Adam Boyer

CS-6323-DF1

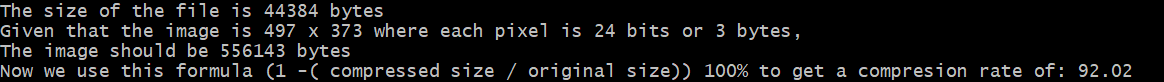
Assignment 2

For my assignment, I chose to use the Python programming language, with the pillow library. My full program will also be included in my deliverables, along with this report.

To run the attached program please follow these steps:

1. Place assignment2.py and Lena.jpg in the same directory
2. Run the following command in terminal or cmd in the same directory, to install dependency
   1. pip install Pillow
3. While still in the same directory run the following command to run the program
   1. python assignment2.py

Questions:

1. Print the dimensions of the image. Assuming that RGB color space (24 bits) has been used in this image, what is the compression rate?
   1. Using the Pillow library I was able to print the dimensions of the image:
      1. *from* PIL *import* Image
      2. *import* os
      3. img = Image.open("Lena.jpg")
      4. width, height = img.size
      5. print("The dimensions of the image are: " + str(width) + " x " + str(height))
   2. The output was as follows:
      1. 
   3. To calculate the compression rate, I had to use the following formula
      1. First calculate original size
         1. Width \* height \* 3 (for 24 bits or 3 bytes)
      2. Then I found the compressed size
         1. *# Getting the image size in bytes*
         2. *with* open("Lena.jpg", "rb") *as* file:
         3. file\_size = os.fstat(file.fileno()).st\_size
      3. Finally using the formula (1 - (compressed size / original size)) x 100% I was able to calculate the compression rate
         1. *# calculate compression rate*
         2. compression\_rate = (1 - (file\_size / original\_file\_size)) \* 100
   4. The final output was as follows
   5. 
2. RGB color model combines color and light intensity. To have a brighter image, we can use YCbCr color model. Convert the image into YCbCr model, increase the value of Y component, convert the image back to RGB. What happened?
   1. The first step was to convert the image to YCbCr model
      1. *# create a copy of the image and convert it to YCbCr*
      2. ycbcr\_img = img.convert("YCbCr")
   2. The next step was to get the Y, Cb, and Cr channels respectively. I did this by using the split method on the new image
      1. *# create a copy of the image and convert it to YCbCr*
      2. ycbcr\_img = img.convert("YCbCr")
   3. To scale the Y channel by a factor of two I used the .point() method and passed a lambda that multiplies each value by two
      1. *# scale the Y or brightness by 2*
      2. new\_y = y.point(lambda *x*: *x* \* 2)
   4. Next, I merged the new Y value and the old Cb and Cr values into yet another new image
      1. *# merge the new y channel with the old cb and cr channels to create a new image*
      2. modified\_ycbcr\_img = Image.merge("YCbCr", (new\_y, cb, cr))
   5. Finally, I converted the image back to RGB format and saved the new file as ‘modified\_Lena.jpg’
      1. *# convert back to RGB and save the image*
      2. new\_img = modified\_ycbcr\_img.convert("RGB")
      3. new\_img.save("modified\_Lena.jpg")
   6. The new image was much brighter than the original image.
      1. Original
         1. 
      2. Modified Y value:
         1. 
3. Assume you are interested in selecting areas having shades of Red. You may consider a specific range for Red component. Change the Cr values of these pixels to zero. Put your image back in RGB and display it. Show your results. Hint: Use YCbCr color space.
   1. The process for changing the Red value to zero, was very similar to changing the Y value. The only difference was I changed the Cr value, and passed a lambda to the .point() method that set each value of Cr to zero
   2. The code is as follows
      1. new\_cr = cr.point(lambda *x*: *x* - *x*)
      2. no\_red\_img = Image.merge("YCbCr", (y, cb, new\_cr))
      3. new\_no\_red\_img = no\_red\_img.convert("RGB")
      4. new\_no\_red\_img.save("no\_red\_Lena.jpg")
      5. new\_no\_red\_img.show()
   3. You will notice I was able to use the old Y and Cb values, and saved the file as ‘no\_red\_Lena.jpg’
   4. The new file looked very green without the Cr values
      1. 
4. In the jpeg compression method, Cb and Cr components are down-sampled. Apply the algorithm to Cb and Cr components. Then up-sample both components and reconstruct the image. Do you see any sensible quality change?
   1. The first step I took to down-sample the images was to use the .resize() method and divide both the width and height by 2 (rounding using // operator).
      1. down\_sampled\_cr = cr.resize((cr.width // 2, cr.height // 2), *resample*=Image.BICUBIC)
      2. down\_sampled\_cb = cb.resize((cb.width // 2, cb.height // 2), *resample*=Image.BICUBIC)
   2. Next I up-sampled by performing the same .resize() on the new down-sampled Cr and Cb channels and multiplying by 2.
      1. up\_sampled\_cr = down\_sampled\_cr.resize((down\_sampled\_cr.width \* 2, down\_sampled\_cr.height \* 2), *resample*=Image.BICUBIC)
      2. up\_sampled\_cb = down\_sampled\_cb.resize((down\_sampled\_cb.width \* 2, down\_sampled\_cb.height \* 2), *resample*=Image.BICUBIC)
   3. Finally, I used to the new up\_sampled Cb, Cr, and the original Y (resized a little to avoid errors) to merge into a new image, convert to RGB format, and save to a file named ‘cb\_cr\_down\_then\_up\_sampled\_Lena.jpg’.
      1. up\_sampled\_img = Image.merge("YCbCr", (y.resize((up\_sampled\_cr.width, up\_sampled\_cr.height), *resample*=Image.BICUBIC), up\_sampled\_cb, up\_sampled\_cr))
      2. up\_sampled\_img\_rgb = up\_sampled\_img.convert("RGB")
      3. up\_sampled\_img\_rgb.save("cb\_cr\_down\_then\_up\_sampled\_Lena.jpg")
   4. I observed that there was no noticeable change between the new image and the original image.
   5. cb\_cr\_down\_then\_up\_sampled\_Lena.jpg
      1. 
5. Repeat step 4, down-sampling all three components. How does the quality change?
   1. Now I repeated the down-sample and up-sample steps for Y channel and merged all modified samples to a new image, converted to RGB, and saved the image as ‘all\_downsampled\_then\_upsampled\_Lena.jpg’
      1. down\_sampled\_y = cb.resize((y.width // 2, y.height // 2), *resample*=Image.BICUBIC)
      2. up\_sampled\_y = down\_sampled\_y.resize((down\_sampled\_y.width \* 2, down\_sampled\_y.height \* 2), *resample*=Image.BICUBIC)
      3. final\_img = Image.merge("YCbCr", (up\_sampled\_y, up\_sampled\_cb, up\_sampled\_cr))
      4. final\_img\_rgb = final\_img.convert("RGB")
      5. final\_img\_rgb.save("all\_downsampled\_then\_upsampled\_Lena.jpg")
   2. After viewing the new file, It was apparent that it had changed. The new image looks almost like a negative of the original.
   3. all\_downsampled\_then\_upsampled\_Lena.jpg
      1. 