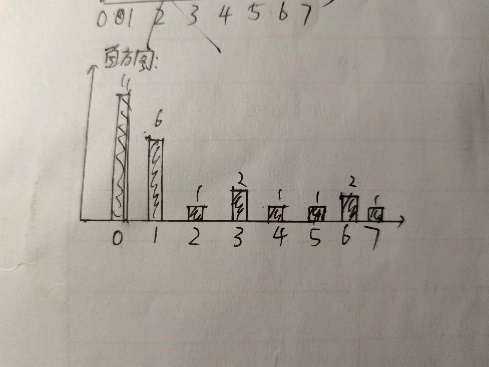
1-



绝对值的直方图 归一化后的直方图

归一化算法：

使用如下转换函数：

x[i] ＝ (x[i] - min)/(max - min)

max: 样本数据的最大值

min: 为样本数据的最小值

结论：

直观印象：灰度值为0的像素点最多，而灰度值较高（灰度值大于3）的像素点较少，因此整体图像亮度偏暗。

1.灰度直方图只能反映图像的灰度分布情况，而不能反映图像像素的位置。

2.一幅图像对应惟一的灰度直方图，反之不成立。不同的图像可对应相同的灰度直方图。

3.一幅图像分成多个区域，多个区域的直方图之和即为原图像的直方图。

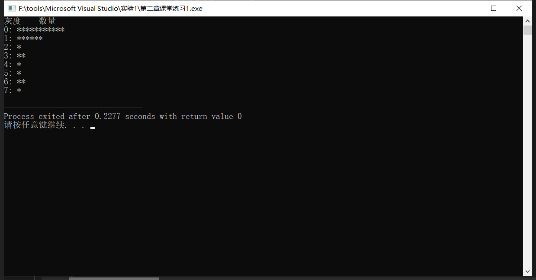
思考：

首先，需要定义每一个灰度级之间的映射关系。然后，需要将每一个像素点的灰度值按照定义的映射关系重新计算，得到新的灰度值，最终得到一张256灰度级的图片。

例如：可以使用灰度级拉伸法，将8灰度级的图像拉伸到256灰度级。具体算法是：

1. 遍历原图片的每一个像素点，获取其灰度级值。
2. 将灰度级值乘以255/7，即255除以最大的灰度级值7，得到新的灰度级值。
3. 将新的灰度级值赋给原图片的每一个像素点，完成拉伸。经过上述操作，便将8灰度级的图片变成256灰度级的图片。

课后作业：

运行截图：

源代码：

#include <iostream>

#include <iomanip>

//#include <opencv2\opencv.hpp>

using namespace std;

int main(){

int a[5][5]={{0, 3 ,6,1,1},{1,4,5,0,0},{1,0,0,7,0},{1,0,2,0,3},{6,0,0,0,1}};

float b[8]={0.0};

int z=0;

string s[8];

for(int i=0;i<5;i++){

for(int j=0;j<5;j++){

z=a[i][j];

b[z]++;

}

}

cout<<"灰度 数量"<<endl;

for(int i=0;i<8;i++){

for(int j=0;j<b[i];j++){

s[i]+='\*';

}

cout<<i<<": "<<s[i]<<endl;

}

return 0;

}