SAMPLE CODE

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.OleDb;
using System.Drawing;
using System.Drawing.Imaging;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Data.OleDb;
using System.Speech;
using System.Speech.Synthesis;
using AForge.Imaging;
using CnetSDK.OCR.Trial;
namespace ImgProImpl
{

public partial class Form1 : Form
   {
      OleDbConnection cn = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=dbase.mdb");
      Canny CannyData;
      OpenFileDialog oDlg;
      SaveFileDialog sDlg;
double zoomFactor = 1.0;
private MenuItem cZoom;
      ImageHandler imageHandler = new ImageHandler();
string fp = "";
public Form1()
      {
InitializeComponent();
fp = "D:\\Image\\";
oDlg = new OpenFileDialog(); // Open Dialog Initialization
         oDlg.RestoreDirectory = true;
         oDlg.InitialDirectory = "D:\\";
         oDlg.FilterIndex = 1;
         oDlg.Filter = "jpg Files (*.jpg)|*.jpg|gif Files (*.gif)|*.gif|png Files (*.png)|*.png
|bmp Files (*.bmp)|*.bmp";

      }
private void button1_Click(object sender, EventArgs e)
      {
if (DialogResult.OK == oDlg.ShowDialog())
         {
            imageHandler.CurrentBitmap = (Bitmap)Bitmap.FromFile(oDlg.FileName);
```

```
            imageHandler.BitmapPath = oDlg.FileName;
this.AutoScroll = true;
            this.AutoScrollMinSize = new
Size(Convert.ToInt32(imageHandler.CurrentBitmap.Width * zoomFactor),
Convert.ToInt32(imageHandler.CurrentBitmap.Height * zoomFactor));
this.Invalidate();
            //menuItemImageInfo.Enabled = true;
            ImageInfo imgInfo = new ImageInfo(imageHandler);
imgInfo.Show();
imageHandler.SaveBitmap(fp + "img1.jpg");
imageHandler.SaveBitmap(fp + "img6.jpg");
            pictureBox1.Image = System.Drawing.Image.FromFile(fp + "img1.jpg");
         }
      }
private void Form1_Load(object sender, EventArgs e)
      {
         panel6.Visible = false;
         panel5.Visible = false;
         panel2.Visible = false;
         panel3.Visible = false;
         panel4.Visible = false;
      }
private void button2_Click(object sender, EventArgs e)
      {
         this.Cursor = Cursors.WaitCursor;
imageHandler.RestorePrevious();
imageHandler.SetColorFilter(ImageHandler.ColorFilterTypes.Red);
this.Invalidate();
         this.Cursor = Cursors.Default;
imageHandler.SaveBitmap(fp + "img2.jpg");
         pictureBox2.Image = System.Drawing.Image.FromFile(fp + "img2.jpg");
imageHandler.ResetBitmap();
         this.AutoScrollMinSize = new
Size(Convert.ToInt32(imageHandler.CurrentBitmap.Width * zoomFactor),
Convert.ToInt32(imageHandler.CurrentBitmap.Height * zoomFactor));
this.Invalidate();
         this.Cursor = Cursors.WaitCursor;
imageHandler.RestorePrevious();
imageHandler.SetColorFilter(ImageHandler.ColorFilterTypes.Green);
this.Invalidate();
         this.Cursor = Cursors.Default;
imageHandler.SaveBitmap(fp + "img3.jpg");
         pictureBox3.Image = System.Drawing.Image.FromFile(fp + "img3.jpg");
imageHandler.ResetBitmap();
         this.AutoScrollMinSize = new
Size(Convert.ToInt32(imageHandler.CurrentBitmap.Width * zoomFactor),
Convert.ToInt32(imageHandler.CurrentBitmap.Height * zoomFactor));
this.Invalidate();
         this.Cursor = Cursors.WaitCursor;
imageHandler.RestorePrevious();
```

```
imageHandler.SetColorFilter(ImageHandler.ColorFilterTypes.Blue);
this.Invalidate();
        this.Cursor = Cursors.Default;
imageHandler.SaveBitmap(fp + "img4.jpg");
        pictureBox4.Image = System.Drawing.Image.FromFile(fp + "img4.jpg");
imageHandler.ResetBitmap();
        this.AutoScrollMinSize = new
Size(Convert.ToInt32(imageHandler.CurrentBitmap.Width * zoomFactor),
Convert.ToInt32(imageHandler.CurrentBitmap.Height * zoomFactor));
this.Invalidate();
        this.Cursor = Cursors.WaitCursor;
imageHandler.RestorePrevious();
imageHandler.SetGrayscale();
this.Invalidate();
        this.Cursor = Cursors.Default;
imageHandler.SaveBitmap(fp + "img5.jpg");
        pictureBox5.Image = System.Drawing.Image.FromFile(fp + "img5.jpg");
imageHandler.ResetBitmap();
        this.AutoScrollMinSize = new
Size(Convert.ToInt32(imageHandler.CurrentBitmap.Width * zoomFactor),
Convert.ToInt32(imageHandler.CurrentBitmap.Height * zoomFactor));
this.Invalidate();
        this.Cursor = Cursors.WaitCursor;
imageHandler.RestorePrevious();
imageHandler.SetContrast(20.0);
this.Invalidate();
        this.Cursor = Cursors.Default;
imageHandler.SaveBitmap(fp + "img8.jpg");
        pictureBox6.Image = System.Drawing.Image.FromFile(fp + "img8.jpg");
imageHandler.ResetBitmap();
        this.AutoScrollMinSize = new
Size(Convert.ToInt32(imageHandler.CurrentBitmap.Width * zoomFactor),
Convert.ToInt32(imageHandler.CurrentBitmap.Height * zoomFactor));
this.Invalidate();
        panel3.Visible = true;
        pictureBox7.Image = System.Drawing.Image.FromFile(fp + "img6.jpg");
    }
private void button3_Click(object sender, EventArgs e)
    {
if (File.Exists(fp + "img11.jpg"))
        {
File.Delete(fp + "img11.jpg");
        }
pictureBox9.Image.Save(fp + "img11.jpg");
        imageHandler.CurrentBitmap = (Bitmap)Bitmap.FromFile(fp + "img11.jpg");
        imageHandler.BitmapPath = fp + "img11.jpg";
imageHandler.SetContrast(50.0);
        this.AutoScroll = true;
```

```
            this.AutoScrollMinSize = new
Size(Convert.ToInt32(imageHandler.CurrentBitmap.Width * zoomFactor),
Convert.ToInt32(imageHandler.CurrentBitmap.Height * zoomFactor));
this.Invalidate();

            panel6.Visible = true;
this.Cursor = Cursors.WaitCursor;
imageHandler.RestorePrevious();
imageHandler.SetInvert();
this.Invalidate();
            this.Cursor = Cursors.Default;
imageHandler.SaveBitmap(fp + "img10.jpg");
            pictureBox12.Image = System.Drawing.Image.FromFile(fp + "img10.jpg");
imageHandler.ResetBitmap();
            this.AutoScrollMinSize = new
Size(Convert.ToInt32(imageHandler.CurrentBitmap.Width * zoomFactor),
Convert.ToInt32(imageHandler.CurrentBitmap.Height * zoomFactor));
this.Invalidate();
        }
private void button4_Click(object sender, EventArgs e)
        {
float TH, TL, Sigma;
int MaskSize;
            TH = (float)Convert.ToDouble("30.0");
            TL = (float)Convert.ToDouble("10.0");
            MaskSize = Convert.ToInt32("5");
            Sigma = (float)Convert.ToDouble("1.5");
            CannyData = new Canny((Bitmap)pictureBox3.Image, TH, TL, MaskSize, Sigma);
            pictureBox8.Image = CannyData.DisplayImage(CannyData.GNL);
            pictureBox9.Image = CannyData.DisplayImage(CannyData.GNH);
            panel4.Visible = true;
            this.Cursor = Cursors.WaitCursor;
imageHandler.RestorePrevious();
imageHandler.SetBrightness(100);
this.Invalidate();
            this.Cursor = Cursors.Default;
imageHandler.SaveBitmap(fp + "img9.jpg");
imageHandler.ResetBitmap();
            this.AutoScrollMinSize = new
Size(Convert.ToInt32(imageHandler.CurrentBitmap.Width * zoomFactor),
Convert.ToInt32(imageHandler.CurrentBitmap.Height * zoomFactor));
this.Invalidate();
            }
private void button5_Click(object sender, EventArgs e)
        {
            panel6.Visible = false;
MessageBox.Show("Classification has been done", "Alert", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }
public void classify1()
```

```
      {
string fn = "";
int flag = 0;
         System.Drawing.Bitmap sourceImage = (Bitmap)Bitmap.FromFile(fp + "img9.jpg");

         DirectoryInfo dir = new DirectoryInfo(@"D:\Dataset\Datas");
FileInfo[] fi = dir.GetFiles();
         System.Drawing.Bitmap sourceIma = (Bitmap)Bitmap.FromFile(fp + "img1.jpg");
         System.Drawing.Bitmap clon = sourceIma.Clone(new Rectangle(0, 0,
sourceIma.Width, sourceIma.Height), PixelFormat.Format24bppRgb);
foreach (FileInfo f in fi)
         {
            System.Drawing.Bitmap templat =
(Bitmap)Bitmap.FromFile(f.FullName.ToString());
            ExhaustiveTemplateMatching tm = new ExhaustiveTemplateMatching(0.999f);
            Bitmap clone = sourceImage.Clone(new Rectangle(0, 0, sourceImage.Width,
sourceImage.Height), PixelFormat.Format8bppIndexed);
            Bitmap template = templat.Clone(new Rectangle(0, 0, templat.Width,
templat.Height), PixelFormat.Format8bppIndexed);
TemplateMatch[] matchings = tm.ProcessImage(clone, template);
            BitmapData data = clone.LockBits(new Rectangle(0, 0, clone.Width,
clone.Height), ImageLockMode.ReadWrite, clone.PixelFormat);
foreach (TemplateMatch m in matchings)
            {
flag = 0;
Drawing.Rectangle(data, m.Rectangle, Color.White);
               Graphics graphics = Graphics.FromImage((System.Drawing.Image)clon);
               Pen whitePen = new Pen(Color.White, 0);
               OleDbCommand cm = new OleDbCommand("select * from dbtab where
fname='" + f.Name + "'", cn);
cn.Open();
               OleDbDataReader dr = cm.ExecuteReader();
while (dr.Read())
               {
whitePen = new Pen(Color.Red, 2);
int x = m.Rectangle.X + 6;
int y = m.Rectangle.Y + 6;
int n = Convert.ToInt32(dr.GetString(1));
int nm = Convert.ToInt32(dr.GetString(2));
                  Rectangle rect = new Rectangle(x, y, n, nm);
graphics.DrawRectangle(whitePen, rect);
graphics.Dispose();
                  textBox2.Text = dr.GetString(3);
               }
cn.Close();
            }
clone.UnlockBits(data);
         }
         pictureBox11.Image = clon;
         panel5.Visible = true;
```

```
        }
public void classify2()
        {
string fn = "";
int flag = 0;
            System.Drawing.Bitmap sourceImage = (Bitmap)Bitmap.FromFile(fp + "img9.jpg");
            DirectoryInfo dir = new DirectoryInfo(@"D:\Dataset\Datas");
FileInfo[] fi = dir.GetFiles();
            System.Drawing.Bitmap sourceIma = (Bitmap)Bitmap.FromFile(fp + "img1.jpg");
            System.Drawing.Bitmap clon = sourceIma.Clone(new Rectangle(0, 0,
sourceIma.Width, sourceIma.Height), PixelFormat.Format24bppRgb);
foreach (FileInfo f in fi)
            {
              System.Drawing.Bitmap templat =
(Bitmap)Bitmap.FromFile(f.FullName.ToString());
                ExhaustiveTemplateMatching tm = new ExhaustiveTemplateMatching(0.999f);
                Bitmap clone = sourceImage.Clone(new Rectangle(0, 0, sourceImage.Width,
sourceImage.Height), PixelFormat.Format8bppIndexed);
                Bitmap template = templat.Clone(new Rectangle(0, 0, templat.Width,
templat.Height), PixelFormat.Format8bppIndexed);
TemplateMatch[] matchings = tm.ProcessImage(clone, template);
                BitmapData data = clone.LockBits(new Rectangle(0, 0, clone.Width,
clone.Height), ImageLockMode.ReadWrite, clone.PixelFormat);
foreach (TemplateMatch m in matchings)
                {
Drawing.Rectangle(data, m.Rectangle, Color.White);
                    Graphics graphics = Graphics.FromImage((System.Drawing.Image)clon);
                    Pen whitePen = new Pen(Color.Red, 2);
if (f.Name.IndexOf('m') >= 0)
                    {
int x = m.Rectangle.X;
int y = m.Rectangle.Y;
int n = clon.Height;
int nm = clon.Width;
                        n = 50;
nm = 50;
                        Rectangle rect = new Rectangle(x, y, n, nm);
graphics.DrawRectangle(whitePen, rect);
graphics.Dispose();
                    }
                }
clone.UnlockBits(data);
            }
            pictureBox10.Image = clon;
            panel5.Visible = true;
        }
private void button6_Click(object sender, EventArgs e)
        {
                            OcrEngine OCRLibrary = new OcrEngine();
                    // Set the absolute path of tessdata.
```

```
            OCRLibrary.TessDataPath = "D:/image/";
                    // Set the target text language.
            OCRLibrary.TextLanguage = "eng";
            // Recognize text from image file.
string Imagetext = OCRLibrary.PerformOCR("D://Image//img8.jpg");
        }
private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
        {
            this.Cursor = Cursors.WaitCursor;
imageHandler.RestorePrevious();
imageHandler.SetBrightness(100);
this.Invalidate();
            this.Cursor = Cursors.Default;
imageHandler.SaveBitmap(fp + "img9.jpg");
imageHandler.ResetBitmap();
            this.AutoScrollMinSize = new
Size(Convert.ToInt32(imageHandler.CurrentBitmap.Width * zoomFactor),
Convert.ToInt32(imageHandler.CurrentBitmap.Height * zoomFactor));
this.Invalidate();
        }
private void button7_Click_1(object sender, EventArgs e)
        {
            SpeechSynthesizer speak = new SpeechSynthesizer();
speak.SpeakAsync(textBox1.Text);
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing.Imaging;
using System.Drawing;
namespace ImgProImpl
{
class Canny
    {
public int Width, Height;
public Bitmap Obj;
public int[,] GreyImage;
        //Gaussian Kernel Data
int[,] GaussianKernel;
int KernelWeight;
int KernelSize = 5;
float Sigma = 1;   // for N=2 Sigma =0.85  N=5 Sigma =1, N=9 Sigma = 2    2*Sigma =
(int)N/2
        //Canny Edge Detection Parameters
float MaxHysteresisThresh, MinHysteresisThresh;
public float[,] DerivativeX;
public float[,] DerivativeY;
```

```
public int[,] FilteredImage;
public float[,] Gradient;
public float[,] NonMax;
public int[,] PostHysteresis;
int[,] EdgePoints;
public float[,] GNH;
public float[,] GNL;
public int[,] EdgeMap;
public int[,] VisitedMap;
public Canny(Bitmap Input)
        {
           // Gaussian and Canny Parameters
           MaxHysteresisThresh = 20F;
           MinHysteresisThresh = 10F;
           Obj = Input;
           Width = Obj.Width;
           Height = Obj.Height;
           EdgeMap = new int[Width, Height];
           VisitedMap = new int[Width, Height];

ReadImage();
DetectCannyEdges();
return;
        }
public Canny(Bitmap Input, float Th, float Tl)
        {
           // Gaussian and Canny Parameters
           MaxHysteresisThresh = Th;
           MinHysteresisThresh = Tl;
          Obj = Input;
           Width = Obj.Width;
           Height = Obj.Height;
           EdgeMap = new int[Width, Height];
           VisitedMap = new int[Width, Height];
ReadImage();
DetectCannyEdges();
return;
        }

public Canny(Bitmap Input, float Th, float Tl, int GaussianMaskSize, float
SigmaforGaussianKernel)
        {
           // Gaussian and Canny Parameters

           MaxHysteresisThresh = Th;
           MinHysteresisThresh = Tl;
           KernelSize = GaussianMaskSize;
           Sigma = SigmaforGaussianKernel;
           Obj = Input;
           Width = Obj.Width;
```

```
        Height = Obj.Height;

        EdgeMap = new int[Width, Height];
        VisitedMap = new int[Width, Height];
ReadImage();
DetectCannyEdges();
return;
    }
public Bitmap DisplayImage()
    {
int i, j;
        Bitmap image = new Bitmap(Obj.Width, Obj.Height);
        BitmapData bitmapData1 = image.LockBits(new Rectangle(0, 0, Obj.Width,
Obj.Height),
                        ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb);
unsafe
        {
byte* imagePointer1 = (byte*)bitmapData1.Scan0;

for (i = 0; i < bitmapData1.Height; i++)
            {
for (j = 0; j < bitmapData1.Width; j++)
                {
                    // write the logic implementation here
imagePointer1[0] = (byte)GreyImage[j, i];
imagePointer1[1] = (byte)GreyImage[j, i];
imagePointer1[2] = (byte)GreyImage[j, i];
imagePointer1[3] = (byte)255;
                    //4 bytes per pixel
                    imagePointer1 += 4;
                }//end for j

                //4 bytes per pixel
                imagePointer1 += (bitmapData1.Stride - (bitmapData1.Width * 4));
            }//end for i
        }//end unsafe
image.UnlockBits(bitmapData1);
return image;// col;
    }     // Display Grey Image

public Bitmap DisplayImage(float[,] GreyImage)
    {
int i, j;
int W, H;
        W = GreyImage.GetLength(0);
        H = GreyImage.GetLength(1);
        Bitmap image = new Bitmap(W, H);
        BitmapData bitmapData1 = image.LockBits(new Rectangle(0, 0, W, H),
                        ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb);
unsafe
```

```
        {
byte* imagePointer1 = (byte*)bitmapData1.Scan0;

for (i = 0; i < bitmapData1.Height; i++)
            {
for (j = 0; j < bitmapData1.Width; j++)
                {
                    // write the logic implementation here
imagePointer1[0] = (byte)((int)(GreyImage[j, i]));
imagePointer1[1] = (byte)((int)(GreyImage[j, i]));
imagePointer1[2] = (byte)((int)(GreyImage[j, i]));
imagePointer1[3] = (byte)255;
                    //4 bytes per pixel
                    imagePointer1 += 4;
                }   //end for j
                //4 bytes per pixel
                imagePointer1 += (bitmapData1.Stride - (bitmapData1.Width * 4));
            }//End for i
        }//end unsafe
image.UnlockBits(bitmapData1);
return image;// col;
    }     // Display Grey Imag

public Bitmap DisplayImage(int[,] GreyImage)
    {
int i, j;
int W, H;
        W = GreyImage.GetLength(0);
        H = GreyImage.GetLength(1);
        Bitmap image = new Bitmap(W, H);
        BitmapData bitmapData1 = image.LockBits(new Rectangle(0, 0, W, H),
                            ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb);
unsafe
        {

byte* imagePointer1 = (byte*)bitmapData1.Scan0;

for (i = 0; i < bitmapData1.Height; i++)
            {
for (j = 0; j < bitmapData1.Width; j++)
                {
                    // write the logic implementation here
imagePointer1[0] = (byte)GreyImage[j, i];
imagePointer1[1] = (byte)GreyImage[j, i];
imagePointer1[2] = (byte)GreyImage[j, i];
imagePointer1[3] = (byte)255;
                    //4 bytes per pixel
                    imagePointer1 += 4;
                }   //end for j
                //4 bytes per pixel
```

```
                imagePointer1 += (bitmapData1.Stride - (bitmapData1.Width * 4));
            }//End for i
        }//end unsafe
image.UnlockBits(bitmapData1);
return image;// col;
    }     // Display Grey Image

private void ReadImage()
    {
int i, j;
        GreyImage = new int[Obj.Width, Obj.Height];  //[Row,Column]
        Bitmap image = Obj;
        BitmapData bitmapData1 = image.LockBits(new Rectangle(0, 0, image.Width,
image.Height),
                        ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb);
unsafe
        {
byte* imagePointer1 = (byte*)bitmapData1.Scan0;

for (i = 0; i < bitmapData1.Height; i++)
            {
for (j = 0; j < bitmapData1.Width; j++)
                {
GreyImage[j, i] = (int)((imagePointer1[0] + imagePointer1[1] + imagePointer1[2]) / 3.0);
                  //4 bytes per pixel
                  imagePointer1 += 4;
                }//end for j
                //4 bytes per pixel
                imagePointer1 += bitmapData1.Stride - (bitmapData1.Width * 4);
            }//end for i
        }//end unsafe
image.UnlockBits(bitmapData1);
return;
    }
private void GenerateGaussianKernel(int N, float S, out int Weight)
    {
float Sigma = S;
float pi;
pi = (float)Math.PI;
int i, j;
int SizeofKernel = N;
float[,] Kernel = new float[N, N];
        GaussianKernel = new int[N, N];
float[,] OP = new float[N, N];
float D1, D2;
        D1 = 1 / (2 * pi * Sigma * Sigma);
        D2 = 2 * Sigma * Sigma;

float min = 1000;
```

```
for (i = -SizeofKernel / 2; i <= SizeofKernel / 2; i++)
        {
for (j = -SizeofKernel / 2; j <= SizeofKernel / 2; j++)
            {
Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j] = ((1 / D1) * (float)Math.Exp(-(i * i + j * j)
/ D2));
if (Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j] < min)
min = Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j];

            }
        }
int mult = (int)(1 / min);
int sum = 0;
if ((min > 0) && (min < 1))
        {

for (i = -SizeofKernel / 2; i <= SizeofKernel / 2; i++)
            {
for (j = -SizeofKernel / 2; j <= SizeofKernel / 2; j++)
                {
Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j] = (float)Math.Round(Kernel[SizeofKernel /
2 + i, SizeofKernel / 2 + j] * mult, 0);
GaussianKernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j] = (int)Kernel[SizeofKernel / 2 + i,
SizeofKernel / 2 + j];
sum = sum + GaussianKernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j];
                }

            }

        }
else
        {
sum = 0;
for (i = -SizeofKernel / 2; i <= SizeofKernel / 2; i++)
            {
for (j = -SizeofKernel / 2; j <= SizeofKernel / 2; j++)
                {
Kernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j] = (float)Math.Round(Kernel[SizeofKernel /
2 + i, SizeofKernel / 2 + j], 0);
GaussianKernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j] = (int)Kernel[SizeofKernel / 2 + i,
SizeofKernel / 2 + j];
sum = sum + GaussianKernel[SizeofKernel / 2 + i, SizeofKernel / 2 + j];
                }
            }
        }
        //Normalizing kernel Weight
        Weight = sum;
return;
    }
private int[,] GaussianFilter(int[,] Data)
```

```
        {
GenerateGaussianKernel(KernelSize, Sigma, out KernelWeight);

int[,] Output = new int[Width, Height];
int i, j, k, l;
int Limit = KernelSize / 2;
float Sum = 0;
            Output = Data; // Removes Unwanted Data Omission due to kernel bias while
convolution
for (i = Limit; i <= ((Width - 1) - Limit); i++)
            {
for (j = Limit; j <= ((Height - 1) - Limit); j++)
                {
                    Sum = 0;
for (k = -Limit; k <= Limit; k++)
                    {
for (l = -Limit; l <= Limit; l++)
                        {
                            Sum = Sum + ((float)Data[i + k, j + l] * GaussianKernel[Limit + k, Limit +
l]);
                        }
                    }
Output[i, j] = (int)(Math.Round(Sum / (float)KernelWeight));
                }
            }
return Output;
        }

private float[,] Differentiate(int[,] Data, int[,] Filter)
        {
int i, j, k, l, Fh, Fw;

            Fw = Filter.GetLength(0);
            Fh = Filter.GetLength(1);
float sum = 0;
float[,] Output = new float[Width, Height];

for (i = Fw / 2; i <= (Width - Fw / 2) - 1; i++)
            {
for (j = Fh / 2; j <= (Height - Fh / 2) - 1; j++)
                {
sum = 0;
for (k = -Fw / 2; k <= Fw / 2; k++)
                    {
for (l = -Fh / 2; l <= Fh / 2; l++)
                        {
sum = sum + Data[i + k, j + l] * Filter[Fw / 2 + k, Fh / 2 + l];
                        }
                    }
Output[i, j] = sum;
```

```
            }

        }
return Output;

    }

private void DetectCannyEdges()
    {
        Gradient = new float[Width, Height];
        NonMax = new float[Width, Height];
        PostHysteresis = new int[Width, Height];
        DerivativeX = new float[Width, Height];
        DerivativeY = new float[Width, Height];
        //Gaussian Filter Input Image
        FilteredImage = GaussianFilter(GreyImage);
        //Sobel Masks
int[,] Dx = {{1,0,-1},
                    {1,0,-1},
                    {1,0,-1}};
int[,] Dy = {{1,1,1},
                    {0,0,0},
                    {-1,-1,-1}};
        DerivativeX = Differentiate(FilteredImage, Dx);
        DerivativeY = Differentiate(FilteredImage, Dy);
int i, j;

        //Compute the gradient magnitude based on derivatives in x and y:
for (i = 0; i <= (Width - 1); i++)
        {
for (j = 0; j <= (Height - 1); j++)
            {
Gradient[i, j] = (float)Math.Sqrt((DerivativeX[i, j] * DerivativeX[i, j]) + (DerivativeY[i, j] *
DerivativeY[i, j]));
            }
        }
        // Perform Non maximum suppression:
        // NonMax = Gradient;

for (i = 0; i <= (Width - 1); i++)
        {
for (j = 0; j <= (Height - 1); j++)
            {
NonMax[i, j] = Gradient[i, j];
            }
        }
int Limit = KernelSize / 2;
int r, c;
float Tangent;
```

```
for (i = Limit; i <= (Width - Limit) - 1; i++)
        {
for (j = Limit; j <= (Height - Limit) - 1; j++)
            {
if (DerivativeX[i, j] == 0)
                Tangent = 90F;
else
                Tangent = (float)(Math.Atan(DerivativeY[i, j] / DerivativeX[i, j]) * 180 /
Math.PI); //rad to degree

                //Horizontal Edge
if (((-22.5 < Tangent) && (Tangent <= 22.5)) || ((157.5 < Tangent) && (Tangent <= -
157.5)))
                {
if ((Gradient[i, j] < Gradient[i, j + 1]) || (Gradient[i, j] < Gradient[i, j - 1]))
NonMax[i, j] = 0;
                }
                //Vertical Edge
if (((-112.5 < Tangent) && (Tangent <= -67.5)) || ((67.5 < Tangent) && (Tangent <=
112.5)))
                {
if ((Gradient[i, j] < Gradient[i + 1, j]) || (Gradient[i, j] < Gradient[i - 1, j]))
NonMax[i, j] = 0;
                }

                //+45 Degree Edge
if (((-67.5 < Tangent) && (Tangent <= -22.5)) || ((112.5 < Tangent) && (Tangent <=
157.5)))
                {
if ((Gradient[i, j] < Gradient[i + 1, j - 1]) || (Gradient[i, j] < Gradient[i - 1, j + 1]))
NonMax[i, j] = 0;
                }

                //-45 Degree Edge
if (((-157.5 < Tangent) && (Tangent <= -112.5)) || ((67.5 < Tangent) && (Tangent <=
22.5)))
                {
if ((Gradient[i, j] < Gradient[i + 1, j + 1]) || (Gradient[i, j] < Gradient[i - 1, j - 1]))
NonMax[i, j] = 0;
                }

            }
        }

        //PostHysteresis = NonMax;
for (r = Limit; r <= (Width - Limit) - 1; r++)
        {
for (c = Limit; c <= (Height - Limit) - 1; c++)
            {
```

```
                PostHysteresis[r, c] = (int)NonMax[r, c];
            }

        }

        //Find Max and Min in Post Hysterisis
float min, max;
min = 100;
max = 0;
for (r = Limit; r <= (Width - Limit) - 1; r++)
for (c = Limit; c <= (Height - Limit) - 1; c++)
            {
if (PostHysteresis[r, c] > max)
                {
max = PostHysteresis[r, c];
                }

if ((PostHysteresis[r, c] < min) && (PostHysteresis[r, c] > 0))
                {
min = PostHysteresis[r, c];
                }
            }

        GNH = new float[Width, Height];
        GNL = new float[Width, Height]; ;
        EdgePoints = new int[Width, Height];

for (r = Limit; r <= (Width - Limit) - 1; r++)
        {
for (c = Limit; c <= (Height - Limit) - 1; c++)
            {
if (PostHysteresis[r, c] >= MaxHysteresisThresh)
                {

                    EdgePoints[r, c] = 1;
                    GNH[r, c] = 255;
                }
if ((PostHysteresis[r, c] < MaxHysteresisThresh) && (PostHysteresis[r, c] >=
MinHysteresisThresh))
                {

                    EdgePoints[r, c] = 2;
                    GNL[r, c] = 255;

                }

            }

        }
```

```
HysterisisThresholding(EdgePoints);

for (i = 0; i <= (Width - 1); i++)
for (j = 0; j <= (Height - 1); j++)
            {
EdgeMap[i, j] = EdgeMap[i, j] * 255;
            }

return;
      }

private void HysterisisThresholding(int[,] Edges)
      {
int i, j;
int Limit = KernelSize / 2;
for (i = Limit; i <= (Width - 1) - Limit; i++)
for (j = Limit; j <= (Height - 1) - Limit; j++)
            {
if (Edges[i, j] == 1)
               {
EdgeMap[i, j] = 1;

               }
            }
for (i = Limit; i <= (Width - 1) - Limit; i++)
         {
for (j = Limit; j <= (Height - 1) - Limit; j++)
            {
if (Edges[i, j] == 1)
               {
EdgeMap[i, j] = 1;
Travers(i, j);
VisitedMap[i, j] = 1;
               }
            }
         }
return;
      }
private void Travers(int X, int Y)
      {
if (VisitedMap[X, Y] == 1)
         {
return;
         }

         //1
if (EdgePoints[X + 1, Y] == 2)
         {
            EdgeMap[X + 1, Y] = 1;
            VisitedMap[X + 1, Y] = 1;
```

```
            Travers(X + 1, Y);
return;
        }
        //2
if (EdgePoints[X + 1, Y - 1] == 2)
        {
            EdgeMap[X + 1, Y - 1] = 1;
            VisitedMap[X + 1, Y - 1] = 1;
            Travers(X + 1, Y - 1);
return;
        }

        //3

if (EdgePoints[X, Y - 1] == 2)
        {
            EdgeMap[X, Y - 1] = 1;
            VisitedMap[X, Y - 1] = 1;
            Travers(X, Y - 1);
return;
        }

        //4

if (EdgePoints[X - 1, Y - 1] == 2)
        {
            EdgeMap[X - 1, Y - 1] = 1;
            VisitedMap[X - 1, Y - 1] = 1;
            Travers(X - 1, Y - 1);
return;
        }
        //5
if (EdgePoints[X - 1, Y] == 2)
        {
            EdgeMap[X - 1, Y] = 1;
            VisitedMap[X - 1, Y] = 1;
            Travers(X - 1, Y);
return;
        }
        //6
if (EdgePoints[X - 1, Y + 1] == 2)
        {
            EdgeMap[X - 1, Y + 1] = 1;
            VisitedMap[X - 1, Y + 1] = 1;
            Travers(X - 1, Y + 1);
return;
        }
        //7
if (EdgePoints[X, Y + 1] == 2)
        {
```

```
                EdgeMap[X, Y + 1] = 1;
                VisitedMap[X, Y + 1] = 1;
                Travers(X, Y + 1);
return;
            }
            //8

if (EdgePoints[X + 1, Y + 1] == 2)
            {
                EdgeMap[X + 1, Y + 1] = 1;
                VisitedMap[X + 1, Y + 1] = 1;
                Travers(X + 1, Y + 1);
return;
            }
            //VisitedMap[X, Y] = 1;
return;
        }

        //Canny Class Ends
    }
}
using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;
using System.Drawing.Drawing2D;

namespace ImgProImpl
{
public class ImageHandler
    {
private string _bitmapPath;
private Bitmap _currentBitmap;
private Bitmap _bitmapbeforeProcessing;
private Bitmap _bitmapPrevCropArea;

public ImageHandler()
        {
        }
public Bitmap CurrentBitmap
        {
get
            {
if (_currentBitmap == null)
                    _currentBitmap = new Bitmap(1, 1);
return _currentBitmap;
            }
set { _currentBitmap = value; }
        }
```

```
public Bitmap BitmapBeforeProcessing
        {
get { return _bitmapbeforeProcessing; }
set { _bitmapbeforeProcessing = value; }
        }

public string BitmapPath
        {
get { return _bitmapPath; }
set { _bitmapPath = value; }
        }

public enum ColorFilterTypes
        {
            Red,
            Green,
            Blue
        };

public void ResetBitmap()
        {
if (_currentBitmap != null && _bitmapbeforeProcessing != null)
            {
                Bitmap temp = (Bitmap)_currentBitmap.Clone();
                _currentBitmap = (Bitmap)_bitmapbeforeProcessing.Clone();
                _bitmapbeforeProcessing = (Bitmap)temp.Clone();
}
        }

public void SaveBitmap(string saveFilePath)
        {
            _bitmapPath = saveFilePath;
if (System.IO.File.Exists(saveFilePath))
System.IO.File.Delete(saveFilePath);
            _currentBitmap.Save(saveFilePath);
        }

public void ClearImage()
        {
            _currentBitmap = new Bitmap(1, 1);
        }
public void RestorePrevious()
        {
            _bitmapbeforeProcessing = _currentBitmap;
        }

public void SetColorFilter(ColorFilterTypes colorFilterType)
        {
            Bitmap temp = (Bitmap)_currentBitmap;
            Bitmap bmap = (Bitmap)temp.Clone();
```

```
            Color c;
for (int i = 0; i < bmap.Width; i++)
            {
for (int j = 0; j < bmap.Height; j++)
                {
                    c = bmap.GetPixel(i, j);
int nPixelR = 0;
int nPixelG = 0;
int nPixelB = 0;
if (colorFilterType == ColorFilterTypes.Red)
                    {
nPixelR = c.R;
nPixelG = c.G - 255;
nPixelB = c.B - 255;
                    }
else if (colorFilterType == ColorFilterTypes.Green)
                    {
nPixelR = c.R - 255;
nPixelG = c.G;
nPixelB = c.B - 255;
                    }
else if (colorFilterType == ColorFilterTypes.Blue)
                    {
nPixelR = c.R - 255;
nPixelG = c.G - 255;
nPixelB = c.B;
                    }

nPixelR = Math.Max(nPixelR, 0);
nPixelR = Math.Min(255, nPixelR);

nPixelG = Math.Max(nPixelG, 0);
nPixelG = Math.Min(255, nPixelG);

nPixelB = Math.Max(nPixelB, 0);
nPixelB = Math.Min(255, nPixelB);

bmap.SetPixel(i, j, Color.FromArgb((byte)nPixelR, (byte)nPixelG, (byte)nPixelB));
                }
            }
            _currentBitmap = (Bitmap)bmap.Clone();
        }

public void SetGamma(double red, double green, double blue)
        {
            Bitmap temp = (Bitmap)_currentBitmap;
            Bitmap bmap = (Bitmap)temp.Clone();
            Color c;
byte[] redGamma = CreateGammaArray(red);
byte[] greenGamma = CreateGammaArray(green);
```

```
byte[] blueGamma = CreateGammaArray(blue);
for (int i = 0; i < bmap.Width; i++)
        {
for (int j = 0; j < bmap.Height; j++)
            {
                c = bmap.GetPixel(i, j);
bmap.SetPixel(i, j, Color.FromArgb(redGamma[c.R], greenGamma[c.G], blueGamma[c.B]));
            }
        }
        _currentBitmap = (Bitmap)bmap.Clone();
    }

private byte[] CreateGammaArray(double color)
    {
byte[] gammaArray = new byte[256];
for (int i = 0; i < 256; ++i)
        {
gammaArray[i] = (byte)Math.Min(255, (int)((255.0 * Math.Pow(i / 255.0, 1.0 / color)) +
0.5));
        }
return gammaArray;
    }

public void SetBrightness(int brightness)
    {
        Bitmap temp = (Bitmap)_currentBitmap;
        Bitmap bmap = (Bitmap)temp.Clone();
if (brightness < -255) brightness = -255;
if (brightness > 255) brightness = 255;
        Color c;
for (int i = 0; i < bmap.Width; i++)
        {
for (int j = 0; j < bmap.Height; j++)
            {
                c = bmap.GetPixel(i, j);
int cR = c.R + brightness;
int cG = c.G + brightness;
int cB = c.B + brightness;

if (cR < 0) cR = 1;
if (cR > 255) cR = 255;

if (cG < 0) cG = 1;
if (cG > 255) cG = 255;

if (cB < 0) cB = 1;
if (cB > 255) cB = 255;

bmap.SetPixel(i, j, Color.FromArgb((byte)cR, (byte)cG, (byte)cB));
            }
```

```
            }
            _currentBitmap = (Bitmap)bmap.Clone();
        }

public void SetContrast(double contrast)
        {
            Bitmap temp = (Bitmap)_currentBitmap;
            Bitmap bmap = (Bitmap)temp.Clone();
if (contrast < -100) contrast = -100;
if (contrast > 100) contrast = 100;
contrast = (100.0 + contrast) / 100.0;
contrast *= contrast;
            Color c;
for (int i = 0; i < bmap.Width; i++)
            {
for (int j = 0; j < bmap.Height; j++)
                {
                    c = bmap.GetPixel(i, j);
double pR = c.R / 255.0;
pR -= 0.5;
pR *= contrast;
pR += 0.5;
pR *= 255;
if (pR < 0) pR = 0;
if (pR > 255) pR = 255;

double pG = c.G / 255.0;
                    pG -= 0.5;
                    pG *= contrast;
                    pG += 0.5;
                    pG *= 255;
if (pG < 0) pG = 0;
if (pG > 255) pG = 255;
double pB = c.B / 255.0;
pB -= 0.5;
pB *= contrast;
pB += 0.5;
pB *= 255;
if (pB < 0) pB = 0;
if (pB > 255) pB = 255;

bmap.SetPixel(i, j, Color.FromArgb((byte)pR, (byte)pG, (byte)pB));
                }
            }
            _currentBitmap = (Bitmap)bmap.Clone();
        }

public void SetGrayscale()
        {
            Bitmap temp = (Bitmap)_currentBitmap;
```

```
            Bitmap bmap = (Bitmap)temp.Clone();
            Color c;
for (int i = 0; i < bmap.Width; i++)
            {
for (int j = 0; j < bmap.Height; j++)
                {
                    c = bmap.GetPixel(i, j);
byte gray = (byte)(.299 * c.R + .587 * c.G + .114 * c.B);

bmap.SetPixel(i, j, Color.FromArgb(gray, gray, gray));
                }
            }
            _currentBitmap = (Bitmap)bmap.Clone();
        }

public void SetInvert()
        {
            Bitmap temp = (Bitmap)_currentBitmap;
            Bitmap bmap = (Bitmap)temp.Clone();
            Color c;
for (int i = 0; i < bmap.Width; i++)
            {
for (int j = 0; j < bmap.Height; j++)
                {
                    c = bmap.GetPixel(i, j);
bmap.SetPixel(i, j, Color.FromArgb(255 - c.R, 255 - c.G, 255 - c.B));
                }
            }
            _currentBitmap = (Bitmap)bmap.Clone();
        }
public void Resize(int newWidth, int newHeight)
        {
if (newWidth != 0 && newHeight != 0)
            {
                Bitmap temp = (Bitmap)_currentBitmap;
                Bitmap bmap = new Bitmap(newWidth, newHeight, temp.PixelFormat);

double nWidthFactor = (double)temp.Width / (double)newWidth;
double nHeightFactor = (double)temp.Height / (double)newHeight;

double fx, fy, nx, ny;
int cx, cy, fr_x, fr_y;
                Color color1 = new Color();
                Color color2 = new Color();
                Color color3 = new Color();
                Color color4 = new Color();
byte nRed, nGreen, nBlue;

byte bp1, bp2;
```

```
for (int x = 0; x < bmap.Width; ++x)
            {
for (int y = 0; y < bmap.Height; ++y)
                {

                    fr_x = (int)Math.Floor(x * nWidthFactor);
                    fr_y = (int)Math.Floor(y * nHeightFactor);
cx = fr_x + 1;
if (cx >= temp.Width) cx = fr_x;
cy = fr_y + 1;
if (cy >= temp.Height) cy = fr_y;
fx = x * nWidthFactor - fr_x;
fy = y * nHeightFactor - fr_y;
nx = 1.0 - fx;
ny = 1.0 - fy;

                    color1 = temp.GetPixel(fr_x, fr_y);
                    color2 = temp.GetPixel(cx, fr_y);
                    color3 = temp.GetPixel(fr_x, cy);
                    color4 = temp.GetPixel(cx, cy);

                    // Blue
                    bp1 = (byte)(nx * color1.B + fx * color2.B);

                    bp2 = (byte)(nx * color3.B + fx * color4.B);

nBlue = (byte)(ny * (double)(bp1) + fy * (double)(bp2));
                    // Green
                    bp1 = (byte)(nx * color1.G + fx * color2.G);

                    bp2 = (byte)(nx * color3.G + fx * color4.G);

nGreen = (byte)(ny * (double)(bp1) + fy * (double)(bp2));

                    // Red
                    bp1 = (byte)(nx * color1.R + fx * color2.R);

                    bp2 = (byte)(nx * color3.R + fx * color4.R);

nRed = (byte)(ny * (double)(bp1) + fy * (double)(bp2));

                    bmap.SetPixel(x, y, System.Drawing.Color.FromArgb(255, nRed, nGreen,
nBlue));
                }
            }
            _currentBitmap = (Bitmap)bmap.Clone();
        }
    }

public void RotateFlip(RotateFlipType rotateFlipType)
```

```
        {
            Bitmap temp = (Bitmap)_currentBitmap;
            Bitmap bmap = (Bitmap)temp.Clone();
bmap.RotateFlip(rotateFlipType);
            _currentBitmap = (Bitmap)bmap.Clone();
        }

public void Crop(int xPosition, int yPosition, int width, int height)
        {
            Bitmap temp = (Bitmap)_currentBitmap;
            Bitmap bmap = (Bitmap)temp.Clone();
if (xPosition + width > _currentBitmap.Width)
width = _currentBitmap.Width - xPosition;
if (yPosition + height > _currentBitmap.Height)
height = _currentBitmap.Height - yPosition;
            Rectangle rect = new Rectangle(xPosition, yPosition, width, height);
            _currentBitmap = (Bitmap)bmap.Clone(rect, bmap.PixelFormat);
        }

public void DrawOutCropArea(int xPosition, int yPosition, int width, int height)
        {
            _bitmapPrevCropArea = (Bitmap)_currentBitmap;
            Bitmap bmap = (Bitmap)_bitmapPrevCropArea.Clone();
            Graphics gr = Graphics.FromImage(bmap);
            Brush cBrush = new Pen(Color.FromArgb(150, Color.White)).Brush;
            Rectangle rect1 = new Rectangle(0, 0, _currentBitmap.Width, yPosition);
            Rectangle rect2 = new Rectangle(0, yPosition, xPosition, height);
            Rectangle rect3 = new Rectangle(0, (yPosition + height), _currentBitmap.Width,
_currentBitmap.Height);
            Rectangle rect4 = new Rectangle((xPosition + width), yPosition,
(_currentBitmap.Width - xPosition - width), height);
gr.FillRectangle(cBrush, rect1);
gr.FillRectangle(cBrush, rect2);
gr.FillRectangle(cBrush, rect3);
gr.FillRectangle(cBrush, rect4);
            _currentBitmap = (Bitmap)bmap.Clone();
        }

public void RemoveCropAreaDraw()
        {
            _currentBitmap = (Bitmap)_bitmapPrevCropArea.Clone();
        }

public void InsertText(string text, int xPosition, int yPosition, string fontName, float fontSize,
string fontStyle, string colorName1, string colorName2)
        {
            Bitmap temp = (Bitmap)_currentBitmap;
            Bitmap bmap = (Bitmap)temp.Clone();
            Graphics gr = Graphics.FromImage(bmap);
if (string.IsNullOrEmpty(fontName))
```

```
fontName = "Times New Roman";
if (fontSize.Equals(null))
fontSize = 10.0F;
        Font font = new Font(fontName, fontSize);
if (!string.IsNullOrEmpty(fontStyle))
        {
           FontStyle fStyle = FontStyle.Regular;
switch (fontStyle.ToLower())
           {
case "bold":
fStyle = FontStyle.Bold;
break;
case "italic":
fStyle = FontStyle.Italic;
break;
case "underline":
fStyle = FontStyle.Underline;
break;
case "strikeout":
fStyle = FontStyle.Strikeout;
break;

           }
font = new Font(fontName, fontSize, fStyle);
        }
if (string.IsNullOrEmpty(colorName1))
           colorName1 = "Black";
if (string.IsNullOrEmpty(colorName2))
           colorName2 = colorName1;
        Color color1 = Color.FromName(colorName1);
        Color color2 = Color.FromName(colorName2);
int gW = (int)(text.Length * fontSize);
gW = gW == 0 ? 10 : gW;
        LinearGradientBrush LGBrush = new LinearGradientBrush(new Rectangle(0, 0, gW,
(int)fontSize), color1, color2, LinearGradientMode.Vertical);
gr.DrawString(text, font, LGBrush, xPosition, yPosition);
        _currentBitmap = (Bitmap)bmap.Clone();
    }

public void InsertImage(string imagePath, int xPosition, int yPosition)
    {
        Bitmap temp = (Bitmap)_currentBitmap;
        Bitmap bmap = (Bitmap)temp.Clone();
        Graphics gr = Graphics.FromImage(bmap);
if (!string.IsNullOrEmpty(imagePath))
        {
           Bitmap i_bitmap = (Bitmap)Bitmap.FromFile(imagePath);
           Rectangle rect = new Rectangle(xPosition, yPosition, i_bitmap.Width,
i_bitmap.Height);
gr.DrawImage(Bitmap.FromFile(imagePath), rect);
```

```
            }
            _currentBitmap = (Bitmap)bmap.Clone();
        }

public void InsertShape(string shapeType, int xPosition, int yPosition, int width, int height,
string colorName)
        {
            Bitmap temp = (Bitmap)_currentBitmap;
            Bitmap bmap = (Bitmap)temp.Clone();
            Graphics gr = Graphics.FromImage(bmap);
if (string.IsNullOrEmpty(colorName))
colorName = "Black";
            Pen pen = new Pen(Color.FromName(colorName));
switch (shapeType.ToLower())
            {
case "filledellipse":
gr.FillEllipse(pen.Brush, xPosition, yPosition, width, height);
break;
case "filledrectangle":
gr.FillRectangle(pen.Brush, xPosition, yPosition, width, height);
break;
case "ellipse":
gr.DrawEllipse(pen, xPosition, yPosition, width, height);
break;
case "rectangle":
default:
gr.DrawRectangle(pen, xPosition, yPosition, width, height);
break;

            }
            _currentBitmap = (Bitmap)bmap.Clone();
        }
    }
}
```