ECOLE CENTRALE DE NANTES

Foundation Masters 2020-21

PROJECT REPORT ON IMAGE COMPRESSION BASED ON JPEG USING PYTHON

Presented by:

Mellacheruvu Prithvi Bharadwaj

Evaluator: Ms. Diana Mateus, Professor, Ecole Centrale de Nantes

DECLARATION

This report has been prepared on the basis of my own work. Data and information sourced from other published and unpublished have been acknowledged.

Word count: 1217

Student name: Mellacheruvu Prithvi Bharadwaj

Date of submission: 01-05-2021

CONTENTS:

JPEG compression implementation in Python

The whole process is divided into 4 functions and the 2 major operations –

1. Image Compression:
    a) YIQ_con_IQ_downsam(): This function takes input the image as an array ranged from 0 to 1. It first converts the RGB image into YIQ color space. This is followed by down sampling the chroma channels i.e. I, Q in half using the resize function from skimage package.

    b) DCT_8x8_maker(): This function takes input the 3 channels of the image and fragments them into 8x8 blocks. This is followed by centering the Y values about 0 by subtracting 127 and then apply DCT to them.

    c) div_qt(): This function takes input the 8x8 blocks from the previous function in the form of dictionaries and divides these blocks by their respective quantization tables based on the type of channel. This marks the successful completion of the image compression process.

2. Image Decompression:
    d) reconstruct(): The decompression of the compressed form of the image is done in this function and the final output image is displayed. It takes the dictionaries which are obtained from the previous function i.e. div_qt() and also the YIQ channels of the original image. The reconstruction begins by first multiplying the dictionaries by their respective quantization tables followed by iDCT for these channels and then stitching the 8x8 blocks into one single entity for each channel. This is followed by adding 127 to the Y channel since we subtracted it before DCT, and then resize these channels into the original dimensions of the image and then combining all the 3 channels into one single final array of uint8 type and then display it.

The below image is a snippet of the code which calls the above defined functions in sequence in the final cell of the document.

```
Y, I, Q = YIQ_con_IQ_downsam(im)  # Converts RGB to YIQ  and downsamples the IQ channels by the resize function

DicY, DicI, DicQ = DCT_8x8_maker(Y, I, Q)  # Applies DCT & divides the image into 8x8 blocks

upDicY, upDicI, upDicQ = div_qt(DicY, DicI, DicQ)  # divides the image by the quantisation tables

reconstruct(upDicY, Y, upDicI, I, upDicQ, Q)  # reconsructs the image by multiplying by quantisation tables,
                                              # followed by stitching the whole image back and then apply iDCT,
                                              # followed by resizing the image into the original dimensions of the image
                                              # and then prints the input and final images.
```

Functions' Code Explained

a) YIQ_con_IQ_downsam(): The conversion from RGB to YIQ happens in the following part.

```
ConMat = [[0.299, 0.587, 0.114], [0.5959, -0.2746, -0.3213], [0.2115, -0.5227, 0.3112]]

YIQ = np.zeros((np.shape(R)[0], np.shape(R)[1], 3))
for i in range(np.shape(R)[0]):
    for j in range(np.shape(R)[1]):
        YIQ[i, j] = np.matmul(ConMat, [R[i, j], G[i, j], B[i, j]])
```

The IQ channels down sampling and Y resizing to facilitate 8x8 blocks is below.

```
Y = YIQ[:,:,0]
if np.shape(Y)[0]%8 != 0 or np.shape(Y)[1]%8 != 0:
    Y = resize(YIQ[:,:,0],((np.shape(Y)[0]-(np.shape(Y)[0]%8)), (np.shape(Y)[1]-(np.shape(Y)[1]%8))), mode='constant')

if np.shape(Y)[0]%16 == 0 and np.shape(Y)[1]%16 == 0:
    I = resize(YIQ[:,:,1],(np.shape(Y)[0]/2, np.shape(Y)[1]/2),mode='constant')
    Q = resize(YIQ[:,:,2],(np.shape(Y)[0]/2, np.shape(Y)[1]/2),mode='constant')
else:
    I = resize(YIQ[:,:,1],((np.shape(Y)[0]-(np.shape(Y)[0]%16))/2, (np.shape(Y)[1]-(np.shape(Y)[1]%16))/2), mode='constant')
    Q = resize(YIQ[:,:,2],((np.shape(Y)[0]-(np.shape(Y)[0]%16))/2, (np.shape(Y)[1]-(np.shape(Y)[1]%16))/2), mode='constant')
return(Y, I, Q)
```

b) DCT_8x8_maker(): The fragmentation of image into 8x8 blocks followed by DCT can be seen below. Y channel is centered about 0.

```
Y = Y - 127

DicY = {}
global xY
global yY
xY = int(np.shape(Y)[0]/8)
yY = int(np.shape(Y)[1]/8)

for i in range(xY):
    for j in range(yY):
        DicY.update({(i, j) : dct_2d(Y[i*8:(i+1)*8 , j*8:(j+1)*8])})
```

c) div_qt(): The quantization tables are loaded below.

```
cutie_lum = quantization_table('lum')       # for Y & BW - Black & White ;)
cutie_chrom = quantization_table('chrom')   # for IQ
```

The logic for the division of an image channel by these quantization tables is below.

```
upDicY = {}
for i in range(xY):
    for j in range(yY):
        upDicY.update({(i, j) : np.round(DicY[(i, j)]/cutie_lum)})
```

d) reconstruct(): The decompression followed by reconstruction of the image all happens here. It starts by first multiplying the 8x8 blocks from the compressed state by their respective quantization tables and perform an iDCT as shown below.

```
DupDicY = {}
for i in range(xY):
    for j in range(yY):
        DupDicY.update({(i, j) : idct_2d(upDicY[(i, j)]*cutie_lum)})
```

These 8x8 blocks are then stitched back into a single array as below.

```
x = np.shape(Y)[0]
y = np.shape(Y)[1]
upY = np.zeros((x, y))
for i in range(x):
    for j in range(y):
        upY[i, j] = DupDicY[(int(i/8), int(j/8))][i%8, j%8]
```

Now, these iDCT values for the channels are then resized back to the dimensions of the original input image as below.

```
iY = resize(upY+127,(np.shape(R)[0], np.shape(R)[1]), mode='constant', preserve_range=True)
iI = resize(upI,(np.shape(R)[0], np.shape(R)[1]), mode='constant', preserve_range=True)
iQ = resize(upQ,(np.shape(R)[0], np.shape(R)[1]), mode='constant', preserve_range=True)
```

Finally, the above YIQ channels are converted back to RGB and combined into a single array named “final” which is the final decompressed RGB form of the image.

```
final = np.zeros((np.shape(R)[0], np.shape(R)[1], 3))
for i in range(np.shape(R)[0]):
    for j in range(np.shape(R)[1]):
        final[i, j] = np.matmul(np.linalg.inv(ConMat), [iY[i, j], iI[i, j], iQ[i, j]])
```

```
final = final.astype(np.uint8)
```

The final displaying of the input and output images along with their respective YIQ channels is done as shown below which is extended to other plots.

```
plt.subplot(4,2,1)
plt.imshow(arr)
plt.title('Input Image')
plt.subplot(4,2,2)
plt.imshow(final)
plt.title('Final Image')
```

To quantify the compression of images, I have performed two operations where one involves checking for the space occupied by saving input and final image arrays in PNG format and the other is by checking the space occupied by the image arrays.

1. Both the input and final image arrays are saved in PNG format to compare memory occupied in same conditions as shown below.

```
data = imagesave.fromarray(final)
data.save("mem_out.png")
data = imagesave.fromarray(arr)
data.save("mem_in.png")
```

Now these image files are checked for their size in the main Jupyter notebook cell and the difference is printed as shown below. The saved files are again deleted.

```
if DicI != 0:
    init_size = os.path.getsize("./mem_in.png")
    fin_size = os.path.getsize("./mem_out.png")
    os.remove("./mem_out.png")
    os.remove("./mem_in.png")
    saved_size = (init_size - fin_size)/1000
    print("Memory difference of input and output images after saving them both in .png (in KB) : ", saved_size)
```

2. The memory required by the image arrays of input and final images is calculated and difference displayed as below.

```
init_size = arr.nbytes/1000
fin_size = final.nbytes/1000
print("Memory difference of input and final image arrays (in KB) : ", init_size - fin_size)
```

e) The main Jupyter notebook cell which takes in all the images in the directory specified in the variable “IMDIR” and passes these images to all the above functions sequentially in order to realize the JPEG compression is as follows.

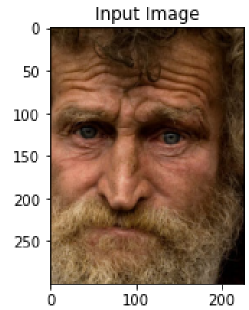
```
for root, dirnames, filenames in os.walk(IMDIR):
    for filename in filenames:
        file = os.path.join(root, filename)

        #filter only image files with the following format
        if file.endswith(('.png', '.jpg', '.jpeg','.JPG', '.tif', '.gif')):
            im = io.imread(file)
            filend = file[-5:]
            print("Image name : ", filename)

            Y, I, Q = YIQ_con_IQ_downsam(im)  # Converts RGB to YIQ  and downsamples the IQ channels by the resize function

            DicY, DicI, DicQ = DCT_8x8_maker(Y, I, Q)  # Applies DCT & divides the image into 8x8 blocks

            upDicY, upDicI, upDicQ = div_qt(DicY, DicI, DicQ)  # divides the image by the quantisation tables

            reconstruct(upDicY, Y, upDicI, I, upDicQ, Q, filend)  # reconsructs the image by multiplying by quantisation tables,
                                                   # followed by stitching the whole image back and then apply iDCT,
                                                   # followed by resizing the image into the original dimensions of the image
                                                   # and then prints the input and final images.
            if DicI != 0:
                init_size = os.path.getsize("./mem_in.png")
                fin_size = os.path.getsize("./mem_out.png")
                os.remove("./mem_out.png")
                os.remove("./mem_in.png")
                saved_size = (init_size - fin_size)/1000
                print("Memory difference of input and output images after saving them both in .png (in KB) : ", saved_size)

            fig=plt.figure()
            plt.show()
            print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")
```

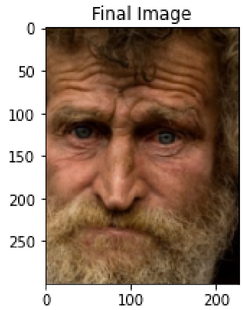
OUTPUT

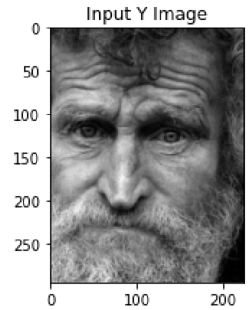
The following images are the output for a few samples:

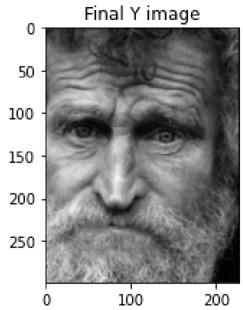
1.

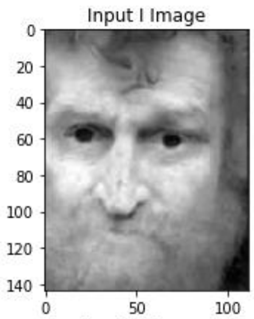
```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Image name :  face2.jpg
Memory difference of input and final image arrays (in KB) :  0.0
Memory difference of input and output images after saving them both in .png (in KB) :  16.445
```

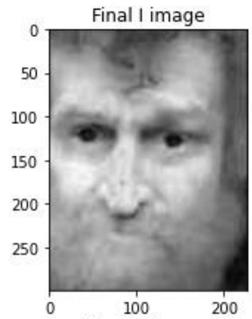


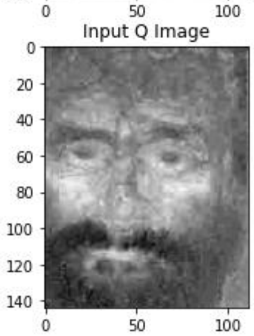


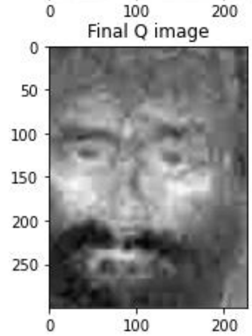












2.

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Image name :  letter.png
Memory difference of input and final image arrays (in KB) :  0.0
```

3. G

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Image name :  zebra-wall.png
Memory difference of input and final image arrays (in KB) :  0.0
Memory difference of input and output images after saving them both in .png (in KB) :  47.634
```

RESULT: All kinds of images given are successfully compressed and decompressed based on the JPEG format and the loss in data due to this type of compression is very clearly visible in the final output images.

NOTABLE CHALLENGES

These are mostly programming challenges as the theoretical aspects are well explained in the resources provided on this topic.

1. Using the same functions defined in the program for both grayscale images and 3-channelled RGB images.

   Solution: There is an if condition in the beginning of each function which checks for this and then the return values of the function are either 0's for other channels if grayscale or the right data is passed. Only the second function takes input the YIQ channels in the form of list and here the if condition is checked as below. [Reference – 6]

```
if not isinstance(I, np.ndarray):
```

   Whereas, for the rest functions they are dictionaries and it happens as below.

```
if DicI == 0:
```
&
```
if upDicI == 0:
```

2. To quantify the extent of compression in the images due to JPEG process

   Solution: Two methods are used where one method compares the memory occupied by the input & final image arrays, while the second method saves the 2 images in PNG format and compares their file sizes. The results are not uniform to all images.

3. Improper ranges for data types resulting in different colored images.

   Solution: Clip the final image RGB's into [0,255] and define the final image array type as uint8 before displaying.

4. Creating a sequence of functions from a crude linear implementation

   Solution: The whole process was first implemented in a linear crude fashion which I then had to segregate into functions and because of that had to take care of variables and their scopes, etc.

Unsolved Problems

1. The I & Q channels of the final output are displayed blank for some images even though the output image is not distorted. The reason for these values being 0 should be further investigated
2. A proper method to quantify the extent of compression in terms of memory saved, although I have implemented 2 methods, the results are not satisfactory.

CONCLUSION

The whole process of JPEG compression is clearly understood and the role of DCT and its workings are explored in a practical and enriching way through this project.

The loss in data due to this type of compression is clearly visible in the final output images. The level of abstraction and the powerful tools offered by the python environment for coding in general and image processing in speific has been understood and greatly appreciated.

REFERENCES

1. https://en.wikipedia.org/wiki/Discrete_cosine_transform
2. https://www.youtube.com/watch?v=Q2aEzeMDHMA
3. https://hippocampus.ec-nantes.fr/course/view.php?id=1522
4. https://numpy.org/devdocs/
5. https://www.tutorialspoint.com/numpy/index.htm
6. https://stackoverflow.com/questions/16807011/python-how-to-identify-if-a-variable-is-an-array-or-a-scalar
7. https://www.tutorialspoint.com/matplotlib/index.htm
8. https://www.youtube.com/watch?v=spUNpyF58BY

All the images shown in this report are screenshots taken from my personal computer during the implementation of this project.