**作业4**

1. 假定我们希望点运算使一输入图像转换为在每一灰度级上都有相同的像素个数的输出图像（即输出的直方图是平的）。这对于在进行图像比较或分割之前将图像转化为一致的格式非常有用。
2. 略
3. 翻译：

虽然点操作是最简单的，但它们包含了所有图像处理操作中最强大、最广泛使用的一些操作。它们在图像预处理中特别有用，在预处理中，需要在绘制主作业之前对图像进行修改。

4.2算术运算这些运算的作用是对图像中的每个灰度值应用一个简单的函数y=f（x）。因此，f（x）是一个将范围0~255映射到其自身上的函数。简单的函数包括给每个像素加上或减去一个常数C：Y=X+C或者将每个像素乘以一个常数：y=Cx。

在每种情况下，我们可能都必须稍微调整输出，以确保结果是0~255范围内的整数。我们可以通过首先对结果进行四舍五入（如有必要）以获得整数，然后通过设置255作为上界（如果y>255）来“剪裁”这些值来实现这一点

虽然点操作是最简单的，但它们包含了所有图像处理操作中最强大和应用最广泛的一些操作。它们在图像预处理中特别有用，在图像预处理中，在尝试主作业之前需要修改图像。

4.2算术操作

这些操作通过应用一个简单的函数来进行

y=f(x)

到图像中的每个灰度值。因此，f(x)是一个映射范围为0··.255到自身上。简单的函数包括为每个像素添加或减去一个常量值:

y=x+C或y=x-C

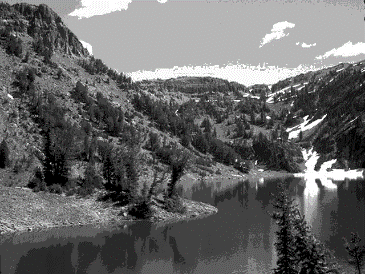
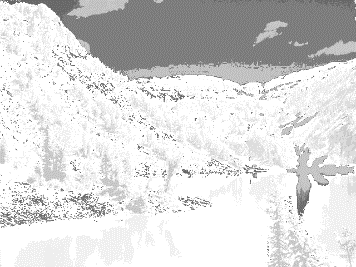
或者将每个像素乘以一个常数:

y=Cx。

在每种情况下，我们可能必须稍微调整输出，以确保结果是0~255的范围，我们可以通过首先舍入结果(如果需要)来获得一个整数，然后通过设置“剪裁”值来实现这一点

如果y大于255，那么取y=255，如果y小于0，那么取y=0。

我们可以通过绘制y=f(x)来了解这些操作如何影响图像。数字4.2是图像中每个像索加或减128的结果。注意，当我们添加128时，所有127或更大的灰色值都将映射到255。当我们减去128时，所有小于或等于128的灰度值都将被映射为0。通过观察这些图表，我们观察到，一般来说，添加一个常数会使图像变亮，而减去一个常数会使图像变暗。

 编程结果：

原图像 图像减100 图像加100

源代码：

void SetColor(int i,RGBQUAD\* &pColorTable,int delta)

{

if(delta>=255){

pColorTable[i].rgbBlue=255;

pColorTable[i].rgbGreen=255;

pColorTable[i].rgbRed=255;

return;

}

if(delta<=-255){

pColorTable[i].rgbBlue=0;

pColorTable[i].rgbGreen=0;

pColorTable[i].rgbRed=0;

return;

}

if(pColorTable[i].rgbBlue+delta<0)pColorTable[i].rgbBlue=0;

else pColorTable[i].rgbBlue+=delta;

if(pColorTable[i].rgbGreen+delta<0)pColorTable[i].rgbGreen=0;

else pColorTable[i].rgbGreen+=delta;

if(pColorTable[i].rgbRed+delta<0)pColorTable[i].rgbRed=0;

else pColorTable[i].rgbRed+=delta;

if(pColorTable[i].rgbBlue+delta>255)pColorTable[i].rgbBlue=255;

else pColorTable[i].rgbBlue+=delta;

if(pColorTable[i].rgbGreen+delta>255)pColorTable[i].rgbGreen=255;

else pColorTable[i].rgbGreen+=delta;

if(pColorTable[i].rgbRed+delta>255)pColorTable[i].rgbRed=255;

else pColorTable[i].rgbRed+=delta; }

void ChangeColor(int biBitCount,RGBQUAD\* &pColorTable,char \*writePath,bool &readFlag){

if(readFlag==true){

if(biBitCount!=24&&biBitCount!=8)

{

cout<<"图片不是24位或8位!"<<endl;

system("pause");

return;

}

else

{

int delta=0;

cout<<"请输入修改亮度的参数：";

cin>>delta;

if(biBitCount==24)

{

cout<<"图像是24位的"<<endl;

int lineByte = (bmpWidth \* biBitCount / 8 + 3) / 4 \* 4;

for (int i = 0; i < bmpHeight \* lineByte; i++)

{

if ((pBmpBuf[i]+delta)> 255)

pBmpBuf[i] = 255;

else if ((pBmpBuf[i]+delta) < 0)

pBmpBuf[i] = 0;

else

pBmpBuf[i]+=delta;

}

delete[]pBmpBuf;

}

if(biBitCount==8)

{

cout<<"图像是8位的"<<endl;

for(int i=0;i<=255;i++)

SetColor(i,pColorTable,delta);

}

}

writebmp(writePath,readFlag);

}

}

4.

图 4.4是运用乘法和除法对图像进行点运算，得到了图 4.5的三幅图像。

其中，第一幅图片的点运算是y=x/2，这样让灰度图像的每个像素点的灰度值变为原来的一般，整体图像亮度降低，变暗了。第二幅图像进行的运算是y=2x，这样放大了每个像素点的灰度值，因此整个图像看起来更亮了。而第三幅图像进行的运算是y=x/2+128的点运算，这样会让每个像素点的灰度值更加接近，对比度下降，因此这个图像看起来更加模糊一些。

和Figure 4.5的关系

5.

翻译

如果图像的类型是uint8，最好的方法是缺陷函数。图4.6显示了补码函数y=255-x以及命令的结果。

图4.6的含义：

就是利用补码函数y=255-x，对每个像素的灰度值进行点运算，得到每个像素的反色，然后汇总起来就得到了整个图像的反色。

6.

翻译：

对于灰度图像，其直方图由灰度级组成；也就是说，它是一个图表，指示图像中每个灰度级出现的次数。我们可以从直方图中推断出图像的外观，如下例所示。

在暗图像中，灰度级（以及直方图）将聚集在下端。

均匀明亮的图像，灰度级将聚集在上端旧值旧值仅采集暗像素仅补充亮像素

在对比度良好的图像中，灰度级将很好地分布在大部分范围内我们可以在MATLAB中使用imhist函数来查看图像的直方图，

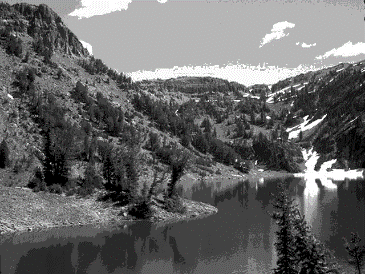
>>p=imread（pout.tif）；

>>imshow（p），figure，imhist（p），axis tight

（axis tight命令确保对轴进行缩放，以便所有直方图条都能完全拟合在图中）结果如图4.8所示。因为灰度值都聚集在直方图的中心，所以我们预计图像的对比度会很差。

对于对比度很差的图像，我们希望通过展开直方图来增强其对比度。有两种方法可以做到这一点。

部分反色结果：

原图像 灰度值较小的反色 灰度值较大的像素反色

其中，灰度值较小值灰度值在0~127的像素，灰度值较大指的是灰度值在128~255的像素点。

源代码：

注：本代码只展示了灰度值较小的反色代码，灰度值较大的反色代码同理，只需要修改if语句和循环语句的条件即可。

void SetColor2(int i,RGBQUAD\* &pColorTable)

{

pColorTable[i].rgbBlue=255-pColorTable[i].rgbBlue;

pColorTable[i].rgbGreen=255-pColorTable[i].rgbGreen;

pColorTable[i].rgbRed=255-pColorTable[i].rgbRed;

}

void invertColor(RGBQUAD\* &pColorTable,char \*writePath,bool readFlag)

{

if(readFlag==true){

if(biBitCount==8){

for(int i=0;i<=127;i++)

{

SetColor2(i,pColorTable);

}

}

if(biBitCount==24){

int lineByte=(bmpWidth\*biBitCount/8+3)/4\*4;

int fix=lineByte-bmpWidth\*3;

for(int i=0;i<bmpHeight;i++)

{

for(int j=0;j<bmpWidth;j++)

{

if(pBmpBuf[j\*3+2+bmpWidth\*i\*3+i\*fix]<128&&pBmpBuf[j\*3+1+bmpWidth\*i\*3+i\*fix]<128&&pBmpBuf[j\*3+bmpWidth\*i\*3+i\*fix]<128){

pBmpBuf[j\*3+2+bmpWidth\*i\*3+i\*fix]=255-pBmpBuf[j\*3+2+bmpWidth\*i\*3+i\*fix];

pBmpBuf[j\*3+1+bmpWidth\*i\*3+i\*fix]=255-pBmpBuf[j\*3+1+bmpWidth\*i\*3+i\*fix];

pBmpBuf[j\*3+bmpWidth\*i\*3+i\*fix]=255-pBmpBuf[j\*3+bmpWidth\*i\*3+i\*fix];

}

}

}

}

writebmp(writePath,readFlag);

}

else

cout<<"ÇëÏÈ¶ÁÎÄ¼þ£¡"<<endl;

}