SYDE 575
Lab 5 - Report

Prepared by:
Timothy Tsang 20556306
Jonathan Wen 20515413

Professor David A. Clausi

Wednesday November 28th, 2018

1. Introduction

This lab attempts to perform image compression and segmentation to improve the quality of the image. First, chroma subsampling is investigated, through performing operations on images in the YCbCr colour space. Subsampling of luma channels and chroma channels are performed to see how it affects the image upon rescaling. Colour segmentation is performed in the L*a*b colour space, through using k-means clustering. Through the clustering, the segmentation of the different parts of the image are done. Image transformation is also done on the test images, through using the discrete cosine transform on a 8x8 image and attempting to reconstruct the image. Lastly, quantization is performed and different amounts of the z quantization matrix are used and investigated upon how that affects the quality of the reconstructed image.

2. Chroma Subsampling

First, chroma subsampling is performed on the peppers image to see the results of the effects upon image compression. First, we converted the image into the YCbCr colour space, as shown in the figure below. There are 3 channels: the Y channel (the luma or brightness component of the image), and the Cr and Cb channel (blue and red component of the image, relative to the green component).

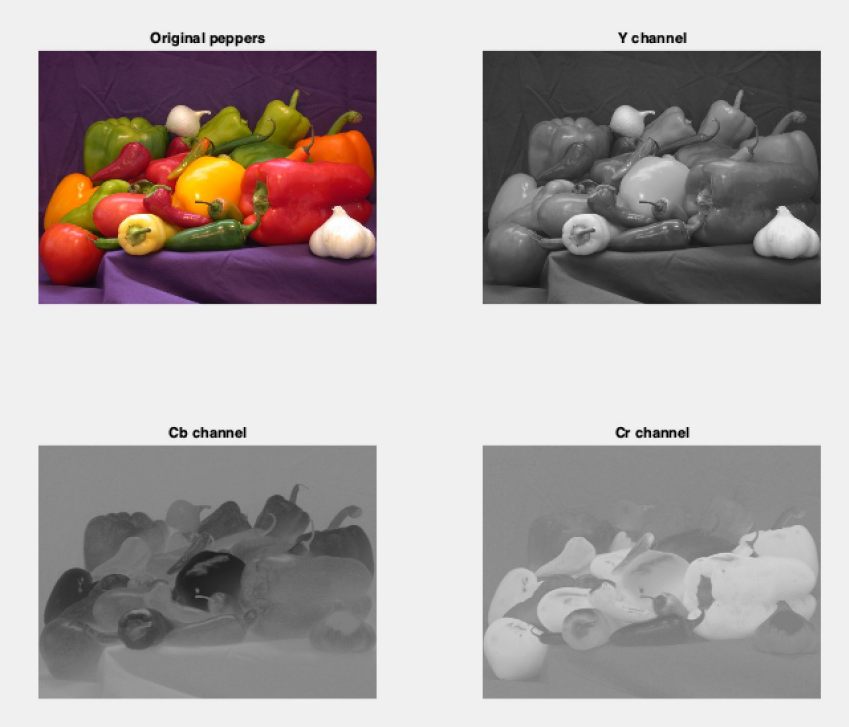

Figure 1 - Separate Channels in the YCbCr colour space

1. The Y channel holds information about the luminosity of the image, while the Cb and Cr channels hold information about the colour of the image. Cb represents the blue difference relative to the green component, and Cr represents the red difference relative to the green component.

2. From the images above, it is evident that the Y luma channel holds the most information about the detail of the image. The Cb and Cr channel do not hold as much detail. This means that Y is the most important channel to reconstruct the image, while the CbCr channels are only needed to generate the colour.

Next, resizing of the image was done, through reducing the resolution of the chroma channels (Cb and Cr) by 2 in both horizontal and vertical directions. After the resolution was reduced, the two chroma channels were then upsampled and combined with the original luma channel. Below shows the image of the resized image:

Figure 2 - Restored Image using the Chroma (Cb and Cr) channels

1. Looking at the images, there is almost no difference in quality and detail between the two images. When reducing the CbCr channels, it does not impact the final image much. We know that the human visual system is not sensitive to chroma change, thus it makes sense to remove as much chroma information when compressing an image because it would not affect the overall perception quality.

2. Chroma subsampling seems to be very effective in reducing the information needed while not reducing the quality. Even when eliminating 75% of the colour information, the final image almost looks identical to the original.

Next, instead of modifying the chroma channels, the luma channel will be modified instead. The luma channel had its resolution reduced by a factor of 2 in both directions and then upsampled. It was then combined with the two original Cb and Cr chroma channels. Below is the result:

Figure 3 - Restored Image using the Luma (Y) channel

1. Looking at the two images, there is a slight difference in the quality. The subsampled image holds less detail in the edges, and seems a bit more blurry. This also confirms that the Luma (Y) channel contains more of the important information that would be imperative for image perception quality of a human for visual purposes.

2. When reducing the Y (luma) channel, it greatly affects the quality of the image, much more than reducing the CbCr channels. Making sure that when upscaling the image if the luma (Y) channel is reduced is very important as the image can lose a lot of very important detail that would influence the image perception quality.

3. Comparing the reduced Y channels with original CbCr and reduced CbCr channels with original Y, there is a significant difference. The first methods of subsampling CbCr is more effective in data reduction (75% reduced compared to 50%), as well as retains more detail in the final image.

3. Colour Segmentation

In this section, colour image segmentation is investigated, through the use of the k-means clustering and classification. In order to segment, the image is first converted from the RGB colour space to the CIELAB or L*a*b colour space, which has three channels: the L dimension (the lightness component), the a dimension (green to red component) and the b dimension (blue

to yellow component). In order to convert to the CIELAB colour space, the built-in MATLAB functions ‘makecform’ and ‘applycform’ were used.

Next, the k-means clustering was done to the image. Two variations were run, the first with k = 2 clusters/groups and the second with k = 4 clusters/groups. After retrieving all of the start coordinates (mu), the k-means clustering was performed. Once the k-means clustering was done, a matrix of all of the classified pixels were created, depending on whether it was closer to one cluster or another through the measuring of squared Euclidean distance. The clustered index was then used to relabel each pixel with a specific colour that represented the cluster it was in, using the ‘reshape’ function. Below is the resulting image for 2 and 4 clusters respectively.

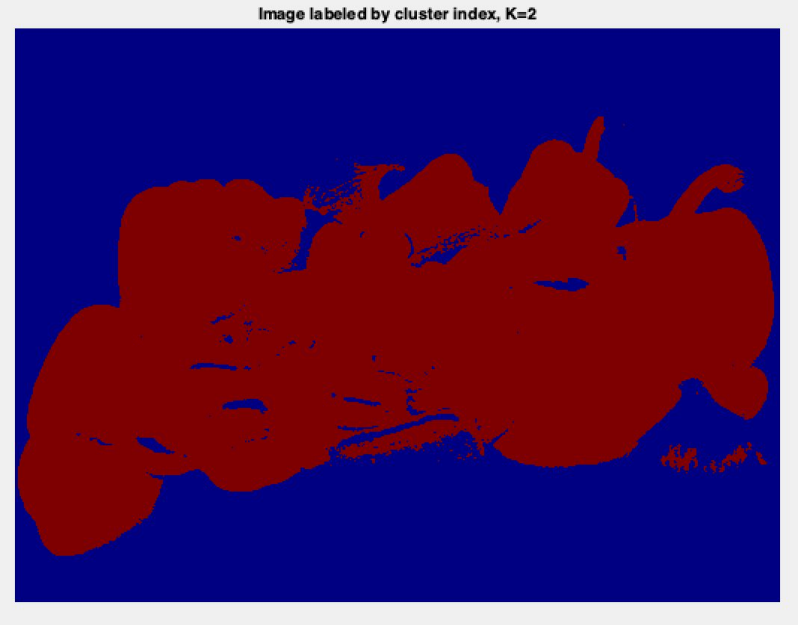

Figure 4 - K-means clustering with 2 clusters (k=2)

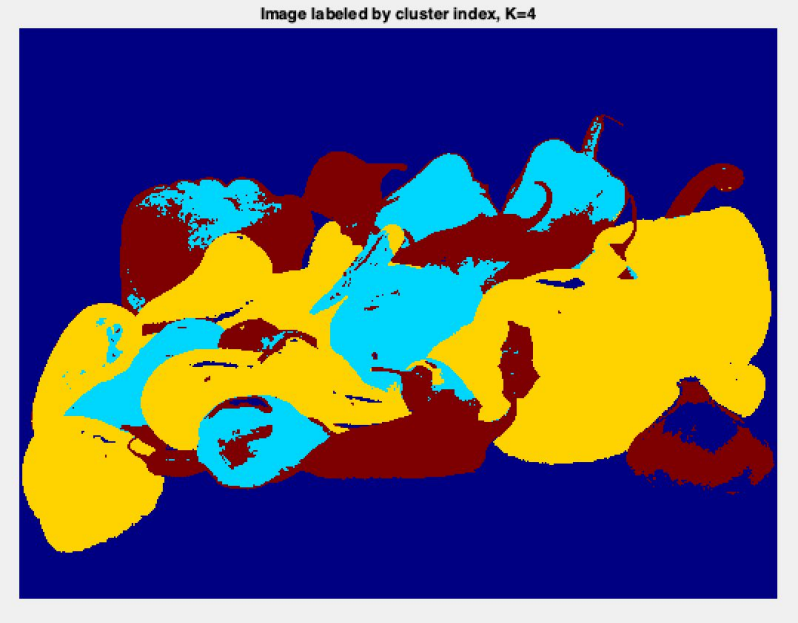

Figure 5 - K-means clustering with 4 clusters (k=4)

1. When K=2, the clustering algorithm tried to group pixels based on colour similarity. Therefore, the foreground (peppers) and background were clustered differently. When K=4, k-means was able to identify groups of colours, instead of the just foreground and background. In this case, 4 different colours were clustered: purple, red, green and green/yellow.

2. The initial points of the clusters does not seem to greatly affect the final segmentation of the image. This is because the clusters will slowly converge to the most optimal positions, no matter where they are placed initially. However, the initial points can affect the speed at which it converges to a solution.

Next, segmentation of the different dimensions were done for the image with 4 clusters (k=4). Reshaping the a* and b* channels were done and each cluster was then outputted with their original colour. Below are the different colour segmentations for each cluster.

Figure 6 - Each segmented cluster

3. Looking at the different images, the clustering algorithm seemed to perform well in segmenting some of the peppers. The red peppers have clearly been segmented out, as well as the purple background. However, it had a harder time trying to segment the yellow and green peppers, as the they were very close in colour similarity.

4. Image Transform

Next, the effects of transforming an image using the discrete cosine transform (DCT) was investigated. In order to create the DCT matrix, the ‘dctmtx’ MATLAB function was used to create a 8x8 DCT transform matrix. Below shows the 8x8 DCT transform matrix.

Figure 7 - 8x8 DCT transform matrix

1. Each row of the DCT matrix represents points of a cosine wave, where the first row is all constant values, and the other rows have 8 equally spaced points to make up an 8x8 matrix. It is important to note that for each row, the sum of all of the points equate to 0.

Using the 8x8 DCT transform matrix, it was applied to the grayscale Lena image on 8x8 sub-image blocks. Below is the DCT matrix of the top left corner at (297, 81) and (1,1) respectively.

Figure 8 - 8x8 sub-image at (297, 81)

Figure 9 - 8x8 sub-image at (1, 1)

1. Looking at the two 8x8 blocks of the DCT image, it seems that the energy distribution is very discrete. Each pixel value looks as if it is either black or white. Each pixel represents the DCT coefficient in that specific row and column of the DCT matrix. This would be very useful for image compression, because the values can be represented much easier with a lot of repetition or values. Hence, they can be encoded using Huffman block coding.

2. Since MATLAB matrices start at 1 instead of 0, the point (1,1) would represent the DC component/coefficient (which is situated at the top left of the image), which would consist of most of the energy stored. This makes sense since most of the energy would come from lower frequencies, which are on the top left of the DCT.

Next, a threshold mask is applied to the sub-images to only keep 6 of the DCT coefficients per sub-image. Performing an inverse DCT on the sub-images results in this image below. The PSNR for the image, relative to the original Lena image was 18.260879324836570.

Figure 10 - Reconstructed Lena image using DCT matrix

1. The reconstructed image looks very similar to the original black and white lena. However, some of the detail is missing, the edges are not as sharp, and the dynamic range is much smaller. This is because the binary mask that was applied eliminated a lot of the image information, since only 6/64 pixels in the block were not eliminated.

2. The most prominent artifacts in the image are the 8x8 blocks that appear. There is a clear distinction between two consecutive blocks, because the patterns do not match up. This artifact appears because of the DCT transform that was applied. The image was divided up into 8x8 blocks, and each block was processed through DCT.

3. In conclusion, it appears that DCT is very effective in terms of image compression. It can properly compress images in a lossy manner, but still recover much of the image's detail. Adjusting the binary mask that is applied can also change the amount of image compression.

5. Quantization

Lastly, quantization is performed to remove redundant and undesirable information that would not affect the overall image once the image is compressed or reconstructed. Therefore, the Lena sub-images was subjected to a transform using the 8x8 DCT transform matrix above. One thing to note is to subtract 128 due to numerical reasons. Then, the sub-images were divided by the quantization matrix (Z) and an inverse DCT transform was done on the sub-images. The images were reconstructed, with the figures shown below. Quantization was done using a quantization matrix Z that was also multiplied by a factor of 3, 5, and 10.

Figure 11 - Quantization Images using Z = 1 and Z = 3

Figure 11 - Quantization Images using Z = 5 and Z = 10

1. In order for quantization to occur, the DCT coefficient is divided by the quantization matrix. For sub-images ranging from 8x8 to 16x16, most of the coefficients (~75%) are truncated or erased due to redundancies in information. This is done as most of the quantized numbers are either very small or 0, which essentially removes the information in the compressed image.

2. The image with 3Z is clearly much more worse in image reconstruction, as it is pixelated due to the fact that it lost a lot more information compared to only dividing by 1 quantization matrix (Z) during the quantization process.

3. As the quantization matrix is multiplied by a greater factor, the image progressively gets worse and worse. This is due to the fact that more and more information is lost during the quantization stage. So when reconstructing the compressed image, the information is not present and would create an image that is much more pixelated and not close to the original image.

4. Clearly, as the quantization increases, the fine details will start to be lost. Thus, a blocking artifact will become seen more easily, which are essentially each pixel block being shown and subimage boundaries being shown and visible.

5. It is important that while quantization is necessary and very important when compressing an image, which is crucial in image storage with images that can be massive, that there is

a tradeoff between the amount of data and information that is lost. One method of making sure that quantization does not greatly affect the quality of the image and produce blocking artifacts, adaptive quantization can be done to make sure that the details of the images (i.e. the edges) are not lost. Within that method, sub-images are measured based on how much they contain image detail; if that particular sub-image contains a lot of image detail, that specific sub-image would go through less quantization, whereas sub-images that have not as much image details would go through more quantization. This would help balance the amount of quantization that is needed, while optimizing for minimizing the amount of high image details and information that is lost.

6. Conclusion

Investigation of how images can remove specific details while still being perceived as the same were done. The first was performing chroma subsampling to see how that affects the image. It was clear that when removing the chroma channels or reducing the information on the chroma channels, the upsampled image was not affected very much, as the human visual system is not as sensitive to chroma changes. Rather, it is more sensitive to luma changes, which was shown that when an image has resized the luma channel, the upsampled image lost a lot more information that was clearly visible. Next, colour segmentation was done in the L*a*b colour space and k-means clustering classification was used to find the different clusters. Afterwards, image transformation was investigated through the use of the discrete cosine transform. Lastly, quantization was performed to see the effects on a quantized image. It was seen that if quantization occurs more, the fine details of the image (i.e. edges) will become lost, and thus needs to have a balance between having the image being quantized while retaining the more important information and details of the image.

7. Appendix

```
%% Section 1
lena = imread('lena.tiff');
peppers = imread('peppers.png');

figure;
imshow(lena);
figure;
imshow(peppers);

%% Section 2 - Chroma Subsampling
ycbcr_peppers = rgb2ycbcr(peppers);
Y = ycbcr_peppers(:,:,1);
Cb = ycbcr_peppers(:,:,2);
Cr = ycbcr_peppers(:,:,3);

figure;
subplot(2,2,1);
imshow(peppers);
title('Original peppers');

subplot(2,2,2);
imshow(Y);
title('Y channel');

subplot(2,2,3);
imshow(Cb);
title('Cb channel');

subplot(2,2,4);
imshow(Cr);
title('Cr channel');

% Reduce resolution of Cr by 2
reduced_cr = imresize(Cr, 0.5);
upscale_cr = imresize(reduced_cr, 2, 'bilinear');

% Reduce resolution of Cr by 2
reduced_cb = imresize(Cb, 0.5);
upscale_cb = imresize(reduced_cb, 2, 'bilinear');

ycbcr_upscale = ycbcr_peppers;
ycbcr_upscale(:,:,2) = upscale_cb;
ycbcr_upscale(:,:,3) = upscale_cr;

recombined_upscale = ycbcr2rgb(ycbcr_upscale);
```

```
figure;
subplot(1,2,1);
imshow(recombined_upscale);
title('Reduced CbCr combined with original Y');
subplot(1,2,2);
imshow(peppers);
title('original');

% Reduce resolution of Cr by 2
reduced_y = imresize(Y, 0.5);
upscale_y = imresize(reduced_y, 2, 'bilinear');

ycbcr_upscale(:,:,1) = upscale_y;
recombined_upscale = ycbcr2rgb(ycbcr_upscale);

figure;
subplot(1,2,1);
imshow(recombined_upscale);
title('Reduced Y combined with original CbCr');
subplot(1,2,2);
imshow(peppers);
title('original');


%% Section 3: Colour Segmentation

C = makecform('srgb2lab');
converted_pepper = applycform(peppers,C);

ab = double(converted_pepper(:,:,2:3)); % NOT im2double
m = size(ab,1);
n = size(ab,2);
ab = reshape(ab,m*n,2);

K = 2;
row = [55 200];
col = [155 400];
mu = zeros(2,2);
% K = 4;
% row = [55 130 200 280];
% col = [155 110 400 470];
% mu = zeros(4,2);
% Convert (r,c) indexing to 1D linear indexing.
idx = sub2ind([size(converted_pepper,1) size(converted_pepper,2)], row, col);
% Vectorize starting coordinates
for k = 1:K
    mu(k,:) = ab(idx(k),:);
```

```
end

cluster_idx = kmeans(ab, K, 'start', mu);

% Label each pixel according to k-means
pixel_labels = reshape(cluster_idx, m, n);
figure;
imshow(pixel_labels, []);
title('Image labeled by cluster index, K=2');
colormap('jet');

mask = repmat(pixel_labels, [1,1,3]);
clustered_peppers = peppers;

figure;
for k = 1:K
    temp = uint8(mask(:,:,1) == k);
    clustered_peppers(:,:,1) = peppers(:,:,1).*temp;
    clustered_peppers(:,:,2) = peppers(:,:,2).*temp;
    clustered_peppers(:,:,3) = peppers(:,:,3).*temp;
    subplot(2,2,k);
    imshow(clustered_peppers);
    title(sprintf('Segmented region %d',k));
end

%% Section 4
gLena = double(rgb2gray(lena));

T = dctmtx(8);
% dct = D*gLena*D';

figure;
imshow(T);
title('8X8 DCT matrix');

% Apply transformation
F_trans = floor(blockproc(gLena-128, [8 8], @(x) T*x.data*T'));

figure;
imshow(F_trans(297:304, 81:88));
title('8x8 F_trans at 297,81');

figure;
imshow(F_trans(1:8, 1:8));
title('8x8 F_trans at 1,1');


% Threshold
```

```
mask = [1 1 1 0 0 0 0 0;
1 1 0 0 0 0 0 0;
1 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0];
F_thresh = blockproc(F_trans, [8 8], @(x) mask.*x.data);

f_thresh = floor(blockproc(F_thresh, [8 8], @(x) T'*x.data*T)) + 128;

figure;
imshow(f_thresh, []);
title('Final output after DCT transform');

lena_psnr = 10*log10(1/mean2((gLena-f_thresh).^2));

%% Section 5 - Quantization
gLena = im2double(rgb2gray(lena));

Z = [16 11 10 16 24 40 51 61;
12 12 14 19 26 58 60 55;
14 13 16 24 40 57 69 56;
14 17 22 29 51 87 80 62;
18 22 37 56 68 109 103 77;
24 35 55 64 81 104 113 92;
49 64 78 87 103 121 120 101;
72 92 95 98 112 100 103 99];

for i = [1,3,5,10]
    F_thresh = blockproc(F_trans, [8 8], @(x) round(x.data./(Z*i)));

    f_thresh = floor(blockproc(F_thresh, [8 8], @(x) T'*(x.data.*(Z*i))*T)) +
128;

    figure;
    imshow(f_thresh, []);
    title(sprintf('Quantized image with Z*%d', i));
end
```