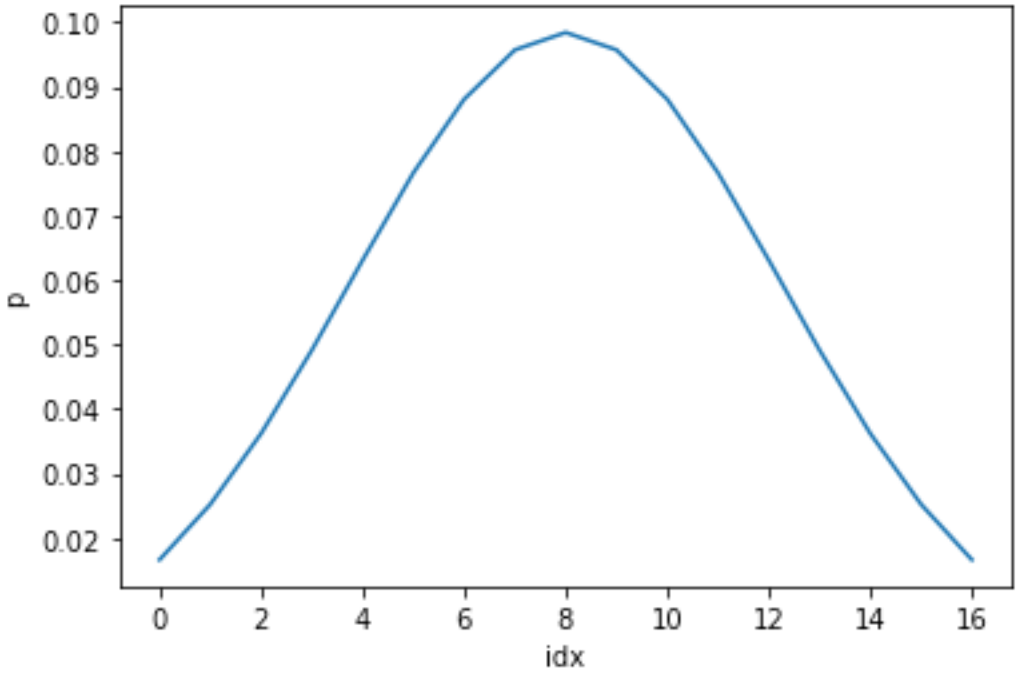
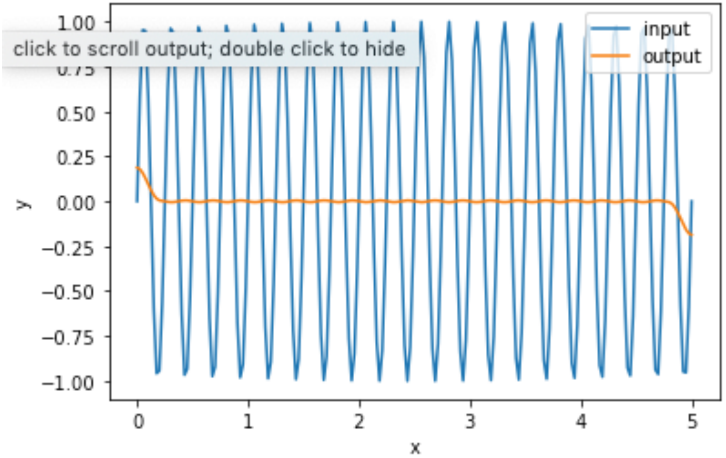
CS 6476 Project 1

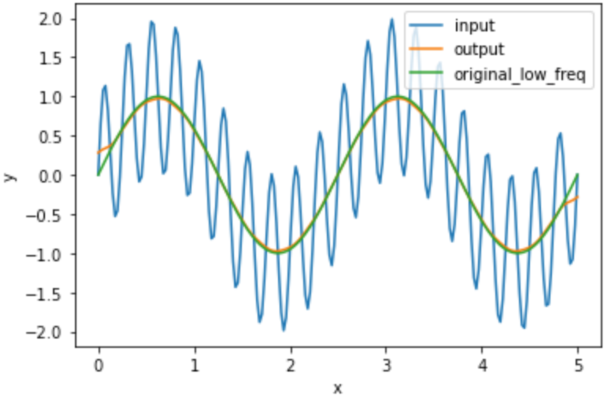
Sai Sandeep Dasari
sdasari38@gatech.eu
sdasari38
903540744

Part 1: 1D Filter



Part 1: 1D Filter





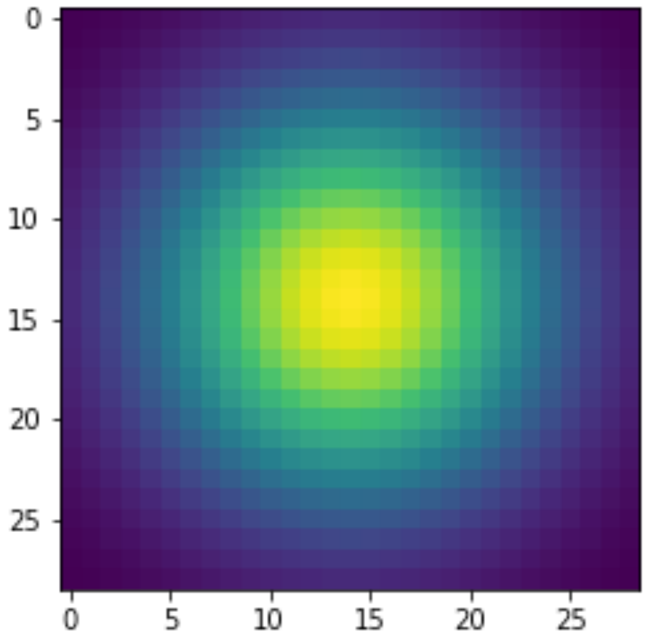
Describe your implementation in words and reflect on the checkpoint questions.

The filter attenuates the high-frequencies in the signal heavily but leaves the low-frequency signal largely untouched. The extreme ends of the low-frequency signal slightly differ and can be seen in the image.

All unit-tests pass with “Correct”

Part 2: Image Filtering

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



1. Padding size is precalculated based on the kernel size. - int(kernel_size/2). Padding is added using torch.nn.functional.pad

2. To improve time complexity of filter (m*n), the filter is flattened before looping over the image

3. Two for loops loop over the padded_image pixel by pixel (sliding filter)

    1. Slice a window of kernel_size

    2. Flatten it and perform dot product to apply filter (for all 3 channels)

Part 2: Image filtering

Identity filter

Small blur with a box filter

Part 2: Image filtering

Sobel filter

Discrete Laplacian filter

Part 2: Hybrid images manually using Pytorch

Create_hybrid_image method uses

1. my_imfilter for the lowpass filtering image1 and image2

2. High frequencies of image2 are calculated by image2 – lowpass(image2)

3. hybrid_image = lowpass(image1) + highpass(image2)

4. Finally we use torch.clamp(hybrid_image) to clamp the values between 0.0 or 1.0 (values of tensors)

Cat + Dog

Cutoff frequency: 7 (standard_deviation)

Part 2: Hybrid images manually using Pytorch

Motorcycle + Bicycle

Cutoff frequency: 5

Plane + Bird

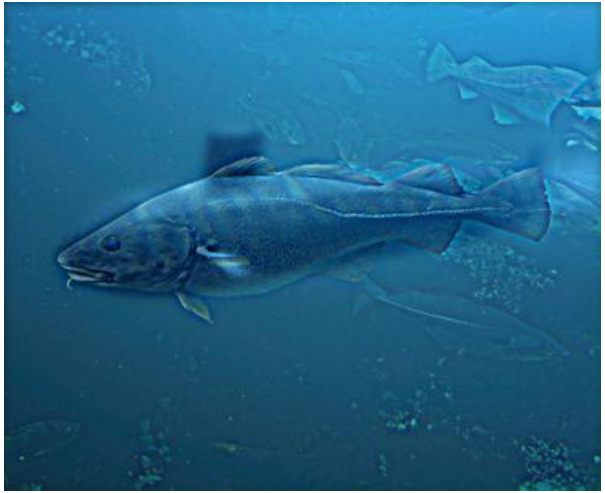
Cutoff frequency: 4

Part 2: Hybrid images manually using Pytorch

Einstein + Marilyn

Cutoff frequency: 3

Submarine + Fish



Cutoff frequency: 3

Part 3: Hybrid images with PyTorch operators

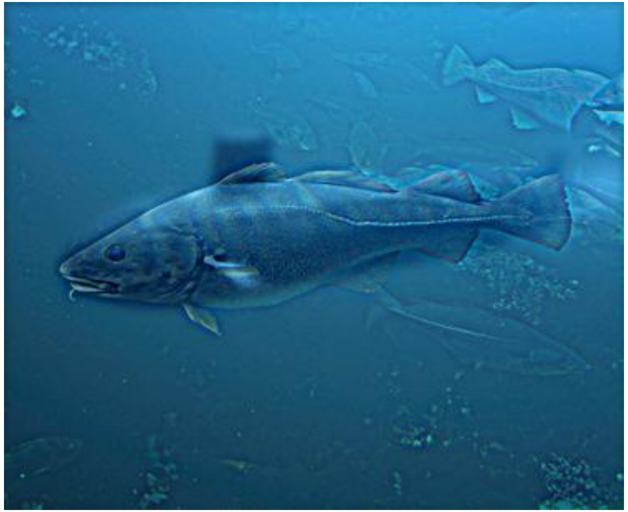
Part 3: Hybrid images with PyTorch operators

Plane + Bird

Einstein + Marilyn

Part 3: Hybrid images with PyTorch operators

Submarine + Fish



Part 1 vs. Part 2

Cat + Dog Timing: Part 1 : 47.688 s Part2: 1.320 s

I personally expected pytorch's implementation to be faster but didn't expect it to be 30 times faster?! Clearly, all the image operations are optimized to the fullest extent.

My filter took m*n iterations to slide over the image, however breaking up the filter into a row vector and a column vector (m + n iterations from Frank's lecture on 09/14) could probably bring the results closer to pytorch's implementation.

Tests

```
(proj1) 2012s-MacBook-Pro:proj1_code a2012$ pytest proj1_unit_tests
============================= test session starts ==============================
platform darwin -- Python 3.6.10, pytest-6.0.1, py-1.9.0, pluggy-0.13.1
rootdir: /Users/a2012/Desktop/cv/proj1_release
collected 13 items

proj1_unit_tests/test_2d.py ......                                        [ 46%]
proj1_unit_tests/test_create_1D_Gaussian_kernel.py ...                    [ 69%]
proj1_unit_tests/test_dft.py FF                                           [ 84%]
proj1_unit_tests/test_my_1dfilter.py ..                                   [100%]
```

Conclusions

This was an amazing intro to computer vision. I learnt how the cutoff frequency and standard deviation relate to the formation of the hybrid images and how to optimize the sliding filter efficiently.

In tweaking the cutoff for images, I learnt that smaller values of cutoff worked better in formation of the hybrid image and corresponding effect of downscaling the image. I ran into challenges with using a FLOAT value for standard deviations and decided to stick to INT only. The hybrid image of eintstein-marilyn (maybe due to differing background colors, color of clothing) was the most challenging to work with for me as the features got too muddy.

Extra Credit

Image Filtering using DFT

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

Describe your implementation in words.

Add some cool hybrid images!