

IMAGE PROCESSING

A project on JPEG compression using Python PL



14 SEPTEMBER 2021

BY: ABIY DEMA

Submitted to: Prof. Diana Mateus

Table of Contents

Introduction	1
Histogram function.....	1
• Objective.....	1
• Condition – image intensity	1
Conversion from RGB to YIQ color space and vice versa.....	2
• RGB_YIQ function.....	2
• YIQ_RGB function.....	4
Reducing the size (to half) of I and Q channels	5
Formation of 8 X 8 Subblocks.....	9
Discrete Cosine Transform (DCT II)	11
Quantization.....	12
Dequantization.....	14
Inverse Discrete Cosine Transform (IDCT)	15
Reconstruction of an image.....	17
• RGB Reconstruction.....	18
JPEG Compression	19
JPEG Decompression.....	19
Gaussian filter.....	25
• Low-frequency image.....	27
• High-frequency image.....	28
• Computing their differences.....	30

Introduction

JPEG is a lossy image compression method where it uses the Discrete Cosine Transform (DCT) approach for coding transformation.

Following are some steps of JPEG compression.

Step 1: Convert images from RGB to YIQ color space

Step 2: Reduce the size (to half) of the I and Q channels

Step 3: An image is first subdivided in 8x8 subblocks (assuming the original image has sides whose length is a multiple of 8, if it is not the case just resize it).

Step 4: A frequency transformation is computed for each block (computed with a discrete cosine transform DCT II)

Step 5: The DCT results are then divided by the elements of the quantization matrices given in quantization table. The resulting values are then rounded and stored as integers.

Step 6: The result of quantization (which should have many zeros) is then compressed Run Length Coding and Huffman coding to reduce the size. The compressed values are saved together with the quantization table.

Step 7: The reconstruction of the image implies inverting the encoding, the quantization and the dct.

(Ref: <https://www.javatpoint.com/jpeg-compression>, Prof. Diana Mateus)

Histogram function

```
#https://datacarpentry.org/image-processing/05-creating-histograms/

def plot_histogram(getimage, row, column, order):
    min_pixel, max_pixel = np.amin(getimage), np.amax(getimage)
    hist, bins = np.histogram(getimage, bins=256, range=(min_pixel, max_pixel))
    bins = (bins[:-1] + bins[1:])/2
    ax=plt.subplot(row, column, order)
    plt.plot(bins, hist)
    plt.xlabel("Color values", fontsize=10)
    plt.ylabel("Pixel values", fontsize=10)
    ax.set_xlim([min_pixel, max_pixel])

    # Takes the image and some information for the histogram graph
    # Gets the minimum and maximum pixel of the image
    # For the graph to have a range
    # Equally-spaced intervals (X-axis)
    # Takes the number of rows, columns and place of the image (order: n-th place)
    # X-axis and Y-axis respectively
    # The graph will be shown including the two extremes
```

Objective

- The objective of the above histogram function is to take the image-data in and draws its corresponding graph having taken the minimum and maximum pixel values of the image into consideration.

Condition – image intensity

- Depending on the image intensities; whether it's from 0 to 255 or 0 to 1, the graph will be bounded similarly within that range.

(Ref: <https://datacarpentry.org/image-processing/05-creating-histograms/>)

Conversion from RGB to YIQ color space and vice versa

```
#https://stackoverflow.com/questions/46990838/numpy-transforming-rgb-image-to-yiq-color-space
#https://en.wikipedia.org/wiki/YIQ

def RGB_YIQ (getimage):

    getimage = (getimage*255).astype(float) # Changing it into 0-255 range since the quantization table is based on intensities in between that range.
    YIQ = np.array(
        [[0.299, 0.587, 0.114],
         [0.5959, -0.274, -0.3213],
         [0.2115, -0.5227, 0.3112]])

    YIQ_img = (np.dot(getimage, YIQ.T))
    Y, I, Q = YIQ_img[:,0], YIQ_img[:,1], YIQ_img[:,2] # Y- index 0, I-index 1, Q-index 2

    return YIQ_img

def YIQ_RGB (Y, I, Q):

    (h,w)=Y.shape
    RGB_img = np.zeros((h,w,3))

    # Back to the original image whose intensities was between 0 and 1
    R = (Y + 0.956*I + 0.619*Q)/255
    G = (Y - 0.272*I - 0.647*Q)/255
    B = (Y - 1.106*I + 1.1703*Q)/255

    RGB_img[:,0], RGB_img[:,1], RGB_img[:,2] = R,G,B # R- index 0, G-index 1, B-index 2

    return RGB_img
```

RGB_YIQ function

- The function `RGB_YIQ` uses the input image-data and changes it into 0-255 intensity range because the quantization tables (it'll be discussed later) are based on the intensity within that range. After having the image intensity changed, we then apply the dot product as follows.

- Let's do a dot product between the given 3 x 3 matrix and the transpose of RGB color space. We can convert the RGB color space into YIQ using the following mathematical equations.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} \approx \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.5959 & -0.2746 & -0.3213 \\ 0.2115 & -0.5227 & 0.3112 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

- $Y = 0.299R + 0.587G + 0.114B$
- $I = 0.596R - 0.275G - 0.321B$
- $Q = 0.212R - 0.523G + 0.311B$

However, we should be careful while indicating which one will be multiplied by which, from the function we can observe that the image data comes first and the transpose of the given matrix comes afterwards.

By default, when printing the image data, it will be printed as “[R, G, B]” having n x 3 dimension. To get exactly like the above matrix, we just need to transpose the given 3x3 matrix. Illustrations are given below.

$$[R, G, B] * \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.5959 & -0.2746 & -0.3213 \\ 0.2115 & -0.5227 & 0.3112 \end{bmatrix} \neq [Y, I, Q]$$

It should be;

$$[R, G, B] * \begin{bmatrix} 0.299 & 0.5959 & 0.2115 \\ 0.587 & 0.2746 & 0.5227 \\ 0.114 & 0.3213 & 0.3112 \end{bmatrix} = [Y, I, Q]$$

NB:

- $[Y, I, Q]$ has $n \times 3$ dimension; **not** 1×3 .

- As an example, let's see the value of **Y-channel** which can be calculated from the following equation.

- $Y = 0.299R + 0.587G + 0.114B$

NB:

- R: index – 0, G: index – 1, B: index – 2

R-channel * 0.299

```
[[ 2.60065839 3.28054461 2.88462977 ... 32.78695644 32.90667009
 31.02443328]
 [ 3.31724021 2.73658111 7.45692558 ... 31.98930009 32.63083292
 30.70046569]
 [ 2.80090547 4.80967721 8.13222048 ... 33.85683238 33.32634452
 31.3024925]
 ...
 [71.04154383 75.25488786 75.51073409 ... 49.41307579 49.82173808
 47.75869299]
 [72.06712974 75.88983208 75.86654489 ... 49.20258815 49.5396681
 47.92409865]
 [67.58069537 70.70982053 70.70977576 ... 48.8164022 48.54526453
 47.38937786]]
```

G-channel * 0.587

```
[[ 4.88036005 6.14563555 7.29763568 ... 120.56973621 120.60748395
 115.17857405]
 [ 6.30992011 4.41763857 11.30838891 ... 130.9011196 131.87991068
 125.83452243]
 [ 3.95679547 9.1834631 8.604341 ... 130.40628179 129.32443631
 122.84891368]
 ...
 [129.97447838 134.53560652 134.26933902 ... 149.30388305 148.0751655
 140.60614053]
 [130.41174353 135.24583212 134.73127959 ... 149.45818252 148.67768307
 141.18623082]
 [123.99221834 129.73167632 129.7312853 ... 135.70497352 136.28338804
 129.4078545]]
```

B-channel * 0.114

```
[[ 1.51911406 1.48071797 0.95470856 ... 26.62868127 26.81641338
 24.99410175]
 [ 1.82427899 1.01428061 1.68526556 ... 28.4720158 28.8344073
 26.91572515]
 [ 1.56399763 1.62794501 0.84266226 ... 28.84889777 28.76427905
 26.81722803]
 ...
 [23.56700872 25.00611472 25.39992968 ... 28.81340956 28.70534451
 27.3341767]
 [24.07961281 25.50078277 25.9716877 ... 28.8894059 28.83697142
 27.49866666]
 [23.92733524 25.03531975 25.03567966 ... 26.35783309 26.39832474
 25.24690022]]
```

Y - channel

```
[[ 9.0001325 10.90689813 11.136974 ... 179.98537392 180.33056743
 171.19710909]
 [ 11.4514393 8.16850028 20.45058005 ... 191.36243548 193.34515091
 183.45071327]
 [ 8.32169857 15.62108532 17.57922374 ... 193.11201194 191.41505988
 180.9686342]
 ...
 [224.58303093 234.79660911 235.1800028 ... 227.53036841 226.60224809
 215.69901022]
 [226.55848608 236.63644697 236.56951217 ... 227.55017656 227.05432259
 216.60899613]
 [215.50024896 225.4768166 225.47674072 ... 210.87920881 211.22697731
 202.04413258]]
```

- If we add up values of [R, G, B] channels, they will give the values of the Y-channel, therefore, Y channel has been formed.

For instance: let's take the first top-left corner channel values that each arrow points to.

- $Y = 0.299R + 0.587G + 0.114B$
- $Y = 2.60065839 + 4.88036005 + 1.51911406 = 9.0001325$

Likewise, for the I and Q channels, the dot product works the same way and finally the function returns the YIQ image.

YIQ_RGB function

- This function `YIQ_RGB` also takes the input image-data and changes it into 0 to 1 image intensity range; here the input is the YIQ image whose intensity ranges from 0 to 255 so dividing it by 255 sets our range back to in between 0 and 1. Moreover, the same process applies for the YIQ to RGB conversion. Finally, it returns the RGB image.

```
plt.rcParams["figure.figsize"] = [20,20]
plt.rc('xtick', labelsiz=10)
plt.rc('ytick', labelsiz=10)

original=im
plt.subplot(6,4,1), plt.imshow(original), plt.title("RGB ", fontsize = 10), plot_histogram(original, 6,4,5) # Original RGB image
plt.subplot(6,4,2), plt.imshow(original[:,0]), plt.title("R CHANNEL", fontsize = 10), plot_histogram(original[:,0], 6,4,6) # R-channel
plt.subplot(6,4,3), plt.imshow(original[:,1]), plt.title("G CHANNEL", fontsize = 10), plot_histogram(original[:,1], 6,4,7) # G-channel
plt.subplot(6,4,4), plt.imshow(original[:,2]), plt.title("B CHANNEL", fontsize = 10), plot_histogram(original[:,2], 6,4,8) # B-channel

YIQ_image=RGB_YIQ (original) #Conversion from RGB ->YIQ
Y_channel = YIQ_image[:,0]
I_channel = YIQ_image[:,1]
Q_channel = YIQ_image[:,2]

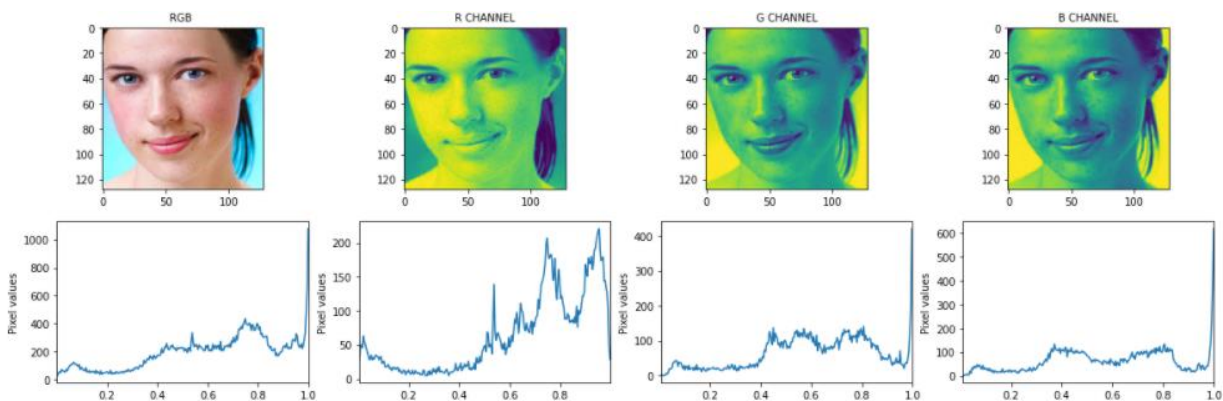
plt.subplot(6,4,9), plt.imshow(YIQ_image), plt.title("YIQ IMAGE", fontsize = 10), plot_histogram(YIQ_image, 6,4,13) # Converted YIQ image
plt.subplot(6,4,10), plt.imshow(Y_channel), plt.title("Y channel", fontsize = 10), plot_histogram(Y_channel, 6,4,14) # Y-channel
plt.subplot(6,4,11), plt.imshow(I_channel), plt.title("I channel", fontsize = 10), plot_histogram(I_channel, 6,4,15) # I-channel
plt.subplot(6,4,12), plt.imshow(Q_channel), plt.title("Q channel", fontsize = 10), plot_histogram(Q_channel, 6,4,16) # Q-channel

RGB_image=YIQ_RGB (Y_channel, I_channel, Q_channel) #Conversion from YIQ ->RGB
R_channel = RGB_image[:,0]
G_channel = RGB_image[:,1]
B_channel = RGB_image[:,2]

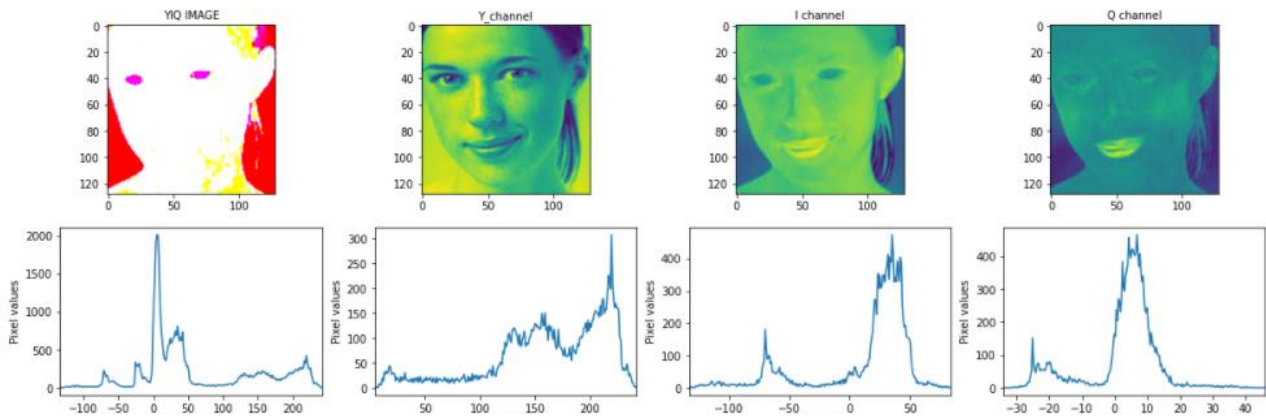
plt.subplot(6,4,17), plt.imshow(RGB_image), plt.title("RGB IMAGE", fontsize = 10), plot_histogram(RGB_image, 6,4,21) # Converted RGB image
plt.subplot(6,4,18), plt.imshow(R_channel), plt.title("R CHANNEL", fontsize = 10), plot_histogram(R_channel, 6,4,22) # R-channel of the image
plt.subplot(6,4,19), plt.imshow(G_channel), plt.title("G CHANNEL", fontsize = 10), plot_histogram(G_channel, 6,4,23) # G-channel of the image
plt.subplot(6,4,20), plt.imshow(B_channel), plt.title("B CHANNEL", fontsize = 10), plot_histogram(B_channel, 6,4,24) # B-channel of the image
plt.show()
```

Here are some images describing those two-color spaces and each channel;

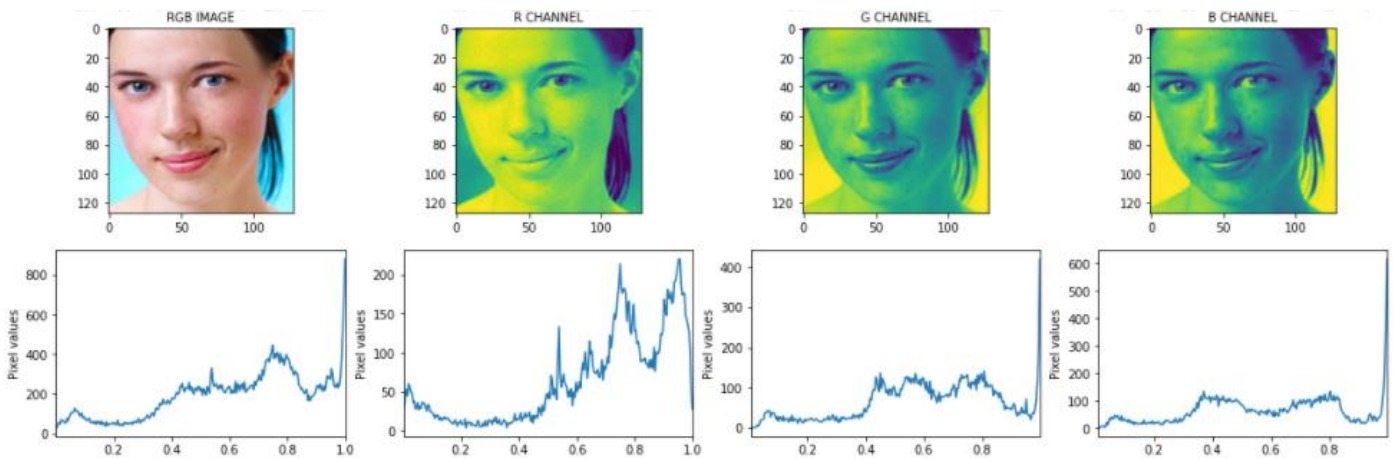
❖ RGB image and its channels



❖ Conversion from RGB to YIQ image and YIQ channels



❖ Conversion from YIQ back to RGB image and RGB channels



(Ref: <https://stackoverflow.com/questions/46990838/numpy-transforming-rgb-image-to-yiq-color-space>, <https://en.wikipedia.org/wiki/YIQ>)

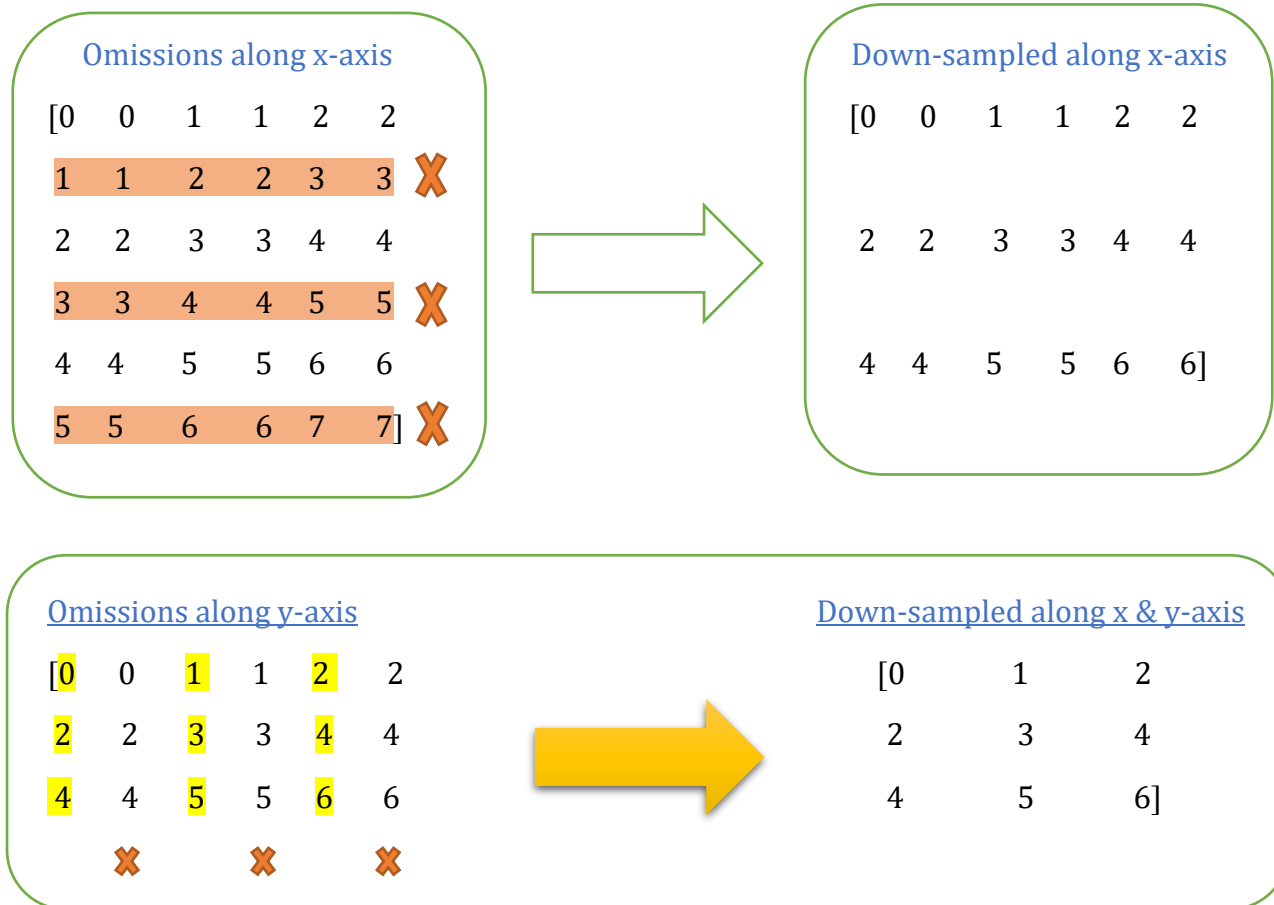
Reducing the size (to half) of I and Q channels

Down-sampling: Here is the concept, it's picking out pixel values according to the sampling frequency indicated. Moreover, this process definitely affects the size of the given channel. I'll explain it using matrix diagram in order to see the idea very well.

Example:

0	0	1	1	2	2
1	1	2	2	3	3
2	2	3	3	4	4
3	3	4	4	5	5
4	4	5	5	6	6
5	5	6	6	7	7

As we can see the given matrix is 6×6 and let's consider the sampling frequency along x and y-axis is 2. Just from the sampling frequency we can understand that the given matrix will finally boil down to a dimension of 3×3 . Here's how;



- From the above matrix representation, the down-sampled matrix along x and y-axis ended up having a dimension of 3×3 . Using the same concept, we apply the process on I and Q channels.

```
def downsample(inputdata, sampling_freq):
    downsampled_img = inputdata[::sampling_freq, ::sampling_freq]
    return downsampled_img
```

- Here in this case our input-data is I and Q channels; then we are supposed to reduce their size by half which indirectly refers to the sampling frequency being equal to 2. After that it'll apply the process along x and y-axis and finally returns the down-sampled image.

Up-sampling: It is the process of repeating the elements of the matrix along the columns and rows; subsequently, it enlarges the size of the input-data.


```
def upsample(inputdata,sampling_freq):
    upsampled_img = inputdata.repeat(sampling_freq, axis=0).repeat(sampling_freq, axis=1)
    return upsampled_img
```

Assume we have 3 x 3 matrix and if the sampling frequency is 2 then the dimension of the matrix will be 6 x 6. Let's take the following matrix as an example;

0	1	2
2	3	4
4	5	6

Up-sampling along the columns.

0	1	2
2	3	4
4	5	6



0	0	1	1	2	2
2	2	3	3	4	4
4	4	5	5	6	6

Up-sampling along the rows.

0	0	1	1	2	2
---	---	---	---	---	---

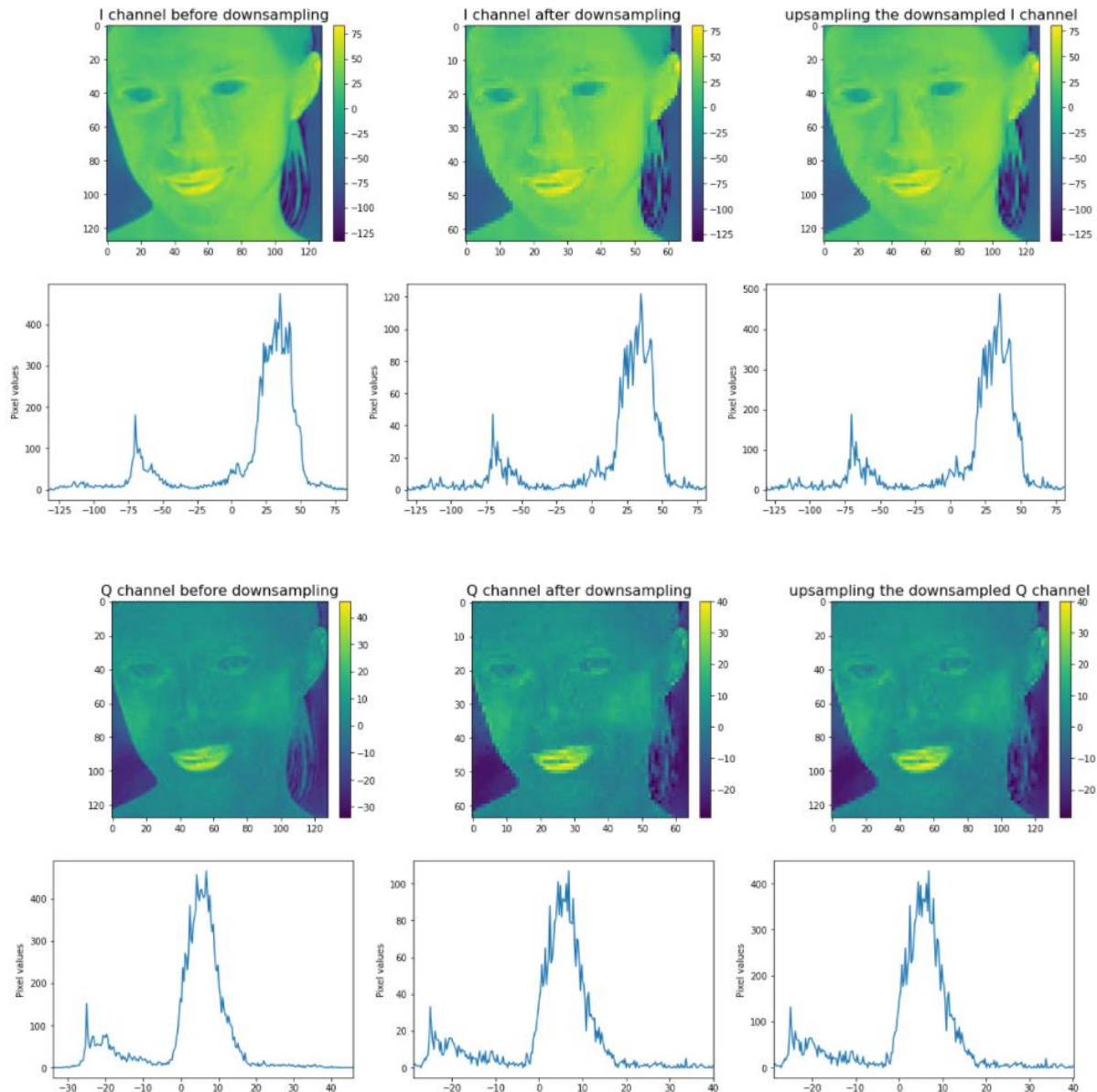
2	2	3	3	4	4
---	---	---	---	---	---

4	4	5	5	6	6
---	---	---	---	---	---



0	0	1	1	2	2
0	0	1	1	2	2
2	2	3	3	4	4
2	2	3	3	4	4
4	4	5	5	6	6
4	4	5	5	6	6

Now we have up-sampled the 3×3 matrix successfully to a dimension of 6×6 . The same process applies for each channel; I and Q. Although it's very hard to tell the difference by just looking at the two images, we can use the histogram-graph to see the subtle effects and there you go, the use of histogram-graph comes into play.



NB:

- The color-bars next to the images of I and Q channels indicate the color scale values and the fraction and pad values set the bar to have equal height as its image.
- Based on the histogram graph, compared to the original I channel, the pixel values of the down-sampled I channel has changed almost by a factor of $400/120 \approx 3.33$ and also noting the changes on the Q channel, it's clear that the Q channel has diminished its values by a factor of $400/100 = 4$.

(Ref: <https://stackoverflow.com/questions/34231244/downsampling-a-2d-numpy-array-in-python/34232507>,
<https://stackoverflow.com/questions/32846846/quick-way-to-upsample-numpy-array-by-nearest-neighbor-tiling>,
https://matplotlib.org/stable/api/as_gen/matplotlib.pyplot.colorbar.html)

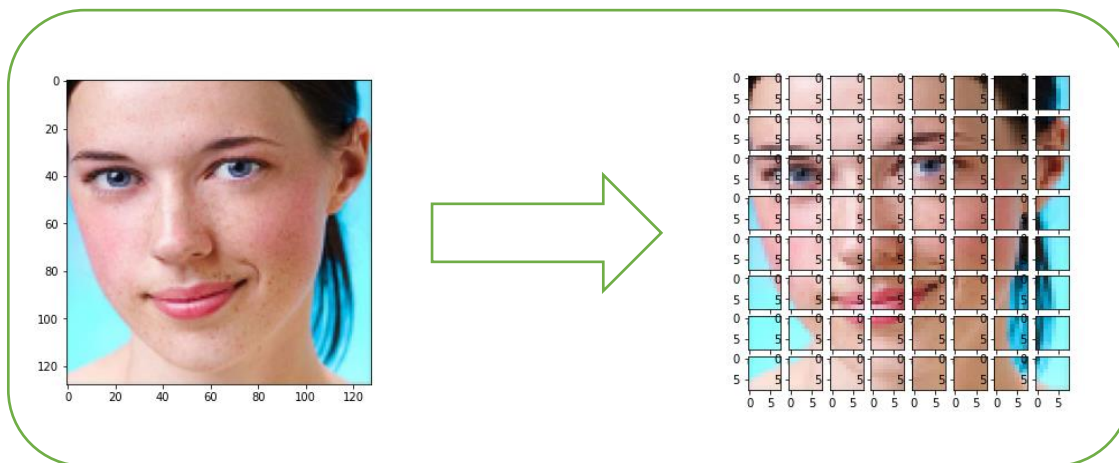
Formation of 8 X 8 Subblocks

```
def subdivision(getimage):
    (x,y) = getimage.shape[0], getimage.shape[1] # A function that takes the data in
    store_blocks = [] # Gets the size of the rows and columns which will make it its height and width
    # Stores the 8 x 8 blocks
    for i in range(0,x-8): # Step along x = 8
        for j in range(0,y-8): # Step along y = 8
            store_blocks.append(getimage[i:i+8,j:j+8]) # 8 x 8 blocks formation
    return store_blocks # Returns the blocks that's been stored
```

- To divide an image or any input-data, we first need to know their sizes along the x and y dimension. From the above code, since “j” is within the loop of “i”, that means for a fixed value of “i” we have j running up to the value of “y” having a step of 8.

`store_blocks.append(getimage[i:i+8,j:j+8])`: for (0:8) value along x-direction; we have (0:8, 8:16, 16:24, 24:32, 32:40, 40:48, 48:56, 56:64) values along y-direction. for one block of the data along x -axis we'll have 8 blocks (counting from 1, including the one on the x-axis) along y-axis.

Look at the following image for clarification.



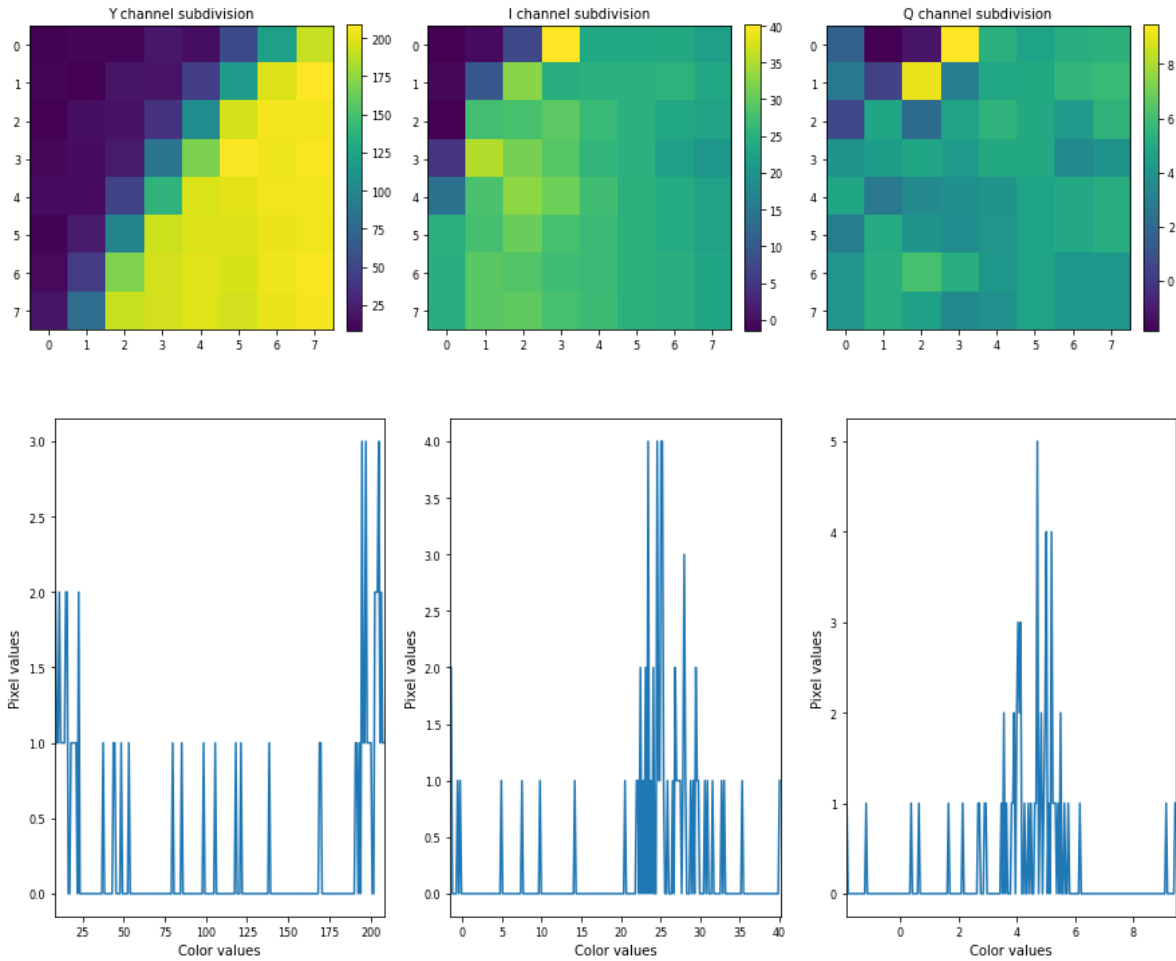
After step 2 which was to reduce the size of I and Q channels by half, we will test each channel including the “Y” through the `subdivision` function. Here we have the code, some 8 x 8 block-output images and pixel values respectively.

```
plt.rcParams["figure.figsize"] = [15,15]
plt.rc('xtick', labelsizes=8)
plt.rc('ytick', labelsizes=8)

Y_subdivision = subdivision(Y_channel) # Calling the Y-channel to make 8 x 8 blocks out of it
I_subdivision = subdivision(I_down) # Calling the downsampled I-channel to make 8 x 8 blocks out of it
Q_subdivision = subdivision(Q_down) # Calling the downsampled Q-channel to make 8 x 8 blocks out of it

plt.subplot(2,3,1).imshow(Y_subdivision[0]).plt.title("Y channel subdivision", fontsize = 10).plt.colorbar(fraction=0.046, pad=0.04).plot_histogram(Y_subdivision[0])
plt.subplot(2,3,2).imshow(I_subdivision[0]).plt.title("I channel subdivision", fontsize = 10).plt.colorbar(fraction=0.046, pad=0.04).plot_histogram(I_subdivision[0])
plt.subplot(2,3,3).imshow(Q_subdivision[0]).plt.title("Q channel subdivision", fontsize = 10).plt.colorbar(fraction=0.046, pad=0.04).plot_histogram(Q_subdivision[0])
plt.show()

#print("Y_subdivision[0] ***\n\n", Y_subdivision[0])
#print("I_subdivision[0] ***\n\n", I_subdivision[0])
#print("Q_subdivision[0] ***\n\n", Q_subdivision[0])
```



```
*** Y_subdivision[0] ***
```

```
[[ 9.0001325 10.90689813 11.136974 20.7211184 15.32368421
 52.876824 121.30786933 190.9872012]
 [11.4514393 8.16850028 20.45058005 18.76634618 44.83721043
 117.96274729 198.01579324 208.54776474]
 [ 8.32169857 15.62108532 17.57922374 37.27562521 105.31937004
 195.80489023 204.92063247 204.78651326]
 [12.43939243 14.88185391 22.87212502 85.40081418 169.20470771
 207.75930082 203.17453934 206.48593038]
 [14.86246576 13.85770166 48.34753653 138.60989735 197.56653649
 200.45595823 204.52022042 204.64583789]
 [10.30696212 22.26306212 98.43990676 193.29598713 197.49865555
 197.15239025 202.78354053 204.34936995]
 [13.54618762 43.57624439 170.14727313 194.46189951 199.20306693
 196.44908615 203.6960401 206.81911393]
 [19.43770159 79.82372056 191.83799586 194.76091909 198.42935828
 194.59755565 203.06452211 206.17380374]]
```

```
*** I_subdivision[0] ***
```

```
[[ -1.37650739 -0.34816322 7.48009577 40.10323423 23.43393055 23.50614674
 23.77449821 22.04084312]
 [ -0.67281877 9.81601163 32.77031007 24.66357793 25.13318892 25.06022458
 25.34709795 23.49811295]
 [ -1.49500304 27.43153729 28.05000943 29.57666412 26.57438595 25.07479459
 23.39341602 22.79067112]
 [ 4.85032311 35.24380517 31.60554463 29.21250959 25.86068235 25.07296597
 22.16198941 20.55683995]
 [14.21769452 28.15846296 32.99963757 30.85775375 27.34163141 25.18286529
 24.15664821 22.46134852]
 [24.87459721 27.82479738 30.66728455 28.00112679 26.8816256 25.18244277
 24.15586812 22.46353152]
 [24.65229049 29.48479958 28.86193202 27.11286345 26.7930454 25.18424037
 24.57082365 23.15532312]
 [24.68292762 29.49137592 29.75943823 28.06003203 26.95166465 25.18161551
 24.57394448 23.15515128]]
```

```
*** Q_subdivision[0] ***
```

```
[[ 1.6407458 -1.8516 -1.15393733 9.4447258 5.30956685 4.71288978
 5.26694467 5.41979362]
 [2.72731369 0.39089901 9.13095415 2.92051962 4.91923476 5.0247
 5.6154118 5.77837341]
 [0.64279961 4.81605559 2.15517927 4.67817649 5.48214215 5.02875288
 4.27252521 5.47565497]
 [3.92788343 4.49380208 4.83153357 4.40387037 5.0268078 5.02880368
 3.56724285 3.89965281]
 [4.81057278 2.6631058 3.47092516 3.83906671 4.0086295 4.71698429
 4.95751219 5.20389939]
 [2.90343916 5.11708435 4.01700796 3.6459422 4.03658489 4.71678578
 4.95838486 5.20451828]
 [4.0530698 5.17598224 6.16872126 5.21222633 4.1319734 4.71678578
 4.0836227 4.12758315]
 [4.054136 5.17552756 4.62417116 3.53309405 3.85620114 4.71635774
 4.08324666 4.12736603]]
```

Discrete Cosine Transform (DCT II)

```
def dct_2d(im):
    return fftpack.dct(fftpack.dct(im.T, norm='ortho').T, norm='ortho')

def idct_2d(im):
    return fftpack.idct(fftpack.idct(im.T, norm='ortho').T, norm='ortho')

plt.rcParams["figure.figsize"] = [15,15]
plt.rc('xtick', labelsz=10)
plt.rc('ytick', labelsz=10)

def getdct(getsubdivisions):
    storedct = [] # Initializing an empty storage
    for i in range(0, len(getsubdivisions)): # step = 1
        storedct.append(dct_2d((getsubdivisions[i]))) # Apply dct_2d for each block
    return storedct # Returns the stored dct values

Y_dct=getdct(Y_subdivision)
plt.subplot(2,3,1),plt.imshow(Y_dct[0]),plt.title("Y_dct"), plot_histogram(Y_dct[0],2,3,4)

I_dct=getdct(I_subdivision)
plt.subplot(2,3,2),plt.imshow(I_dct[0]),plt.title("I_dct"), plot_histogram(I_dct[0],2,3,5)

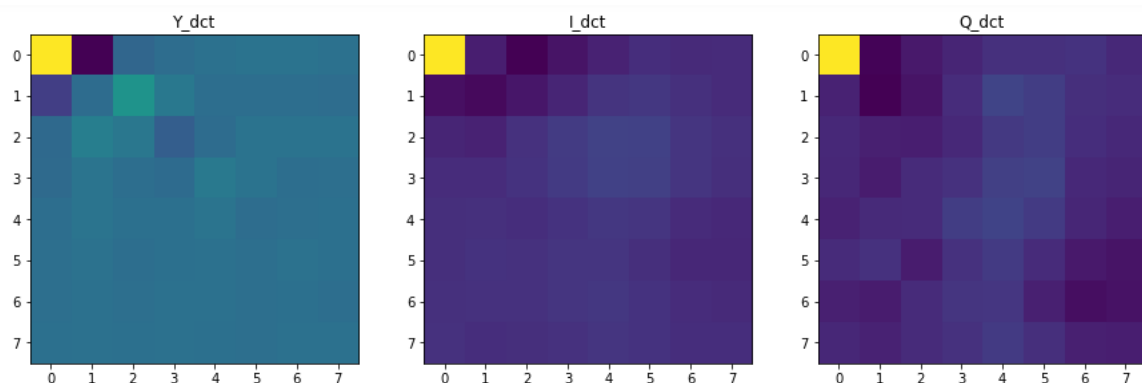
Q_dct=getdct(Q_subdivision)
plt.subplot(2,3,3),plt.imshow(Q_dct[0]),plt.title("Q_dct"), plot_histogram(Q_dct[0],2,3,6)

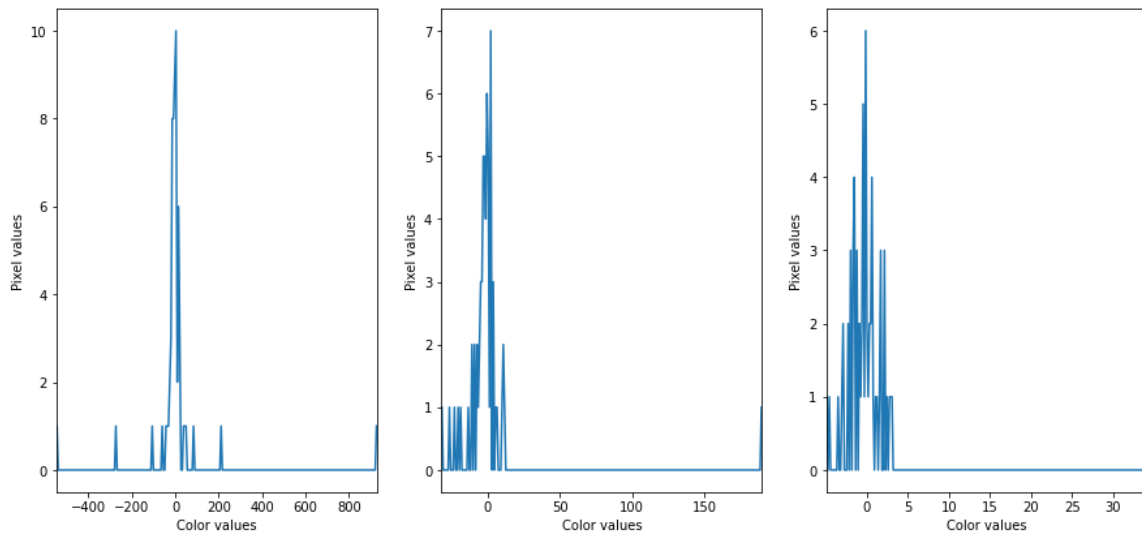
plt.show()

print("*** Y_dct[0] ***\n\n", Y_dct[0])
print("\n\n*** I_dct[0] ***\n\n", I_dct[0])
print("\n\n*** Q_dct[0] ***\n\n", Q_dct[0])
```

The aim of “`getdct`” function is to take the subblocks in and apply “`dct_2d`” to each block and returns “`storedct`” as stored values.

NB: In the for loop, Step = 1;





*** Y_dct[0] ***

```
[[ 9.29661163e+02 -5.46949718e+02 -5.85628064e+01 -1.98412801e+01
 6.04801501e+00 1.75526248e+01 1.27265765e+01 5.31405103e+00]
 [-2.73051715e+02 -2.65645201e+01 2.09177682e+02 4.37808690e+01
 -1.15358267e+01 -1.34790467e+01 -1.36567584e+01 -1.64498349e+01]
 [-4.07310870e+01 8.58769829e+01 4.06851823e+01 -1.05377176e+02
 -2.48234188e+01 1.42471186e+01 1.37132353e+01 1.27149669e+01]
 [-3.68964203e+01 2.23418521e+01 -1.55258312e+01 -3.04824818e+01
 4.99047353e+01 1.75882710e+01 -1.18904778e+01 -8.30054035e+00]
 [-1.32933887e+01 2.12476768e+01 -2.98837444e+00 8.56995692e-01
 1.96089614e+01 -2.07202561e+01 -8.23983269e+00 -1.36271264e-01]
 [-6.47765313e+00 8.58194900e+00 -1.10348952e+01 -1.62417705e+00
 4.49463439e+00 -9.85557876e+00 1.11456636e+01 -2.07899645e+00]
 [-5.46364822e+00 5.86052691e+00 -6.55346735e+00 4.23638581e+00
 2.61857578e+00 -6.25652780e+00 -5.48414752e-01 -1.52113582e+01]
 [-2.62331050e+00 2.21787938e+00 -5.79176675e-01 3.21455115e+00
 -3.80475580e+00 -4.81460677e+00 6.07521280e+00 2.23313006e+00]]
```

*** I_dct[0] ***

```
[[ 1.89661213e+02 -1.31899354e+01 -3.17276503e+01 -1.98669506e+01
 -1.05579024e+01 -3.01392059e+00 -5.02863005e+00 -4.24106873e+00]
 [-2.23824628e+01 -2.60766244e+01 -1.81917347e+01 -8.41913747e+00
 1.50466413e+00 4.38712918e+00 -1.24317979e+00 -4.66143518e+00]
 [-9.16305997e+00 -1.07777942e+01 1.72952308e-01 6.70569046e+00
 1.22608880e+01 1.08094792e+01 2.60302073e+00 -1.08349513e+00]
 [-3.21114048e+00 -3.22974349e+00 7.10501852e-01 6.32129125e+00
 1.11545178e+01 9.93307464e+00 2.00894766e+00 -1.58192608e+00]
 [-2.05848033e+00 -1.05935982e+00 -3.17589787e+00 6.90849966e-01
 4.16019946e+00 2.19074270e+00 -4.23367892e+00 -6.19726268e+00]
 [-2.10724799e+00 4.90080546e-01 -3.21008113e-01 2.23488467e+00
 2.59065830e+00 -1.66433920e+00 -6.71310634e+00 -7.49904420e+00]
 [-7.30795856e-01 -3.43974940e-01 -5.53421994e-01 2.12492720e+00
 4.05443233e+00 5.40919577e-01 -4.01023669e+00 -4.95377361e+00]
 [-5.22336230e-01 -2.67932650e+00 -1.74318835e+00 1.70367077e-02
 2.66016985e+00 6.86109758e-01 -2.38118904e+00 -2.61238326e+00]]
```

*** Q_dct[0] ***

```
[[34.14223959 -4.60176869 -2.32404092 -0.9043234 0.46915142 0.40127385
 0.7825551 -0.38967532]
 [-1.25297724 -4.9599843 -2.83191365 -0.05144127 3.12570217 2.11654979
 0.29153981 0.07499496]
 [-0.48680294 -1.57448824 -1.60479773 -0.49012291 1.47040322 2.14768402
 0.1483944 -0.4374043]
 [-0.54385489 -1.95655331 -0.08688305 0.56103152 2.41006858 2.71632457
 -0.40225353 -0.93525237]
 [-1.1898591 -0.28000914 -0.09686727 2.14827883 2.83147414 1.71139425
 -0.79343322 -1.67195669]
 [-0.17577038 0.58855662 -1.90731203 0.55033479 1.66735559 -0.19021788
 -2.26457756 -2.92483005]
 [-1.54606993 -2.01565754 -0.09600518 1.10492705 1.26426371 -1.54023813
 -3.51689227 -3.10082251]
 [-0.63410297 -1.20738431 -0.11508448 0.67616209 1.6494174 0.30002872
 -1.47875218 -1.73576462]]
```

-Observing the top-left corner, we can notice a big difference compared to the other blocks located anywhere else in the matrix, the matrix figure also notes that the top-left corner values are much higher than the other elements of the matrix which they basically represent the general intensity of the block.

Quantization

```
def quantization_table(component='lum'):
    # Quantization Table for: Photoshop
    # (http://www.impulseadventure.com/photo/jpeg-quantization.html)
    if component == 'lum':
        q = np.array([[2, 2, 2, 2, 3, 4, 5, 6],
                       [2, 2, 2, 2, 3, 4, 5, 6],
                       [2, 2, 2, 2, 4, 5, 7, 9],
                       [2, 2, 2, 4, 5, 7, 9, 12],
                       [3, 3, 4, 5, 8, 10, 12, 12],
                       [4, 4, 5, 7, 10, 12, 12, 12],
                       [5, 5, 7, 9, 12, 12, 12, 12],
                       [6, 6, 9, 12, 12, 12, 12, 12]])
    elif component == 'chrom':
        q = np.array([[3, 3, 3, 5, 9, 13, 15, 15, 15],
                       [3, 4, 6, 11, 14, 12, 12, 12],
                       [5, 6, 9, 14, 12, 12, 12, 12],
                       [9, 11, 14, 12, 12, 12, 12, 12],
                       [13, 14, 12, 12, 12, 12, 12, 12],
                       [15, 12, 12, 12, 12, 12, 12, 12],
                       [15, 12, 12, 12, 12, 12, 12, 12],
                       [15, 12, 12, 12, 12, 12, 12, 12]])
    else:
        raise ValueError((
            "component should be either 'lum' or 'chrom',"
            "but '{comp}' was found").format(comp=component))
    return q
```

```
def Quantize(getdct, cst = " "): # A function that takes dct as a data, divides each element by the quantization table, rounds them off and finally stores it.
    if (cst == "lum" or cst == "chrom"):
        component = cst
    else:
        print("***Please enter 'lum' or 'chrom'***")

    quantized = []
    for i in range(0, len(getdct)): # Step = 1
        ratio = (getdct[i] / quantization_table(component)) # Dividing each element of the dct by the quantization table
        quantized.append(ratio.astype(np.int32)) # rounding them off
    return quantized

quantizedY = Quantize(Y_dct, "lum")
quantizedI = Quantize(I_dct, "chrom")
quantizedQ = Quantize(Q_dct, "chrom")

print("*** Quantized block of Y ***\n\n", quantizedY[0])
print("***\n\n Quantized block of I ***\n\n", quantizedI[0])
print("***\n\n Quantized block of Q ***\n\n", quantizedQ[0])
```

Let's look at the quantized block of each channel at index = 0.

```
[[ 464 -273 -29 -9  2  4  2  0]
 [-136 -13 104 21 -3 -3 -2 -2]
 [-20  42  20 -52 -6  2  1  1]
 [-18  11  -7 -7  9  2 -1  0]
 [ -4  7  0  0  2 -2  0  0]
 [ -1  2 -2  0  0  0  0  0]
 [ -1  1  0  0  0  0  0 -1]
 [  0  0  0  0  0  0  0  0]]
```

```
[[63 -4 -6 -2 0 0 0 0]
[-7 -6 -3 0 0 0 0 0]
[-1 -1 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0]]
```

[illegible]

From the above figures, the top-left corner values are much bigger, and the others have been rounded off to zero; this plays profoundly important role in freeing up spaces and it won't have an eye-visible effect on the images.

(Ref: <https://www.delftstack.com/howto/numpy/python-element-wise-division-numpy/>)

Dequantization

```
def Dequantize(getquantized, cst = ""): # It takes the quantized-data, multiplies each element by the quantization -table, rounds them off and finally stores and return
    if (cst == "lum" or cst == "chrom"):
        component = cst
    else:
        print("Please enter 'lum' or 'chrom'")

    dequantized = []
    for i in range (0, len(getquantized)):
        ratio = (getquantized[i] * quantization_table(component)) # Step = 1
        dequantized.append(ratio.astype(np.int32)) # Dividing each element of the dct by the quantization table
        # Rounding them off
    return dequantized

dequantizedY = Dequantize(quantizedY, "lum")
dequantizedI = Dequantize(quantizedI, "chrom")
dequantizedQ = Dequantize(quantizedQ, "chrom")

print("Dequantized block of Y", dequantizedY[0])
print("Dequantized block of I", dequantizedI[0])
print("Dequantized block of Q", dequantizedQ[0])
```

The function gets the quantized-data as an input and will multiply each element by the same quantization table that we used to divide them. Similarly, depending on the channel it's dealing with, the luminance or chrominance table will be chosen. If it's not between these two, an error message will be printed out to let the user choose just from the given two choices.

Let's look at the dequantized block of each channel at index = 0.

*** Dequantized block of Y ***

```
[[ 928 -546 -58 -18  6 16 10  0]
 [-272 -26 208 42 -9 -12 -10 -12]
 [-40 84 40 -104 -24 10 7 9]
 [-36 22 -14 -28 45 14 -9 0]
 [-12 21 0 0 16 -20 0 0]
 [-4 8 -10 0 0 0 0 0]
 [-5 5 0 0 0 0 0 -12]
 [ 0 0 0 0 0 0 0 0]]
```

*** Dequantized block of I ***

```
[[189 -12 -30 -18 0 0 0 0]
 [-21 -24 -18 0 0 0 0 0]
 [-5 -6 0 0 12 0 0 0]
 [ 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0]]
```

*** Dequantized block of Q ***

```
[[33 -3 0 0 0 0 0 0]
 [ 0 -4 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0]]
```

- From the above figures, in fact, multiplying the quantization-table values by zero won't have an effect but the effect is quite obvious for the top-left corner values.

Inverse Discrete Cosine Transform (IDCT)

```
plt.rcParams["figure.figsize"] = [15,15]
plt.rc('xtick', labels=10)
plt.rc('ytick', labels=10)

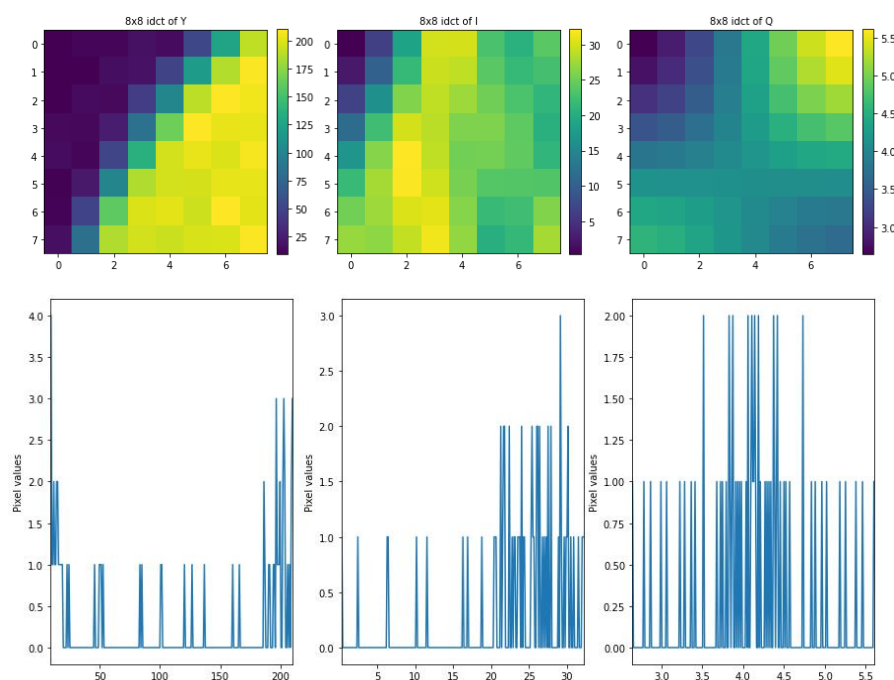
def getidct(getdequantized):
    storeidct = []
    for i in range(0, len(getdequantized)):
        storeidct.append(idct_2d(getdequantized[i]))
    return storeidct

idctY = getidct(dequantizedY)
idctI = getidct(dequantizedI)
idctQ = getidct(dequantizedQ)

plt.subplot(2,3,1), plt.imshow(idctY[0]), plt.title("8x8 idct of Y", fontsize = 10), plt.colorbar(fraction=0.046, pad=0.04), plot_histogram(idctY[0], 2,3,4)
plt.subplot(2,3,2), plt.imshow(idctI[0]), plt.title("8x8 idct of I", fontsize = 10), plt.colorbar(fraction=0.046, pad=0.04), plot_histogram(idctI[0], 2,3,5)
plt.subplot(2,3,3), plt.imshow(idctQ[0]), plt.title("8x8 idct of Q", fontsize = 10), plt.colorbar(fraction=0.046, pad=0.04), plot_histogram(idctQ[0], 2,3,6)
plt.show()
```

- Each block (8 x 8) will be undertaken through the inverse cosine transform function.

- Examples are shown below;



Reconstruction of an image

```
plt.rcParams["figure.figsize"] = [13,13]
plt.rc('xtick', labelsize=10)
plt.rc('ytick', labelsize=10)

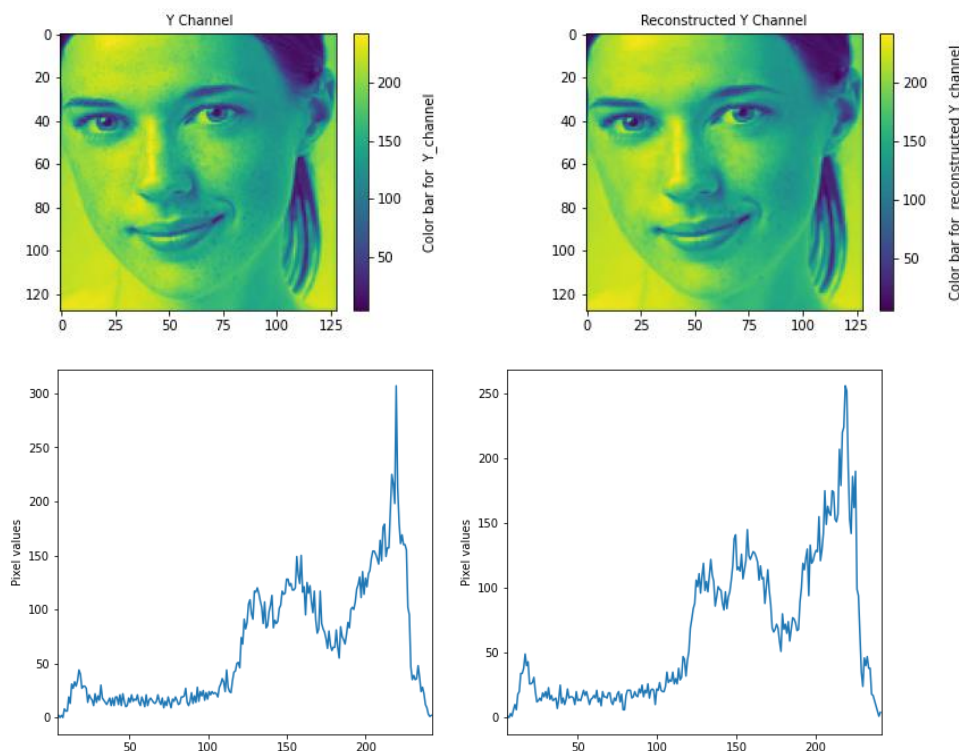
def form(getdict, w):
    container = []
    for i in range(0, len(getdict), round(w/8)):
        storeX = np.hstack(getdict[i:i+round(w/8)])
        container.append(storeX)
    storeY = np.vstack(container)
    return storeY

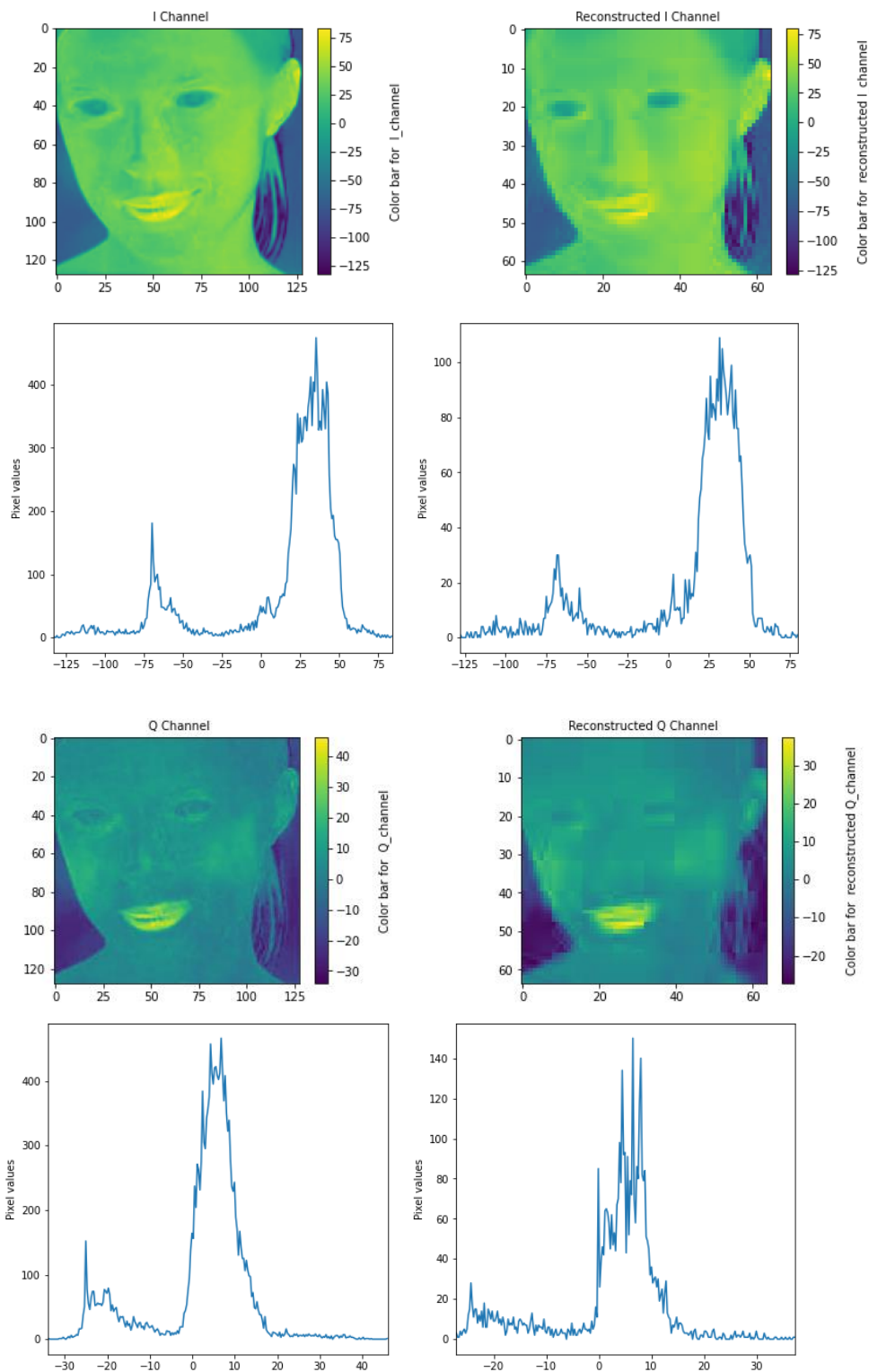
Y2 = form(idctY, len(idctY)/2)
I2 = form(idctI, len(idctI))
Q2 = form(idctQ, len(idctQ))

plt.subplot(3,2,1), plt.imshow(Y_channel), plt.colorbar(label="\nColor bar for Y_channel", fraction=0.046, pad=0.04), plt.title("Y Channel", fontsize=10) #, plot_hist
plt.subplot(3,2,3), plt.imshow(I_channel), plt.colorbar(label="\nColor bar for I_channel", fraction=0.046, pad=0.04), plt.title("I Channel", fontsize=10) #, plot_hist
plt.subplot(3,2,5), plt.imshow(Q_channel), plt.colorbar(label="\nColor bar for Q_channel", fraction=0.046, pad=0.04), plt.title("Q Channel", fontsize=10) #, plot_hist

plt.subplot(3,2,2), plt.imshow(Y2), plt.colorbar(label="\nColor bar for reconstructed Y_channel", fraction=0.046, pad=0.04), plt.title("Reconstructed Y Channel", fontsi
plt.subplot(3,2,4), plt.imshow(I2), plt.colorbar(label="\nColor bar for reconstructed I_channel", fraction=0.046, pad=0.04), plt.title("Reconstructed I Channel", fontsi
plt.subplot(3,2,6), plt.imshow(Q2), plt.colorbar(label="\nColor bar for reconstructed Q_channel", fraction=0.046, pad=0.04), plt.title("Reconstructed Q Channel", font
plt.show()
```

- The function uses the inverse discrete cosine transform (IDCT) as an input-data and we just need to form 8 x 8 blocks having a step = 8 as usual, put them horizontally and finally stack all those together vertically. Basically, there is no significance difference between the original and reconstructed channels if we just had a quick glance at them. but the difference is noticeable on the histogram-graph.





- Why does the reconstructed Y channel look exactly the same as the original one?

- I was wondering about this question and finally I noticed that its quantization table values are relatively smaller than that of I and Q channels.

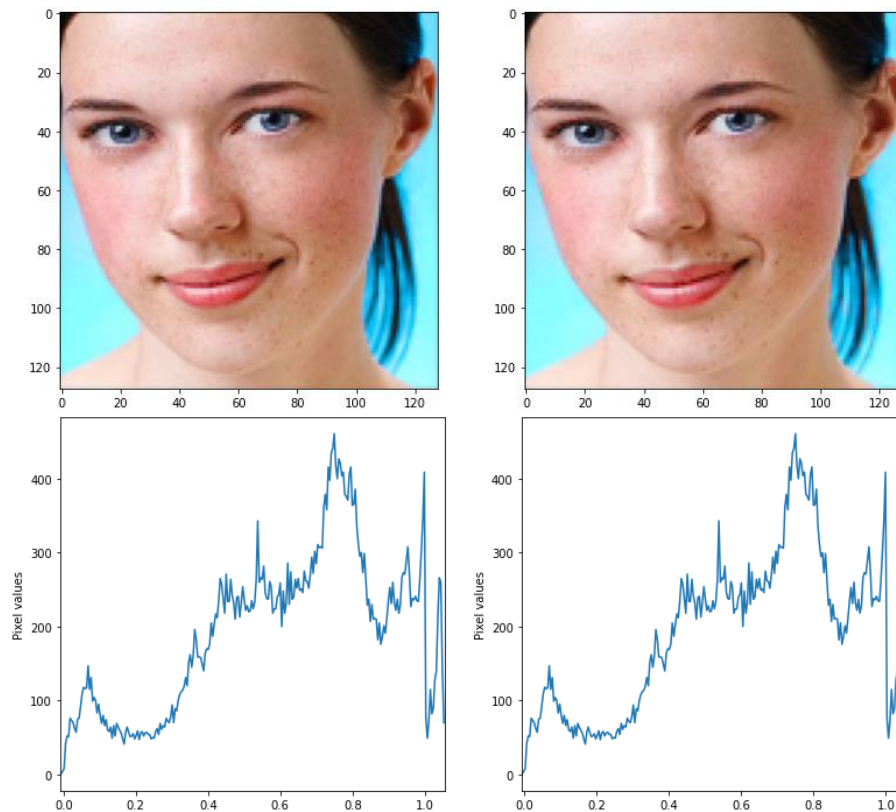
(Ref: <https://www.geeksforgeeks.org/numpy-vstack-in-python/>, <https://www.geeksforgeeks.org/numpy-hstack-in-python/>)

RGB Reconstruction

```
plt.rcParams["figure.figsize"] = [13,13]
plt.rc('xtick', labels=10)
plt.rc('ytick', labels=10)

RGB_reconstruction = YIQ_RGB(Y_channel, Iup, Qup)
plt.subplot(2,2,1), plt.imshow(RGB_image), plot_histogram(RGB_image, 2,2,3)
plt.subplot(2,2,2), plt.imshow(RGB_reconstruction), plot_histogram(RGB_image, 2,2,4)
plt.show()
```

Here are some images in order to compare the difference; in fact, the difference's so subtle we can't really tell the subtle change.



JPEG Compression

```
def JPEGCompression(getimage):
    print(getimage.shape)

    # Step 1 : Changing the color space to YIQ
    YIQimage = RGB_YIQ(getimage)

    # Step 2: Down-sampling
    Idown = downsample(YIQimage[:, :, 1], 2)
    Qdown = downsample(YIQimage[:, :, 2], 2)

    # Step 3: 8 x 8 blocks formation
    Y_subdivision = subdivision(YIQimage[:, :, 0])
    I_subdivision = subdivision(Idown)
    Q_subdivision = subdivision(Qdown)

    # Step 4: Apply dct to each block
    Y_dct = getdct(Y_subdivision)
    I_dct = getdct(I_subdivision)
    Q_dct = getdct(Q_subdivision)

    # Step 5: Divide by the quantization table accordingly
    quantizedY = Quantize(Y_dct, "lum")
    quantizedI = Quantize(I_dct, "chrom")
    quantizedQ = Quantize(Q_dct, "chrom")

    # Step 6 : I returned these quantized channels, because this'll be the first step of the decompression
    return quantizedY, quantizedI, quantizedQ
```

JPEG Decompression

```
def JPEGDecompression(quantizedY, quantizedI, quantizedQ, imageshape):
    #print("Shape: ", imageshape)
    (h,w,c) = imageshape

    # Step 5: Get the quantized values and multiply by the quantization table accordingly
    dequantizedY = Dequantize(quantizedY, "lum")
    dequantizedI = Dequantize(quantizedI, "chrom")
    dequantizedQ = Dequantize(quantizedQ, "chrom")

    # Step 4: Apply IDCT to each block
    idctY = getidct(dequantizedY)
    idctI = getidct(dequantizedI)
    idctQ = getidct(dequantizedQ)

    # Step 3: Form subdivisions
    Y_subdivision = form(idctY, w)
    I_subdivision = form(idctI, w/2)
    Q_subdivision = form(idctQ, w/2)

    # Step 2: Up-sample I and Q subdivided channels
    Iup = upsample(I_subdivision, 2)
    Qup = upsample(Q_subdivision, 2)

    # Step 1: Conversion to an RGB image
    RGBImage = YIQ_RGB(Y_subdivision, Iup, Qup)

    # Boom ! you have successfully formed an RGB-image
    return RGBImage
```

-Here in the JPEG compression, we write all the functions together under a function called “JPEGCompression”. And, in “JPEGDecompression”, we follow the step backwards from getting the quantized values all the way up to the conversion of an RGB image; which is our final output.

```
filename = "images/Frequency/face2.jpg" # Let's test the function
image = io.imread(filename)
image = (image/255).astype(float) # To be sure is always between 0 and 1 after this cell
```

```
plt.rcParams['figure.figsize'] = [10,10]
plt.rc('xtick', labelsz=10)
plt.rc('ytick', labelsz=10)

ycomp, icoomp, qcomp, imageshape = JPEGCompression(image)
newimage = JPEGDecompression(ycomp, icoomp, qcomp, imageshape)

h = 128
w = 128
resizedimage = resize(image, (h,w), mode='constant')

#####
print("\n*** After Resizing it ***\n")
#####

Ycomp, Icomp, Qcomp, ImageShape = JPEGCompression(resizedimage)
newimageresized = JPEGDecompression(Ycomp, Icomp, Qcomp, ImageShape)

plt.subplot(2,2,1), plt.imshow(image), plt.title("RGB Image", fontsize = 15), plot_histogram(image, 2,2,3)
plt.subplot(2,2,2), plt.imshow(newimage), plt.title('Reconstructed RGB Image', fontsize = 10), plot_histogram(newimage, 2,2,4)
plt.show()
```

- I have been stuck for quite a while wondering why it'd been printing out error messages shown below.

```
(300, 227, 3)

ValueError                                Traceback (most recent call last)
<ipython-input-72-9d00a8d24be3> in <module>
      3 plt.rc('ytick', labelsz=10)
      4
----> 5 ycomp, icoomp, qcomp, shape = JPEGCompression(image)
      6
      7 h = 128

<ipython-input-66-9f813d27e4c3> in JPEGCompression(getimage)
     21
     22 # Step 5: Divide by the quantization table accordingly
--> 23 quantizedY = Quantize(Y_dct, "lum")
     24 quantizedI = Quantize(I_dct, "chrom")
     25 quantizedQ = Quantize(Q_dct, "chrom")

<ipython-input-14-67191f3cab3c> in Quantize(getdct, cst)
      9 quantized = []
     10 for i in range(0, len(getdct)):
--> 11     ratio = (getdct[i]/quantization_table(component))) #Dividing each element of the dct by the quantization table
     12     quantized.append(ratio.astype(np.int32)) #rounding them off
     13 return quantized

ValueError: operands could not be broadcast together with shapes (8,3) (8,8)
```

- As we can see the shape of the image is not compatible, it's not even a multiple of 8; so we need to adjust that whenever any shape is introduced in the function, it has to approximate to the nearest multiple of 8.

Here's how;

```
if( getimage.shape[0] % 8 != 0 or getimage.shape[1] % 8 != 0 ):
    getimage = resize(getimage, ((getimage.shape[0] + (8 - getimage.shape[0] % 8)), (getimage.shape[1] + (8 - getimage.shape[1] % 8))))
```

(Ref: <https://stackoverflow.com/questions/4666787/getting-random-number-divisible-by-16>)

Original shape = (300, 227, 3)
 Resized shape (304, 232, 3)
 Shape of I after down-sampled= (152, 116)
 Shape of Q after down-sampled= (152, 116)

```

ValueError                                Traceback (most recent call last)
<ipython-input-130-462e32c53d1b> in <module>
      3 plt.rc('ytick', labelsize=10)
      4
----> 5 ycomp, icomp, qcomp = JPEGCompression(image)
      6
      7 h = 128

<ipython-input-128-d13f67bf80e4> in JPEGCompression(getimage)
     29 # Step 5: Divide by the quantization table accordingly
     30 quantizedY = Quantize(Y_dct, "lum")
--> 31 quantizedI = Quantize(I_dct, "chrom")
     32 quantizedQ = Quantize(Q_dct, "chrom")
     33

<ipython-input-14-67191f3cab3c> in Quantize(getdct, cst)
      9 quantized = []
     10 for i in range(0, len(getdct)):
--> 11     ratio = (getdct[i] / quantization_table(component))) #Step = 1
     12     quantized.append(ratio.astype(np.int32)) #Dividing each element of the dct by the quantization table
     13     return quantized #rounding them off
  
```

ValueError: operands could not be broadcast together with shapes (8,4) (8,8)

- After having it resized, there is also an error, but let's analyze it. As we can see the original shape of the image was 300 x 227 where each dimension doesn't fulfill the requirement as a multiple of 8. So, right after resizing it to the nearest multiple of 8, we got 304 x 232. Next, we need to down-sample the I and Q channels; the sampling frequency is equal to 2;

- $(304 / 2, 232 / 2) = (152, 116)$

Moving on, it's time to form 8 x 8 blocks, we know that the blocks are formed along x and y axis;

```

def subdivision(getimage):
    (x,y) = getimage.shape[0], getimage.shape[1] # A function that takes the data in
    store_blocks = [] # Gets the size of the rows and columns which will make it its height and width
    for i in range(0,x,8): # Stores the 8 x 8 blocks
        for j in range(0,y,8): # Step along x = 8
            store_blocks.append(getimage[i:i+8,j:j+8]) # Step along y = 8
    return store_blocks # 8 x 8 blocks formation
                        # Returns the blocks that's been stored
  
```

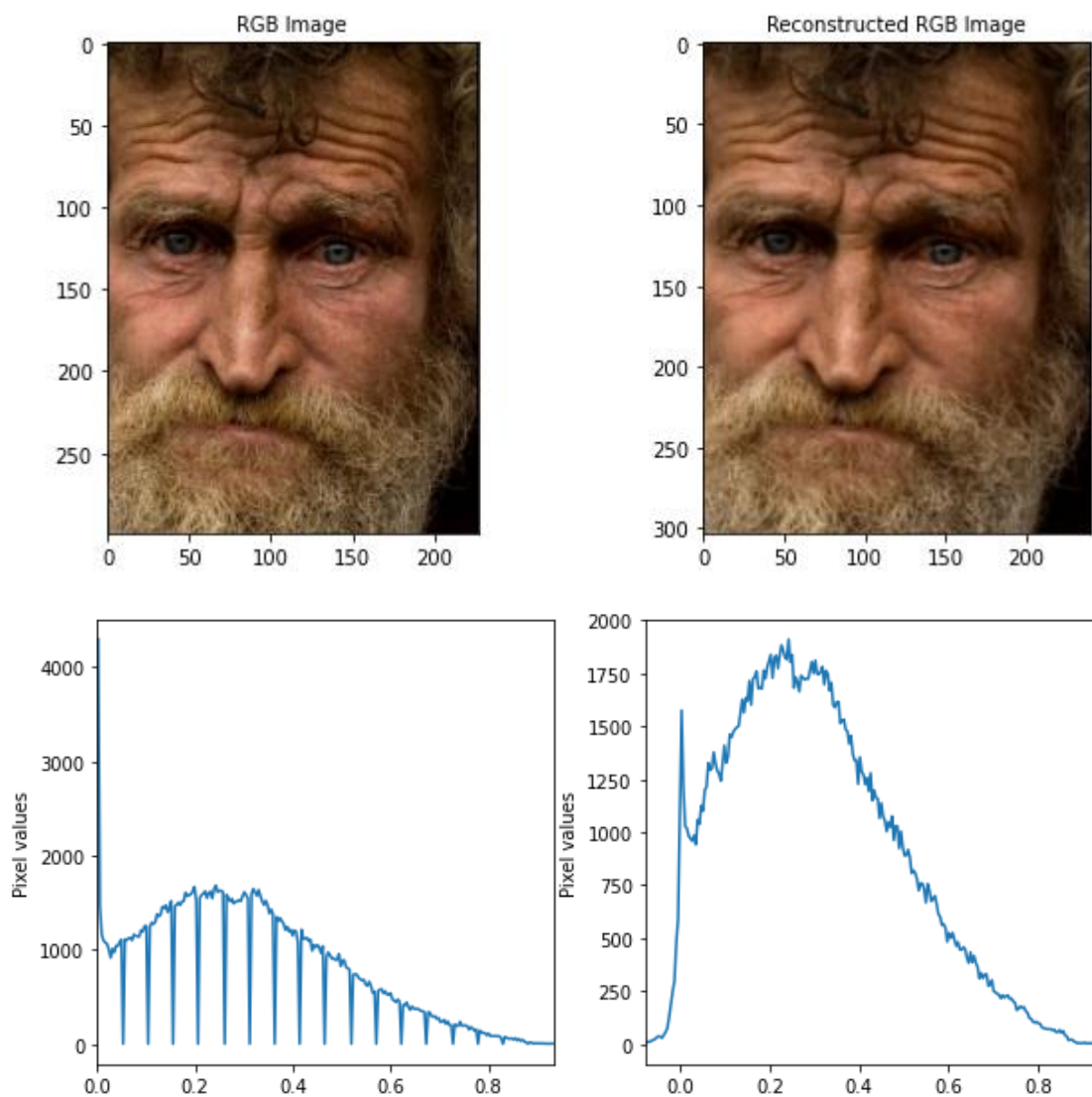
Therefore, we'll have 19 blocks along x direction, 14 blocks of 8 x 8 and remaining 4 blocks along y direction. Observing where the error is, it still passes the subdivision and getdct function. But the problem is while dividing with the quantization table because the last block of ours is (8, 4) whereas the dimension of the quantization table is 8 x 8 which indicates shape-incompatibility.

➤ So, we need to upgrade the condition from 8 to 16;

```

if( getimage.shape[0] % 16 != 0 or getimage.shape[1] % 16 != 0 ):
    getimage = resize(getimage, ((getimage.shape[0] + (16 - getimage.shape[0] % 16)), (getimage.shape[1] + (16 - getimage.shape[1] % 16))))
  
```

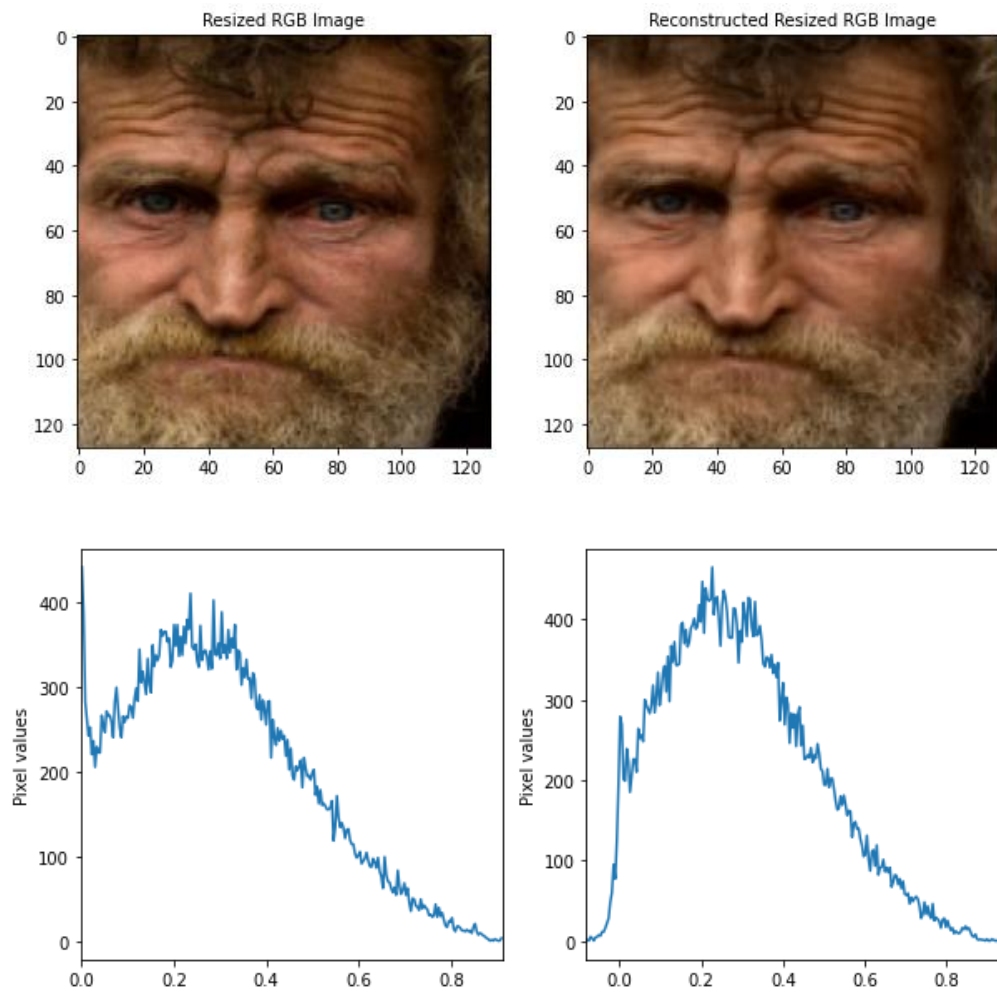
- The problem has been resolved that way and see the images below;



```
plt.rcParams['figure.figsize'] = [10,10]
plt.rc('xtick', labels=10)
plt.rc('ytick', labels=10)

plt.subplot(2,2,1), plt.imshow(resizedimage), plt.title("Resized RGB Image", fontsize = 10), plot_histogram(resizedimage,2,2,3)
plt.subplot(2,2,2), plt.imshow(newimageresized), plt.title("Reconstructed Resized RGB Image", fontsize = 10), plot_histogram(newimageresized,2,2,4)
plt.show()
```

- See images below for resized RGB image and Reconstructed-resized RGB image



```
print("*****")
print("Image shape :", image.shape)
print("New image shape :", newimage.shape)
newimage = resize(newimage, (image.shape[0], image.shape[1]), mode='constant') # Making the shape compatible to each other, otherwise operations aren't possible
print("New resized image shape :", newimage.shape)

D1 = image - newimage # original - reconstructed
D2 = resizedimage - newimageresized # original resized - reconstructed resized

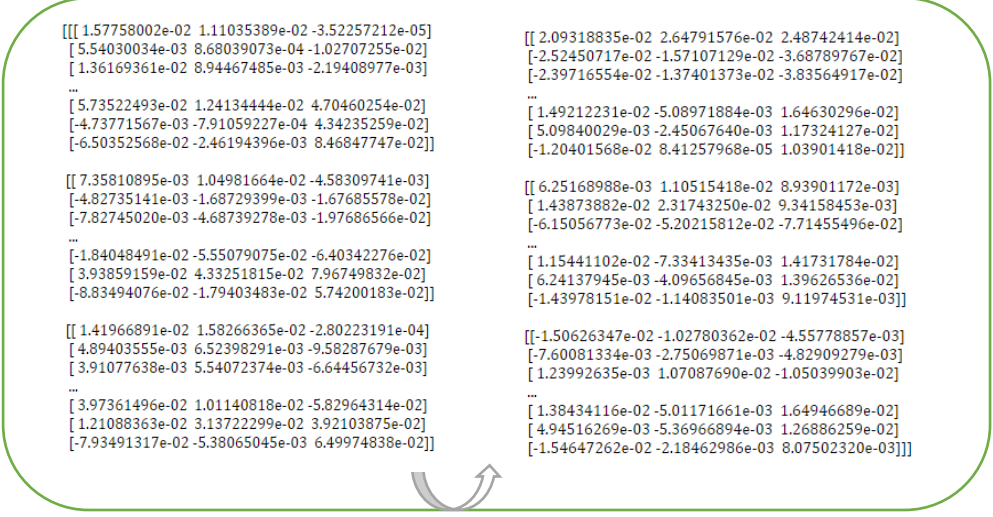
print("*****")
print("D1 Min = ", D1.min(), "\nD1 Max = ", D1.max())
print("*****")
print("D2 Min = ", np.amin(D2), "\nD2 Max = ", np.amax(D2))
print("*****\n\n")

# print(D1)
# print(D2)

*****
Image shape : (300, 227, 3)
New image shape : (300, 227, 3)
New resized image shape : (300, 227, 3)
*****
D1 Min = -0.1909008461603524
D1 Max = 0.16203716834390924
*****
D2 Min = -0.08895161276249738
D2 Max = 0.09478106198045255
*****
```

- (Original RGB image – reconstructed image) is shown below having;

[min, max] = [-0.19090084616034697, 0.1620371683439018]



```

[[[ 1.57758002e-02  1.11035389e-02 -3.52257212e-05]
 [ 5.54030034e-03  8.68039073e-04 -1.02707255e-02]
 [ 1.36169361e-02  8.94467485e-03 -2.19408977e-03]
 ...
 [ 5.73522493e-02  1.24134444e-02  4.70460254e-02]
 [-4.73771567e-03 -7.91059227e-04  4.34235259e-02]
 [-6.50352568e-02 -2.46194396e-03  8.46847747e-02]]]

[[[ 2.09318835e-02  2.64791576e-02  2.48742414e-02]
 [-2.52450717e-02 -1.57107129e-02 -3.68789767e-02]
 [-2.39716554e-02 -1.37401373e-02 -3.83564917e-02]
 ...
 [ 1.49212231e-02 -5.08971884e-03  1.64630296e-02]
 [ 5.09840029e-03 -2.45067640e-03  1.17324127e-02]
 [-1.20401568e-02  8.41257968e-05  1.03901418e-02]]]

[[[ 7.35810895e-03  1.04981664e-02 -4.58309741e-03]
 [-4.82735141e-03 -1.68729399e-03 -1.67685578e-02]
 [-7.82745020e-03 -4.68739278e-03 -1.97686566e-02]
 ...
 [-1.84048491e-02 -5.55079075e-02 -6.40342276e-02]
 [ 3.93859159e-02  4.33251815e-02  7.96749832e-02]
 [-8.83494076e-02 -1.79403483e-02  5.74200183e-02]]]

[[[ 6.25168988e-03  1.10515418e-02  8.93901172e-03]
 [ 1.43873882e-02  2.31743250e-02  9.34158453e-03]
 [-6.15056773e-02 -5.20215812e-02 -7.71455496e-02]
 ...
 [ 1.15441102e-02 -7.33413435e-03  1.41731784e-02]
 [ 6.24137945e-03 -4.09656845e-03  1.39626536e-02]
 [-1.43978151e-02 -1.14083501e-03  9.11974531e-03]]]

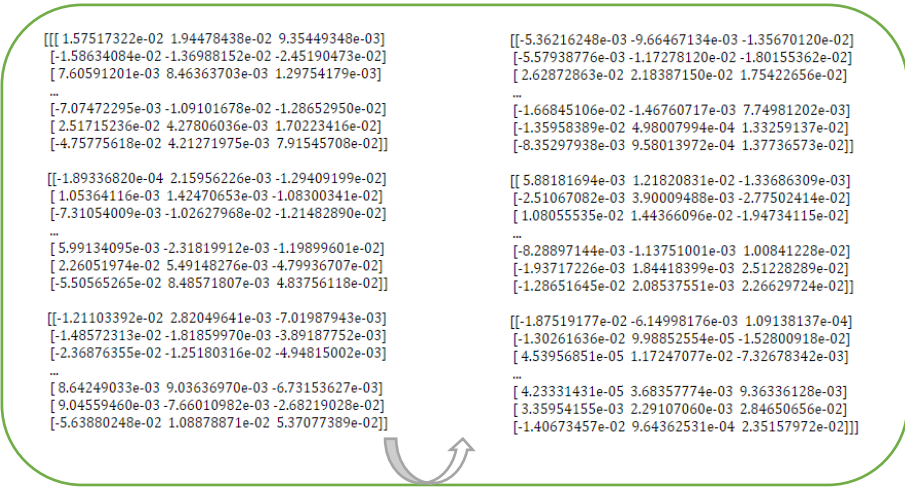
[[[ 1.41966891e-02  1.58266365e-02 -2.80223191e-04]
 [ 4.89403555e-03  6.52398291e-03 -9.58287679e-03]
 [ 3.91077638e-03  5.54072374e-03 -6.64456732e-03]
 ...
 [ 3.97361496e-02  1.01140818e-02 -5.82964314e-02]
 [ 1.21088363e-02  3.13722299e-02  3.92103875e-02]
 [-7.93491317e-02 -5.38065045e-03  6.49974838e-02]]]

[[[ -1.50626347e-02 -1.02780362e-02 -4.55778857e-03]
 [-7.60081334e-03 -2.75069871e-03 -4.82909279e-03]
 [ 1.23992635e-03  1.07087690e-02 -1.05039903e-02]
 ...
 [ 1.38434116e-02 -5.01171661e-03  1.64946689e-02]
 [ 4.94516269e-03 -5.36966894e-03  1.26886259e-02]
 [-1.54647262e-02 -2.18462986e-03  8.07502320e-03]]]

```

- (Original resized image – Reconstructed resized image) is shown below having;

[min, max] = [-0.08895161276249738, 0.09478106198045255]



```

[[[ 1.57517322e-02  1.94478438e-02  9.35449348e-03]
 [-1.58634084e-02 -1.36988152e-02 -2.45190473e-02]
 [ 7.60591201e-03  8.46363703e-03  1.29754179e-03]
 ...
 [-7.07472295e-03 -1.09101678e-02 -1.28652950e-02]
 [ 2.51715236e-02  4.27806036e-03  1.70223416e-02]
 [-4.75775618e-02  4.21271975e-03  7.91545708e-02]]]

[[[ -5.36216248e-03 -9.66467134e-03 -1.35670120e-02]
 [-5.57938776e-03 -1.17278120e-02 -1.80155362e-02]
 [ 2.62872863e-02  2.18387150e-02  1.75422656e-02]
 ...
 [-1.66845106e-02 -1.46760717e-03  7.74981202e-03]
 [-1.35958389e-02  4.98007994e-04  1.33259137e-02]
 [-8.35297938e-03  9.58013972e-04  1.37736573e-02]]]

[[[ -1.89336820e-04  2.15956226e-03 -1.29409199e-02]
 [ 1.05364116e-03  1.42470653e-03 -1.08300341e-02]
 [-7.31054009e-03 -1.02627968e-02 -1.21482890e-02]
 ...
 [ 5.99134095e-03 -2.31819912e-03 -1.19899601e-02]
 [ 2.26051974e-02  5.49148276e-03 -4.79936707e-02]
 [-5.50565265e-02  8.48571807e-03  4.83756118e-02]]]

[[[ 5.88181694e-03  1.21820831e-02 -1.33686309e-03]
 [-2.51067082e-03  3.90009488e-03 -2.77502414e-02]
 [ 1.08055535e-02  1.44366096e-02 -1.94734115e-02]
 ...
 [-8.28897144e-03 -1.13751001e-03  1.00841228e-02]
 [-1.93717226e-03  1.84418399e-03  2.51228289e-02]
 [-1.28651645e-02  2.08537551e-03  2.26629724e-02]]]

[[[ -1.21103392e-02  2.82049641e-03 -7.01987943e-03]
 [-1.48572313e-02 -1.81859970e-03 -3.89187752e-03]
 [-2.36876355e-02 -1.25180316e-02 -4.94815002e-03]
 ...
 [ 8.64249033e-03  9.03636970e-03 -6.73153627e-03]
 [ 9.04559460e-03 -7.66010982e-03 -2.68219028e-02]
 [-5.63880248e-02  1.08878871e-02  5.37077389e-02]]]

[[[ -1.87519177e-02 -6.14998176e-03  1.09138137e-04]
 [-1.30261636e-02  9.98852554e-05 -1.52800918e-02]
 [ 4.53956851e-05  1.17247077e-02 -7.32678342e-03]
 ...
 [ 4.23331431e-05  3.68357774e-03  9.36336128e-03]
 [ 3.35954155e-03  2.29107060e-03  2.84650656e-02]
 [-1.40673457e-02  9.64362531e-04  2.35157972e-02]]]

```

- From the above two figures, D1 and D2; we can realize that there is a small difference between them.

Gaussian filter

```
filename = "images/jpeg/pink.jpg"
image = io.imread(filename)
image = (image/255).astype(float) # (0/255, 255/255) = (0, 1)
```

```
from scipy.ndimage import gaussian_filter
```

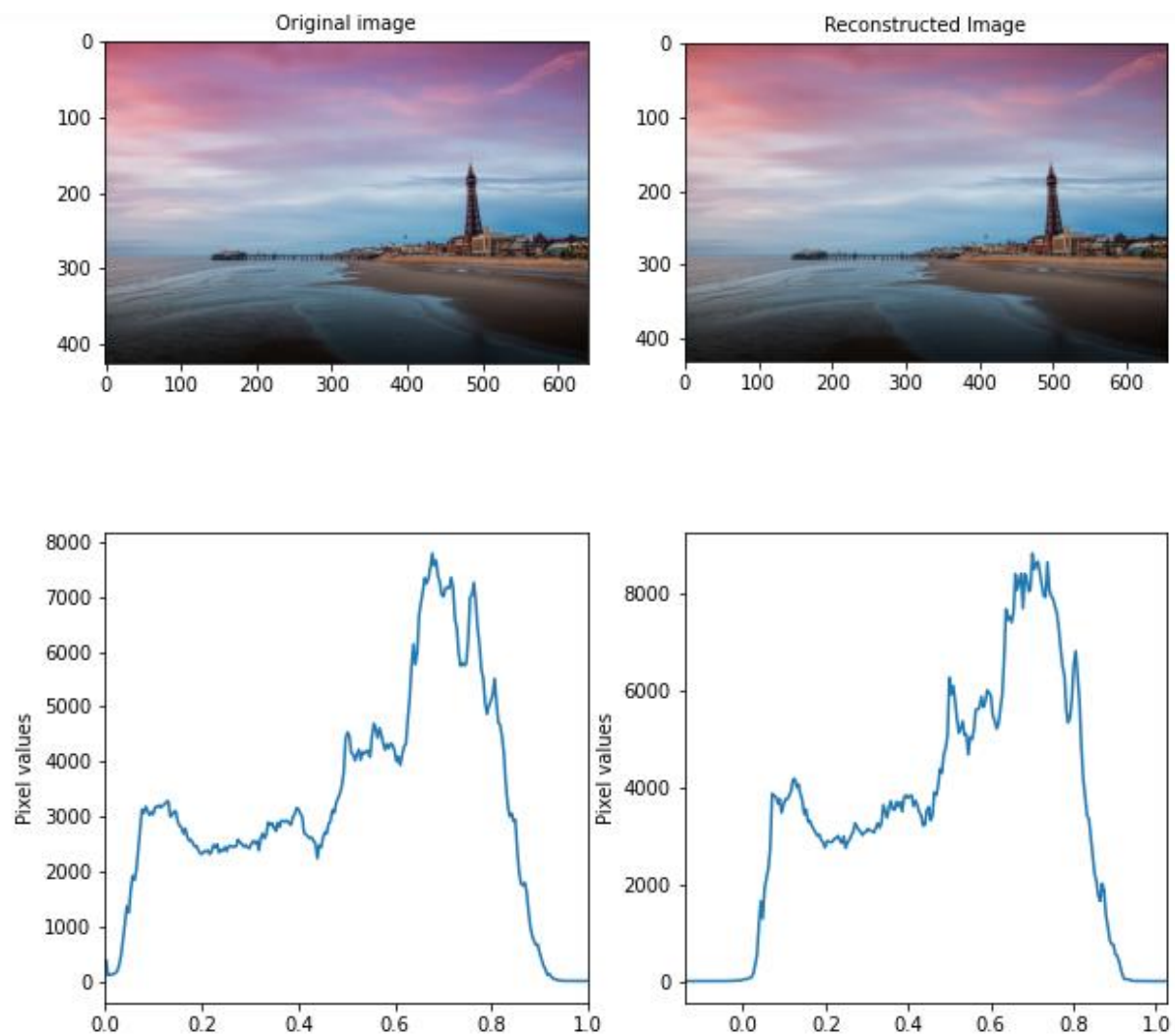
```
plt.rcParams['figure.figsize'] = [10,10]
plt.rc('xtick', labels=10)
plt.rc('ytick', labels=10)
```

```
gaussian_image = gaussian_filter(image, sigma=1) # Applying gaussian filter
y, i, q, shape = JPEGCompression(image)
Reconstructed = JPEGDecompression(y, i, q, shape)
Y, I, Q, Shape = JPEGCompression(gaussian_image)
Reconstructed_filtered = JPEGDecompression(Y, I, Q, Shape)
```

```
plt.rcParams['figure.figsize'] = [10,10]
plt.rc('xtick', labels=10)
plt.rc('ytick', labels=10)
```

```
plt.subplot(2,2,1), plt.imshow(image), plt.title("Original image", fontsize = 10), plot_histogram(image,2,2,3)
plt.subplot(2,2,2), plt.imshow(Reconstructed), plt.title("Reconstructed Image", fontsize = 10), plot_histogram(Reconstructed,2,2,4)
plt.show()
```

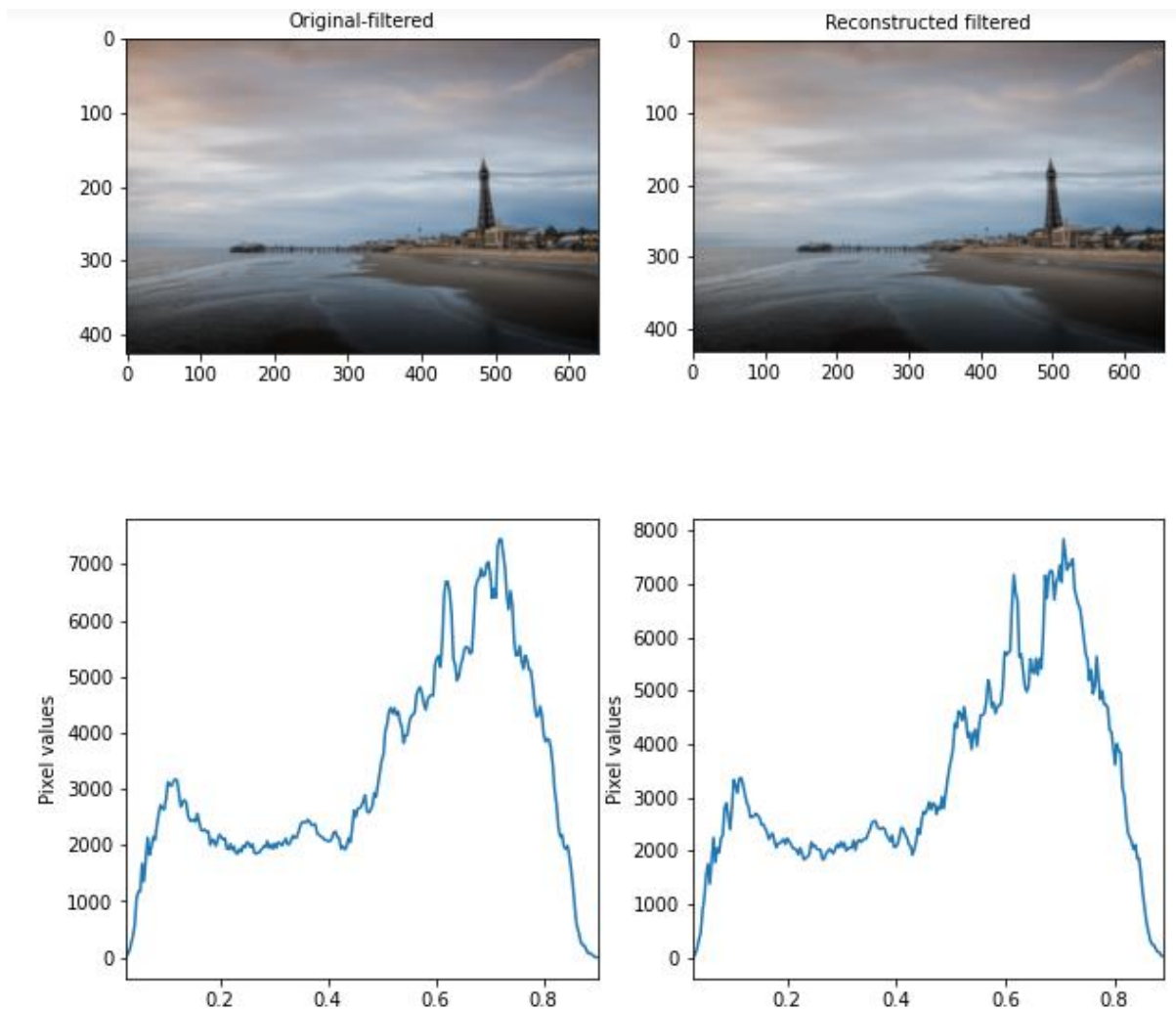
Original and reconstructed image, the result is shown below;




```
plt.rcParams['figure.figsize'] = [10,10]
plt.rc('xtick', labels=10)
plt.rc('ytick', labels=10)

plt.subplot(2,2,1), plt.imshow(gaussian_image), plt.title("Original-filtered", fontsize = 10), plot_histogram(gaussian_image,2,2,3)
plt.subplot(2,2,2), plt.imshow(Reconstructed_filtered), plt.title("Reconstructed filtered", fontsize = 10), plot_histogram(Reconstructed_filtered,2,2,4)
plt.show()
```

Applied the gaussian filter on the original and reconstructed image and got the following as an output.



Let's look at;

- (Original image - Reconstructed image)
- (Original filtered image – reconstructed filtered image)

NB: We need to resize the original filtered image to the same shape of the original image; the same thing also applies to the second part which is resizing the reconstructed filtered image to the size of reconstructed image.

```

Reconstructed = resize(Reconstructed,(image.shape[0], image.shape[1]),mode='constant')
Reconstructed_filtered = resize(Reconstructed_filtered,(gaussian_image.shape[0], gaussian_image.shape[1]),mode='constant')
D3 = image - Reconstructed           # original - reconstructed
D4 = gaussian_image - Reconstructed_filtered # original filtered - reconstructed filtered

print("*****")
print("D3 Min = ", np.amin(D3), "\nD3 Max = ", np.amax(D3))
print("*****")
print("D4 Min = ", np.amin(D4), "\nD4 Max = ", np.amax(D4))
print("*****\n\n")

print(D3, "\n\n")
#print(D4)

```

```

*****
D3 Min = -0.49371855048044677
D3 Max = 0.5192696462058362
*****
D4 Min = -0.11216184838376808
D4 Max = 0.09557066290480964
*****

```

D3

```

[[[ 4.69339702e-03 -1.01738730e-03 5.49056029e-02]
 [ 3.81827363e-03 -1.89251068e-03 5.40304795e-02]
 [ 6.14215169e-03 4.31367380e-04 5.63543576e-02]
 ...
 [ 7.51039767e-04 -2.36350961e-03 4.13943189e-02]
 [ 1.54988505e-03 -1.56466433e-03 4.21931642e-02]
 [ 1.98744674e-03 -1.12710263e-03 4.26307258e-02]]

[[[ 7.08575512e-03 1.37497081e-03 5.72979610e-02]
 [ 6.21063173e-03 4.99847420e-04 5.64228376e-02]
 [ 4.61294117e-03 -1.09784314e-03 5.48251471e-02]
 ...
 [ -3.99252821e-03 7.02503471e-04 4.05013716e-02]
 [ -3.19368293e-03 1.50134875e-03 4.13002169e-02]
 [ -2.75612124e-03 1.93891045e-03 4.17377786e-02]]

[[[ 4.95676300e-03 -7.54021311e-04 5.51689689e-02]
 [ 4.08163961e-03 -1.62914470e-03 5.42938455e-02]
 [ 6.40551768e-03 6.94733365e-04 5.66177236e-02]
 ...
 [ -5.97136395e-05 2.75840818e-03 4.04658317e-02]
 [ -3.18243699e-03 -3.64315163e-04 3.73431083e-02]
 [ -2.74487529e-03 7.32465308e-05 3.78067000e-02]]

```



D4

```

[[[ 1.12026807e-03 -5.37983406e-04 1.55387580e-03]
 [ 9.79052906e-04 -6.79198537e-04 1.41266067e-03]
 [ 9.30211658e-04 -7.28037629e-04 1.36382215e-03]
 ...
 [ -8.14917661e-04 9.85615982e-05 4.80196517e-03]
 [ -3.11187083e-04 6.02292176e-04 5.30569574e-03]
 [ 5.46718984e-05 9.68151158e-04 5.67155473e-03]]

[[[ 1.95416484e-03 2.95913396e-04 2.38777261e-03]
 [ 1.76419094e-03 1.05941624e-04 2.19780140e-03]
 [ 1.63168947e-03 -2.64988161e-05 2.06537719e-03]
 ...
 [ -1.56387593e-03 -7.29368279e-05 4.77720572e-03]
 [ -1.34836289e-03 1.42576215e-04 4.99271877e-03]
 [ -1.16021540e-03 3.30723705e-04 5.18086626e-03]]

[[[ 2.45786802e-03 7.99618734e-04 2.89147852e-03]
 [ 2.50498364e-03 8.46795361e-04 2.93867137e-03]
 [ 2.52518453e-03 8.67968974e-04 2.96010364e-03]
 ...
 [ -2.39823084e-04 -5.78048020e-04 2.97307153e-03]
 [ -3.83677272e-04 -7.21902208e-04 2.82921735e-03]
 [ -3.79352026e-04 -7.17576962e-04 2.83354259e-03]]

[[[ -6.76168630e-04 -1.47902710e-03 -1.69281423e-03]
 [ -1.84822139e-03 -2.22473827e-03 -2.32515576e-03]
 [ -1.17052127e-03 -1.31724248e-03 -1.35655439e-03]
 ...
 [ -3.45662857e-04 1.48351045e-03 6.89877295e-03]
 [ -6.11515116e-05 1.30662828e-03 6.26892714e-03]
 [ -1.10704834e-03 -2.29170886e-04 4.34336370e-03]]

[[[ -9.34346238e-04 -2.01481855e-03 -2.30212985e-03]
 [ -1.20034293e-03 -1.70687613e-03 -1.84156974e-03]
 [ -8.38155670e-05 -2.80982235e-04 -3.33411353e-04]
 ...
 [ 1.34193934e-04 1.81290068e-03 7.88265648e-03]
 [ -1.55380471e-03 8.73974608e-05 6.07695930e-03]
 [ -3.02061687e-03 -1.50131140e-03 4.39969287e-03]]

[[[ -4.96468303e-03 -6.10696611e-03 -6.41071367e-03]
 [ -2.68172253e-03 -3.21723306e-03 -3.35963210e-03]
 [ 4.98073627e-07 -2.07947940e-04 -2.63376380e-04]
 ...
 [ -4.63761679e-04 1.96920915e-03 8.66223602e-03]
 [ -4.22430855e-03 -1.82621058e-03 4.84921863e-03]
 [ -6.82257400e-03 -4.48882194e-03 2.15898927e-03]]]

```



Remark: The difference for the reconstructed images is much less than the original images.

(Ref: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.gaussian_filter.html)

Low-frequency image

```

filename = "images/Frequency/snow.jpg"
low = io.imread(filename)
low = (low/255).astype(float)
low = resize(low,(128,128),mode='constant')

```

```

plt.rcParams['figure.figsize'] = [10,10]
plt.rc('xtick', labels=10)
plt.rc('ytick', labels=10)

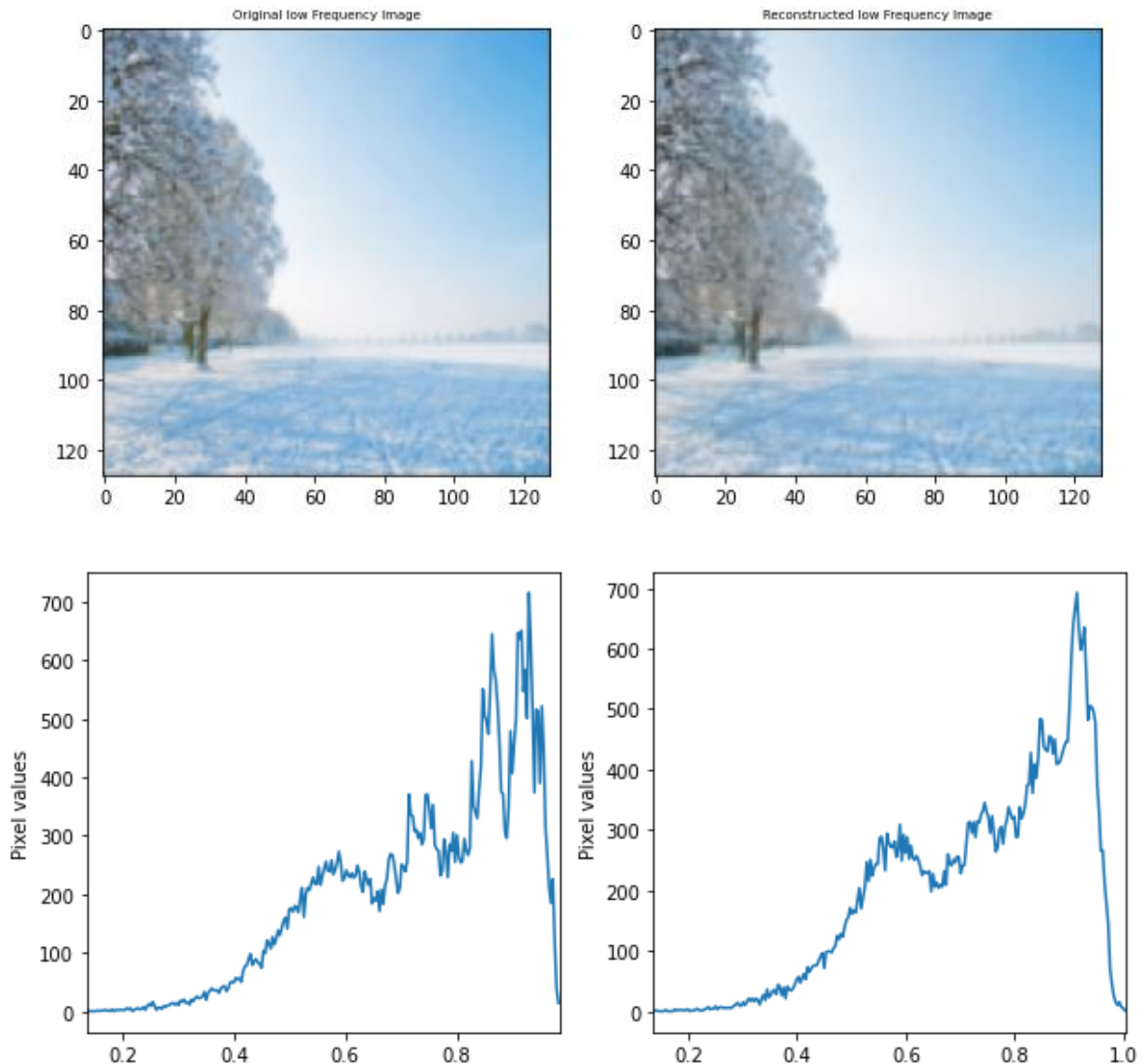
Y.I.Q.Shape = JPEGCompression(low)
newlow = JPEGDecompression(Y.I.Q.Shape)

plt.subplot(2,2,1), plt.imshow(low), plt.title('Original low Frequency Image', fontsize = 7), plot_histogram(low,2,2,3)
plt.subplot(2,2,2), plt.imshow(newlow), plt.title('Reconstructed low Frequency Image', fontsize = 7), plot_histogram(newlow,2,2,4)
plt.show()

```


Original shape: (128, 128, 3)
 Resized shape: (128, 128, 3)
 Shape of I after down-sampled: (64, 64)
 Shape of Q after down-sampled: (64, 64)
 Shape: (128, 128, 3)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



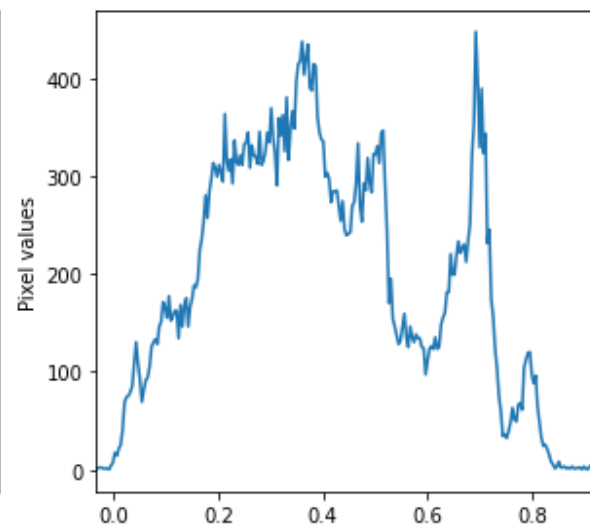
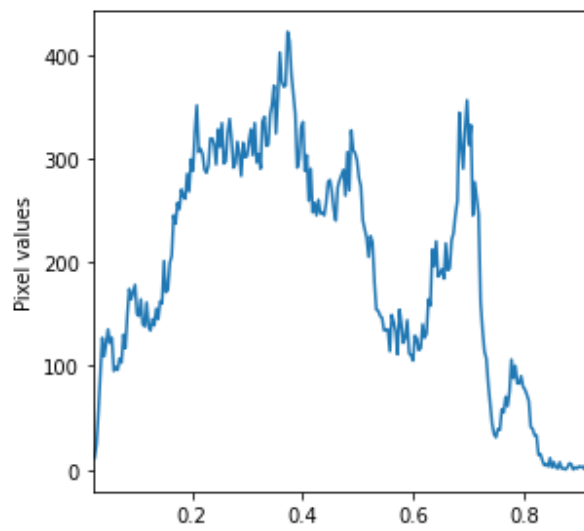
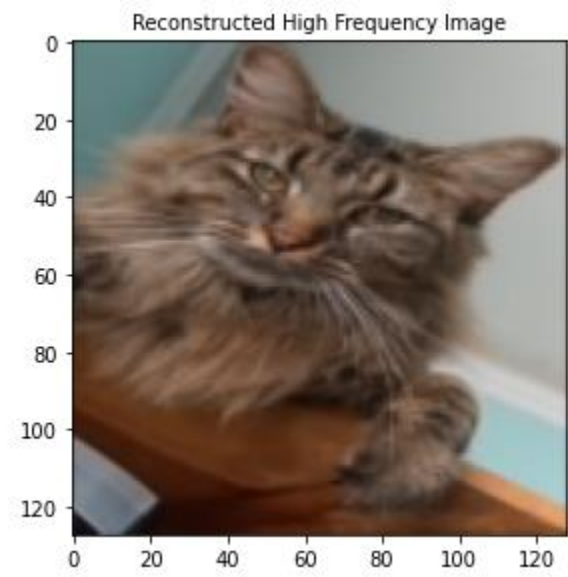
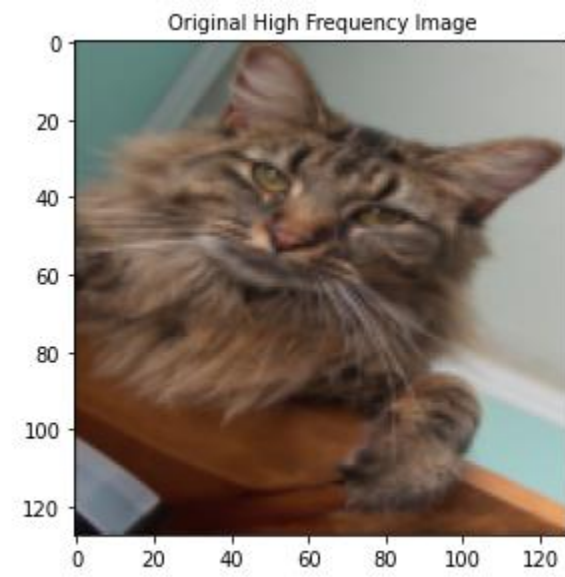
High-frequency image

```
filename = "images/PhaseMag/cat.jpg"
high = io.imread(filename)
high = (high/255).astype(float)
high = resize(high,(128,128),mode='constant')
```

```
plt.rcParams['figure.figsize'] = [10,10]
plt.rc('xtick', labels=10)
plt.rc('ytick', labels=10)
```

```
Y,I,QShape = JPEGCompression(high)
newhigh = JPEGDecompression(Y,I,QShape)
```

```
plt.subplot(2,2,1), plt.imshow(high), plt.title("Original High Frequency Image", fontsize = 10), plot_histogram(high,2,2,3)
plt.subplot(2,2,2), plt.imshow(newhigh), plt.title("Reconstructed High Frequency Image", fontsize = 10), plot_histogram(newhigh,2,2,4)
plt.show()
```



Computing their differences

```
D5 = low-newlow
D6 = high-newhigh

print("*****")
print("D5 Min = ", np.amin(D5), "\nD5 Max = ", np.amax(D5))
print("*****")
print("D6 Min = ", np.amin(D6), "\nD6 Max = ", np.amax(D6))
print("*****\n\n")

#print(D5)
#print(D6)
```

```
*****
D5 Min = -0.14474111259507763
D5 Max = 0.1279036751051239
*****
D6 Min = -0.1456441780751575
D6 Max = 0.13223922249565778
*****
```

```
[[[-0.00293944 -0.01505712 -0.00080706]
 [-0.0091888 -0.01162605 0.05432563]
 [0.03107381 0.0207622 0.02851493]
 ...
 [-0.00451161 -0.00890368 -0.01512663]
 [0.00606938 -0.00319499 -0.01055556]
 [0.04422796 -0.0125517 -0.05068374]]

 [[-0.01019523 0.01791213 0.0692398]
 [-0.02844054 -0.00684189 0.00875609]
 [-0.0291224 -0.00971505 -0.01348883]
 ...
 [-0.02920734 0.01324619 0.04208485]
 [-0.01867792 0.01643805 0.04595431]
 [0.01390613 -0.00477623 -0.01178444]]

 [[-0.02065033 -0.00688034 0.00189005]
 [0.01489038 0.0098898 0.00226787]
 [0.00665701 0.02476758 0.0184355]
 ...
 [-0.03252094 0.0010276 0.0182606]
 [-0.01972612 0.00542855 0.02276719]
 [0.01209837 -0.01987079 -0.03658525]]

 [[-0.00347971 0.00122752 -0.00884239]
 [-0.10338902 -0.00952454 0.05168205]
 [-0.03022802 0.00540774 0.03262378]
 ...
 [0.00722686 0.01164528 0.02153384]
 [-0.03722335 0.00783705 0.04737348]
 [0.00214347 -0.00913158 -0.00687516]]

 [[0.01643109 0.00311489 -0.01655421]
 [0.02301741 0.01744215 0.00428724]
 [0.02192918 0.01091894 -0.00719671]
 ...
 [-0.03381422 -0.01487267 -0.00104899]
 [0.00786415 0.02367743 0.02767678]
 [0.01936001 -0.00510292 -0.02250132]]

 [[0.00363524 -0.01439487 -0.03215913]
 [0.0329338 0.00595655 -0.02302138]
 [-0.0053355 -0.00686102 -0.02043445]
 ...
 [0.04997102 0.0008406 -0.0406833]
 [0.03304916 -0.01312211 -0.04962978]
 [0.06785121 -0.01985859 -0.08329769]]]
```

```
[[[0.00772096 -0.01232292 -0.02420842]
 [-0.00218483 -0.0095638 -0.02387374]
 [-0.00281406 -0.01275206 -0.02937167]
 ...
 [-0.00850173 -0.01066906 -0.01427931]
 [0.0085669 0.00386091 0.00021621]
 [-0.00325516 -0.00646725 -0.00660123]]

 [[0.00529684 -0.00178534 -0.01321979]
 [-0.00015132 0.01080594 -0.00876032]
 [-0.00528077 0.00787303 -0.01459635]
 ...
 [0.00200858 -0.00188471 -0.00792793]
 [0.01679401 0.01240988 0.00596166]
 [-0.00395372 -0.00447184 -0.01052878]]

 [[-0.00343663 -0.00866353 -0.02154961]
 [-0.00384493 0.00098419 -0.01565642]
 [-0.01235496 -0.00325068 -0.02159428]
 ...
 [0.00016017 -0.00362809 -0.00723212]
 [0.01064505 0.00761825 0.00413586]
 [-0.00791483 -0.00530598 -0.00623043]]

 [[0.00668748 0.01291502 -0.00344047]
 [-0.01238965 -0.00784029 -0.02378861]
 [0.0112058 0.00276337 -0.01107454]
 ...
 [-0.0268198 0.01357941 0.03962313]
 [0.00711757 0.03715967 0.0471349]
 [-0.04471374 0.00431362 0.02075726]]

 [[-0.00982952 0.0017432 0.00407265]
 [-0.00860073 0.00325919 0.0053492]
 [-0.01078473 -0.00513496 -0.01109097]
 ...
 [-0.00089216 -0.01230814 -0.01817969]
 [0.019695 -0.02236787 -0.03592472]
 [0.03387225 0.00759059 -0.00531095]]

 [[-0.01030838 -0.00059896 0.01070918]
 [-0.01116588 0.00315532 0.00974555]
 [0.00431816 0.01063103 0.00344639]
 ...
 [-0.01501318 0.00296579 0.00877053]
 [-0.0072685 -0.0063331 0.01048819]
 [-0.01027771 0.00116937 0.03775661]]]
```