NITTE EDUCATION TRUST | N.M.A.M. INSTITUTE OF TECHNOLOGY
(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)
Nitte – 574 110, Karnataka, India
(ISO 9001:2015 Certified), Accredited with 'A' Grade by NAAC
☎: 08258 - 281039 – 281263, Fax: 08258 – 281265

Department of Computer Science and Engineering

B.E. CSE Program Accredited by NBA, New Delhi from 1-7-2018 to 30-6-2021

Report on Mini Project

Pixel Editor

Course Code: 16CSE82

Course Name: Image Processing

Semester: 6th Section: C

Submitted To,

Dr. Aravinda C V

Submitted By:

Name: Rahul D Shetty USN: 4NM16CS111

Date of submission:

Signature of Course Instructor

NITTE EDUCATION TRUST | N.M.A.M. INSTITUTE OF TECHNOLOGY
(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)
Nitte – 574 110, Karnataka, India
(ISO 9001:2015 Certified), Accredited with 'A' Grade by NAAC
☎: 08258 - 281039 – 281263, Fax: 08258 – 281265
Department of Computer Science and Engineering
B.E. CSE Program Accredited by NBA, New Delhi from 1-7-2018 to 30-6-2021

Department of Computer Science and Engineering

CERTIFICATE

Certified that the Mini Project work entitled ***Pixel Editor*** *carried out by* ***Rahul D Shetty 4NM16CS111*** *bona fide student of* ***NMAM Institute of Technology**, **Nitte*** *has been carried out satisfactorily. The Mini Project report for the Subject* ***Image Processing**, has been prepared as per the prescribed format.*

Name & Signature of Guide	**Name & Signature of HOD**
Dr Aravinda C V	Dr. K R Udaya Kumar Reddy
Assistant Professor Gd III	Head of the Department
Department of CSE	Department of CSE
NMAMIT, NITTE	NMAMIT, NITTE

Name of the Student	USN	Signature	Date
Rahul D Shetty	4NM16CS111		

ACKNOWLEDGEMENT

We believe that our project will be complete only after we thank the people who have contributed to make this project successful.

First and foremost, our sincere thanks to our beloved principal, **Dr. Niranjan N. Chiplunkar** for giving us an opportunity to carry out our project work at our college and providing us with all the needed facilities.

We express our deep sense of gratitude and indebtedness to our guide **Dr. Aravinda C V**, Assistant Professor Gd III, Department of Computer Science and Engineering, for his inspiring guidance, constant encouragement, support and suggestions for improvement during the course of our project.

We sincerely thank **Dr. K.R. Udaya Kumar Reddy**, Head of Department of Computer Science and Engineering, Nitte Mahalinga Adyantaya Memorial Institute of Technology, Nitte.

We also thank all those who have supported us throughout the entire duration of our project.

Finally, we thank the staff members of the Department of Computer Science and Engineering and all our friends for their honest opinions and suggestions throughout the course of our project.

Rahul D Shetty 4NM16CS111

ABSTRACT

Nowadays “Image Processing” normally used by wide range of applications and in different types of electronics like computers, digital cameras, mobile phones etc. The image properties can be changed with the least investment such as contrast enhancement, borders detection, intensity measurement and apply different mathematical functions to enhance the imagery. Even though these methods with dump, but understanding the fundamental values behind the effortless image processing routine is rare. The method of image processing is used to do some processes on a picture like an image enhancement or to remove some functional data from the image.

Image processing is one kind of signal processing, where the input is a picture, as well as the output, are feature or characteristics allied with the image. The goal of this project is to build a multipurpose image editing tool which contains all the important image enhancement, morphological operations. The tool is mainly focused on faster image processing by making use of latest technological features provided by the programming language C#. All the processing code is designed from scratch and in order for faster calculation of the convolution operations, a special code segment in C# called *unsafe* is used.

There are many modern-day tools available in market which does most of the operations that is included in this project, but most of them require a high demanding PC specification and use up the most of the memory while being executed. This application focuses to provide all the high level or complex operations at a real low cost in terms of space and time. The project makes use of various morphological transformations, image enhancement techniques, various filtering options, thresholding and much more.

These operations are the basic for various image processing applications which is used in real time systems like CCTV face recognition where different noises can be reduced by applying particular filters. The project provides a way for users to switch between multiple images and work on them individually and independently.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Basics of Image Processing

In computer science, Digital Image Processing is the use of computer algorithms to perform image processing on digital images. As a popular subcategory of Digital Signal Processing, in Digital Image Processing has major advantages over Analog Image processing. It allows wide range of algorithms to be applied to the input data and avoid problems such as the build-up of noise and signal distortion during processing. Here we can make use of much complex algorithms and hence can offer both Sophisticated performance at simple tasks and implementations of methods which would be impossible by analog means.

Having said the above main features of using Digital Image Processing, the following project uses the same for all the mathematical computations on image. The main objective of this project is to build an easy to use Image editing tool for multipurpose applications. The application provides a basic layout to load and observe the changes that are performed on the image. Some of the key operations provided by the applications include Gray scaling, Image Blurring or Smoothening, Image Rotations and Flipping, Edge detection, Various Filtering Techniques, Histogram Equalization for both color and black/white images and morphological operations like Open, Close, Erosion and Dilation.

Most of the operations mentioned above involves simple and complex operations on image. For most of the filtering techniques, a Convolutions is used for processing the image pixel by pixel.

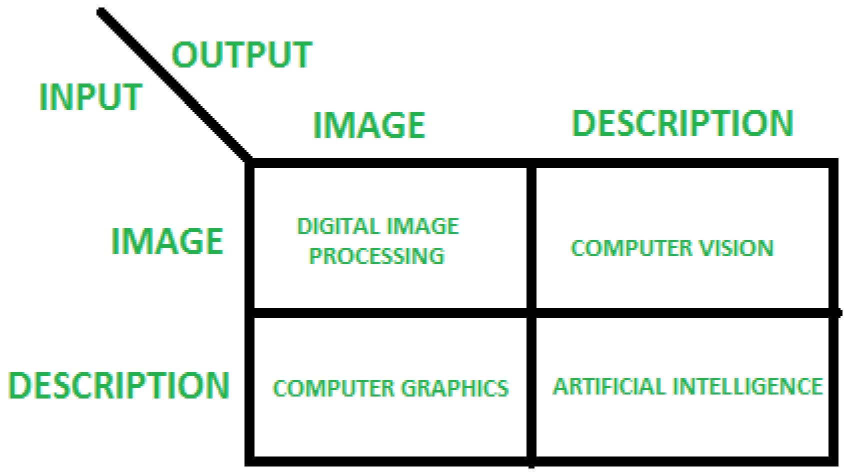


Figure 1: Different Fields of Image Processing

Figure 1 shows how we categorize the applications of image processing based on inputs and outputs. This projects solely focuses on the first part where both the input and outputs are images.

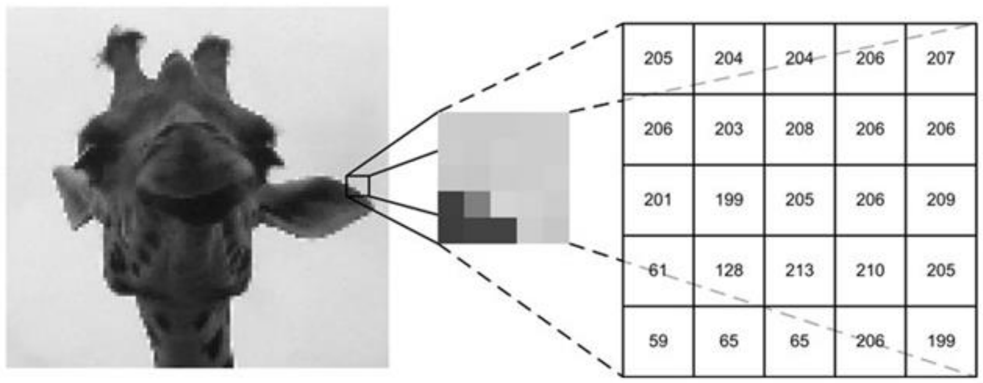
1.2 Image Representation

Pixel is the smallest element that can be represented on a computer monitor and each pixel or picture element consist of a value that represent the intensity at that point. A Digital Image is a two-dimensional arrangement of these pixels in tabular form where each and every points in the table represents the intensity of the pixels at that coordinates which will be similar to that of when represented in special coordinates. Since computer can only understand 0’s and 1’s, an image is stored in computers as matrix of these values ranging from 0 to 255 for each Red, Green and Blue color channels.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & f(0,2) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & f(1,2) & \dots & f(1,N-1) \\ \vdots & \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & f(M-1,2) & \dots & f(M-1,N-1) \end{bmatrix}$$

Figure 2: Matrix Representation of Image

Figure 2 shows how an image is represented in terms of a matrix of size M x N where it contains M rows and N columns. Each cell contains a value that ranges from 0 to 255. For Black and White images, there is only one color channel that shows the intensity at each point while Color images are having the same but across Red (R), Green (G) and Blue (B) channels. The application is designed to work on both the image channels and get a proper output in terms of image again. Figure 3 shows how a 5 x 5 region of a larger image will contain different pixel values.



205	204	204	206	207
206	203	208	206	206
201	199	205	206	209
61	128	213	210	205
59	65	65	206	199

Figure 3: Pixel Values

Chapter 2

PROBLEM STATEMENT AND OBJECTIVES

2.1 Problems

In the recent market there are various tools or applications that does most of the previously mentioned operations but most of these types of applications requires higher system configuration in terms of CPU, memory and space. And some of these applications even require special hardware like a graphical processing unit (GPU) for running it smoothly in the systems. This is a major drawback, as most systems still run on older CPU and do not use any GPU as they are expensive to purchase.

There are tools like MATLAB which performs even complex and custom operations on images. The only constraint here is that the user needs to know the programming language associated with this tool in order to perform any operations and this becomes cumbersome for majority of normal users. Also the tools available demands a high storage space. This is an another issue since space available in the system in limited and one cannot provide all the space for only one application.

Most users are in need of an application that does complex image operations in ease and gives them in understandable format. Also these users do expect the output to be shown in reasonable amount of time. Hence there is a major issue with tradeoffs on all these factors like speed, memory usage, storage required and easy to use interface.

2.2 Objectives

The main objective or goals of this project is to overcome most of the problems stated earlier. Along with those, the application provides various operations more efficiently. Some of the operations that were the primary goals of this are stated below:

1. Converting a RGB color image to a gray scale image.
2. Filtering techniques to provide image smoothening.
3. Filtering techniques to identify edges in a given image.
4. Histogram Balancing or Equalization for both Color and gray scale Images.
5. Image flipping (Horizontal and Vertical).
6. Image Rotation (Clockwise and Anti-Clockwise).
7. Loading Multiple Images at a time.
8. Saving changes to the images.
9. Undo any operation performed on the image.
10. Brightness and Contrast adjustments.
11. Image Thresholding.
12. Morphological Operations like Erosion, Dilation, Open and Close.
13. Faster convolutional processing.
14. Image Enhancements with various filters.

Chapter 3

PROJECT

3.1 Methodology

3.1.1 Convolution

The major part of the project is applying the convolution across the entire image and obtain the result in terms of an image. In order to apply this, we need to traverse each and every pixel and also at each level of the pixel we need to identify all the 8 neighbor pixels in order to apply the given kernel. There are various algorithms involved in doing this operation, but for this project the basic strategy of taking two loops to traverse each and every element in the given image and applying the convolution by going across the window of that kernel.

Figure 4 represents the operation of applying the filter across a 3 x 3 region of the first 3 rows and first 3 columns. Based on application we can apply different kernel to obtain different outcomes. In the below figure an Emboss filter for image enhancement is used. Every filters have their own weights associated with them which are important for finding the new value of the pixel at a location. In the below example the weights associated are +4 and -4 across the end diagonals. Most kernels have equal weights of positive and negative value which on added generates a sum 0. This is done to average the overall effect on applying the kernel.

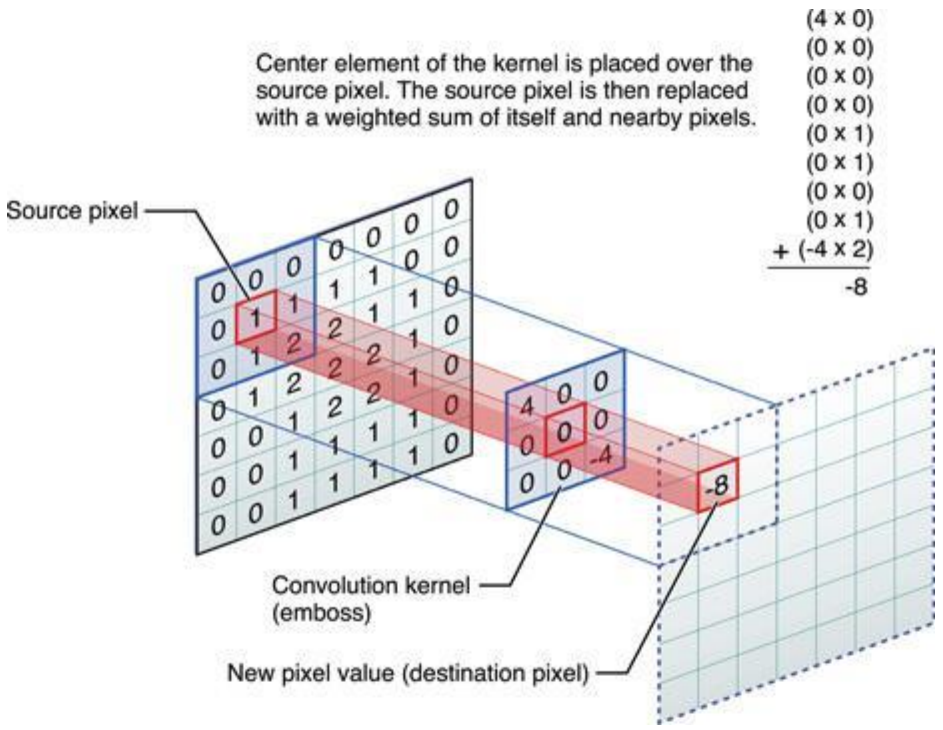
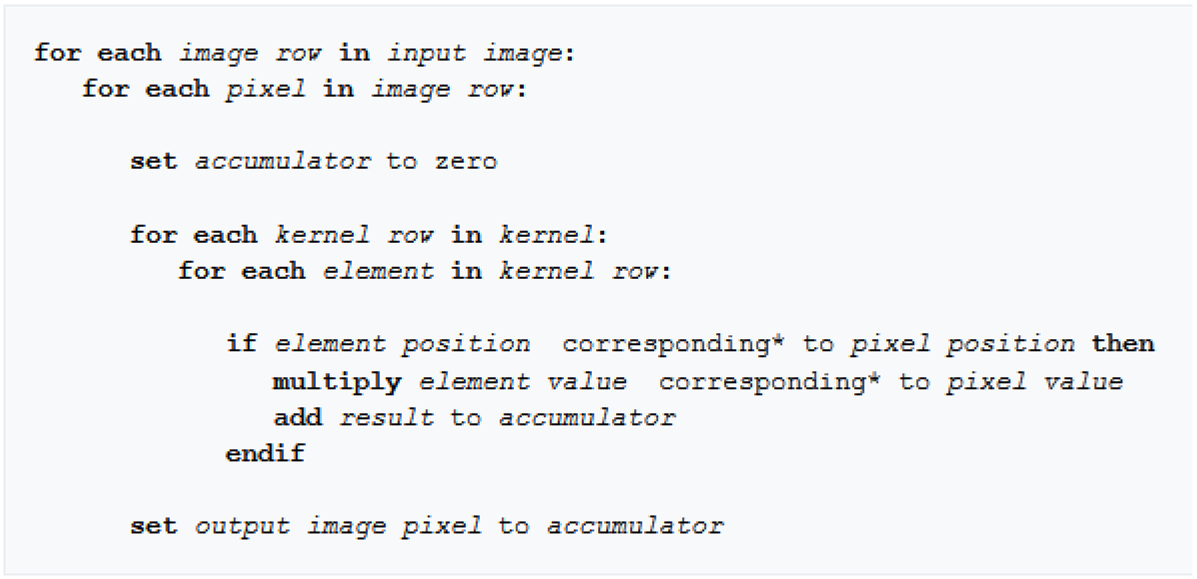

Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution kernel (emboss)
New pixel value (destination pixel)

Figure 4: Applying Convolution

A kernel or Mask is a filter which are used for spatial filtering. There are different types of these filters, but majorly they are categorized as Linear and Frequency Domain Filters. Based on purpose usually filters are applied in order to do either blurring or smoothening otherwise for finding or enhancing the edges. Blurring cancels out all the noise present in the image which is a major part in many Image Application Domain while Edge detection is used for Object Recognition similar applications.

In order to perform any convolution general logic used is:

- Append additional 0’s or padding to the given image.
- Flip the mask (horizontally and vertically) only once
- Slide the mask onto the image.
- Multiply the corresponding elements and then add them
- Repeat this procedure until all values of the image has been calculated.



```
for each image row in input image:
   for each pixel in image row:

      set accumulator to zero

      for each kernel row in kernel:
         for each element in kernel row:

            if element position  corresponding* to pixel position then
               multiply element value  corresponding* to pixel value
               add result to accumulator
            endif

      set output image pixel to accumulator
```

Figure 5: Convolutional Processing

Figure 5 shows the pseudo code in order to apply the kernel of some size to our image. There are 5 schemes used to handle convolutions involving edge pixels.

<u>Extend</u>:
The nearest border pixels are conceptually extended as far as necessary to provide values for the convolution. Corner pixels are extended in 90° wedges. Other edge pixels are extended in lines.

<u>Wrap</u>:
The image is conceptually wrapped (or tiled) and values are taken from the opposite edge or corner.

<u>Mirror</u>:
The image is conceptually mirrored at the edges. For example, attempting to read a pixel 3 units outside an edge reads one 3 units inside the edge instead.

Crop:
Any pixel in the output image which would require values from beyond the edge is skipped. This method can result in the output image being slightly smaller, with the edges having been cropped.

Kernel Crop:
Any pixel in the kernel that extends past the input image isn't used and the normalizing is adjusted to compensate.

$$\left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}\right)[2,2] = (i \cdot 1) + (h \cdot 2) + (g \cdot 3) + (f \cdot 4) + (e \cdot 5) + (d \cdot 6) + (c \cdot 7) + (b \cdot 8) + (a \cdot 9).$$

Figure 6: Mathematical Outcomes

In Figure 6 there is a pixel values represented by alphabets and kernel is shown by digits. According to the algorithm the resultant pixel value is computed as shown in Figure 6. All the filter techniques use the same principal but with different values of kernel in order to get a particular result.

3.1.2 Gray Scaling

Additional features included in this application is gray scaling which is to convert a color image to a black and white image. There are various techniques to perform this operation which may include applying a formula or by applying a mask. The given program uses the mathematical approach to find the value of the newer pixels.

$$Y' = 0.299R' + 0.587G' + 0.114B'$$

Figure 7: Calculation of Gray scale given RGB values

In order to perform gray scaling following logic is used:

- Create a new temporary result matrix of same size as the image.
- Traverse across each pixel one by one.
- Obtain R’, G’, B’ values which correspond to Red, Green and Blue pixel value at that pixel point.
- Apply the formula in Figure 7 to find Y’ which is the required output and fill it into the new matrix.
- Repeat the above 4 steps until no more pixels can be traversed.

In most applications gray scaling provides an efficient way to perform various tasks and then map the result back to color images. Gray scale image processing is faster when compared to color images as we only have 1 channel in a gray scale image. For a black and white images each pixel represents the intensity of the light at that position in spatial domain.

3.1.3 Morphological Operations

Here we try to apply a structuring element to an image in order to obtain a particular result. Each of these structuring elements is similar to that of kernel but under this we do some mathematical operation such as XOR, BITWISE_OR, AND, OR and finding the minimum or maximum elements in the window.

Dilation:
The value of the output pixel is the *maximum* value of all pixels in the neighbourhood. In a binary image, a pixel is set to `1` if any of the neighbouring pixels have the value `1`. Morphological dilation makes objects more visible and fills in small holes in objects. Figure 8 below shows this operation.

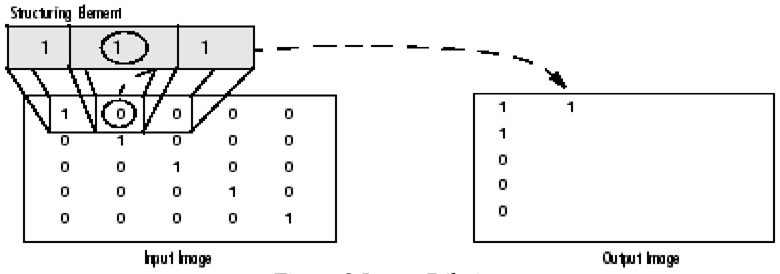


Figure 8:Image Dilation

Erosion:
The value of the output pixel is the *minimum* value of all pixels in the neighbourhood. In a binary image, a pixel is set to `0` if any of the neighbouring pixels have the value `0`. Morphological erosion removes islands and small objects so that only substantive objects remain.

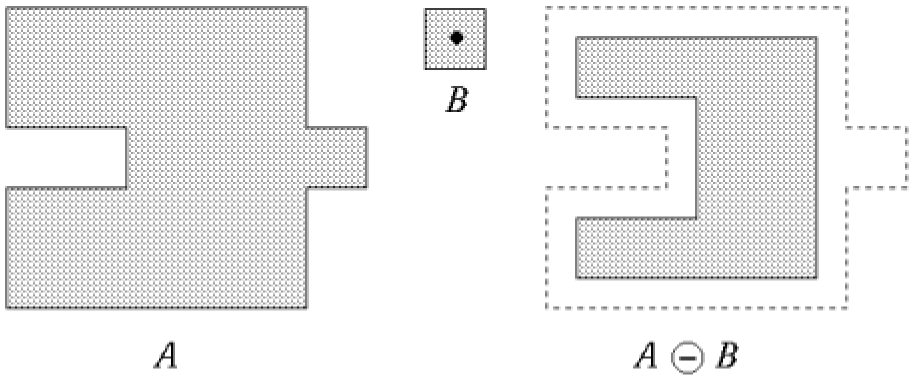


Figure 9: Image Erosion

Open:
The operation of erosion followed by dilation using some specified structuring elements is known as Open operation.

Close:
The operation of dilation followed by erosion using some specified structuring elements is known as Open operation.

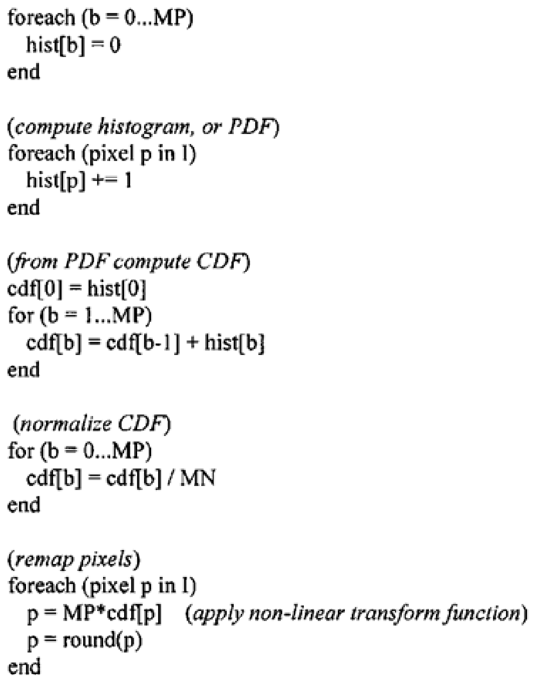
3.1.4 Transform Operation

Image transforms are used to change the orientation or for finding the mirror images across some axis. Rotation and Flipping are the two most operations used in this technique. Rotations involves changing the image by rotating it in clockwise or anti-clockwise direction. These operations involve transposing the matrix to change from M x N to N x M order. Flipping is another transform operation where we take the mirror image across X or Y axis. All these operations involve accessing each pixel and then positioning them accordingly.

3.1.5 Histogram Equalization

Histogram equalization is a technique for adjusting image intensities to enhance contrast. This method usually increases the global contrast of many images, especially when the usable data of the image is represented by close contrast values. Through this adjustment, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values.

The method is useful in images with backgrounds and foregrounds that are both bright or both dark. In particular, the method can lead to better views of bone structure in x-ray images, and to better detail in photographs that are over or under-exposed. A key advantage of the method is that it is a fairly straightforward technique and an invertible operator. So in theory, if the histogram equalization function is known, then the original histogram can be recovered. The calculation is not computationally intensive. A disadvantage of the method is that it is indiscriminate. It may increase the contrast of background noise, while decreasing the usable signal.



```
foreach (b = 0...MP)
  hist[b] = 0
end

(compute histogram, or PDF)
foreach (pixel p in I)
  hist[p] += 1
end

(from PDF compute CDF)
cdf[0] = hist[0]
for (b = 1...MP)
  cdf[b] = cdf[b-1] + hist[b]
end

 (normalize CDF)
for (b = 0...MP)
  cdf[b] = cdf[b] / MN
end

(remap pixels)
foreach (pixel p in I)
  p = MP*cdf[p]    (apply non-linear transform function)
  p = round(p)
end
```

Figure 10: Algorithm for Histogram Equalization

Figure 10 shows the algorithm used for computing the new pixel values after evening out the original distribution. For black and white images, it is straightforward to apply this to one of the channels but for color images different methods are considered. In our application we apply equalization over all the 3 channels and then combine to get the final result.

3.2 Pixel Editor

Pixel Editor is a multipurpose image processing tool for Windows platform. It is designed in such a way that to provide processing of higher or complex convolutional operations on any image at faster rate and making efficient use of available resources. Pixel Editor provides users with easy to use and neat Graphical User Interface designed by using Metro Theme.

This application provides users with various operations like gray scaling, image smoothening using different types of filters, edge detection based on filters, image flipping and rotating operations, morphological operations like closing, opening, erosion and dilation. A histogram is shown for different color channels individually and we can perform Equalization of these using this application.

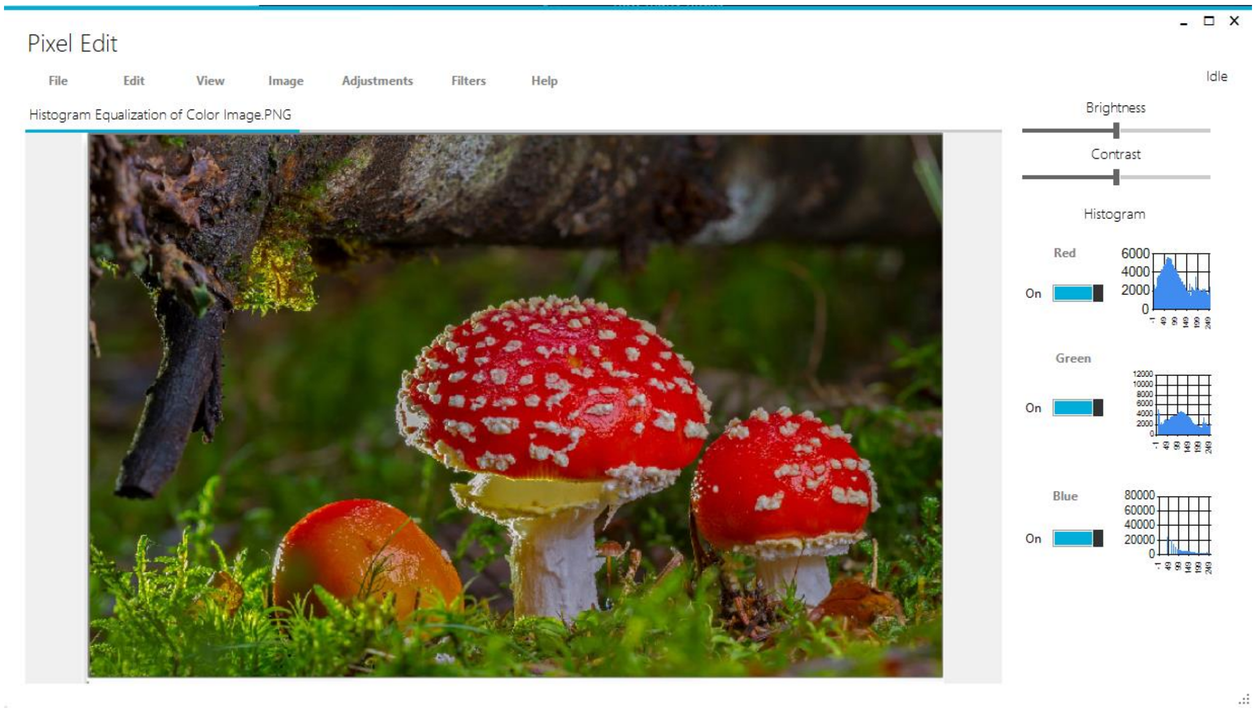


Figure 11: Program GUI

Figure 4 shows the look of the final application. The above UI is designed based on Metro UI theming concepts to make it easy for end users to operate. Various image processing features are available from different Menu’s provided in the application.

Pixel Editor also allows users to open up multiple images and work on them independently and switch between them at any time. All the operations are saved in stack so that the users can go back to previous state of the image if any wrong operation was performed. This makes it easier for users to work with the image.

3.3 Implementation

The entire source code is designed in C# programming language. This language was preferred because of its easy to use GUI editing toolkit. The Metro UI in Figure 11 uses Metro Mash Theme. The application is a Windows Form application which can be executed on all Windows system that support .NET Frameworks 4.6 and above. All the components used are provided by the Metro Mash and Windows Toolbox.

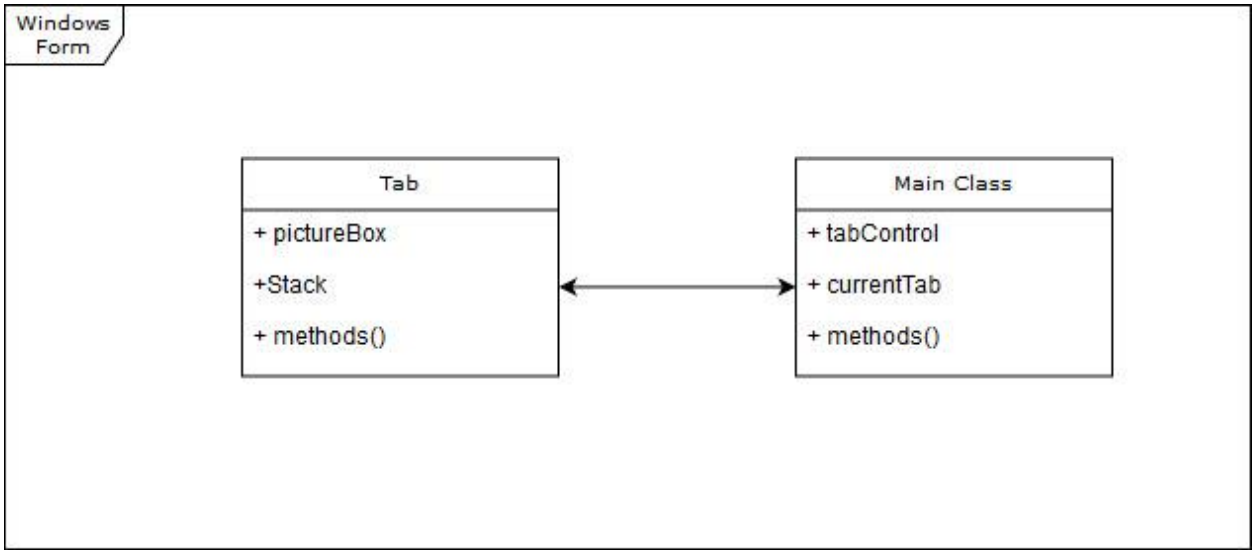


Figure 12: General Structure

Figure 12 shows how the entire code is divided as. Basically we have 2 prime modules, Main and Tab. Main class controls how the entire windows form looks like and we define events under this class. Primarily this class form contains a Tab Control Layout which is used for providing the users with loading multiple images and working on them independently. The class contains an insert Tab method which is used to create a new Tab and add it to the list. Each Tab when created includes various variables and methods, out of which pictureBox element is used to display the image onto the screen.

C# uses Object Oriented Programming concepts which helps us in reusing the already written code. Creation of GUI from scratch is easy and fast when compared to doing it by coding each components. Each element in the GUI contains properties which are values associated with them and event which can be overridden to do something when that occurs.

All the methods are defined in general way to access the methods at any time. Each Tab maintains a stack which is used for saving the state of images just before the edit. This is used in order to revert back the changes thereby undoing any operation which was done on the image. All the operations on the image is done by loading the image into Bitmap Object and then accessing the elements from them.

In order to apply different kernel to the image, a general method is written called as *applyKernel()* which takes in the kernel information as input and produces a Bitmap object. After Locking the image, we can read it in terms of bytes by various methods and then apply the kernel to each pixel one by one. Kernel is a 2D matrix that represents the same in frequency domain.
All the pixel accessing operations use the same strategy of locking and unlocking the images before editing for faster execution. For Histogram Equalization we first perform this operation based on the color scheme that is selected. Then by using Chart Element from Windows Form GUI Toolkit we then represent the values of pixel distribution on to this Chart Element and present it to the user. Figure 11 shows a typical RGB histogram for that particular image.

All the operation on the image is performed as per the algorithms mentioned in the previous Methodology Section.Some of the kernels used in this program includes from blurring techniques to edge detection. Some of these are as follows:

Kernel	Operation
$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \cdot \frac{1}{9}$	Simple Mean Blur
$\begin{pmatrix} -1 & -2 & -1 \\ -2 & 13 & -2 \\ -1 & -2 & -1 \end{pmatrix}$	Edge Detection Filter
$\begin{pmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}$	Emboss
$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$	Gaussian Filter

$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$	Laplacian Filter

Kernel	Operation
$\begin{pmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{pmatrix}$	Laplacian of Gaussian
$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$	Prewitt Operator
$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{pmatrix}$	Sharpening Filter
$\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$	Sobel Horizontal Operator
$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$	Sobel Vertical Operator

In the above table we can notice various filters used by the program to do the operations defined along with them. We create a 2-dimensional array of these integers and pass them to *applyFilter* method to apply this to our image. Every time the tab that is selected is used for image processing.

3.4 Result

This sections covers the output of our program for different operations. Figure 13 is used as a sample data for different operations. Each Figure captions describe the various operations.

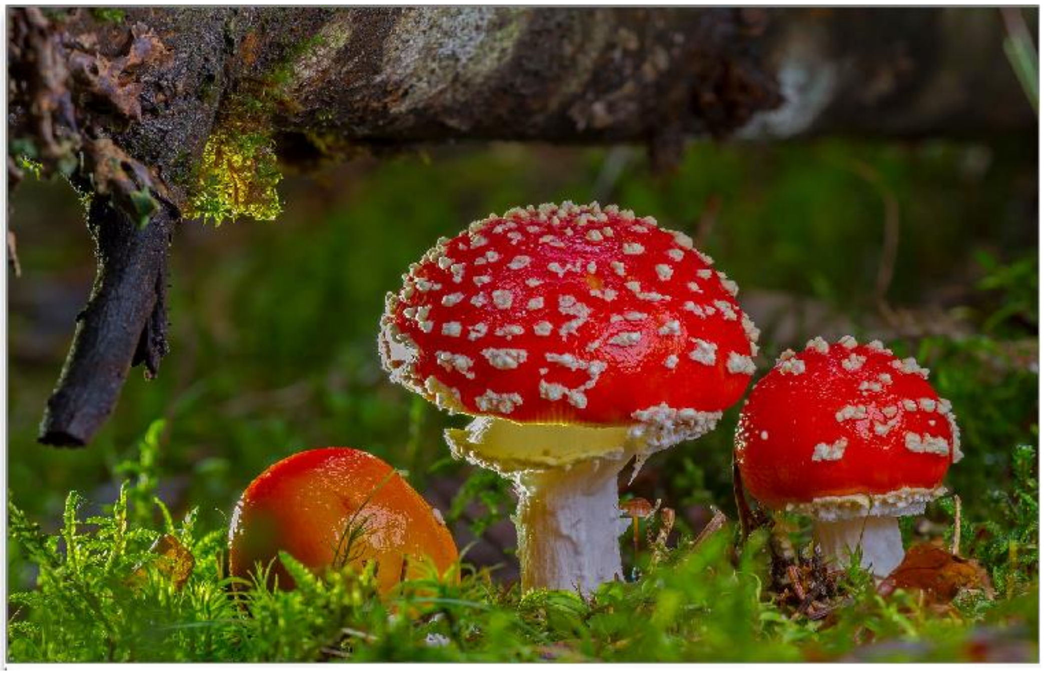


Figure 13: Original Image

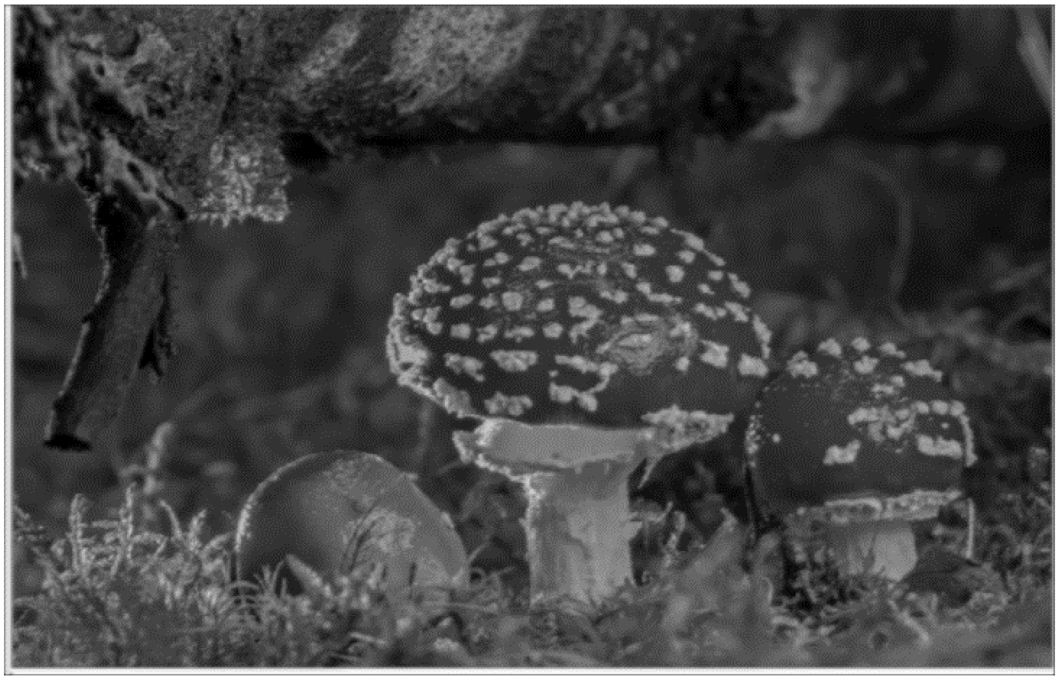


Figure 14: Mean Blur

The operations on image is done by calling the method *applyKernel()* over the different filters shown in the previous table.

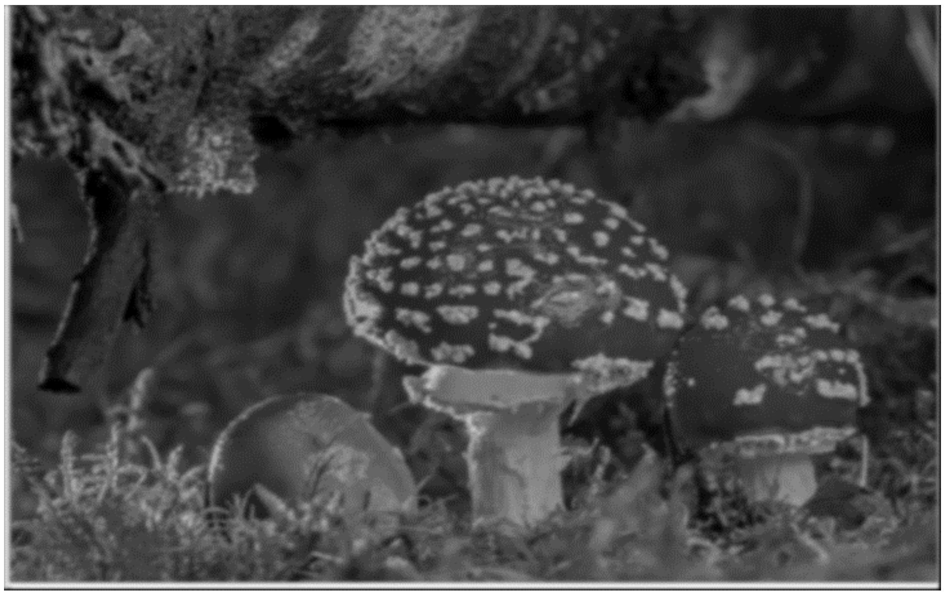


Figure 16: Gaussian Blur

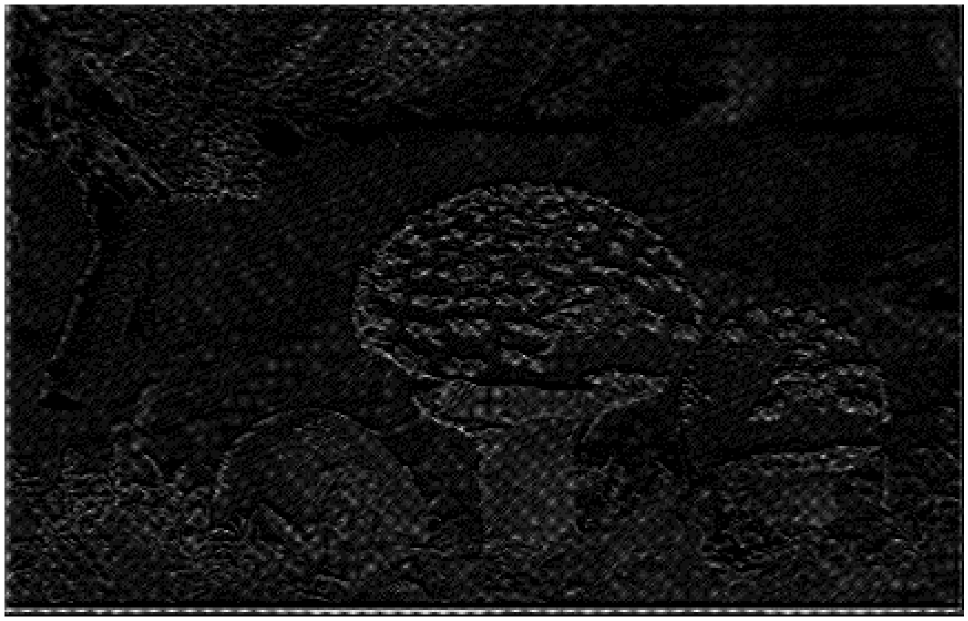


Figure 15: Edge enhancement

In Figure 15, a medium level Edge enhancement technique is used in order to obtain the above results. Figure 18 shows the output of Invert function which just subtracts the bit value from 255 and stores them in that location. This is helpful for taking the negative of an image.

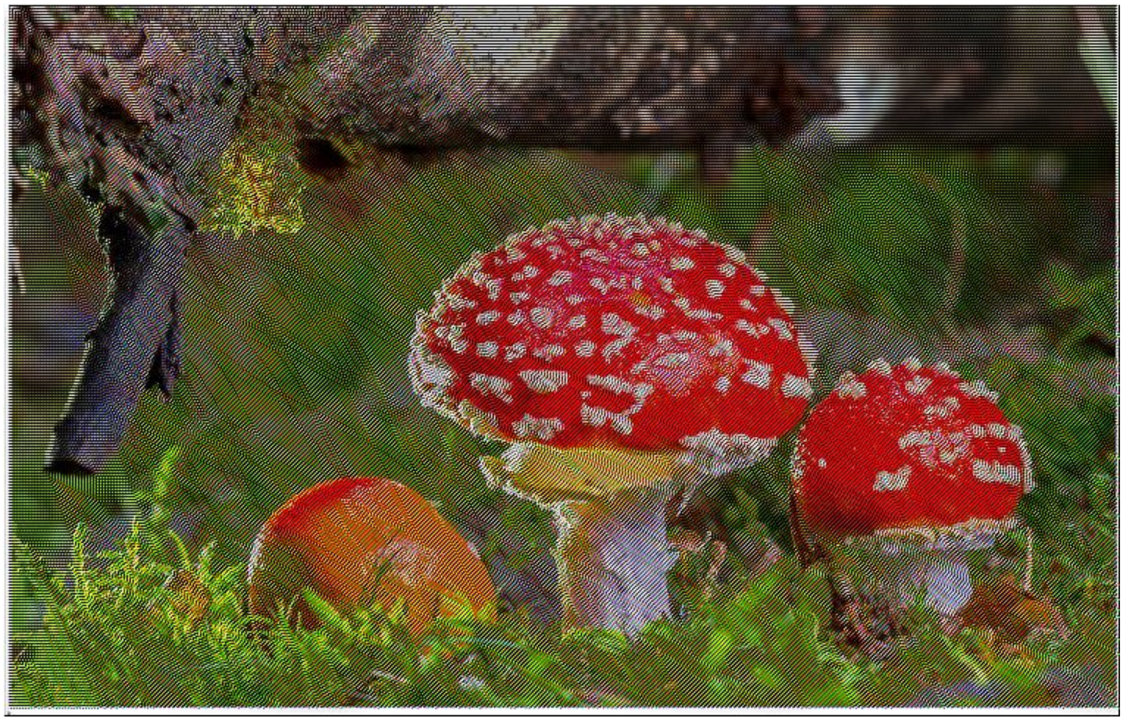


Figure 17: Emboss Filter

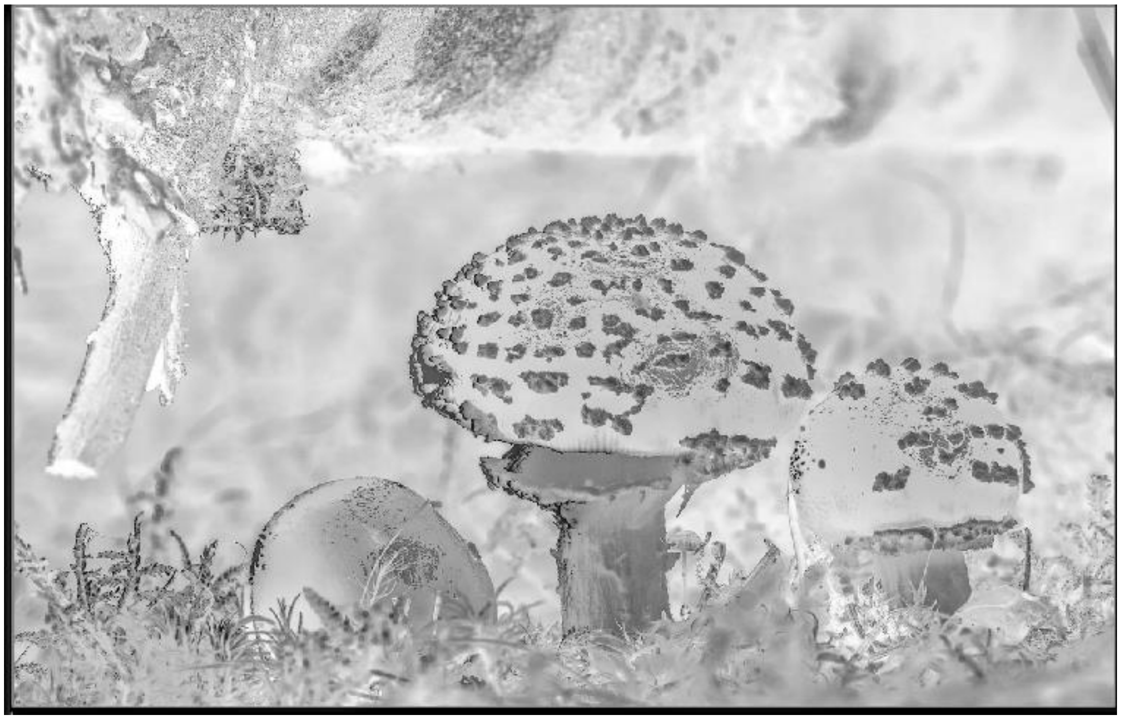


Figure 18: Image Inversion

Figures from 13 to 18 show various filtering techniques output provided by the program.

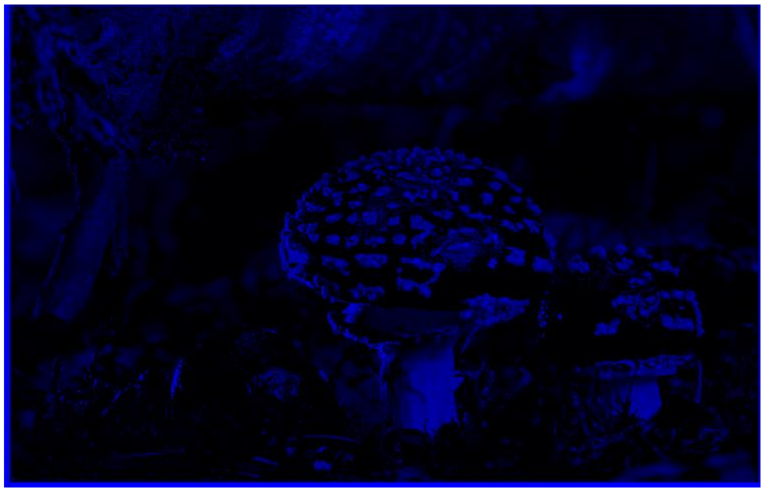


Figure 19: Blue Channel

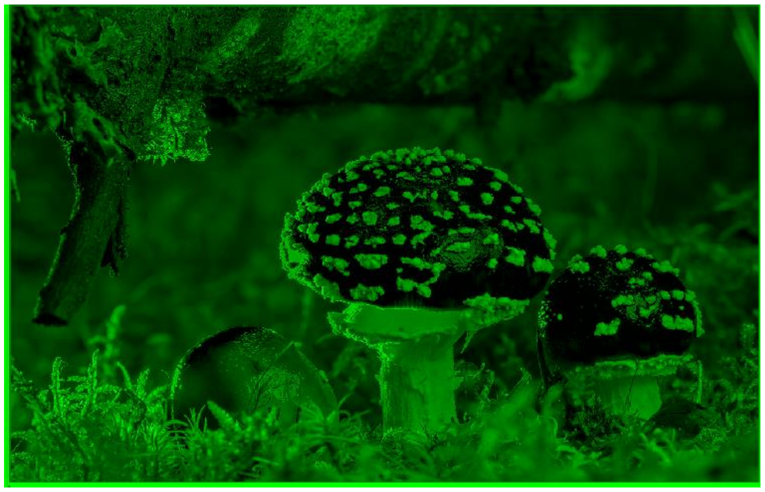


Figure 20:Green Channel

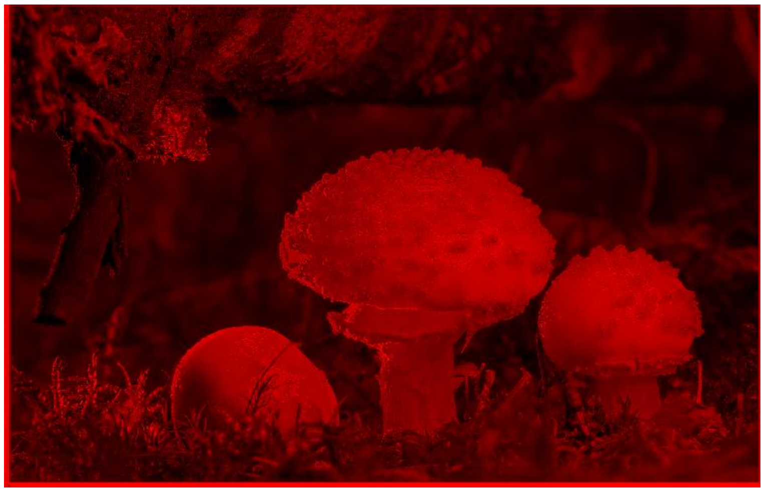


Figure 21: Red Channel

Figure 19 – 21 shows different channels of the original image shown in Figure 13

Outputs for Histogram Equalization for Figure 22 is shown in Figure 23 for color image processing when we apply this technique across all the channels R, G and B.

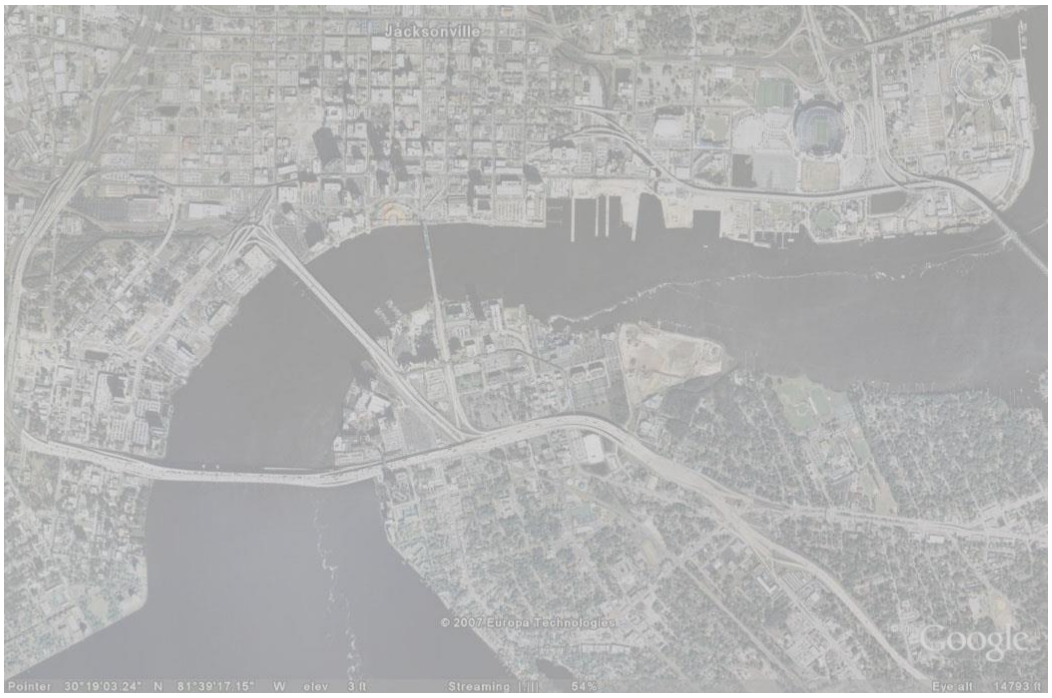

Jacksonville
© 2007 Europa Technologies
Google
Pointer 30°19'03.24" N 81°39'17.15" W elev 3 ft
Streaming 54%
Eye alt 14793 ft

Figure 22: Original Image

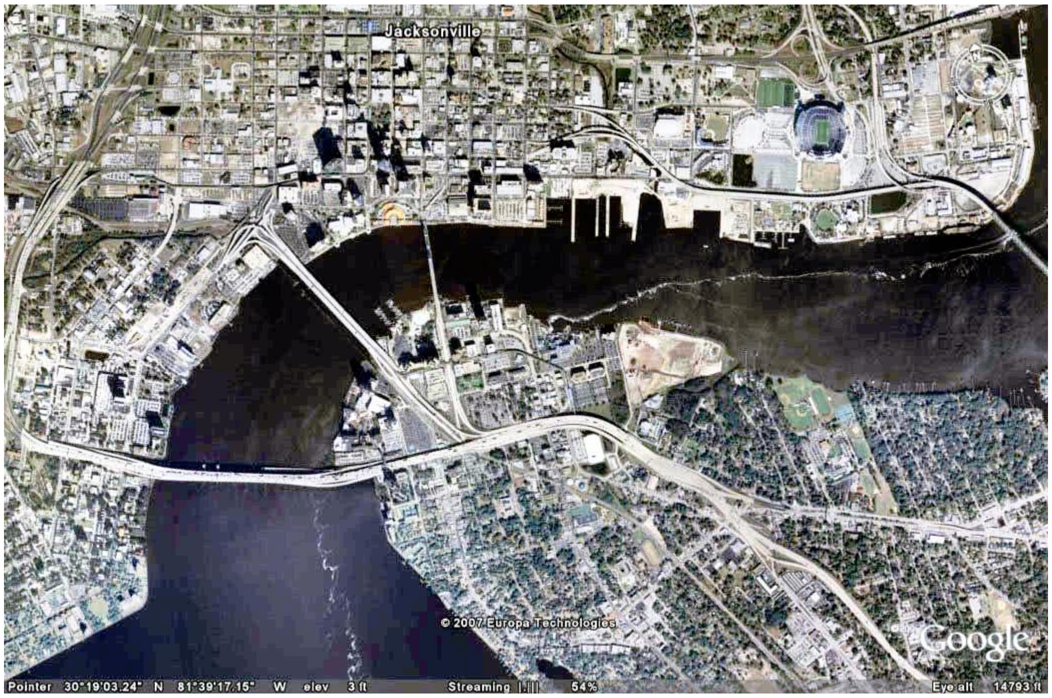

Jacksonville
© 2007 Europa Technologies
Google
Pointer 30°19'03.24" N 81°39'17.15" W elev 3 ft
Streaming 54%
Eye alt 14793 ft

Figure 23: Color Image Histogram Equalization

Outputs of the morphological operations are shown in the below Figures for original Image shown in Figure 24.

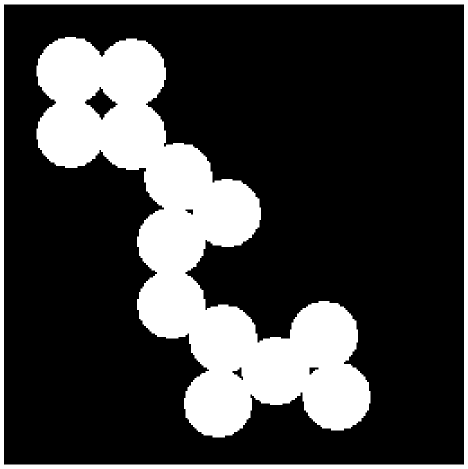


Figure 24; Original Image

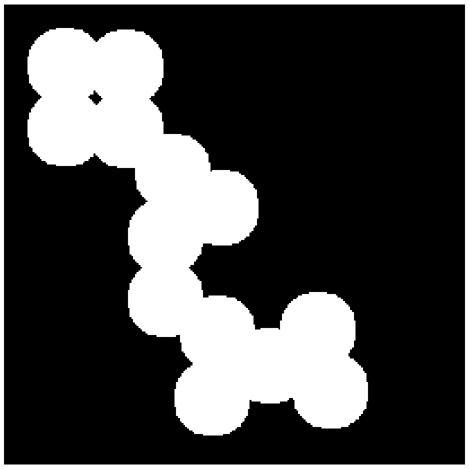


Figure 25: Dilation

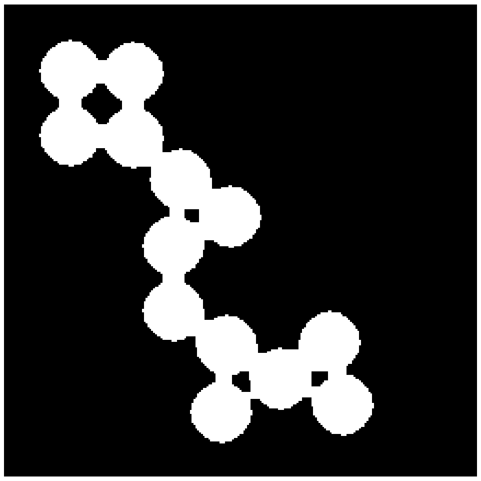


Figure 26: Erosion

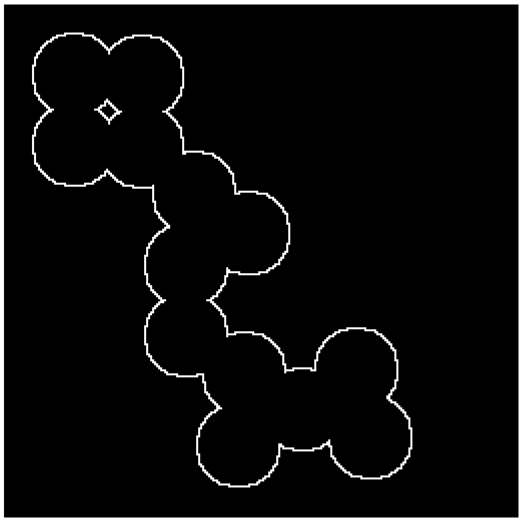


Figure 27: Perimeter Identification

Figure 25 to 27 represents various morphological operations which are performed on the original image shown in Figure 21. These techniques are helpful for identifying the shape of an object. Notice how in Figure 24 a technique called Perimeter Identification is performed which is basically operations involving Erosion followed by BITWISE XOR with the original image there by highlighting only the edges.

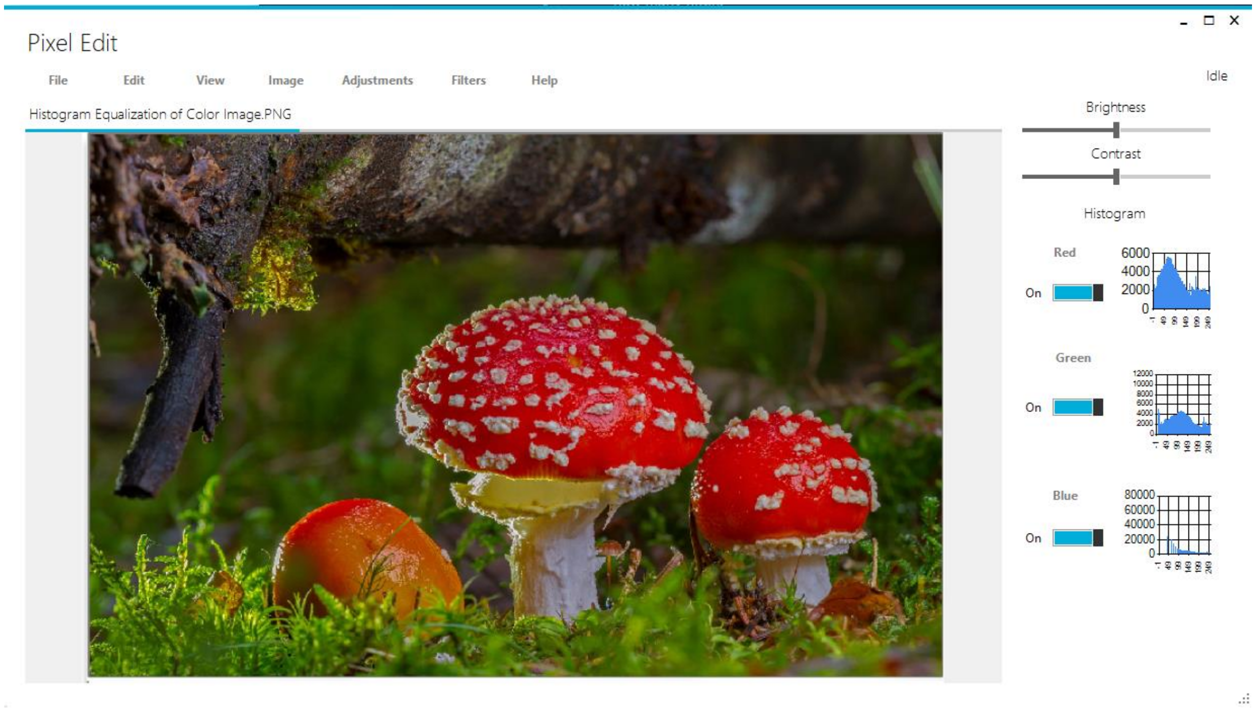


Figure 28: Histogram Example

Figure 25 shows the histogram for the given image sample and represents the histogram across RGB channels.

Chapter 4

CONCLUSION

The overall goal of the project was to design a user friendly multi-purpose application for image processing. The users are given an option to load in multiple images from the directory and work on them independently. The project focuses on implementing Convolution Operation technique to apply different filters for different applications. Along with this some set of morphological operations are provided by the application which includes Erosion, Dilation, Open and Close.

The given application can work in any systems running on Windows Operating System and has .NET Frameworks (Version 4.6 or above) installed. An average configuration is sufficient to run this application without any issues. The designed application meets all the basic requirements as per the problem statement and objective.

There is definitely scope for further improvements to this application in terms of Design, Code, Features and so on. Complex operations could be included to improve the software in terms of features. Better logic can be applied to improve the performance of the application. Layering ability can be added to work with different layers of images at a time.

Chapter 5

REFERENCES

The following are the references used while designing the given application project:

- For obtaining the kernels and implementation of the convolution operation: https://en.wikipedia.org/wiki/Kernel_(image_processing).
- C# Programming concepts: https://www.tutorialspoint.com/csharp/
- For further information about different classes and methods: https://stackoverflow.com/
- Additional kernels: http://setosa.io/ev/image-kernels/