OpenCV--基本矩阵和图像操作

下面将介绍一下openCV提供的基本函数。这类函数十分有用。

函数名称	描述
`cvAbs`	计算绝对值
`cvAbsDiff`	计算两个数组差值绝对值
`cvAbsDiffS`	计算标量和数组差值绝对值
cvAdd	两个数组相加
cvAdds	一个数组和标量元素集合累加

其它函数都在cvcore.h中有标明，提供的函数库非常强大。下面列举几个例子。

- cvAbs cvAbsDiff和cvAbsDiffS

```
/** dst(x,y,c) = abs(src1(x,y,c) - src2(x,y,c)) */
CVAPI(void) cvAbsDiff( const CvArr* src1, const CvArr* src2, CvArr* dst );

/** dst(x,y,c) = abs(src(x,y,c) - value(c)) */
CVAPI(void) cvAbsDiffS( const CvArr* src, CvArr* dst, CvScalar value );

#define cvAbs( src, dst ) cvAbsDiffS( (src), (dst), cvScalarAll(0))
```

cvAbs是计算数组的绝对值，并把它写入dst数组，cvAbsDiff和cvAbsDiffS是对比两个数组相减，并把结果写入到dst中。

- cvAdd cvAddS cvAddWeighted 和alpha融合

```
/** dst(mask) = src1(mask) + src2(mask) */
CVAPI(void)  cvAdd( const CvArr* src1, const CvArr* src2, CvArr* dst,
                    const CvArr* mask CV_DEFAULT(NULL));

/** dst(mask) = src(mask) + value */
CVAPI(void)  cvAddS( const CvArr* src, CvScalar value, CvArr* dst,
                     const CvArr* mask CV_DEFAULT(NULL));

/** dst = src1 * alpha + src2 * beta + gamma */
```

```
CVAPI(void)  cvAddWeighted( const CvArr* src1, double alpha,
                            const CvArr* src2, double beta,
                            double gamma, CvArr* dst );
```

cvadd是一个简单的加法器，SRC1和SRC2元素累加然后放到dst。如果mask非NULL，则由mask中非零元素指定dstyuansuzhi .adds只是加了一个恒定值。cvAddweighted函数与add类似，只是满足以下条件：

```
dst(I)=src1(I)*alpha+src2(I)*beta+gamma
```

这个方法主要用于图像的融合，是吧src2通过加权的方式融合到src1的方式。可能这两幅图像属于不同的大小尺寸，但是他们的ROI区域大小必须相同。取值参数在1-0之间，可以换算公式：

```
dst(I)=src1(I)*alpha+src2(I)*（1-alpha）+gamma
```

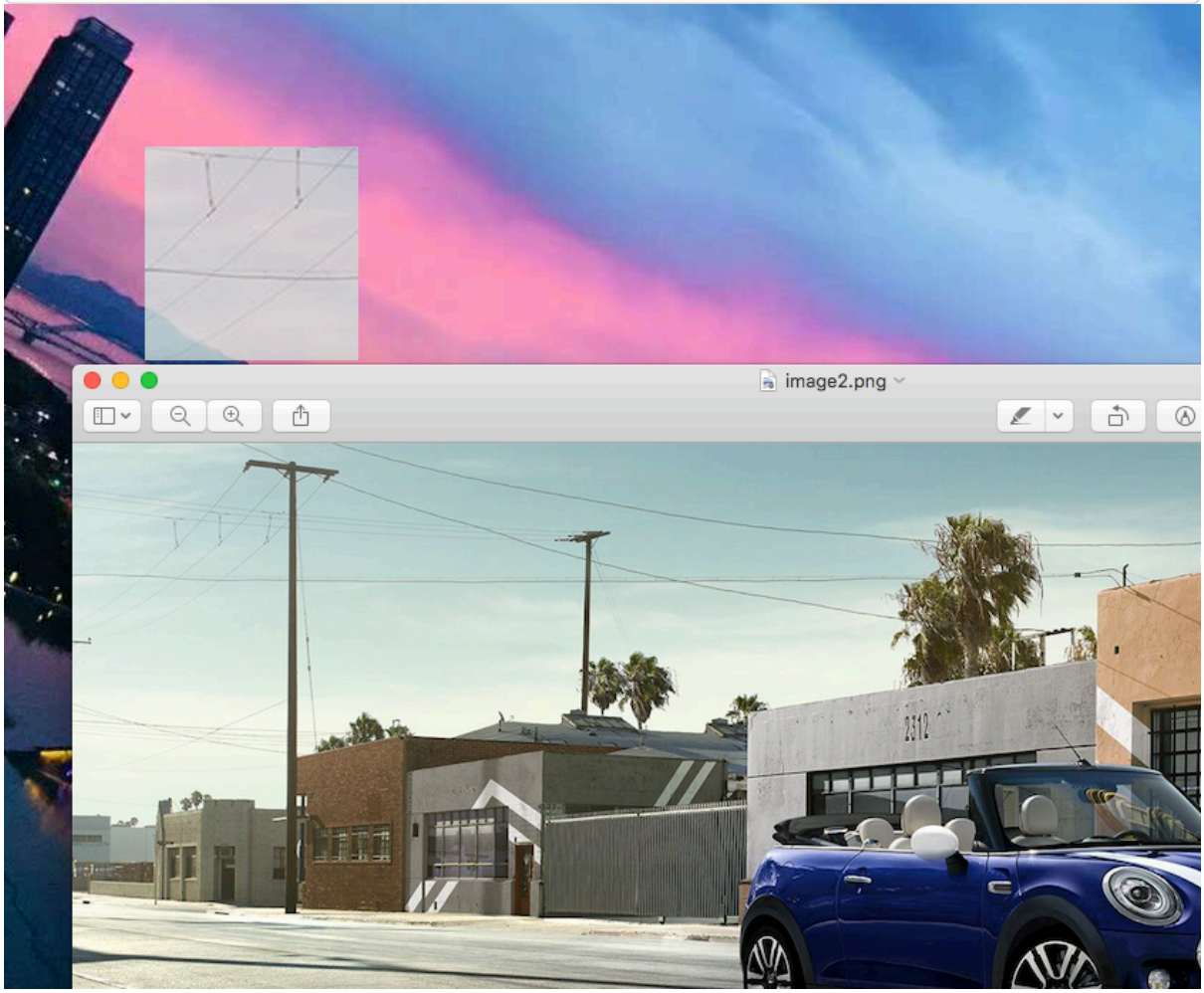
简单举例：

```
#include <iostream>
#include "opencv/highgui.h"
#include "opencv/cv.h"
using namespace std;
using namespace cv;
#define IMAGE_LOAD_PATH  "/Users/genesis/Pictures/IMG_3393.JPG"
#define IMAGE_LOAD_PATH2    "/Users/genesis/Pictures/image2.png"
IplImage* addWidgeted(IplImage* src1,double alpha,IplImage* src2,double beta ,double gamma)
{
    cvAddWeighted(src1,alpha, src2, beta,gamma,src1);
    return src1;
}
int main(int argc, const char * argv[]) {
    IplImage *input = cvLoadImage(IMAGE_LOAD_PATH);
    IplImage *input2 = cvLoadImage(IMAGE_LOAD_PATH2);
    int x= 100;
    int y =100;
    int width = 150;
    int height = 150;
    double  alpha = 0.2;
    double  beta = 0.8;
    double gamma = 0.5;

    cvSetImageROI(input, cvRect(x, y, width, height));
    cvSetImageROI(input2, cvRect(x, y, width, height));
    addWidgeted(input,alpha, input2, beta,gamma);
    cvResetImageROI(input);

    cvShowImage("roi ",input);
```

```
    //fixme 清空Input Image
    cvReleaseImage(&input);
    cvReleaseImage(&input2);
    cvWaitKey();
    return 0;
}
```



- cvAnd和cvAndS

```
/** dst(idx) = src1(idx) & src2(idx) */
CVAPI(void) cvAnd( const CvArr* src1, const CvArr* src2,
                   CvArr* dst, const CvArr* mask CV_DEFAULT(NULL));

/** dst(idx) = src(idx) & value */
CVAPI(void) cvAndS( const CvArr* src, CvScalar value,
                    CvArr* dst, const CvArr* mask CV_DEFAULT(NULL));
```

cvAnd和cvAndS是做两幅图像的与操作，这个函数必须要保证两个数组长度一致性。最后输出到dst中。

- cvAvg和cvAvgSdv

```
/** Calculates mean value of array elements */
CVAPI(CvScalar)  cvAvg( const CvArr* arr, const CvArr* mask
CV_DEFAULT(NULL) );

/** Calculates mean and standard deviation of pixel values */
CVAPI(void)  cvAvgSdv( const CvArr* arr, CvScalar* mean, CvScalar* std_dev,
                       const CvArr* mask CV_DEFAULT(NULL) );
```

cvAvg是计算数组arr中的平均值，和上述一致的mask参数，cvAvgSdv不仅仅可以计算平均值，还可以计算标准差。

- cvCalcCovarMatrix

```
/** flag for cvCalcCovarMatrix, transpose([v1-avg, v2-avg,...]) * [v1-
avg,v2-avg,...] */
#define CV_COVAR_SCRAMBLED 0

/** flag for cvCalcCovarMatrix, [v1-avg, v2-avg,...] * transpose([v1-
avg,v2-avg,...]) */
#define CV_COVAR_NORMAL    1

/** flag for cvCalcCovarMatrix, do not calc average (i.e. mean vector) -
use the input vector instead
   (useful for calculating covariance matrix by parts) */
#define CV_COVAR_USE_AVG   2

/** flag for cvCalcCovarMatrix, scale the covariance matrix coefficients by
number of the vectors */
#define CV_COVAR_SCALE     4

/** flag for cvCalcCovarMatrix, all the input vectors are stored in a
single matrix, as its rows */
#define CV_COVAR_ROWS      8

/** flag for cvCalcCovarMatrix, all the input vectors are stored in a
single matrix, as its columns */
#define CV_COVAR_COLS     16

/** Calculates covariation matrix for a set of vectors
@see @ref core_c_CovarFlags "flags"
*/
CVAPI(void)  cvCalcCovarMatrix( const CvArr** vects, int count,
                                CvArr* cov_mat, CvArr* avg, int flags );
```

给定一个向量，cvCalcCovarMatrix将计算这些点的均值和协方差矩阵。flags的标记作用

如上所示。

- cvCmp和cvCmpS

```
#define CV_CMP_EQ   0
#define CV_CMP_GT   1
#define CV_CMP_GE   2
#define CV_CMP_LT   3
#define CV_CMP_LE   4
#define CV_CMP_NE   5

/** The comparison operation support single-channel arrays only.
   Destination image should be 8uC1 or 8sC1 */

/** dst(idx) = src1(idx) _cmp_op_ src2(idx) */
CVAPI(void) cvCmp( const CvArr* src1, const CvArr* src2, CvArr* dst, int
cmp_op );

/** dst(idx) = src1(idx) _cmp_op_ value */
CVAPI(void) cvCmpS( const CvArr* src, double value, CvArr* dst, int cmp_op
);
```

这个是比较操作，操作符代表意思如下：

Cmp 值	方法公式
CV_CMP_EQ	Src == src2
CV_CMP_GT	Src > src2
CV_CMP_GE	Src > =src2
CV_CMP_LT	Src < src2
CV_CMP_LE	Src <= src2
CV_CMP_NE	Src != src2

这类函数只适用于单通道的矩阵。

- cvConvertScale 和 cvConvertScaleAbs

```
/** @brief Converts one array to another with optional linear
```

```
transformation.
The function has several different purposes, and thus has several different
names. It copies one
array to another with optional scaling, which is performed first, and/or
optional type conversion,
performed after:
\f[\texttt{dst} (I) =  \texttt{scale} \texttt{src} (I) + ( \texttt{shift}
_0, \texttt{shift} _1,...)\f]
All the channels of multi-channel arrays are processed independently.
The type of conversion is done with rounding and saturation, that is if the
result of scaling +
conversion can not be represented exactly by a value of the destination
array element type, it is
set to the nearest representable value on the real axis.
@param src Source array
@param dst Destination array
@param scale Scale factor
@param shift Value added to the scaled source array elements
*/

CVAPI(void)  cvConvertScale( const CvArr* src, CvArr* dst,
                             double scale CV_DEFAULT(1),
                             double shift CV_DEFAULT(0) );

/** Performs linear transformation on every source array element,
   stores absolute value of the result:
   dst(x,y,c) = abs(scale*src(x,y,c)+shift).
   destination array must have 8u type.
   In other cases one may use cvConvertScale + cvAbsDiffS */
CVAPI(void)  cvConvertScaleAbs( const CvArr* src, CvArr* dst,
                                double scale CV_DEFAULT(1),
                                double shift CV_DEFAULT(0) );
```

这个函数集合了很多的功能，如果有需要，他可以执行多个功能，第一个功能就是把特定图像数据转变成目标图像的数据类型。例如一个8bit的RGB灰度图像，想要把他变成16bit有符号的图像，就可以使用这个方法。

这个方法的第二个功能是对图像数据执行线性的变换，在转换成功后，每个元素乘以一个比例值scale，然后加上shift的数值在附加到每个像素点上去。

至关重要的一点是，convert和scale在名词上和实际是相反的顺序执行，意味着，在执行转换前，就已经将scale和shift计算进行成功了。

最后注意是图片的原图数据通道必须要一致，否则无法转换成功。

- CVCopy

```
/** @brief Copies one array to another.
The function copies selected elements from an input array to an output
array:
```

```
\f[\texttt{dst} (I)= \texttt{src} (I)  \quad \text{if} \quad \texttt{mask}
(I)  \ne 0.\f]
If any of the passed arrays is of IplImage type, then its ROI and COI
fields are used. Both arrays
must have the same type, the same number of dimensions, and the same size.
The function can also
copy sparse arrays (mask is not supported in this case).
@param src The source array
@param dst The destination array
@param mask Operation mask, 8-bit single channel array; specifies elements
of the destination array
to be changed
*/
CVAPI(void)  cvCopy( const CvArr* src, CvArr* dst,
                     const CvArr* mask CV_DEFAULT(NULL) );
```

简单的理解为图像copy，如果是单一一个稀疏矩阵的复制，mask参数无效果。

- cvCrossProduct

```
/** @brief Calculates the cross product of two 3D vectors.
The function calculates the cross product of two 3D vectors:
\f[\texttt{dst} =  \texttt{src1} \times \texttt{src2}\f]
or:
\f[\begin{array}{l} \texttt{dst} _1 =  \texttt{src1} _2  \texttt{src2} _3
-  \texttt{src1} _3  \texttt{src2} _2 \\ \texttt{dst} _2 =  \texttt{src1}
_3  \texttt{src2} _1 -  \texttt{src1} _1  \texttt{src2} _3 \\ \texttt{dst}
_3 =  \texttt{src1} _1  \texttt{src2} _2 -  \texttt{src1} _2  \texttt{src2}
_1 \end{array}\f]
@param src1 The first source vector
@param src2 The second source vector
@param dst The destination vector
*/

CVAPI(void)  cvCrossProduct( const CvArr* src1, const CvArr* src2, CvArr*
dst );
```

这个方法主要是三维矩阵的差积计算。

- cvCvtColor

```
/** Constants for color conversion */
enum
{
    CV_BGR2BGRA    =0,
    CV_RGB2RGBA    =CV_BGR2BGRA,
    CV_BGRA2BGR    =1,
    CV_RGBA2RGB    =CV_BGRA2BGR,
```

```
    CV_BGR2RGBA    =2,
    CV_RGB2BGRA    =CV_BGR2RGBA,
    CV_RGBA2BGR    =3,
    CV_BGRA2RGB    =CV_RGBA2BGR,
    CV_BGR2RGB     =4,
    CV_RGB2BGR     =CV_BGR2RGB,
    CV_BGRA2RGBA   =5,
    CV_RGBA2BGRA   =CV_BGRA2RGBA,
    CV_BGR2GRAY    =6,
    CV_RGB2GRAY    =7,
    CV_GRAY2BGR    =8,
    CV_GRAY2RGB    =CV_GRAY2BGR,
    CV_GRAY2BGRA   =9,
    CV_GRAY2RGBA   =CV_GRAY2BGRA,
    CV_BGRA2GRAY   =10,
    CV_RGBA2GRAY   =11,
    CV_BGR2BGR565  =12,
    CV_RGB2BGR565  =13,
    CV_BGR5652BGR  =14,
    CV_BGR5652RGB  =15,
    CV_BGRA2BGR565 =16,
    CV_RGBA2BGR565 =17,
    CV_BGR5652BGRA =18,
    CV_BGR5652RGBA =19,
    CV_GRAY2BGR565 =20,
    CV_BGR5652GRAY =21,
    CV_BGR2BGR555  =22,
    CV_RGB2BGR555  =23,
    CV_BGR5552BGR  =24,
    CV_BGR5552RGB  =25,
    CV_BGRA2BGR555 =26,
    CV_RGBA2BGR555 =27,
    CV_BGR5552BGRA =28,
    CV_BGR5552RGBA =29,
    CV_GRAY2BGR555 =30,
    CV_BGR5552GRAY =31,
    CV_BGR2XYZ     =32,
    CV_RGB2XYZ     =33,
    CV_XYZ2BGR     =34,
    CV_XYZ2RGB     =35,
    CV_BGR2YCrCb   =36,
    CV_RGB2YCrCb   =37,
    CV_YCrCb2BGR   =38,
    CV_YCrCb2RGB   =39,
    CV_BGR2HSV     =40,
    CV_RGB2HSV     =41,
    CV_BGR2Lab     =44,
    CV_RGB2Lab     =45,
    CV_BayerBG2BGR =46,
    CV_BayerGB2BGR =47,
    CV_BayerRG2BGR =48,
```

```
    CV_BayerGR2BGR =49,
    CV_BayerBG2RGB =CV_BayerRG2BGR,
    CV_BayerGB2RGB =CV_BayerGR2BGR,
    CV_BayerRG2RGB =CV_BayerBG2BGR,
    CV_BayerGR2RGB =CV_BayerGB2BGR,
    CV_BGR2Luv     =50,
    CV_RGB2Luv     =51,
    CV_BGR2HLS     =52,
    CV_RGB2HLS     =53,
    CV_HSV2BGR     =54,
    CV_HSV2RGB     =55,
    CV_Lab2BGR     =56,
    CV_Lab2RGB     =57,
    CV_Luv2BGR     =58,
    CV_Luv2RGB     =59,
    CV_HLS2BGR     =60,
    CV_HLS2RGB     =61,
    CV_BayerBG2BGR_VNG =62,
    CV_BayerGB2BGR_VNG =63,
    CV_BayerRG2BGR_VNG =64,
    CV_BayerGR2BGR_VNG =65,
    CV_BayerBG2RGB_VNG =CV_BayerRG2BGR_VNG,
    CV_BayerGB2RGB_VNG =CV_BayerGR2BGR_VNG,
    CV_BayerRG2RGB_VNG =CV_BayerBG2BGR_VNG,
    CV_BayerGR2RGB_VNG =CV_BayerGB2BGR_VNG,
    CV_BGR2HSV_FULL = 66,
    CV_RGB2HSV_FULL = 67,
    CV_BGR2HLS_FULL = 68,
    CV_RGB2HLS_FULL = 69,
    CV_HSV2BGR_FULL = 70,
    CV_HSV2RGB_FULL = 71,
    CV_HLS2BGR_FULL = 72,
    CV_HLS2RGB_FULL = 73,
    CV_LBGR2Lab     = 74,
    CV_LRGB2Lab     = 75,
    CV_LBGR2Luv     = 76,
    CV_LRGB2Luv     = 77,
    CV_Lab2LBGR     = 78,
    CV_Lab2LRGB     = 79,
    CV_Luv2LBGR     = 80,
    CV_Luv2LRGB     = 81,
    CV_BGR2YUV      = 82,
    CV_RGB2YUV      = 83,
    CV_YUV2BGR      = 84,
    CV_YUV2RGB      = 85,
    CV_BayerBG2GRAY = 86,
    CV_BayerGB2GRAY = 87,
    CV_BayerRG2GRAY = 88,
    CV_BayerGR2GRAY = 89,
    //YUV 4:2:0 formats family
    CV_YUV2RGB_NV12 = 90,
```

```
    CV_YUV2BGR_NV12 = 91,
    CV_YUV2RGB_NV21 = 92,
    CV_YUV2BGR_NV21 = 93,
    CV_YUV420sp2RGB = CV_YUV2RGB_NV21,
    CV_YUV420sp2BGR = CV_YUV2BGR_NV21,
    CV_YUV2RGBA_NV12 = 94,
    CV_YUV2BGRA_NV12 = 95,
    CV_YUV2RGBA_NV21 = 96,
    CV_YUV2BGRA_NV21 = 97,
    CV_YUV420sp2RGBA = CV_YUV2RGBA_NV21,
    CV_YUV420sp2BGRA = CV_YUV2BGRA_NV21,
    CV_YUV2RGB_YV12 = 98,
    CV_YUV2BGR_YV12 = 99,
    CV_YUV2RGB_IYUV = 100,
    CV_YUV2BGR_IYUV = 101,
    CV_YUV2RGB_I420 = CV_YUV2RGB_IYUV,
    CV_YUV2BGR_I420 = CV_YUV2BGR_IYUV,
    CV_YUV420p2RGB = CV_YUV2RGB_YV12,
    CV_YUV420p2BGR = CV_YUV2BGR_YV12,
    CV_YUV2RGBA_YV12 = 102,
    CV_YUV2BGRA_YV12 = 103,
    CV_YUV2RGBA_IYUV = 104,
    CV_YUV2BGRA_IYUV = 105,
    CV_YUV2RGBA_I420 = CV_YUV2RGBA_IYUV,
    CV_YUV2BGRA_I420 = CV_YUV2BGRA_IYUV,
    CV_YUV420p2RGBA = CV_YUV2RGBA_YV12,
    CV_YUV420p2BGRA = CV_YUV2BGRA_YV12,
    CV_YUV2GRAY_420 = 106,
    CV_YUV2GRAY_NV21 = CV_YUV2GRAY_420,
    CV_YUV2GRAY_NV12 = CV_YUV2GRAY_420,
    CV_YUV2GRAY_YV12 = CV_YUV2GRAY_420,
    CV_YUV2GRAY_IYUV = CV_YUV2GRAY_420,
    CV_YUV2GRAY_I420 = CV_YUV2GRAY_420,
    CV_YUV420sp2GRAY = CV_YUV2GRAY_420,
    CV_YUV420p2GRAY = CV_YUV2GRAY_420,
    //YUV 4:2:2 formats family
    CV_YUV2RGB_UYVY = 107,
    CV_YUV2BGR_UYVY = 108,
    //CV_YUV2RGB_VYUY = 109,
    //CV_YUV2BGR_VYUY = 110,
    CV_YUV2RGB_Y422 = CV_YUV2RGB_UYVY,
    CV_YUV2BGR_Y422 = CV_YUV2BGR_UYVY,
    CV_YUV2RGB_UYNV = CV_YUV2RGB_UYVY,
    CV_YUV2BGR_UYNV = CV_YUV2BGR_UYVY,
    CV_YUV2RGBA_UYVY = 111,
    CV_YUV2BGRA_UYVY = 112,
    //CV_YUV2RGBA_VYUY = 113,
    //CV_YUV2BGRA_VYUY = 114,
    CV_YUV2RGBA_Y422 = CV_YUV2RGBA_UYVY,
    CV_YUV2BGRA_Y422 = CV_YUV2BGRA_UYVY,
    CV_YUV2RGBA_UYNV = CV_YUV2RGBA_UYVY,
```

```
    CV_YUV2BGRA_UYNV = CV_YUV2BGRA_UYVY,
    CV_YUV2RGB_YUY2 = 115,
    CV_YUV2BGR_YUY2 = 116,
    CV_YUV2RGB_YVYU = 117,
    CV_YUV2BGR_YVYU = 118,
    CV_YUV2RGB_YUYV = CV_YUV2RGB_YUY2,
    CV_YUV2BGR_YUYV = CV_YUV2BGR_YUY2,
    CV_YUV2RGB_YUNV = CV_YUV2RGB_YUY2,
    CV_YUV2BGR_YUNV = CV_YUV2BGR_YUY2,
    CV_YUV2RGBA_YUY2 = 119,
    CV_YUV2BGRA_YUY2 = 120,
    CV_YUV2RGBA_YVYU = 121,
    CV_YUV2BGRA_YVYU = 122,
    CV_YUV2RGBA_YUYV = CV_YUV2RGBA_YUY2,
    CV_YUV2BGRA_YUYV = CV_YUV2BGRA_YUY2,
    CV_YUV2RGBA_YUNV = CV_YUV2RGBA_YUY2,
    CV_YUV2BGRA_YUNV = CV_YUV2BGRA_YUY2,
    CV_YUV2GRAY_UYVY = 123,
    CV_YUV2GRAY_YUY2 = 124,
    //CV_YUV2GRAY_VYUY = CV_YUV2GRAY_UYVY,
    CV_YUV2GRAY_Y422 = CV_YUV2GRAY_UYVY,
    CV_YUV2GRAY_UYNV = CV_YUV2GRAY_UYVY,
    CV_YUV2GRAY_YVYU = CV_YUV2GRAY_YUY2,
    CV_YUV2GRAY_YUYV = CV_YUV2GRAY_YUY2,
    CV_YUV2GRAY_YUNV = CV_YUV2GRAY_YUY2,
    // alpha premultiplication
    CV_RGBA2mRGBA = 125,
    CV_mRGBA2RGBA = 126,
    CV_RGB2YUV_I420 = 127,
    CV_BGR2YUV_I420 = 128,
    CV_RGB2YUV_IYUV = CV_RGB2YUV_I420,
    CV_BGR2YUV_IYUV = CV_BGR2YUV_I420,
    CV_RGBA2YUV_I420 = 129,
    CV_BGRA2YUV_I420 = 130,
    CV_RGBA2YUV_IYUV = CV_RGBA2YUV_I420,
    CV_BGRA2YUV_IYUV = CV_BGRA2YUV_I420,
    CV_RGB2YUV_YV12  = 131,
    CV_BGR2YUV_YV12  = 132,
    CV_RGBA2YUV_YV12 = 133,
    CV_BGRA2YUV_YV12 = 134,
    // Edge-Aware Demosaicing
    CV_BayerBG2BGR_EA = 135,
    CV_BayerGB2BGR_EA = 136,
    CV_BayerRG2BGR_EA = 137,
    CV_BayerGR2BGR_EA = 138,
    CV_BayerBG2RGB_EA = CV_BayerRG2BGR_EA,
    CV_BayerGB2RGB_EA = CV_BayerGR2BGR_EA,
    CV_BayerRG2RGB_EA = CV_BayerBG2BGR_EA,
    CV_BayerGR2RGB_EA = CV_BayerGB2BGR_EA,
    CV_BayerBG2BGRA =139,
    CV_BayerGB2BGRA =140,
```

```
    CV_BayerRG2BGRA =141,
    CV_BayerGR2BGRA =142,
    CV_BayerBG2RGBA =CV_BayerRG2BGRA,
    CV_BayerGB2RGBA =CV_BayerGR2BGRA,
    CV_BayerRG2RGBA =CV_BayerBG2BGRA,
    CV_BayerGR2RGBA =CV_BayerGB2BGRA,
    CV_COLORCVT_MAX  = 143
};
/** Sub-pixel interpolation methods */
enum
{
    CV_INTER_NN        =0,
    CV_INTER_LINEAR    =1,
    CV_INTER_CUBIC     =2,
    CV_INTER_AREA      =3,
    CV_INTER_LANCZOS4  =4
};
/** @brief Converts input array pixels from one color space to another
@see cv::cvtColor
*/
CVAPI(void)  cvCvtColor( const CvArr* src, CvArr* dst, int code )
```

这个方法是颜色通道切换的通道，并且附上对应的代码定义。

值得注意的是HSV彩色模式或者HLS色彩模式来说，色调通常是在0·360度之间，在8位图设计中，转换可能出现问题，因此，转换HSV色彩模式时，转换8位图的形式输出需要色调/2的方式。

- cvDel

```
/** Calculates determinant of input matrix */
CVAPI(double) cvDet( const CvArr* mat );
```

此方法用于计算一个方阵的行列式。这个数组可以是任何数据类型，但它必须是单通道，此方法对应小型矩阵，大型矩阵计算效率不是很高。

- cvDiv

```
/** element-wise division/inversion with scaling:
    dst(idx) = src1(idx) * scale / src2(idx)
    or dst(idx) = scale / src2(idx) if src1 == 0 */

CVAPI(void)  cvDiv( const CvArr* src1, const CvArr* src2,
                    CvArr* dst, double scale CV_DEFAULT(1));
```

cvDiv是一个简单的除法器，应用于src2除以src1对应的元素。然后把结果保存到dst中。如果src1是NuLL，则默认是除以1的矩阵。

- cvDotProduct

```
/** @brief Calculates the dot product of two arrays in Euclidean metrics.
The function calculates and returns the Euclidean dot product of two
arrays.
\f[src1  \bullet src2 =  \sum _I ( \texttt{src1} (I)  \texttt{src2} (I))\f]
In the case of multiple channel arrays, the results for all channels are
accumulated. In particular,
cvDotProduct(a,a) where a is a complex vector, will return
\f$||\texttt{a}||^2\f$. The function can
process multi-dimensional arrays, row by row, layer by layer, and so on.
@param src1 The first source array
@param src2 The second source array
*/
CVAPI(double)  cvDotProduct( const CvArr* src1, const CvArr* src2 );
```

cvDotProduct计算两个N维的向量积。切src1和src2都是单通道数组。

- cvEigenVV

```
/** Finds eigen values and vectors of a symmetric matrix */

CVAPI(void)  cvEigenVV( CvArr* mat, CvArr* evects, CvArr* evals,
                        double eps CV_DEFAULT(0),
                        int lowindex CV_DEFAULT(-1),
                        int highindex CV_DEFAULT(-1));
```

cvEigenVV是对称矩阵计算特征值和相应的特征向量。函数返回时，会在evects中找到以行列为例的特征向量，对应的特征值会存储到evals中。

- cvFlip

```
/** Mirror array data around horizontal (flip=0),
   vertical (flip=1) or both(flip=-1) axises:
   cvFlip(src) flips images vertically and sequences horizontally (inplace)
*/

CVAPI(void)  cvFlip( const CvArr* src, CvArr* dst CV_DEFAULT(NULL),
                     int flip_mode CV_DEFAULT(0));
```

本方法是提供图像的旋转，围绕X轴或者Y轴，亦或者是XY轴同时旋转。当flipmode是正时，会沿着Y轴旋转，当为负值时候，会沿着XY轴旋转。

- cvGEMM

```
/** Matrix transform: dst = A*B + C, C is optional */
#define cvMatMulAdd( src1, src2, src3, dst ) cvGEMM( (src1), (src2), 1.,
(src3), 1., (dst), 0 )
```

```
#define cvMatMul( src1, src2, dst )  cvMatMulAdd( (src1), (src2), NULL,
(dst))
#define CV_GEMM_A_T 1
#define CV_GEMM_B_T 2
#define CV_GEMM_C_T 4
/** Extended matrix transform:
   dst = alpha*op(A)*op(B) + beta*op(C), where op(X) is X or X^T */

CVAPI(void)  cvGEMM( const CvArr* src1, const CvArr* src2, double alpha,
                     const CvArr* src3, double beta, CvArr* dst,
                     int tABC CV_DEFAULT(0));
```

该方法是广义矩阵乘法，可实现矩阵乘法，转至后相乘，比例缩放等。计算的公式如下：

D = alpha* op(a)* op(b)+ beta* op(C)
其中A,B,C分别代表了src1,src2,src3.且只有当矩阵大小符合条件，才能实现矩阵的乘法。

- cvGetCol和cvGetCols

```
/** @brief Returns one of more array columns.
The function returns the header, corresponding to a specified column span
of the input array. That
is, no data is copied. Therefore, any modifications of the submatrix will
affect the original array.
If you need to copy the columns, use cvCloneMat. cvGetCol(arr, submat, col)
is a shortcut for
cvGetCols(arr, submat, col, col+1).
@param arr Input array
@param submat Pointer to the resulting sub-array header
@param start_col Zero-based index of the starting column (inclusive) of the
span
@param end_col Zero-based index of the ending column (exclusive) of the
span
*/
CVAPI(CvMat*) cvGetCols( const CvArr* arr, CvMat* submat,
                         int start_col, int end_col );
/** @overload
@param arr Input array
@param submat Pointer to the resulting sub-array header
@param col Zero-based index of the selected column
*/

CV_INLINE  CvMat*  cvGetCol( const CvArr* arr, CvMat* submat, int col )
{
    return cvGetCols( arr, submat, col, col + 1 );
}
```

提取矩阵中的某一列，并以向量的形式返回。两者函数作用类似。

- cvGetDiag

```
/** @brief Returns one of array diagonals.
The function returns the header, corresponding to a specified diagonal of
the input array.
@param arr Input array
@param submat Pointer to the resulting sub-array header
@param diag Index of the array diagonal. Zero value corresponds to the main
diagonal, -1
corresponds to the diagonal above the main, 1 corresponds to the diagonal
below the main, and so
forth.
*/
CVAPI(CvMat*) cvGetDiag( const CvArr* arr, CvMat* submat,
                             int diag CV_DEFAULT(0));
```

获取一个矩阵的对角线，并且将其返回。

- cvGetDims和cvGetDimSize

```
/** @brief Return number of array dimensions
The function returns the array dimensionality and the array of dimension
sizes. In the case of
IplImage or CvMat it always returns 2 regardless of number of image/matrix
rows. For example, the
following code calculates total number of array elements:
@code
    int sizes[CV_MAX_DIM];
    int i, total = 1;
    int dims = cvGetDims(arr, size);
    for(i = 0; i < dims; i++ )
        total *= sizes[i];
@endcode
@param arr Input array
@param sizes Optional output vector of the array dimension sizes. For 2d
arrays the number of rows
(height) goes first, number of columns (width) next.
*/

CVAPI(int) cvGetDims( const CvArr* arr, int* sizes CV_DEFAULT(NULL) );

/** @brief Returns array size along the specified dimension.
@param arr Input array
```

```
@param index Zero-based dimension index (for matrices 0 means number of
rows, 1 means number of
columns; for images 0 means height, 1 means width)
*/

CVAPI(int) cvGetDimSize( const CvArr* arr, int index );
```

返回指定数组的维数，如果size不等于空，则被写入size。

- cvGetRow和cvGetRows

```
/** @brief Returns array row or row span.
The function returns the header, corresponding to a specified row/row span
of the input array.
cvGetRow(arr, submat, row) is a shortcut for cvGetRows(arr, submat, row,
row+1).
@param arr Input array
@param submat Pointer to the resulting sub-array header
@param start_row Zero-based index of the starting row (inclusive) of the
span
@param end_row Zero-based index of the ending row (exclusive) of the span
@param delta_row Index step in the row span. That is, the function extracts
every delta_row -th
row from start_row and up to (but not including) end_row .
*/
CVAPI(CvMat*) cvGetRows( const CvArr* arr, CvMat* submat,
                         int start_row, int end_row,
                         int delta_row CV_DEFAULT(1));

/** @overload
@param arr Input array
@param submat Pointer to the resulting sub-array header
@param row Zero-based index of the selected row
*/
CV_INLINE  CvMat*  cvGetRow( const CvArr* arr, CvMat* submat, int row )
{
    return cvGetRows( arr, submat, row, row + 1, 1 );
}
```

获取矩阵的某一行，并且返回对应的向量值。

- cvGetSize

```
/** @brief Returns size of matrix or image ROI.
The function returns number of rows (CvSize::height) and number of columns
(CvSize::width) of the
input matrix or image. In the case of image the size of ROI is returned.
@param arr array header
```

```
*/
CVAPI(CvSize) cvGetSize( const CvArr* arr );
```

与getDims差不多的设计，只是getsize是为了图像和矩阵设计的，维度总是2，在创建一个大小相同的图形时，使用这个方法会更简单些。

- cvGetSubRect

```
/** @brief Returns matrix header corresponding to the rectangular sub-array
of input image or matrix.
The function returns header, corresponding to a specified rectangle of the
input array. In other
words, it allows the user to treat a rectangular part of input array as a
stand-alone array. ROI is
taken into account by the function so the sub-array of ROI is actually
extracted.
@param arr Input array
@param submat Pointer to the resultant sub-array header
@param rect Zero-based coordinates of the rectangle of interest
*/
CVAPI(CvMat*) cvGetSubRect( const CvArr* arr, CvMat* submat, CvRect rect
);
```

cvGetSubRect的设计和getcol,getrow类似，只是可以通过参数对比，获取一个区域的子矩阵。

- cvInRange和cvInRangeS

```
/** dst(idx) = lower(idx) <= src(idx) < upper(idx) */
CVAPI(void) cvInRange( const CvArr* src, const CvArr* lower,
                       const CvArr* upper, CvArr* dst );
/** dst(idx) = lower <= src(idx) < upper */
CVAPI(void) cvInRangeS( const CvArr* src, CvScalar lower,
                        CvScalar upper, CvArr* dst );
```

这个函数是监测图像中像素的灰度是否属于某一指定的范围区间。其实类似二值化，在区间内的为0xff，否则为0x00。

- cvInvert

```
#define CV_LU  0
#define CV_SVD 1
#define CV_SVD_SYM 2
#define CV_CHOLESKY 3
#define CV_QR  4
```

```
#define CV_NORMAL 16

/** Inverts matrix */
CVAPI(double)  cvInvert( const CvArr* src, CvArr* dst,
                         int method CV_DEFAULT(CV_LU));
```

cvInvert求得矩阵的逆，并且将结果保存到dst中。这个方法可以使用多种方式来求解矩阵的逆。CV_LU 高斯消去法、CV_SVD 奇异值分解法，CV_SVD_SYM 对称矩阵SVD。

- cvMahalonobls

```
/** Calculates Mahalanobis(weighted) distance */
CVAPI(double)  cvMahalanobis( const CvArr* vec1, const CvArr* vec2, const
CvArr* mat );
```

cvMahalonobls 是求一点和高斯分布中心之间的向量距离，该距离是方阵协方差的逆归一化标准。计算公式如下：

$$D_{\text{Mahalanobis}}(x,y)=\sqrt{(x-y)^T\Sigma^{-1}(x-y)}$$

- cvMax和cvMaxS或者 cvMin cvMinS

```
** dst(idx) = max(src1(idx),src2(idx)) */
CVAPI(void) cvMax( const CvArr* src1, const CvArr* src2, CvArr* dst );
/** dst(idx) = max(src(idx),value) */
CVAPI(void) cvMaxS( const CvArr* src, double value, CvArr* dst );
```

cvMax是计算src1和src2方阵中得最大值.最小值。

- cvMerge

```
/** Merges a set of single-channel arrays into the single multi-channel
array
   or inserts one particular [color] plane to the array */
CVAPI(void)  cvMerge( const CvArr* src0, const CvArr* src1,
                      const CvArr* src2, const CvArr* src3,
                      CvArr* dst );
```

cvSplit的逆运算。合并数组到dst中。

- cvMinMaxLoc

```
/** Finds global minimum, maximum and their positions */
CVAPI(void)  cvMinMaxLoc( const CvArr* arr, double* min_val, double*
max_val,
```

```
                          CvPoint* min_loc CV_DEFAULT(NULL),
                          CvPoint* max_loc CV_DEFAULT(NULL),
                          const CvArr* mask CV_DEFAULT(NULL) );
```

找到一个矩阵最大值和最小值，极值位置会写入minloc或者maxloc中。

- cvMul

```
/** dst(idx) = src1(idx) * src2(idx) * scale
   (scaled element-wise multiplication of 2 arrays) */
CVAPI(void)  cvMul( const CvArr* src1, const CvArr* src2,
                    CvArr* dst, double scale CV_DEFAULT(1) );
```

cvMul矩阵乘法，然后把结果塞入到dst中。

- cvNor

```
/** dst(idx) = ~src(idx) */
CVAPI(void) cvNot( const CvArr* src, CvArr* dst );
```

矩阵取反。

- cvNorm

```
/** @anchor core_c_NormFlags
  @name Flags for cvNorm and cvNormalize
  @{
*/
#define CV_C            1
#define CV_L1           2
#define CV_L2           4
#define CV_NORM_MASK    7
#define CV_RELATIVE     8
#define CV_DIFF         16
#define CV_MINMAX       32

/** @} */
/** Finds norm, difference norm or relative difference norm for an array
(or two arrays)
@see ref core_c_NormFlags "flags"
*/
CVAPI(double)  cvNorm( const CvArr* arr1, const CvArr* arr2
CV_DEFAULT(NULL),
                       int norm_type CV_DEFAULT(CV_L2),
                       const CvArr* mask CV_DEFAULT(NULL) );
```

计算相对距离。计算方式：

CV_C	arr = max(xy) abs(arr1(x,y)) 或者 ||arr1-arr2||C = maxI abs(ar
CV_L1	||arr1||L1 = sumI abs(arr1(I)), sumI abs(arr1(I)-arr2(I))
CV_L2	||arr1||L2 = sqrt(sumI arr1(I)2) sqrt(sumI (arr1(I)-arr2(I)
CV_RELATIVE_C	norm = ||arr1-arr2||C/||arr2||C
CV_RELATIVE_L1	norm = ||arr1-arr2||L1/||arr2||L1
CV_RELATIVE_L2	norm = ||arr1-arr2||L2/||arr2||L2

- cvNormalize

```
/** @see ref core_c_NormFlags "flags" */
CVAPI(void)  cvNormalize( const CvArr* src, CvArr* dst,
                          double a CV_DEFAULT(1.), double b CV_DEFAULT(0.),
                          int norm_type CV_DEFAULT(CV_L2),
                          const CvArr* mask CV_DEFAULT(NULL) );
```

和上述类似。

- cvOr和cvOrS

```
/** dst(idx) = src1(idx) | src2(idx) */
CVAPI(void) cvOr( const CvArr* src1, const CvArr* src2,
                  CvArr* dst, const CvArr* mask CV_DEFAULT(NULL));
/** dst(idx) = src(idx) | value */
CVAPI(void) cvOrS( const CvArr* src, CvScalar value,
                   CvArr* dst, const CvArr* mask CV_DEFAULT(NULL));
```

或运算。

- cvReduce

```
/** @anchor core_c_ReduceFlags
  @name Flags for cvReduce
  @{
*/
#define CV_REDUCE_SUM 0
#define CV_REDUCE_AVG 1
#define CV_REDUCE_MAX 2
#define CV_REDUCE_MIN 3
/** @} */
/** @see @ref core_c_ReduceFlags "flags" */
```

```
CVAPI(void)  cvReduce( const CvArr* src, CvArr* dst, int dim
CV_DEFAULT(-1),
                       int op CV_DEFAULT(CV_REDUCE_SUM) );
```

向量计算，具体看定义传入的TYPE值。

- cvRepeat

```
/** Repeats source 2d array several times in both horizontal and
   vertical direction to fill destination array */
CVAPI(void) cvRepeat( const CvArr* src, CvArr* dst );
```

复制函数。

- cvScale 或者CVConvertScale的宏定义。
- cvSet和cvSetZero

```
/** @brief Sets every element of an array to a given value.
The function copies the scalar value to every selected element of the
destination array:
\f[\texttt{arr} (I)= \texttt{value} \quad \text{if} \quad \texttt{mask}
(I)  \ne 0\f]
If array arr is of IplImage type, then is ROI used, but COI must not be
set.
@param arr The destination array
@param value Fill value
@param mask Operation mask, 8-bit single channel array; specifies elements
of the destination
array to be changed
*/
CVAPI(void)  cvSet( CvArr* arr, CvScalar value,
                    const CvArr* mask CV_DEFAULT(NULL) );
/** @brief Clears the array.
The function clears the array. In the case of dense arrays (CvMat, CvMatND
or IplImage),
cvZero(array) is equivalent to cvSet(array,cvScalarAll(0),0). In the case
of sparse arrays all the
elements are removed.
@param arr Array to be cleared
*/
CVAPI(void)  cvSetZero( CvArr* arr );
```

设置值函数。

- cvSetIdentity

```
///* Finds selected eigen values and vectors of a symmetric matrix */
//CVAPI(void)  cvSelectedEigenVV( CvArr* mat, CvArr* evects, CvArr* evals,
//                                int lowindex, int highindex );
```

```
/** Makes an identity matrix (mat_ij = i == j) */
CVAPI(void)  cvSetIdentity( CvArr* mat, CvScalar value
CV_DEFAULT(cvRealScalar(1)) );
```

会把除了行列数相等的元素保留，其它都置于0。

- cvSolve

```
#define CV_LU  0
#define CV_SVD 1
#define CV_SVD_SYM 2
#define CV_CHOLESKY 3
#define CV_QR  4
#define CV_NORMAL 16

/** Solves linear system (src1)*(dst) = (src2)
   (returns 0 if src1 is a singular and CV_LU method is used) */
CVAPI(int)  cvSolve( const CvArr* src1, const CvArr* src2, CvArr* dst,
                     int method CV_DEFAULT(CV_LU));
```

求解线性方程。

- cvSplit

```
/** Splits a multi-channel array into the set of single-channel arrays or
   extracts particular [color] plane */
CVAPI(void)  cvSplit( const CvArr* src, CvArr* dst0, CvArr* dst1,
                      CvArr* dst2, CvArr* dst3 );
```

把一个图形单通道提取出来。

- cvSub cvSubS和CVSubRS

```
/** dst(mask) = src1(mask) - src2(mask) */
CVAPI(void)  cvSub( const CvArr* src1, const CvArr* src2, CvArr* dst,
                    const CvArr* mask CV_DEFAULT(NULL));
/** dst(mask) = src(mask) - value = src(mask) + (-value) */
CV_INLINE  void  cvSubS( const CvArr* src, CvScalar value, CvArr* dst,
                         const CvArr* mask CV_DEFAULT(NULL))
{
    cvAddS( src, cvScalar( -value.val[0], -value.val[1], -value.val[2], -
value.val[3]),
            dst, mask );
}
/** dst(mask) = value - src(mask) */
CVAPI(void)  cvSubRS( const CvArr* src, CvScalar value, CvArr* dst,
                      const CvArr* mask CV_DEFAULT(NULL));
```

减法函数方法。

- cvSum

```
/** Finds sum of array elements */
CVAPI(CvScalar)  cvSum( const CvArr* arr );
```

cvSum计算所有像素的总和。

- cvSVD

```
#define CV_SVD_MODIFY_A   1          //允许改变矩阵
#define CV_SVD_U_T        2          //返回UT而不是U
#define CV_SVD_V_T        4          //返回VT而不是V
/** Performs Singular Value Decomposition of a matrix */

CVAPI(void)   cvSVD( CvArr* A, CvArr* W, CvArr* U CV_DEFAULT(NULL),
                     CvArr* V CV_DEFAULT(NULL), int flags CV_DEFAULT(0));
```

奇异值分解按照以下公式分解：
A= U* W * V（T)

- cvTrace

```
/** Calculates trace of the matrix (sum of elements on the main diagonal)
*/
CVAPI(CvScalar) cvTrace( const CvArr* mat );
```

cvTrace 是矩阵对角线的总和。

- cvTranspose和cvT

```
/** Tranposes matrix. Square matrices can be transposed in-place */

CVAPI(void)  cvTranspose( const CvArr* src, CvArr* dst );

#define cvT cvTranspose
```

将src中的元素值复制到dst中，行号和列好相调换位置。该函数不支持多通道数组，cvt是宏定义。

- cvXor和cvXorS

```
/** dst(idx) = src1(idx) ^ src2(idx) */
CVAPI(void) cvXor( const CvArr* src1, const CvArr* src2,
                   CvArr* dst, const CvArr* mask CV_DEFAULT(NULL));

/** dst(idx) = src(idx) ^ value */
CVAPI(void) cvXorS( const CvArr* src, CvScalar value,
                    CvArr* dst, const CvArr* mask CV_DEFAULT(NULL));
```

对两个数组进行异或操作。