CompE565, Spring 2022

HW1: Basic Image Processing Operations

Prepared by

Abraham Carranza and Ryan Shimizu

E-mails: acarranza4510@sdsu.edu and rshimizu3229@sdsu.edu

RedID: 822338381 and 823053121

Electrical and Computer Engineering

San Diego State University

Introduction

Discuss the need for image and video coding from the point of storage and transmission bandwidth.

When we are storing data of an image, the naive approach would be to store the color values on a per-pixel basis. But as we include more colors, this becomes a problem in terms of memory management and storage since each image can contain thousands or even millions of pixels. This problem gets compounded when we have to send images to another device since transmission bandwidth is expensive. Hence, image and video coding can help alleviate some of these issues. Many algorithms have been developed but all of them aim to reduce *information redundancy*. The algorithms we will use in this assignment are 4:2:0 subsampling and upsampling via linear interpolation and row column replication.

Procedural Section

In this assignment, we completed 11 tasks to strengthen our understanding of basic image processing operations. We will discuss each task sequentially and in detail.

Task 1, we must read and display an image of a flooded house included in the assignment description. To set up the display of the image we use functions: *figure* and *subplot*. We will be using *figure* and *subplot* to create presentable layouts when displaying our results. We then must store our image into a variable using *imread*. We finish by displaying the image using *imshow*. The displayed result can be found in the section *Results*.

Task 2, we must display the red, green and blue bands of the image. So, we must create three copies of the stored image to easily reference each component after extraction. Afterwards, each copy will be appropriately manipulated to assign components of red, green, and blue appropriately. For example, for red, the green and blue components are set to zero. Similar to how we displayed the image in Task 1, we display the red, green, and blue components as subplots to compare with the original image. The displayed results can be found in section *Results*.

Task 3 and 4, we must convert the image into YCbCr color space and display the bands separately. Conveniently we can use a predefined function *rgb2ycbcr* to convert an image from the RGB color space to the YCbCr colorspace. We separate the Y, Cb, and Cr components using the *imsplit* function. We display the components using the same technique in the former tasks. The displayed results can be found in section *Results*.

Task 5, we must subsample the Cb and Cr bands using 4:2:0 and display both bands. According to lecture topic 1, “Subsampling does not mean simply make the pixel value zero, it is the process of removing the pixels by reducing the image size” [1]. Therefore, we must reduce the matrices. We do so, in 4:2:0, by storing every other pixel in a row and column from the Cr and Cb components into their respective subsampled matrices. We display them as before using the formerly described functions and their results can be found in section *Results*.

Task 6, we separate the tasks into Tasks 6.1 and 6.2.

Task 6.1, we must upsample the Cb and Cr bands using linear interpolation. Since we know the dimensions we are upsampling to, we can create empty matrices with dimensions equal to the subsampled dimensions multiplied by the scale factor, 2. We populate the matrices by storing each pixel, in Cb and Cr bands, in every other column and row. Then we proceed with the linear interpolation upsampling algorithm. For every even column and odd row in our matrices we store the average of the former and following pixel. We included an edge case to copy the former column in case we exceed our matrix bounds. Similarly, we must populate the even rows by performing the former procedure. The results of upsampling can be found in section *Results*.

Task 6.2, we must upsample the Cb and Cr bands using simple row and column replication. To do so, for every odd row, we must populate each even pixel with the pixel preceding it. Afterwards, we store every odd row in it’s following even row. The results can be found in section *Results*.

Task 7 and 8, we must convert the image into RGB format. Conveniently we can use *ycbrcr2rbg* function to convert our YCbCr color space image to the RGB color space. However, when converting we must recombine our Y, Cb, and Cr bands using the *cat* function. The reconstructed results and original images can be found in section *Results*.

Task 9, we are to comment on the visual quality of the reconstructed image for both upsampling cases in Tasks 6.1 and 6.2. The visual quality between the two reconstructed images is virtually identical from afar. Only minor differences can be seen in areas that are well defined, such as the Coke can in the bottom right floating on water. In the RCR image, there is a black outline around the E whereas this is not present in both the LI and image. This could be due to the fact that there is a black lettering ("diet"?) directly to the left of the E. Since RCR fills the missing pixels by sampling the pixel directly to the left, this makes sense. In the LI image, there is a slight discoloration or "hue" surrounding areas where the difference between pixels is large. Using the same example as before, the E in the LI image has a slight red tint to the white pixels surrounding it whereas this is not present in the original image. This also makes sense since LI fills in the missing pixels with an average of the pixels around it, leading to a "blur" effect on areas with high contrast.

Task 10, we must measure the MSE between the original and reconstructed images. We calculate the MSE using the following structured equation [1]:

$$\frac{1}{MN} \sum_{y=1}^{M} \sum_{x=1}^{N} \left[I\,(x,y) - I'\,(x,y) \right]^2$$

The calculated result is relatively small considering the size of our matrix: 506x704x3. Based on the image from Canvas, this number is 25.1865. This low number aligns with our previous observation that all the reconstructed images look roughly the same as the original image.

Task 11, we will discuss the compression ratio achieved by subsampling Cb and Cr components for the 4:2:0 approach. Using the 4:2:0 approach, we can compress our original image by a factor of essentially 4. This is because we scale down our Cb and Cr components by 2x2 and then recalculate those missing pixels by virtue of the other surrounding pixels that we didn't throw away. By "throwing away" some of the redundancy information, we can deploy either the Linear interpolation or Row column replication technique to reconstruct the compressed image back into its size. As noted by our observations in 9 and 10, our losses are very minimal considering how many bits we had actually thrown away in the process of compressing the image. Essentially despite throwing away 3/4 of the original bits, we are able to reconstruct our original image with impressive quality.

Results

As seen in Figure 1, we are able to separate our original image into our red green and blue bands.

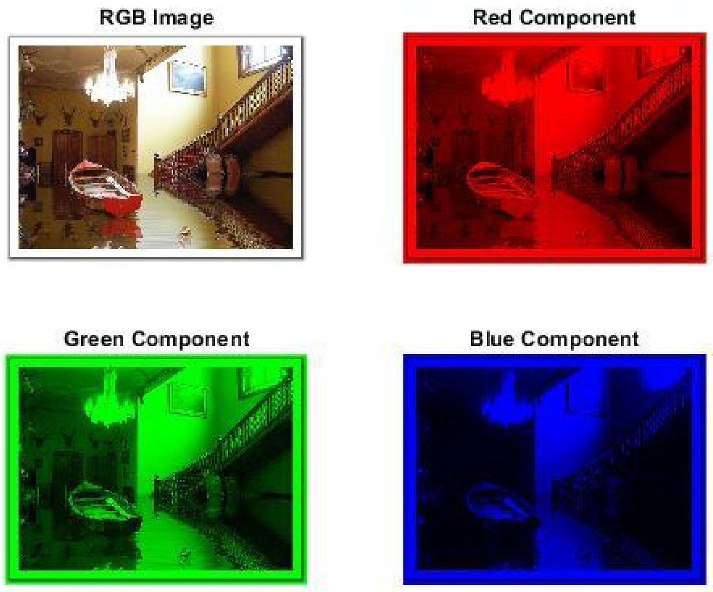

Figure 1: RGB Image decomposed into Red, Green, and Blue components

This is the traditional approach to encoding our image. However, this is not ideal in terms of storage space since every band is equally important in terms of information held. Thus, our options for eliminating redundancy are limited. We can resolve this by converting our image to Y-Cb-Cr as seen in Figure 2.

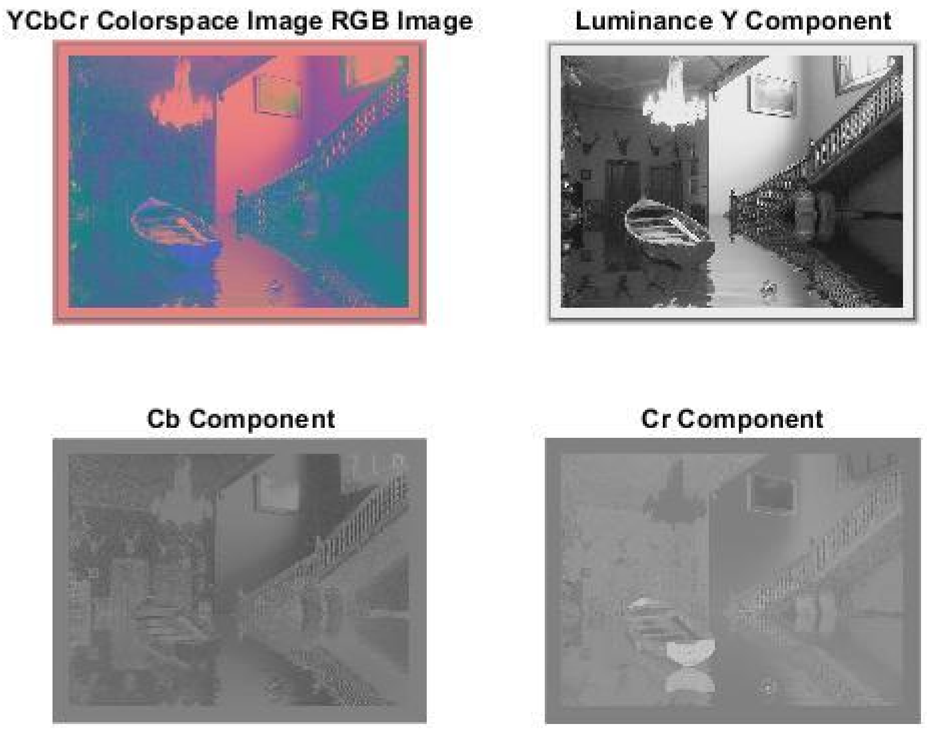

Figure 2: RGB Image converted into YCbCr components.

Since the human eye is more sensitive to changes in luminous intensity, this information is the most valuable in preserving the overall image quality (the Y band). The Cb and Cr bands, on the other hand, are less important in terms of the overall image quality but provide a lot of the smaller details that are not present in the Y band. Since these bands are less important for the overall image quality, we can control how many bits we want to allocate to these maps. This is where subsampling comes in. We can see in Figure 3 that we can reduce these pixels by sampling every other column and every other row, resulting in a 75% loss of our original Cb and Cr bands.

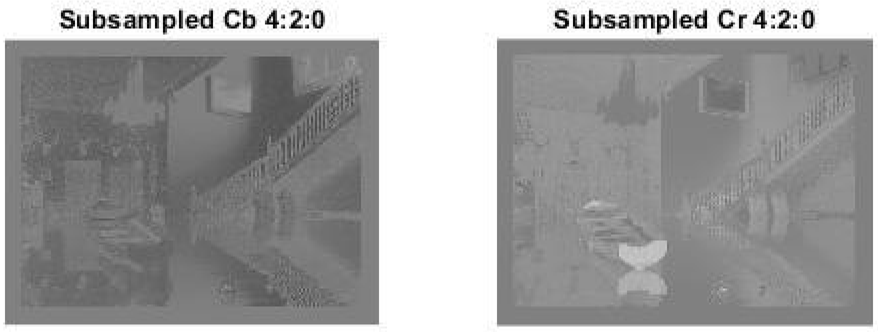

Figure 3: Subsampled results of the Cb and Cr components using 4:2:0 compression.

After we subsample these bands, we are ready to store this data and transmit. When we want to retrieve this data again, we can employ our upsampling techniques that can be seen in Figure 4.

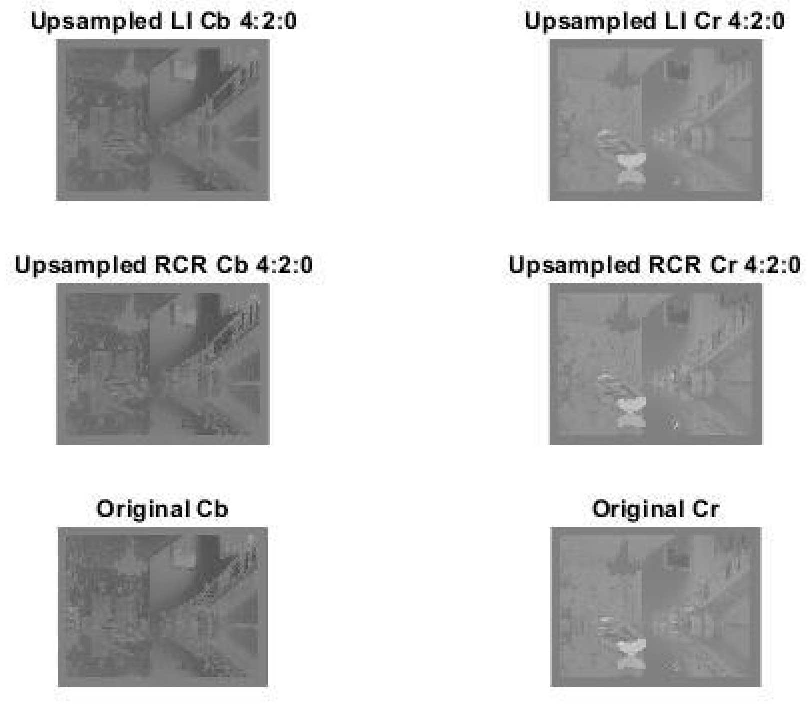

Figure 4: Upsampled to restore our original dimensions.
Calculated using LI and RCR methods.

Through linear interpolation, we calculate the average value between our missing pixels that we threw away and "restore" our original values even though this will not be exactly the same value as we had originally (hence this is a lossy compression technique). Row-column replication is a similar process except we can simply copy the missing bits from its leftmost pixel and replicate the missing row by sampling the row before it. Once we fill in the missing values, we want to both visually check the quality of the images and measure the amount of loss that we suffered from performing these compression techniques. As seen in Figure 5, there are slight differences in quality when you zoom in to each pixel but the overall image quality is preserved.

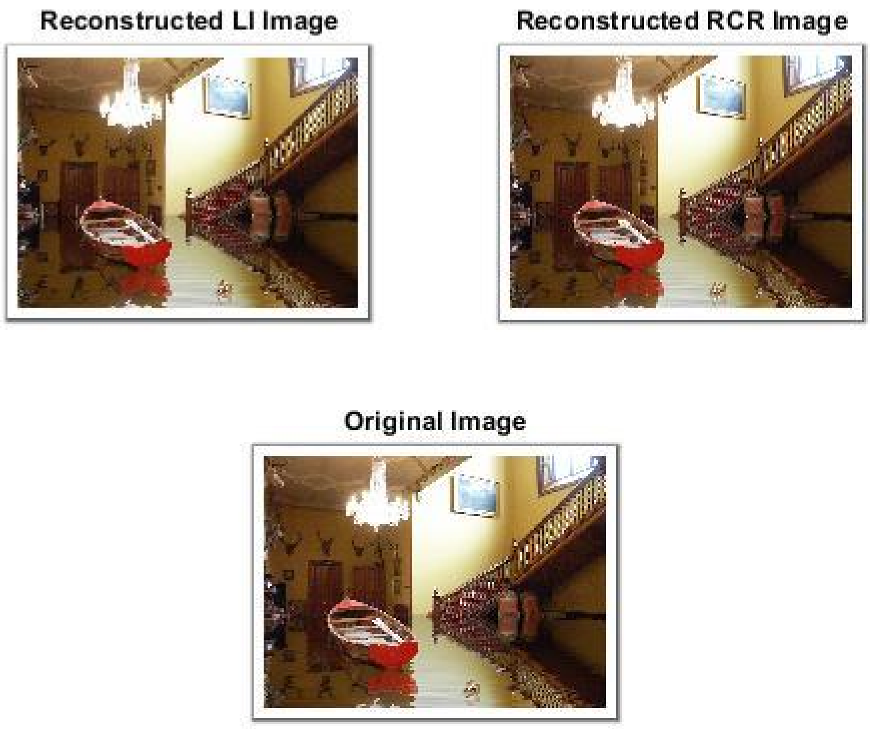

Figure 5: Comparing the results from our compressed images versus original.

We calculated our MSE to be 25.1865 for the sample image which is excellent considering we were able to compress our Cb and Cr bands by ¼. Some visual differences can be noted in detail in the MATLAB code on Part 9.

Conclusion

In this assignment, we were able to successfully compress our image using the techniques we learned in class. As discussed in the Results section, we were able to compress our image in two different ways and achieve optimal results. Some things to note is that there are slight visual differences between the Row-column replication and linear interpolation techniques (see part 9). Since these are lossy functions, this makes sense since we are trying to interpret the missing information that we had intentionally thrown away. Regardless, this is still an impressive feat considering we used a heavy compression ratio of 4:2:0.

References

[1] S. Kumar, "Lecture_Topic#1_MMFundas.pptx," Canvas. Web.