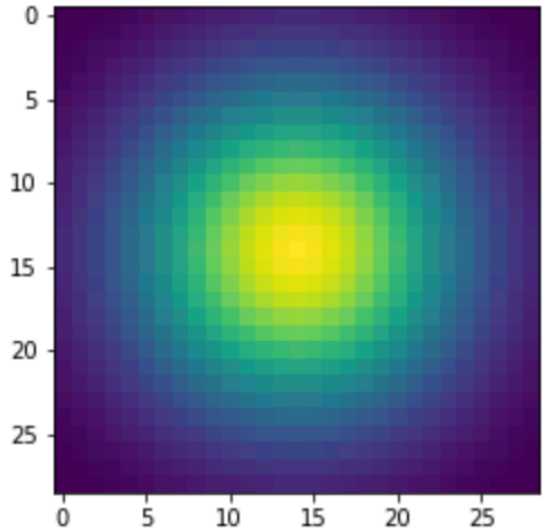
CS 4476 Project 1

Ashwin Rathie
Ashwin.rathie@gatech.edu
arathie6
903281887

Part 1: Image filtering

<insert visualization of Gaussian kernel from proj1.ipynb here>

```
Success -- kernel values are correct.
True
```



<Describe your implementation of my_imfilter() in words.>

Overall, the point of the my_imfilter() method was to 1) perform the “sliding” of the kernel over the entire image, 2) calculating the new pixel value using convolution, and 3) assigning the pixel value to the new image.

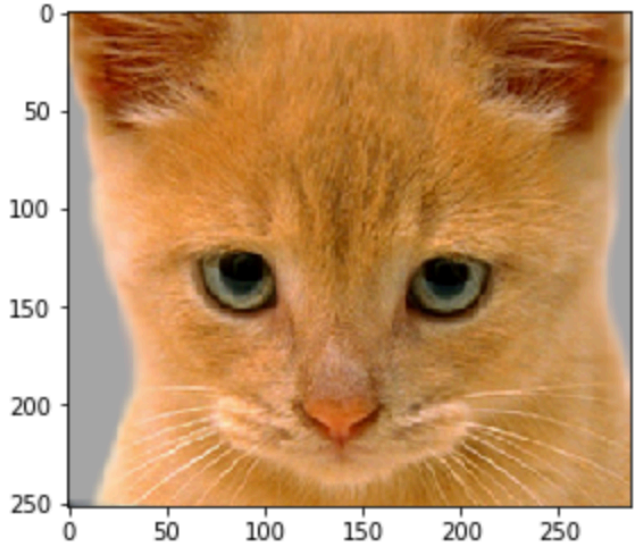
My implementation pads the image with the appropriate kernel dimension//2 on each side of the image.

It then looks at each pixel and each channel of each pixel as well as the surrounding pixels, the “window”, performs the dot product of the kernel and the ”window”, sums it up and assigns it to the filtered image.

Part 1: Image filtering

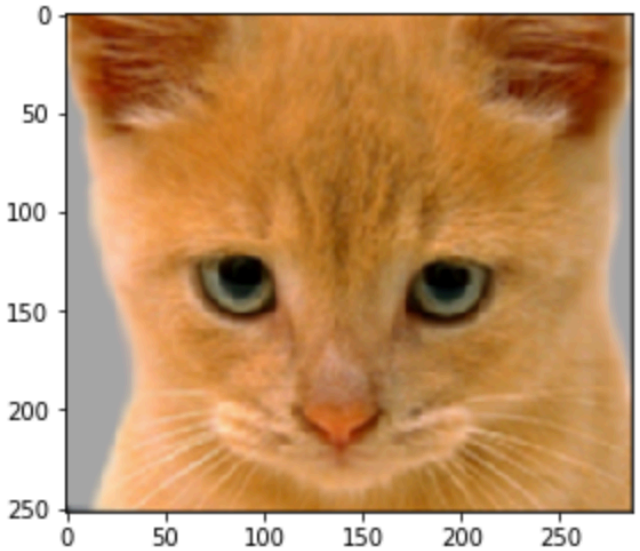
Identity filter

<insert the results from proj1_test_filtering.ipynb using 1b_cat.bmp with the identity filter here>



Small blur with a box filter

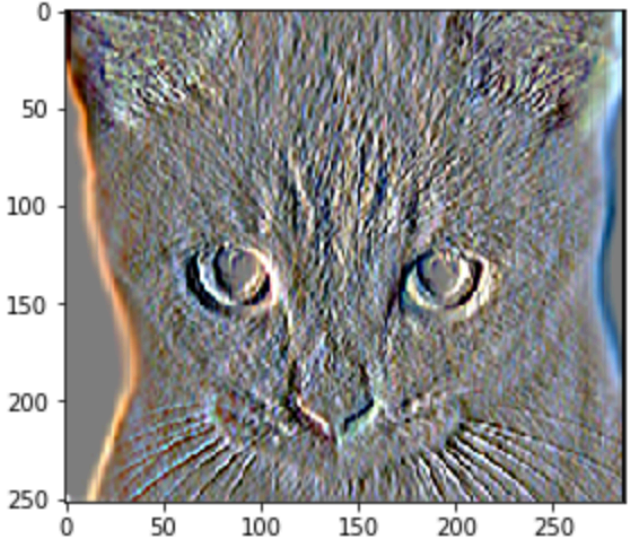
<insert the results from proj1_test_filtering.ipynb using 1b_cat.bmp with the box filter here>



Part 1: Image filtering

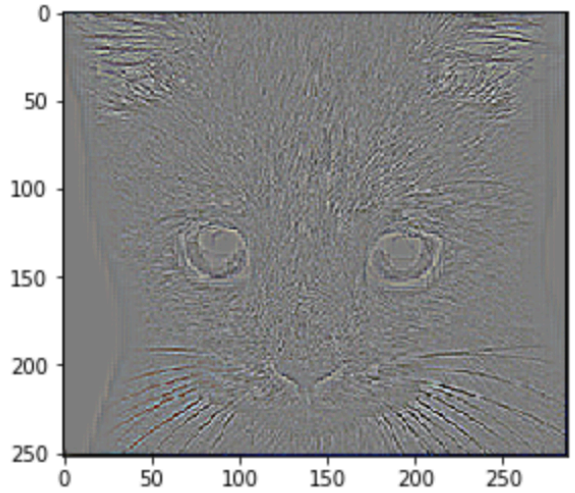
Sobel filter

<insert the results from proj1_test_filtering.ipynb using 1b_cat.bmp with the Sobel filter here>



Discrete Laplacian filter

<insert the results from proj1_test_filtering.ipynb using 1b_cat.bmp with the discrete Laplacian filter here>



Part 1: Hybrid images

<Describe your implementation of create_hybrid_image() here.>

I created the low_freq image by calling my_imfilter() on image1 and the kernel. I then created the high_freq image by subtracting the result of a call to my_imfilter() on image2 and the kernel from the original image 2 (orig – low_freq = high_freq). The hybrid was created by simply adding the low_freq and high_freq and clipping the values to be between 0 and 1.

Cat + Dog

<insert your hybrid image here>

Cutoff frequency: 7

Part 1: Hybrid images

Motorcycle + Bicycle

<insert your hybrid image here>

Cutoff frequency: 4

Plane + Bird

<insert your hybrid image here>

Cutoff frequency: 9

Part 1: Hybrid images

Einstein + Marilyn

<insert your hybrid image here>

Cutoff frequency: 4

Submarine + Fish

<insert your hybrid image here>

Cutoff frequency: 4

Part 2: Hybrid images with PyTorch

Cat + Dog

<insert your hybrid image here>

Motorcycle + Bicycle

<insert your hybrid image here>

Part 2: Hybrid images with PyTorch

Plane + Bird

<insert your hybrid image here>

Einstein + Marilyn

<insert your hybrid image here>

Part 2: Hybrid images with PyTorch

Submarine + Fish

<insert your hybrid image here>

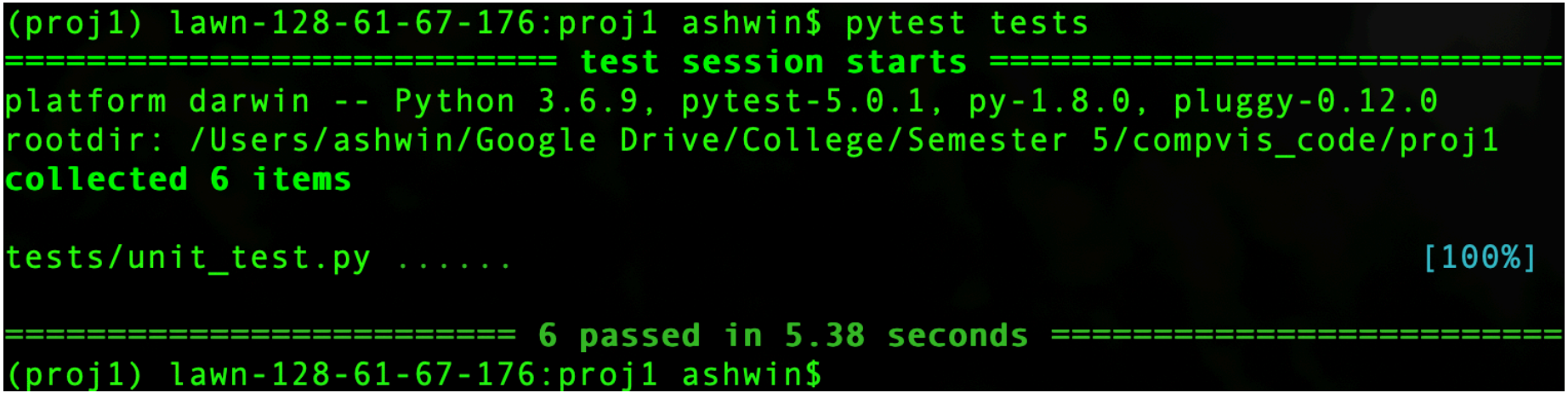
Part 1 vs. Part 2

<Compare the run-times of Parts 1 and 2 here, as calculated in proj1.ipynb. What can you say about the two methods?>

Part 1 ran in 8.384s while part 2 ran in 1.683s. It seems that the tensor/PyTorch method is significantly faster. This could be due to code optimizations in the PyTorch library (using conv2d() instead of our own manual method of convolution), as well as possibly quicker mathematical operations using the Tensor structure rather than ndarrays.

Tests

<Provide a screenshot of the results when you run `pytest tests` on your final code implementation (note: we will re-run these tests).>



Conclusions

<Describe what you have learned in this project. Consider questions like how varying the cutoff frequency value or swapping images within a pair influences the resulting hybrid image. Feel free to include any challenges you ran into.>

Through experimentation, it appears that if you want the low frequency image to be more apparent, you should decrease the cutoff frequency (and increase it to have more of the high frequency image). Swapping images within the pair (low_fre image becomes the high_freq image and vice versa) also swaps which image appears more prevalent during closer viewing and which is more prevalent during farther viewing. Overall, this was in interesting project and a good intro to PyTorch

One of the significant challenges I ran into was when writing the get_item() method. The code comment instructions said to normalize the pixels and transpose the dimensions before transforming the image into a Tensor. Only after much, much debugging, experimentation, and trial and error *did I realize that the ToTensor() transformation method transposes and normalizes for you, and that trying to transpose and normalize yourself before calling the transformation actually causes problems.*