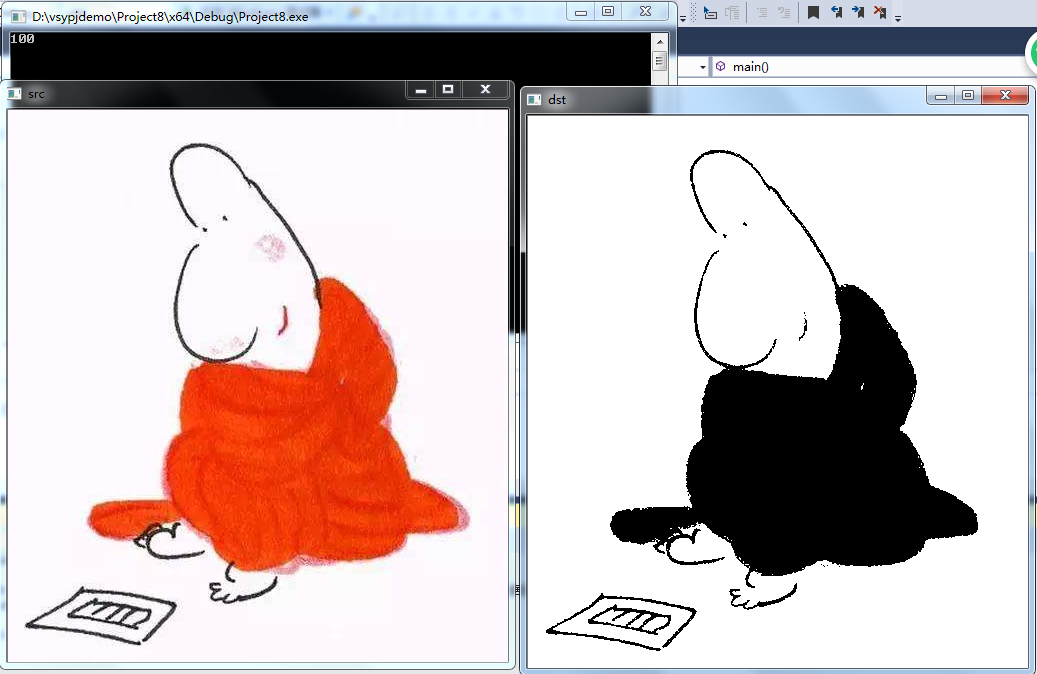
# 代码功能是？

答：



#include <iostream>

#include <opencv2\opencv.hpp>

using namespace std;

using namespace cv;

int main()

{

Mat src=imread("e:\\dao.jpg",1);

if (!src.data) { cout << "error" << endl; return -1; }

imshow("src",src);

int thresh;

//cout<<"请输入阈值"<<endl;

cin>>thresh;

Mat srcgray ;

cvtColor(src,srcgray,COLOR\_RGB2GRAY);

Mat dst(srcgray.rows,srcgray.cols,CV\_8UC1);

for (int i = 0; i < srcgray.rows; i++)

{

for (int j = 0; j < srcgray.cols; j++)

{

if (srcgray.at<uchar>(i, j) > thresh) dst.at<uchar>(i, j) = 255;

else dst.at<uchar>(i, j) = 0;

}

}

imshow("dst", dst);

waitKey(0);

return 0;

}

# 2、学生编写代码

参考链接：

<http://www.opencv.org.cn/opencvdoc/2.3.2/html/doc/tutorials/imgproc/threshold/threshold.html>

使用threshold函数中第五个参数type表示阈值化处理的类型

THRESH\_BINARY 0

THRESH\_BINARY\_INV 1

THRESH\_TRUNC 2

THRESH\_TOZERO 3

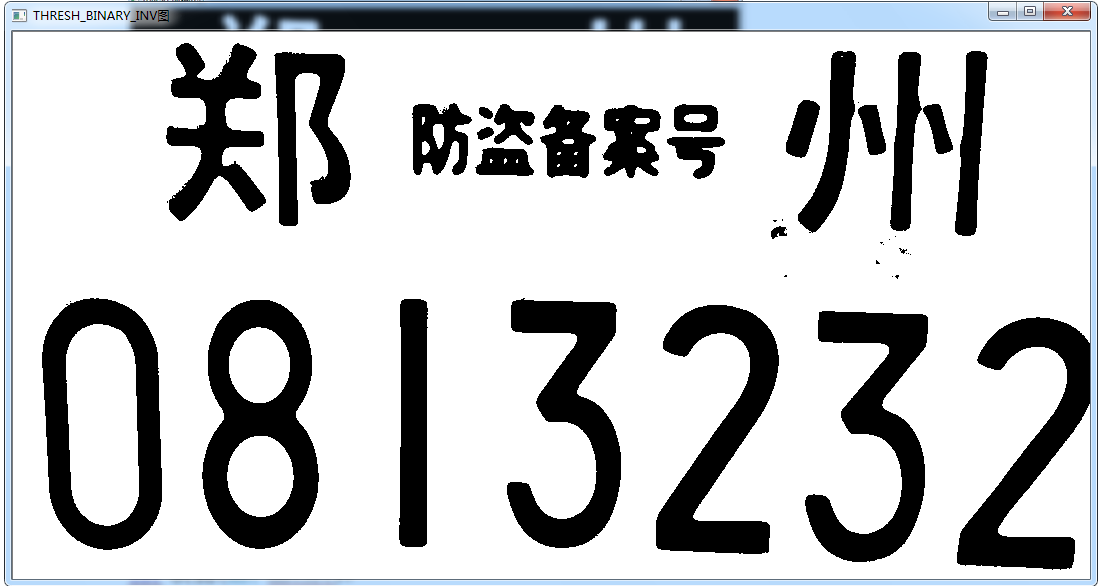
THRESH\_TOZERO\_INV 4

# 2.1编写代码，分别使用上述

## 五个类型，展示5幅阈值化后的图像。











## 2.2滑块 threshold练习

滑块：阈值类型 0-4

滑块： 阈值 0-255



# OSTU方法的实现

int Otsu(Mat matSrc)

{

if (CV\_8UC1 != matSrc.type())

return -1;

int nCols = matSrc.cols;

int nRows = matSrc.rows;

int nPixelNum = nCols \* nRows;

// 初始化

int pixelNum[256];

double probability[256];

for (int i = 0; i < 256; i++)

{

pixelNum[i] = 0;

probability[i] = 0.0;

}

// 统计像素数和频率

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

{

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

{

pixelNum[matSrc.at<uchar>(j, i)]++;

}

}

for (int i = 0; i < 256; i++)

{

probability[i] = (double)0.1\*pixelNum[i] / nPixelNum;

}

// 计算

int nThreshold = 0; // 最佳阈值

double dMaxDelta = 0.0; // 最大类间方差

double dMean\_0 = 0.0; // 左边平均值

double dMean\_1 = 0.0; // 右边平均值

double dDelta = 0.0; // 类间方差

double dMean\_0\_temp = 0.0; // 左边平均值中间值

double dMean\_1\_temp = 0.0; // 右边平均值中间值

double dProbability\_0 = 0.0; // 左边频率值

double dProbability\_1 = 0.0; // 右边频率值

for (int j = 0; j < 256; j++)

{

for (int i = 0; i < 256; i++)

{

if (i < j)// 前半部分

{

dProbability\_0 += probability[i];

dMean\_0\_temp += i \* probability[i];

}

else // 后半部分

{

dProbability\_1 += probability[i];

dMean\_1\_temp += i \* probability[i];

}

}

// 计算平均值

// fMean\_0\_teamp计算的是前半部分的灰度值的总和除以总像素数，

// 所以要除以前半部分的频率才是前半部分的平均值，后半部分同样

dMean\_0 = dMean\_0\_temp / dProbability\_0;

dMean\_1 = dMean\_1\_temp / dProbability\_1;

dDelta = (double)(dProbability\_0 \* dProbability\_1 \* pow((dMean\_0 - dMean\_1), 2));

if (dDelta > dMaxDelta)

{

dMaxDelta = dDelta;

nThreshold = j;

}

// 相关参数归零

dProbability\_0 = 0.0;

dProbability\_1 = 0.0;

dMean\_0\_temp = 0.0;

dMean\_1\_temp = 0.0;

dMean\_0 = 0.0;

dMean\_1 = 0.0;

dDelta = 0.0;

}

return nThreshold;

}

## 3.1 学习OSTU算法的理论内容

OSTU最大类间方差 法 (有时也称之为大津法)

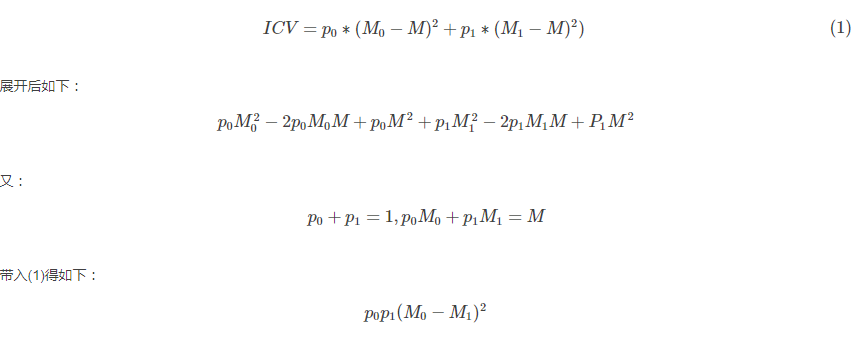
在阈值化处理中，常用的算法就是OTSU。OTSU算法是由日本学者OTSU于1979年提出的一种对图像进行二值化的高效算法。计算简单，稳定有效，一直广为使用。从模式识别的角度看，最佳阈值应该当具有最佳的目标类与背景类的分离性能，此性能用类比方差来表征，为此引入类内方差，类间方差和总体方差。

基本思路是将直方图在某一阈值处分成两组，当被分成的两组的方差最大时，得到阈值。

这种二值化操作阈值的选取非常重要，阈值选取的不合适，可能得到的结果就毫无用处。简单的说，这种算法假设衣服图像由前景色和背景色组成。通过统计学的方法来选取一个阈值，使这个阈值可以将前景色和背景色尽可能分开。

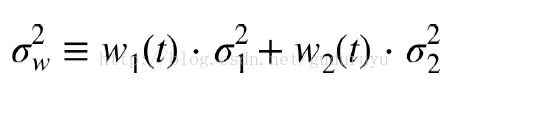
该方法的基本思想是：设阈值将图像分割成两组，一组灰度对应目标，另一组灰度对应背景，则这两组灰度值的类内方差最小，两组的类间方差最大

我们知道一幅灰度图像，可以计算它的颜色平均值，或者更进一步，可以计算出灰度直方图。我们可以把这幅图的灰度平均值为M，任意选取一个灰度值t，则可以将这个直方图分成前后两部分。我们称这两部分分别为A和B，对应的就是前景色和背景色。这两部分各自的平均值为M0 M0和M1 M1。A部分里的像素站总像素数的比例为p0 p0，B部分里的像素数站总像素数的比例为p1p1。Nobuyuki Ostu给出的类间方差定义为：



方差是灰度分布均匀性的一种度量，方差值越大说明构成图像的两部分差别越大，当部分目标点错分为背景或部分背景点错分为目标点都会导致两部分差别变小，因此使类间方差最大的分割意味着错分概率最小。

在opencv3中cv::threshold()函数还支持一种特殊的阈值化操作方式，即Otsu算法。该算法的主要思想是，在进行阈值化时，考虑所有可能的阈值，分别计算低于阈值和高于阈值像素的方差，使下式最小化的值作为阈值：



其中，两类像素方差的权值由两类像素的个数决定。这种阈值化的结果相对来说比较理想，可以避免寻找合适阈值的操作，但是这种方式运算量较大，费时。处理的结果如下：



当方差最大时，可以认为此时前景和背景差异最大，此时的灰度T是最佳阈值。类间方差法对噪声以及目标大小十分敏感，它仅对类间方差为单峰的图像产生较好的分割效果。当目标与背景的大小比例悬殊时（例如受光照不均、反光或背景复杂等因素影响），类间方差准则函数可能呈现双峰或多峰，此时效果不好。

## 3.2 调用Ostu方法获得阈值，进行固定化阈值操作下面图像。

