**Compe 565,** **Semester 2021**

**HW 1: Basic Digital Image Processing Operations**

**Prepared by**

Ethan Nagelvoort (821234668)

E-mail: enagelvoort@sdsu.edu

Electrical and Computer Engineering

San Diego State University

**Table of Contents**

* List of figures …………………………………………………………………………………….1
* Introduction ……………………………………………………………………………………....2
* Procedural Section ………………………………………………………………………………. 3
  1. Read and display the image using Matlab………………………………………………..3
  2. Display each band (Red, Green and Blue) of the image file……………………………..3
  3. Convert the image into YCbCr color space……………………………………………...4
  4. Display each band separately (Y, Cb and Cr bands)..........................................................4
  5. Subsample Cb and Cr bands using 4:2:0 and display both bands………………………..4
  6. Upsample and display the Cb and Cr bands using linear interpolation and simple row or column replication………………………………………………………………………..4
  7. Convert the image into RGB format……………………………………………………..5
  8. Display the original and reconstructed images (the image restored from the YCbCr coordinate)...........................................................................................................................5
  9. Comment on the visual quality of the reconstructed image for both the upsampling cases………………………………………………………………………………………6
  10. Measure MSE between the original and reconstructed images (obtained using linear interpolation only). Comment on the results……………………………………………..6
  11. Comment on the compression ratio achieved by subsampling Cb and Cr components for 4:2:0 approach…………………………………………………………………………….6
* Results……………………………………………………………………………………………..7
* Conclusion………………………………………………………………………………………..11
* References ………………………………………………………………………………………..11

**List of figures**

Figure 1.1: Original RGB image

Figure 1.2 : Red band of original RGB image

Figure 1.3: Green band of original RGB image

Figure 1.4: Blue band of original RGB image

Figure 1.5: YCbCr colorspace of original image

Figure 1.6: Y band of original YCbCr image

Figure 1.7: Cr band of original YCbCr image

Figure 1.8: Cb band of original YCbCr image

Figure 1.9: Cb band and Cr band subsampled image

Figure 1.10: Linear interpolation of subsampled Cb and Cr images

Figure 1.11: Simple row or column replication of subsampled Cb and Cr images

Figure 1.12: Reconstructed RGB image from linear interpolation vs. the original RGB image

Figure 1.13: Reconstructed RGB image from Simple row or column replication vs. the original RGB image

**Introduction**

Image and video coding can help a program deal with storage and transmission bandwidth issues. Many images and videos have storage and processing requirements that are too large for a programmer to deal with. Compression can help a programmer save storage and channel bandwidth. This is because compression offers the programmer a more effective way in transmitting data. In compression, the programmer reduces the size of the image without drastically changing the overall look of it. This reduction is done by eliminating unneeded pixels and because of this, the file size is also decreased.

In this homework, I utilized Matlab to display and manipulate an 536 by 704 pixel JPG image. These manipulations include, converting RGB to YCbCr, subsampling the JPG to a ratio of 4:2:0, and upsampling the JPG with linear interpolation and simple row-column replication. After upsampling in both methods, I was able to discover which of the two methods produced the more quality image. Also, I calculated the MSE values of the new images from upsampling (linear interpolation method) and the original RGB image. I also discovered how much the original RGB image was compressed after subsampling.

**Procedural Section**

1. Read and display the image using Matlab.

In order to read the image called Flooded\_house , I used imread and stored it into I. I then used imshow(I) to display the image. I made it figure 1 through figure(1) and titled the image, ‘RGB image’.

1. Display each band (Red, Green and Blue) of the image file.

To display each band, I made three variables , Green, Blue, Red, and had them all equal to I. I then loop through all the columns and rows of the images and make its corresponding colors 0. Hence, when looping through Green, I would turn the blue and red aspects of each pixel to 0, when looping through Red, I would turn the blue and green aspects of each pixel to 0, and when looping through Blue, I would turn the green and red aspects of each pixel to 0. I then used imshow() to display each of the different images.

1. Convert the image into YCbCr color space.

I used the function rgb2ycbcr() to convert I, which is in RGB, to YCbCr. I then used imshow() to display the YCbCr image with an appropriate title. I then used ycbcr2rgb() to convert I back into RGB.

1. Display each band separately (Y, Cb and Cr bands).

I first made a variable named ycbcr to equal rgb2ycbcr(I), which gave the YCbCr form of I. I then displayed the Y through imshow(ycbcr(:,:,1)). To display the other two aspects, Cb and Cr, you would just have to replace 1 in the previous code with 2 and 3. I then gave each image appropriate titles.

1. Subsample Cb and Cr bands using 4:2:0 and display both bands.

I first had two variable cb and cr equal to the Cb and Cr bands of ycbcr. I then looped through the rows and columns of both Cb and Cr starting at 2 and increasing by 2. Hence, the loop would only access every other row and column. In the loop, I would set the pixel to 0. After the loop was finished, I then would set cbsub equal to cb(1:2:536,1:2:704). This would make cbsub equal to the pixels of cb that were not 0. Getting rid of these 0s made the image smaller according to the 4:2:0 ratio. I did the same thing but with cr to a new variable called crsub. I then used the subplot() and subimage() functions to display both cbsub and crsub in one figure.

1. Upsample and display the Cb and Cr bands using linear interpolation and simple row or column replication.

I first did linear interpolation. I set two variables, linecb and linecr, and have them equal cb and cr from part 5. It is important to note that cb and cr still have the 0s from earlier. I then looped rows 1 - 535 and columns 1 - 703. I then stated that if linecb(row, col) == 0 and mod(row,2)==0, then linecb(row, col) = 0.5\*(linecb(row-1, col)) + 0.5\*(linecb(row+1, col)) and linecr(row, col) = 0.5\*(linecr(row-1, col)) + 0.5\*(linecr(row+1, col)). I only needed to check for linecb since linecb and linecr 0 locations were similar and I used the mod function to check if the row was even. The equation used in the if statement lets us take the average of the two pixels around in the same column and have it equal to the pixel that we are checking. This is in accordance with linear interpolation of even rows. For odd numbered rows, I had the same for loop but from rows 1-536 and instead used mod to check if the column was even. This is because on odd rows, even columns had 0s. Then if that was all true, linecb(row, col) = 0.5\*(linecb(row, col-1)) + 0.5\*(linecb(row, col+1)) and linecr(row, col) = 0.5\*(linecr(row, col-1)) + 0.5\*(linecr(row, col+1)). This gave the average of two neighboring pixels in the same row. I then displayed linecb and linecr using subplot() and subimage().

I then performed simple row or column replication. To do this I made two variables, cbcr and crrc, and had them equal to cb and cr. I then looped through the rows and columns. Then I used if statements to check if the current pixel is 0 and that the row is even. Then, for both cbcr and crrc, I made the current pixel equal to the corresponding pixel in the row above. I then had the same for loop but this time checked if the row was not even. Then, for both cbcr and crrc, I made the current pixel equal to the corresponding pixel in the column to the left. I then displayed cbcr and crrc using subplot() and subimage().

1. Convert the image into RGB format.

I converted the images gained from linear interpolation through lineRegRBG = ycbcr2rgb(cat(3, ycbcr(:,:,1), linecb, linecr)). ycbcr2rgb () is used to convert ycbcr to rgb and cat() is used to combine different values into one. I needed to combine the Y, Cb, and Cr aspect together and so used ycbcr(:, :, 1) to gain the Y part. I then placed linecb and linecr to act as Cb and Cr. I did the same thing for a variable called rowColRepRBG to convert what I got from row and column replication back to RGB.

1. Display the original and reconstructed images (the image restored from the YCbCr coordinate).

For this part, I simply used subplot() and subimage() to display the original RGB image and the two images gained from part 7.

1. Comment on the visual quality of the reconstructed image for both the upsampling cases.

In order to see which reconstructed image had a better quality, I first examined the images compared to the original and examined the method of reconstruction. I came to the conclusion that the image reconstructed from linear interpolation. This is because linear interpolation uses the averages of pixels around the 0 pixels. This gives the image a much smoother look. In row/column replication, 0 pixels were replaced by a pixel that was before or above that pixel. This gives the image a rougher look and so is not of a good quality.

1. Measure MSE between the original and reconstructed images (obtained using linear interpolation only). Comment on the results.

In order to calculate the correct MSE between the reconstructed image from linear interpolation and the original RGB image, I used the equation provided in lecture. This prompted me to write the following equation MSE= (sum(sum((I-lineRegRBG).^2)))/(536\*704). I use the sum array to return the sum of the vectors which is in accordance to the riemann sums used in the equation. I then divide by column\*rows. I then used fprintf to print out all the different MSE values for red, green, and blue. I got Red MSE =1.865722, Green MSE = 0.952889, and Blue MSE = 3.926873. It seems, from the MSE values, that blue had the highest distortion and green had the least distortion.

1. Comment on the compression ratio achieved by subsampling Cb and Cr components for 4:2:0 approach.

I first computed compression by computing the size of the first image and then dividing that by the size of the subsampled image. To get the size of the first image, I used the size function to get the length of the different dimensions of the original image and multiplied them together. Then to get the overall size of the subsampled image, I did the same thing but did it to the Y aspect of the image. Then I did the same thing to the Cr, using crsub, and Cb, using cbsub, aspects of the image. I then added these three values up to gain the overall size of the subsampled image. For the overall compression, I got 2.000000. This means the original image was reduced by half when it was subsampled.

**Results**



Figure 1.1: Original RGB image

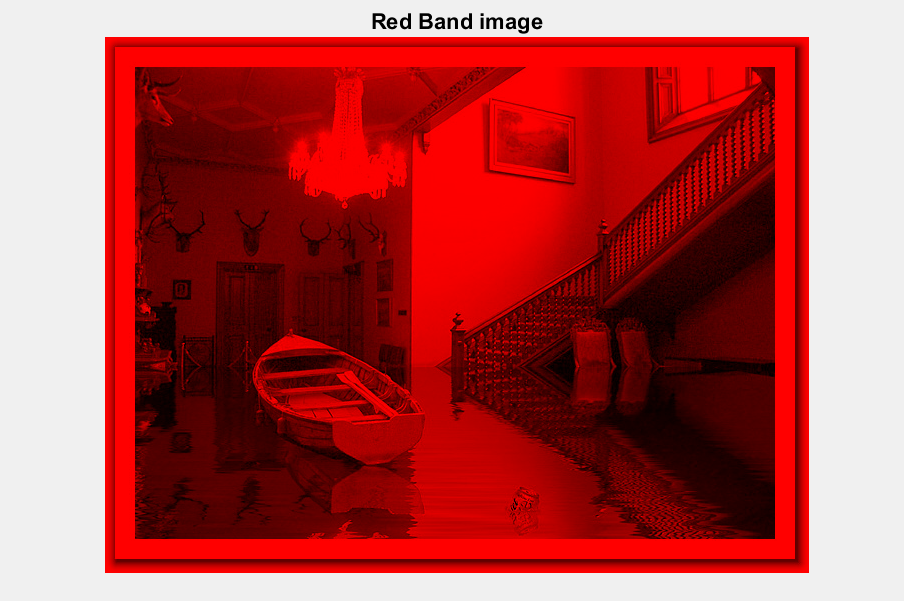


Figure 1.2 : Red band of original RGB image



Figure 1.3: Green band of original RGB image

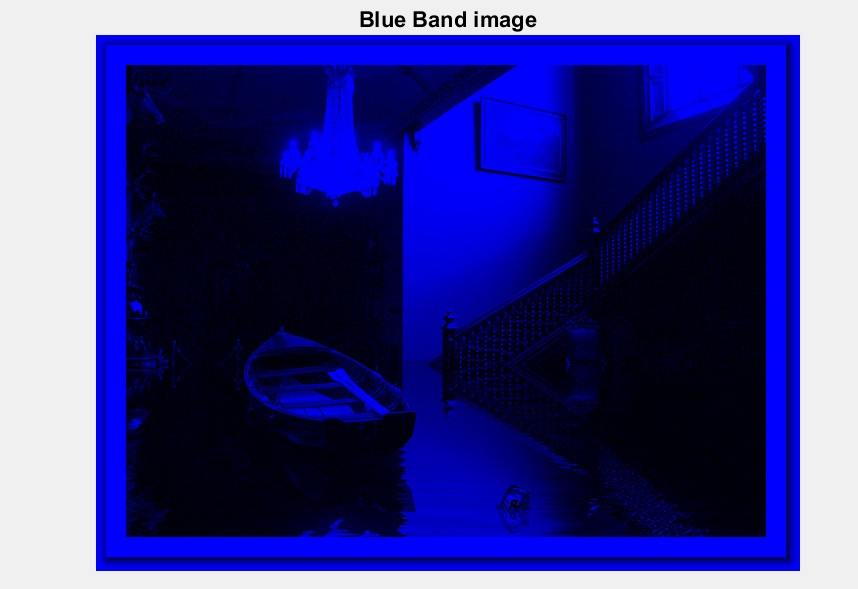


Figure 1.4: Blue band of original RGB image

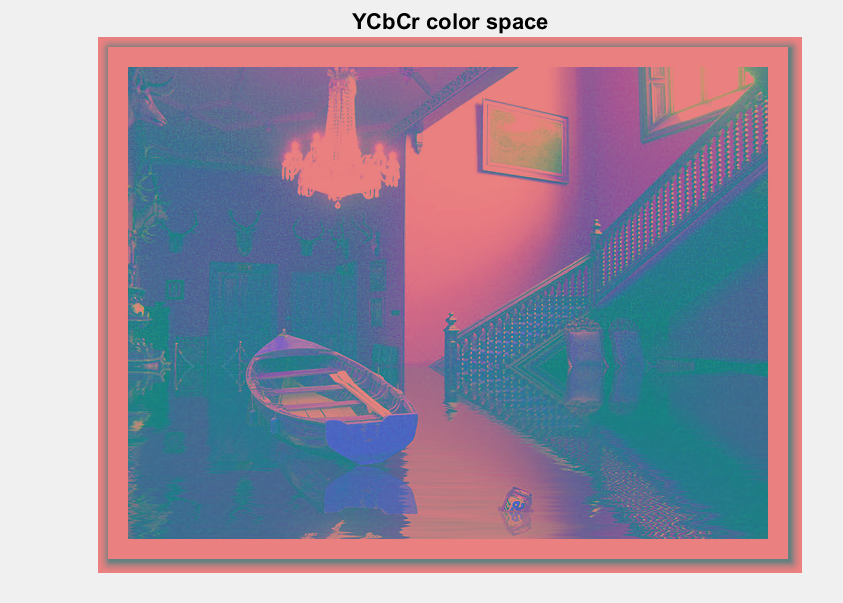


Figure 1.5: YCbCr colorspace of original image



Figure 1.6: Y band of original YCbCr image



Figure 1.7: Cr band of original YCbCr image



Figure 1.8: Cb band of original YCbCr image

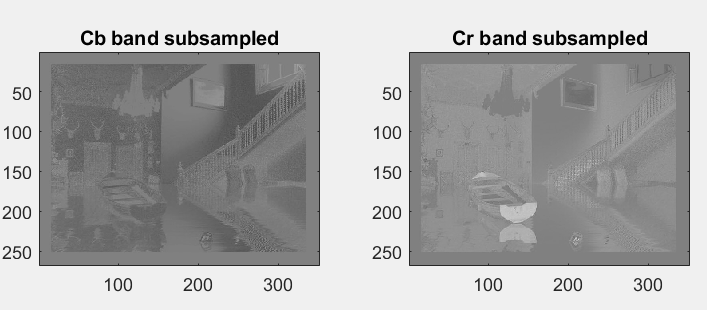


Figure 1.9: Cb band and Cr band subsampled image

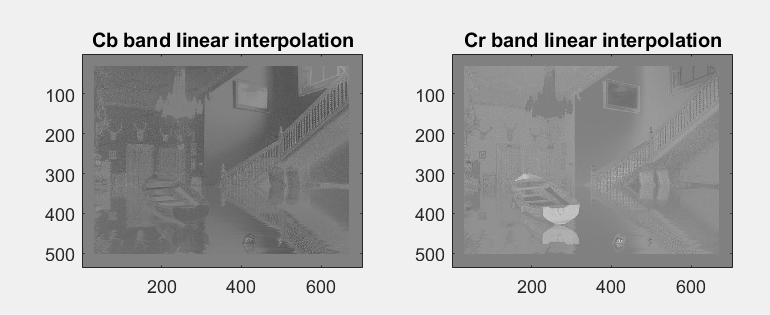


Figure 1.10: Linear interpolation of subsampled Cb and Cr images

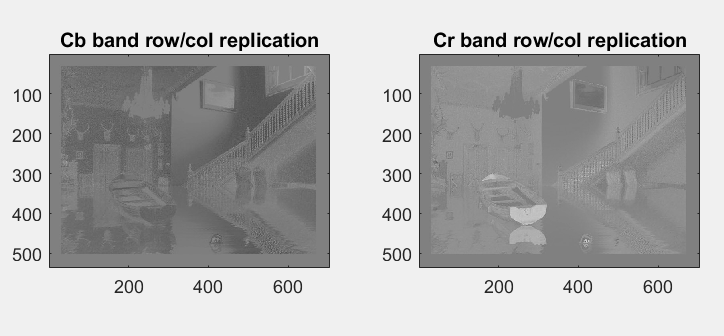


Figure 1.11: Simple row or column replication of subsampled Cb and Cr images

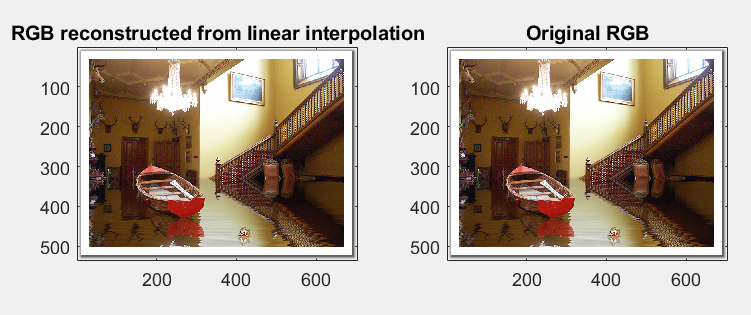


Figure 1.12: Reconstructed RGB image from linear interpolation vs. the original RGB image



Figure 1.13: Reconstructed RGB image from Simple row or column replication vs. the original RGB image

**Conclusion**

In this assignment, I learned how to display images different band components. This includes its RGB bands; red, blue, green, and YCbCr bands; Y, Cb, and Cr. I then learned how to subsample, using 4:2:0, an image to reduce its size. I did this with the Cb and Cr bands of YCbCr. I then used linear interpolation and simple row or column replication to put these images to their original sizes. The purpose of this assignment was to learn the three previous methods. I learned that subsampling with a ratio of 4:2:0 will reduce the image by half its size. I also learned that linear interpolation gives a smoother image than simple row or column replication due to linear interpolation replacing the 0 pixel with an average of the pixels around it.

**Reference**

[1] S. Kumar, “Lecture Topic #1: Multimedia Fundamentals Slides”, Compe 565, 2021

[2]S. Kumar, “Lecture Topic #2: Compression Fundamentals”, Compe 565, 2021