EE569: Homework #1

ISSUED: 1/12/2018 DUE DATE: 11:59PM, 2/4/2018

ISHAN MOHANTY
USC ID: 4461-3447-18
EMAIL: IMOHANTY@USC.EDU

EE569 HOMEWORK #1
JANUARY 31, 2018

Problem 1: Basic Image Manipulation

(a) Color Space Transformation

1. **Color-to-Grayscale Conversion**
2. **CMY(K) Color Space**

I. Abstract and Motivation

"**Color Space**" is described as a particular arrangement of colors based on a color model. In this section we discuss three very important color models namely, RGB color model, Gray-Scale color model & CMYK color model.

The **RGB** (**Red,Green,Blue**) color space is based on the 24-bit RGB color model. The RGB color model has 3 distinct channels corresponding to R, G & B respectively. Each pixel in the channel contains 8 bits or 1 byte of data. Therefore, every pixel in the corresponding RGB channel has 2^8= 256 color intensity levels. These 256 color levels describe the "gray scale" values of a pixel and range from 0 to 255 values where 0 represents black , 1 represents white and the other values in between represent different shades of gray. The RGB color image has 3 bytes per pixel and can portray 256x256x256 = 1,67,77,216 colors as each RGB channel can depict 256 colors. The RGB color model is an additive color model and is used widely in computer monitors and computer graphics as they are highly compatible with the human visual system.

The **Gray-Scale** image contains only 1 byte or 8 bits per pixel and therefore can only illustrate 256 color levels. These images are also known as Black and white or monochrome images. The Gray scale model forms a basis for the RGB color model as each R,G & B channel is a separate gray scale channel stacked into one image.

The **CMYK (Cyan, Magenta, Yellow & Key(Black))** color space uses a subtractive color model and is used extensively for color printing. When we perform a color-inversion of an RGB color image we end up with the CMYK image model. It is complementary to the RGB color model.

Color Space Transformation is the process by which we convert one color space to another using specific operations. Our motivation in this exercise is to transform the RGB color space to the Gray-Scale equivalent and the CMYK color space.

II. Approach and Procedures

Approach for Color-to-GrayScale Conversion:

There are 3 approaches to achieve the Color to GrayScale Conversion described below:

- The **lightness** method: This method averages the most prominent and least prominent colors: (max (R, G, B) + min (R, G, B)) / 2.
- The **average** method: Calculates the average values of: (R + G + B) / 3.
- The **luminosity** method: This Performs the weighted average on the pixel intensities using the formula: 0.21 R + 0.72 G + 0.07 B.

Approach for CMY(K) color space:

we need to perform color inversion from RGB color space. So in this method we calculate the pixel intensities of cyan, magenta and yellow by the following equations: C = 1 – R, M = 1 – G, Y= 1 – B. After we perform these steps we achieve the CMY(K) color space.

Procedure for Color-to-GrayScale Conversion (Algorithm):

1. Read Input image which is in 24-bit RGB image format.
2. Extract Red, Green and Blue pixel intensity values and store in variables.
3. Manipulate the Red, Green and Blue pixel intensities according to the lightness, average and luminosity methods and store the resulting value as a gray pixel intensity variable.
   i. Gray = (max(R,G,B) + min(R,G,B)) / 2 ;
   ii. Gray = (R+G+B)/3;
   iii. Gray = (0.21 R + 0.72 G + 0.07 B);
4. Put the Gray pixel intensity into an output buffer of 8-bit Grayscale format.
5. Repeat procedure for all pixels in the input image.
6. This achieves the Gray Scale conversion and the output image has only 256 color levels.

Procedure for CMY(K) color space (Algorithm):

1. Read Input image which is a 24-bit RGB image format.
2. Extract Red, Green and Blue pixel intensity values and store in variables.
3. Perform the following operations:
   i. C = 255 – R;
   ii. M = 255 – G;
   iii. Y = 255 – B;
4. Store the results of the above operations in the 3 variables corresponding to cyan, magenta and yellow.
5. Put the pixel intensities for C, M & Y into 3 different output buffers in 8-bit Grayscale format.
6. Repeat this procedure for all pixels in the input image.
7. We successfully obtain the C, M & Y GrayScale images corresponding to R,G,B.

III. Experimental Results

Shown below are the results for Problem 1 (a) & (b):

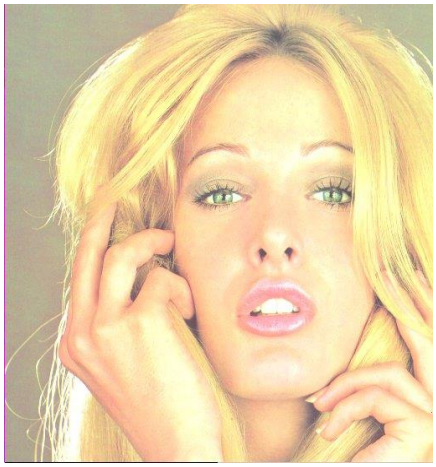


Fig : Color Tiffany Image

Fig 2 : Lightness Method

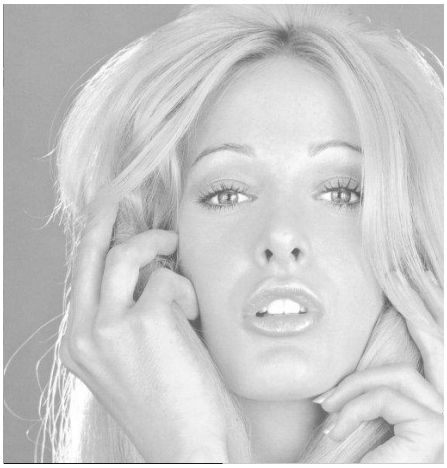


Fig 3: Average Method

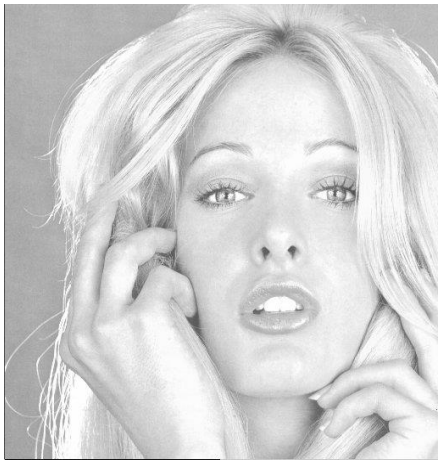


Fig 4: Luminosity Method

Fig 5: Input Image – Bear

Fig 6: GrayScale Image of Cyan

Fig 7: GrayScale of Magenta

Fig 8: GrayScale Image of Yellow

Fig 9: Input Image – Dance

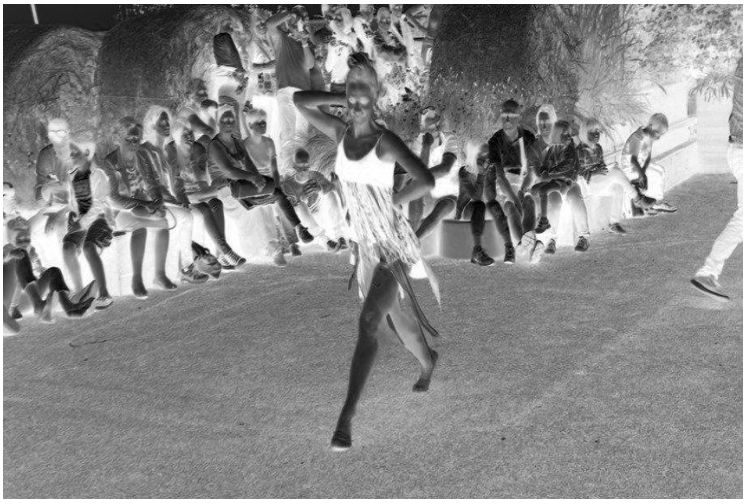


Fig 10: GrayScale Image of Cyan

Fig 11: GrayScale Image of Magenta

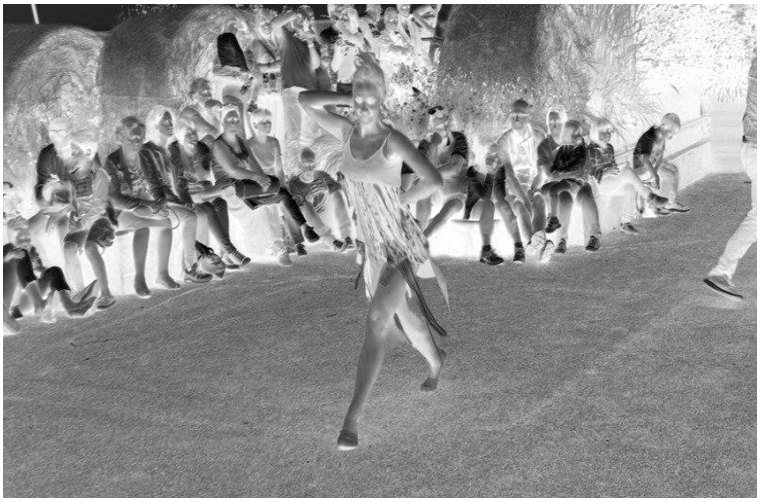


Fig 12: GrayScale Image of Yellow

IV. Discussion

Discussion for the **Color-to-Grayscale Conversion:**

Comparing Figures 2, 3 & 4 we observe that Figure 4, The Luminosity method produces an output image that is visually better than the Average and Lightness method. The contrast is much better and clearly seen in the Image produced by the Luminosity method.

Discussion for the **CMY(K) Color Space:**

The corresponding gray scale images for cyan, magenta and yellow have been generated and displayed in figures 6, 7 & 8 for the Bear input image and figures 10,11 & 12 for the Dance input image.

Problem 1: Basic Image Manipulation

(b) Image Resizing via Bilinear Interpolation

I. Abstract and Motivation

Bilinear Interpolation is the key algorithm used to resize images. It is achieved by performing linear interpolation in one direction and then performing linear interpolation in the other direction. It is used extensively in the field of image processing and computer vision as an image reconstruction (Resampling) technique.

II. Approach and Procedures

Approach for Bilinear Interpolation:

We use the following algorithm to perform bilinear interpolation:

Algorithm:

Let **I** be an $R \times C$ image.
We want to resize **I** to $R' \times C'$ to make a new image **J** with pixel locs (r',c').

Let $s_R = R / R'$ and $s_C = C / C'$.

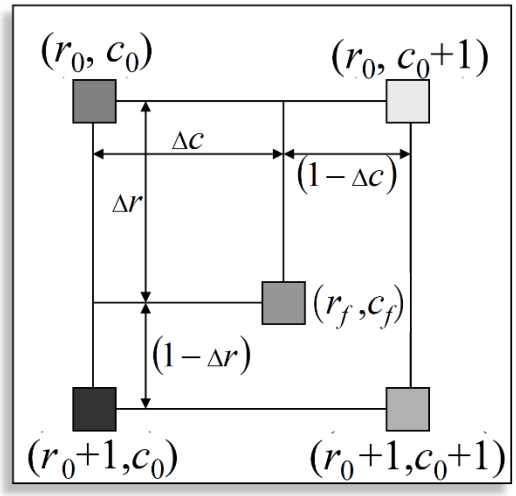
Let $r_f = r' \cdot s_R$ for $r' = 1,\ldots,R'$

and $c_f = c' \cdot s_C$ for $c' = 1,\ldots,C'$.

Let $r_0 = \lfloor r_f \rfloor$ and $c_0 = \lfloor c_f \rfloor$.

Let $\Delta r = r_f - r_0$ and $\Delta c = c_f - c_0$.

Then $\mathbf{J}(r',c') = \mathbf{I}(r_0,c_0)\cdot(1-\Delta r)\cdot(1-\Delta c)$
$+\mathbf{I}(r_0+1,c_0)\cdot\Delta r\cdot(1-\Delta c)$
$+\mathbf{I}(r_0,c_0+1)\cdot(1-\Delta r)\cdot\Delta c$
$+\mathbf{I}(r_0+1,c_0+1)\cdot\Delta r\cdot\Delta c.$



Here: source : Reference [1]

1. S_R & S_C are scaled width ratio and scaled height ratio respectively.

2. r_f & c_f are mapped coordinates of the pixels corresponding to the input image pixel coordinates.

3. r_0, c_0 , Δr & Δc are intermediate parameters used in mathematical manipulation on input image pixel coordiantes.

3. r′ & c′ are the coordinates of the pixels in the resized image.

Procedure for Bilinear Interpolation

1. Read input image.
2. Calculate S_R, S_C then use these parameters to calculate r_f & r_c.
3. With this info calculate r_0,c_0 and then calculate Δr & Δc.
4. Now using the above formula in the approach, Resize Input image I(r,c) to Output resized image J(r′,c′).

III. Experimental Results

Shown below are the results for bilinear interpolation:

Figure 13: Original Airplane Image of size 512x512

Figure 14: Resized Airplane Image of size 650x650

IV. Discussion

After running the algorithmic code on the Airplane.raw image (original image) of dimensions 512x512, we observed that it was resized to an Image of dimensions 650x650. We can clearly observe the difference in the size shown in the experimental results above. One drawback of bilinear interpolation is that there will be loss of pixel data when we resize and this would result in a blurred image but nevertheless this is a good method for image reconstruction.

Problem 2: Histogram Equalization

(a) Histogram Equalization

I. Abstract and Motivation

Histogram Equalization is the process by which we spread out the most frequent pixel intensities uniformly over the entire histogram region. This is an extremely useful technique for contrast adjustment, hence a very dark or very bright image will look better once we perform this technique. It helps in background and foreground enhancement. The disadvantage of this technique is that in many cases it increases noise and hence decreases the image quality. We are motivated to perform two methods of histogram equalization, one which is transfer function based and the other is cumulative distribution based.

II. Approach and Procedures

<u>Approach & Procedure for Transfer Function Based method A:</u>

1. Read the input RGB image.
2. Write a function to get the histogram.
3. Calculate the normalized probability values.
4. With these probability values for each gray level create the CDF table.
5. Make a lookup table.
6. Get the output image based on corresponding CDF values with respect to intensity values.
7. We obtain the equalized histogram output image.

<u>Approach & Procedure for CDF Based method B:</u>

1. We use the Bucket filling method to accomplish the histogram equalization.
2. In our input image there are 400x300 = 120000 pixel which needed to be equally divided into 256 buckets. Therefore,there are 120000/256 = 468.75 pixels that go into each. We round it to 468 pixels and put 192 pixels in the 0 to 191 pixel value intensity. So that there are 469 pixels in the first 192 buckets and 468 pixels in the rest of the buckets.
3. Then we create a look up table and transfer the corresponding number of pixels of particular intensity to the output image.
4. We have successfully created the histogram equalized output image.

III. Experimental Results

(a) Original Desk.raw image

(b) Histogram Equalized Image

Figure 15: Histogram Equalization Transfer function based Method A

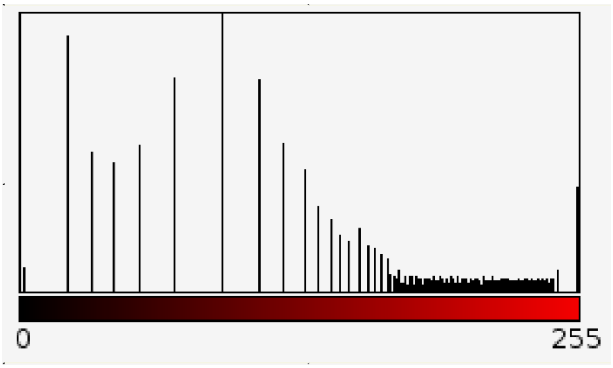


Figure 16: Histogram for Red channel After Equalization

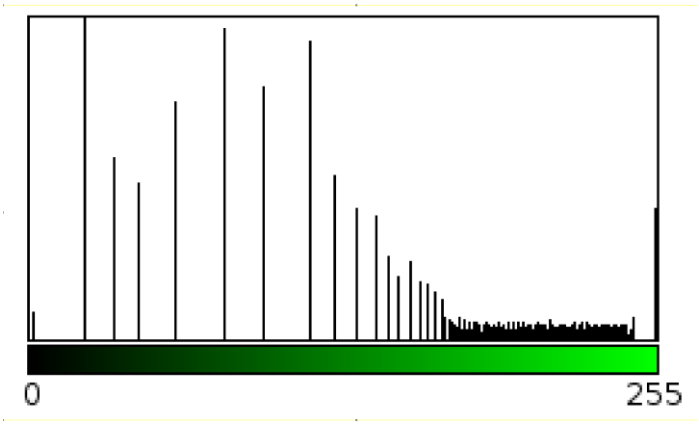


Figure 17: Transfer function for Green channel After Equalization

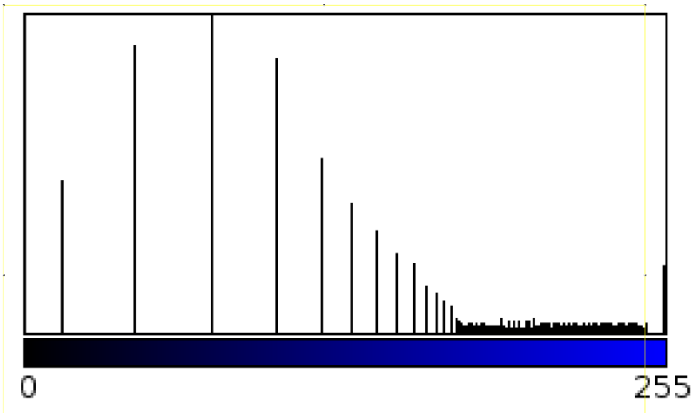


Figure 18: Transfer function for Blue channel After Equalization

(a) Original Desk.raw image (b) Histogram Equalized Image

Figure 19: Histogram Equalization Transfer function based Method

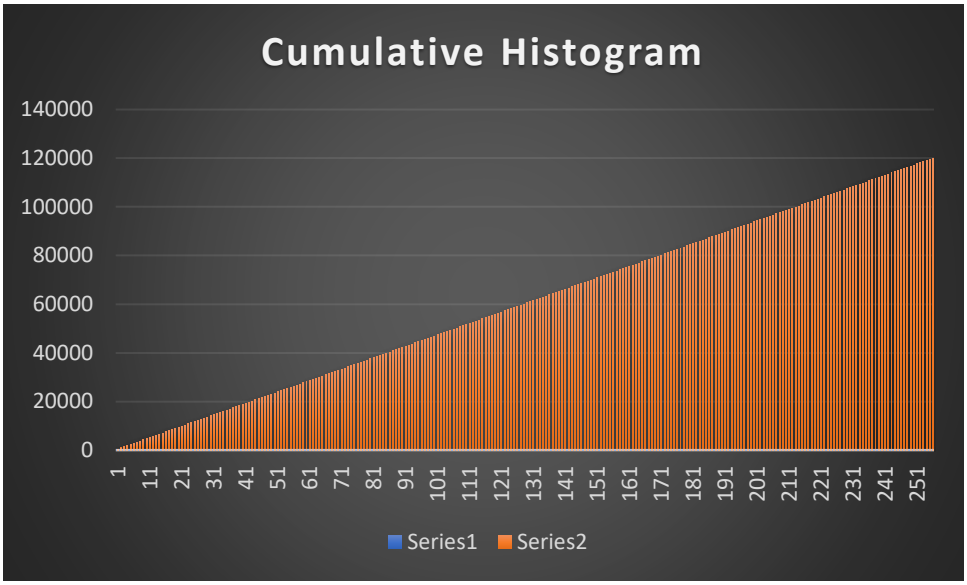


Figure 20: Cumulative Histogram

IV. Discussion

We observe that both images almost give the same result in terms of contrast enhancement but transfer function method is better as making the histogram exactly equal using cdf based terms is not the best way to go forward. We can improve this method by converting the image to HSI format and then perform histogram equalization as it is more visually appealing . We can also perform adaptive histogram equalization in areas where we need to adjust contrast dynamically.

Problem 2: Histogram Equalization

(b) Image Filtering – Creating Oil Painting Effect

I. Abstract and Motivation

To create visually different and appealing images to the human eye, image filters with specific algorithms are created and have been marketed by leading companies like Adobe, SnapChat etc.The Oil paint effect produces a blurry kind of effect and removes distinct color features. Here we attempt to study one such Image filter for producing the "**Oil Painting Effect**" which looks very artistic and also like it is hand-made.

II. Approach and Procedures

Approach for Oil Painting Effect:

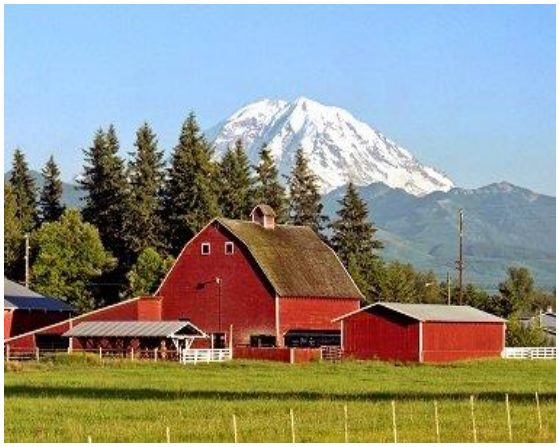
Firstly, We Quantize all colors of the input image to a 64 color image so that each channel R,G & B have only 4 colors. Second step would be to create a window of size N = 3 ,5 ,7 or 11 around a pixel with the pixel being in the centre. Then we choose the most frequent color level of the pixel in the window and replace the centre pixel with that value. This gives us the Oil

painting effect. We can also Quantize all colors of the input image to a 512 color image, here each R,G & B channel has 8 colors each and repeat the windowing algorithm above.

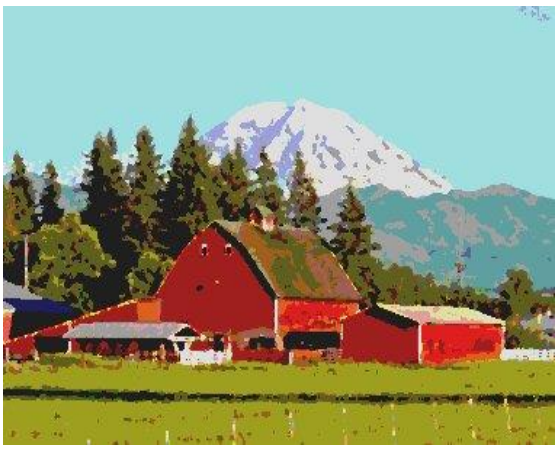
Procedure for Oil Painting Effect:

1. Read input image of 24-bit format.
2. This input image has 256x256x256 = 16777216 colors.
3. We Quantize this image either to 64 colors or 512 colors depending on our requirement.
4. Each R,G & B channel has 4 colors for the 64 color format , we select the 4 color values by making 4 bins from 256 colors and taking the median of the 4 bins. In this case the 4 bins are 0 – 63, 64 – 127, 128 – 191, 192 – 255. The median of these 4 bins give us the 4 Quantized color values which are 31, 95, 159 & 223.
   For 512 color Quantization, we have 8 bins of size 32. The color intervals of the bins are: 0-31, 32 – 63, 64 – 95, 96 – 127, 128 – 159, 160 – 191, 192 – 233, 234 – 255.
   The 8 color values based on Median values of the bin interval are 15, 47, 79, 111, 143, 175, 207 & 239.
5. Next use windowing concept around each pixel to get a window of N*N, where N = 3,5,7 or 11. Get the most frequent value or most recurring value in the window and replace this value with that of the centre pixel value.
6. Store the result in an output buffer and we get the Oil Painting effect.

III. Experimental Results



(a) Original Barn Image



(b) 64 color Quantized, Window N =3

(c) 64 color Quantized, Window N =5

(d) 64 color Quantized, Window N =7

(e) 64 color Quantized, Window N =11

Figure 21: Oil Painting Effect on Barn Image

Figure 22: 64 color Quantization without Oil Painting effect

(a) 64 Quantized color, Window N = 3

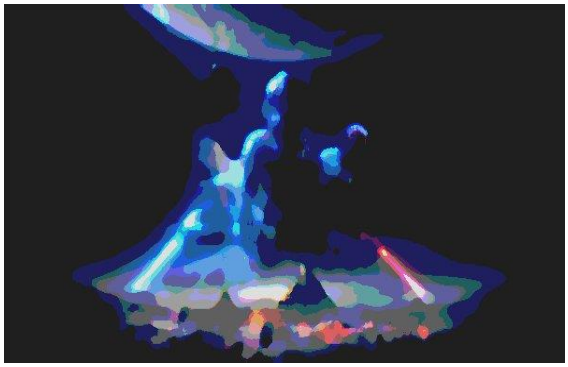
(b) 512 Quantized color, Window N = 3

(c) 64 Quantized color, Window N = 5

(d) 512 Quantized color, Window N = 5

(e) 64 Quantized color, Window N = 7

(f) 512 Quantized color, Window N = 7



(g) 64 Quantized color, Window N = 11

(h) 512 Quantized color, Window N = 11

Figure 23: Oil Painting Effect on Star Wars Image

(a) 64 Quantized color, Window N = 3

(b) 512 Quantized color, Window N = 3

(c) 64 Quantized color, Window N = 5

(d) 512 Quantized color, Window N = 5

(e) 64 Quantized color, Window N = 7

(f) 512 Quantized color, Window N = 7

(g) 64 Quantized color, Window N = 11

(h) 512 Quantized color, Window N = 11

Figure 24: Oil Painting Effect on Trojan Image

IV. Discussion

The Following questions are answered in this discussion:

1. Figure 16 shows 64 color quantization on both (Star wars & Trojan) images without implementing oil painting effect.

   Choosing Threshold:
   For 64 color Quantization, Each R,G & B channel has 4 colors. we select the 4 color values by making 4 bins from 256 colors and taking the median of the 4 bins. In this case the 4 bins are 0 – 63, 64 – 127, 128 – 191, 192 – 255. The median of these 4 bins give us the 4 Quantized color values which are 31, 95, 159 & 223.
   For 512 color Quantization, we have 8 bins of size 32. The color intervals of the bins are: 0-31, 32 – 63, 64 – 95, 96 – 127, 128 – 159, 160 – 191, 192 – 233, 234 – 255.
   The 8 color values based on Median values of the bin interval are 15, 47, 79, 111, 143, 175, 207 & 239.

2. Step 2 of oil painting was done on the Barn, Star wars and Trojan images for different values of N like N= 3, 5, 7, 11. It was observed that higher the window size N better the oil painting effect was observed. So in our case N= 11 gave the best result. We clearly see that for N=11, the Oil paint effect looks visually artistic and as though it was painted by a human.

3. When the input image has 512 colors instead of 64 colors we see that the oil painting effect seems visually better and smoother. The reason for this is, when we have a 512 color image, after we quantize it we get 8 bins and therefore 8 colors in each RGB channel, therefore we have more colors than 64 color image that is quantized to 4 colors in each RGB channel. So once you have more colors the transition from one level color of pixel intensity to another is not too much so the image will be smooth. For example, when we have 512 color image we quantize it to color intensity 15, 47, 79, 111, 143, 175, 207 & 239, here the difference in the color intensity is by a factor 32 whereas in the case of 64 color image we quantize it to color intensity 31, 95, 159 & 223, here the difference in the color intensity is by a factor of 64. Therefore if we observe figure 18 and see (g) and (f) image, we observe that the 512 image has really smooth transition in the clouds regions whereas there is a sudden spike or jump when we see the clouds in the 64 color image of the Trojan image. You can also observe the sword held by the trojan in the 64 color level and 512 color level, we observe that in the 64 color level it is blurred due to merging of colors in the sky and in the 512 color image it is very clear as the sky color does not match with the sword. Hence, we can see that there will be better colorr effect, oil paint effect and also better detail in the 512 color image than 64 color image.

Problem 2: Histogram Equalization

(c) Image Filtering – Creating Film Special Effect

I. Abstract and Motivation

Film effects using image filters often exploit the use of histogram manipulation or histogram matching, mirroring and color space transformation. When all these effects are applied to an image we often see surreal changes that are visually attractive to the human eye. This exercise motivates us to see the mirror effect coupled with negative of an RGB image to a CMY(K) image space and also requires us to understand histogram manipulation to produce contrast. These effects are very popular in the entertainment industry especially in sci-fi movies where in audiences would witness surreal visual effects such as the above.

II. Approach and Procedures

Approach for Creating Film Effect:

We need to perform 3 operations here, firstly we mirror the image, Secondly perform image negation or color space transformation to CMY(K) color space and then manipulate the histogram of the Red, Blue and Green channel via a transfer function. After mirroring we can

also perform histogram matching but I have implemented histogram manipulation using a custom designed Transfer function.

Procedure for Oil Painting Effect:

1. Read Input RGB image.
2. Create a Logarithmic Transfer function to manipulate the histogram of the R, G & B channel.
3. use the log4x function where x=0 to 255 values.
4. then store values of log(4x) where 4x ranges from 0 to 1020 (4*255) into an array.
5. Use formula (log(300+j) – log(300)) / log(555) – log(300) , where j = 0 to 255 in a loop , store these values in a transfer function array.
6. Scale these values in the transfer function array back to 0 to 255 by multiplying by 255.
7. After creating transfer function, convert image to CMY(K) image and multiply each cyan pixel intensity by 1.3, magenta by 0.65 and yellow by 0.75 to change contrast. These values were selected by trial and error.
8. Pass the C,M,Y values to the index of the logarithmic transfer function and store the corresponding values in the output buffer.
9. Then Perform Mirroring on this output buffer and store in the final Output Buffer.
10. Film effect is created.

III. Experimental Results

(a) Original Image

(b) Film Effect Created by above Procedure

(c) Original Image of Girl

(d) Film Effect Created by above Procedure

Figure 25: Film Effect on Original & Girl Image

IV. Discussion

After applying the procedure mentioned above we observe that the algorithm implemented is correct as we get color inversion, mirroring and histogram manipulation in the image. Hence, Figure 19 shows the right results.

Problem 3: Noise Removal

(a) Mix noise in color image

I. Abstract and Motivation

Noise is a random signal that corrupts images if present in large amounts. Noise is typically a high frequency signal whereas the image content is of low frequency. To remove this noise we use a low pass filter to remove noise from images to make it look cleaner. This process is known as denoising. There are several sophisticated algorithms used to denoise images like NLM, PLPCA & BM3D. In this section we are motivated to learn more about the "impulse noise" or the "salt and pepper noise", salt is 1 or white regions in dark areas and pepper is 0 or black regions in bright areas and also about the additive white gaussian noise. Remove the impulse noise we use the median filter and to remove gaussian noise we can use the gaussian filter or a linear filter.

II. Approach and Procedures

Approach for Denoising Mixed Noise in an Image:

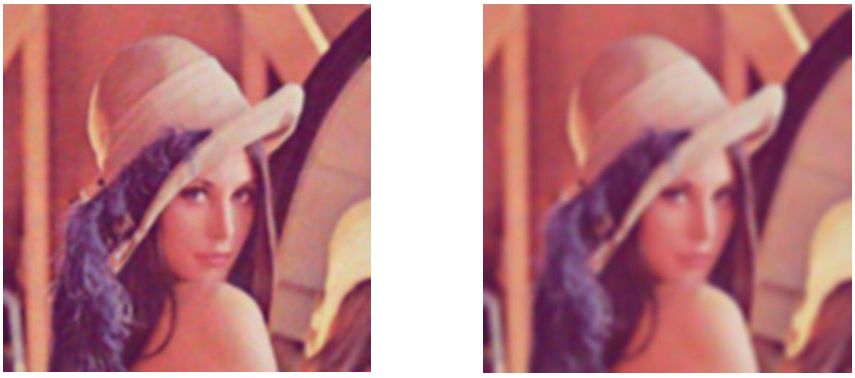
we first discern the types of noise present in the image. In this case there is high amounts of impulse noise and additive white gaussian noise. Therefore we first remove the impulse noise by a median filter and then use a linear filter to remove the additive white gaussian noise. The output would look much cleaner.

Procedure for Denoising Mixed Noise in an Image:

1. Read input image
2. Apply Median Filter to remove salt and pepper noise.
3. Then Apply the Linear filter to remove the white gaussian noise.
4. Calculate PSNR.
5. Get the Output image after passing it through the two filters.

III. Experimental Results

(a) Original Lena Image (b) Filtered Image, N=3 (c) Filtered Image, N=5



(d) Filtered Image, N=7 (e) Filtered Image, N=11

Figure 26: Denoising Lena Image with different Window size

Window size	PSNR For Median & Linear filter	MSE For Median & Linear Filter
N=3	25.42	173.53
N=5	25.3	175.4
N=11	23.40	278.4

Figure 27: PSNR and MSE values for different Window sizes

IV. Discussion

1. (1) All Channels do have the Same noise type i.e they have impulse noise and gaussian noise.
   (2) In the algorithm used above, we have done median and linear filtering for each channel R, G & B for both noise types.
   (3) we use the Median Filter to remove salt and pepper noise (Impulse noise) and then use linear filter to remove gaussian noise.
   (4) No, we cannot cascade the filters in any order, first median filter must be used followed by Linear filter. If we do not do this and perform Linear filtering first and then Median filtering, we get a blurred image as Linear filter will scatter the salt and pepper noise by averaging and after this we wont be able to remove the Impulse noise by median filtering. Hence, We need to do median filtering as it will take the proper median pixel intensity rather than averaging the pixel intensity to a lower value which causes a blurry effect.
   (5) If we observe the figure 20, as the window size N increases from 3 to 11, the image becomes over-smoothened or blurred. Hence, this proves that some amount of noise is necessary for producing sharpness in the image
2. (1) In the method used, we first remove salt and pepper noise using median filter and after that we cascade it with a linear filter which removes some gaussian noise. The PSNR values are mentioned above in the experimental results.
   (2) there some shortcomings when it comes to using the above procedure. We observe that there is some gaussian noise that we can still observe after filtering and hence we don't get the best Peak-signal to noise ratios and also for increasing window sizes N , we observe a over-smoothened or blurred effect which is not desirable. Hence for higher window sizes image clarity reduces which shows that some noise is needed for image sharpness.
   (3) A way to improve denoising would be either to user Patch based PCA or to get even better results we can proceed with the BM3D algorithm.

(b) **Principal component analysis (PCA)**
Source: Reference[3]

I. Abstract and Motivation

Principal component Analysis is a dimensionality reduction algorithm that is used for noise filtering to extract meaningful data. In this section we attempt to understand the Patch based local PCA which is a powerful algorithm used for denoising images.

II. Approach and Procedure

Approach and Procedure for PLPCA

1.Divide Image into Patches of small subsets.

2. Put a sliding window of size WsxWs inside the patch and perform PCA Functionality to remove pixels based on a threshold and remove noise.

3. In the overlapping patches where pixels are common perform averaging and Replace.

III. Experimental Results:

Window Size (N)	Threshold	Delta	PSNR (DB)
7	55	11	32.37 -- highest
7	137.5	11	29.49
11	151	11	32.31
5	151	11	32.14

For sigma value = 20

Figure 28: Table for different experimental values for PLPCA

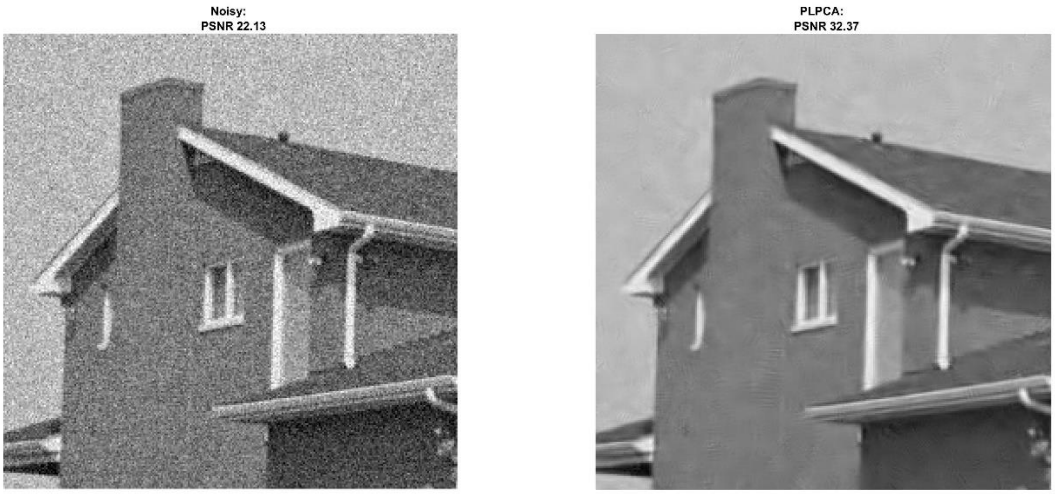


Figure 29: Best Result for different parameters in PLPCA (PSNR : 32.37 db)

IV. Discussion

1. PCA helps in dimensionality reduction by choosing another basis that is a linear combination of the Initial basis of the data. This change in basis which is orthogonal in nature helps identify the proper data and discards or decomposes redundant or noisy data by making all the small values of the components in the patch of interest. This property of PCA helps in filtering of noisy data, that is why we choose this approach.
   The Principal Components are the amount or number of pixels of particular intensity in the particular patch.
   Choosing the number of components in image denoising depends on the noise variance and hence will be different for different noisy images.

2. In Patch Based Local PCA we divide a patch into a number of smaller sets and perform PCA in a loop like fashion. So now the single best basis which is chosen to re-express our image data is applied to each of the smaller sets and hence to a local region of the image. Limited patches for training and computational complexity are severe draw back of the Patch Based Local PCA. There is a sliding window of size Ws x Ws which overlaps while moving from one patch to another therefore there will be pixels common to the overlapping section, for this we average out the estimates of the pixels in the noise free patches and project again the pixel from the many common patches that exist and carry out averaging for the estimate of the pixels. In the computational part hard thresholding is also done to remove out any noisy components in the patches. This is the procedure adopted by Patch-Based Local PCA.
3. Results are tabulated above in the experimental section. For Window size N=7, Threshold = 55 & delta = 11.
4. Applying the Median and Linear filter with N=5, we get PSNR=28.3 dB . We see that with PLPCA we get better PSNR values and hence better image. PCA works well as it

reduces dimensionality and hence helps filter out noise much better than manual pixel manipulation using traditional filters.

Figure: House for Median and Linear Filter

(c) **Block matching and 3-D (BM3D) transform filter**
Source : Reference[2]

I. Abstract and Motivation

Block Matching 3-D is by far the most efficient algorithm for reducing noise in images. Latest research shows how BM3D integrates with convolutional neural networks for better performance denoising. Hence, it is in our interest to learn this powerful algorithm.

II. Approach and Procedure

Approach and Procedure for BM3D

1. Locate 4 matching blocks.
2. Put it into a vertical stack.
3. Perform DCT.
4. Truncate high frequency.
5. Perform Inverse DCT.
6. Put Back to Stack.

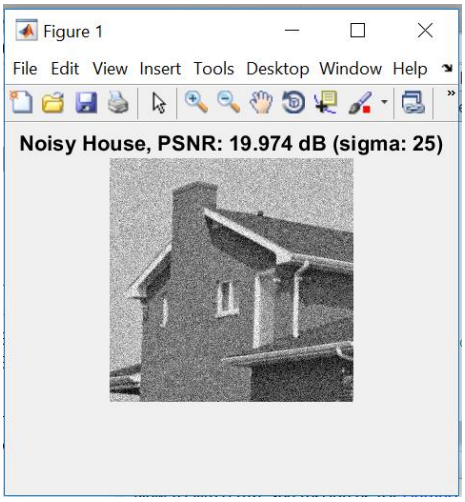
III. Discussion

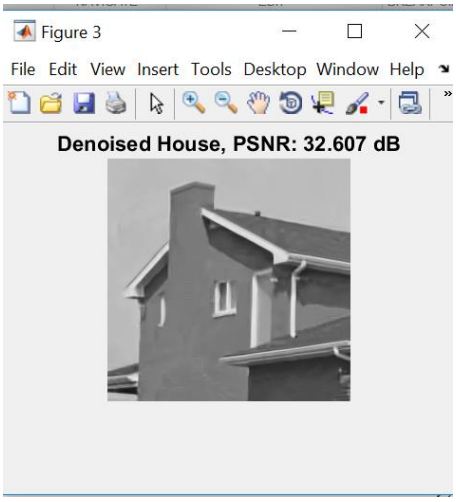
1. BM3d locates neighbouring similar blocks and puts them into a vertical stack. There it performs DCT and uses information to put them in one 3d stack. After block matching, hard thresholding is done to remove noise. After truncation of these high frequencies, inverse DCT and passed through the weiner filter is done and the pixels are averaged and put back to the stack.

2. block matching cuts down noise. Step 2 increases the PSNR value and weiner filter produces much better results than hard thresholding.

3. BM3D is both spatial domain & frequency domain filter.

4. visually BM3D is much better than PLPCA. Results are shown below





References:

[1]:https://ia802707.us.archive.org/23/items/Lectures_on_Image_Processing/EECE_4353_15_Resampling.pdf

[2] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform- domain collaborative filtering," Image Processing, IEEE Transactions on, vol. 16, no. 8, pp. 2080–2095,2007

[3]http://josephsalmon.eu/code/index_codes.php?page=PatchPCA_denoising