John Radice
GTID 903095161
Assignment 6

reduce():

First a 5x5 kernel was generated using the provided generatingKernel() function with the parameter a = 0.4. This kernel will be used as a weighting function that will be convolved with the image. The scipy.signal.convolve2d() function was then used to convolve the image with the weighting function. To actually reduce the image numpy's slicing features were used to return an image comprised of every other pixel of the convolved image. This was done by increasing the step between each element as shown in the code below:

```
Args:
  image (numpy.ndarray): a grayscale image of shape (r, c)

Returns:
  output (numpy.ndarray): an image of shape (ceil(r/2), ceil(c/2))
    For instance, if the input is 5x7, the output will be 3x4.

"""
# WRITE YOUR CODE HERE.
kernel = generatingKernel(0.4)

convolved = scipy.signal.convolve2d(image, kernel, 'same')

return convolved[::2, ::2]
# END OF FUNCTION.
```

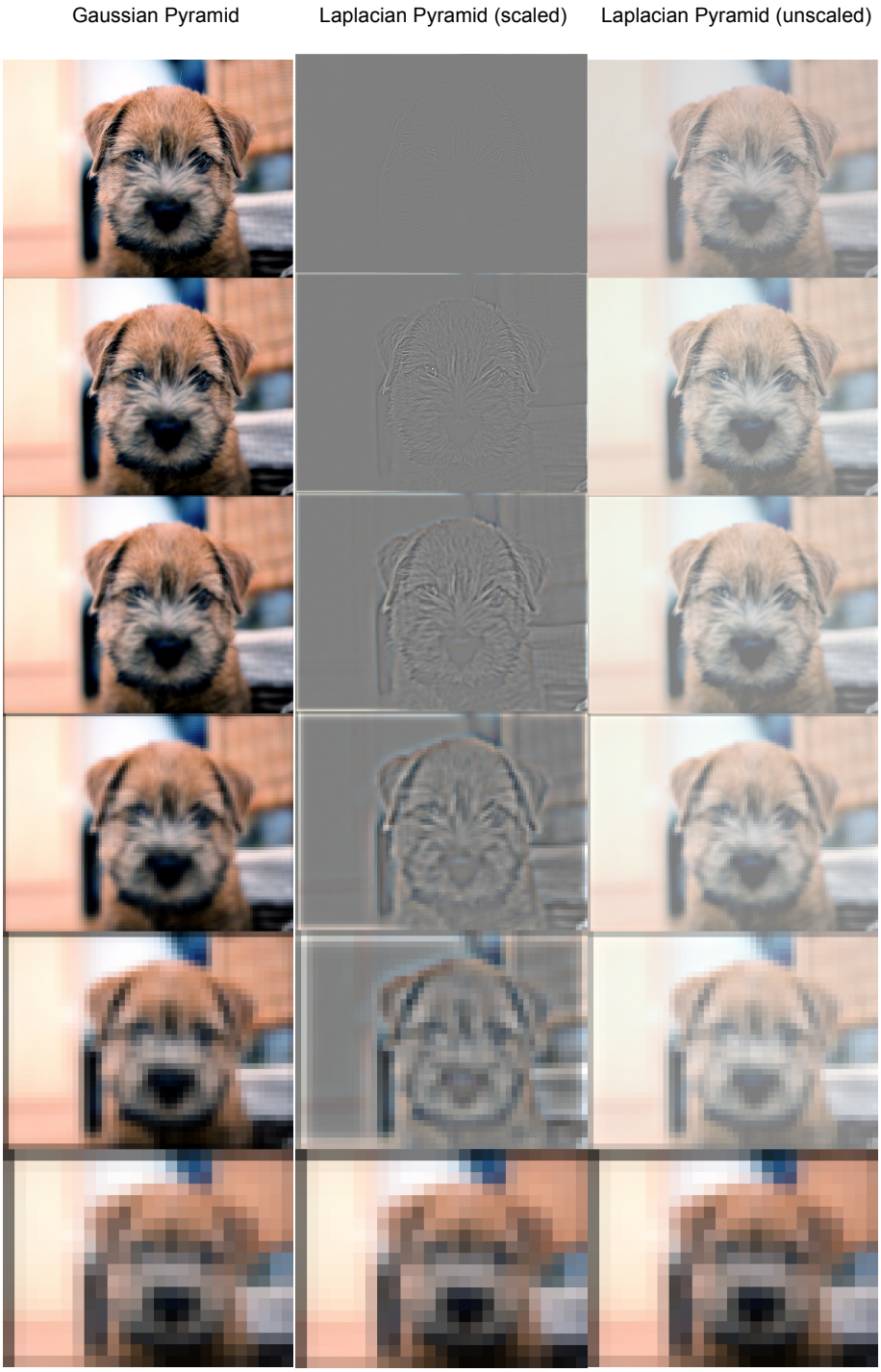
expand():

As in the previous reduce() function a 5x5 kernel was generated with the parameter a = 0.4 to be used as a weighting function. However, in expand() the image size will be increased prior to being convolved with the weighting function as opposed to decreasing the image size after convolution with the weighting function. A black (all zeros) output image was created 2x the size of the input image. Every other element of the output image was then set to the values of the input image using numpy's slicing features. Finally the output image is convolved with the kernel then scaled by a factor of 4 as show in the code below:

```
Args:
  image (numpy.ndarray): a grayscale image of shape (r, c)

Returns:
  output (numpy.ndarray): an image of shape (2*r, 2*c)
"""
# WRITE YOUR CODE HERE.
kernel = generatingKernel 0.4)

expandedSize = tuple np.array image.shape) * 2)

output = np.zeros expandedSize)

output[::2, ::2] = image

return scipy.signal.convolve2d output, kernel, 'same') * 4
# END OF FUNCTION.
```

John Radice
GTID 903095161
Assignment 6

The factor of 4 is used to lighten the image without it the convolved image will appear much darker. This may be an affect of expanding the original image by interpolating with zeros. Below is an example of the Laplacian Pyramids with scaling and without:



John Radice
GTID 903095161
Assignment 6

gaussPyramid():

When the level is 0 a list containing only the original image is returned. For levels greater than 0 the previous reduce() function will be called on the original image and all subsequently generated images for each level. These images are appended to a list starting with the largest (original) image and ending with the smallest and returned as seen below:

```
Args:
  image (numpy.ndarray): A grayscale image of dimension (r,c) and dtype float.
  levels (uint8): A positive integer that specifies the number of reductions
                  you should do. So, if levels = 0, you should return a list
                  containing just the input image. If levels = 1, you should
                  do one reduction. len(output) = levels + 1

Returns:
  output (list): A list of arrays of dtype np.float. The first element of the
                 list (output[0]) is layer 0 of the pyramid (the image
                 itself). output[1] is layer 1 of the pyramid (image reduced
                 once), etc. We have already included the original image in
                 the output array for you. The arrays are of type
                 numpy.ndarray.

Consult the lecture and README for more details about Gaussian Pyramids.
"""
output = [image]
# WRITE YOUR CODE HERE.
if levels == 0:
  return output

for i in range 1, levels + 1):
  output.append reduce output[i - 1]))

return output
# END OF FUNCTION.
```

laplPyramid():

Given a Gaussian Pyramid with the first element containing the base of the pyramid (the largest image) and the last element containing the top of the pyramid (the smallest image) the smaller images will be expanded and subtracted from the larger images. Looping through each next image is expanded, and cropped if necessary using numpy's slicing features, subtracted from the current image, and finally appended to a list of images. The "top" (smallest) image will be appended to the end of the list, as there are no more images to subtract.

```
output = []
# WRITE YOUR CODE HERE.
for k in range(0, len gaussPyr)):
  # Append the "top" of the pyramid/last element to the list
  if k + 1 == len gaussPyr):
    output.append gaussPyr[k])
  else:
    expandedImage = expand gaussPyr[k + 1])
    # Crop expanded image if necessary
    if expandedImage.size > gaussPyr[k].size:
      expandedImage = expandedImage[:gaussPyr[k].shape[0],
                                    :gaussPyr[k].shape[1]]

    output.append gaussPyr[k] - expandedImage)

return output
# END OF FUNCTION.
```

John Radice
GTID 903095161
Assignment 6

blend():

To blend the images each image (layer) was iterated through. For each layer of the pyramids the “white” image was multiplied by the white region of the mask and the “black” image was multiplied by the black region of the mask, by inverting the mask. The scaled images were then blended together by adding the two images. These operations took advantage of numpy’s matrix operators. Each blended layer was then append to a list and returned.

```
blended_pyr = []
# WRITE YOUR CODE HERE.
for i in range 0, len gaussPyrMask)):
  blendedLayer = gaussPyrMask[i]*laplPyrWhite[i] \
               +  1 - gaussPyrMask[i])*laplPyrBlack[i]
  blended_pyr.append blendedLayer)

return blended_pyr
# END OF FUNCTION.
```

collapse():

To collapse each layer back into the complete blended image the blended pyramid will be iterated through. Because the previous gaussPyramid() function produces a list starting with the base (largest) image and ending with the top (smallest) image the iteration will happen in reverse. The smallest image will be expanded, and cropped using numpy’s slicing features if necessary then added to the next largest layer. This process will repeat until all of the smaller layers are expanded and added to the largest (base) image in the pyramid, which is then returned as the output.

```
# WRITE YOUR CODE HERE.
for i in range(len(pyramid) - 1, 0, -1):
  expandedLayer = expand(pyramid[i])
  # crop expanded layer if necessary
  if expandedLayer.size > pyramid[i - 1].size:
    expandedLayer = expandedLayer[:pyramid[i - 1].shape[0],
                                  :pyramid[i - 1].shape[1]]

  pyramid[i - 1] += expandedLayer

return pyramid[0]
# END OF FUNCTION.
```

John Radice
GTID 903095161
Assignment 6

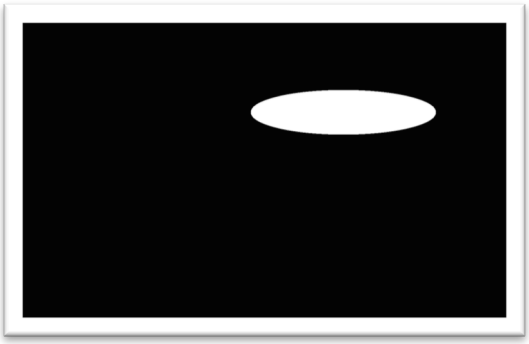
The resulting image was created using the following “black” and “white” images

“black” image

“white” image

I used the following code to approximate the position of the sunset in the “white” image and created the following mask:

```
>>> mask = cv2.Canny(whiteImage, 128, 96)
```



John Radice
GTID 903095161
Assignment 6
Using the previous “black”, “white” and mask images along with the gaussPyramid(), laplPyramid(), blend(), and collapse() functions the following image was created.