Learn OpenCV by Examples

OpenCV simplified for beginners by the use of examples. Learn OpenCV with basic implementation of different algorithms. Please visit LearnOpenCV.com for newer articles.

Histogram Calculation

void **calcHist**(const Mat* images, int nimages, const int* channels, InputArray mask, OutputArray hist, int dims, const int* histSize, const float** ranges, bool uniform=true, bool accumulate=false)

Calculates a histogram of a set of arrays.

Parameters:

- **images** – Source arrays. They all should have the same depth, `CV_8U` or `CV_32F`, and the same size. Each of them can have an arbitrary number of channels.
- **nimages** – Number of source images.
- **channels** – List of the `dims` channels used to compute the histogram. The first array channels are numerated from 0 to `images[0].channels()-1`, the second array channels are counted from `images[0].channels()` to `images[0].channels() + images[1].channels()-1`, and so on.
- **mask** – Optional mask. If the matrix is not empty, it must be an 8-bit array of the same size as `images[i]`. The non-zero mask elements mark the array elements counted in the histogram.
- **hist** – Output histogram, which is a dense or sparse `dims`-dimensional array.
- **dims** – Histogram dimensionality that must be positive and not greater than `CV_MAX_DIMS` (equal to 32 in the current OpenCV version).
- **histSize** – Array of histogram sizes in each dimension.
- **ranges** – Array of the `dims` arrays of the histogram bin boundaries in each dimension. When the histogram is uniform (`uniform`=true), then for each dimension `i` it is enough to specify the lower (inclusive) boundary L_0 of the 0-th histogram bin and the upper (exclusive) boundary U_{\texttt{histSize}[i]-1} for the last histogram bin `histSize[i]-1`. That is, in case of a uniform histogram each of `ranges[i]` is an array of 2 elements. When the histogram is not uniform (`uniform=false`), then each of `ranges[i]` contains `histSize[i]+1` elements: L_0, U_0=L_1, U_1=L_2, ..., U_{\texttt{histSize[i]}-2}=L_{\texttt{histSize[i]}-1}, U_{\texttt{histSize[i]}-1} . The array elements, that are not between L_0 and U_{\texttt{histSize[i]}-1}, are not counted in the histogram.
- **uniform** – Flag indicating whether the histogram is uniform or not (see above).
- **accumulate** – Accumulation flag. If it is set, the histogram is not cleared in the beginning when it is allocated. This feature enables you to compute a single histogram from several sets of arrays, or to update the histogram in time.

void **normalize**(InputArray src, OutputArray dst, double alpha=1, double beta=0, int norm_type=NORM_L2, int dtype=-1, InputArray mask=noArray())

(or)

void **normalize**(const SparseMat& src, SparseMat& dst, double alpha, int normType)

Normalizes the norm or value range of an array.

Parameters:

- **src** – input array.
- **dst** – output array of the same size as `src`.
- **alpha** – norm value to normalize to or the lower range boundary in case of the range normalization.
- **beta** – upper range boundary in case of the range normalization; it is not used for the norm normalization.
- **normType** – normalization type (`NORM_MINMAX`, `NORM_INF`, `NORM_L1`, or `NORM_L2`).
- **dtype** – when negative, the output array has the same type as `src`; otherwise, it has the same number of channels as `src` and the depth =`CV_MAT_DEPTH(dtype)`.
- **mask** – optional operation mask.

The functions `normalize` scale and shift the input array elements so that

(where p=Inf, 1 or 2) when `normType=NORM_INF`, `NORM_L1`, or `NORM_L2`, respectively; or so that

$$\min_I \texttt{dst}(I)=\texttt{alpha}, \quad \max_I \texttt{dst}(I)=\texttt{beta}$$

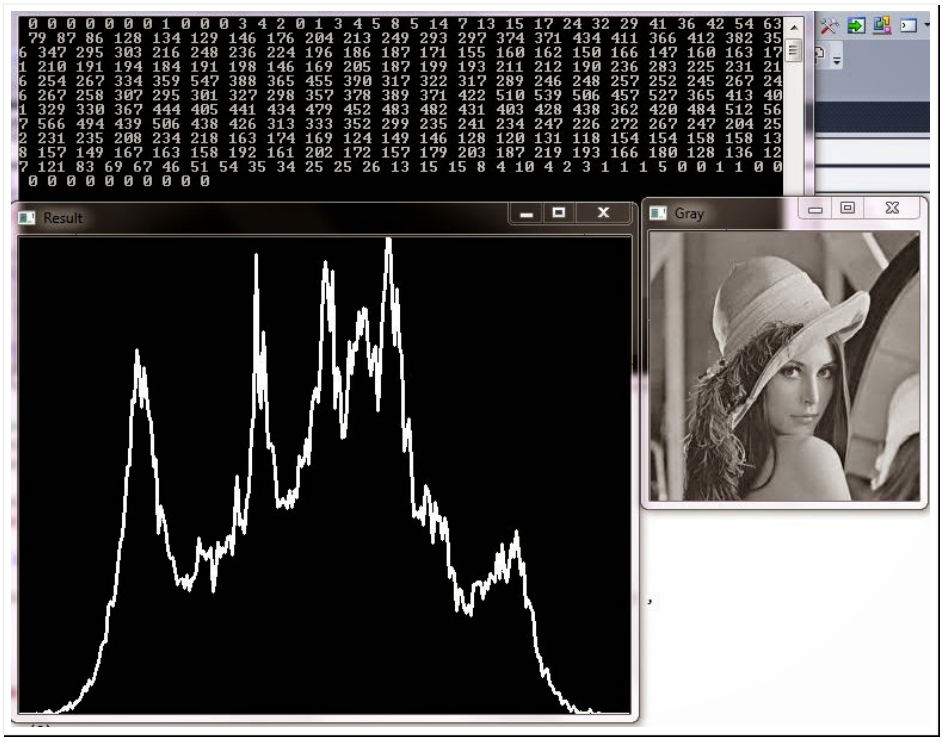
when `normType=NORM_MINMAX` (for dense arrays only). The optional mask specifies a sub-array to be normalized. This means that the norm or min-n-max are calculated over the sub-array and then this sub-array is modified to be normalized.

Example:

Find some more examples in OpenCV documentation. (example 1, example 2)

```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>

using namespace std;
using namespace cv;

int main(int, char**)
{
    Mat gray=imread("image.jpg",0);
    namedWindow( "Gray", 1 );    imshow( "Gray", gray );

    // Initialize parameters
    int histSize = 256;    // bin size
    float range[] = { 0, 255 };
    const float *ranges[] = { range };

    // Calculate histogram
    MatND hist;
    calcHist( &gray, 1, 0, Mat(), hist, 1, &histSize, ranges,true, 1

    // Show the calculated histogram in command window
    double total;
    total = gray.rows * gray.cols;
    for( int h = 0; h < histSize; h++ )
         {
            float binVal = hist.at<float>(h);
            cout<<" "<<binVal;
         }

    // Plot the histogram
    int hist_w = 512; int hist_h = 400;
    int bin_w = cvRound( (double) hist_w/histSize );

    Mat histImage( hist_h, hist_w, CV_8UC1, Scalar( 0,0,0) );
    normalize(hist, hist, 0, histImage.rows, NORM_MINMAX, -1, Mat()

    for( int i = 1; i < histSize; i++ )
    {
      line( histImage, Point( bin_w*(i-1), hist_h - cvRound(hist.at<
                       Point( bin_w*(i), hist_h - cvRound(hist.at<flo
                       Scalar( 255, 0, 0), 2, 8, 0  );
    }

    namedWindow( "Result", 1 );    imshow( "Result", histImage );

    waitKey(0);
    return 0;
}
```

Result:



Labels: Basics

7 comments:

lg_more November 22, 2014 at 11:31 PM

Very useful post! nevertheless, you need to set your range like this float range[] = { 0, 256 }; in order to ge all the bins. I tested counting the pixels you get in the histogram, and I got the right value when I set range like this.

Reply

Birimbau April 2, 2015 at 4:43 PM

This comment has been removed by the author.

Reply

Birimbau April 2, 2015 at 4:45 PM

@lg_more, that's not because the range declaration. That's is because the cycle should be:
for(int i = 0; i < histSize; i++)
OR
for(int i = 1; i <= histSize; i++)

Very useful post indeed! :) thanks!

Reply

Anonymous July 16, 2015 at 7:43 PM

Thanks,
Helpful post!

Ronen

Reply

Đào Quân December 7, 2015 at 10:02 AM

dịch vụ kế toán thuế tại ninh bình
dịch vụ kế toán thuế tại vĩnh phúc
dịch vụ kế toán thuế tại hưng yên
dịch vụ kế toán thuế tại phú thọ
dịch vụ dọn dẹp sổ sách kế toán
dịch vụ dọn dẹp sổ sách kế toán tại thái bình
dịch vụ dọn dẹp sổ sách kế toán tại phú thọ
dịch vụ dọn dẹp sổ sách kế toán tại hưng yên
dịch vụ dọn dẹp sổ sách kế toán tại quận hải dương

dịch vụ dọn dẹp sổ sách kế toán tại hải phòng
dịch vụ dọn dẹp sổ sách kế toán tại quận thanh trì
dịch vụ dọn dẹp sổ sách kế toán tại quận hoàng mai
dịch vụ dọn dẹp sổ sách kế toán tại quận hai bà trưng
dịch vụ dọn dẹp sổ sách kế toán tại quận hoàn kiếm
dịch vụ dọn dẹp sổ sách kế toán tại quận từ liêm
dịch vụ dọn dẹp sổ sách kế toán tại quận ba đình
dịch vụ dọn dẹp sổ sách kế toán tại quận tây hồ
dịch vụ dọn dẹp sổ sách kế toán tại quận đống đa
dịch vụ dọn dẹp sổ sách kế toán tại bắc ninh
dịch vụ dọn dẹp sổ sách kế toán tại quận tphcm
dịch vụ dọn dẹp sổ sách kế toán tại quận cầu giấy
dịch vụ dọn dẹp sổ sách kế toán tại quận long biên
dịch vụ dọn dẹp sổ sách kế toán tại quận hà đông
dịch vụ dọn dẹp sổ sách kế toán tại quận thanh xuân
dich vu hoan thue gtgt

Reply

Tu Hoa Bach May 7, 2016 at 3:09 AM

Dịch vụ kế toán thuế trọn gói hàng đầu giúp doanh nghiệp hiện nay an tâm phát triển bền vững
Uy Tín Nhất- Chuyên Nghiệp Nhất- Giá Rẻ Nhất

Dịch vụ kế toán ACB tư vấn, chia sẻ kinh nghiệm về kế toán thuế cho những ai đang thắc mắc muốn tìm hiểu
dịch vụ kế toán
dich vu ke toan
dịch vụ kế toán thuế uy tín hàng đầu tại TPHCM

Reply

Dave January 23, 2017 at 3:32 PM

Nice example but the links are broken to the images used in the equations

Reply

Enter your comment...

Comment as: Unknown (Goo ▾ Sign out

Publish Preview Notify me

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

If you liked the articles on this blog, sign up for our weekly newsletter and receive a free **"Computer Vision Resources"** guide.

Subscribe Now