## Table of Contents

## Image Histograms

The distribution of pixel values within an image is an important characteristic of that image. By manipulating an image histogram, we can change its appearance, such as improving contrast, etc. Additionally, it is possible to detect specific textures and objects in images by matching histograms, among other possibilities.

### Computing an Image Histogram

A grayscale image is typically made up of pixels that have values between zero and 255. Some tones of gray may be more represented than others in an image, often leading to poor visual quality.

Suppose we are interested in improving contrast in a grayscale image. How would we solve this problem? A first intuition would be to look at the distribution of pixels in terms of their values. Then we would be able to detect if the gray tones distribute narrowly (poor contrast) or not. If the distribution is narrow, then we may want to attempt to stretch it within the range of possible values.

An image histogram is composed of bins, each representing a gray tone in the image. Hence a histogram for a grayscale image has 256 bins. Each one of these bins contains the number of pixels in the image that has the corresponding gray tone. The OpenCV `calcHist` function performs this operation.

If the image has color, then three channels are present (RGB) and they can be separated from each other using the function `split`. Then each channel can be histogrammed individually. Alternatively, a 3D histogram could be computed by the function if desired.

### Look-Up Tables

By using the information contained in a histogram, it is possible to improve the image. For this purpose we may use a mapping function in the form of a look up table which tells us where pixels in the current image should map in the improved image in terms of gray scale.

A look-up table is a function that defines how pixel values are transformed into new values. It is a 1D array with 256 entries (in the case of grayscale images). The $i^{th}$ entry of the table gives the new intensity value of the corresponding gray level in the following way:

```
newIntensity = lookup[oldIntensity] ;
```

## Histogram Equalization

If we could make the histogram of an image as flat as possible and then recompute the gray values of this image according this new histogram, we certainly would enhance contrast.

The way to do it is to compute a lookup table that would perform this transformation. Let's use a simple example to illustrate the equalization algorithm. Suppose we have 8 possible gray values (from 0 to 7) and a 5 by 5 image such as this one:

| | | | | |
|---|---|---|---|---|
| 4 | 4 | 4 | 4 | 4 |
| 3 | 4 | 5 | 4 | 3 |
| 3 | 5 | 5 | 5 | 3 |
| 3 | 4 | 5 | 4 | 3 |
| 4 | 4 | 4 | 4 | 4 |

We can now compute the histogram of this image, which is just counting the number of pixels per existing gray value:

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $h(i)$ | 0 | 0 | 0 | 6 | 14 | 5 | 0 | 0 |

And also obtain the cumulative frequency distribution:

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $h(i)$ | 0 | 0 | 0 | 6 | 14 | 5 | 0 | 0 |
| $h_c(i)$ | 0 | 0 | 0 | 6 | 20 | 25 | 0 | 0 |

From this, we want to derive a lookup function that transforms $H_c(i)$ into its ideal form which is a straight ramp, just as it would be the case if the gray values were evenly distributed over the image. In this case each bin should contain a number of pixels equal to the total number of pixels in the image divided by the possible gray values. In other words, $\frac{25}{8}=3.125$ pixels. We are dealing with integer numbers of pixels, and so we will add one somewhere to make things work:

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $h(i)$ | 0 | 0 | 0 | 6 | 14 | 5 | 0 | 0 |
| $h_c(i)$ | 0 | 0 | 0 | 6 | 20 | 25 | 0 | 0 |
| $h_{ce}(i)$ | 3 | 6 | 9 | 13 | 16 | 19 | 22 | 25 |

Thus, $h_{ce}(i)$ is the histogram we would obtain if the gray values in the image were perfectly distributed over the range of possible gray values.

The next step is to design the mapping (or lookup table) that equalizes the image histogram. The steps are the following:

- For each possible gray value $i$ , find the corresponding value in $H_c(i)$ . For instance, in the case where $i=4$ , the corresponding value in $h_c(i)$ is 20.
- Find the closest value to this $h_c(i)$ in $h_{ce}(i)$ . Keeping with our example, this value turns out to be 19.
- The $i$ value that corresponds to this $h_{ce}(i)$ is the mapping for the gray value $i$ from the first step.

Using this method, we can construct the lookup table, one mapping at a time. This would result in:

| $i$ | LookUp $(i)$ |
|---|---|
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 1 |
| 4 | 5 |
| 5 | 7 |
| 6 | 7 |
| 7 | 7 |

With this mapping, our small image gets transformed into this:

| | | | | |
|---|---|---|---|---|
| 5 | 5 | 5 | 5 | 5 |
| 1 | 5 | 7 | 5 | 1 |
| 1 | 7 | 7 | 7 | 1 |
| 1 | 5 | 7 | 5 | 1 |
| 5 | 5 | 5 | 5 | 5 |

Because this is an integer mapping, the attentive reader will notice that the three initial non-zero bins of the input image can only map to three possibly different output bins. Indeed, on what criteria could we divide a single input bin into more than one?

Now, we can compute the new histogram for the transformed image:

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $h(i)$ | 0 | 0 | 0 | 6 | 14 | 5 | 0 | 0 |
| $h_c(i)$ | 0 | 0 | 0 | 6 | 20 | 25 | 0 | 0 |
| $h_{ce}(i)$ | 3 | 6 | 9 | 13 | 16 | 19 | 22 | 25 |
| $h'(i)$ | 0 | 6 | 0 | 0 | 0 | 14 | 0 | 5 |

The new cumulative distribution function can also be directly computed as:

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $h(i)$ | 0 | 0 | 0 | 6 | 14 | 5 | 0 | 0 |
| $h_c(i)$ | 0 | 0 | 0 | 6 | 20 | 25 | 0 | 0 |
| $h_{ce}(i)$ | 3 | 6 | 9 | 13 | 16 | 19 | 22 | 25 |
| $h'(i)$ | 0 | 6 | 0 | 0 | 0 | 14 | 0 | 5 |
| $h'_c(i)$ | 0 | 6 | 6 | 6 | 6 | 20 | 20 | 25 |

In general, once an image is captured by a sensor, there is little we can change with the way the sensor recorded the pixels values, and the unprocessed result may be unsatisfactory. Dark areas within the visual extent require longer exposure time while bright areas require less. But the exposure time is the same for all image areas. Fortunately, histogram equalization provides a partial solution to problems like these.

The process of equalization can be performed in OpenCV with a call to a single function known as `cvEqualHist()`. The function can only deal with images with no more more than one channel.

## A Simple Histogram Transform Example

It is instructive to perform histogram equalization on images that display poor contrast, as the results are usually visually impressive. A simple program performs this transform:

```
#include <stdio.h>

#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv/cvaux.h>

int main(int argc, char** argv) {

  IplImage *LCimage ;
  IplImage *HCimage ;

  LCimage = cvLoadImage(argv[1],CV_LOAD_IMAGE_GRAYSCALE) ;
  HCimage = cvCreateImage(cvGetSize(LCimage),IPL_DEPTH_8U,1) ;

  cvEqualizeHist(LCimage,HCimage) ;
  cvNamedWindow("Low Contrast Image",CV_WINDOW_AUTOSIZE) ;
  cvShowImage("Low Contrasst Image",LCimage) ;

  cvNamedWindow("High Contrast Image",CV_WINDOW_AUTOSIZE) ;
  cvShowImage("High Contrast Image",HCimage) ;

  cvWaitKey(0) ;

  cvReleaseImage(&LCimage) ;
  cvReleaseImage(&HCimage) ;

  cvDestroyAllWindows() ;
}
```

The input satellite image and histogram equalized output image are displayed below, from left to right: