CS 4476 Project 1

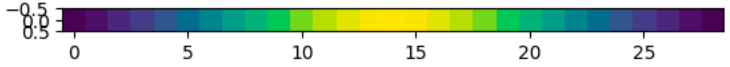
Vidit Pokharna
vidit@gatech.edu
vpokharna3
903772087

(1 pts) Part 1: Gaussian Kernels

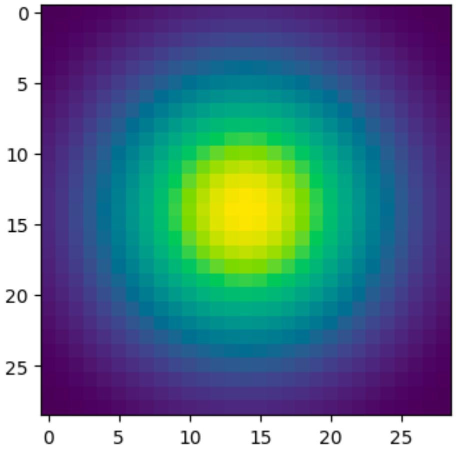
Separated kernels speed up filtering operation because they reduce the amount of computation. Instead of using k^2 multiplications per pixel with a 2D kernel, a separable kernel allows for $2k$ operations. This linear time complexity is more efficient, making the process more efficient, especially for larger filters.

(1 pts) Part 1: Image filtering

1D:



2D:

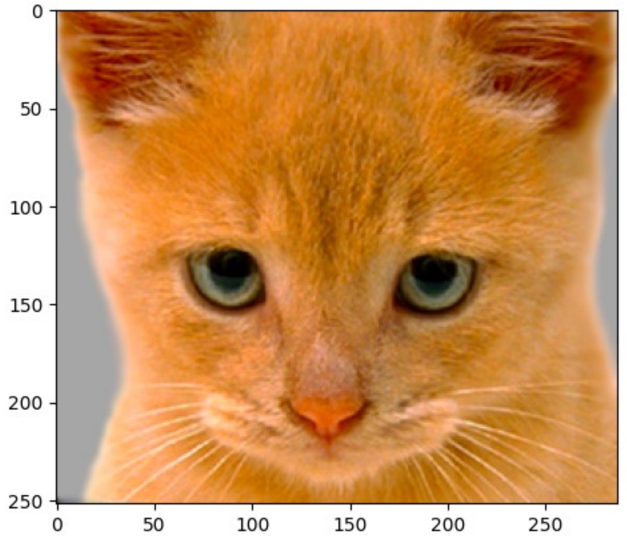


In the function my_conv2d_numpy(), a 2D filter is applied to each channel of the input image. The image is first padded with zeros to maintain its size after filtering. This padding is calculated based on the filter's dimensions, ensuring the filtered image has the same shape as the input.

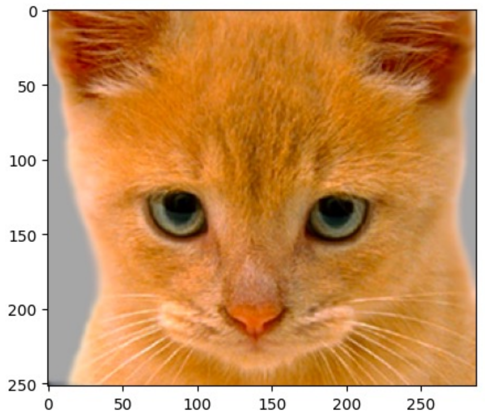
The function iterates over each channel of the image. For each pixel, a region of the image corresponding to the filter size is selected. The filter is then element-wise multiplied with this region, and the results are summed up to produce a single output pixel in the filtered image. This process is repeated for every pixel in each channel, resulting in the final filtered image.

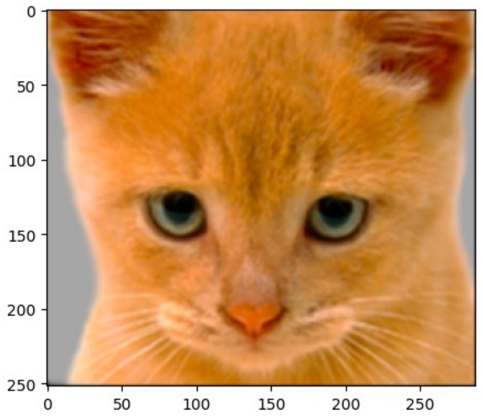
(1 pts) Part 1: Image filtering

Identity filter



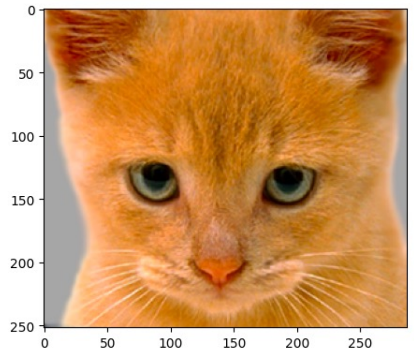
Small blur with a box filter

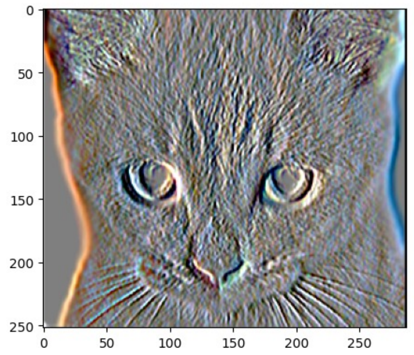




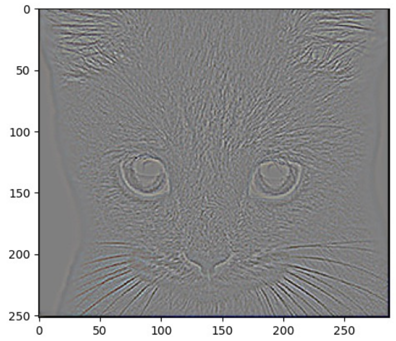
(1 pts) Part 1: Image filtering

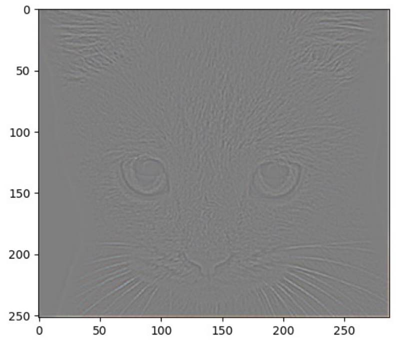
Sobel filter





Discrete Laplacian filter





(1.5 pts) Part 1: Hybrid images

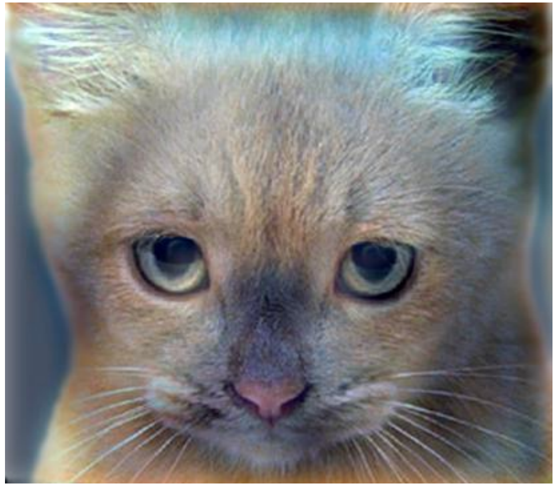
1. Apply a low-pass filter to image1 to retain its broad, smooth variations (low-frequency components).

2. Subtract the low-pass filtered image2 from the original image2 to isolate its fine details and edges (high-frequency components).

3. Combine the low-frequency content of image1 with the high-frequency content of image2.

The output values are normalized to the [0, 1] range by clipping the final hybrid image using np.clip, ensuring compatibility with matplotlib visualizations.

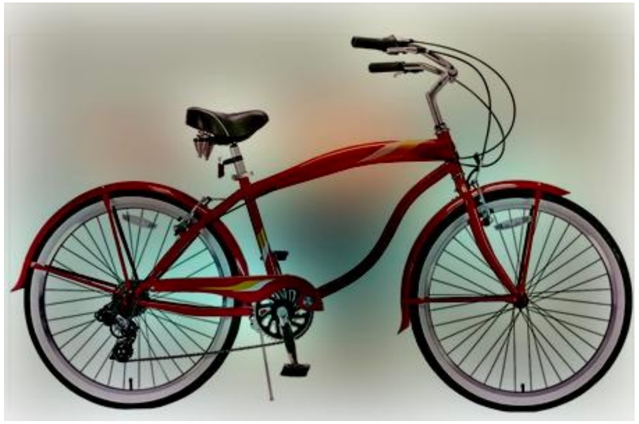
Cat + Dog



Cutoff frequency: 10

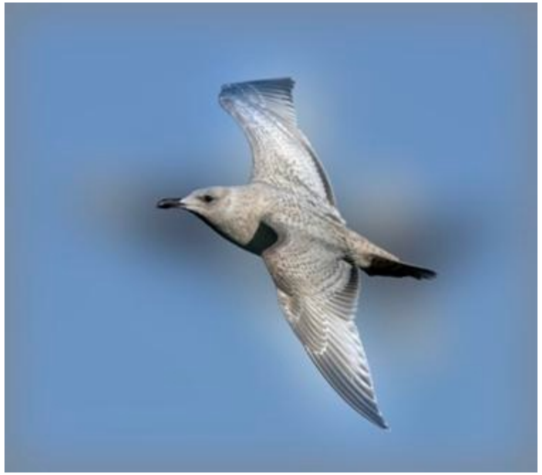
(1.5 pts) Part 1: Hybrid images

Motorcycle + Bicycle



Cutoff frequency: 25

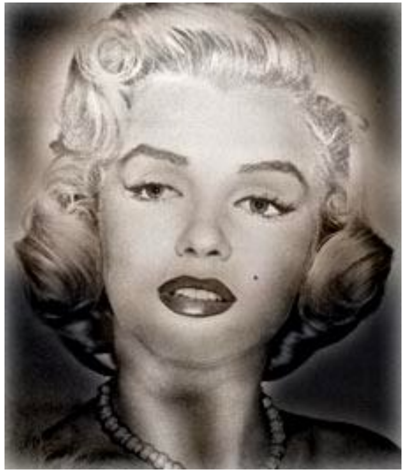
Plane + Bird



Cutoff frequency: 20

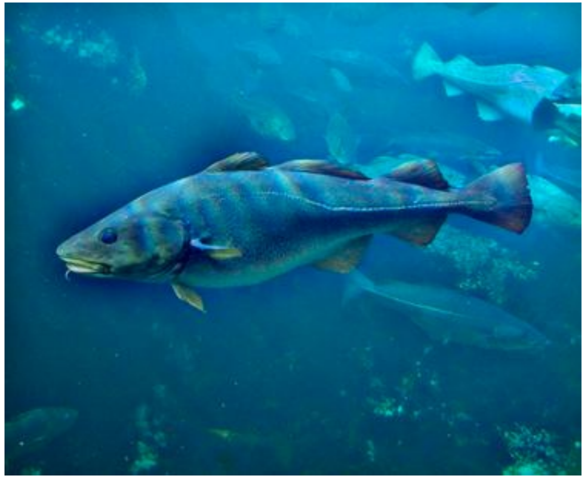
(1.5 pts) Part 1: Hybrid images

Einstein + Marilyn



Cutoff frequency: 7

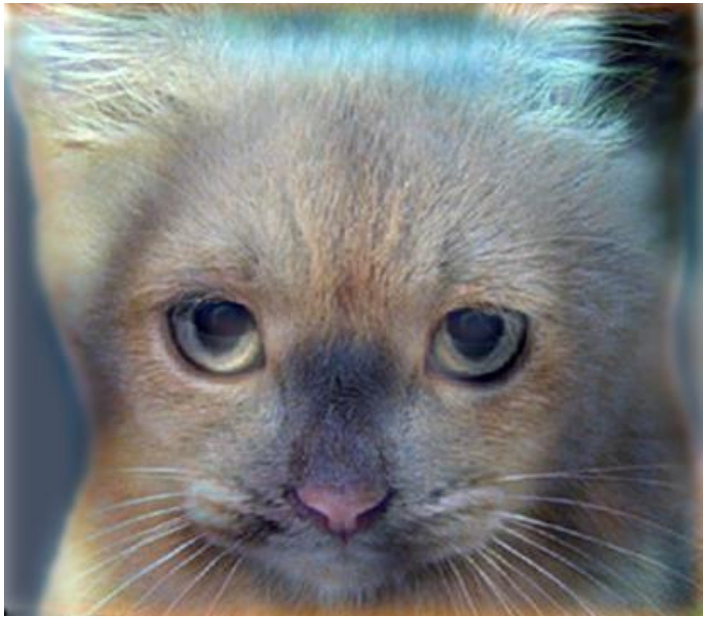
Submarine + Fish



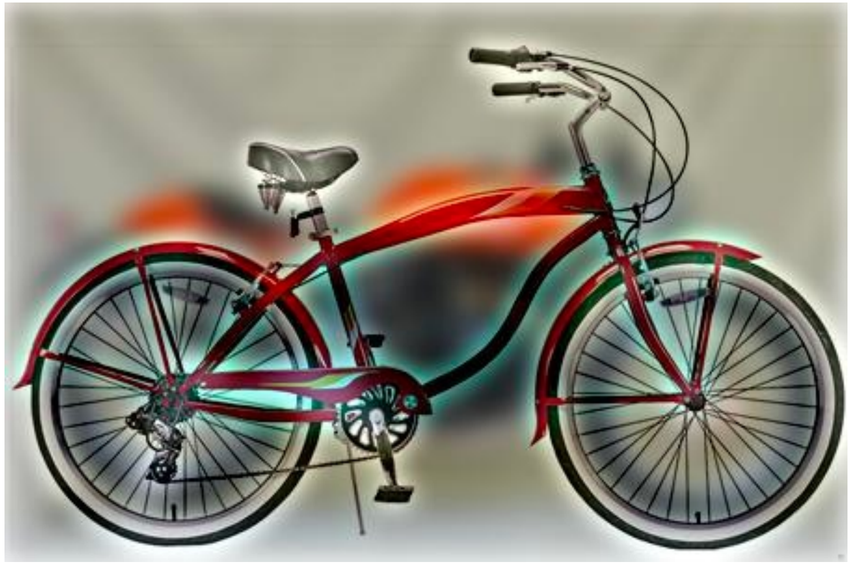
Cutoff frequency: 14

(1.5 pts) Part 2: Hybrid images with PyTorch

Cat + Dog

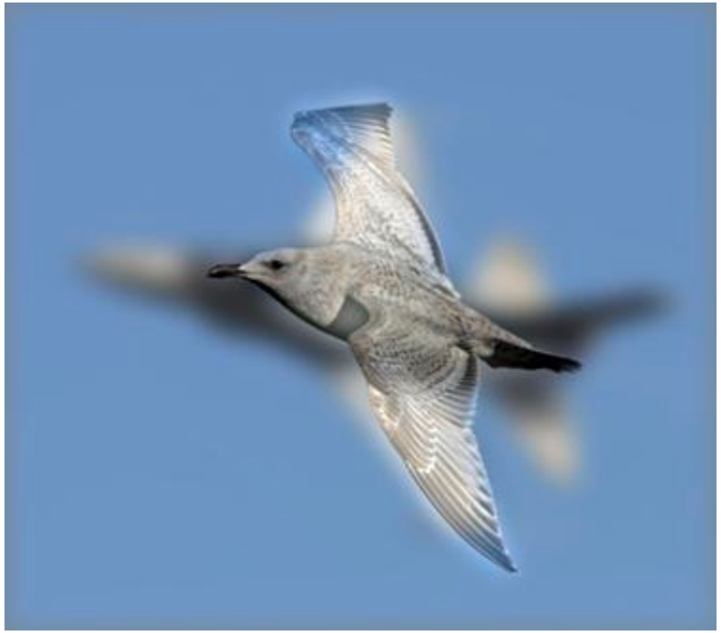


Motorcycle + Bicycle

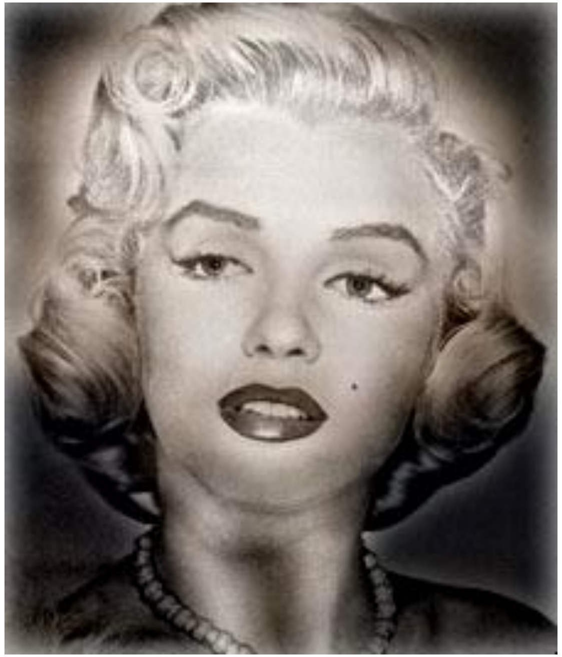


(1.5 pts) Part 2: Hybrid images with PyTorch

Plane + Bird

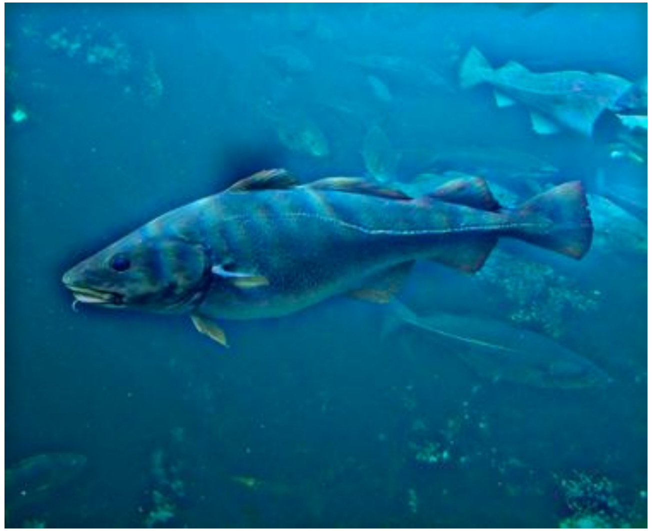


Einstein + Marilyn



(1.5 pts) Part 2: Hybrid images with PyTorch

Submarine + Fish



Part 1 vs. Part 2

It is clear that Part 2 is significantly faster. Part 1 has an average runtime of 9.2 seconds per pair of pictures, totaling 46 seconds for all 5 pairs. In contrast, for Part 2, all 5 pairs took just 1.5 seconds. This makes Part 2 approximately 30.67 times faster than Part 1, demonstrating a substantial improvement in computational efficiency.

(2 pts) 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: 3x3
Stride = 2, padding = 0: 2x2
Stride = 1, padding = 1: 5x5
Stride = 2, padding = 1: 3x3

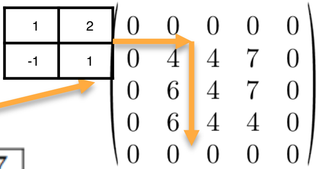
What are the input & output dimensions of the convolutions of the dog image and a 3x3 filter with the following parameters:

Dog image dimensions: 361x410x3
Stride = 1, padding = 0: 359x408x3
Stride = 2, padding = 0: 180x204x3
Stride = 1, padding = 1: 361x410x3
Stride = 2, padding = 1: 181x205x3

(1 pts) Part 3: Understanding input/output shapes in PyTorch

$$\mathbf{M} = \begin{array}{|c|c|c|}\hline 4 & 4 & 7 \\ \hline 6 & 4 & 7 \\ \hline 6 & 4 & 4 \\ \hline \end{array}$$

$$\mathbf{K} = \begin{array}{|c|c|}\hline 1 & 2 \\ \hline -1 & 1 \\ \hline \end{array}$$



$$\begin{array}{|c|c|}\hline 1 & 2 \\ \hline -1 & 1 \\ \hline \end{array}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 4 & 7 & 0 \\ 0 & 6 & 4 & 7 & 0 \\ 0 & 6 & 4 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \rightarrow$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1\cdot 4 & -1\cdot 4+1\cdot 4 & -1\cdot 4+1\cdot 7 & -1\cdot 7 \\ 0 & 2\cdot 4+1\cdot 6 & 1\cdot 4+2\cdot 4+1\cdot 4-1\cdot 6 & 1\cdot 4+2\cdot 7+1\cdot 7-1\cdot 4 & 1\cdot 7-1\cdot 7 \\ 0 & 2\cdot 6+1\cdot 6 & 1\cdot 6+2\cdot 4+1\cdot 4-1\cdot 6 & 1\cdot 4+2\cdot 7+1\cdot 4-1\cdot 4 & 1\cdot 7-1\cdot 4 \\ 0 & 2\cdot 6 & 1\cdot 6+2\cdot 4 & 1\cdot 4+2\cdot 4 & 1\cdot 4 \end{pmatrix}$$

$$\downarrow$$

$$\boxed{\begin{pmatrix} 4 & 0 & 3 & -7 \\ 14 & 10 & 21 & 0 \\ 18 & 12 & 18 & 3 \\ 12 & 14 & 12 & 4 \end{pmatrix}} \leftarrow \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 3 & -7 \\ 0 & 14 & 10 & 21 & 0 \\ 0 & 18 & 12 & 18 & 3 \\ 0 & 12 & 14 & 12 & 4 \end{pmatrix}$$

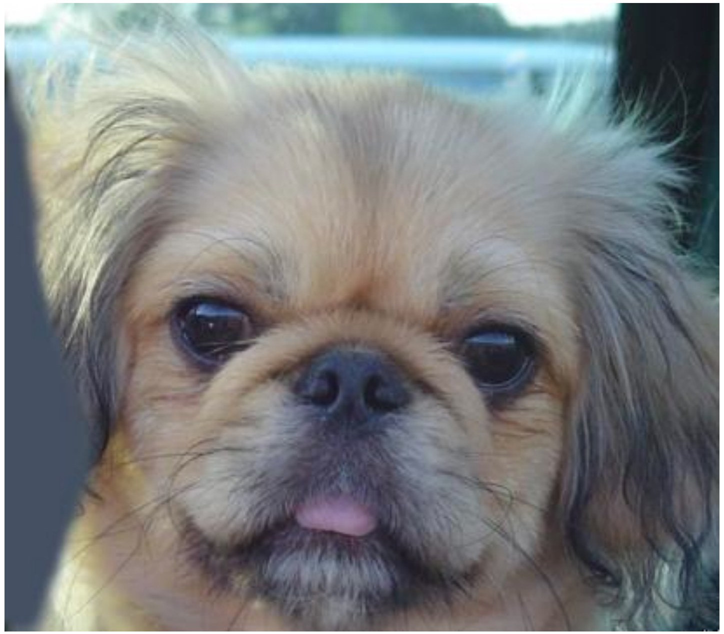
(1 pts) Part 3: Understanding input/output shapes in PyTorch

We applied 12 filters to the dog image in Part 3

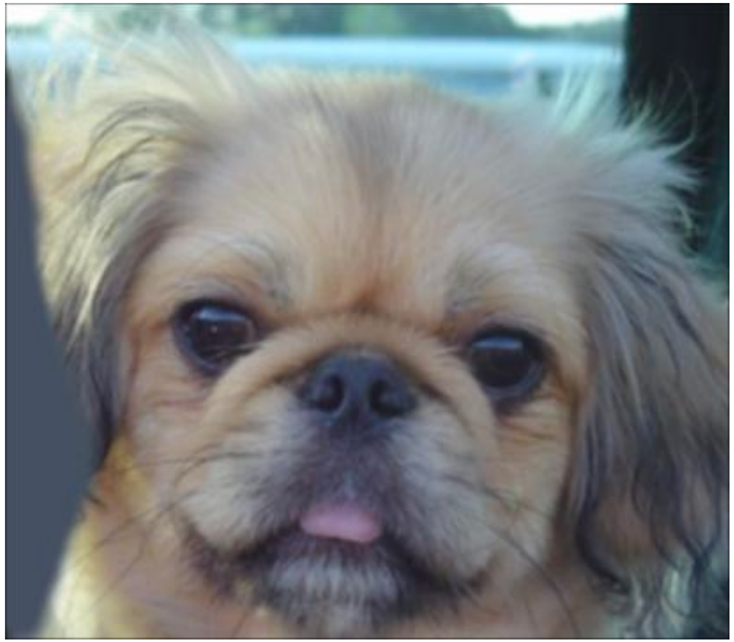
The equation for convolution output dimensions accounts for how a filter covers an input image. It factors in the filter's size, the stride's jump between applications, and any extra padding around the image. This determines the number of positions the filter can occupy, directly influencing the size of the output.

(1 pts) Part 3: Understanding input/output shapes in PyTorch

Visualization 0

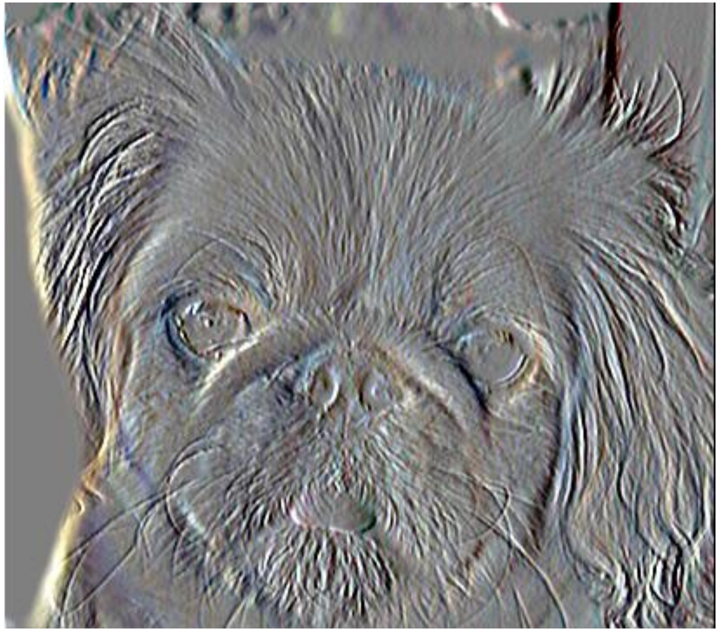


Visualization 1

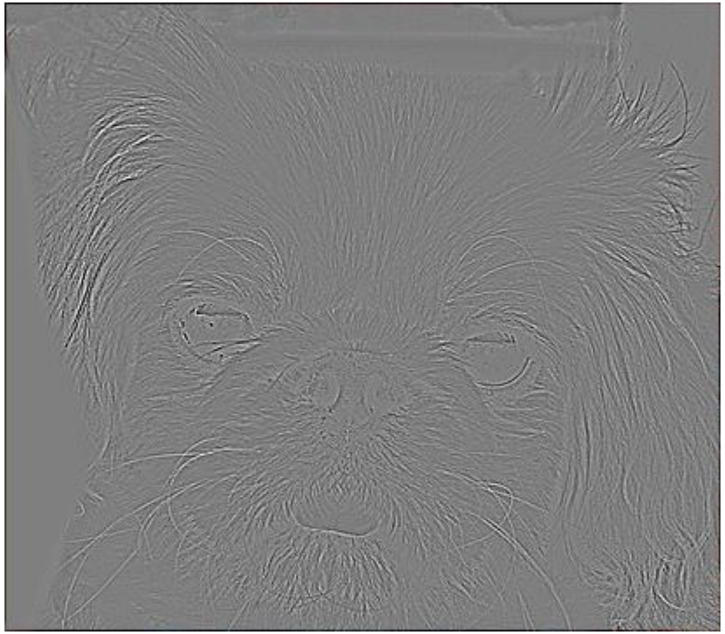


(1 pts) Part 3: Understanding input/output shapes in PyTorch

Visualization 2



Visualization 3



(1 pts) Conclusion

Varying the cutoff frequency changes the amount of high and low frequencies allowed in each component image: a lower cutoff retains more low-frequency information (broad features), and a higher cutoff allows more high-frequency information (fine details). Swapping images in a pair shifts which image contributes fine versus broad features to the hybrid image.