# Histograms

Image processing

## Enumerations

enum **cv::HistCompMethods** {
**cv::HISTCMP_CORREL** = 0,
**cv::HISTCMP_CHISQR** = 1,
**cv::HISTCMP_INTERSECT** = 2,
**cv::HISTCMP_BHATTACHARYYA** = 3,
**cv::HISTCMP_HELLINGER** = HISTCMP_BHATTACHARYYA,
**cv::HISTCMP_CHISQR_ALT** = 4,
**cv::HISTCMP_KL_DIV** = 5
}

## Functions

void **cv::calcBackProject** (const **Mat** *images, int nimages, const int *channels, **InputArray** hist, **OutputArray** backProject, const float **ranges, double scale=1, bool uniform=true)
Calculates the back projection of a histogram. More...

void **cv::calcBackProject** (const **Mat** *images, int nimages, const int *channels, const **SparseMat** &hist, **OutputArray** backProject, const float **ranges, double scale=1, bool uniform=true)

void **cv::calcBackProject** (**InputArrayOfArrays** images, const std::vector< int > &channels, **InputArray** hist, **OutputArray** dst, const std::vector< float > &ranges, double scale)

void **cv::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. More...

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

void **cv::calcHist** (**InputArrayOfArrays** images, const std::vector< int > &channels, **InputArray** mask, **OutputArray** hist, const std::vector< int > &histSize, const std::vector< float > &ranges, bool **accumulate**=false)

double **cv::compareHist** (**InputArray** H1, **InputArray** H2, int method)
Compares two histograms. More...

double **cv::compareHist** (const **SparseMat** &H1, const **SparseMat** &H2, int method)

float **cv::EMD** (**InputArray** signature1, **InputArray** signature2, int distType, **InputArray** cost=**noArray**(), float *lowerBound=0, **OutputArray** flow=**noArray**())
Computes the "minimal work" distance between two weighted point configurations. More...

void **cv::equalizeHist** (**InputArray** src, **OutputArray** dst)
Equalizes the histogram of a grayscale image. More...

## Detailed Description

## Enumeration Type Documentation

### § HistCompMethods

enum **cv::HistCompMethods**

Histogram comparison methods

| Enumerator | |
|---|---|
| HISTCMP_CORREL | Correlation $$d(H_1,H_2)=\frac{\sum_I (H_1(I)-\bar{H_1})(H_2(I)-\bar{H_2})}{\sqrt{\sum_I (H_1(I)-\bar{H_1})^2 \sum_I (H_2(I)-\bar{H_2})^2}}$$ where $$\bar{H_k}=\frac{1}{N}\sum_J H_k(J)$$ and $N$ is a total number of histogram bins. |
| HISTCMP_CHISQR | Chi-Square $$d(H_1,H_2)=\sum_I \frac{(H_1(I)-H_2(I))^2}{H_1(I)}$$ |
| HISTCMP_INTERSECT | Intersection $$d(H_1,H_2)=\sum_I \min(H_1(I),H_2(I))$$ |
| HISTCMP_BHATTACHARYYA | Bhattacharyya distance (In fact, OpenCV computes Hellinger distance, which is related to Bhattacharyya coefficient.) $$d(H_1,H_2)=\sqrt{1-\frac{1}{\sqrt{\bar{H_1}\bar{H_2}N^2}}\sum_I \sqrt{H_1(I)\cdot H_2(I)}}$$ |
| HISTCMP_HELLINGER | Synonym for HISTCMP_BHATTACHARYYA. |
| HISTCMP_CHISQR_ALT | Alternative Chi-Square $$d(H_1,H_2)=2*\sum_I \frac{(H_1(I)-H_2(I))^2}{H_1(I)+H_2(I)}$$ This alternative formula is regularly used for texture comparison. See e.g. **[127]** |
| HISTCMP_KL_DIV | Kullback-Leibler divergence $$d(H_1,H_2)=\sum_I H_1(I)\log\left(\frac{H_1(I)}{H_2(I)}\right)$$ |

## Function Documentation

### § calcBackProject() [1/3]

```
void cv::calcBackProject ( const Mat *    images,
                           int            nimages,
                           const int *    channels,
                           InputArray     hist,
                           OutputArray    backProject,
                           const float ** ranges,
                           double         scale = 1,
                           bool           uniform = true
                         )
```

Calculates the back projection of a histogram.

The function **cv::calcBackProject** calculates the back project of the histogram. That is, similarly to **cv::calcHist** , at each location (x, y) the function collects the values from the selected channels in the input images and finds the corresponding histogram bin. But instead of incrementing it, the function reads the bin value, scales it by scale , and stores in backProject(x,y) . In terms of statistics, the function computes probability of each element value in respect with the empirical probability distribution represented by the histogram. See how, for example, you can find and track a bright-colored object in a scene:

- Before tracking, show the object to the camera so that it covers almost the whole frame. Calculate a hue histogram. The histogram may have strong maximums, corresponding to the dominant colors in the object.
- When tracking, calculate a back projection of a hue plane of each input video frame using that pre-computed histogram. Threshold the back projection to suppress weak colors. It may also make sense to suppress pixels with non-sufficient color saturation and too dark or too bright pixels.
- Find connected components in the resulting picture and choose, for example, the largest component.

This is an approximate algorithm of the CamShift color object tracker.

**Parameters**

| | |
|---|---|
| **images** | Source arrays. They all should have the same depth, CV_8U, CV_16U or CV_32F , and the same size. Each of them can have an arbitrary number of channels. |
| **nimages** | Number of source images. |
| **channels** | The list of channels used to compute the back projection. The number of channels must match the histogram dimensionality. 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. |
| **hist** | Input histogram that can be dense or sparse. |
| **backProject** | Destination back projection array that is a single-channel array of the same size and depth as images[0] . |
| **ranges** | Array of arrays of the histogram bin boundaries in each dimension. See **cv::calcHist** . |
| **scale** | Optional scale factor for the output back projection. |
| **uniform** | Flag indicating whether the histogram is uniform or not (see above). |

**See also**

**cv::calcHist**, **cv::compareHist**

## § calcBackProject() [2/3]

```
void cv::calcBackProject ( const Mat *          images,
                           int                  nimages,
                           const int *          channels,
                           const SparseMat &    hist,
                           OutputArray          backProject,
                           const float **       ranges,
                           double               scale = 1,
                           bool                 uniform = true
                         )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

## § calcBackProject() [3/3]

```
void cv::calcBackProject ( InputArrayOfArrays          images,
                           const std::vector< int > &  channels,
                           InputArray                  hist,
                           OutputArray                 dst,
                           const std::vector< float > &  ranges,
                           double                      scale
                         )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

## § calcHist() [1/3]

```
void cv::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.

The function **cv::calcHist** calculates the histogram of one or more arrays. The elements of a tuple used to increment a histogram bin are taken from the corresponding input arrays at the same location. The sample below shows how to compute a 2D Hue-Saturation histogram for a color image. :

```
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>

using namespace cv;

int main( int argc, char** argv )
{
    Mat src, hsv;
    if( argc != 2 || !(src=imread(argv[1], 1)).data )
        return -1;

    cvtColor(src, hsv, COLOR_BGR2HSV);

    // Quantize the hue to 30 levels
    // and the saturation to 32 levels
    int hbins = 30, sbins = 32;
    int histSize[] = {hbins, sbins};
    // hue varies from 0 to 179, see cvtColor
    float hranges[] = { 0, 180 };
    // saturation varies from 0 (black-gray-white) to
    // 255 (pure spectrum color)
    float sranges[] = { 0, 256 };
    const float* ranges[] = { hranges, sranges };
    MatND hist;
    // we compute the histogram from the 0-th and 1-st channels
    int channels[] = {0, 1};

    calcHist( &hsv, 1, channels, Mat(), // do not use mask
             hist, 2, histSize, ranges,
             true, // the histogram is uniform
             false );
    double maxVal=0;
    minMaxLoc(hist, 0, &maxVal, 0, 0);

    int scale = 10;
    Mat histImg = Mat::zeros(sbins*scale, hbins*10, CV_8UC3);

    for( int h = 0; h < hbins; h++ )
        for( int s = 0; s < sbins; s++ )
        {
            float binVal = hist.at<float>(h, s);
            int intensity = cvRound(binVal*255/maxVal);
            rectangle( histImg, Point(h*scale, s*scale),
                        Point( (h+1)*scale - 1, (s+1)*scale - 1),
                        Scalar::all(intensity),
                        CV_FILLED );
        }

    namedWindow( "Source", 1 );
    imshow( "Source", src );

    namedWindow( "H-S Histogram", 1 );
    imshow( "H-S Histogram", histImg );
    waitKey();
}
```

**Parameters**

| | |
|---|---|
| **images** | Source arrays. They all should have the same depth, CV_8U, CV_16U 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: $L_0, U_0 = L_1, U_1 = L_2, \ldots, U_{\texttt{histSize[i]}-2} = L_{\texttt{histSize[i]}-1}, U_{\texttt{histSize[i]}-1}$ . 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. |

**Examples:**

**demhist.cpp**.

## § calcHist() [2/3]

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

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

this variant uses **cv::SparseMat** for output

## § calcHist() [3/3]

```
void cv::calcHist ( InputArrayOfArrays        images,
                    const std::vector< int > &  channels,
                    InputArray                  mask,
                    OutputArray                 hist,
                    const std::vector< int > &  histSize,
                    const std::vector< float > &  ranges,
                    bool                        accumulate = false
                  )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

## § compareHist() [1/2]

```
double cv::compareHist ( InputArray H1,
                         InputArray H2,
                         int        method
                       )
```

Compares two histograms.

The function **cv::compareHist** compares two dense or two sparse histograms using the specified method.

The function returns $d(H_1, H_2)$ .

While the function works well with 1-, 2-, 3-dimensional dense histograms, it may not be suitable for high-dimensional sparse histograms. In such histograms, because of aliasing and sampling problems, the coordinates of non-zero histogram bins can slightly shift. To compare such histograms or more general sparse configurations of weighted points, consider using the **cv::EMD** function.

**Parameters**

| | |
|---|---|
| **H1** | First compared histogram. |
| **H2** | Second compared histogram of the same size as H1 . |
| **method** | Comparison method, see **cv::HistCompMethods** |

## § compareHist() [2/2]

```
double cv::compareHist ( const SparseMat & H1,
                         const SparseMat & H2,
                         int               method
                       )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

## § EMD()

```
float cv::EMD ( InputArray  signature1,
                InputArray  signature2,
                int         distType,
                            cost =
                InputArray  noArray(),
                float *     lowerBound = 0,
                OutputArray flow = noArray()
              )
```

Computes the "minimal work" distance between two weighted point configurations.

The function computes the earth mover distance and/or a lower boundary of the distance between the two weighted point configurations. One of the applications described in **[135]**, **[136]** is multi-dimensional histogram comparison for image retrieval. EMD is a transportation problem that is solved using some modification of a simplex algorithm, thus the complexity is exponential in the worst case, though, on average it is much faster. In the case of a real metric the lower boundary can be calculated even faster (using linear-time algorithm) and it can be used to determine roughly whether the two signatures are far enough so that they cannot relate to the same object.

**Parameters**

| | |
|---|---|
| **signature1** | First signature, a $\texttt{size1} \times \texttt{dims} + 1$ floating-point matrix. Each row stores the point weight followed by the point coordinates. The matrix is allowed to have a single column (weights only) if the user-defined cost matrix is used. The weights must be non-negative and have at least one non-zero value. |
| **signature2** | Second signature of the same format as signature1 , though the number of rows may be different. The total weights may be different. In this case an extra "dummy" point is added to either signature1 or signature2. The weights must be non-negative and have at least one non-zero value. |
| **distType** | Used metric. See **cv::DistanceTypes**. |
| **cost** | User-defined $\texttt{size1} \times \texttt{size2}$ cost matrix. Also, if a cost matrix is used, lower boundary lowerBound cannot be calculated because it needs a metric function. |
| **lowerBound** | Optional input/output parameter: lower boundary of a distance between the two signatures that is a distance between mass centers. The lower boundary may not be calculated if the user-defined cost matrix is used, the total weights of point configurations are not equal, or if the signatures consist of weights only (the signature matrices have a single column). You must** initialize *lowerBound . If the calculated distance between mass centers is greater or equal to *lowerBound (it means that the signatures are far enough), the function does not calculate EMD. In any case *lowerBound is set to the calculated distance between mass centers on return. Thus, if you want to calculate both distance between mass centers and EMD, *lowerBound should be set to 0. |
| **flow** | Resultant $\texttt{size1} \times \texttt{size2}$ flow matrix: $\texttt{flow}_{i,j}$ is a flow from $i$ -th point of signature1 to $j$ -th point of signature2 . |

## § equalizeHist()

void cv::equalizeHist ( **InputArray** src,
**OutputArray** dst
)

Equalizes the histogram of a grayscale image.

The function equalizes the histogram of the input image using the following algorithm:

- Calculate the histogram $H$ for src .
- Normalize the histogram so that the sum of histogram bins is 255.
- Compute the integral of the histogram:

$$H'_i = \sum_{0 \le j < i} H(j)$$

- Transform the image using $H'$ as a look-up table: $\texttt{dst}(x,y) = H'(\texttt{src}(x,y))$

The algorithm normalizes the brightness and increases the contrast of the image.

**Parameters**

**src** Source 8-bit single channel image.

**dst** Destination image of the same size and type as src .