西安交通大學

School of Computer Science and Technology

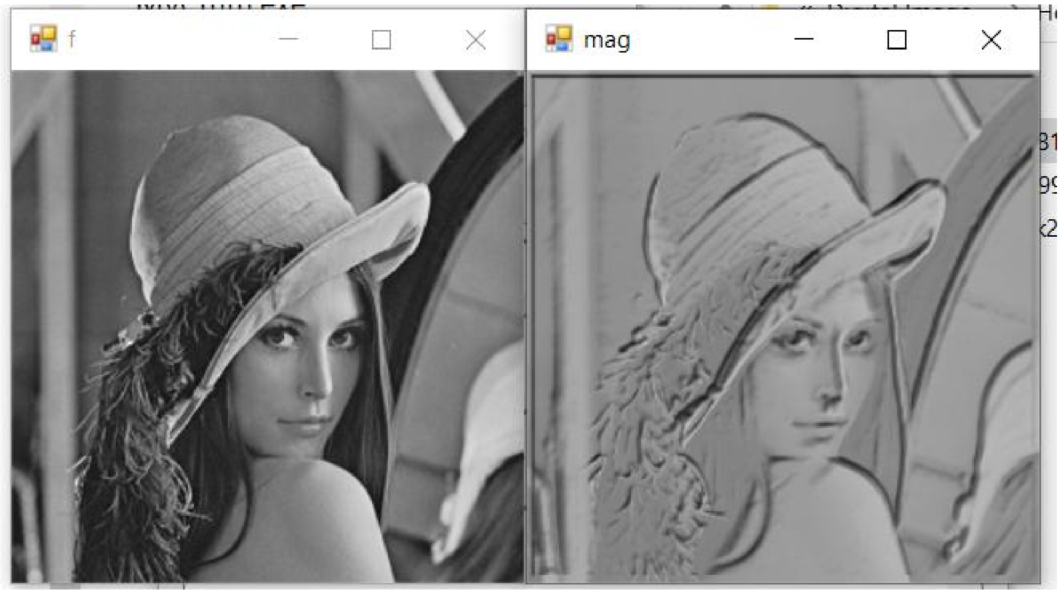
Name:	MD MAHFUZUR RAHMAN
Student ID:	3123999081
Course Name:	Digital Image Processing
Instructor:	梁毅军
Homework:	02
Submission Date:	12.11.23
Email:	the_bengal_tiger@stu.xjtu.edu.cn
WeChat:	MahfuzRonnie

Source Code

```
/*
  /*    This function performs convolution using a given mask on the input image f.
     It is a generic filter function that can be used for various operations.
*/

byte[,] filter(byte[,] f,float[,] mask)
{
  int w = f.GetLength(0);
  int h = f.GetLength(1);

  int M = mask.GetLength(0)/2;
  int N = mask.GetLength(1)/2;

  byte[,] g = new byte[w,h];

  for (int y=N;y<h-N;y++)
    for (int x=M;x<w-M;x++)
    {
       float sum = 0;
       for (int m=-M;m<=M;m++)
          for (int n=-N;n<=N;n++)
              sum+=f[x+m,y+n]*mask[M+m,N+n];
       g[x,y]=S(sum);
    }

  return g;
}


// Similar to filter but with an additional offset of 128 applied to the result, effectively shifting the
intensity range.
byte[,] filter_highpass(byte[,] f,float[,] mask)
{
  int w = f.GetLength(0);
  int h = f.GetLength(1);

  int M = mask.GetLength(0)/2;
  int N = mask.GetLength(1)/2;

  byte[,] g = new byte[w,h];

  for (int y=N;y<h-N;y++)
    for (int x=M;x<w-M;x++)
    {
       float sum = 0;
       for (int m=-M;m<=M;m++)
          for (int n=-N;n<=N;n++)
              sum+=f[x+m,y+n]*mask[M+m,N+n];
       g[x,y]=S(sum + 128);
    }

  return g;
}


// Performs brightness and contrast adjustment on the input image f using parameters k and c
byte S(double v)
{
  if (v<0) return 0;
  if (v>255) return 255;
  return (byte)v;
}

byte[,] bca(byte[,] f,double k,double c)
{
  int w = f.GetLength(0);
  int h = f.GetLength(1);

  byte[,] g = new byte[w,h];

  for (int y=0;y<h;y++)
    for (int x=0;x<w;x++)
       g[x,y]=S((f[x,y]-128)*k+128+c);

  return g;
}

// Decreases the intensity of each pixel by 130, effectively creating a darkened version of the image
byte[,] dark(byte[,] f)
{
  int w = f.GetLength(0);
  int h = f.GetLength(1);

  byte[,] g = new byte[w,h];

  for (int y=0;y<h;y++)
    for (int x=0;x<w;x++)
       g[x,y]=S(f[x,y]-130);

  return g;
}


// Generates a Gaussian mask based on the given radius r
float[,] g_mask(double r)
{
   int M = (int)(3*r);
   if (M<1) M = 1;
   int N = M;

  float[,] mask= new float[2*M+1,2*N+1];

  for (int n=-N;n<=N;n++)
    for (int m=-M;m<=M;m++)
       mask[m+M,n+N] = (float)Exp(-(m*m+n*n)/(2*r*r));

  float s = 0;
  for (int n=-N;n<=N;n++)
    for (int m=-M;m<=M;m++)
       s+=mask[m+M,n+N];

  for (int n=-N;n<=N;n++)
    for (int m=-M;m<=M;m++)
       mask[m+M,n+N] /= s;

  return mask;
}
```

```
// Generates a directional mask for edge detection based on the given angle a
float[,] dir_g(double a)
{
   float k1 = (float)Cos(a);
   float k2 = (float)Sin(a);

   float[,] mask = new float[3,3];

   mask[0,0] =-k1-k2;   mask[1,0] =-2*k2;   mask[2,0] =k1-k2;
   mask[0,1] =-2*k1;    mask[1,1] = 0;      mask[2,1] = 2*k1;
   mask[0,2] =-k1+k2;   mask[1,2] =2*k2;    mask[2,2] =k1+k2;

  return mask;
}


// Computes the magnitude of gradients from four input images representing edges in different
directions
byte[,] get_meg(byte[,] f1, byte[,] f2, byte[,] f3, byte[,] f4)
{
   int w = f1.GetLength(0);
   int h = f1.GetLength(1);

   byte[,] g = new byte[w,h];
   for (int y=1;y<h-1;y++)
         for (int x=1;x<w-1;x++)
             g[x,y] = S(Sqrt((Pow(f1[x, y], 2) / 4 + Pow(f2[x, y], 2) / 4+ Pow(f3[x, y], 2) / 4+ Pow(f4[x, y],
2) / 4)));
   return g;
}


// Performs element-wise subtraction between two images
byte[,] minus(byte[,] img1, byte[,] img2)
{
    int width = img1.GetLength(0);
    int height = img1.GetLength(1);

    byte[,] result = new byte[width, height];

    for (int y = 0; y < height; y++)
    {
        for (int x = 0; x < width; x++)
        {
            int subtractionResult = img1[x, y] - img2[x, y];

            result[x, y] = S(subtractionResult);
        }
    }

    return result;
}


// Calls the functions sequentially to process and display the final result
void main()
{
   byte[,] f = LoadImg();
   ShowImg("f",f);
   double pi = 3.14159;
   byte[,] Gblur_img = filter(f, g_mask(1));

   byte[,] edge_0 = filter_highpass(Gblur_img, dir_g(0));
   byte[,] edge_45 = filter_highpass(Gblur_img, dir_g(pi / 4));
   byte[,] edge_90 = filter_highpass(Gblur_img, dir_g(pi / 2));
   byte[,] edge_135 = filter_highpass(Gblur_img, dir_g(pi * 3/4));

   byte[,] edge = dark(get_meg(edge_0, edge_45, edge_90, edge_135));
   byte[,] img = bca(f, 0.3, 30);

   ShowImg("mag", minus(img , edge));

}
```

Input and Output Image



Algorithm Description

Step 1: Loading and Displaying the Original Image

```
byte[,] f = LoadImg();
ShowImg("f", f);
```

- The original image is loaded into the 2D array f.
- The ShowImg function is called to display the original image.

Step 2: Gaussian Blur

```
byte[,] Gblur_img = filter(f, g_mask(1));
```

- The image is convolved with a Gaussian mask (g_mask) to perform Gaussian blurring.
- The result is stored in the 2D array Gblur_img.

Step 3: Edge Detection in Different Directions

```
byte[,] edge_0 = filter_highpass(Gblur_img, dir_g(0));byte[,] edge_45 =
filter_highpass(Gblur_img, dir_g(pi / 4));byte[,] edge_90 =
filter_highpass(Gblur_img, dir_g(pi / 2));byte[,] edge_135 =
filter_highpass(Gblur_img, dir_g(pi * 3/4));
```

- Edge detection is performed in four directions (0 degrees, 45 degrees, 90 degrees, and 135 degrees) using the filter_highpass function.
- The results are stored in edge_0, edge_45, edge_90, and edge_135.

Step 4: Combining Edge Information

```
byte[,] edge = dark(get_meg(edge_0, edge_45, edge_90, edge_135));
```

- The edge information from different directions is combined using the get_meg function.
- The result is stored in the 2D array edge.

Step 5: Brightness and Contrast Adjustment

```
byte[,] img = bca(f, 0.3, 30);
```

- The brightness and contrast of the original image are adjusted using the bca function.
- The result is stored in the 2D array img.

Step 6: Displaying the Resulting Image

```
ShowImg("mag", minus(img, edge));
```

- The final result is obtained by subtracting the combined edge information (edge) from the adjusted original image (img).
- The resulting image is displayed using the ShowImg function.