**CompE565, Semester 2020**

**HW Basic Digital Image Processing Operations**

Prepared by

Joseph Morga & Brandon Fong

jmorga@sdsu.edu & fong.m.brandon97@gmail.com

Electrical and Computer Engineering

San Diego State University

**Table of Contents**

**List of figures ………………………………………………………… 3-9**

**List of tables ……………………………………………………….. 10-11**

**Introduction ……………………………………………………………. 12**

**Procedural section ……………………………………………….. 12-13**

**Results …………………………………………………………….... 13-14**

**Conclusion …………………………………………………………….. 15**

**References …………………………………………………………….. 16**

**List of figures**

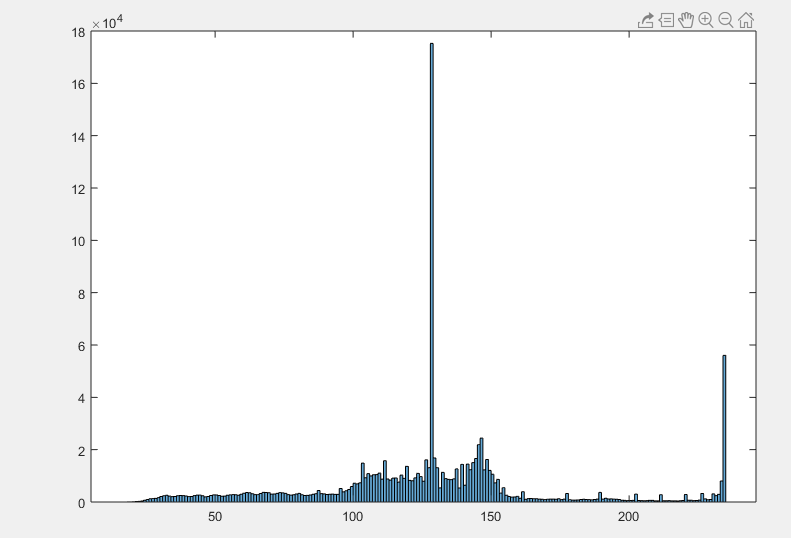


|  |  |
| --- | --- |
| *Figure 1: Green Component* | *Figure 2: Blue Component* |
|  |  |

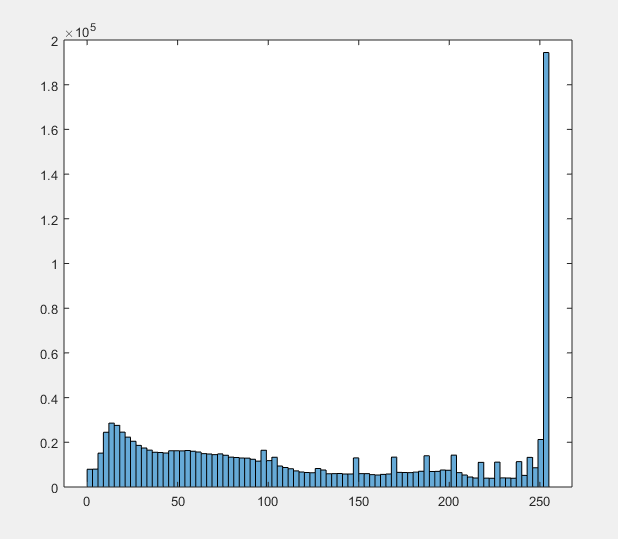
|  |  |
| --- | --- |
| *Figure 3: Red Component* | *Figure 4: Cr Component* |
|  |  |
| *Figure 5: Cb Component* | *Figure 6: Y Component* |
|  |  |

|  |  |
| --- | --- |
| *Figure 7: Cb Subsampled* | *Figure 8: Cr Subsampled* |
|  | |
| *Figure 9: Interpolation Reconstruction* | *Figure 10: Replication Reconstruction* |
|  |  |

**List of Tables**

****

*(Graph 1, YCbCr Histogram)*

****

*(Graph 2, RGB Histogram)*

**Introduction**

Storage has become a hot topic. Without compression, transmitting an image with a size of 1GB over a network with 5kbps (very low specs) will take a long time, approximately 7 hours. Consumers today want their information instantaneously, so us Engineers are just stringing along with that notion. Engineers develop technology that utilizes compression algorithms to reduce the size of an image and as a result can transmit 1GB data faster over a 5kbps network.

**Procedural Section**

Our team worked together and tackled each question one at a time. We helped each other out with questions that we were struggling in and kept a record of our progress via source control. In our source code, README.m is our main program that converts an image to YCbCr and then converts it back at the end of the program.

For a high level description, we conducted a step by step conversion to our JPEG image. First, we extracted the 3D RGB array and equated it to a matlab variable (array[x-axis, y-axis, color component]). Displaying the color components was as easy as isolating the 3rd field to either 1, 2, or 3 (red, green , and blue respectively)[1]. Second, using the rgb2ycbcr() method we converted the rgb 3d array to a ycbcr 3d array. In the ycbcr domain the variable had similar indexes as the rgb array, where the color component field actually isolates the y, cb, and cr components respectfully.

Subsampling and Upsampling requires work to be done in the ycbcr domain. Subsampling the ycbcr array was simply attenuating the cb and cr component in every other row/column to only retain the y component. Subsampling reduces the size of the array and hence reduces the size of the image, resulting in a compressed image (figure 8). Upsampling can be done through methods of interpolation or row/column replication. Our team attempted both methods and came out to similar results.

With our upsampled data, we measured the Mean-Squared Error. I used logic from matlab answers to use mean(mean()) to calculate the MSE [2]. I also crossed-checked my values for the MSE against the built-in matlab immse() method. Below are our results:

MSE (Red): 113.777741

MSE (Green): 0.187137

MSE (Blue): 15.292463

Our red component had a higher MSE, we believe because the red component might have been very significant in the Flooded\_house.jpg image.

**Results**

Displaying the components was not as simple as isolating them individually in the index. I had to utilize matlab’s zero() and cat() method to display the components correctly [2].

For subsampling, we took every other column and row in the image array and equated the values to another 3d array. This resulted in an array that is smaller but retained the essential Cb and Cr components (code below).



For our interpolation algorithm, we simply went to every even column on an odd row and averaged out the columns to the right and left of the index. For the even rows we did the same and averaged out the rows from top and below of the index (code snippet below).

for r = 1:rows

for c = 1:columns

if ((mod(c, 2) == 0) && mod(r, 2) ~= 0) % if we are in an odd row and it's an even column

if(c ~= columns)

ycbcrReconstructed(r,c,Cb:Cr) = (ycbcrReconstructed(r,c-1,Cb:Cr) + ycbcrReconstructed(r,c+1,Cb:Cr))/2;

end

end

end

end

for r = 1:rows

for c = 1:columns

if ((mod(r, 2) == 0)) % if we are in an even row

if(r ~= rows)

ycbcrReconstructed(r,c,Cb:Cr) = (ycbcrReconstructed(r-1,c,Cb:Cr) + ycbcrReconstructed(r+1,c,Cb:Cr))/2;

end

end

end

end

if(mod(columns,2) == 0)

ycbcrReconstructed(:,end,Cb:Cr) = ycbcrReconstructed(:,end-1,Cb:Cr);

end

if(mod(rows,2) == 0)

ycbcrReconstructed(end,:,Cb:Cr) = ycbcrReconstructed(end-1,:,Cb:Cr);

end

The replication algorithm basically took each of the previous rows and columns and copied them to the current indexed row and column (code snippet below).

or r = 1:rows

for c = 1:columns

% if we are in an odd row and it's an even column

if ((mod(c, 2) == 0) && mod(r, 2) ~= 0)

% copying the value from the column before index

ycbcrReconstructed62(r,c,Cr) = ycbcrReconstructed62(r,c-1,Cr);

ycbcrReconstructed62(r,c,Cb) = ycbcrReconstructed62(r,c-1,Cb);

% if it is an even row

elseif (mod(r, 2) == 0)

% copies the entire previous row

ycbcrReconstructed62(r,:,Cr) = ycbcrReconstructed62(r-1,:,Cr);

ycbcrReconstructed62(r,:,Cb) = ycbcrReconstructed62(r-1,:,Cb);

end

end

end

**Conclusion**

In conclusion, our project showcased the steps of the compression with 4:2:0 subsampling and interpolation. Without the steps of transferring the data over a network, our reconstructed image after compression had low error when comparing it to the original image. Our next goal is to actually achieve an identically reconstructed image even after transmitting data over a network.

**References**

[1] “r g b components of an image,” Matlab Answers, 21 August 2013

[2] “MSE Mean Square Error,” Matlab Answers, 27 March 2019