domain Convolutions



Part 4: Frequency Domain Convolutions

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

1. Uncertainty of filters. We cannot always know the exact kernel that was that led to the given image
2. Noise. Any real measurement will have some noise, affecting the results of the deconvolution

The two convolutions are approximately the same result, as they are both slightly blurring the dog image.

When in the frequency domain, we had to use a one-layer (black and white) image versus the color image in the spatial domain. The efficiency of both algorithms also differ.

Conclusion

Varying the cutoff frequency changes which image is more visible at different scales. A lower cutoff frequency will favor the low frequency image to make it more visible at larger scales.

Swapping the images will change which one has high frequencies and which one has low frequencies, and thus the scales at which one dominates the other flip as well.