Thomas Cousins

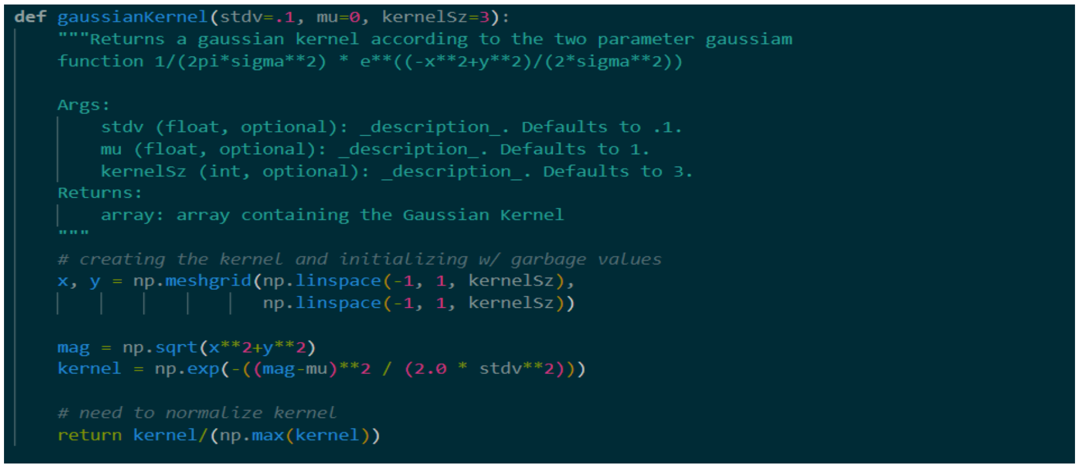
UIN: 927004846

CSCE 748

Programming Assignment 3 Report

Task 1:

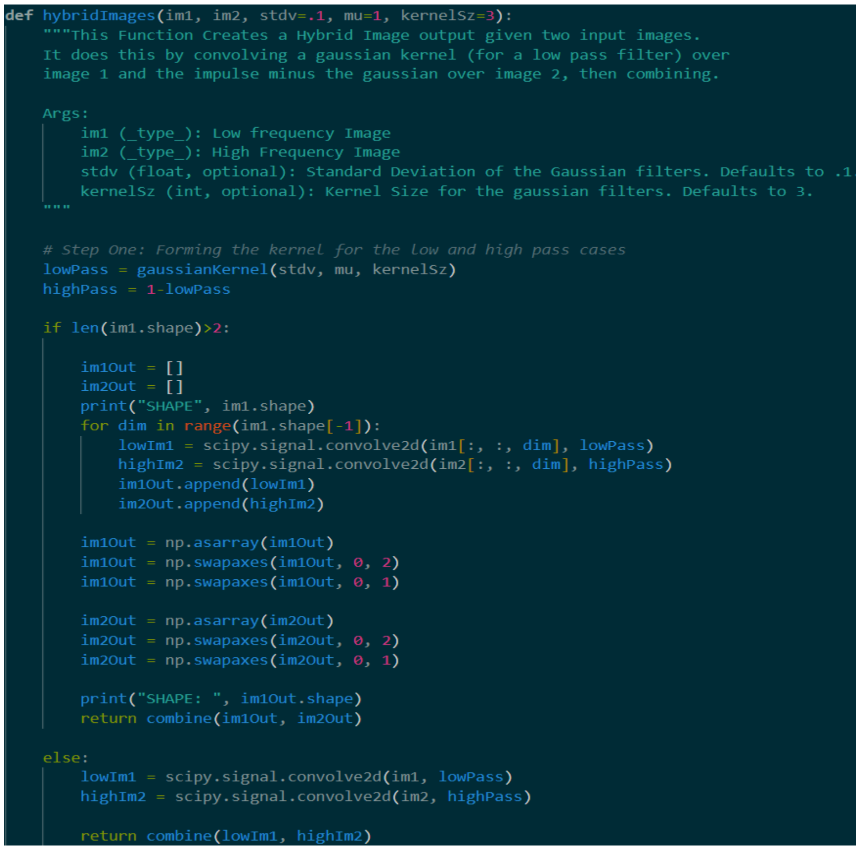
This task consisted of combining two images by applying a low pass filter to one, a high pass filter to the other, and stacking them. These images were usually done in grayscale, though I have some color implementations for the extra credit. To implement this code we first needed to generate the Gaussian kernel (gaussianKernel function) and then code was needed to actually perform the operation (hybridImages).



```python
def gaussianKernel(stdv=.1, mu=0, kernelSz=3):
    """Returns a gaussian kernel according to the two parameter gaussiam
    function 1/(2pi*sigma**2) * e**((-x**2+y**2)/(2*sigma**2))

    Args:
        stdv (float, optional): _description_. Defaults to .1.
        mu (float, optional): _description_. Defaults to 1.
        kernelSz (int, optional): _description_. Defaults to 3.
    Returns:
        array: array containing the Gaussian Kernel
    """
    # creating the kernel and initializing w/ garbage values
    x, y = np.meshgrid(np.linspace(-1, 1, kernelSz),
                       np.linspace(-1, 1, kernelSz))

    mag = np.sqrt(x**2+y**2)
    kernel = np.exp(-((mag-mu)**2 / (2.0 * stdv**2)))

    # need to normalize kernel
    return kernel/(np.max(kernel))
```

Code containing the method for generating the gaussian kernel. The kernel size, standard deviation, and mean can all be specified via parameters.

The code for creating the hybrid image simply convolved this kernel by all axes of the image (1 for grayscale, 3 for RGB) and then combined the images. A separate case had to be added for covering RGB images for the bonus, but the same principle applies.

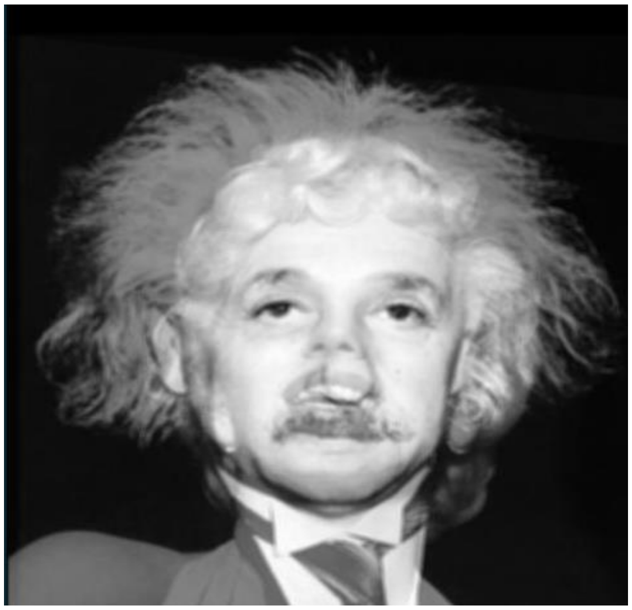


```
def hybridImages(im1, im2, stdv=.1, mu=1, kernelSz=3):
    """This Function Creates a Hybrid Image output given two input images.
    It does this by convolving a gaussian kernel (for a low pass filter) over
    image 1 and the impulse minus the gaussian over image 2, then combining.

    Args:
        im1 (_type_): Low frequency Image
        im2 (_type_): High Frequency Image
        stdv (float, optional): Standard Deviation of the Gaussian filters. Defaults to .1
        kernelSz (int, optional): Kernel Size for the gaussian filters. Defaults to 3.
    """

    # Step One: Forming the kernel for the low and high pass cases
    lowPass = gaussianKernel(stdv, mu, kernelSz)
    highPass = 1-lowPass

    if len(im1.shape)>2:

        im1Out = []
        im2Out = []
        print("SHAPE", im1.shape)
        for dim in range(im1.shape[-1]):
            lowIm1 = scipy.signal.convolve2d(im1[:, :, dim], lowPass)
            highIm2 = scipy.signal.convolve2d(im2[:, :, dim], highPass)
            im1Out.append(lowIm1)
            im2Out.append(highIm2)

        im1Out = np.asarray(im1Out)
        im1Out = np.swapaxes(im1Out, 0, 2)
        im1Out = np.swapaxes(im1Out, 0, 1)

        im2Out = np.asarray(im2Out)
        im2Out = np.swapaxes(im2Out, 0, 2)
        im2Out = np.swapaxes(im2Out, 0, 1)

        print("SHAPE: ", im1Out.shape)
        return combine(im1Out, im2Out)

    else:
        lowIm1 = scipy.signal.convolve2d(im1, lowPass)
        highIm2 = scipy.signal.convolve2d(im2, highPass)

        return combine(lowIm1, highIm2)
```

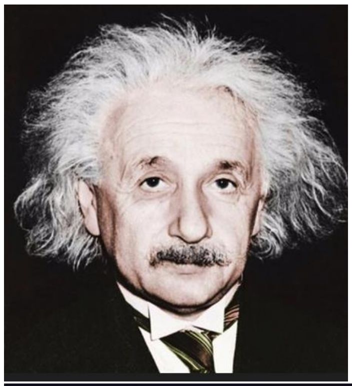
The hybrid Images code. Applies the gaussian kernel and the impulse minus the gaussian kernel to each respective image before summing.

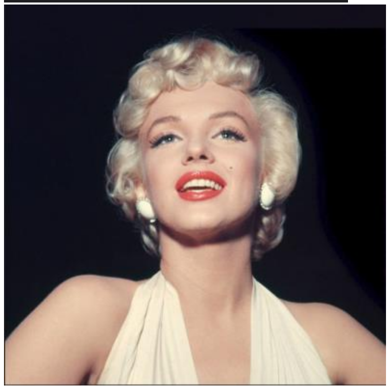
Results:

Monroe Einstein Results:

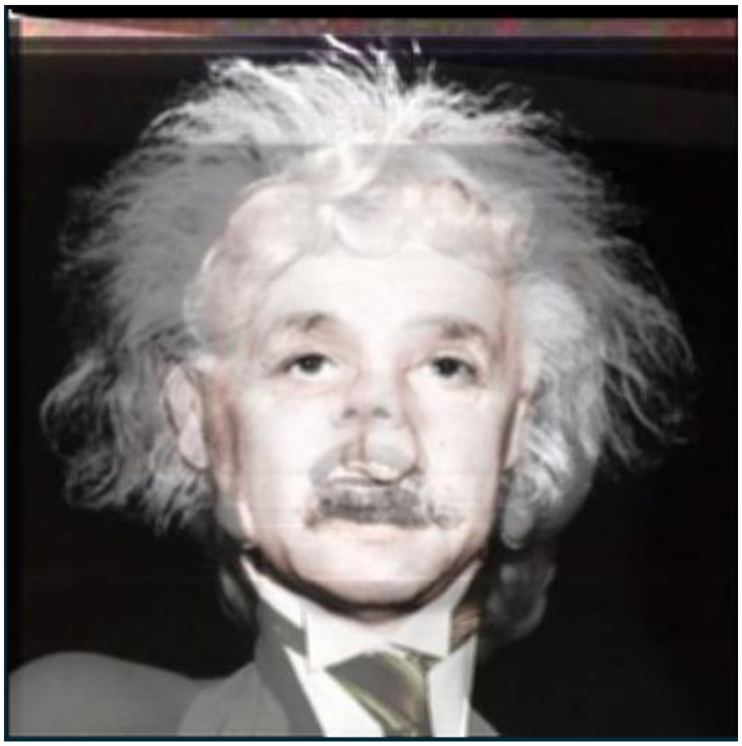


Monroe Einstein Original Images:

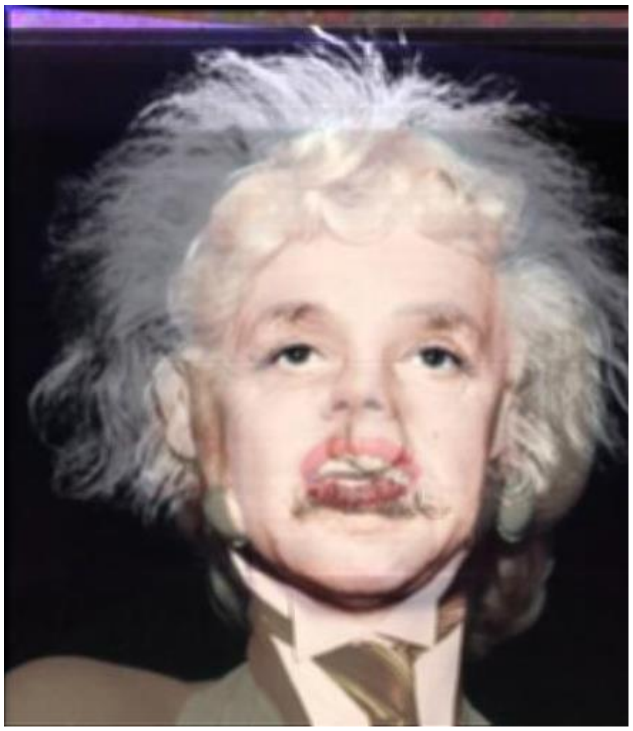




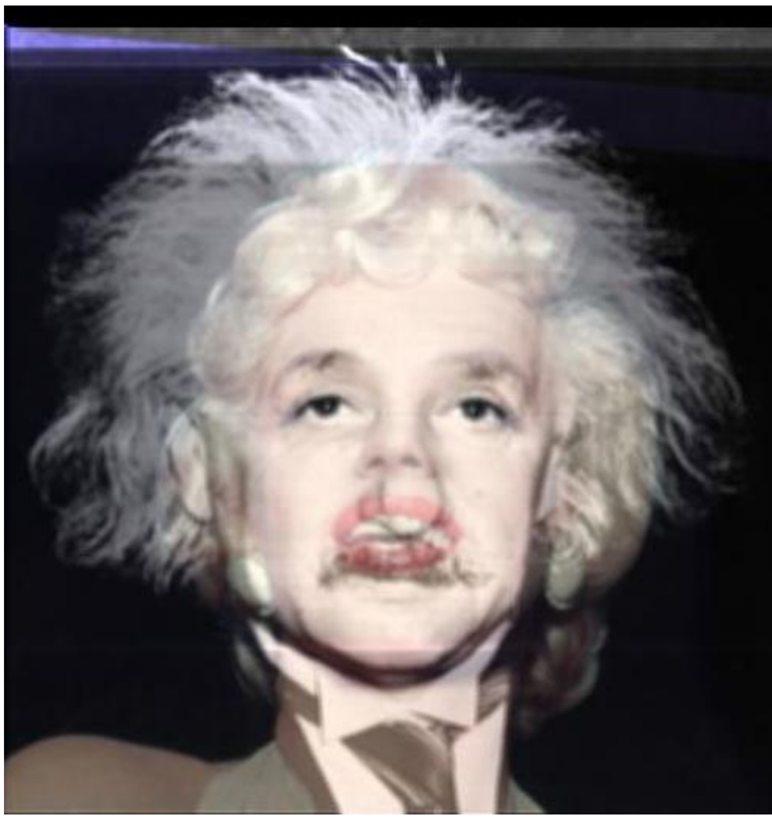
Extra Credit, high pass colored (Einstein):



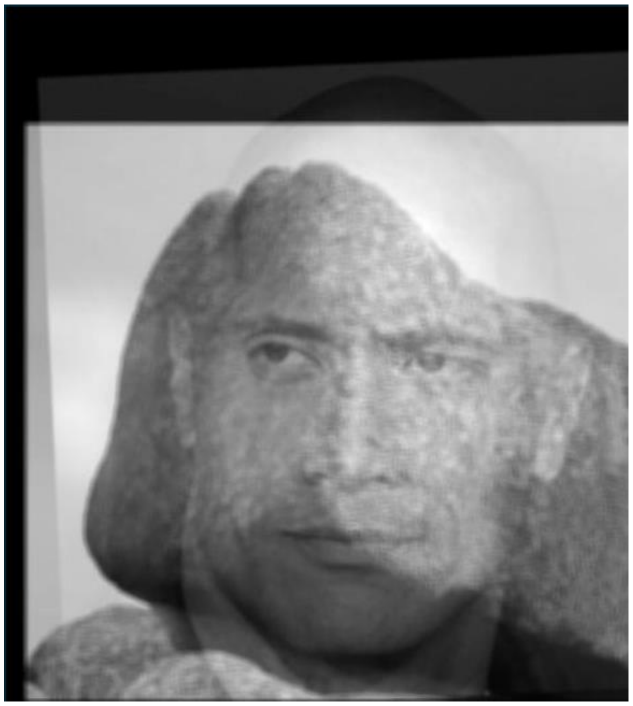
Extra Credit, Both Colored:



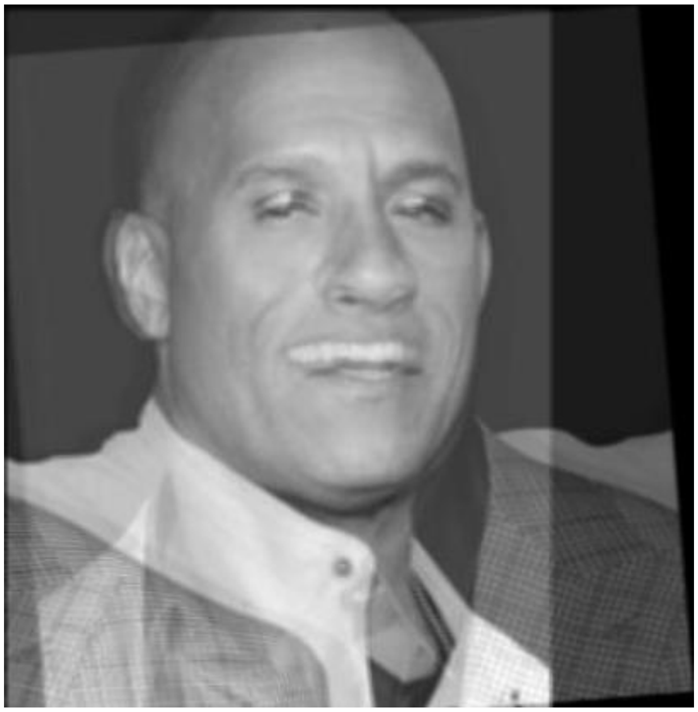
Extra Credit, Low Pass Colored (Monroe):



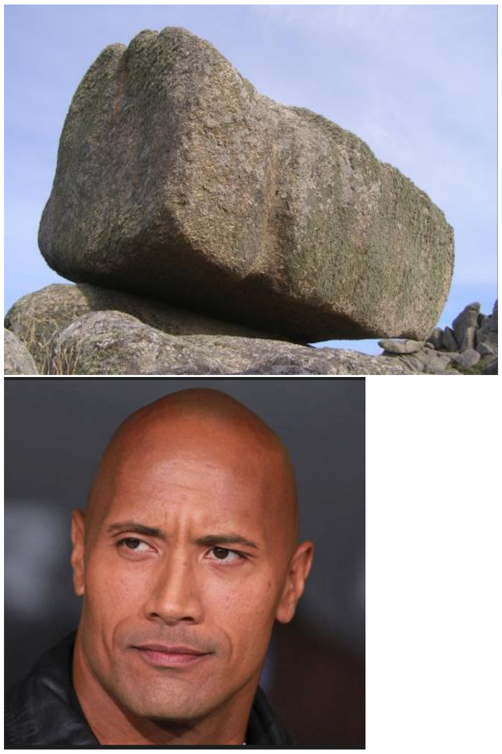
Bad Case (The Rock and an actual Rock):



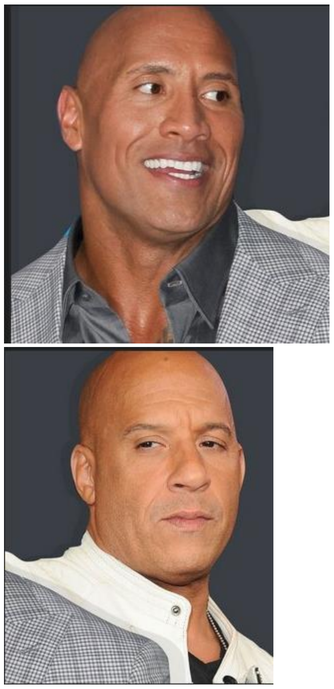
Good case (The Rock and Vin Diesel):



Original Images (The Rock and an actual Rock):



Original Images (The Rock and Vin Diesel):



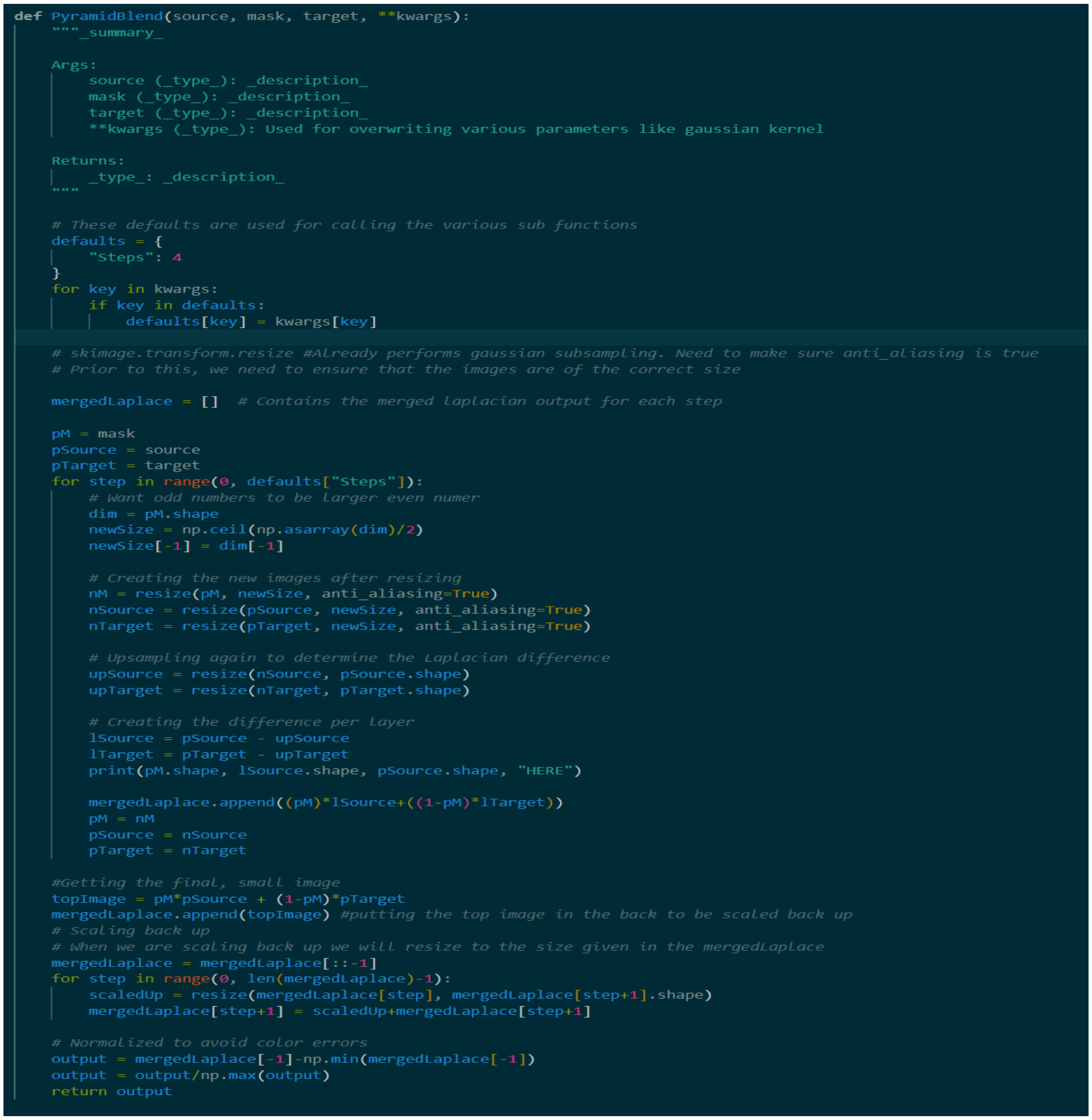
Discussion:

The successful case and the failure case, despite having similar parameters, have different outcomes because the underlying pictures were vastly dissimilar. Despite his name, the Rock does not actually look like a stone, and as such does not possess many similar features. This means there is not much that lines up between the two images, making the hybrid image a failure. However, in the successful case, the Rock is being compared to another buff, bald man, leaving them with many similar features. This made the hybrid image result much better.

Iny my testing with the color image it seemed better to use on the high pass filtered image, though that could be because the Einstein image (the high pass one in my code) possessed more muted colors than the Monroe image. The more muted tones, with similar tones where their skin and hair overlap, led to a clearer image than when Monroe was the colored image, as her red lipstick and other vibrant colors overshadow the Edison image.

Task 2:

This task consisted of creating an image pyramid in order to generate images that blend into each other. This was done by taking the Laplacian pyramid of both images at every step and applying the gaussian pyramid of the mask. By the time the last step is taken, the top image is created and the image is upsampled. These steps are shown in the code screenshot below:

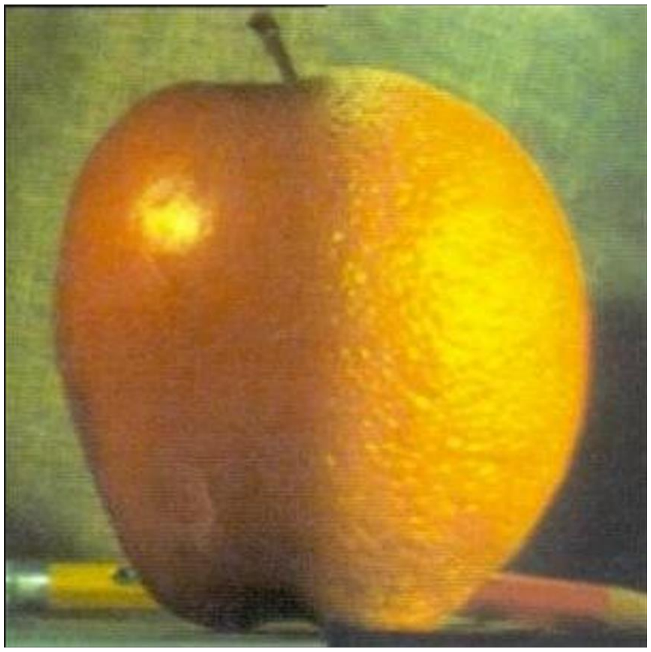


```python
def PyramidBlend(source, mask, target, **kwargs):
    """_summary_

    Args:
        source (_type_): _description_
        mask (_type_): _description_
        target (_type_): _description_
        **kwargs (_type_): Used for overwriting various parameters like gaussian kernel

    Returns:
        _type_: _description_
    """

    # These defaults are used for calling the various sub functions
    defaults = {
        "Steps": 4
    }
    for key in kwargs:
        if key in defaults:
            defaults[key] = kwargs[key]

    # skimage.transform.resize #Already performs gaussian subsampling. Need to make sure anti_aliasing is true
    # Prior to this, we need to ensure that the images are of the correct size

    mergedLaplace = []  # Contains the merged laplacian output for each step

    pM = mask
    pSource = source
    pTarget = target
    for step in range(0, defaults["Steps"]):
        # Want odd numbers to be larger even numer
        dim = pM.shape
        newSize = np.ceil(np.asarray(dim)/2)
        newSize[-1] = dim[-1]

        # Creating the new images after resizing
        nM = resize(pM, newSize, anti_aliasing=True)
        nSource = resize(pSource, newSize, anti_aliasing=True)
        nTarget = resize(pTarget, newSize, anti_aliasing=True)

        # Upsampling again to determine the Laplacian difference
        upSource = resize(nSource, pSource.shape)
        upTarget = resize(nTarget, pTarget.shape)

        # Creating the difference per layer
        lSource = pSource - upSource
        lTarget = pTarget - upTarget
        print(pM.shape, lSource.shape, pSource.shape, "HERE")

        mergedLaplace.append((pM)*lSource+((1-pM)*lTarget))
        pM = nM
        pSource = nSource
        pTarget = nTarget

    #Getting the final, small image
    topImage = pM*pSource + (1-pM)*pTarget
    mergedLaplace.append(topImage) #putting the top image in the back to be scaled back up
    # Scaling back up
    # When we are scaling back up we will resize to the size given in the mergedLaplace
    mergedLaplace = mergedLaplace[::-1]
    for step in range(0, len(mergedLaplace)-1):
        scaledUp = resize(mergedLaplace[step], mergedLaplace[step+1].shape)
        mergedLaplace[step+1] = scaledUp+mergedLaplace[step+1]

    # Normalized to avoid color errors
    output = mergedLaplace[-1]-np.min(mergedLaplace[-1])
    output = output/np.max(output)
    return output
```

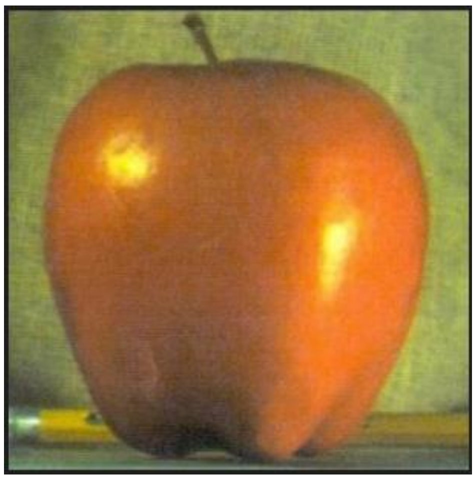
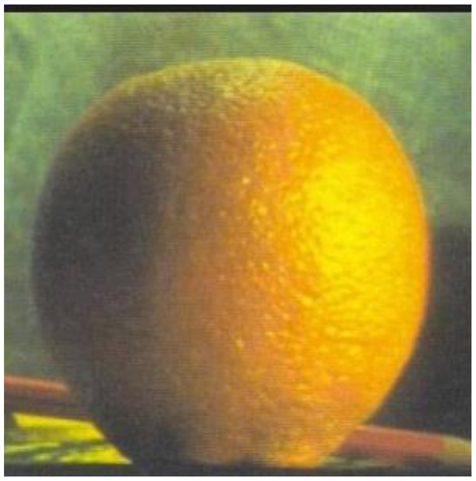
Pyramid Blend Code

Results:

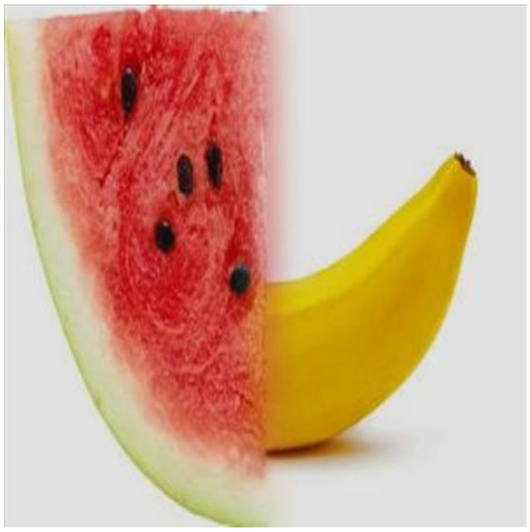
Pyramid_01 (example):



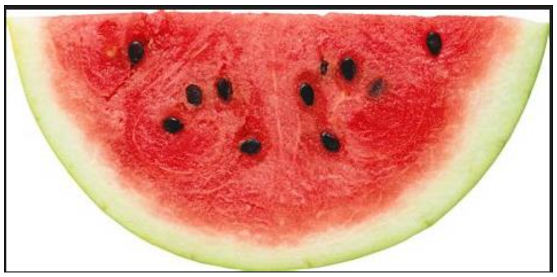
Pyramid_01 original Images:

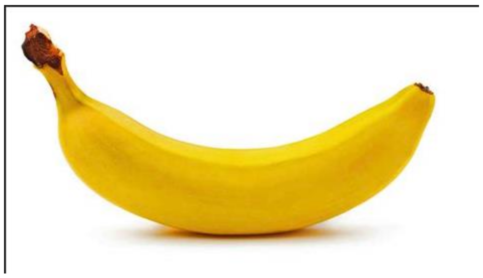



Pyramid_02 (BAD case):

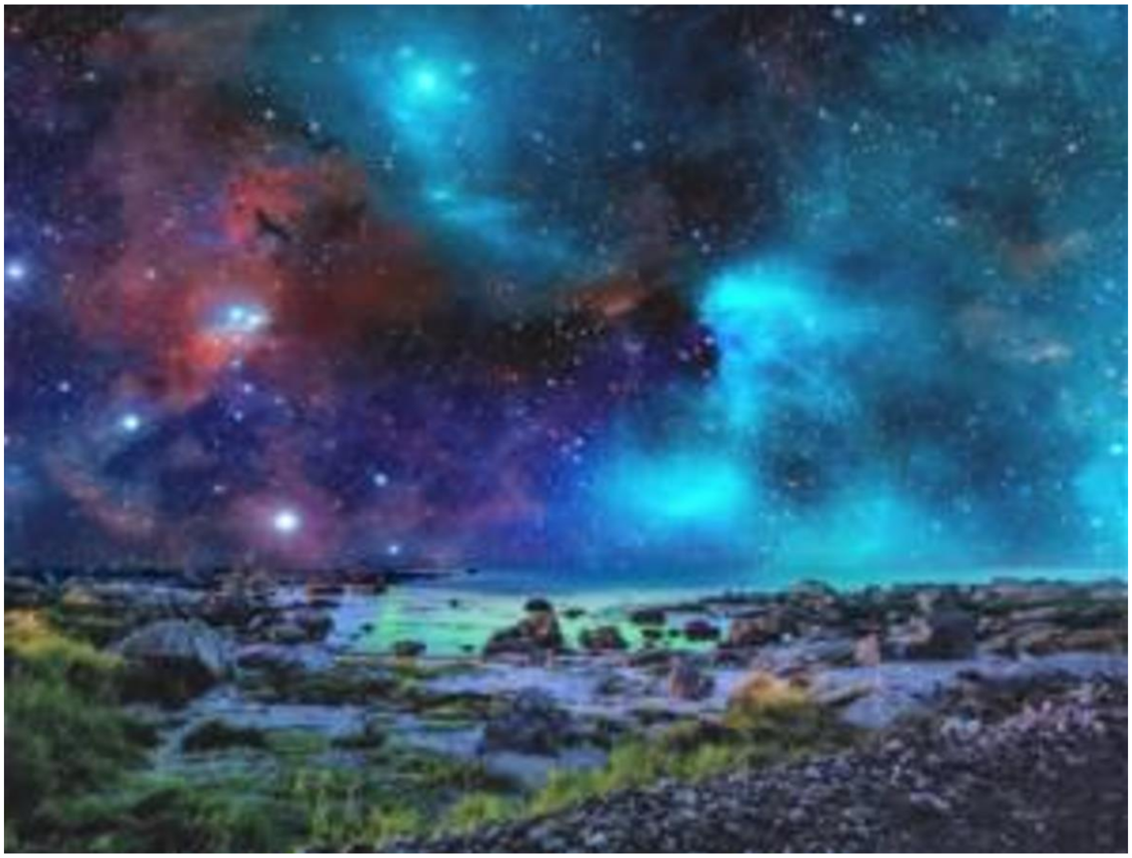


Pyramid_02 Original Images:





Pyramid_03(GOOD case):



Pramid_03 original images:

Discussion:

In the pyramid_01 case and the pyramid_03 case, the images merge well without any incident, whereas in the pyramid_02 case it is evident that those are completely separate images superimposed on top of each other. The reason for this in pyramid_02 is because the objects do not line up at the borders whatsoever. It is shown in the third case that a mask that has been modified to cover the proper area can lead to a good blend through this method.