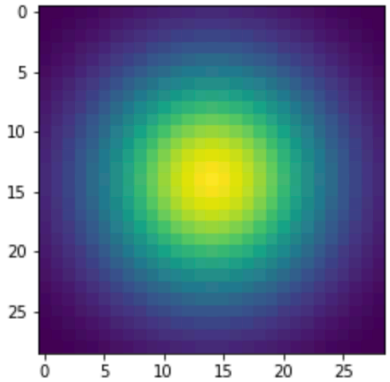
CS 6476 Project 1

Bojun Yang
byang301@gatech.edu
byang301
903254309

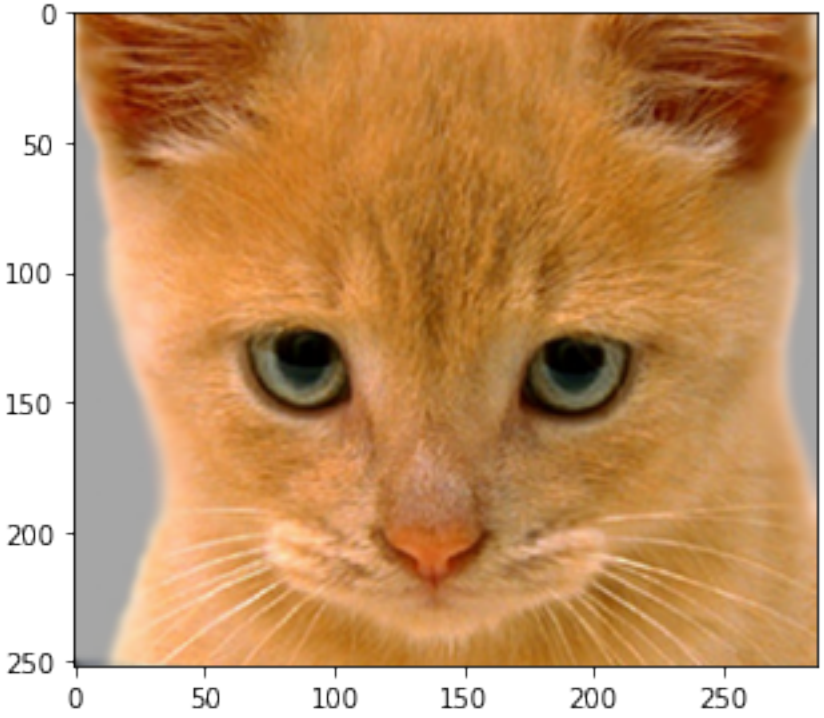
Part 1: Image filtering



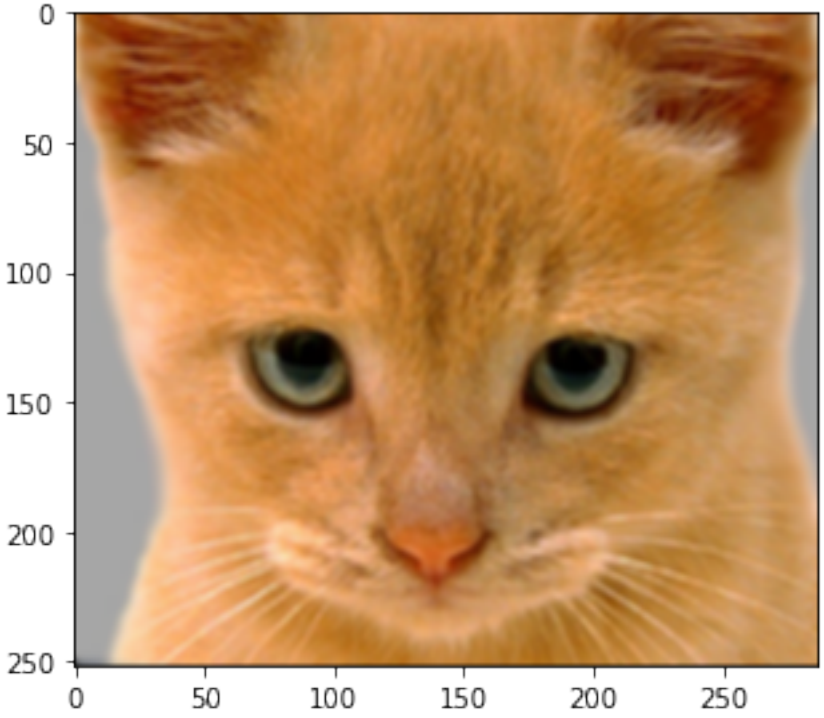
My_conv2d_numpy() applies a single 2d filter to each channel of an image. In our case, it was a 2d Gaussian kernel, which is the graph on the bottom. My function first pads the input image with (kernel.shape[0] // 2) 0s on both left and right sides and with (kernel.shape[1] // 2) on both top and bottom sides. This ensures that after applying the filter/kernel, the filtered image will have the same dimensions as the input image. I take advantage of numpy's vectorization to apply the filter to each channel for each pixel. Applying the filter at the center of each pixel of the input image is essentially using a stride of 1. The filtered image is then returned.

Part 1: Image filtering

Identity filter

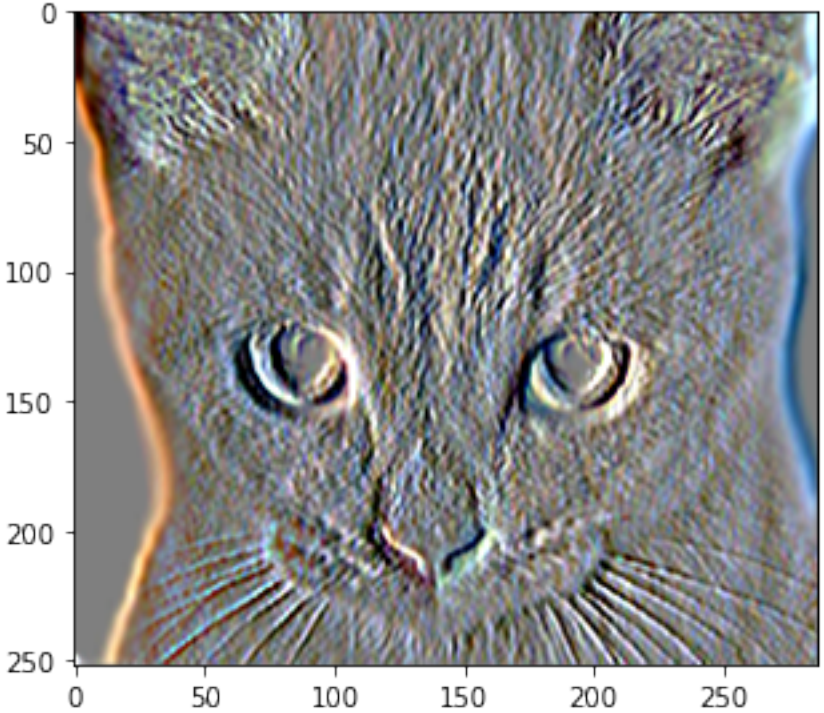


Small blur with a box filter

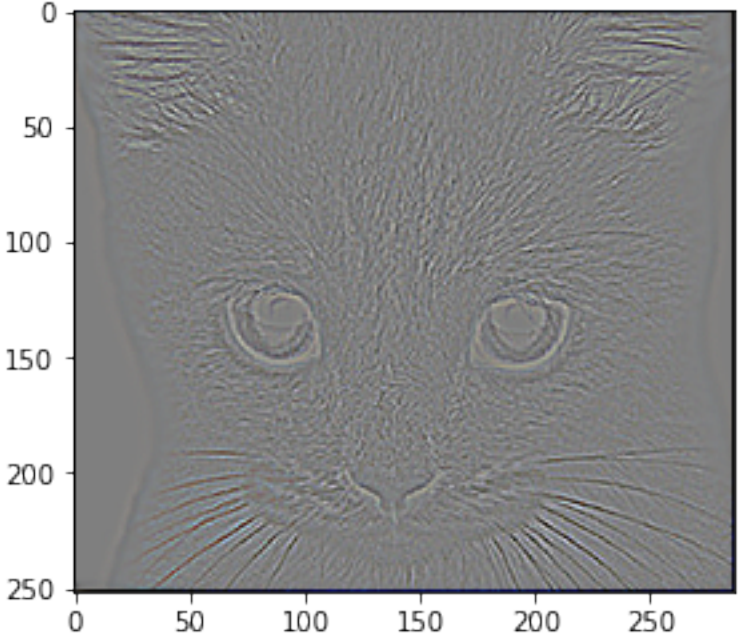


Part 1: Image filtering

Sobel filter



Discrete Laplacian filter

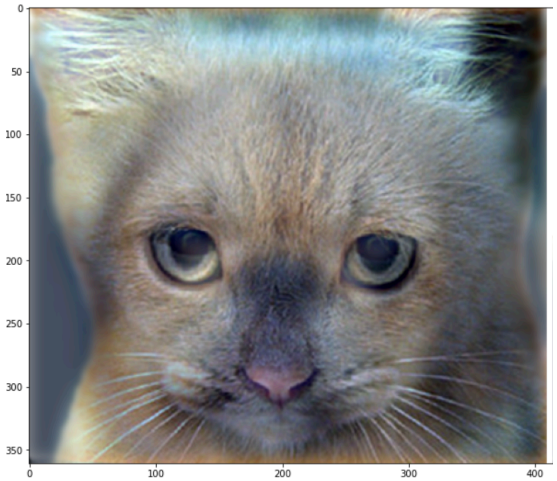


Part 1: Hybrid images

1. Use my_conv2d_numpy to filter image1 with the input Gaussian filter to get the low frequency image.
2. To get the high frequency image, obtain a low frequency image of image2 like in step 1. Subtract low frequency of image2 from the original image2.
3. To get the hybrid image, sum the low frequency image and the high frequency image.

To ensure output values of hybrid image is in the range for matplotlib visualizations, I used the np.clip() function to set all negative values to 0 and all values greater than 1 to 1.

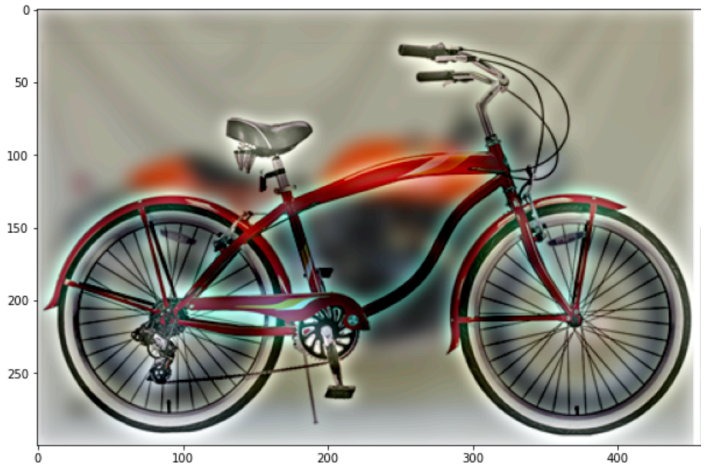
Cat + Dog



Cutoff frequency: 7

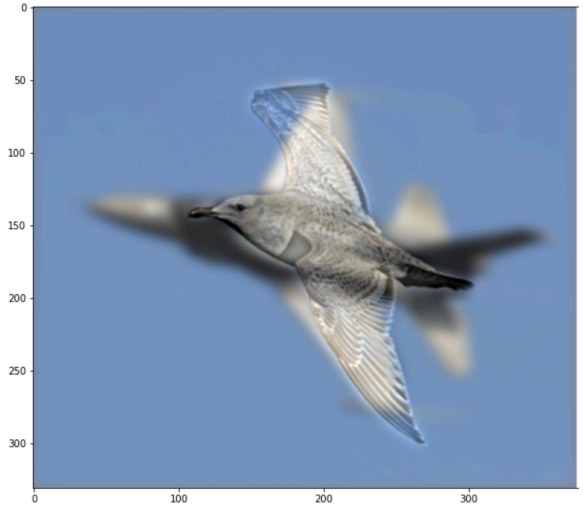
Part 1: Hybrid images

Motorcycle + Bicycle



Cutoff frequency: 6

Plane + Bird



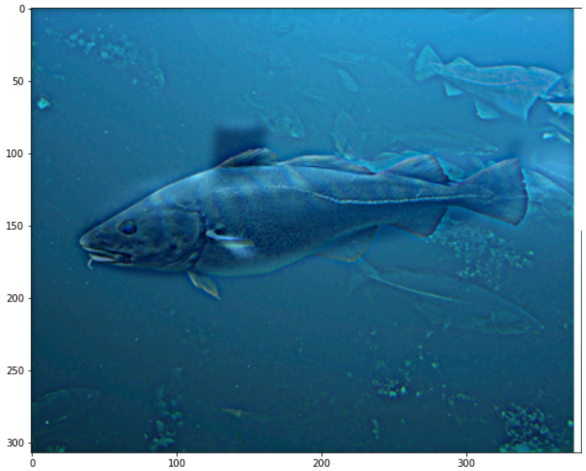
Cutoff frequency: 4

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

The runtime for linearly applying the filter with part1 took 7.103 seconds. The runtime for applying the filter with pytorch 2.486 seconds. I believe this is because pytorch takes advantage of GPUs and other accelerators to speed up calculations. For applying a filter to an image that has 3 channels, I believe pytorch applied the filter on the 3 channels in parallel, taking advantage of multithreaded processing. The runtimes would support this theory as it took around 1/3rd the time for pytorch to run opposed to linearly applying the filter with numpy.

Part 3

[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:
The dog image will always be an input of (361, 410, 3) corresponding to (height, width, channels)
Stride = 1, padding = 0 → (359, 408, 3)
Stride = 2, padding = 0 → (180, 204, 3)
Stride = 1, padding = 1 → (361, 410, 3)
Stride = 2, padding = 1?] → (181, 2055, 3)

Part 3

We applied 4 filters to the dog image.

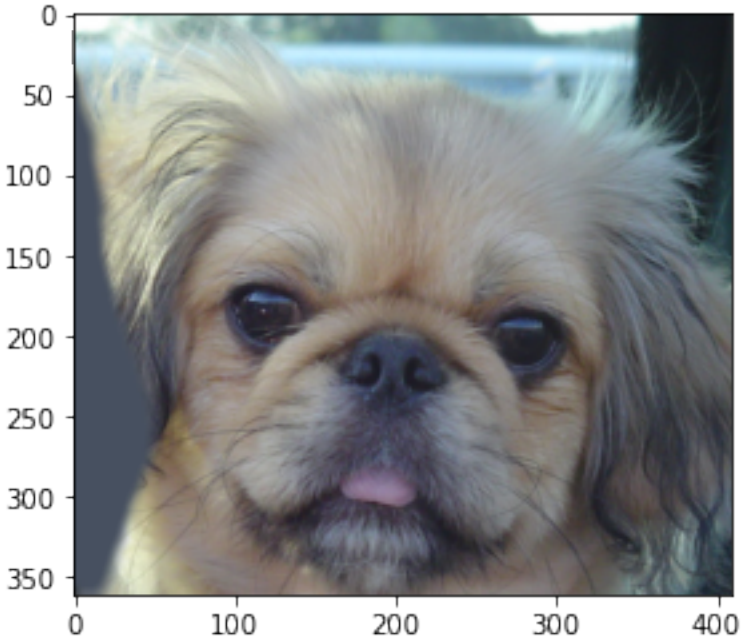
[Why do the output dimensions adhere to the equations given in the instructions handout?] The dimensions work out like the instruction in the handout because the groups number was equal to the number of channels. Since they were equal, we just need to stack our 4 filters 3 times in the filter bank and pytorch’s conv2d will apply all four filters to each channel of the image.

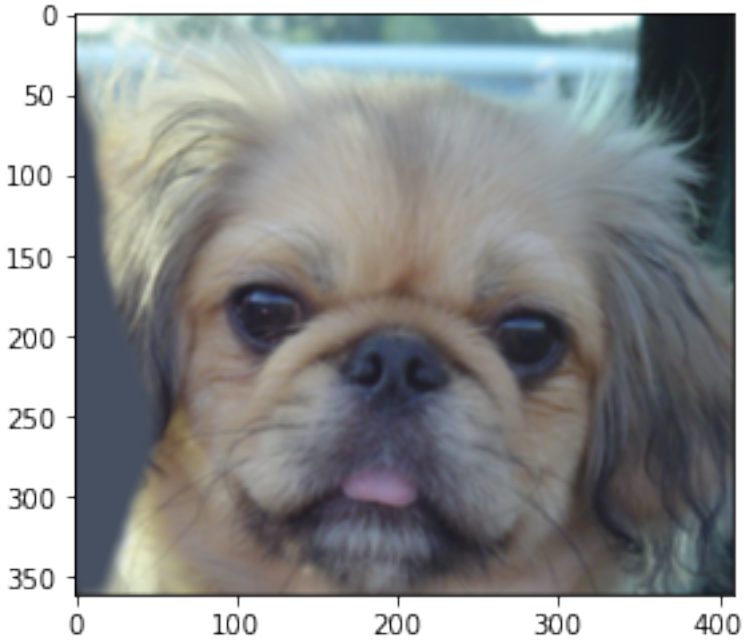
Part 3

[What is the intuition behind this equation?]

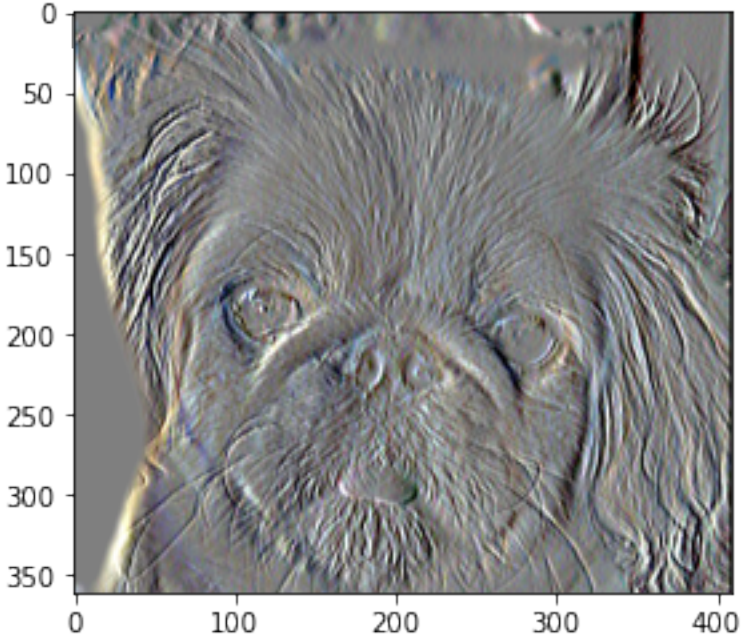
The intuition is how padding and stride affect the filter application process. Since a filter must have odd dimensions, the middle block of the filter is the selected pixel. Then there are even numbers of blocks on 4 sides of the center block. These are the numbers that we need to pad the image by if we want our output to have the same dimensions as our input. For terminology, if we say we add 1 padding vertically, we pad the top and the bottom, which corresponds to the 2*p. For stride = 1, there really is no change. But for stride = 2, we skip applying the filter every other pixel which effectively cuts the image in half. This is why there is a divide by stride number. The last +1 is to account for the subtracting the entire width/height of the filter.

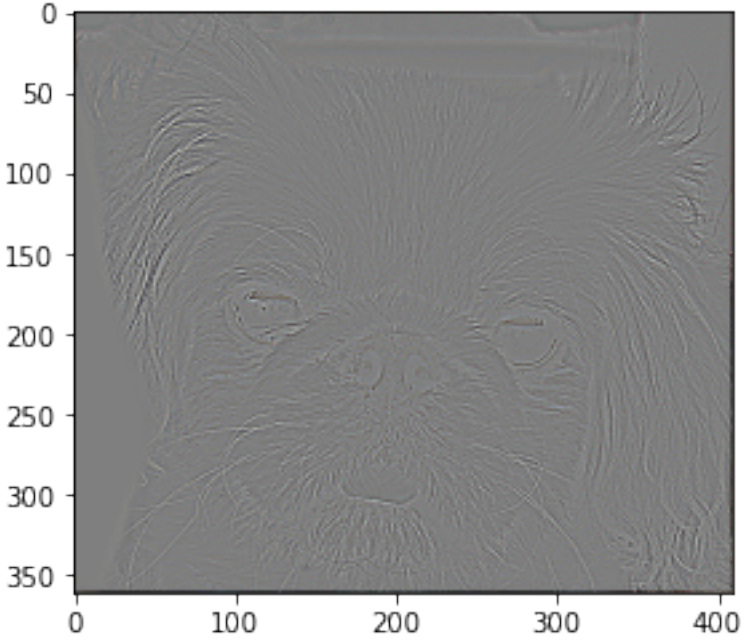
Part 3


0
50
100
150
200
250
300
350
0
100
200
300
400


0
50
100
150
200
250
300
350
0
100
200
300
400

Part 3


0
50
100
150
200
250
300
350
0
100
200
300
400


0
50
100
150
200
250
300
350
0
100
200
300
400

Conclusion

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

Increasing the cutoff frequency increases the size of the filter. The smaller the cutoff frequency, the more visible the first image will be because the Gaussian filter applied on image1 is a lot smaller (thus less blurring effect) than a bigger one. The larger the cutoff frequency, the more image1 is blurred, and the sharper image2 gets because image2.
The high frequency details of image2 are visible when the viewer is closer and less visible with the viewer is further. If we swap the images within a pair, image2 will be visible when viewer is far away and image1 will be visible when viewer is close.