Learn OpenCV by Examples

OpenCV simplified for beginners by the use of examples. Learn OpenCV with basic implementation of different algorithms.

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).

SEARCH CONTENTS OF THIS BLOG

Search

POPULAR POSTS

1 Find Contour

2 Basic drawing examples

3 Line Detection by Hough Line Transform

4 Face Detection using Haar-Cascade Classifier

5 Perspective Transform

6 Sobel Edge Detection

- **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

CATEGORIES

$$\|\texttt{dst}\|_{L_p} = \texttt{alpha}$$

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

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

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.

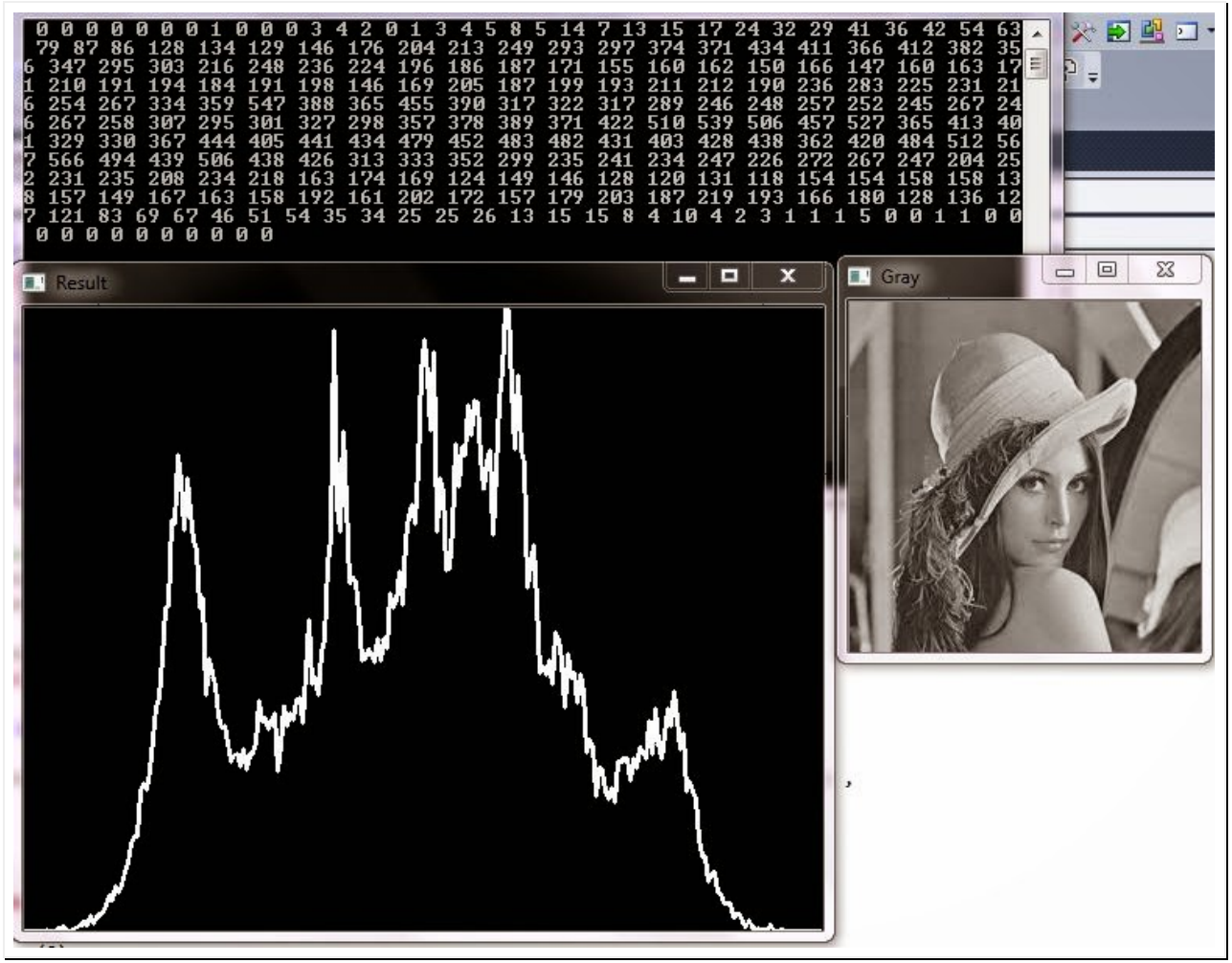
Example:

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,

    // 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()
```

```
38
39      for( int i = 1; i < histSize; i++ )
40      {
41        line( histImage, Point( bin_w*(i-1), hist_h - cvRound(hist.at
42                         Point( bin_w*(i), hist_h - cvRound(hist.at<f
43                         Scalar( 255, 0, 0), 2, 8, 0  );
44      }
45
46      namedWindow( "Result", 1 );    imshow( "Result", histImage );
47
48      waitKey(0);
49      return 0;
50  }
```

Result:



Labels: Basics

3 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