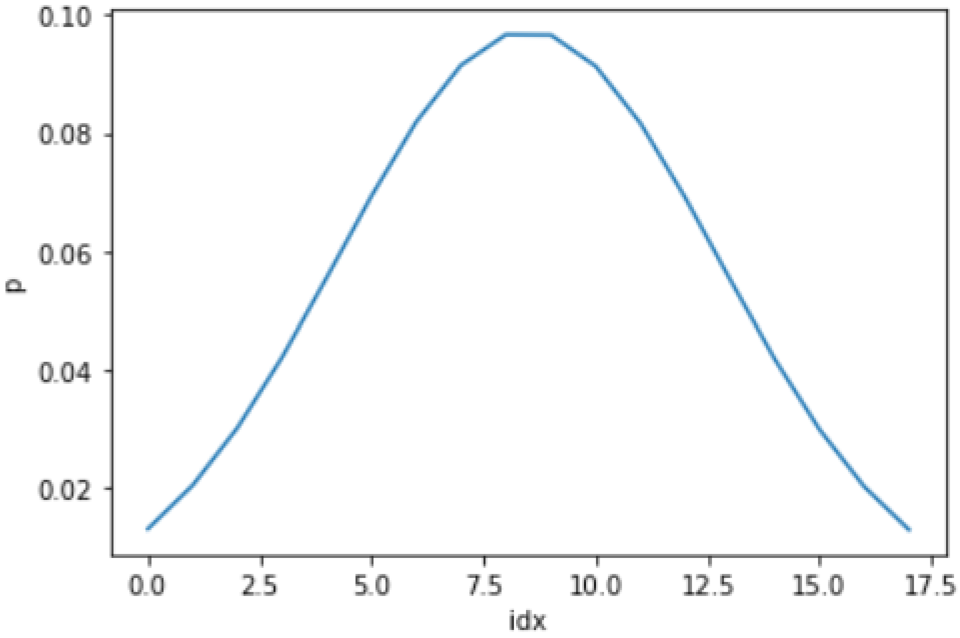
CS x476 Project 1

Bipin Koirala
bkoirala3@gatech.edu
bkoirala3
GT ID: 9037.15.285

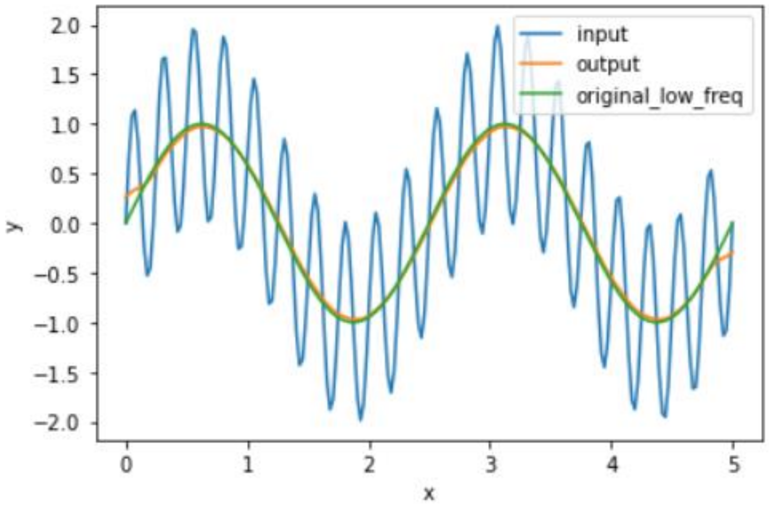
Part 1: 1D Filter

<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>



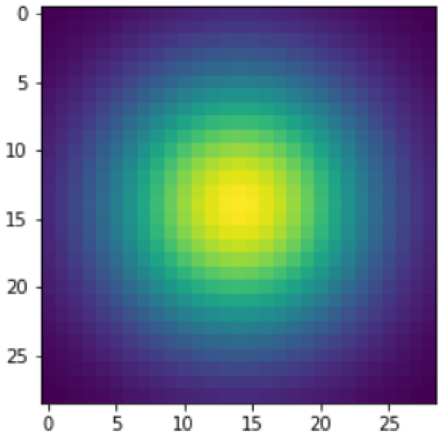
Describe your implementation in words and reflect on the checkpoint questions.

- We created our low pass filter based on one-dimensional normal gaussian distribution function.
- This distribution takes in standard deviation as a function of cut-off frequency. After filtering, it attenuates any signals that are higher than cut-off frequency and retains frequencies that are lower than cut-off frequency.
- We define kernel parameter for filtering based on standard deviation.
- After kernel is setup, we pad the signal in 1D so that the filtered signal do not lose its resolution.
- Then, we correlate the padded signal with kernel to achieve gaussian distribution in the overall signal

Part 2: Image Filtering

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

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



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

my_imfilter() takes in an image, applies filter and returns the filtered image after processing.

I first created a tensor of same shape like the original image. Then row pads and column pads were created based on the formula:

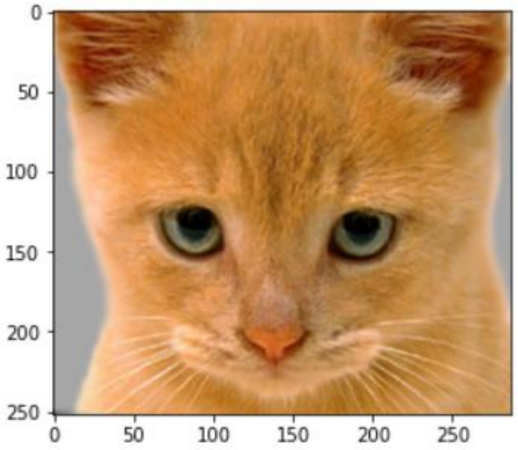
#pad = (#kernel-1)/2; since the output image and input image must have same size.

Then I padded zeros around the image using F.pad function followed by loop over each rows and columns to implement the filter.

Part 2: Image filtering

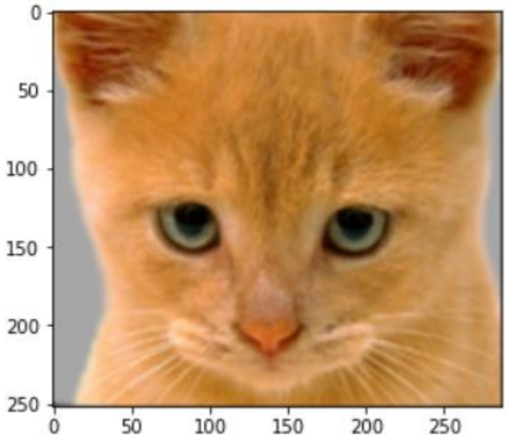
Identity filter

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



Small blur with a box filter

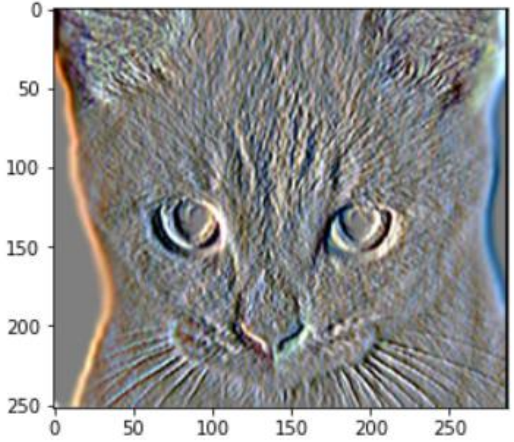
<insert the results from proj1_test_filtering.ipynb using 1b_cat.bmp with the box filter here>



Part 2: Image filtering

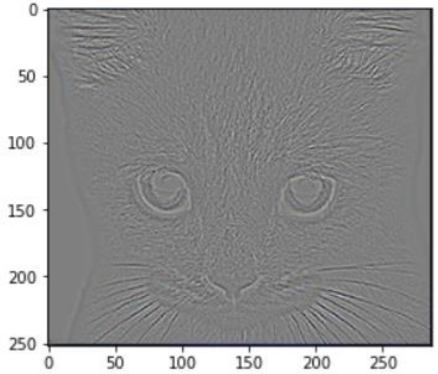
Sobel filter

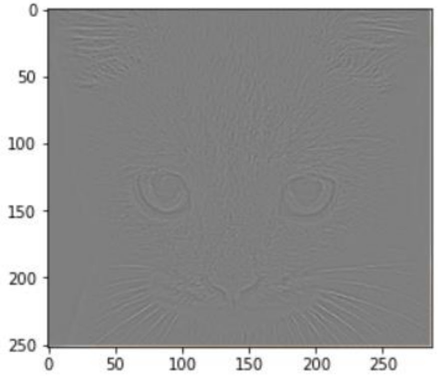
<insert the results from proj1_test_filtering.ipynb using 1b_cat.bmp with the Sobel filter here>



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

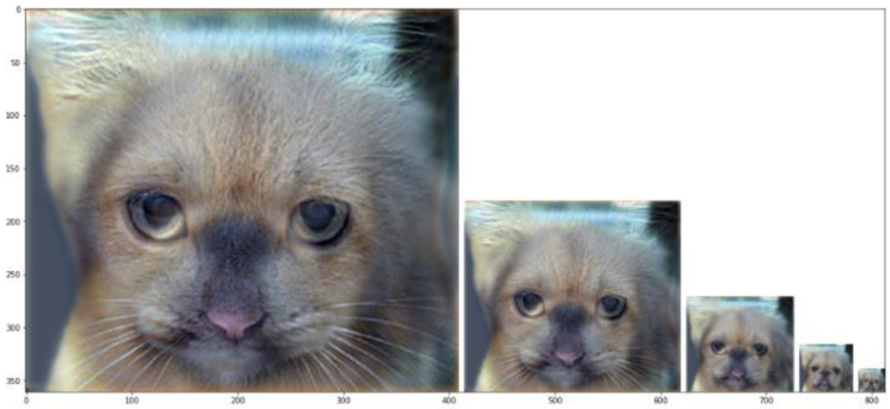
<Describe your implementation of create_hybrid_image() here.>

First two images i.e. image1 and image 2 corresponding to dog and cat were loaded along with a low pass gaussian filter. Both images were appied with low filter. However, in the case of cat image the low pass filtered tensors were removed from the original image tensors. Thus, creating a high pass filter in this case.

Finally both images were stacked together to create a hybrid image that contained low frequencies of dog image and high frequencies of cat image.

Cat + Dog

<insert your hybrid image here>



Cutoff frequency: 5. Although, using 7 produced a more consistent cat image on the first plot

Part 2: Hybrid images manually using Pytorch

Motorcycle + Bicycle

<insert your hybrid image here>

Cutoff frequency: 4

Plane + Bird

<insert your hybrid image here>

Cutoff frequency: 8

Part 2: Hybrid images manually using Pytorch

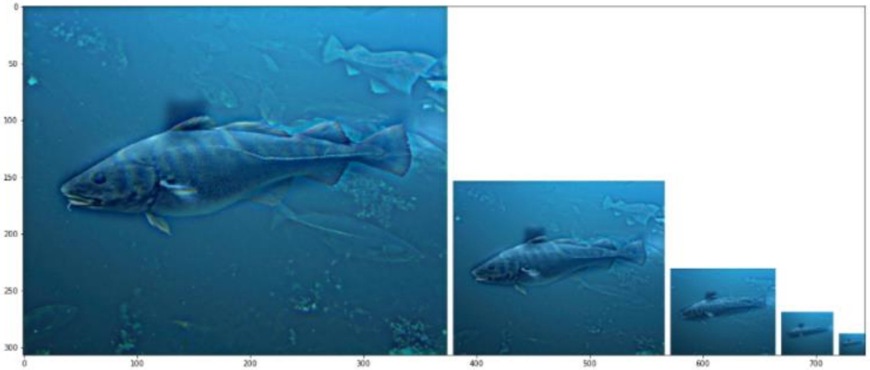
Einstein + Marilyn

<insert your hybrid image here>

Cutoff frequency: 4

Submarine + Fish

<insert your hybrid image here>



Cutoff frequency: 4

Part 3: Hybrid images with PyTorch operators

Cat + Dog

Cutt-off frequency: 4

Motorcycle + Bicycle

Cutt-off frequency: 2

Part 3: Hybrid images with PyTorch operators

Plane + Bird

Cutt-off frequency: 6

Einstein + Marilyn

Cutt-off frequency: 4

Part 3: Hybrid images with PyTorch operators

Submarine + Fish

Cutt-off frequency: 2

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?>

```
In [80]: start = time.time()
cutoff_standarddeviation = 7
kernel = create_2d_gaussian_kernel(cutoff_standarddeviation)
low_frequencies, high_frequencies, hybrid_image = create_hybrid_image(image1,
end = time.time() - start
print('Part 1: {:.3f} seconds'.format(end))
```

```
Part 1: 29.718 seconds
```

Timing Part 3

```
In [81]: model = HybridImageModel()

start = time.time()
low_frequencies, high_frequencies, hybrid_image = model(image_a, image_b, torc
end = time.time() - start
print('Part 2: {:.3f} seconds'.format(end))
```

```
Part 2: 1.423 seconds
```

Runtime for part 3 is 19.88% faster than runtime for part 2. It is because we have created the filter manually in part 2 and this method of implementing kernel contributes to slower computing because of lack of robust algorithm on user’s end. In part 2 function is explicitly written but in part 3 the in-built function is simply called.

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).>

```
In [10]: import proj1_unit_tests.test_create_1d_gaussian_kernel as test_create_1d_gaussian_kernel

         print('test_tensor_datatype: ',
               verify(test_create_1d_gaussian_kernel.test_tensor_datatype))
         print('test_create_kernel_with_sigma_int: ',
               verify(test_create_1d_gaussian_kernel.test_create_kernel_with_sigma_int))
         print('test_kernel_sum: ',
               verify(test_create_1d_gaussian_kernel.test_kernel_sum))

         test_tensor_datatype:  "Correct"
         test_create_kernel_with_sigma_int:  "Correct"
         test_kernel_sum:  "Correct"
```

```
In [16]: import proj1_unit_tests.test_my_1d_filter as test_my_1d_filter

         print('test_filter_with_box_kernel: ',
               verify(test_my_1d_filter.test_filter_with_box_kernel))
         print('test_filter_with_asymmetric_kernel: ',
               verify(test_my_1d_filter.test_filter_with_asymmetric_kernel))

         test_filter_with_box_kernel:  "Correct"
         test_filter_with_asymmetric_kernel:  "Correct"
```

Select Anaconda Prompt (miniconda3)

```
    from proj1_code.student_code import my_median_filter
student_code.py:13: in <module>
    from utils import load_image, save_image
E   ModuleNotFoundError: No module named 'utils'
__________________________ ERROR collecting proj1_code/proj1_unit_tests/test_my_1d_filter.py __________________________
ImportError while importing test module 'C:\Users\kbipi\proj1_release\proj1_release\proj1_code\proj1_unit_tests\test_
1d_filter.py'.
Hint: make sure your test modules/packages have valid Python names.
Traceback:
..\..\..\miniconda3\envs\cv_proj1\lib\importlib\__init__.py:126: in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
proj1_unit_tests\test_my_1d_filter.py:5: in <module>
    from student_code import my_1d_filter
E   ModuleNotFoundError: No module named 'student_code'
================================================= warnings summary =================================================
..\..\..\miniconda3\envs\cv_proj1\lib\site-packages\torchvision\io\video.py:2
  C:\Users\kbipi\miniconda3\envs\cv_proj1\lib\site-packages\torchvision\io\video.py:2: DeprecationWarning: the imp mod
e is deprecated in favour of importlib; see the module's documentation for alternative uses
    import imp

-- Docs: https://docs.pytest.org/en/stable/warnings.html
=============================================== short test summary info ===============================================
ERROR proj1_unit_tests\test_2d.py
ERROR proj1_unit_tests\test_create_1d_gaussian_kernel.py
ERROR proj1_unit_tests\test_dft.py
ERROR proj1_unit_tests\test_median_filter.py
ERROR proj1_unit_tests\test_my_1d_filter.py
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Interrupted: 5 errors during collection !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
============================================ 1 warning, 5 errors in 1.33s ============================================
```

Note: due to directory mismatch; I had to remove “proj1_code.” from both ‘proj1.ipynb’ as well as files under proj1_unit_tests; before importing test cases. Unit test ran from .ipynb file are attached here.

In [20]:
```
from student_code import create_2d_gaussian_kernel
import sys
from proj1_unit_tests.test_2d import verify_gaussian_kernel

cutoff_standard_deviation = 7
kernel = create_2d_gaussian_kernel(cutoff_standard_deviation)

# let's take a look at the filter!
plt.figure(figsize=(4,4)); plt.imshow(kernel);

## Verify that the Gaussian kernel was created correctly
print(verify_gaussian_kernel(kernel, cutoff_standard_deviation))
```

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

In [22]:
```
from student_code import create_hybrid_image
from utils import vis_image_scales_numpy

from proj1_unit_tests.test_2d import (
    verify_low_freq_sq_kernel_torch_manual,
    verify_high_freq_sq_kernel_torch_manual,
    verify_hybrid_image_torch_manual)

low_freq_image, high_freq_image, hybrid_image = create_hybrid_image(image1, image2, kernel)
vis = vis_image_scales_numpy(hybrid_image)
## Verify that results are as expected
print(verify_low_freq_sq_kernel_torch_manual(image1, kernel, low_freq_image))
print(verify_high_freq_sq_kernel_torch_manual(image2, kernel, high_freq_image))
print(verify_hybrid_image_torch_manual(image1, image2, kernel, hybrid_image))
```

```
Success! Low frequency values are correct.
True
Success! High frequency values are correct.
True
Success! Hybrid image values are correct.
True
```

In [28]:
```
## Verify that the results are correct, with cutoff_frequency of 7
from proj1_unit_tests.test_2d import (
    verify_low_freq_sq_kernel_pytorch,
    verify_high_freq_sq_kernel_pytorch,
    verify_hybrid_image_pytorch
)

dataset = HybridImageDataset(data_root, cf_file)
dataloader = torch.utils.data.DataLoader(dataset)
image_a, image_b, cutoff = next(iter(dataloader))
low_frequencies, high_frequencies, hybrid_image = model(image_a, image_b, cutoff)

cutoff_sd = torch.Tensor([7])
## On first dog/cat pair, verify that the Pytorch results are as expected
print(verify_low_freq_sq_kernel_pytorch(image_a, model, cutoff_sd, low_frequencies))
print(verify_high_freq_sq_kernel_pytorch(image_b, model, cutoff_sd, high_frequencies))
## Verify that the Pytorch hybrid images are created correctly
print(verify_hybrid_image_pytorch(image_a, image_b, model, cutoff_sd, hybrid_image))
```

```
Success! PyTorch low frequency values are correct.
True
Success! PyTorch high frequency values are correct.
True
Success! PyTorch hybrid image values are correct.
True
```

In [34]:
```
from proj1_unit_tests.test_dft import test_dft_matrix

print(test_dft_matrix())
```

```
Success! The DFT matrix for dimension 4 is correct!
True
```

In [37]:
```
from proj1_unit_tests.test_dft import test_dft

print(test_dft())
```

```
Success! The DFT matrix for A is correct!
True
```

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.>

* Lower values for cutoff standard images makes the first image less blur and more prominent. This also has an opposite effect on high frequency of the second image. The hybrid image shows the first image dominant over the second image.

* Increasing the value of cutoff frequency has the opposite effect. In this case the first image is blurred more, and the second image appears prominent in the hybrid image when viewed closely.

* Being able to manipulate an image based on my own filter design was exciting. I had hard time in implementing dataset for hybrid images.

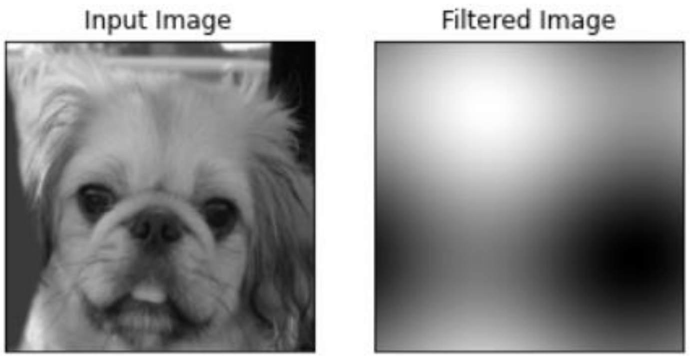
Note

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.

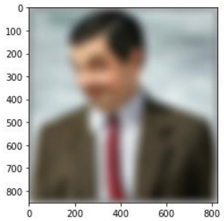
*2D DFT involves complex array structure. Therefore, it needs complex multiplication and addition. First the exponential term was broken down using Euler's rule and computed for all rows and columns in a loop. The real part consisted of cosine term and imaginary part consisted of sine term. Thus generated DFT matrix was used to get the DFT result of an image using complex*real and complex*complex. Then, the filter was applied to the image.

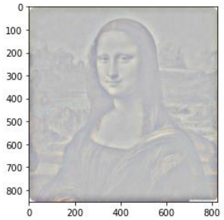
Note

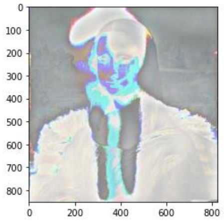
The following slide is:

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

Add some cool hybrid images!







Here is my failed attempt to create hybrid image of Mr.Bean and Mona Lisa. I should have picked a set of image with a similar background and similar subject shape.

Found on web.

https://courses.grainger.illinois.edu/cs445/fa2015/projects/hybrid/ComputationalPhotography_ProjectHybrid.html