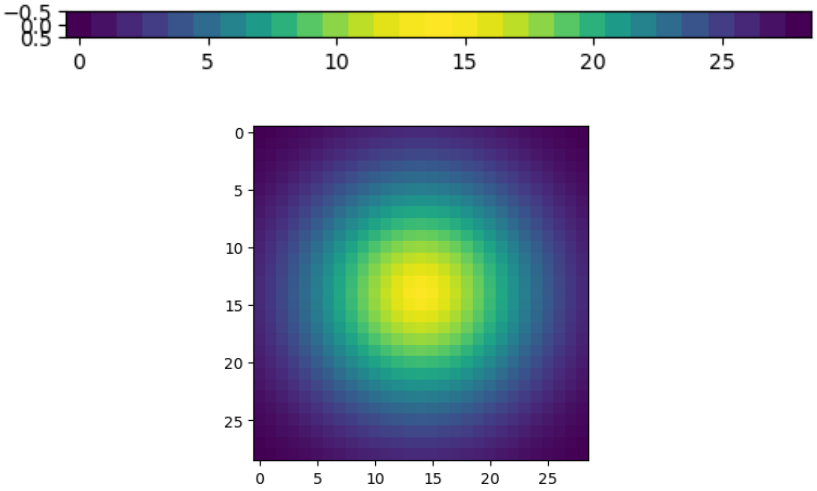
CS 6476 Project 1

Shuvo Newaz
shuvo.newaz@gatech.edu
snewaz3
903614132

Part 1: Image filtering

[insert visualization of Gaussian kernel from project-1.ipynb here]



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

For a filter $\boldsymbol{W}$ of dimension $k \times j$, The row and column padding were determined as

$$pad_{\text{row}} = \text{int}\left(\frac{k-1}{2}\right), pad_{\text{col}} = \text{int}\left(\frac{j-1}{2}\right)$$

The original image $\boldsymbol{I}$ was padded with zeros, Giving a new matrix $\boldsymbol{I}_{\text{pad}}$. For an image of size $m \times n$, a matrix $\boldsymbol{I}'$ of size $m' \times n'$ was created, where,

$$m' = m + 2pad_{\text{row}}, n' = n + 2pad_{\text{col}}$$

$\boldsymbol{I}_{\text{pad}}$ and $\hat{\boldsymbol{I}}$ have the same dimension. This matrix was filled using the following equation:

$$\hat{\boldsymbol{I}}(h, w, c) = \text{sum}\left(\boldsymbol{W} \odot \boldsymbol{I}_{\text{pad}}(h\!:\!h+k, w\!:\!w+j, c)\right)$$

The final filtered image is given by:

$$\hat{\boldsymbol{I}}(0\!:\!m' - 2pad_{\text{row}}, 0\!:\!n' - 2pad_{\text{col}}, :)$$

Part 1: Image filtering

Identity filter

[insert the results from project-1.ipynb using 1b_cat.bmp with the identity filter here]

Small blur with a box filter

[insert the results from project-1.ipynb using 1b_cat.bmp with the box filter here]

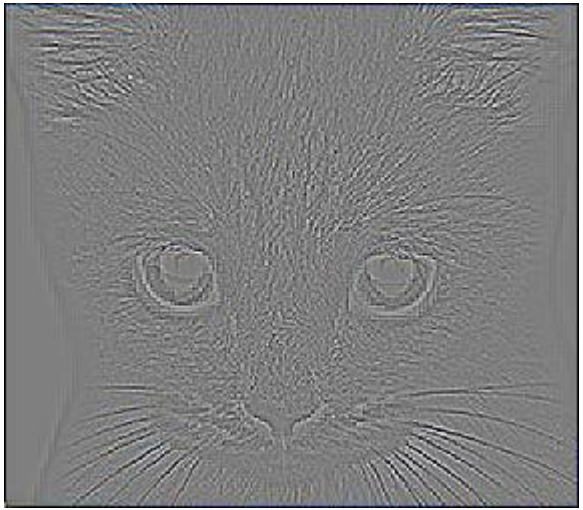
Part 1: Image filtering

Sobel filter

[insert the results from project-1.ipynb using 1b_cat.bmp with the Sobel filter here]

Discrete Laplacian filter

[insert the results from project-1.ipynb using 1b_cat.bmp with the discrete Laplacian filter here]



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 the low-frequency contents from both image a and image b using Gaussian filters.

2. Subtract low-frequency contents of image b from original image b to get the high-frequency contents.

3. Add the low-frequency contents of image a and high-frequency contents of image b. Limit the pixel values within $[0,1]$.

Cat + Dog

Cutoff frequency: 5

Part 1: Hybrid images

Motorcycle + Bicycle

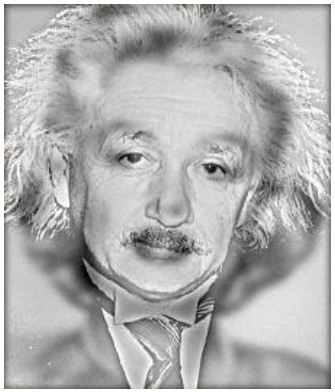
Cutoff frequency: 6

Plane + Bird

Cutoff frequency: 7

Part 1: Hybrid images

Einstein + Marilyn



Cutoff frequency: 3

Submarine + Fish

Cutoff frequency: 3

Part 2: Hybrid images with PyTorch

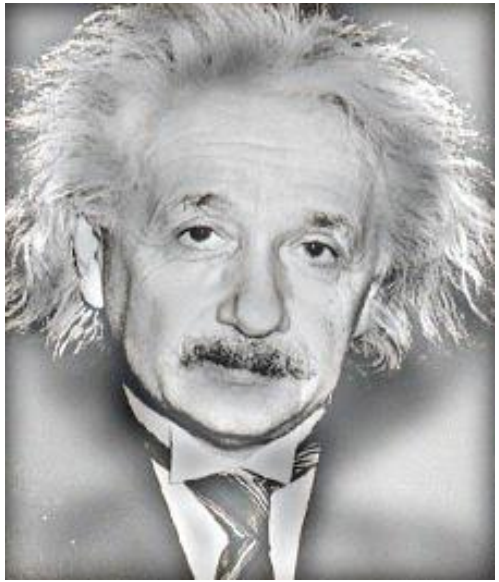
Cat + Dog

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: 11.595s

Part 2: 1.536s

Part 2 is much faster than Part 1.

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? 3 × 3
Stride = 2, padding = 0? 2 × 2
Stride = 1, padding = 1? 5 × 5
Stride = 2, padding = 1? 3 × 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? 359 × 408
Stride = 2, padding = 0? 180 × 204
Stride = 1, padding = 1? 361 × 410
Stride = 2, padding = 1? 181 × 205

The input dimension of the dog image is 361 × 410.

Part 3: Understanding input/output shapes in PyTorch

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

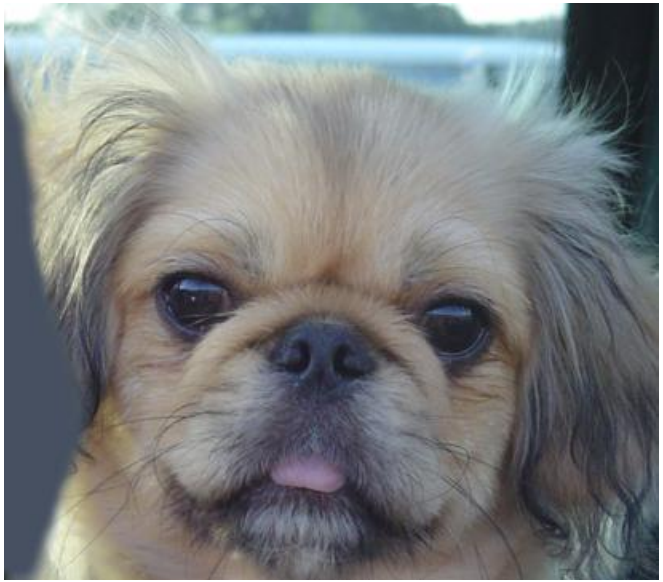
We applied 4 different filters to the dog image.

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

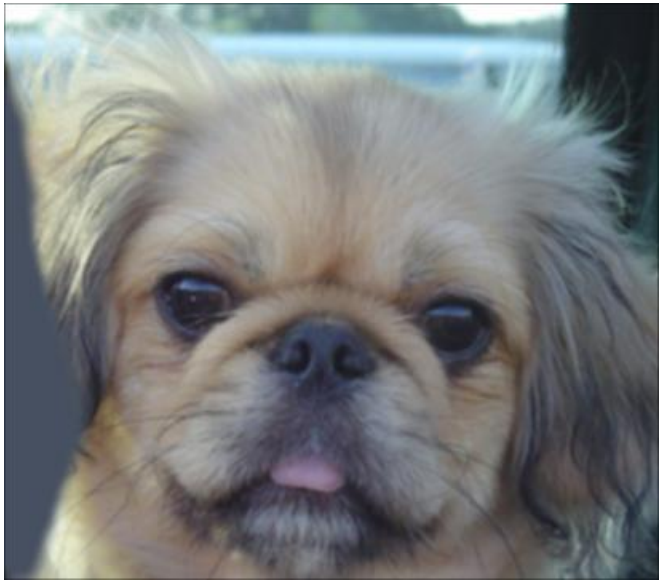
In any form of pooling operation, the original dimension of the image will get reduced because several cells of the matrix get lumped into 1 cell of the resulting matrix. The amount of this reduction depends on the size of the filter and the stride of the pooling operation. To maintain the correct matrix shape, we pool the original matrix with zeros so we can retain the original shape when it does get reduced.

Part 3: Understanding input/output shapes in PyTorch

[insert visualization 0 here]



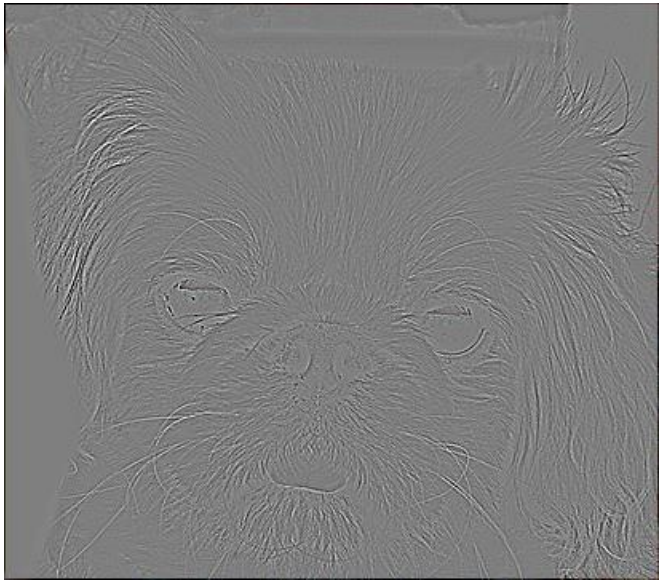
[insert visualization 1 here]



Part 3: Understanding input/output shapes in PyTorch

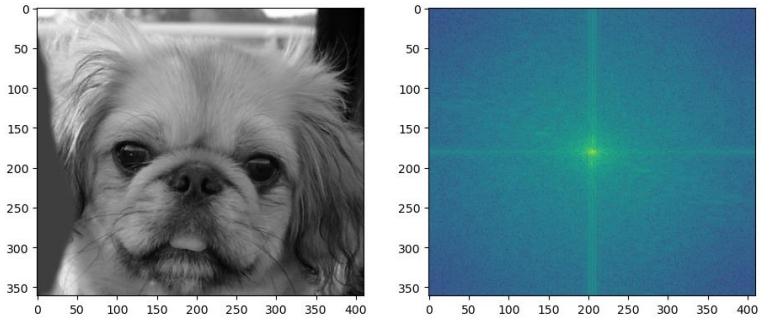
[insert visualization 2 here]

[insert visualization 3 here]

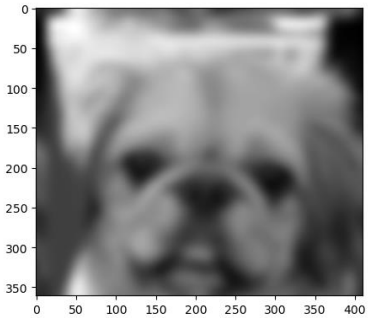


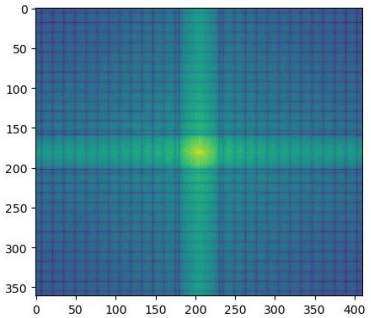
Part 4: Frequency Domain Convolutions

[Insert the visualizations of the dog image in the spatial and frequency domain]



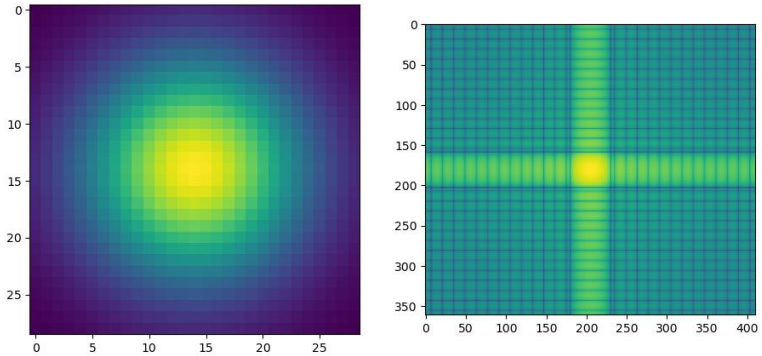
[Insert the visualizations of the blurred dog image in the spatial and frequency domain]





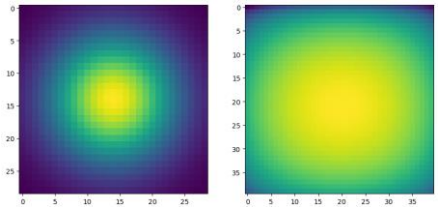
Part 4: Frequency Domain Convolutions

[Insert the 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 wider (higher standard deviation) the Gaussian is in spatial domain, the narrower the Gaussian will be in frequency domain. If we zoom in on the frequency domain representation, we can see that it is in fact a Gaussian.



Part 4: Frequency Domain Convolutions

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

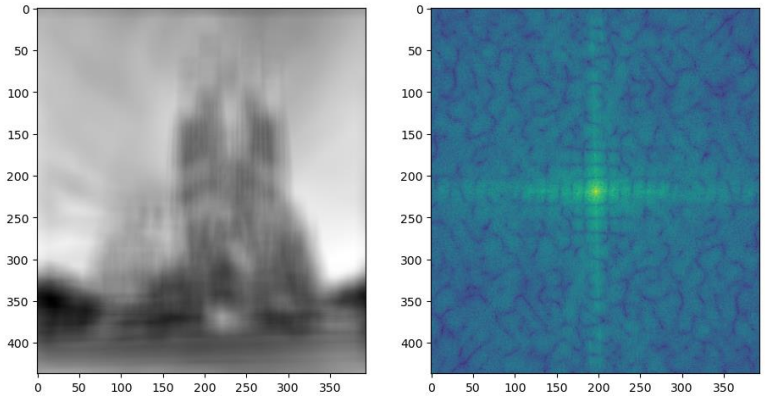
Convolution of an image with a kernel is the mathematical analogue of the filtering operation. It is related to deconvolution in the way that if we get image $\hat{\boldsymbol{I}}$ by convolving image $\boldsymbol{I}$ and filter $\boldsymbol{W}$, we can get image $\boldsymbol{I}$ by deconvolving image $\hat{\boldsymbol{I}}$ using filter $\boldsymbol{W}$. This is readily seen in frequency domain. The convolution operation in frequency domain is given by

$$\hat{\boldsymbol{I}} = \boldsymbol{I} \odot \boldsymbol{W}$$

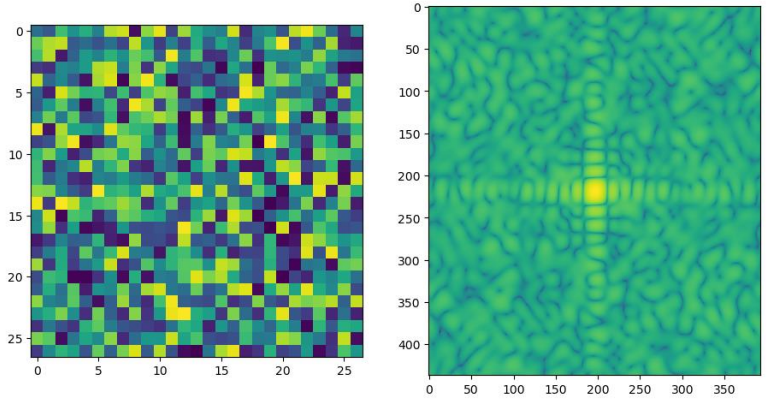
Where $\odot$ represents the elementwise product operation. From the above equation, we can determine that recovering the original image is simply the elementwise division of the filtered image by the filter.

Part 4: Frequency Domain Convolutions

[Insert the visualizations of the mystery image in the spatial and frequency domain]

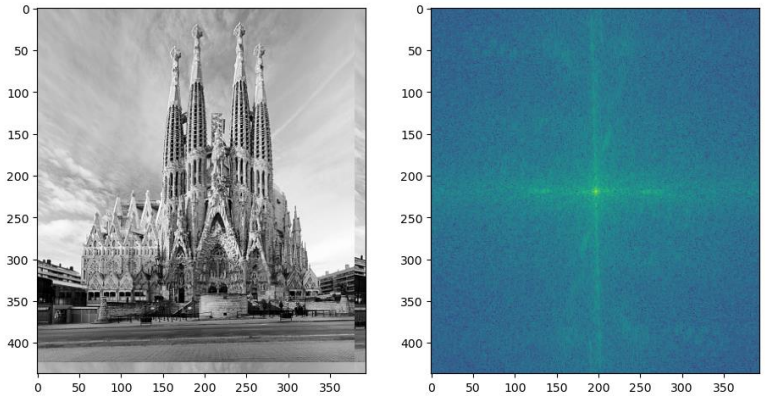


[Insert the visualizations of the mystery kernel in the spatial and frequency domain]

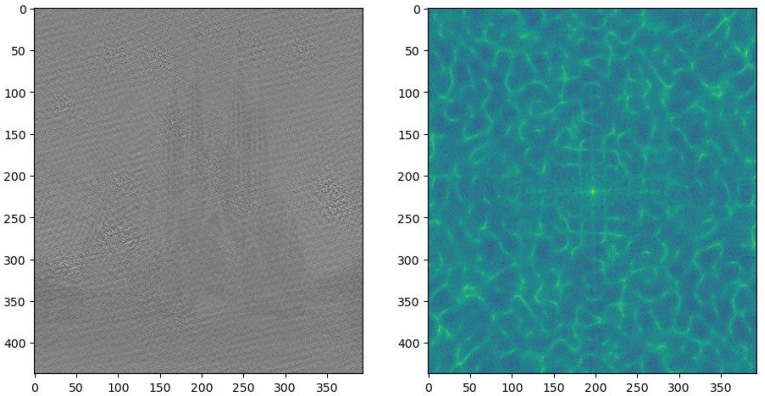


Part 4: Frequency Domain Convolutions

[Insert the de-blurred mystery image and its visualizations in the spatial and frequency domain]



[Insert the 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]

As seen from the results, deconvolution is very susceptible to noise. Since deconvolution is a division of the image by the filter contents, even small amounts of noise make a big difference in the result. Also, there is always a possibility of a divide-by-zero error to occur with deconvolution.

[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?]

Convolution in spatial domain is essentially an integration of the product of the image and the filter, whereas in frequency domain, it is simply the elementwise product of the Fourier transform of the image and the Fourier transform of the filter. In general, integrations are computationally heavier than multiplications.

Conclusion

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

Lowering the cutoff frequency emphasizes the low-frequency image. Increasing the cutoff frequency emphasizes the high-frequency image.