Image and Video Processing Lab
Lab 6:Image Pyramids

Aruna Shaju Kollannur (SC21M111)

Sub: Image and Video Processing Lab
Date of Lab sheet: November 16, 2021
Department: Avionics(DSP)
Date of Submission: November 22, 2021

Question 1 a: Generating Gaussian and Laplacian Pyramid by changing Image Size.

Aim

Generating Gaussian and Laplacian Pyramid of an image by downsampling the image at different levels.

Discussion

The elements of a Gaussian Pyramids are smoothed copies of the image at different scales. Generation of Gaussian Pyramid involves two steps:
- Smoothing: This removes the high-frequency components that could cause aliasing.
- Down-sampling: Reduction of image size by downsampling by a factor of 2.

The downsampled image is taken as reference image for next level and the process of smoothing and down-sampling is repeated according to the height of gaussian pyramid to be created.

The Laplacian Pyramid is a stack of the differences between a Gaussian pyramid level with the up-sampled Gaussian pyramid of next level.

If the gaussian pyramid is represented as G$(I) = [I_0, I_1, \ldots, I_K]$
.
and u() be the upsampler,then the coefficients at each level of the Laplacian pyramid L(I) are constructed by taking the difference between adjacent levels in the Gaussian pyramid, upsampling the smaller one with u(.) so that the sizes are compatible:
$h_k = \mathcal{L}_k(I) = G_k(I)\, u(G_{k+1}(I)) = I_k u(I_{k+1})$

Algorithm

```
Step 1: Start
Step 2: Read the image and convert in into gray-scale.
Step 3: Initialise Gaussian filter kernel to be used.
Step 4: Convert Odd size images into even size by neglecting first row/column of image.
Step 5: Declare the height of Image pyramid to be created.
Step 6: Create lists to hold Gaussian and Laplacian pyramids. Initialise first element of Gaussian
        pyramid list as image itself.
Step 7: For each level:
       Step 7.1: Blur the image using gaussian filter and downsample it by a factor of 2.
       Step 7.2: Interpolate the output at Step 7.1 by a factor of two.
       Step 7.3: Obtain difference between last element in Gaussian pyramid list and output of 7.2.
```

Step 7.4: Append result of 7.3 to Laplacian pyramid list.
Step 7.5: Append the blurred downsampled image of Step 7.1 to Gaussian pyramid list.
Step 8: Display the Gaussian and Laplacian pyramids generated.

Program Code

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Image size changed while kernel size kept constant

# 3x3 Gaussian kernel
gaussian_kernel = (1.0/16)*np.array([[1, 2, 1],[2,4,2],[1,2,1]])


# Odd size images converted into even size images to enable
ease in downsampling at every pyramid level
def preprocess_odd_images(img):
    M, N = img.shape[:2]
    if M % 2 == 1 and N % 2 == 1:
        return img[1:][1:] # If both height and width odd, crop 1st row and column ( index 0)
    elif M%2 == 1:
        return img[1:][:]
    elif N%2 == 1:
        return img[:][1:]
    else:
        return img # no change if image height and width is even

def convolve(f, g): # f is image and g is kernel
    # Height and width of image and kernel
    f_height = f.shape[0]
    f_width = f.shape[1]
    g_height = g.shape[0]
    g_width = g.shape[1]

    g_height_mid = g_height // 2
    g_width_mid = g_width // 2

    # convolved image size made to incorporate zero padding size
    xmax = f_height + 2*g_height_mid
    ymax = f_width + 2*g_width_mid

    # Allocate result image.
    h = np.zeros([xmax, ymax], dtype=f.dtype)
    # Do convolution
    for x in range(xmax):
        for y in range(ymax):
            # Calculate pixel value for h at (x,y). Sum one component
            # for each pixel (s, t) of the filter g.
            s_from = max(g_height_mid - x, -g_height_mid)
            s_to = min((xmax - x) - g_height_mid, g_height_mid + 1)

            t_from = max(g_width_mid - y, -g_width_mid)
            t_to = min((ymax - y) - g_width_mid, g_width_mid + 1)

            value = 0
            for s in range(s_from, s_to):
```

```
                for t in range(t_from, t_to):
                    v = x - g_height_mid + s
                    w = y - g_width_mid + t
                    value += g[g_height_mid - s, g_width_mid - t] * f[v, w]
            h[x, y] = value
    return h

def interpolate(image):
    # Upsampling gaussian blur images for Laplacian pyramid

    image_up = np.zeros((2*image.shape[0], 2*image.shape[1]))
    # Inserting zeros at alternate positions forinterpolating by factor 2
    image_up[::2, ::2] = image
    # 4 multiplied with actual kernel as equvalent weight will be reduced due to convolving
    # with upsampled image containing alternate zeroes
    return convolve(image_up, 4*gaussian_kernel)

def gaussian_pyramid(image):
    # Convolution with gaussian blur
    image_blur = convolve(image, gaussian_kernel)
    # image downsampled by factor for next level
    return image_blur[::2, ::2]

def plot_input(img, title):
    plt.imshow(img, cmap = 'gray')
    plt.title(title), plt.xticks([]), plt.yticks([])
    plt.show()

def handle_img_padding(img1, img2):
    # height and width of both images
    M1, N1 = img1.shape[:2]
    M2, N2 = img2.shape[:2]
    padding_x = int(np.abs(M2 - M1)/2)
    padding_y = int(np.abs(N2 - N1)/2)
    img2 = img2[padding_x:M1+padding_x, padding_y: N1+padding_y]
    return img2

def create_gaussian_laplacian_pyramids(image, level):
    G = [image]
    L = []
    while level > 0:
        level -= 1
        image_blur = gaussian_pyramid(image)
        G.append(image_blur)
        expanded_img = interpolate(image_blur)
        if image.shape[:2] != expanded_img.shape[:2]:
            expanded_img = handle_img_padding(image, expanded_img)
        laplacian = image - expanded_img
        L.append(laplacian)
        image = image_blur
    return G, L

original_img = cv2.imread('./Lenna.jpg', 0)
plot_input(original_img, 'Level0')
original_img = preprocess_odd_images(original_img)
M,N = original_img.shape[:2]

pyramid_levels = 6
```

```
# 6 Gaussian Levels, inclusive of original image, lead to 5 Laplacian levels.

gaussian_pyramid, laplacian_pyramid = create_gaussian_laplacian_pyramids(original_img,
                                    pyramid_levels)

for i in range(len(laplacian_pyramid)-1, 0, -1):
        plot_input(laplacian_pyramid[i], 'Laplacian Pyramid - Level '+ str(i))

for i in range(len(gaussian_pyramid)-1, 0, -1):
        plot_input(gaussian_pyramid[i], 'Gaussian Pyramid - Level '+ str(i-1))
```

Result : Lenna Image

Figure 1: Lenna : (a)Original Image

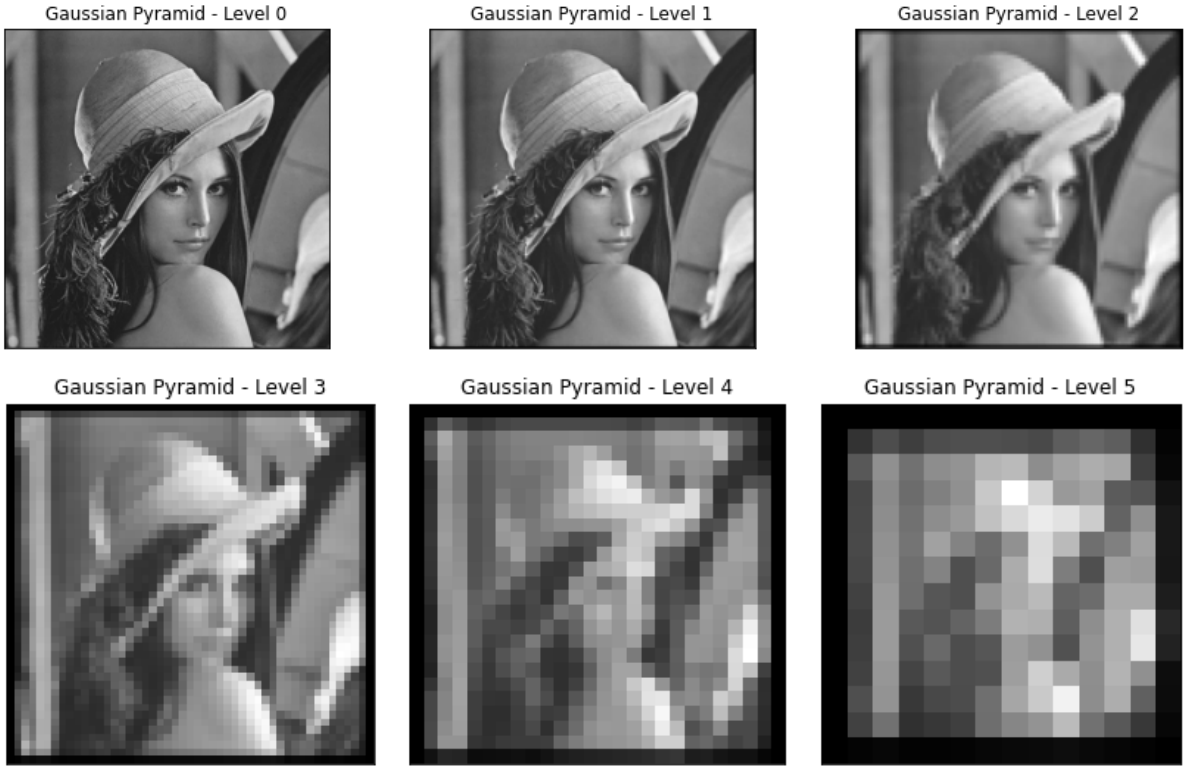


Figure 2: Lenna : (a)Gaussian Pyramid Level 0-5

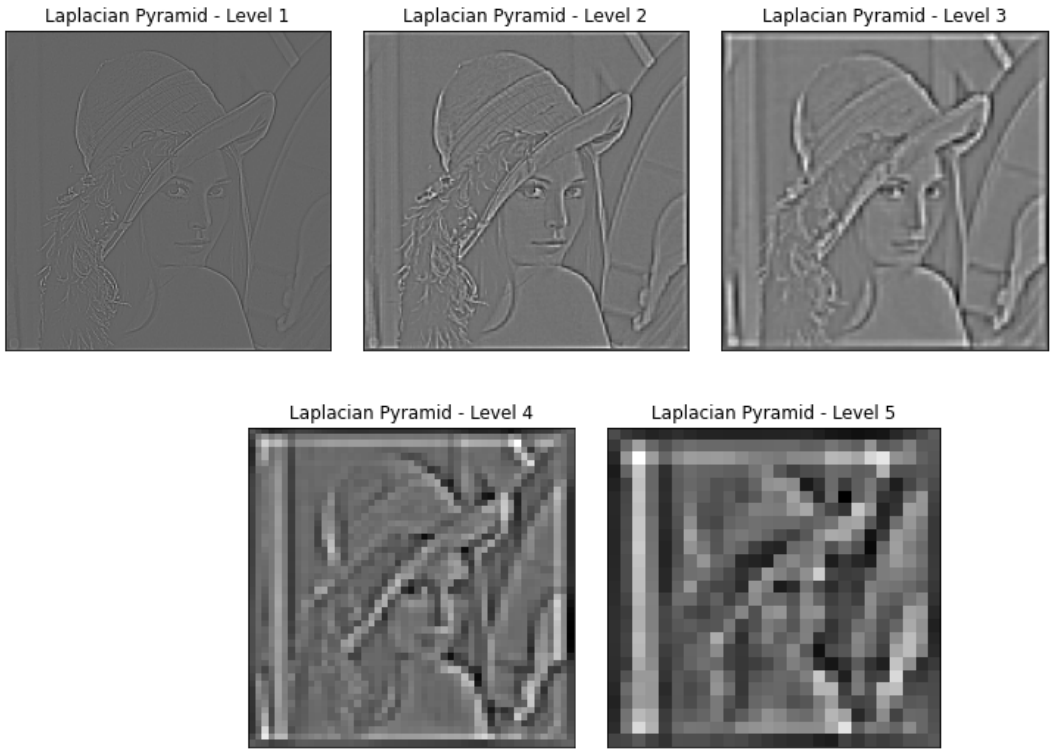


Figure 3: Lenna : (a)Laplacian Pyramid Level 1-5

Results : Moon Image

Figure 4: Moon : (a)Original Image

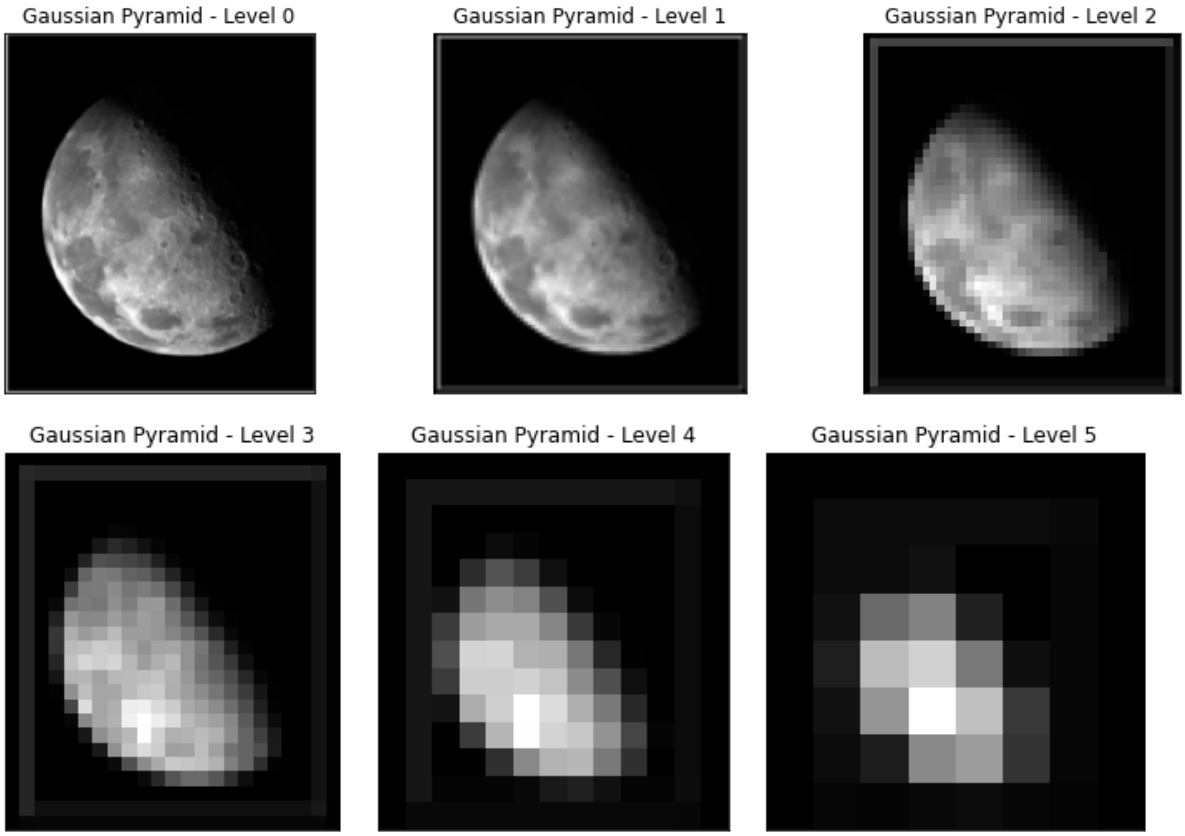


Figure 5: Moon : (a)Gaussian Pyramid Level 0-5

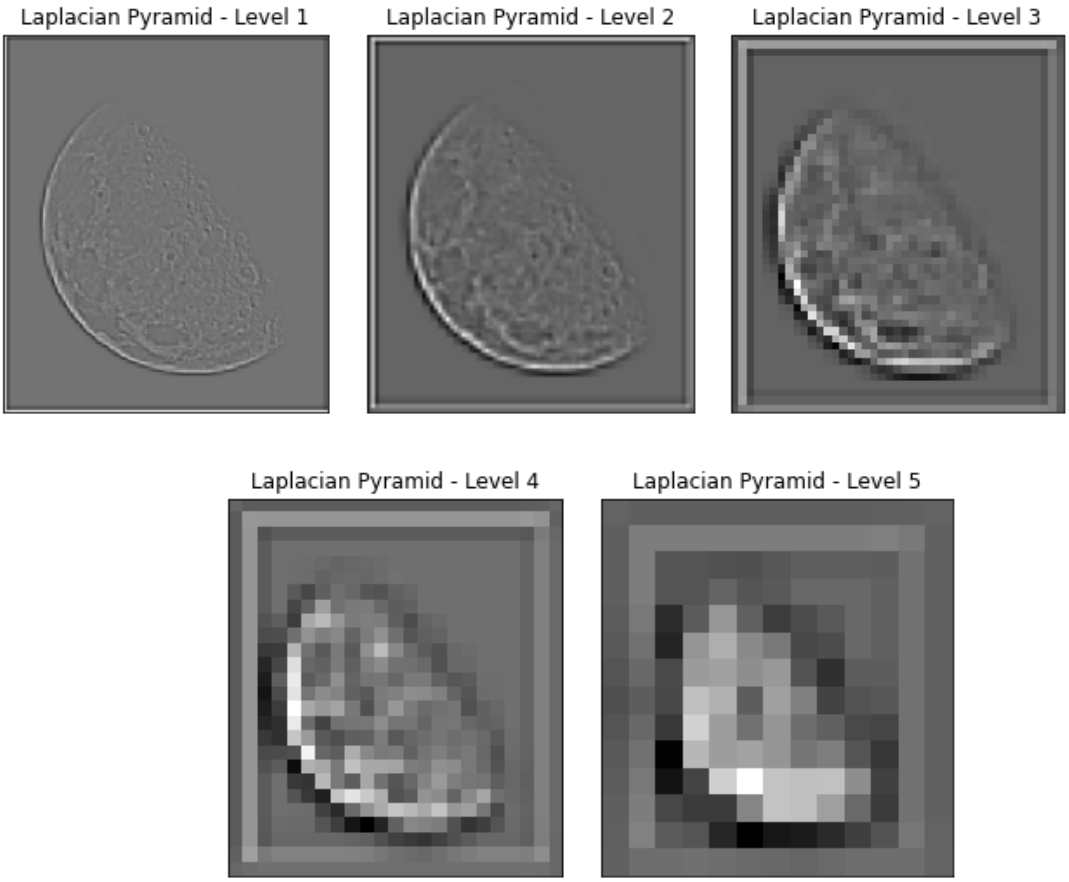


Figure 6: Moon : (a)Laplacian Pyramid Level 1-5

Inference

Gaussian Pyramid Stack is a set of blurred images at different scales.Blurring is done to remove the aliasing effect of scaling down images due to decimation at each level.
Gaussian image pyramid step is an intermediary step to generating Laplacian pyramids.
The applications of Gaussian Pyramids are:

-Efficient Feature Search : By analysing the different levels of a Gaussian pyramid, certain features of the image can be picked out that are invariant throughout all levels,i.e. invariant at multiple scales of the image, which can then be used for various other applications in processing images

- Scale invariant template matching - By looking at different levels of Gaussian pyramid, certain image areas can be observed to exist at all scales which can be used in template matching at different scales. This can be used in applications of image recognition such as Object recognition,face recognition.

Progressive Image Transmission- the top level pyramid can be transmitted rapidly due to its small size, it is then expanded and displayed at the receiver . The difference images (Laplacian pyramid) are also transmitted to improve the image quality successively. This helps the user always have a understanding of the image displayed even when the conditions of channel is bad.

Image Blending - Used for smooth transitioning of two images contours etc. by using different levels of the Gaussian pyramid.

The Laplacian pyramid is used in the efficient representation of an image as the difference of the Gaussian pyramid levels.It is used in image compression applications.

Question 1 b: Reconstruction of image using Laplacian Pyramid and Gaussian Pyramid

Aim

Reconstruction of image using Laplacian Pyramid and Gaussian Pyramid(obtained by scaling-down image size)

Discussion

The Laplacian Pyramid is a stack of the differences between a Gaussian pyramid level with the upsampled Gaussian pyramid of next level.Thus, affectively, the original image can be obtained from the Laplacian pyramid stack and the top-level gaussian pyramid. Reconstruction from a Laplacian pyramid coefficients [h1,h1..hK], k being the number of levels in pixel, is performed using the backward recurrence:

$I_k = u\left(I_{k+1}\right) + h_k$
which is started with IK = hK and the reconstructed image being I = Io. starting at the coarsest level, we repeatedly upsample and add the difference image h at the next finer level until we get back to the full resolution image.

Algorithm

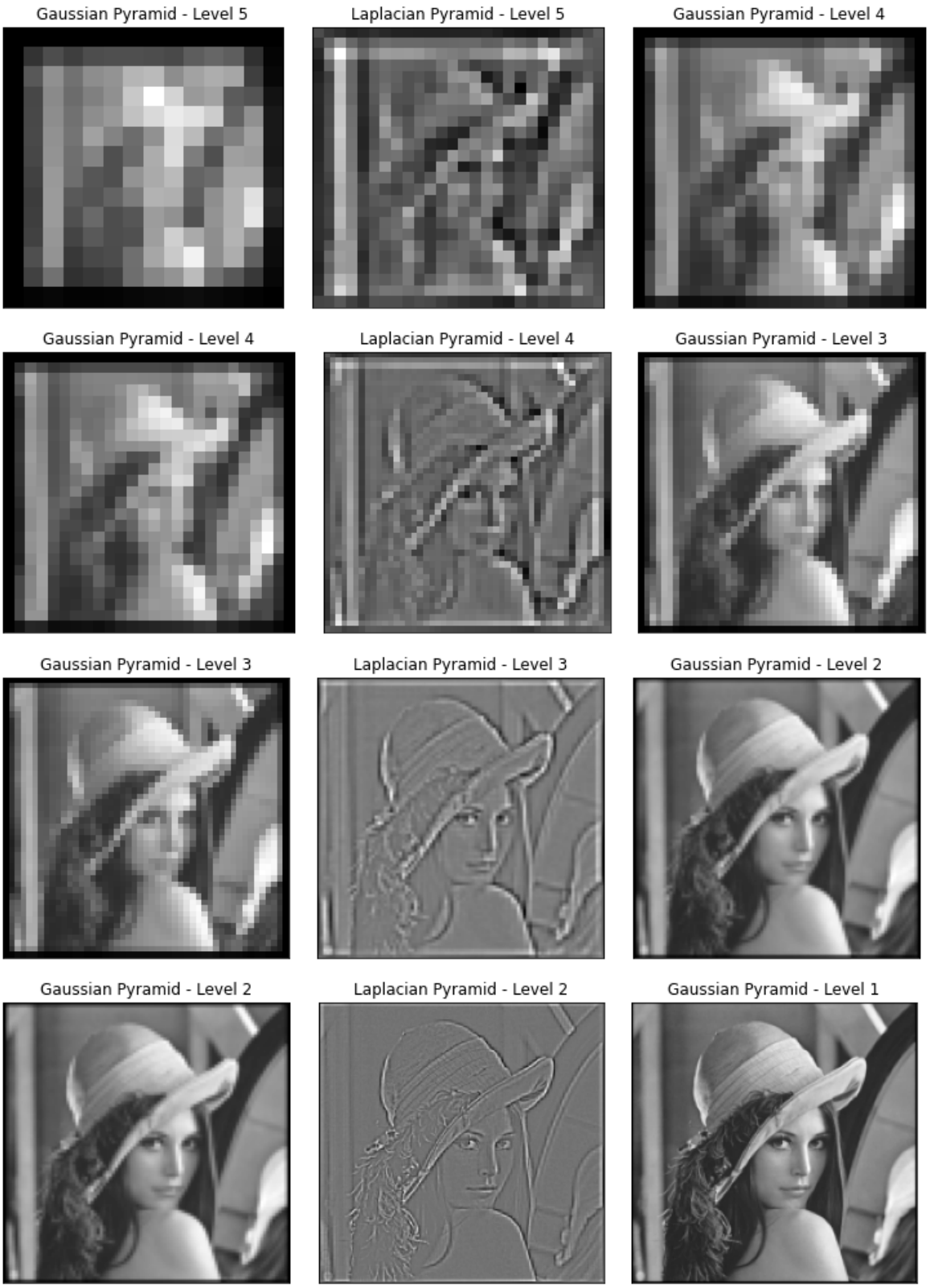
```
Step 1: Start
Step 2: Obtain the Laplacian Pyramid and Gaussian pyramid for an image.
Step 3: Declare an empty list to hold reconstructed images at each gaussian pyramid level.
Step 4: For each level in the Laplacian pyramid,
        Step 4.1: Interpolate the current top-most level of Gaussian pyramid.
        Step 4.2: Add the output of Step 4.1 to current top-most level of Laplacian pyramid.
        Step 4.3: Pop out the last element of Laplacian Pyramid list.
        Step 4.4:Pop out the last element of Gaussian Pyramid list.
Step 5: Display the last output of Step 4.2 as the reconstructed image.
Step 6: Calculate the minimum square error of the reconstructed and original image.
```

Program Code

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from Lab6_Question_1a import interpolate, handle_img_padding,
create_gaussian_laplacian_pyramids, preprocess_odd_images

def plot_input(img, title):
    plt.imshow(img, cmap = 'gray')
    plt.title(title), plt.xticks([]), plt.yticks([])
    plt.show()

def min_sqr_err(mat1, mat2):
    min_sq = 0
    M, N = mat1.shape[:2]
    for i in range(M):
        for j in range(N):
            min_sq += np.square(mat1[i][j] - mat2[i][j])
    return min_sq

def reconstruct_original_img(G, L):
```

```
    reconstructed_images = []
    for i in range(len(G)-1 , 0, -1):
        expanded_img = interpolate(G[i])
        if expanded_img.shape[:2]!= L[i-1].shape[:2]:
            resized_img = handle_img_padding(L[i-1], expanded_img)
        reconstructed_images.append(resized_img + L[i-1])

    return reconstructed_images

original_img = cv2.imread('./Lenna.jpg', 0)
plot_input(original_img, 'Level0')
original_img = preprocess_odd_images(original_img)

pyramid_levels = 6

gaussian_pyramid, laplacian_pyramid = create_gaussian_laplacian_pyramids(original_img,
                                    pyramid_levels)

reconstructed_images = reconstruct_original_img(gaussian_pyramid, laplacian_pyramid)
final_image = reconstructed_images[-1]
plot_input(original_img, 'Original Image')
plot_input(final_image, 'Reconstructed Image')

print ("Minimum Squared Error (MSE) : ", min_sqr_err(original_img, final_image))
```

Result - Lenna



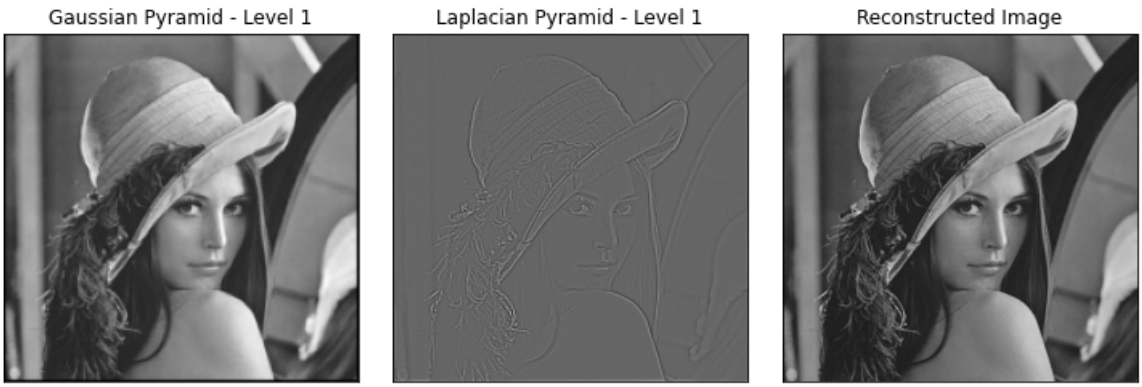
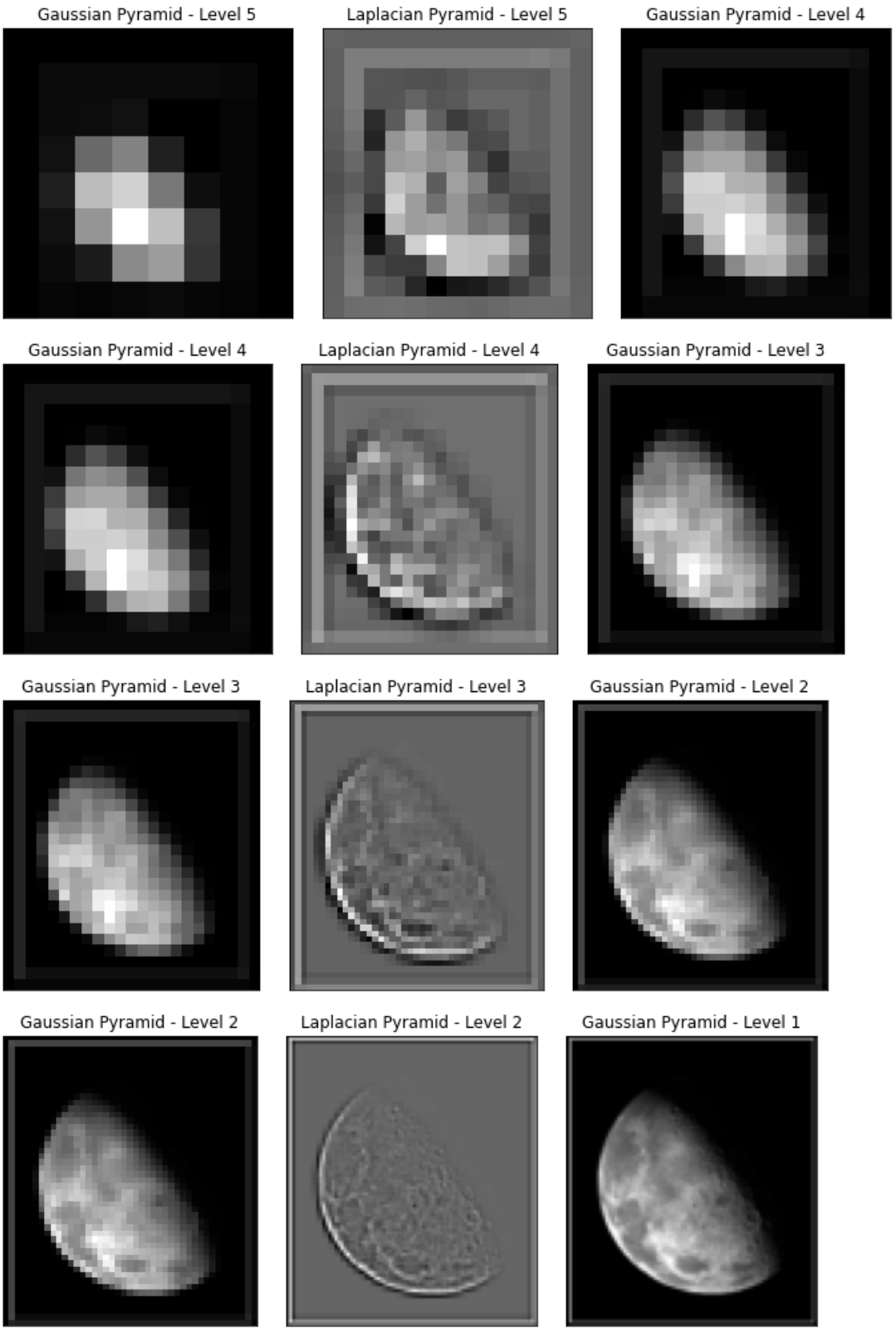


Figure 7: (a) + (b) = (c): (a) Gaussian Pyramid Level (i) (b) Laplacian Level(i) (c) Gaussian Pyramid Level (i-1)

Figure 8: Lenna : (a)Original (b) Reconstructed

Minimum Squared Error (MSE) : 0.0

Result - Moon


Gaussian Pyramid - Level 5
Laplacian Pyramid - Level 5
Gaussian Pyramid - Level 4
Gaussian Pyramid - Level 4
Laplacian Pyramid - Level 4
Gaussian Pyramid - Level 3
Gaussian Pyramid - Level 3
Laplacian Pyramid - Level 3
Gaussian Pyramid - Level 2
Gaussian Pyramid - Level 2
Laplacian Pyramid - Level 2
Gaussian Pyramid - Level 1

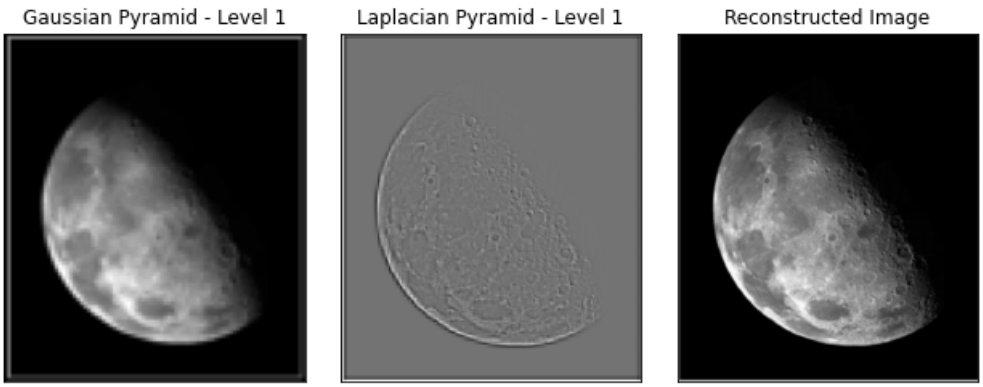


Figure 9: (a) + (b) = (c): (a) Gaussian Pyramid Level (i) (b) Laplacian Level(i) (c) Gaussian Pyramid Level (i-1)

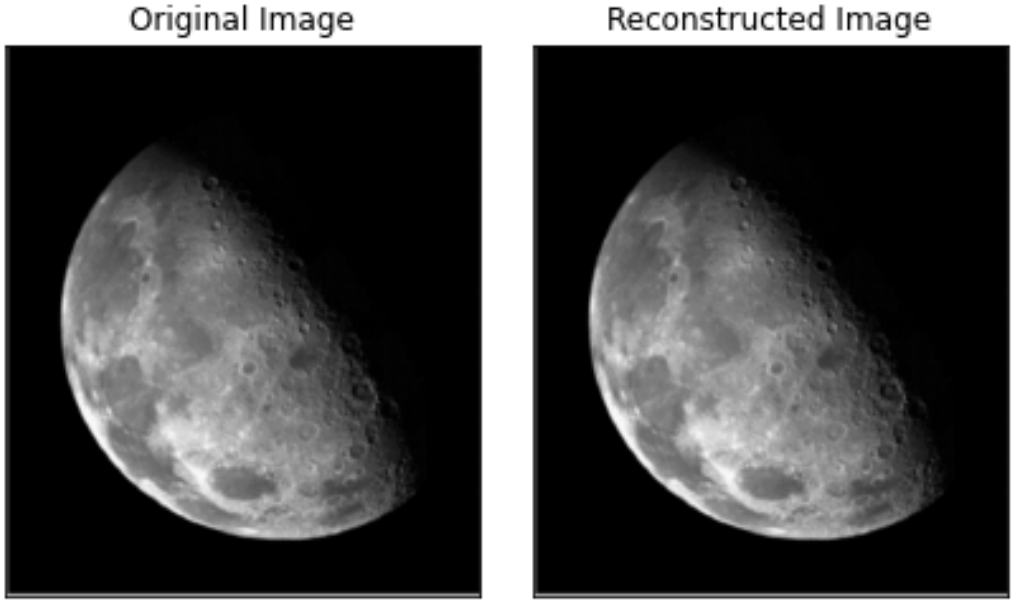


Figure 10: Moon : (a)Original (b) Reconstructed

Minimum Squared Error (MSE) : 0.0

Inference

The original image can be reconstructed using the laplacian pyramid and the top-most level of Gaussian pyramid.

The details in Laplacian pyramid levels reduces as levels increases, i,e, effective difference between adjacent gaussian pyramid levels decreases, thus as laplacian pyramid representation stores the image at different levels with lower memory requirements.

Question 2 : Generating Gaussian and Laplacian Pyramid by changing Kernel Size. Reconstructing original image from Laplacian pyramid and top level image of Gaussian Pyramid

Aim

Generating Gaussian and Laplacian Pyramid by changing Kernel Size and reconstructing original image from Laplacian pyramid and top level image of Gaussian Pyramid

Discussion

The gaussian pyramid is obtained by smoothing and down-sampling image repeatedly to form a gauusian pyramid at different levels.
The blurring effect of Gaussian pyramid increases with its level. This effect obtained by down-scaling the image can replicated by increasing the kernel size(Variance) of filter used for blurring consecutively for each level and keeping the image size as constant.

Algorithm

```
Step 1: Start
Step 2: Read the image and convert in into gray-scale.
Step 3: Declare the height of Image pyramid to be created.
Step 4: Create lists to hold Gaussian and Laplacian pyramids. Initialise first element of Gaussian
        pyramid list as image itself.
Step 5: Run a loop for varying variances to be used for gaussian filter.
        Step 5.1: Obtain the last image of Gaussian pyramid list.
        Step 5.2: Obtain gaussian filtered image of current variance with output of Step 5.1.
        Step 5.3: Append the output of 5.2 to Gaussian pyramid list.
        Step 5.4: Find difference in outputs at Steps 5.1 and 5.2 and append it to Laplacian
        pyramid list.
        Step 5.5: Increment Variance
Step 6: Display Gaussian and Laplacian pyramid
Step 7: Initialise reconstructed image with last element of gaussian pyramid.
Step 8: Run a loop to obtain reconstructed image
        Step 8.1: Cumulatively sum all elements of Laplacian pyramid with initialised
        reconstructed image at Step 7.
Step 9: Obtain minimum square error between reconstructed and original image.
Step 10: Display reconstructed image.
```

Program Code

```
from libc.math cimport exp
from libc.math cimport sqrt

import numpy as np

def gaussian(float[:, :] im, double sigma):
    cdef int height = im.shape[0]  # cdef int tells Cython that
    #this variable should be converted to a C int
    cdef int width = im.shape[1]   #

    # cdef double[:, :, :] to store this as a 3D array of doubles
    cdef double[:, :] img_filtered = np.zeros([height, width])
```

```
    cdef int n = np.int(sqrt(sigma) * 3)

    cdef int p_y, p_x, i, j, q_y, q_x

    cdef double g, gpr
    cdef double w = 0

    for p_y in range(height):
        for p_x in range(width):
            gpr = 0
            w = 0
            for i in range(-n, n):
                for j in range(-n, n):
                    q_y = max([0, min([height - 1, p_y + i])])
                    q_x = max([0, min([width - 1, p_x + j])])
                    g = exp( -((q_x - p_x)**2 + (q_y - p_y)**2) / (2 * sigma**2) )

                    gpr += g * im[q_y, q_x]

                    w += g

            img_filtered[p_y, p_x] = gpr / (w + 1e-5)

    return img_filtered

    # Main File
    import numpy as np
from PIL import Image
import filt
import cv2
import matplotlib.pyplot as plt

def min_sqr_err(mat1, mat2):
    min_sq = 0
    M, N = mat1.shape[:2]
    for i in range(M):
        for j in range(N):
            min_sq += np.square(mat1[i][j] - mat2[i][j])
    return min_sq

img = cv2.imread('Lenna.png', 0)
G = [img]
L = []
outfile = '%s.png' % ("Lenna")

for dev in range(1,6):
 p = np.array(Image.open(outfile), dtype=np.float32)

 img_filtered = np.asarray(filt.gaussian(p, dev))
 Image_gaussian_filtered = Image.fromarray(img_filtered.astype(np.uint8))
 # Gaussian Pyramid
 G.append(Image_gaussian_filtered)
 outfile = '%s%d.jpg' % ("Lenna_Gaussian_Pyramid",dev)
 Image_gaussian_filtered.save(outfile)
 # Laplacian Pyramid
 L.append(np.array(G[dev-1], dtype=np.float32) - np.array(G[dev], dtype=np.float32))

# Reconstruction
```

```
Reconstructed_Image = np.array(G[-1])

for dev in range(5):
    Reconstructed_Image = Reconstructed_Image + L[dev]

Mean_square_error = min_sqr_err(img,Reconstructed_Image )
print(f'Mean square error is {Mean_square_error}')
plt.imshow(Reconstructed_Image, cmap = 'gray')
```

Result

Figure 11: Lenna : (a)Original Image

Figure 12: Lenna : (a)Gaussian Pyramid Level 0-5

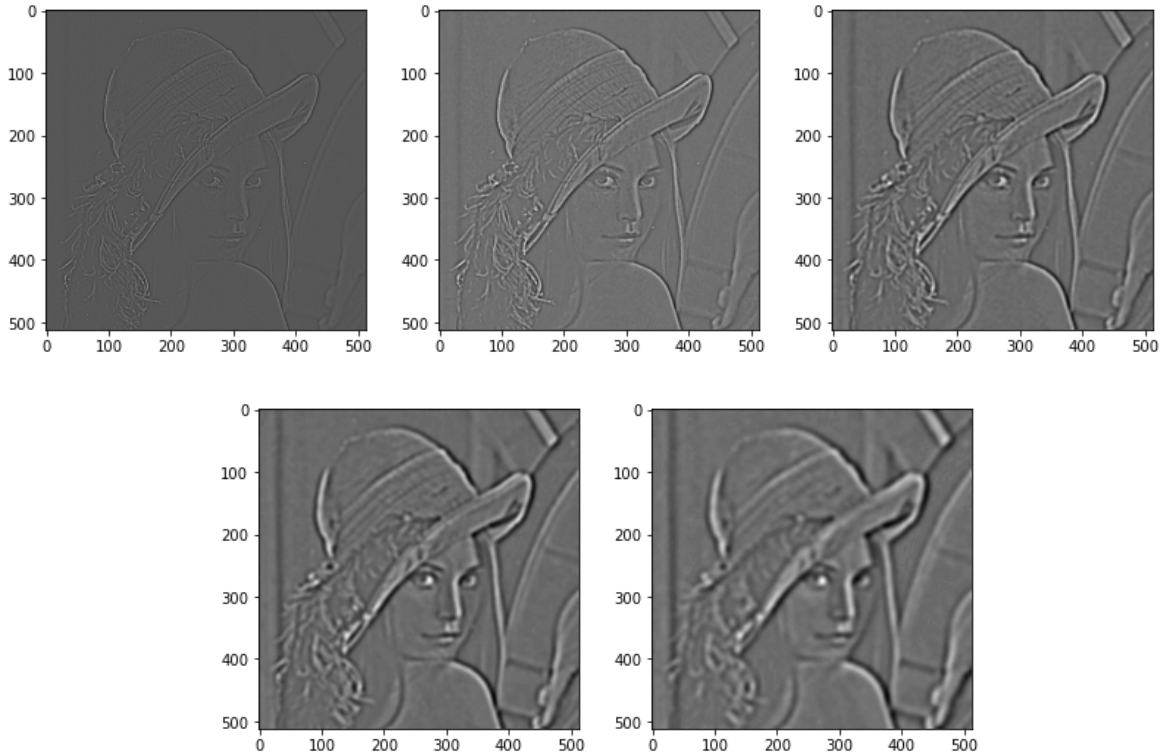


Figure 13: Lenna : (a)Laplacian Pyramid Level 1-5

Result - Reconstruction

Figure 14: Lenna : (a)Original (b) Reconstructed

Minimum Squared Error (MSE) : 0.0

Inference

The Gaussian pyramid can be created with increasing variance of smoothing filter as level of pyramid increases. The reconstructed image can be obtained using the Laplacian Pyramid and top-level Gaussian Pyramid without any errors. This method is not as efficient as scaling down the image as:

- Computation required for blurring of images at constant size and increasing filter size is more.
- Memory occupied by Gaussian and Laplacian Pyramid is more.