实验 5 图像增强

【1】实验目的：掌握图像增强的基本原理及方法，实现常用图像增强算法

【2】实验原理：

图像增强是对图像灰度进行改变，增强图像信息的基本算法，主要可分为空域处理（包含滤波）和频域滤波。

2.1 图像灰度拉伸

图像灰度变换是以像素为单位，对图像每一像素的灰度实施某种变换，通常可以表示为：$g(x,y)=T[f(x,y)]$，其中T表示变换函数，由于处理时把输入图像的每一点映射为输出图像对应的像素点，也称为点运算。实际使用中，可根据图像增强的需要，选择不同的变换函数对图像进行处理以达到我们的要求。通过本部分的实验，让学生深入理解图像灰度变换的基本原理，学习编程实现图像灰度变换，并分析各种算法的效果。

（1）对比度调整

如果原图像 f(x, y)的灰度范围是[m, M]，我们希望对图像的灰度范围进行线性调整，调整后的图像 g(x, y)的灰度范围是[n, N]，那么下述变换：

$$g(x,y)=\frac{N-n}{M-m}[f(x,y)-m]+n$$

就可以实现这一要求。

（2）线性变换、分段线性变换及非线性变换

线性变换

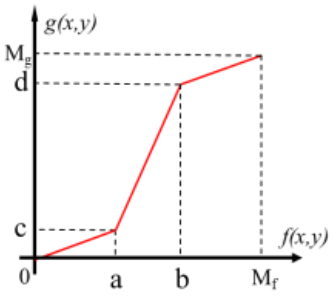
线性变换使用线性算子对图像灰度进行变换，设原图像灰度主 r,变换后灰度为 s，则可表示为 $s=kr+b$,其中 k 表示变换的性质，通常表示变换线性函数的斜率，b 表示初始值，即在新灰度轴的截距。

分段线性变换

有时单一斜率的线性变换可能不满足要求，需要对图像整体进行考虑，提高感兴趣的灰度范围，抑制像素较少或不感兴趣的区域，可以不同的灰度范围使用不同的斜率：

设 f(x,y)灰度范围为[0,Mf]，g(x,y)灰度范围为[0,Mg],则有如下关系：

$$g(x,y)=\begin{cases}\dfrac{M_g-d}{M_f-b}[f(x,y)-b]+d & b\le f(x,y)\le M_f\\ \dfrac{d-c}{b-a}[f(x,y)-a]+c & a\le f(x,y)<b\\ \dfrac{c}{a}f(x,y) & 0\le f(x,y)<a\end{cases}$$



非线性变换

非线性点运算的输出灰度级与输入灰度级呈非线性关系，常见的非线性灰度变换为对数变换和幂次变换。对数变换：$s=c*lg(1+f(x,y))$，幂变换：$z=c*[f(x,y)]^{\gamma}$，也

可使用指数函数$z = c * a^{[f(x,y)]}$对图像进行变换。

示例 1，建立工程，编写线性变换的函数，并建立主调函数，测试线性变换函数的功能。

详细理论见课本及相关参考书

```
void mylinetrans(cv::Mat &inimage, cv::Mat &outimage,float k,float b)
{
    outimage.create(inimage.size(), inimage.type());
    int h = inimage.rows;
    int w = inimage.cols*inimage.channels();
    for (int y = 0; y < h;y++)
    for (int x = 0; x < w; x++)
    {
        outimage.data[y*w + x] = cv::saturate_cast<uchar>(inimage.data[y*w + x] * k + b);
    }
    }
```

对数变换示例代码：

```
void exp2_2()
{
    cv::Mat image1 = cv::imread("d:\\mary.bmp", -1);
    cv::namedWindow("the first image", CV_WINDOW_NORMAL);
    cv::imshow("the first image", image1);
    int h = image1.rows;
    int w = image1.cols*image1.channels();
    cv::Mat outimage;
    outimage.create(image1.size(), image1.type());
    for (int y = 0; y < h; y++)
    for (int x = 0; x < w; x++)
    {
        double temp = 100*log10(image1.data[y*w + x] +1);
        if (temp>255)
            temp = 255;
        outimage.data[y*w + x] = cv::saturate_cast<uchar>(temp);

    }
    cv::imshow("log(image)", outimage);
    cv::waitKey(0);
}
```

2.2 直方图均衡化增强图像

具体原理请查阅课本，下面给出教师自编的直方图均衡化函数，请根据函数代码，深入体会算法实现的原理，使用代码进行后续实验。

```
void equahist(const cv::Mat &image, cv::Mat &result)
{
    double count[256] = { 0 };
```

```
long mymax(0);
float map[256] = { 0 };
result.create(image.size(), image.type());
long x, y;
int w = image.cols*image.channels();
int h = image.rows;
for (y = 0; y<h; y++)
for (x = 0; x<w; x++)
{
    count[*(image.data + y*w + x)]++;
}
for (int i = 0; i < 256; i++)
{
    map[i] = 0;
    float t = 0;
    for (int j = 0; j <= i; j++)
    {
        t+= 1.0*count[j]/w/h;
    }
    map[i] = t;

}
for (int y = 0; y < h;y++)
for (int x = 0; x < w; x++)
{
    result.data[y*w + x] =
        cv::saturate_cast<uchar>(map[*(image.data+y*w + x)] * 255);
}
}
```

2.3 图像空域滤波原理

使用图像像素之间的积分运算使图像钝化，通常有邻域平均算法，即使用模板邻域中的像素进行平均值或加权平均值运算得到输出像素灰度值的方法，可以弱化噪声，但图像模糊。中值滤波：对待处理像素模板邻域内的像素灰度进行排序，然后取中间值作为输出值的方法，可以较好消除椒盐噪声。图像锐化主要是使用微分算法对图像进行处理，通常有一阶和二阶算子，如梯度算子，交叉微分算子（Roberts）、Sobel 算子等，二阶的有拉普拉斯算子，高斯算子等，详细见课本。

1. 邻域平均：对要处理的像素使用给定模板对其求加权平均值的过程，示例代码如下：

```
void myaverage(cv::Mat &image, cv::Mat &result,int m,int n)
{
  result.create(image.size(), image.type());
  if (image.channels() != 1)
      return;
```

```
  int w = image.cols;
  int h = image.rows;
  int k(0);
  int l(0);
  int iresult(0);
  for (int y = m/2; y < h-m/2;y++)
  for (int x = n/2; x < w-n/2; x++)
  {
      iresult = 0;
      for (k = -m / 2; k < m / 2;k++)
      for (l = -n / 2; l < m / 2; l++)
      {
          iresult += image.data[(y + k)*w + x + l];
      }
      result.data[y*w + x] = cv::saturate_cast<uchar>(iresult/m/n);

  }
}
```

中值滤波:

起泡排序:

```
void BubbleSort(int *a, int n)
{
  int i,j;
  bool flag=0;
  unsigned char temp=0;
  for(i=0;i<n;i++)
  {
      flag=0;
      for(j=0;j<n-i;j++)
      {

          if(a[j]>a[j+1])
          {
              temp=a[j];
              a[j]=a[j+1];
              a[j+1]=temp;
              flag=1;
          }
                  }
      if(flag==0)
              return;
  }
}
```

2．中值滤波：对要处理的像素周围模板范围内的像素进行排序，然后取中间

值的算法，示例代码如下：

```
void medifilter(cv::Mat &inimage, cv::Mat &outimage,int m,int n)
{
  outimage.create(inimage.size(), inimage.type());
  int w = inimage.cols*inimage.channels();
  int h = inimage.rows;
  int y, x;
  uchar *map = new uchar[m*n];
  for (y = m / 2; y < h - m / 2; y++)
  for (x = n / 2; x < w - n / 2; x++)
  {
      for (int k = -m / 2; k <= m / 2;k++)
      for (int l = -n / 2; l <= n / 2; l++)
      {
          map[(k + m / 2)*n + l + n / 2] = inimage.data[(y + k)*w + x +
l];
      }
      bubblesort(map, m*n);
      outimage.data[y*w + x] = map[m*n / 2];

  }

}
```

3. 图像锐化的算法代码示例，参数 type 表示锐化算子类型。

```
void ruihua(cv::Mat &image, cv::Mat &result, int type)
{
result.create(image.size(), image.type());
  //if (image.channels() != 1)
  //  return;
  int w = image.cols*image.channels();
  int h = image.rows;
  int y, x;
  switch (type)
  {
  case 0:
      for (y = 0; y < h - 1; y++)
      {
          unsigned char *lpsrc = image.ptr<uchar>(y);
          unsigned char *lpsrc1 = image.ptr<uchar>(y + 1);
          for (x = 0; x < w - 1; x++)
          {
              float temp = abs(lpsrc[x] - lpsrc[x+1]) + abs(lpsrc1[x]-
lpsrc[x]);
```

```
            result.data[y*w + x] = cv::saturate_cast<uchar>(temp);
        }

    }

    break;
  case 1:
    for (y = 0; y < h - 1; y++)
    {
        unsigned char *lpsrc = image.ptr<uchar>(y);
        unsigned char *lpsrc1 = image.ptr<uchar>(y + 1);
        for (x = 0; x < w - 1; x++)
        {
            float temp = abs(lpsrc[x] - lpsrc1[x + 1]) +
abs(lpsrc1[x] - lpsrc[x+1]);
            result.data[y*w + x] = cv::saturate_cast<uchar>(temp);
        }

    }

    break;
  case 2:
    for (y = 1; y < h - 1; y++)
    {
        unsigned char *lpsrc = image.ptr<uchar>(y);
        unsigned char *lpsrc1 = image.ptr<uchar>(y + 1);
        uchar *lpsrc2 = image.ptr<uchar>(y - 1);
        for (x = 1; x < w - 1; x++)
        {
            float sx = 2 * (lpsrc[x + 1] - lpsrc[x - 1]) + (lpsrc1[x
+ 1] - lpsrc1[x - 1]) +
                (lpsrc2[x + 1] - lpsrc2[x - 1]);
            float sy = 2 * (lpsrc2[x] - lpsrc1[x]) + (lpsrc2[x + 1] -
lpsrc1[x + 1]) +
                (lpsrc2[x - 1] - lpsrc1[x - 1]);
            uchar temp = cv::saturate_cast<uchar>(abs(sx) + abs(sy));
            result.data[y*w + x] = temp;
        }

    }
    break;
  case 3:
    for (y = 0; y < h - 1; y++)
    {
```

```
            unsigned char *lpsrc = image.ptr<uchar>(y);
            unsigned char *lpsrcnext = image.ptr<uchar>(y + 1);
            unsigned char *lpsrcpre = image.ptr<uchar>(y - 1);
            for (x = 0; x < w - 1; x++)
            {
                int fxy =lpsrc[x];
                int fxy0 = lpsrcpre[x];
                int fxy1 = lpsrcnext[x];
                int fxy2 = lpsrc[x - 1];
                int fxy3 = lpsrc[x + 1];
                int lpls = fxy0 + fxy1 + fxy2 + fxy3 - 4 * fxy;
                result.data[y*w + x] = cv::saturate_cast<uchar>(fxy-
lpls);
            }
        }
        break;
    default:
        break;
    }
}
```

4. 频域滤波实验

图像频域处理是使用傅立叶变换工具，把图像信号变换到频域，通过设计不同滤波器，实现图像滤波处理，按照滤波器的特性，可以分为低通滤波、高通滤波和带通（带阻）滤波器。各滤波器的频率特性如下:

（1） 低通滤波器:

理想低通滤波器:

$$H(u,v)=\begin{cases}1, D(u,v)\le D_0\\ 0, D(u,v)>D_0\end{cases}$$

$$D(u,v)=\sqrt{(u-M/2)^2+(v-N/2)^2}$$

布特沃斯低通滤波器:

$$H(u,v)=\frac{1}{1+[D(u,v)/D_0]^{2n}}$$

高斯低通滤波器: $H(u,v)=exp^{-D^2(u,v)/(2\sigma^2)}$

$$\sigma=D_0$$

（2） 高通滤波器:

$$H(u,v)=\begin{cases}0, D(u,v)\le D_0\\ 1, D(u,v)>D_0\end{cases}$$

理想高通滤波器: $D(u,v)=\sqrt{(u-M/2)^2+(v-N/2)^2}$

布特沃斯高通滤波器：$$H(u,v)=\frac{1}{1+[D_0/D(u,v)]^{2n}}$$

高斯高通滤波器：$$H(u,v)=1-\exp^{-D^2(u,v)/(2\sigma^2)}$$ $$\sigma=D_0$$

（3） 带通（带阻）滤波器:

理想带通：

$$H(u,v)=\begin{cases}1, D_0-W/2\le D(u,v)\le D_0+W/2;\\ 0, others\end{cases}$$

布特沃斯带阻:

$$H(u,v)=\frac{1}{1+\left[\frac{DW}{D^2-D_0^2}\right]^{2n}}$$

高斯带阻:

$$H(u,v)=1-e^{-\left[\frac{D^2-D_0^2}{DW}\right]}$$

滤波器实现的代码如下，以低通滤波器为例：

（对图像进行傅立叶变换并显示频谱）

```
Mat mydft(cv::Mat srcImage)
{
        //2 将输入图像扩展到最佳尺寸,边界用 0 补充
    int m = getOptimalDFTSize(srcImage.rows);
    int n = getOptimalDFTSize(srcImage.cols);

    //将添加的像素初始化为 0
    Mat padded;
    copyMakeBorder(srcImage, padded, 0, m - srcImage.rows,
        0, n - srcImage.cols, BORDER_CONSTANT, Scalar::all(0));

    //3 为傅里叶变换的结果(实部和虚部)分配存储空间
    //将数组组合合并为一个多通道数组
    Mat planes[] = { Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F) };
```

```
    Mat complexI;
    merge(planes, 2, complexI);

    //4 进行傅里叶变换
    dft(complexI, complexI);
    //idft(complexI, complexI);
    //将复数转换为幅值，即=> log(1 + sqrt(Re(DFT(I))^2 + Im(DFT(I))^2))
    //将多通道数组分离为几个单通道数组
    split(complexI, planes);//planes[0] = Re(DFT(I), planes[1] = Im(DFT(I))
    magnitude(planes[0], planes[1], planes[0]);
    Mat magImage = planes[0];
    //对数尺度缩放
    magImage += Scalar::all(1);
    log(magImage, magImage);//求自然对数
    //剪切和重分布幅度图象限
    //若有奇数行或奇数列，进行频谱剪裁
    magImage = magImage(Rect(0, 0, magImage.cols & -2, magImage.rows & -2));
    //重新排列傅立叶图像中的象限，使得原点位于图像中心
    fftshift(magImage);
        //归一化，用 0 到 1 的浮点值将矩阵变换为可视的图像格式
    normalize(magImage, magImage, 0, 1, NORM_MINMAX);
    return magImage;
}
//对图像频谱进行移位操作，使用频谱分布在中心
void fftshift(cv::Mat& magImage)
{
    magImage = magImage(Rect(0, 0, magImage.cols & -2, magImage.rows & -2));
    int cx = magImage.cols / 2;
    int cy = magImage.rows / 2;
    Mat q0(magImage, Rect(0, 0, cx, cy));
```

```
    Mat q1(magImage, Rect(cx, 0, cx, cy));
    Mat q2(magImage, Rect(0, cy, cx, cy));
    Mat q3(magImage, Rect(cx, cy, cx, cy));
    //交换象限(左上与右下进行交换)
    Mat tmp;
    q0.copyTo(tmp);
    q3.copyTo(q0);
    tmp.copyTo(q3);
    //交换象限（右上与左下进行交换）
    q1.copyTo(tmp);
    q2.copyTo(q1);
    tmp.copyTo(q2);
}
//理想低通滤波器
Mat LowPass(cv::Mat srcImage,int D0)
{
    //2 将输入图像扩展到最佳尺寸,边界用 0 补充
    int m = getOptimalDFTSize(srcImage.rows);
    int n = getOptimalDFTSize(srcImage.cols);

    //将添加的像素初始化为 0
    Mat padded;
    copyMakeBorder(srcImage, padded, 0, m - srcImage.rows,
        0, n - srcImage.cols, BORDER_CONSTANT, Scalar::all(0));

    //3 为傅里叶变换的结果(实部和虚部)分配存储空间
    //将数组组合合并为一个多通道数组
    Mat planes[] = { Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F) };
    Mat complexI;
    merge(planes, 2, complexI);
```

```
//4 进行傅里叶变换
dft(complexI, complexI);
//idft(complexI, complexI);
//将复数转换为幅值，即=> log(1 + sqrt(Re(DFT(I))^2 + Im(DFT(I))^2))
//将多通道数组分离为几个单通道数组
split(complexI, planes);//planes[0] = Re(DFT(I), planes[1] = Im(DFT(I))
//magnitude(planes[0], planes[1], planes[0]);
Mat Re;
Re= planes[0];
Mat Im;
Im= planes[1];
long h = complexI.rows;
long w = complexI.cols;
fftshift(planes[0]);
fftshift(planes[1]);
for(int y=0;y<h;y++)
    for (int x=0; x<w; x++)
    {
        if ((y - h / 2) * (y - h / 2) + (x - w / 2) * (x - w / 2) >D0 * D0)
        {
            planes[0].at<float>(y, x) = 0;
            planes[1].at<float>(y, x) = 0;
        }
    }
Mat complex2;
merge(planes, 2, complex2);
idft(complex2, complex2);
split(complex2, planes);
magnitude(planes[0], planes[1], planes[0]);
```

```
    Mat magImage = planes[0].clone();
    //归一化，用 0 到 1 的浮点值将矩阵变换为可视的图像格式
    normalize(magImage, magImage, 0, 1, NORM_MINMAX);
    return magImage;
}
//布特沃斯滤波器卷积核生成
Mat butterworth_lbrf_kernel(Mat& scr, float sigma, int n)
{
    Mat butterworth_low_pass(scr.size(), CV_32FC1); //，CV_32FC1
    double D0 = sigma;//半径 D0 越小，模糊越大；半径 D0 越大，模糊越小
    for (int i = 0; i < scr.rows; i++) {
        for (int j = 0; j < scr.cols; j++) {
            double d = sqrt(pow((i - scr.rows / 2), 2) + pow((j - scr.cols / 2), 2));//分子,
计算 pow 必须为 float 型
            butterworth_low_pass.at<float>(i, j) = float(1.0 / (1 + pow(d / D0, 2 * n)));
        }
    }
    std::string name = "巴特沃斯低通滤波器 d0=" + std::to_string(sigma) + "n=" +
std::to_string(n);
    imshow(name, butterworth_low_pass);
    return butterworth_low_pass;
}
Mat Butterworth_Low_Paass_Filter(Mat& src, float d0, int n)
{
    //H = 1 / (1+(D/D0)^2n)      n 表示巴特沃斯滤波器的次数
    //调整图像加速傅里叶变换
    int M = getOptimalDFTSize(src.rows);
    int N = getOptimalDFTSize(src.cols);
```

```
        Mat padded;
        copyMakeBorder(src, padded, 0, M - src.rows, 0, N - src.cols, BORDER_CONSTANT, Scalar::all(0));
        padded.convertTo(padded, CV_32FC1); //将图像转换为 float 型
        Mat butterworth_kernel = butterworth_lbrf_kernel(padded, d0, n);//理想低通滤波器
        Mat result = freqfilt(padded, butterworth_kernel);
        return result;
    }
    //使用卷积核进行频域滤波
    Mat freqfilt(Mat& scr, Mat& blur)
    {
        //**********************DFT*******************
        Mat planes[] = { scr, Mat::zeros(scr.size() , CV_32FC1) }; //创建通道，存储 dft 后的实部与虚部（CV_32F，必须为单通道数）
        Mat complexIm;
        merge(planes, 2, complexIm);//合并通道 （把两个矩阵合并为一个 2 通道的 Mat 类容器）
        dft(complexIm, complexIm);//进行傅立叶变换，结果保存在自身
        split(complexIm, planes);//分离通道（数组分离）
        fftshift(planes[0]);
        fftshift(planes[1]);
        //*****************滤波器函数与 DFT 结果的乘积****************
        Mat blur_r, blur_i, BLUR;
        multiply(planes[0], blur, blur_r); //滤波（实部与滤波器模板对应元素相乘）
        multiply(planes[1], blur, blur_i);//滤波（虚部与滤波器模板对应元素相乘）
        Mat plane1[] = { blur_r, blur_i };
        merge(plane1, 2, BLUR);//实部与虚部合并
        idft(BLUR, BLUR);       //idft 结果也为复数
```

```
    split(BLUR, planes);//分离通道，主要获取通道
    magnitude(planes[0], planes[1], planes[0]);   //求幅值(模)
    normalize(planes[0], planes[0], 1, 0, NORM_MINMAX);   //归一化便于显示
    return planes[0];
}
    //主函数调用
    #include "freqapi.h"
    int main()
    {
        Mat img = imread("images/baboon.bmp", -1);
        if (img.data == NULL)
            return -1;
        imshow("testimage", img);
        Mat fre = mydft(img); //生成频谱
        imshow("图像频谱",fre);
       Mat result=Butterworth_Low_Paass_Filter(img, 50, 2);// Butterworth 低通滤波
            imshow("图像低通滤波",result);
    result=LowPass(img, 30);//  理想低通
        imshow("图像低通滤波",result);
        waitKey();
}
```

【3】实验内容及要求：

1、 使用前面建立的工程，参照示例 1，添加四个函数，分别实现图像的线性变换，分段线性变换，对数变换及幂变换。图像及变换系数从参数中传入。在主函数中编写代码，测试上述函数，调整不同的参数，观察图像处理的效果。给出变换的代码及测试的结果。加入代码绘制上述图像处理前后的直方图分布，并进行比较。

2、 编写主函数，从磁盘中调入图像，对图像进行直方图均衡化处理，并比较处理前后图像的直方图。报告中给出直方图均衡化函数的流程图，对上述要求的实现的过程及对结果进行分析。

3、 打开一幅图像，对图像进行各种不同模板大小的邻域平均滤波。观察图像的变化情况并比较变化前后图像直方图的差异；编写排序函数（或使用教师给的排序函数），利用排序函数对图像进行中值滤波处理，并比较不同大小模板的效果；打开一幅灰度图像，对图像进行交叉微分算子、梯度法、及 sobel 算子的一阶锐化处理，比较图像变化前后的直方图；打开一幅图像并进行二阶拉普拉斯算子的锐化处理，并分析算子的作用。

4、 建立工程，配置好环境，使用给定的快速二维傅立叶变换函数或 OpenCV 函数库的离散傅立叶变换函数实现对给定图像的傅立叶变换，观察傅立叶变换后频谱的分布情况，并分析结果；对一幅图像进行傅立叶变换，分别设置虚部为 0 和模为常数，重构图像并观察图像，分析实验结果；读入图像，使用理想低通滤波器和布特沃斯低通滤波器对图像进行频率域的低通滤波。调整滤波器半径，观察不同半径的滤波器实现的效果，并分析原因；读入图像，对图像进行频率域的高通滤波（任意一种）。调整滤波器半径，观察不同半径的滤波器实现的效果，并分析原因；读入图像，并对图像进行傅立叶变换，请选择一对和单个频率点，其他设为零，然后进行傅立叶反变换，观察现象，并分析结果。

【4】实验报告要求：

（1） 写出工程建立及程序编写的步骤。

（2） 编写相关程序。

（3） 写出程序的流程图（简单描述）。

（4） 描述程序的功能。

（5） 给出程序运行结果。

【5】思考题：

1.给你一幅图像，如何选择合适的函数实现灰度变换以增强图像？

2.试说明为什么直方图均衡能实现图像的增强。

3.要求说明图像平滑与锐化的原理，说明各种平滑方法对图像的操作结果有什么不同？

4.说明哪一种锐化方法最好？为什么？