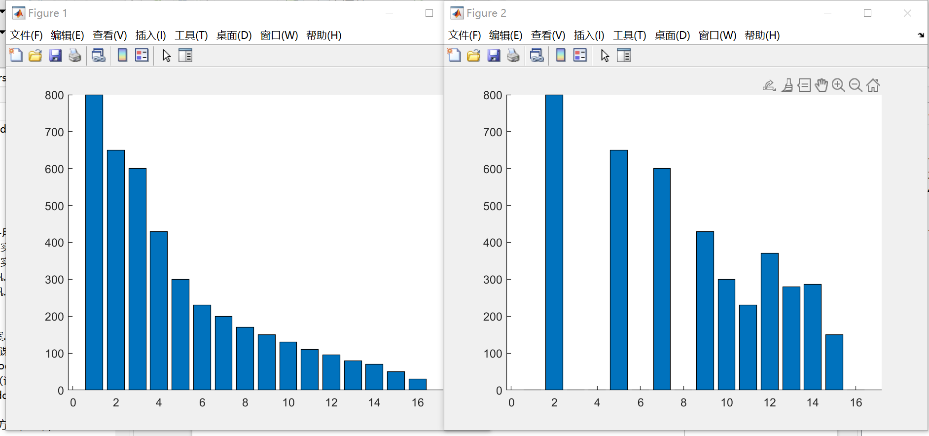
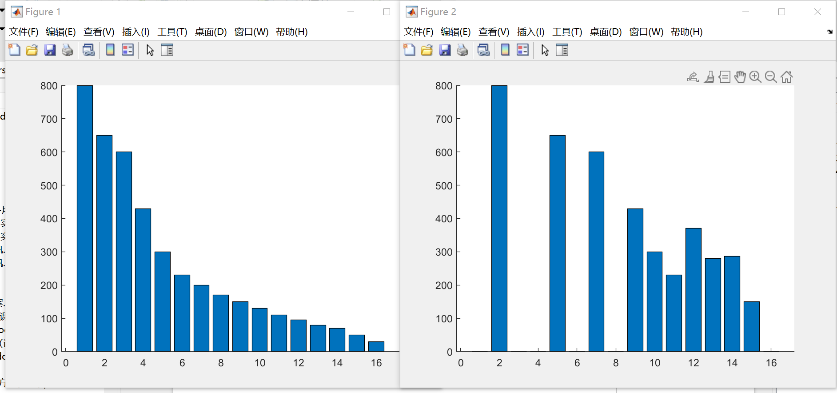
作业5

1. 假定有64\*64大小的图像,灰度为16级,概率分布如图,进行直方图的均衡化,并画出处理前后的直方图。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| r | nk | Pr(rk) | r | nk | Pr(rk) |
| r0 | 800 | 0.195 | r8 | 150 | 0.037 |
| r1 | 650 | 0.160 | r9 | 130 | 0.031 |
| r2 | 600 | 0.147 | r10 | 110 | 0.027 |
| r3 | 430 | 0.106 | r11 | 96 | 0.013 |
| r4 | 300 | 0.073 | r12 | 80 | 0.019 |
| r5 | 230 | 0.056 | r13 | 70 | 0.017 |
| r6 | 200 | 0.049 | r14 | 50 | 0.012 |
| r7 | 170 | 0.041 | r15 | 30 | 0.007 |

处理之前 处理之后

1. 对图做３＊３的中值滤波，写出处理结果并比较．（边界像素处理时，对原图像进行用0填充，结果图像的大小不变）

１　７　１　８　１　７　１　１

１　１　１　５　１　１　１　１

１　１　５　５　５　１　１　７

１　１　５　５　５　１　８　１

８　１　１　５　１　１　１　１

８　１　１　５　１　１　８　１

１　１　１　５　１　１　１　１

１　７　１　８　１　７　１　１

实验结果:

0 1 1 1 1 1 1 0

1 1 5 5 5 1 1 1

1 1 5 5 5 1 1 1

1 5 5 5 5 1 1 1

1 1 1 5 1 1 1 1

1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1

比较分析：

去除椒盐噪声：中值滤波在图像中存在椒盐噪声（即黑白像素点）时表现出色。它通过将像素值替换为邻域内像素值的中值，有效地去除离群点和噪声。

保留边缘信息：相比于一些线性滤波器（如均值滤波器），中值滤波器能够在去除噪声的同时较好地保留图像的边缘信息。它不会模糊边缘，而是通过选择邻域内的中值来平滑非边缘区域，从而实现噪声去除和边缘保持的平衡。

同时，中值滤波也有一些限制，例如可能会导致细节丢失或模糊化，对于连续的噪声（如高斯噪声）效果可能不如线性滤波器。

1. 使用Sobel算子对第２题进行计算,给出结果并进行分析.

实验结果：

8 0 6 0 0 0 0 0

8 4 8 0 0 0 0 0

4 8 8 0 0 3 8 0

4 5 8 0 0 2 6 0

4 0 8 0 0 8 0 0

4 0 8 0 0 8 0 0

8 0 8 0 0 7 0 0

8 0 6 0 0 0 0 0

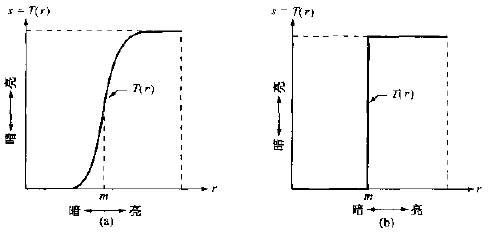
比较分析：

方向敏感：Sobel算子是方向敏感的，可以分别检测水平和垂直方向的边缘。这使得它在确定边缘的方向和斜率方面非常有用。

平滑效果：Sobel算子在边缘检测的同时也具有平滑效果。它通过对图像进行滤波来平滑噪声，并在边缘处保留边缘信息。这种平滑效果可以减少图像中的噪声和细节，并使边缘更加清晰。

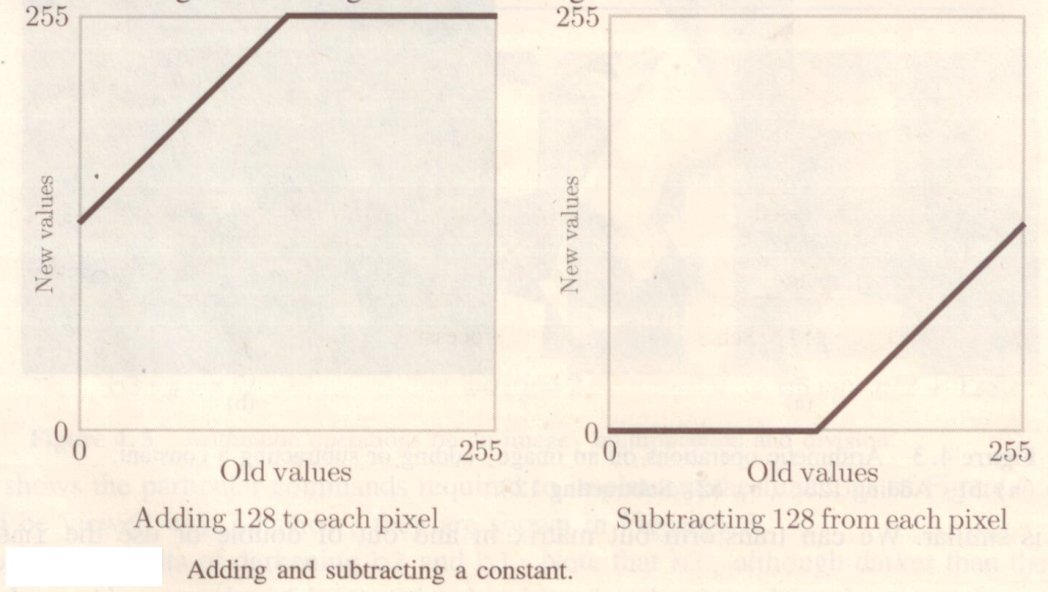
Sobel算子也存在一些限制。例如，它对噪声比较敏感，可能会将噪声错误地检测为边缘。此外，Sobel算子对边缘的宽度和方向有一定的限制，可能无法准确检测某些特定类型的边缘。

* 4.(1)下图a,b分别是灰度变换函数的图形.说明变换的结果.m是某个灰度值.



a图像在处理之后在暗的地方会变得更暗，亮的地方会变得更亮，对比度增强。而b图中是实现了灰度阈值变换，灰度值小于m的像素变成黑色，灰度值大于m的像素变成白色，实现了图像二值化处理。因为是灰度变换函数的图形，因此衡量暗亮的值只能是灰度值，因此m是某个灰度值。

* 4.(2)用MATLAB或者C++对图像进行灰度变换，变换函数如下图，说明图像变换前后的效果。

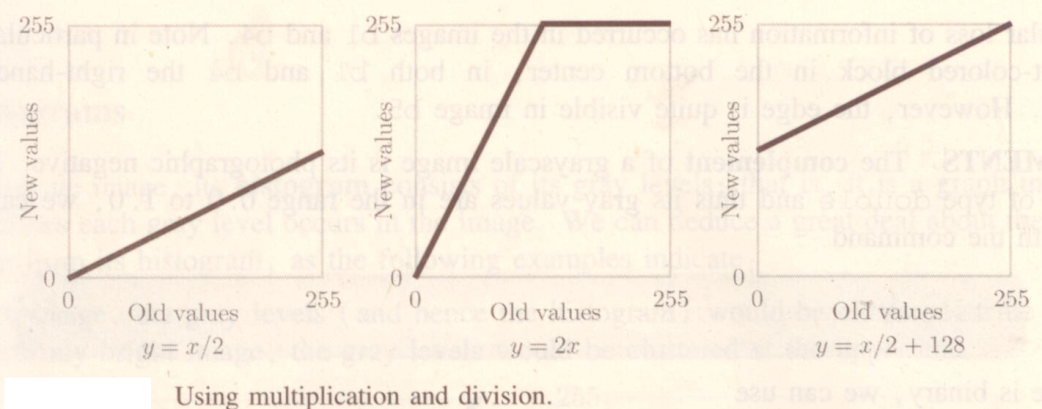


变换图像的效果：

第一个图像是给每个像素的灰度值加128，若灰度值加到128之后超过了255，则将灰度值设置为255，整体图像是变得更亮了。

第二个变换函数是给每个像素的灰度值减128，若灰度值减128之后小于0，则将灰度值设置为0，整体图像变得更暗了。

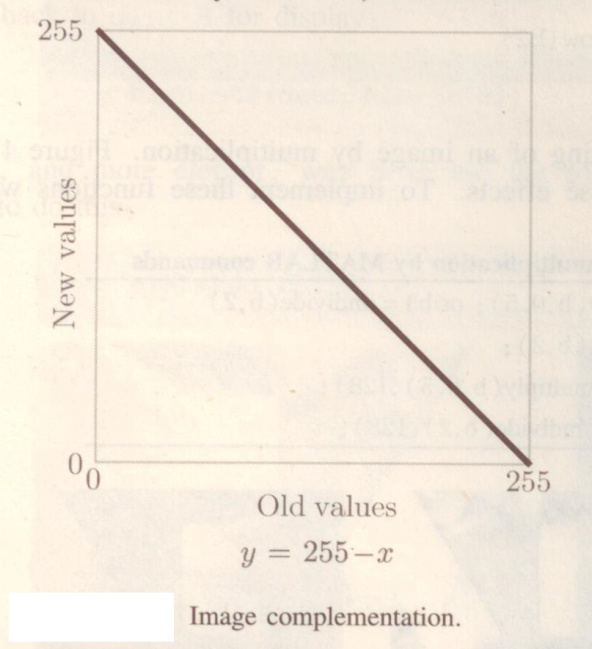
* 4.(3)用MATLAB或者C++对图像进行灰度变换，变换函数如下图，说明图像变换前后的效果。



图像a：y=x/2是对图像的每个像素点的灰度值都取原来的一半，这意味着图像中的每个像素都会变得更暗，整体亮度减少。这种线性变换通常会导致图像的对比度减小，细节变得模糊。

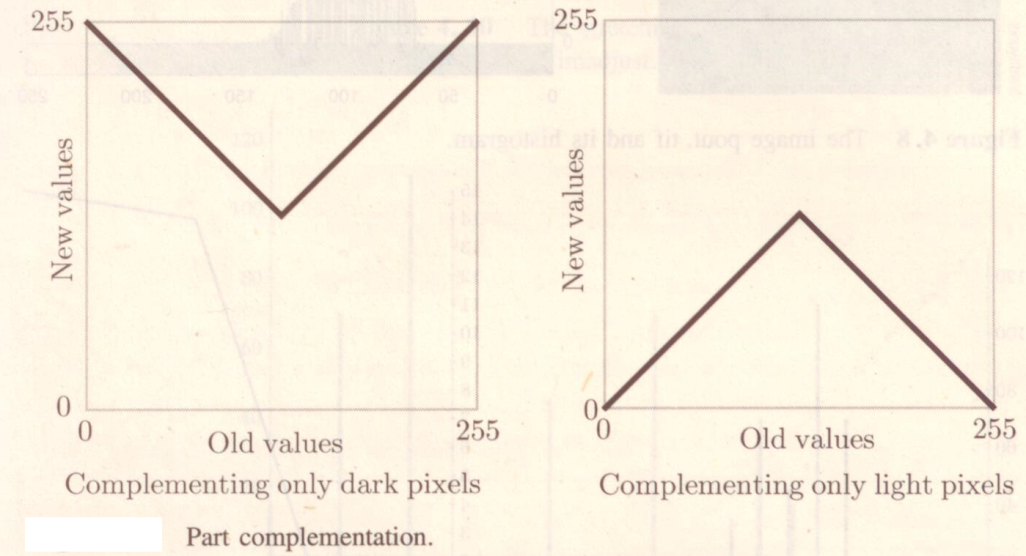
图像b：y=2x是对图像的每个像素点的灰度值都乘以2，若灰度值超过了255则取灰度值为255。这意味着图像中的每个像素都会变得更亮，整体亮度增加。这种线性变换通常会导致图像的对比度增加，细节更加突出。

图像c：y=x/2+128是对每个像素点的灰度值除以2，然后再加上128，相当于将图像的亮度减小一半并向高亮度方向平移。这种线性变换通常会使图像整体的亮度增加，并且改变图像的整体对比度。

*  4.(4)用MATLAB或者C++对图像进行灰度变换，变换函数如下图，说明图像变换前后的效果。

该线性变换实现了图像的反色，每个像素的灰度值是用255减去x得到的。

* 4.(5)用MATLAB或者C++对图像进行灰度变换，变换函数如下图，说明图像变换前后的效果。

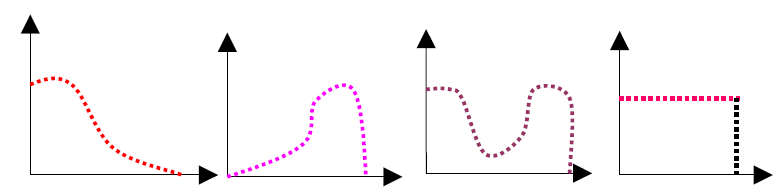


图a中，实现v字变换，在图像亮度小于128的部分进行反色，大于128的部分不变。增强图像的局部对比度，使细节更加突出。

图b中，实现了倒v字变换，在图像亮度大于128的部分进行反色，小于128的部分不变。倒V型变换可以减弱图像的对比度，使图像整体变得更加平滑。



* （1）根据直方图的形状说明图像的灰度情况。

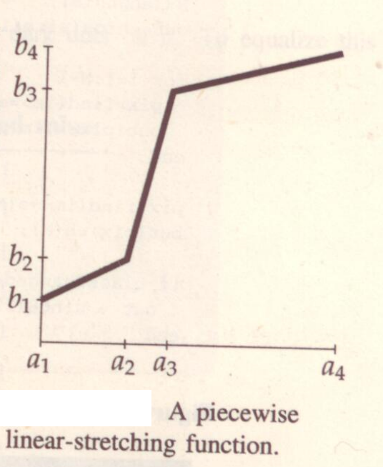


图a中，整体图像暗的地方多，整体图像都比较暗，图像的对比度较差。

图b中，亮度地方较少，暗的地方较多，整体图像亮的地方多，偏亮。直方图的峰值较为集中且较低,因此对比度比较差。

图c中，直方图具有两个明显的峰值，即直方图的分布呈现双峰形状，那么图像可能具有双重灰度模式。这可能表示图像包含不同的对象或场景，每个对象或场景具有不同的灰度范围。最暗和最亮的地方部分都比较多，整体上看对比度大，亮度中间部分的比较少。

图d中，灰度分布均匀，画面整体效果较好。这意味着图像中的各个灰度级都有相近的数量的像素值。

* （2）写出以下分段线性变换的函数。

a1~a2: y=(b2-b1)/(a2-a1)\*(x-a1) + b1

a2~a3: y=(b3-b2)/(a3-a2)\*(x-a2)+ b2

a3~a4: y= (b4 – b3) / (a4 – a3)\*(x-a3)+ b3

1. 自己制作一个8\*8的8个灰度级的图像.

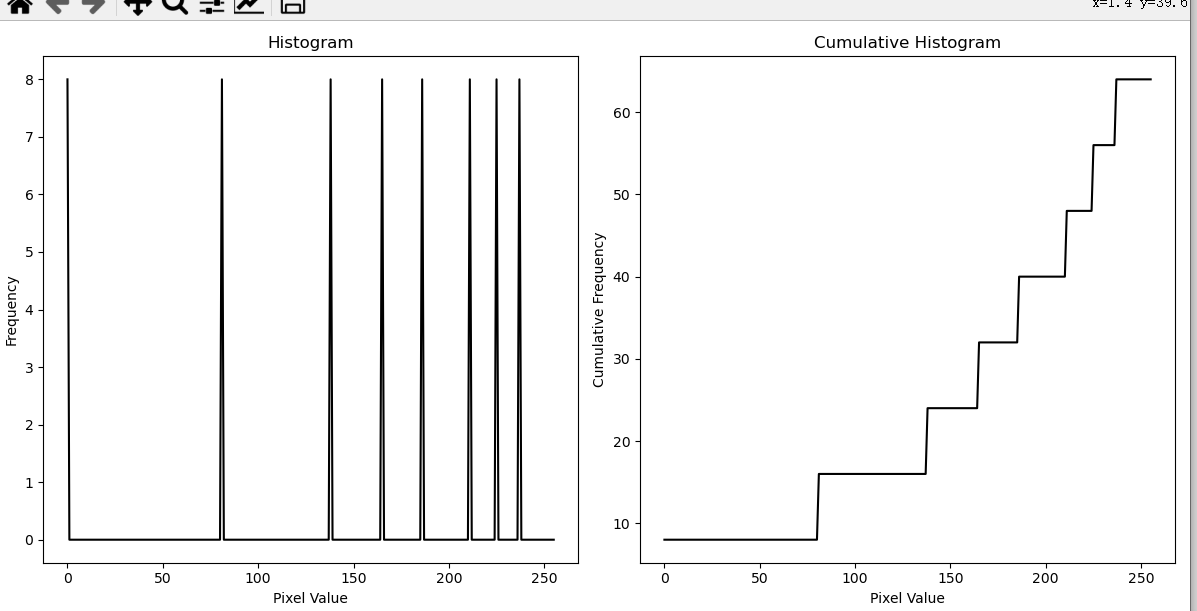
1)画出直方图和累积直方图;

2)对图像进行直方图均衡化处理,制作一个表格写出处理步骤;

3)画出图像被直方图均衡化操作处理后的直方图.



原图像

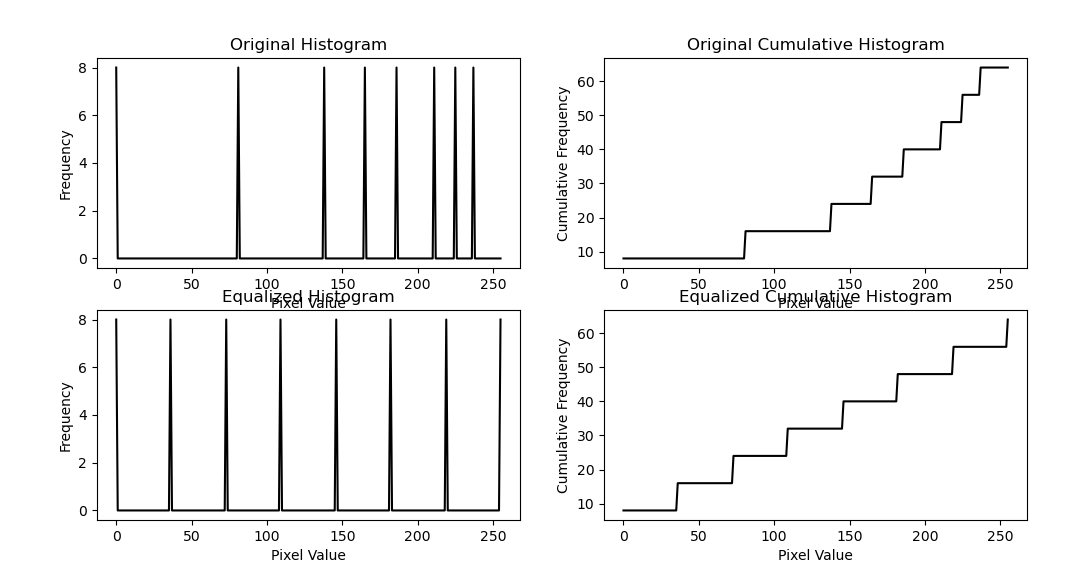


直方图和累计直方图

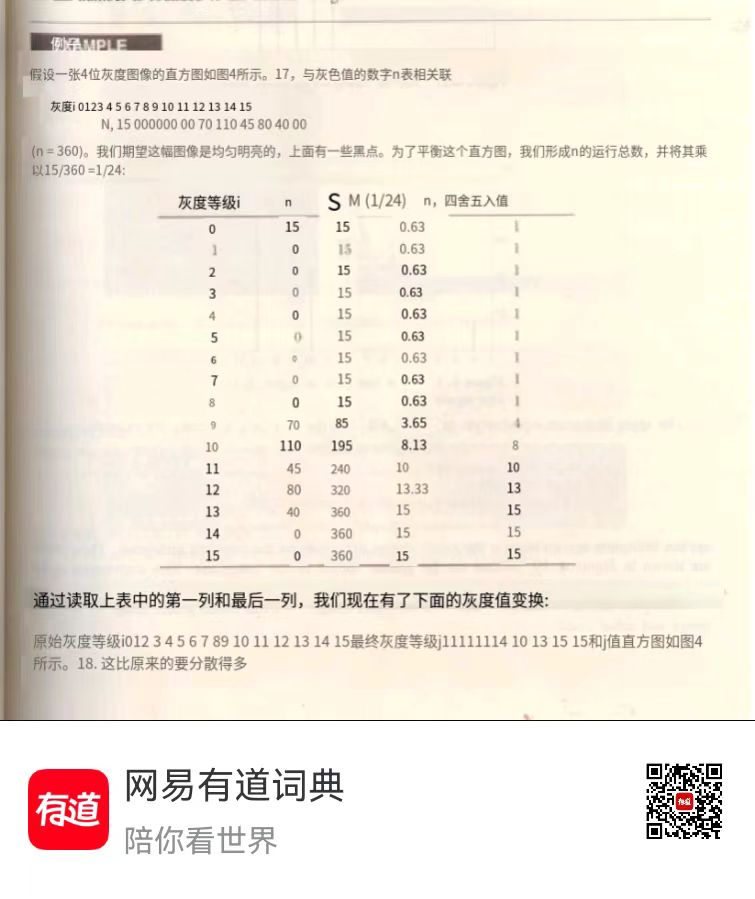
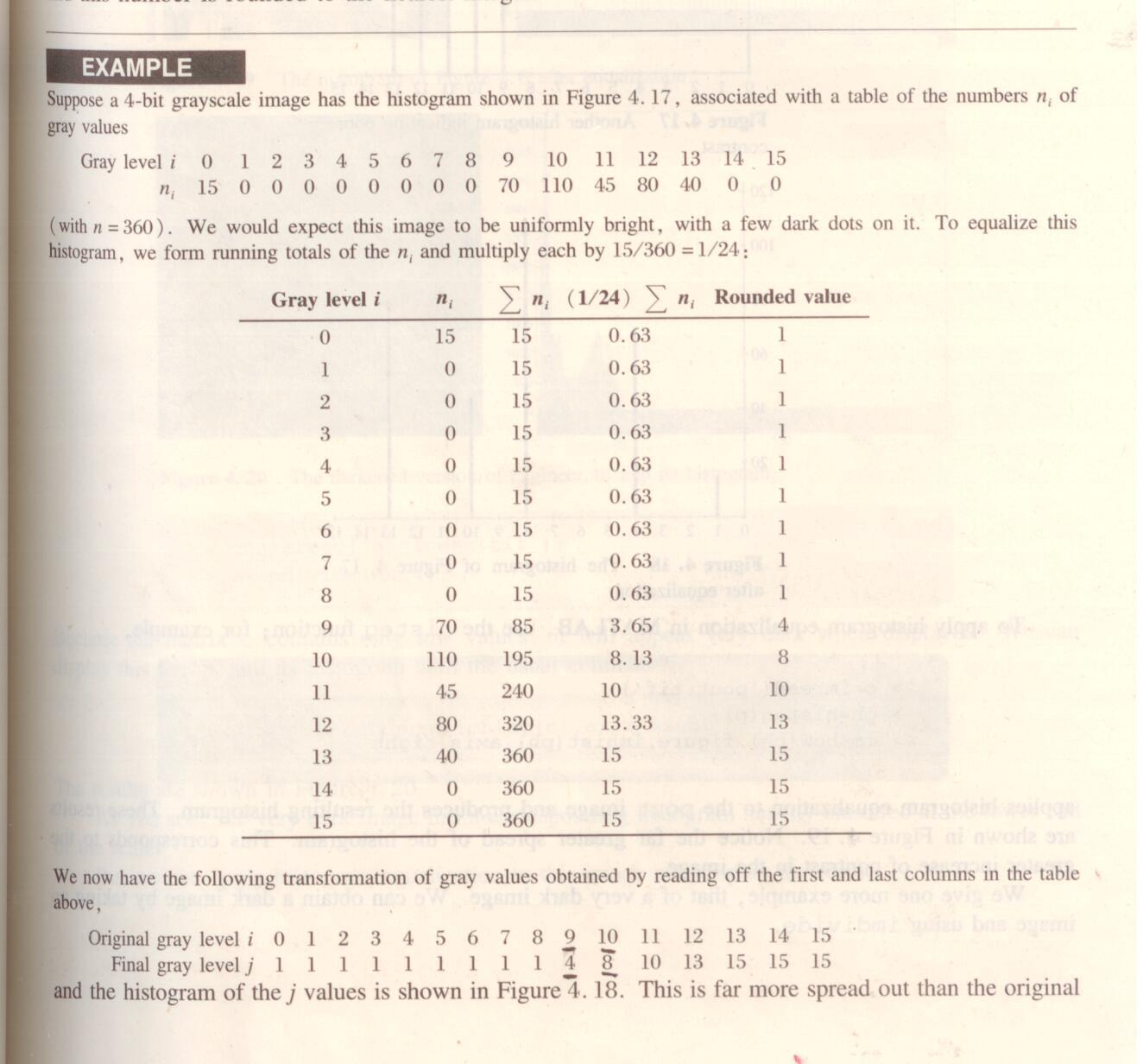
直方图均衡化步骤：

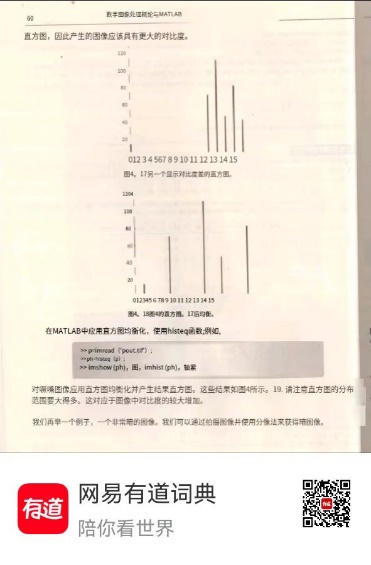
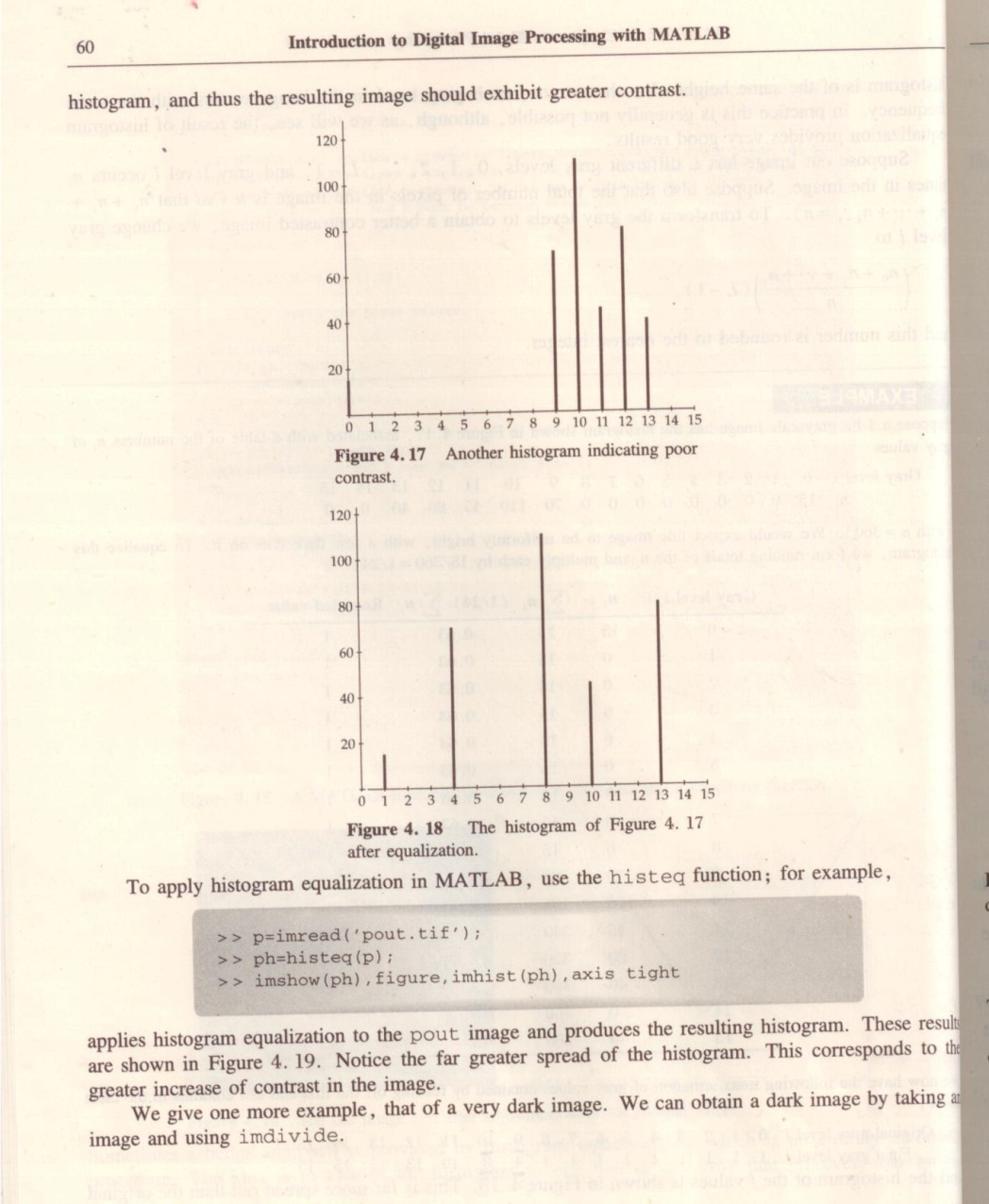
|  |
| --- |
| 确定图像的灰度级 |
| 计算原始直方图分布概率p(i) |
| 计算直方图概率累计值s(i) |
| 根据公式求取像素映射关系 |
| 灰度映射 |

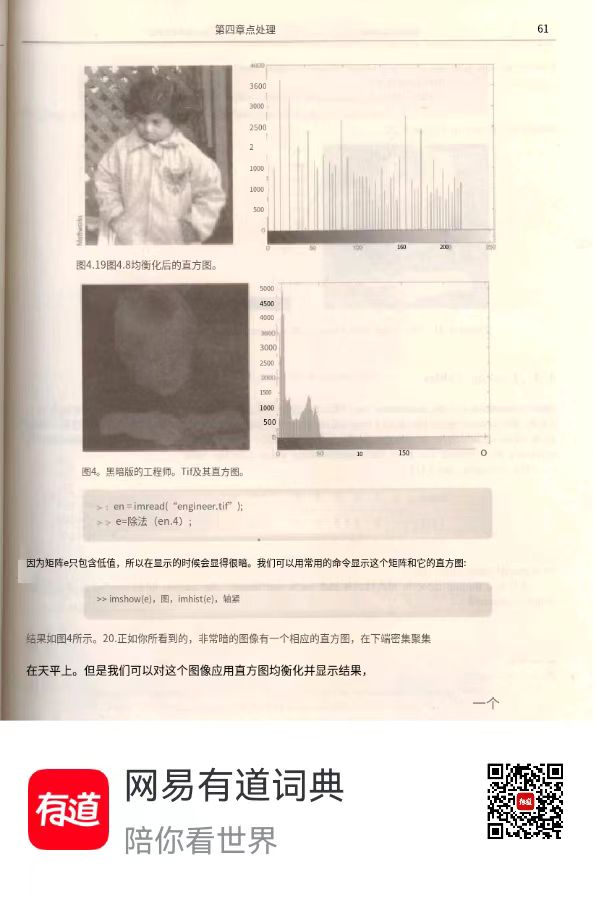
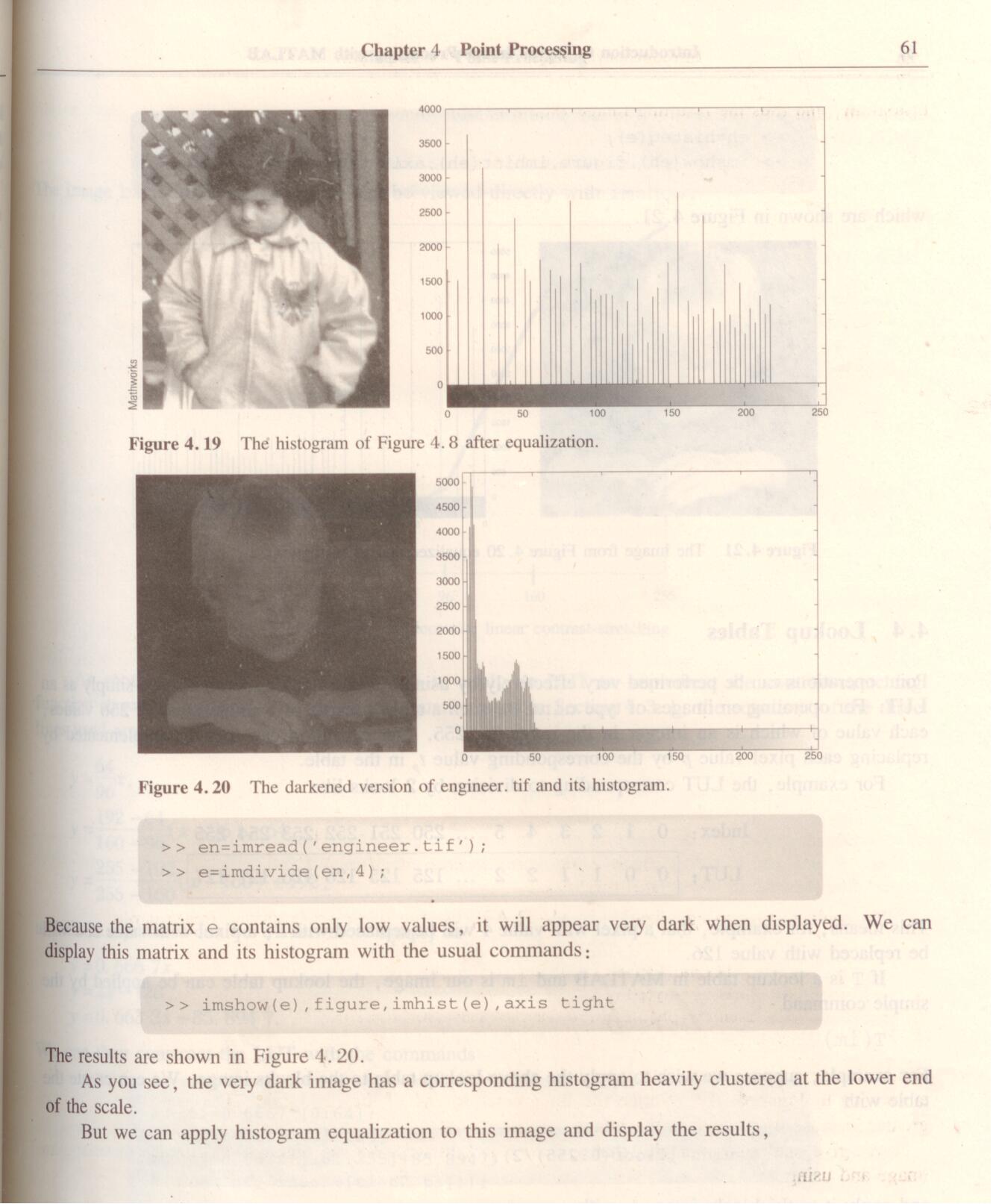
被均衡化之后的直方图：

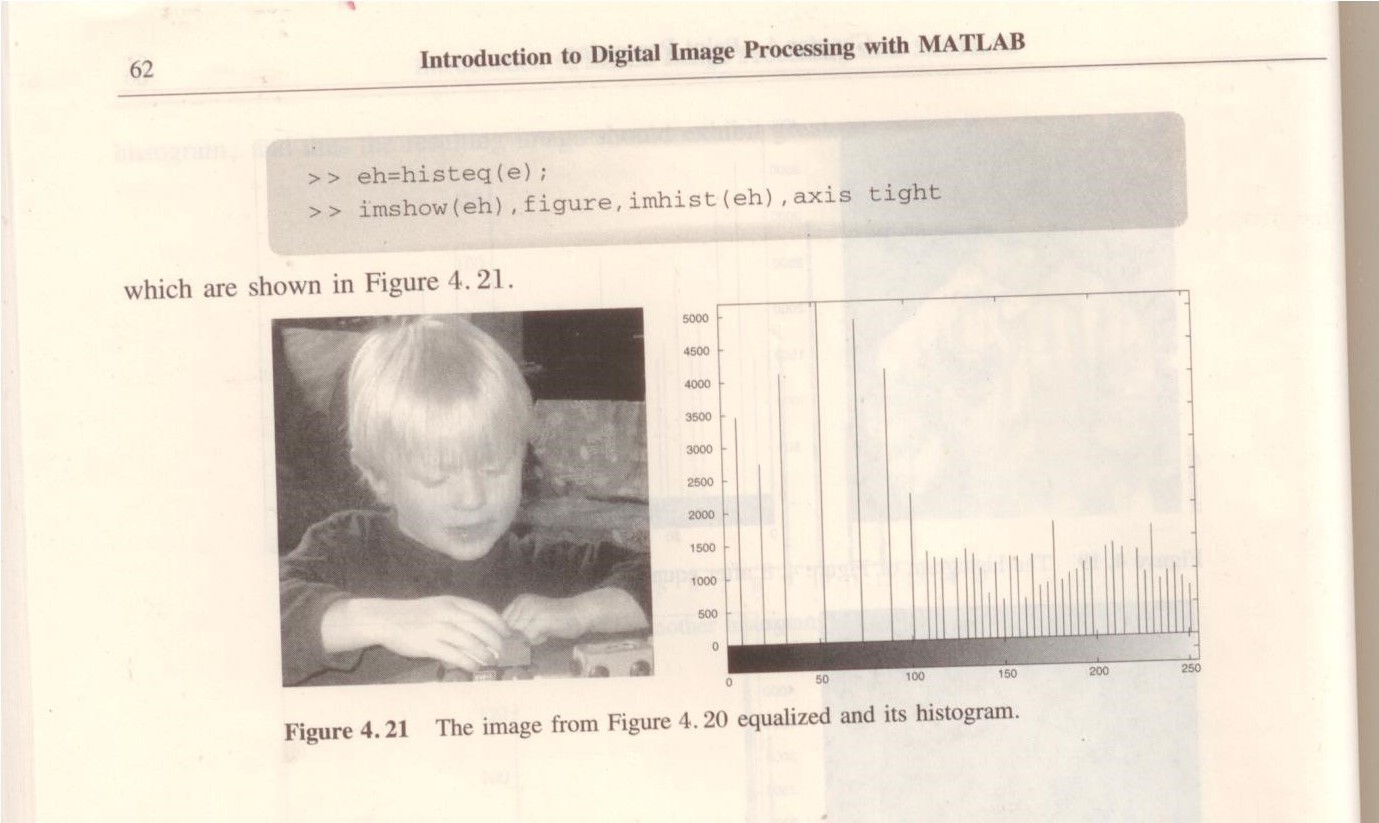


1. 小论文见《小论文》
2. 作业5\_8.翻译以下内容



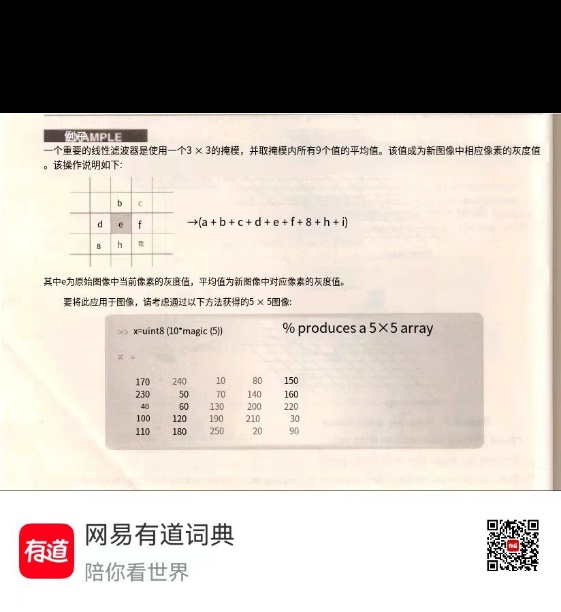
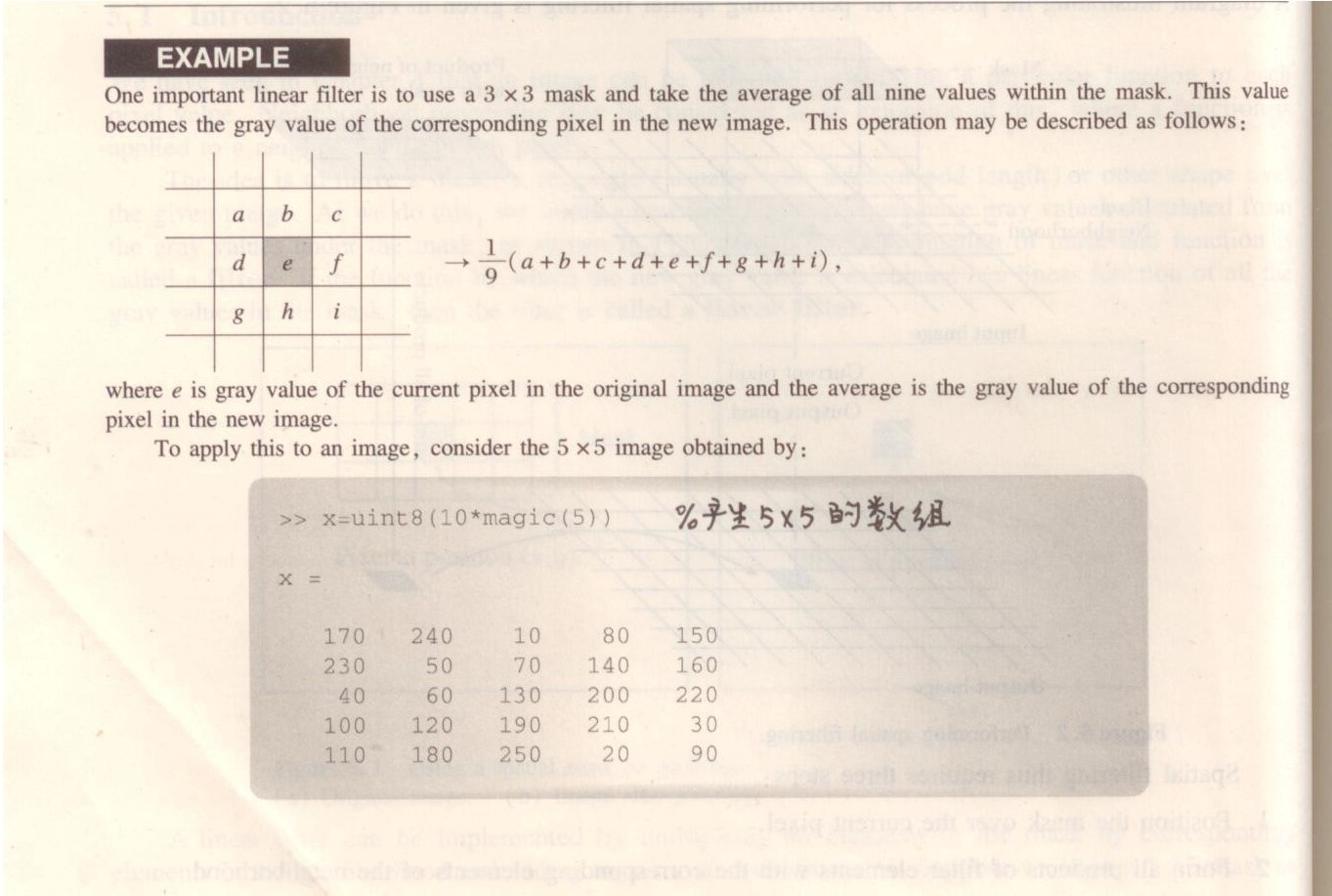


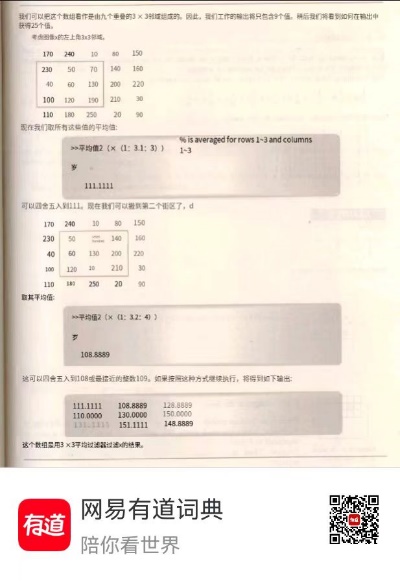
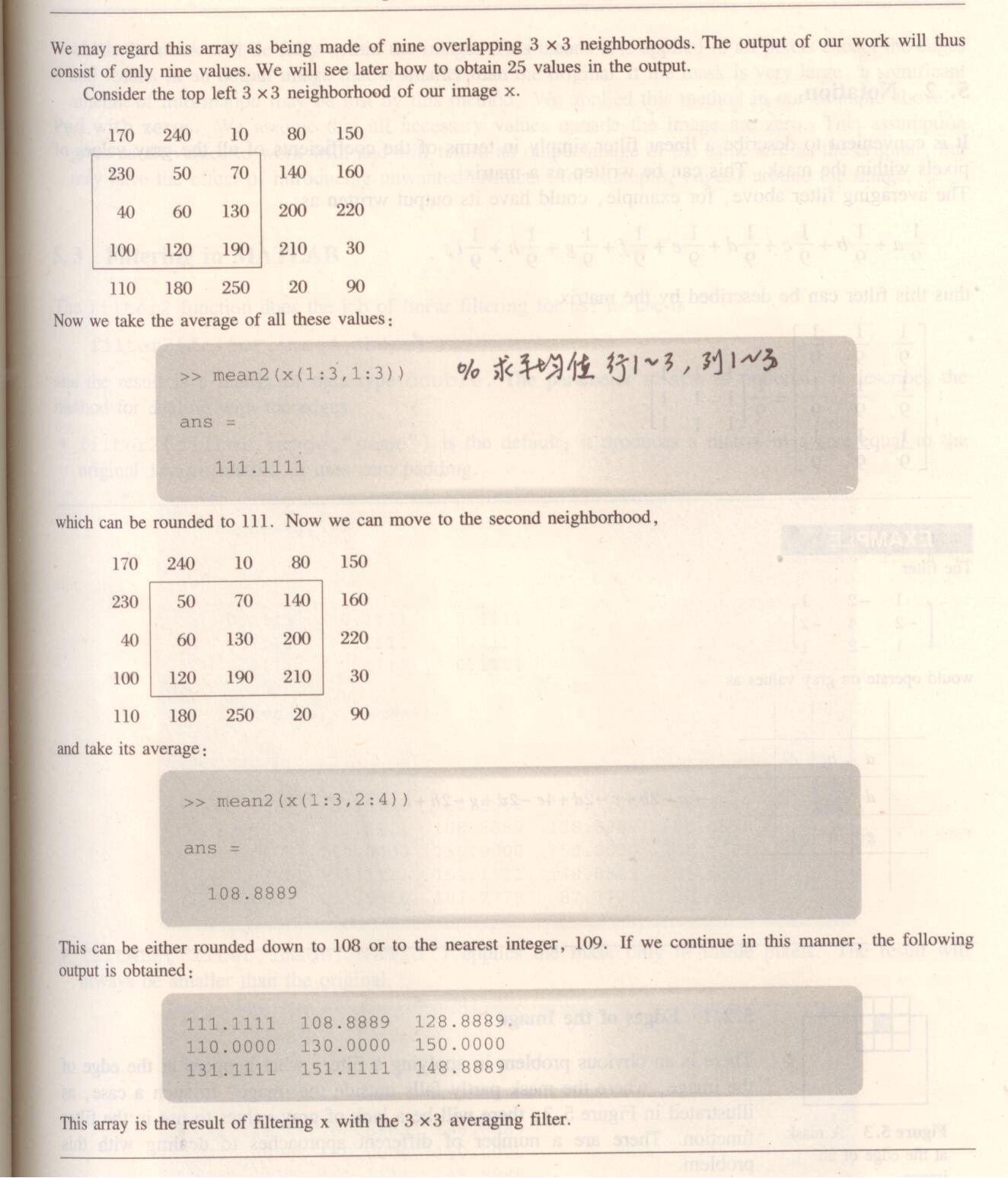




这部分内容讲的是直方图均衡化的方法，以及实现的效果具体方法和在第6部分的方法相同。

1. 作业5\_9翻译以下内容：





这部分内容讲的是滤波的过程，运用个3x3的掩模取这9个像素的平均值得到新图像对应像素点的灰度值，然后讲5x5的掩模该如何操作？该掩模就是由九个重叠的3 x 3邻域组成的。因此可以推广使用。

1. 用c++实现使用直方图统计进行图像增强：

#include <iostream>

#include <opencv2/opencv.hpp>

using namespace std;

using namespace cv;

int main()

{

// 读取图像

Mat image = imread("input\_image.jpg", IMREAD\_GRAYSCALE);

if (image.empty())

{

cout << "无法加载图像！" << endl;

return -1;

}

// 计算全局平均灰度和标准差

Scalar meanVal, stdDev;

meanStdDev(image, meanVal, stdDev);

double globalMean = meanVal[0];

double globalStdDev = stdDev[0];

// 参数设置

double k0 = 0.8; // 暗区域判断阈值因子

double k1 = 0.5; // 对比度限制因子

double k2 = 1.5; // 对比度判断阈值因子

// 图像增强

Mat enhancedImage = image.clone();

for (int y = 0; y < image.rows; y++)

{

for (int x = 0; x < image.cols; x++)

{

// 计算局部平均灰度和标准差

Mat neighborhood;

getRectSubPix(image, Size(3, 3), Point2f(x, y), neighborhood);

meanStdDev(neighborhood, meanVal, stdDev);

double localMean = meanVal[0];

double localStdDev = stdDev[0];

// 判断是否满足增强条件

if (localMean <= k0 \* globalMean && localStdDev <= k2 \* globalStdDev && k1 \* globalStdDev <= localStdDev)

{

// 增强像素值

enhancedImage.at<uchar>(y, x) = saturate\_cast<uchar>(image.at<uchar>(y, x) \* E);

}

}

}

// 显示原始图像和增强后的图像

namedWindow("原始图像", WINDOW\_NORMAL);

namedWindow("增强后的图像", WINDOW\_NORMAL);

imshow("原始图像", image);

imshow("增强后的图像", enhancedImage);

waitKey(0);

return 0;

}

请注意，上述代码中的参数k0、k1、k2和E需要根据具体的需求进行调整和优化。这只是一个基本的示例，可以根据需要进行修改和扩展。

在增强阶段，代码遍历图像的每个像素，计算局部平均灰度和标准差，并根据条件判断是否满足增强的要求。对于满足条件的像素，将其像素值乘以常数E来增强。不满足条件的像素保持不变。

* Unsharp Mask (USM) 锐化滤镜。

#include <iostream>

#include <opencv2/opencv.hpp>

using namespace std;

using namespace cv;

int main()

{

// 读取图像

Mat image = imread("input\_image.jpg");

if (image.empty())

{

cout << "无法加载图像！" << endl;

return -1;

}

// 转换为灰度图像

Mat grayImage;

cvtColor(image, grayImage, COLOR\_BGR2GRAY);

// 高斯模糊

Mat blurredImage;

GaussianBlur(grayImage, blurredImage, Size(0, 0), 1.5);

// 计算锐化图像

Mat sharpenedImage = grayImage - blurredImage;

sharpenedImage = grayImage + sharpenedImage;

// 对比度增强

double alpha = 1.5; // 增强系数

sharpenedImage.convertTo(sharpenedImage, -1, alpha, 0);

// 显示原始图像和锐化后的图像

namedWindow("原始图像", WINDOW\_NORMAL);

namedWindow("锐化后的图像", WINDOW\_NORMAL);

imshow("原始图像", image);

imshow("锐化后的图像", sharpenedImage);

waitKey(0);

return 0;

}

请确保将程序中的"input\_image.jpg"替换为你要进行锐化的图像的路径。此示例使用OpenCV库来处理图像，因此你需要安装OpenCV并将其与你的编译器配置连接。

在该份代码中，首先将彩色图像转换为灰度图像。然后，应用高斯模糊以平滑图像。接下来，通过从原始图像中减去模糊图像并添加到灰度图像上，得到一个锐化图像。最后，使用对比度增强来提高锐化图像的对比度。

这部