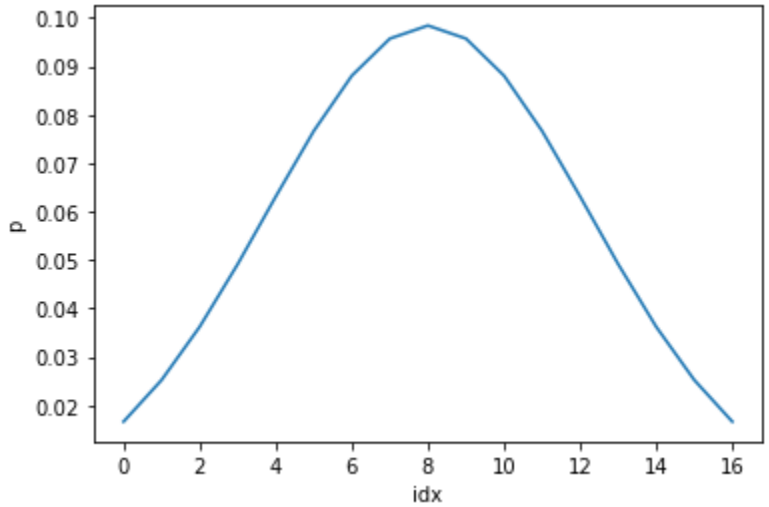
CS x476 Project 1

Rakshit Dabhi
rdabhi3@gatech.edu
rdabhi3
903491952

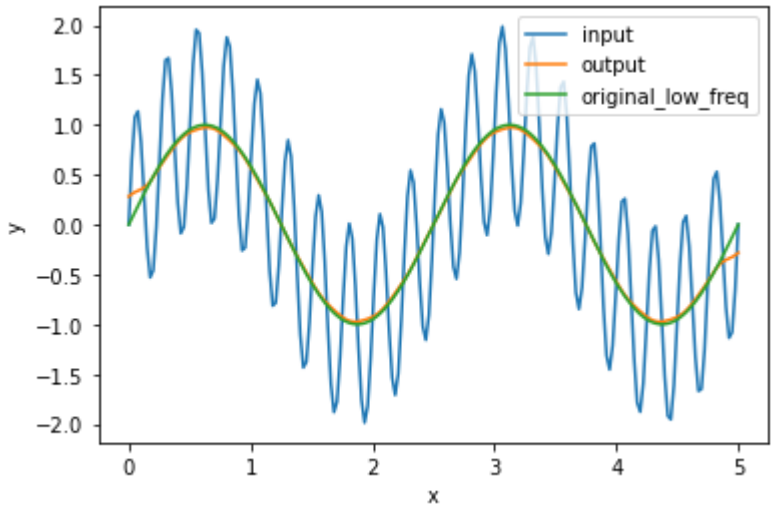
Part 1: 1D Filter

<insert visualization of the low-pass filter from proj1.ipynb here>



Part 1: 1D Filter

<insert visualization of filtered combined signal from proj1.ipynb here>

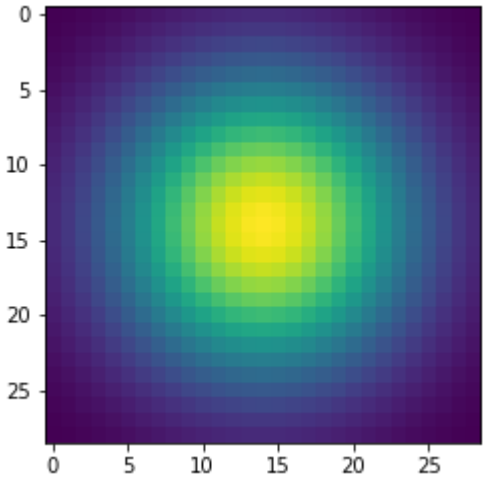


Describe your implementation in words and reflect on the checkpoint questions.

For my 1D filter, I looped through each index of the 1D FloatTensor while aligning the middle index of the kernel with the current index in the loop. For each alignment, I multiplied each element of the kernel with its corresponding element in the 1D FloatTensor, and then summed them to calculate the new pixel value of the current index in the loop. After running though each element in the 1D FloatTensor, I was able to calculate a new filtered signal.

Part 2: Image Filtering

<insert visualization of the 2D Gaussian kernel from proj1.ipynb here>



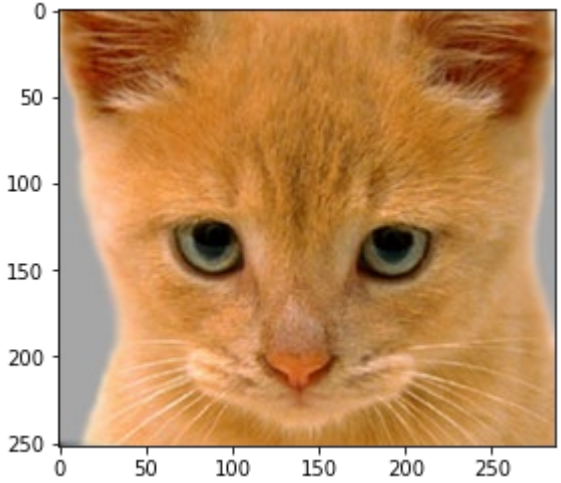
<Describe your implementation of my_imfilter() in words.>

Similar to the 1D filtering, I iterated over each element/pixel in a padded image, while aligning the middle element of the kernel with this pixel. I then multiplied each element in the kernel with its corresponding element in the image and took the weighted sum to find the new pixel value of the current pixel in the loop. After iterating through each pixel in the image, I was able to calculate a new filtered image.

Part 2: Image filtering

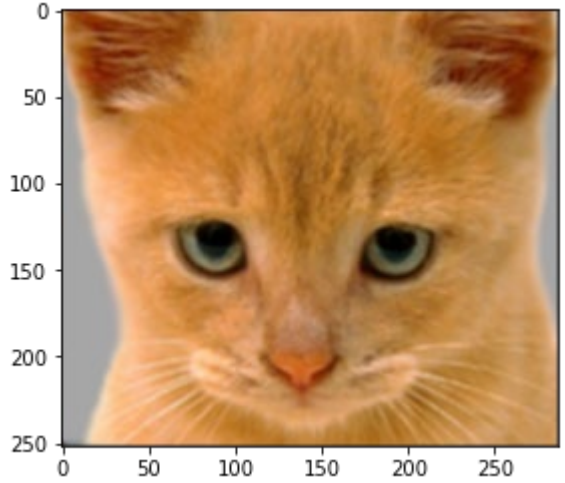
Identity filter

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



Small blur with a box filter

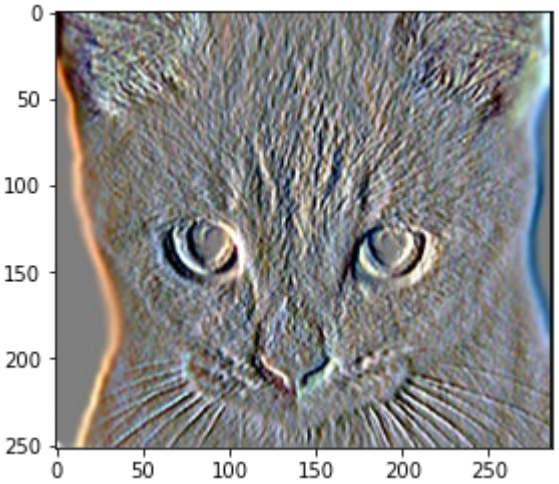
<insert the results from proj1_test_filtering.ipynb using 1b_cat.bmp with the box filter here>



Part 2: Image filtering

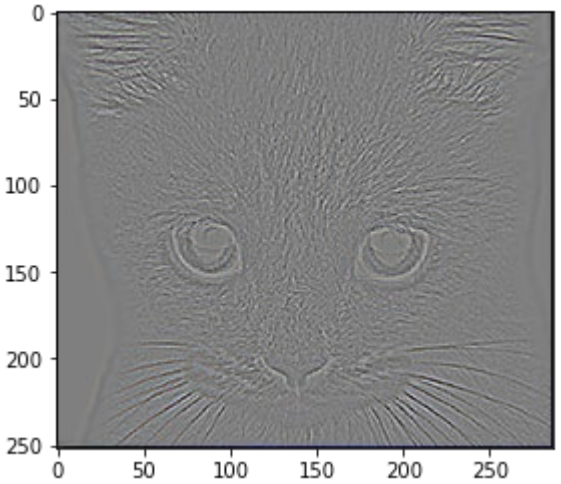
Sobel filter

<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 2: Hybrid images manually using Pytorch

<Describe your implementation of create_hybrid_image() here.>

To create a hybrid image, I first applied a low-pass filter to image1 (dog) and image2 (cat) to create low-frequency images. I then subtracted the low-pass filtered image2 from the original image2 to create a high-frequency image of image2. Then, I added the low-frequency image of image1 with the high-frequency image of image2 to create a hybrid image (while clamping the values of the elements to be between 0 and 1).

Cat + Dog

<insert your hybrid image here>

Cutoff frequency: 7

Part 2: Hybrid images manually using Pytorch

Motorcycle + Bicycle

<insert your hybrid image here>

Cutoff frequency: 5

Plane + Bird

<insert your hybrid image here>

Cutoff frequency: 6

Part 2: Hybrid images manually using Pytorch

Einstein + Marilyn

<insert your hybrid image here>

Cutoff frequency: 3

Submarine + Fish

<insert your hybrid image here>

Cutoff frequency: 4

Part 3: Hybrid images with PyTorch operators

Cat + Dog

<insert your hybrid image here>

Motorcycle + Bicycle

<insert your hybrid image here>

Part 3: Hybrid images with PyTorch operators

Plane + Bird

<insert your hybrid image here>

Einstein + Marilyn

<insert your hybrid image here>

Part 3: Hybrid images with PyTorch operators

Submarine + Fish

<insert your hybrid image here>

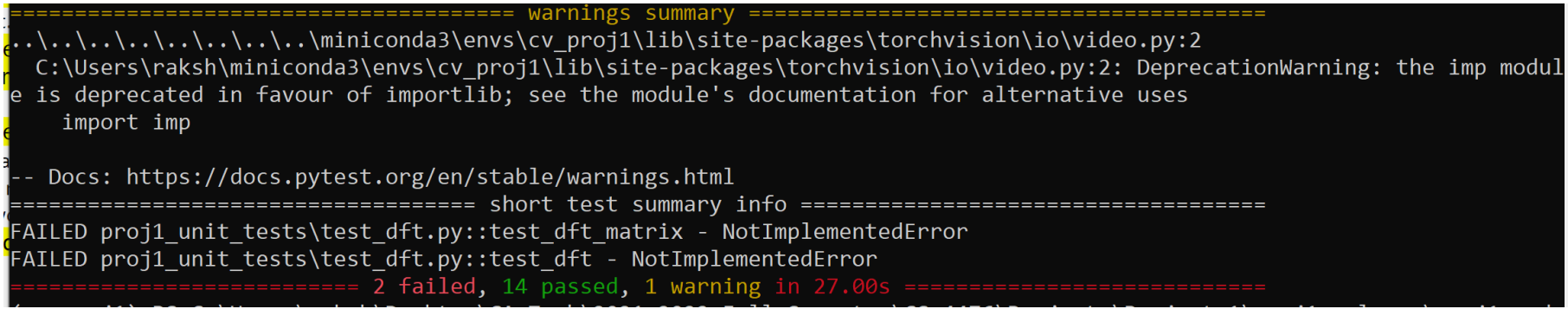
Part 1 vs. Part 2

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

For Part 1, I got a run-time of 27.076 seconds. For Part 2, I got a run-time of 1.190 seconds. From these two run-times, I can say that using the PyTorch operators to create hybrid images is much faster than manually creating hybrid images.

Tests

<Provide a screenshot of the results when you run `pytest proj1_unit_tests/` in proj1_code folder 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 standard deviation value or swapping images within a pair influences the resulting hybrid image. Feel free to include any challenges you ran into.>

From this project, I learned how filtered images are created using kernels. I also learned that varying the standard deviation value creates different hybrid images with varying low-frequency and high-frequency images. In addition, different kernels/filters (ex: identity filter, Sobel filter, discrete Laplacian filter, median filter) will output different resulting images (good for different end goals).

Note

The following slide is:

- REQUIRED for **6476** students
- Extra credits for **4476** students.

Image Filtering using DFT

<insert visualization of the DFT filtered 6a_dog.bmp from proj1.ipynb here>

Describe your implementation in words.

Note

The following slide is:

- Extra credit for **ALL** (4476+6476)

Add some cool hybrid images!