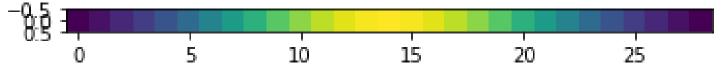
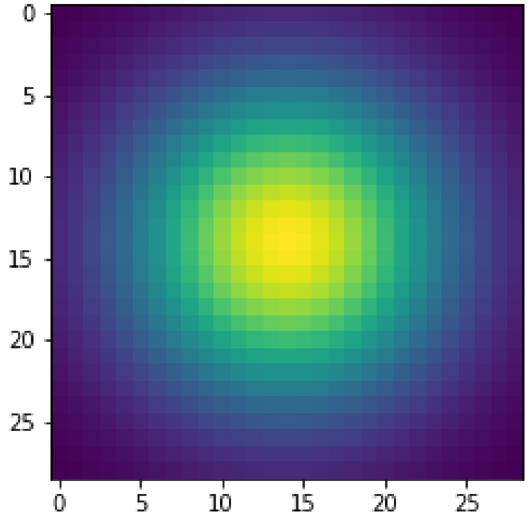
CS 6476 Project 1

Pooja Pache
ppache3@gatech.edu
ppache3
903813050

Part 1: Image filtering



1D Gaussian Kernel



2D Gaussian Kernel

1. Split the input image into 3 matrices as per the channels: red, blue and green.

2. Create 3 new zero matrices: redTemp, blueTemp, greenTemp of the same size as the above matrices.

3. Set the padding for top and bottom as number of kernel rows//2 and for left and right as number of kernel rows//2.

4. Pad the above 3 matrices with mode = constant(default constant_value = 0)

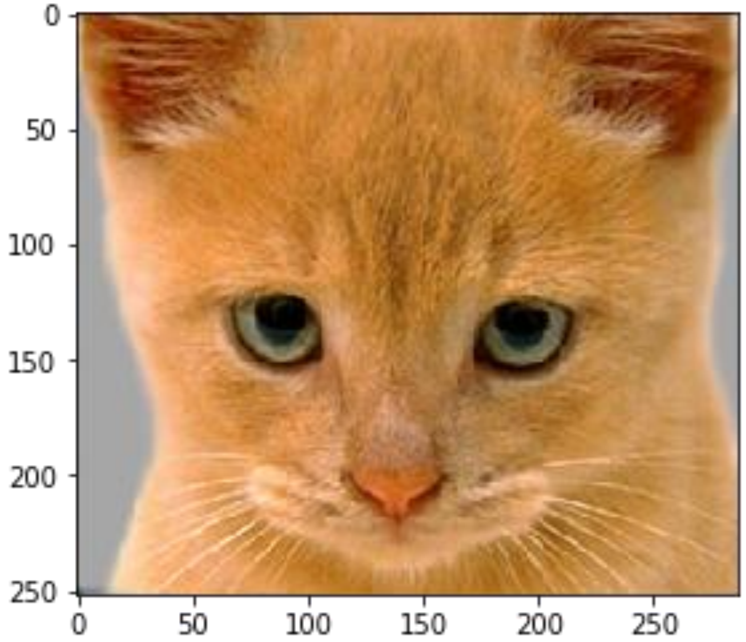
5. Use a nested loop to multiply the sub-matrix of the image(kernel size x kernel size) and the kernel.

6. Store the sum of multiplied values in their respective new matrices.

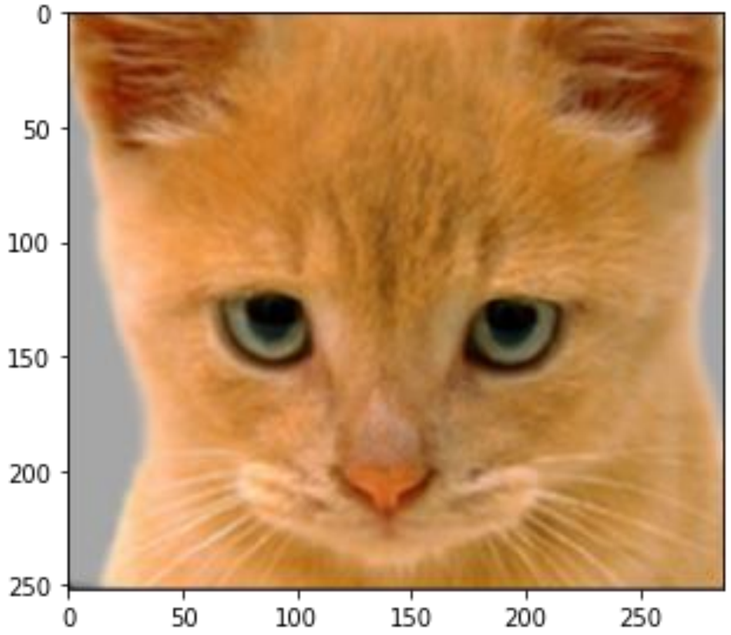
7. Finally, after looping the kernel over the entire padded image, combine the 3 new matrices redTemp, blueTemp, greenTemp using np.dstack() and store this value in the filtered image variable.

Part 1: Image filtering

Identity filter

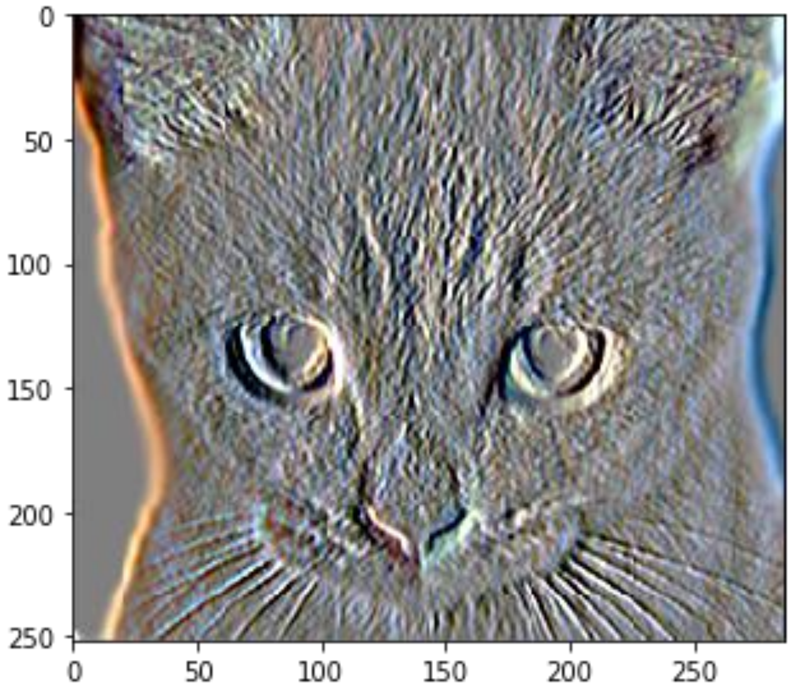


Small blur with a box filter

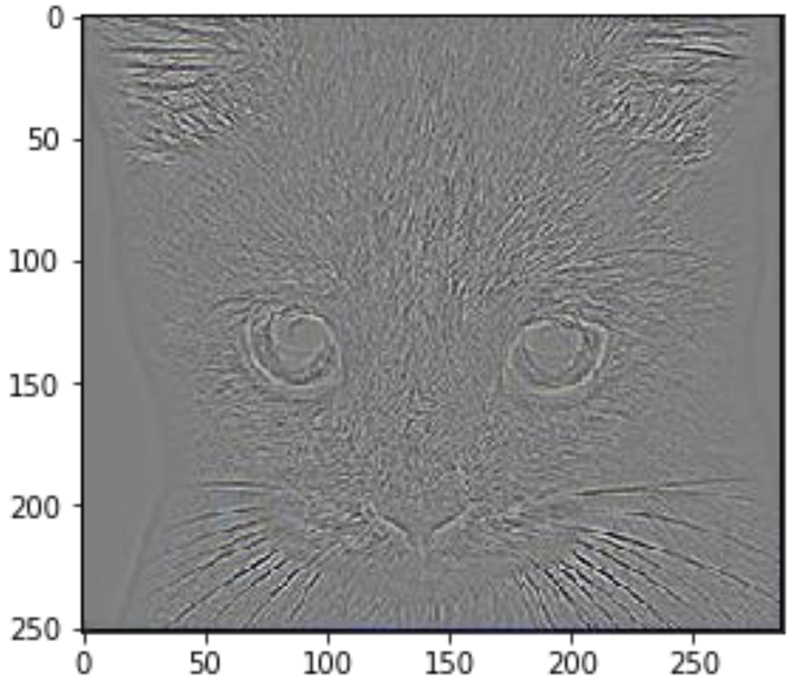


Part 1: Image filtering

Sobel filter



Discrete Laplacian filter



Part 1: Hybrid images

1. Get the low frequencies of image 1 by applying a single 2D filter to each channel of an image.(As per code: Call my_conv2d_numpy(image1, filter))

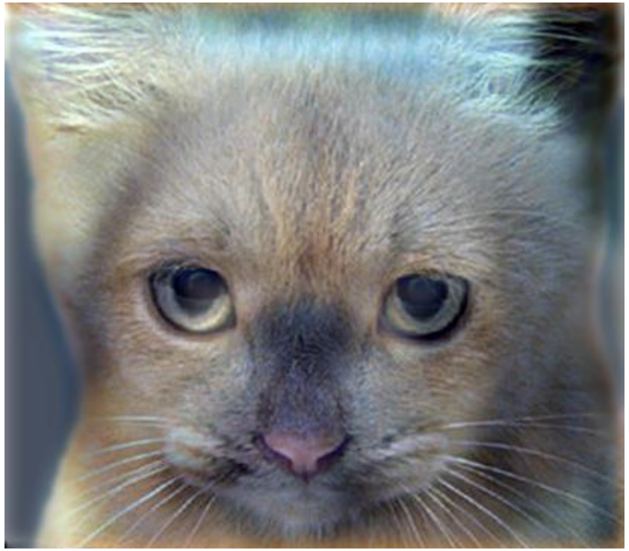
2. Get the low frequencies of image 2 by applying a single 2D filter to each channel of an image.(As per code: Call my_conv2d_numpy(image2, filter))

3. Get the high frequencies for image n by subtracting low frequencies of image n matrix from the original image n matrix.

4. Finally, add the high frequencies of image n and low frequencies of image 1 to get the hybrid image.

5. In order to ensure, that the output values are within the appropriate range for matplotlib visualizations, clip the hybrid image such that its values are between 0 and 1 using np.clip().

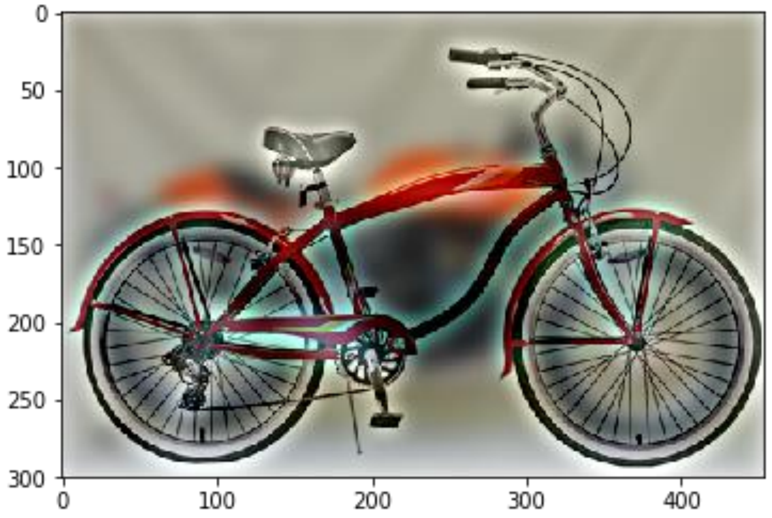
Cat + Dog



Cutoff frequency:7

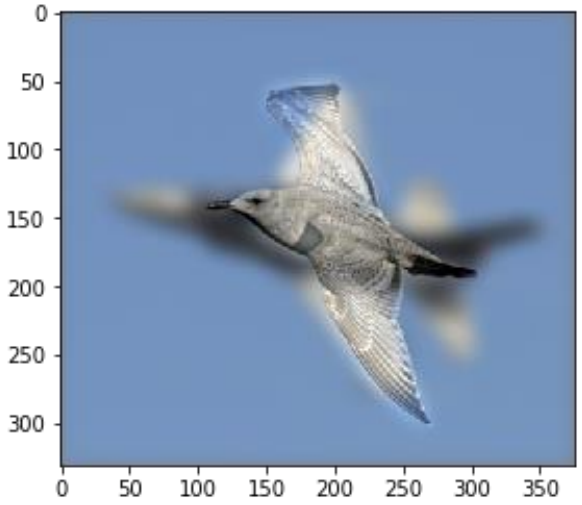
Part 1: Hybrid images

Motorcycle + Bicycle



Cutoff frequency:7

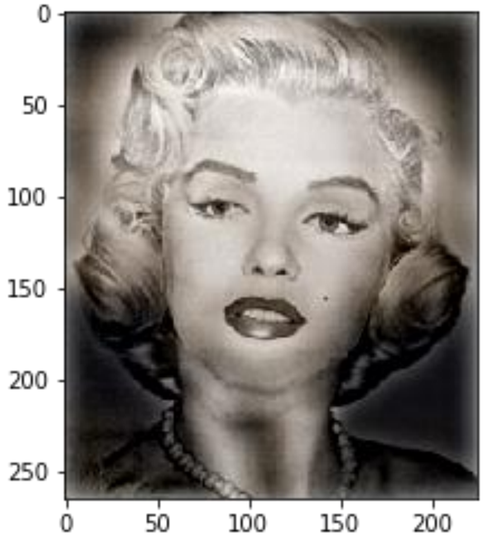
Plane + Bird



Cutoff frequency:7

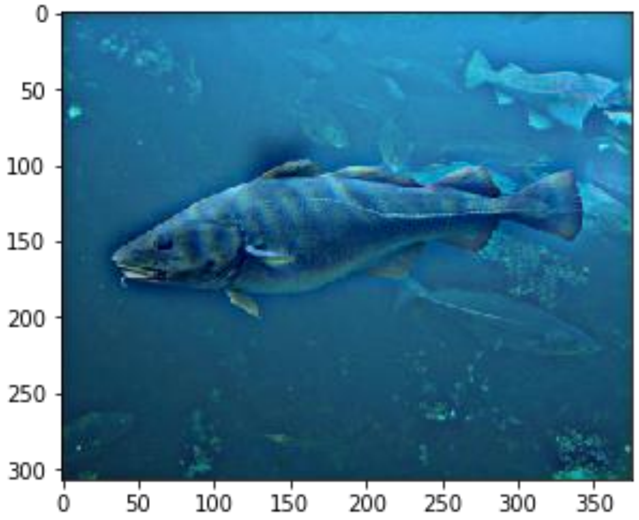
Part 1: Hybrid images

Einstein + Marilyn



Cutoff frequency:7

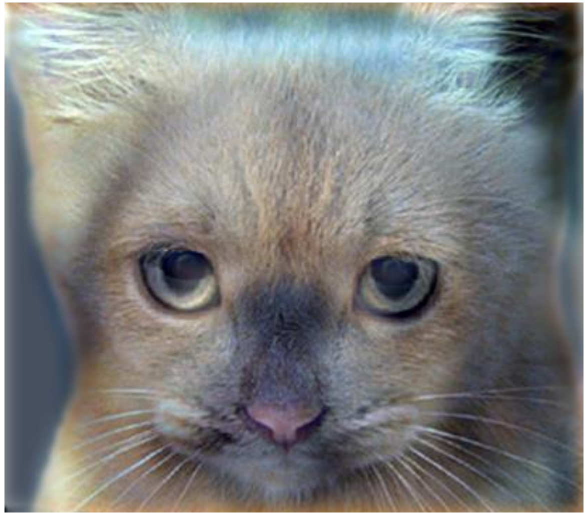
Submarine + Fish



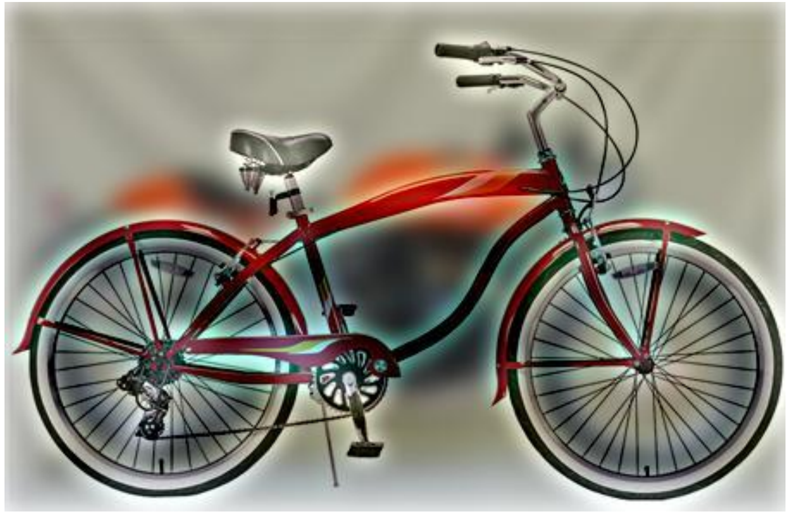
Cutoff frequency: 7

Part 2: Hybrid images with PyTorch

Cat + Dog

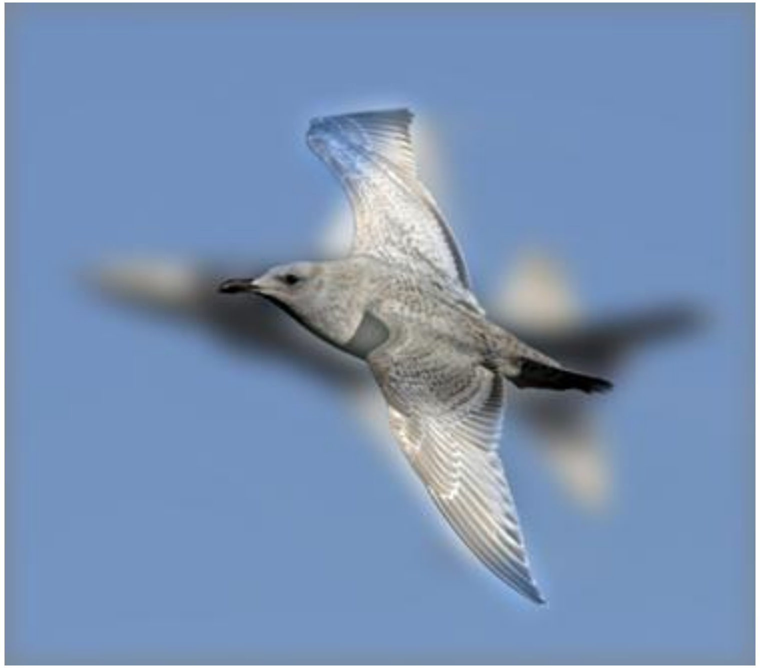


Motorcycle + Bicycle

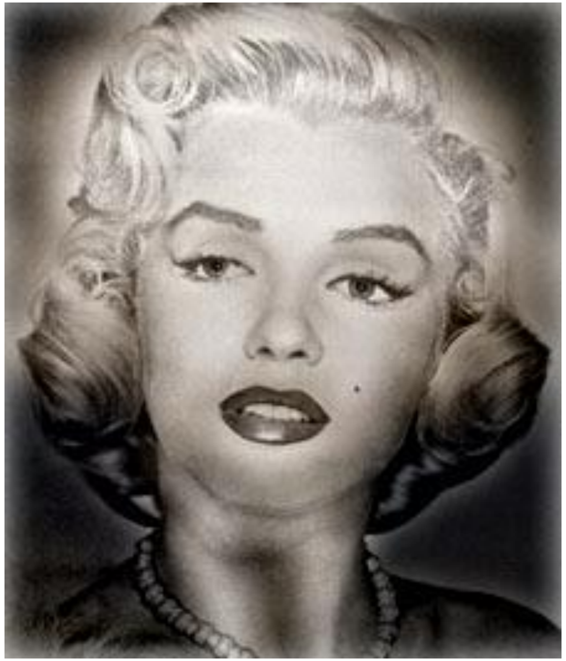


Part 2: Hybrid images with PyTorch

Plane + Bird

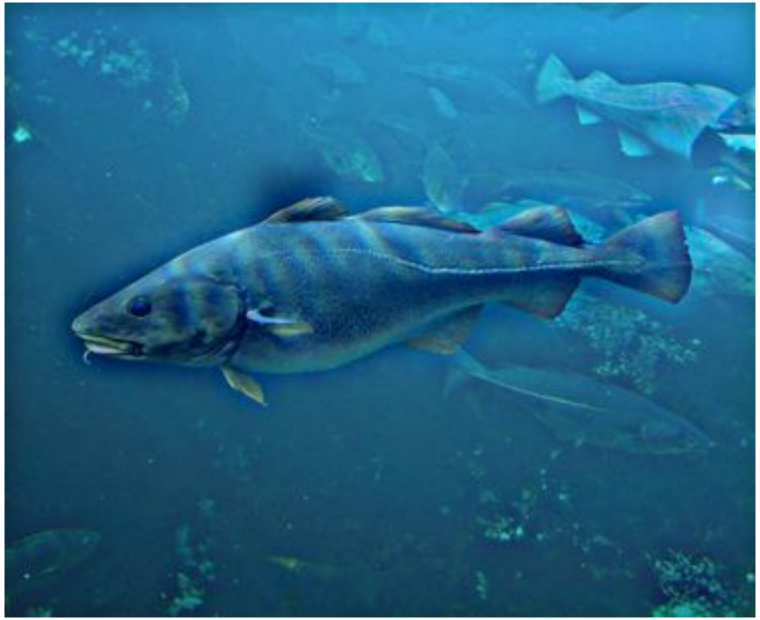


Einstein + Marilyn



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: 9.061 seconds

Part 2 runtime: 2.026 seconds

Conclusion: The process of generating image using PyTorch is faster than the Part 1 method.

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 = 1, padding = 0?
Stride = 2, padding = 0?
Stride = 1, padding = 1?
Stride = 2, padding = 1?]
Using the formula:

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

1. w1 = 5, h1 = 5, k = 3, p = 0, s = 1
Convolution Dimensions: h2 = 3 w2 = 3
2. w1 = 5, h1 = 5, k = 3, p = 0, s = 2
Convolution Dimensions: h2 = 2 w2 = 2
3. w1 = 5, h1 = 5, k = 3, p = 1, s = 1
Convolution Dimensions: h2 = 5 w2= 5
4. w1 = 5, h1 = 5, k = 3, p = 1, s = 2
Convolution Dimensions: h2 = 3 w2 = 3

[What are the input & output dimensions of the convolutions of the dog image and a 3x3 filter with the following parameters:
Stride = 1, padding = 0
Stride = 2, padding = 0
Stride = 1, padding = 1
Stride = 2, padding = 1?]

1. w1 = 410, h1 = 361, k = 3, p = 0, s = 1
Convolution Dimensions: h2 = 359 w2 = 408
2. w1 = 410, h1 = 361, k = 3, p = 0, s = 2
Convolution Dimensions: h2 = 180 w2 = 205
3. w1 = 410, h1 = 361, k = 3, p = 1, s = 1
Convolution Dimensions: h2 = 361 w2 = 410
4. w1 = 410, h1 = 361, k = 3, p = 1, s = 2
Convolution Dimensions: h2 = 181 w2 = 206

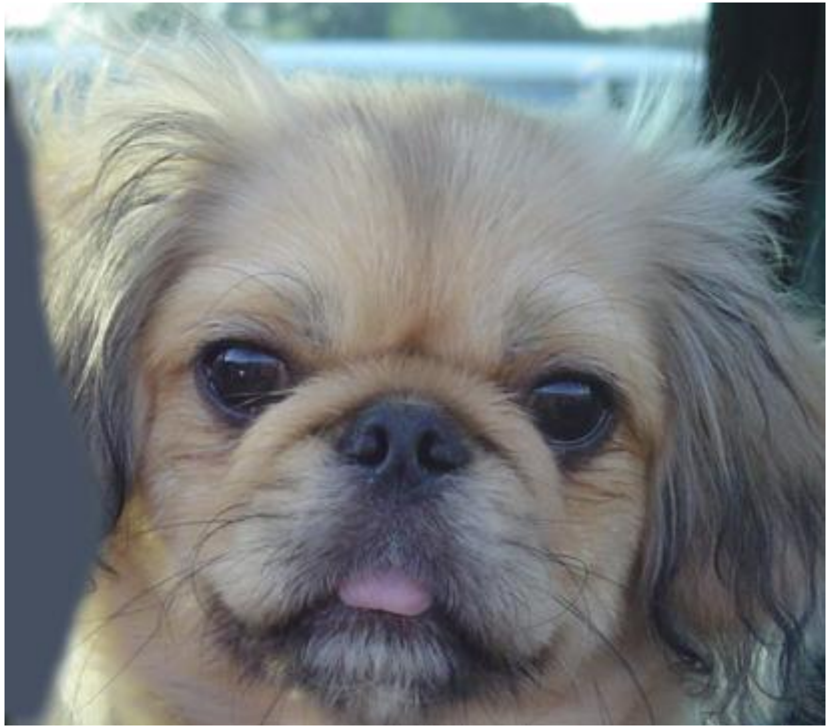
Part 3: Understanding input/output shapes in PyTorch

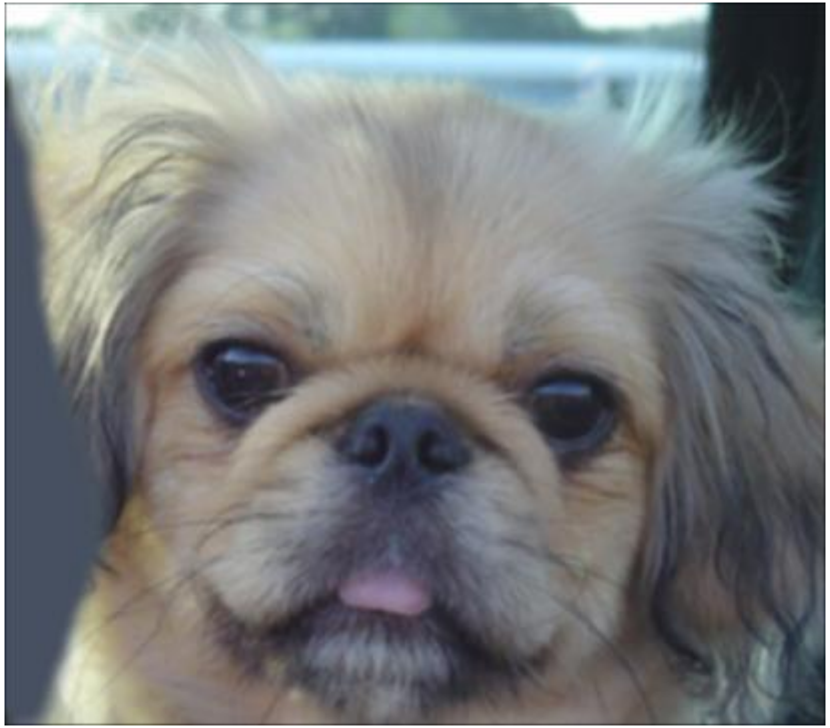
[How many filters did we apply to the dog image?]
We applied 4 unique filters to the dog image:
identity_filter
blur_filter
sobel_filter
laplacian_filter
We applied these 4 filters thrice.

[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 behind section 3 of the handout is if there is an image of known dimension (h1,w1), a kernel of size k, padding of size p, and stride s then the dimensions of the output image can be easily calculated. Furthermore, the equation shows that the dimension of the input image and that of the output image don’t need to be the same.

Part 3: Understanding input/output shapes in PyTorch



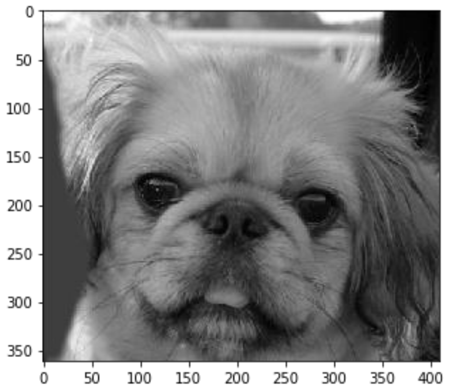


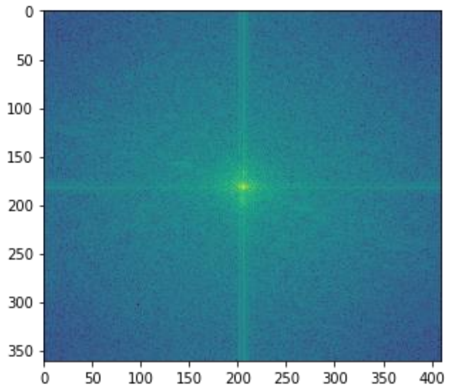
Part 3: Understanding input/output shapes in PyTorch

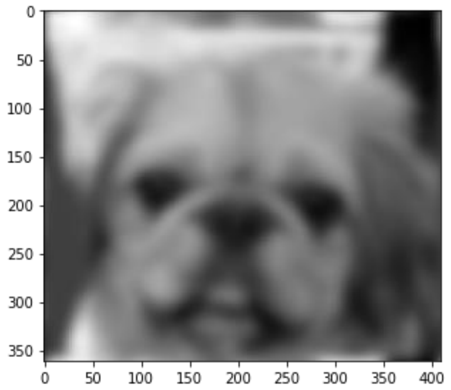
Part 4: Frequency Domain Convolutions

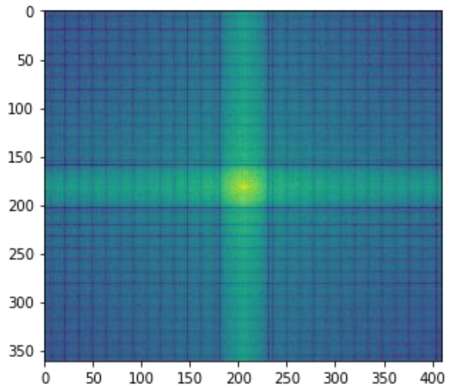
Visualizations of the dog image in the spatial and frequency domain

Visualizations of the blurred dog image in the spatial and frequency domain



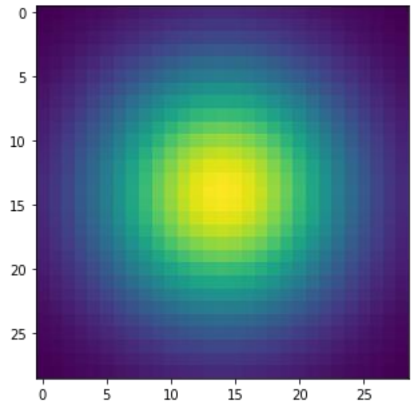


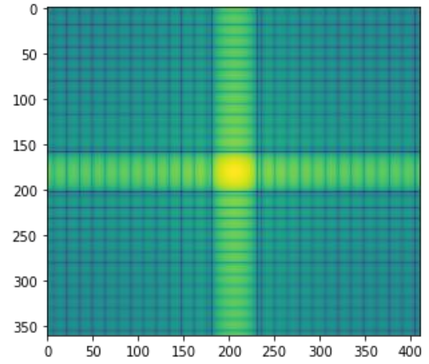




Part 4: Frequency Domain Convolutions

Visualizations of the 2D Gaussian in the spatial and frequency domain]





[Why does our frequency domain representation of a Gaussian not look like a Gaussian itself? How could we adjust the kernel to make these look more similar?]

The first image on the left side is the spatial domain representation of the Gaussian kernel whereas the one on the right-hand side is the frequency domain representation and therefore the frequency domain representation of a Gaussian does not look like a Gaussian itself. We can adjust to make these look more similar by removing the padding, extracting real and imaginary components of the FFT output of the kernel, and then applying inverse FFT over it using np.fft.ifft2(). Finally, we can use np.log() and np.abs() to display a similar image.

Part 4: Frequency Domain Convolutions

[Briefly explain the Convolution Theorem and why this is related to deconvolution]

The convolution theorem states that the Fourier transform of convolution of two functions is the product of their Fourier transforms. Convolution in the spatial domain is equivalent to multiplication in the frequency domain. It is stated as follows:

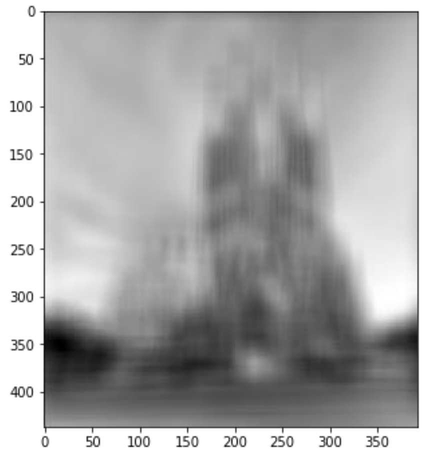
$$F[p*q] = F[p]F[q]$$

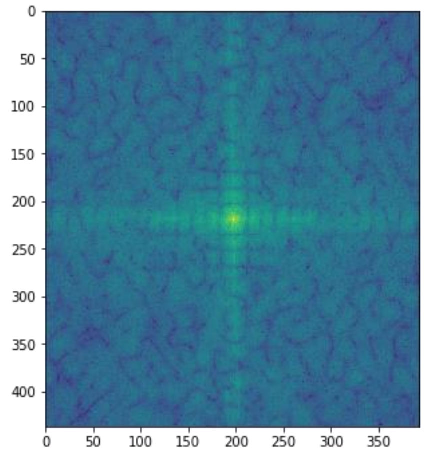
On the other hand, deconvolution is the reverse of convolution. The blurred image produced by convolution can be converted to the original image(deblurred) with the help of deconvolution provided there are no external factors like noise involved. The relationship between convolution and deconvolution can be stated as follows:

$$F[p*q] = F[p]F[q] \quad p*q \quad = F^{-1}(F[p]F[q])$$

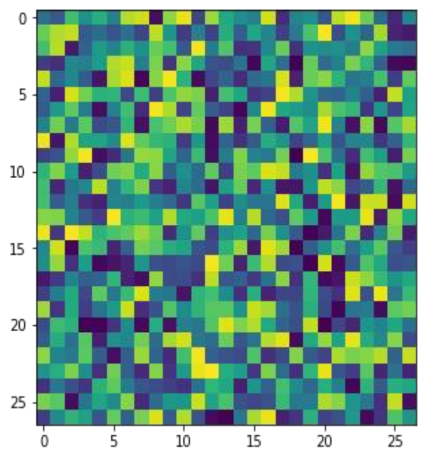
Part 4: Frequency Domain Convolutions

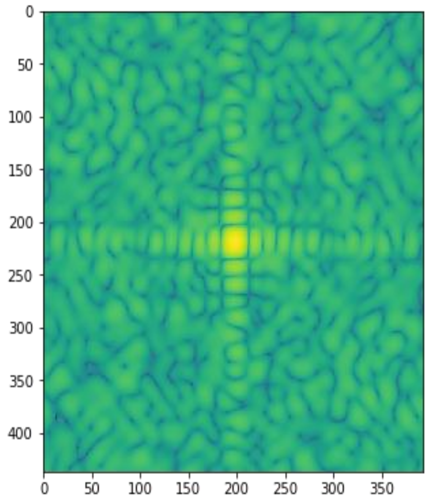
Visualizations of the mystery image in the spatial and frequency domain





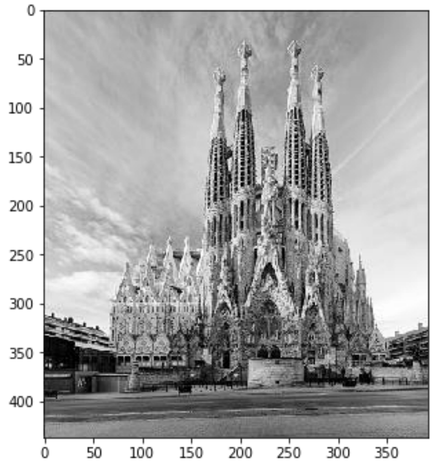
Visualizations of the mystery kernel in the spatial and frequency domain

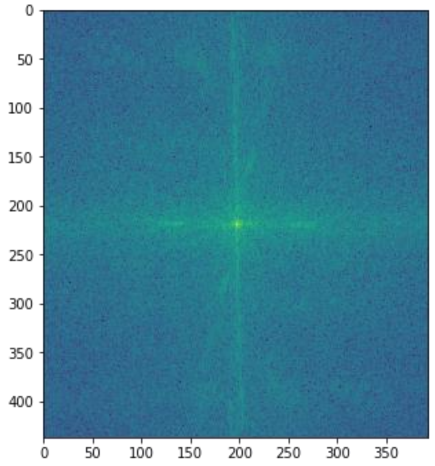




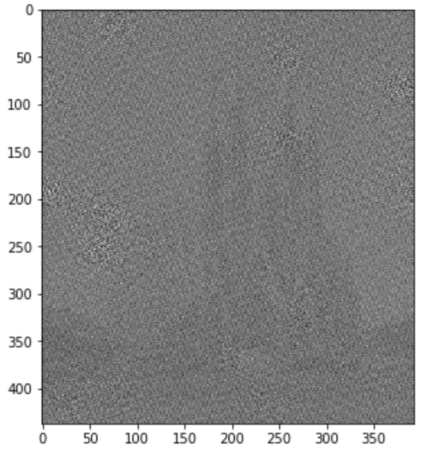
Part 4: Frequency Domain Convolutions

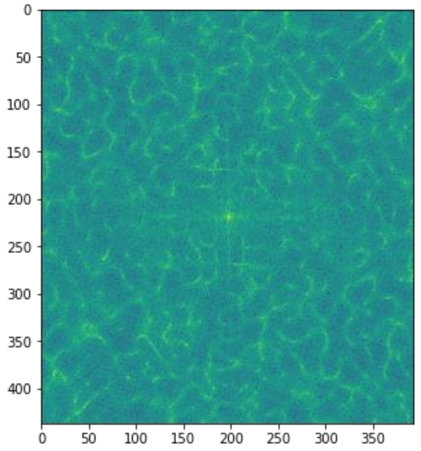
De-blurred mystery image and its visualizations in the spatial and frequency domain





De-blurred mystery image and its visualizations in the spatial and frequency domain after adding salt and pepper noise





Part 4: Frequency Domain Convolutions

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

The following factors limit the potential use of deconvolution in the real world: 1. A lot of research is still going on, on this particular topic.2. In the real-world scenario, the filter used may be unknown (known as blind deconvolution) or known (known as non-blind convolution). In both cases, the filter can have some or many values that are very small and closer to zero. These smaller values can cause issues while dividing the blurred image by the filter. Another issue is noise in the input image used for deconvolution. In a real-world scenario, it is difficult to calculate how much noise was present in the original image and how much was added during the convolution process.

[We performed two convolutions of the dog image with the same Gaussian (one in the spatial domain, one in the frequency domain). How do the two compare, and why might they be different?]

When the dog image convolved in the spatial domain, the speed was slow due to the nested loops. However, the convolution in the frequency domain provides a way to reduce this computational complexity and thus increase the overall speed.

Conclusion

[How does varying the cutoff frequency value or swapping images within a pair influences the resulting hybrid image?]

Cutoff frequency represents the standard deviation of the Gaussian kernel. Therefore, by altering the magnitude of cutoff frequency, we can get the appropriate hybrid image in which high frequencies are visible nearby whereas low frequencies are visible far away. However, when the high and low-frequency images are swapped, a lot depends on the finer details within the image. For example, when Dog and Cat images are interchanged, such that the cat image represents lower frequency and the dog image the higher frequency, then, in this case, we can see the cat image even if the hybrid image is nearby. The reason for the visibility of the cat from nearby is because it has finer details than the dog image. Therefore the cat image should be used as a higher frequency image rather than a lower frequency image.