**Assignment 1**

CP467: Image Processing & Recognition

Professor: Dr. Zia Ud Din

Due Date: Saturday September 30, 2023. 11:59 p.m.

Chandler Mayberry 190688910

Table of Contents

[Introduction 3](#_Toc146997269)

[Image Interpolation 3](#_Toc146997270)

[Image Shrink 3](#_Toc146997271)

[Nearest Neighbor Interpolation 4](#_Toc146997272)

[Bilinear Interpolation 5](#_Toc146997273)

[Bicubic Interpolation 6](#_Toc146997274)

[Comparison of Results 6](#_Toc146997275)

[Implementation Issues 8](#_Toc146997276)

[Point Operations 8](#_Toc146997277)

[Negative of Image 8](#_Toc146997278)

[Power-Law Transformation 9](#_Toc146997279)

[Contrast Stretching 10](#_Toc146997280)

[Histogram Processing 11](#_Toc146997281)

[Histogram Equalization 11](#_Toc146997282)

[Histogram Specification 13](#_Toc146997283)

# Introduction

As described in the assignment instructions, the purpose of this assignment was to give in-depth low-level insight into the functions of image interpolation, point operations, and histogram contrast methods.

All the code for this assignment may be run using the a01\_190688910.py file in the source code folder. Running this file will perform the operations and display the resulting images to the user using python OpenCV.imshow() method. ***This was the only other high-level function used from the OpenCV library.***

**Note**: The saved images with extensions “.tif” will not open and display on my machine using Windows default photo viewers and editors. However, they do display correctly when running the code and do not affect the overall demonstration of the image manipulations in action.

# Image Interpolation

## Image Shrink

In this operation to shrink the image ½ in both dimensions (for a total reduction in size of ¼) both the row and column pixels of the image were averaged from pair values to a single value. The process I chose to do this was to go through every row pixel value in the image matrix, add the pair values, and divide by the M\*N (denoted by DIV) dimensions to get the new greyscale value in place. The same operation is made on the column values by transposing the matrix and performing the same width operation. See the code below:

A screen shot of a computer program

Description automatically generated

Figure 1: Image Size Reduction Method

## Nearest Neighbor Interpolation

Nearest neighbor interpolation is a simple interpolation technique for the resizing of digital images. The process of nearest neighbor is in the name, the new pixel values being generated take the value of its adjacent original pixel as its own. The implementation of nearest neighbor interpolation of the cameraman image back to its original size was a relatively easy, involving moving through the matrix of grayscale values and simply duplicating each pixel value and placing it into the updated matrix. See the code below:

A computer screen shot of code

Description automatically generated

Figure 2: Nearest Neighbor Implementation

## Bilinear Interpolation

Bilinear interpolation involves the process of taking the average value of adjacent pixel pairs with respect to the distance/weight of neighboring values. In action, bilinear interpolation fills the new pixel values when increasing the size of the image with averaged gray values for a smoother (less jagged) image. My implementation of this starts with the rows of the image, generating two new pair pixel values between two original pixel values, and taking their average result. Once each new pixel row value is generated, new arrays need to be made in the matrix for the new column pixels. The new rows take the average of the above and below row values and place the result in the new matrix, which is placed between above and below in the final matrix. See the code below:

A screen shot of a computer program

Description automatically generated

Figure 3: Bilinear Interpolation Implementation

## Bicubic Interpolation

Bicubic interpolation takes the approach of using the 16 adjacent neighbors in consideration of the new pixel value being inserted. The polynomials at each control point determine the new pixel value. This method results in an image that is smoother than bilinear and nearest neighbor, as it considers more pixel values to find the closest color match in the image. My final implementation uses the Python OpenCV built-in method to create the scaled-up version of the cameraman image. See the code below:

A computer code on a black background

Description automatically generated

Figure 4: Bicubic Interpolation Implementation

## Comparison of Results

Comparing the results, we can see that none of the image techniques captures the clarity and grayscale contrast of the original image:

1. Nearest Neighbor resulted in jagged lines when scaling up the image. It’s clear to see that this was the cheapest computational option due to its simplicity but results in a poor looking image.

A person using a camera

Description automatically generated

Figure 5: Nearest Neighbor Cameraman

1. Bilinear interpolation resulted in an image with slightly more averaged gray values (can be seen at the edges of the man). However, I do not believe that my implementation of this method was correct as it seems to result in a worse image than nearest neighbor interpolation.

A person using a camera

Description automatically generated

Figure 6: Bilinear Cameraman

1. Bicubic interpolation results in the best image out of the three but still does not capture the detail of the original image. It does make a lot more accurate value guesses for the new pixels than the other options, as can be seen from the clarity of the image edges.

A person using a camera

Description automatically generated

Figure 7: Bicubic Cameraman

## Implementation Issues

As mentioned in the comparison of results, I ran into implementation issues regarding bilinear interpolation as I am not convinced my implementation is completely correct looking at the results. I was unsure of how to approach the problem, whether to create the new pixel values in the rows or columns first, or whether to do each at the same time and place the values in the resulting matrix. After several attempts, I landed with my current solution as it makes the most sense when comparing it to the theory in the lecture slides and on paper from my understanding.

# Point Operations

## Negative of Image

Taking the Negative of an image is a rather simple operation, its goal is to simply make the equivalent reverse of an image’s pixel values. The equation s = L-1-r is the representation of the logic in taking the negative of an image. The implementation of the negative image method goes through all the pixel values in the matrix performing this operation on the values. See the code and result below:

A computer screen shot of a program code

Description automatically generated

Figure 8: Negative Implementation

A screenshot of a cellphone

Description automatically generated

Figure 9: Negative Cameraman

## Power-Law Transformation

Power-law transformation on an image is used primarily to increase the range of gray level values in an image. This operation is denoted by the equation s = crY, where y is the gamma value. The implementation of this technique is simple, apply the equation above on all the pixel values while hard coding the gamma value to get the desired output. The gamma values I tested ranged from 0 to 10, choosing random values in between. I chose my final output to be 0.58 as I found it produced an image that wasn’t too bright or too dim on my monitor (the higher the value the darker, the lower the value the brighter). See the code and result below:

A screen shot of a computer code

Description automatically generated

Figure 10: Power Law Implementation

A person using a camera

Description automatically generated

Figure 11: Power Law Cameraman

## Contrast Stretching

Contrast stretching is a piecewise linear function that is used to increase the dynamic range of the gray values in an image and may be particularly useful on photos that are too bright or dark with a low dynamic range. The implementation of this technique is as follows, each pixel in the matrix value is subtracted by the smallest gray value in the image and divided by the largest gray value in the image subtracted by the smallest. See the code and result below:

A screen shot of a computer code

Description automatically generated

Figure 12: Contrast Stretching Implementation

A person using a camera

Description automatically generated

Figure 13: Contrast Stretching Cameraman

# Histogram Processing

## Histogram Equalization

Histogram equalization is used to increase the contrast of an image by essentially reducing the amount of gray level values by assigning them uniformly to the most frequent intensity values in the image. A brief description of the implementation is as follows; a new array is created using the pixel gray level values to act as a histogram denoting the total gray level values from index 0-255. An equalized histogram is then created from this by creating a running total of the number of pixels from index 0 to 255. Then all the values within the equalized histogram are normalized, multiplied by (L-1), and rounded to create the final CDF values. Once the equalized histogram is complete, all the values within the image matrix are used as indexes in the equalized histogram to attain their new final values. See the code and result below:

A screenshot of a computer program

Description automatically generated

Figure 14: Histogram Equalization Implementation

## A person with a mustache holding his hands together Description automatically generated

Figure 15: Equalized Einstein

## Histogram Specification

Histogram specification has the same goal of increasing the contrast of an image as does histogram equalization but does this by matching an input image’s contrast to a target image’s contrast (gray) levels. The implementation of this is relatively straight forward from histogram equalization, first create two equalized histograms of the original and target image gray values. Then the equalized values from the original image histogram are mapped to the gray level value of the target image histogram and stored in a matching array. This matching array is then applied to the original image matrix to attain the new gray level values for the image. See the code and result below:

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A computer code on a black background

Description automatically generated

Figure 16: Histogram Specification Implementation

A x-ray of a person's chest

Description automatically generated

Figure 17: Original Image

A chest x-ray of a person

Description automatically generated

Figure 18: Target Image

A x-ray of a person's chest

Description automatically generated

Figure 19: Output Image