Contrast Limited Adaptive Histogram Equalization

Shane Snover
CSCI 631 Foundations of Computer Vision
Course Project

Abstract

MATLAB is a great prototyping tool for development of imaging processing and computer vision applications, but it's quite unsuitable for deployment in a larger application with many components outside of MATLAB's ecosystem or in embedded systems where there are constraints on power consumption and processes must run in deterministic time. C++ is a low-level compiled programming language which can be compiled to run on many different processor architectures and for FPGAs using high level synthesis. As a computationally intensive image processing algorithm, adaptive histogram equalization is implemented in C++ and uses the language's flexibility to allow the user to set the gray level mapping function of their choice. The implementation is then compared for performance against MATLAB and OpenCV's implementations.

Introduction

Many applications involve imaging a medium and returning an image which must be interpreted by a human. The process requires the computer to take an image of something which a human is unable to see, whether due to the wavelength of the light, the scale, or for other reasons and output an image which makes relevant features in an image identifiable to a human being. This problem arises in electronics manufacturing when a technician needs to inspect a printed circuit board to validate it's quality or in biomedical imaging when a doctor needs to review a computed tomography scan.

Computers have many advantages over the human visual system for imaging a surface. Artificial sensors may be designed to capture light over vastly different regions of the electromagnetic spectrum or even to be sensitive to very small regions of the spectrum whereby small differences can be acutely discovered. A frequent challenge is how to enhance the abilities of a human visual system through computer vision. A representation of an image visible only to a machine must be manipulated to be meaningful to the human brain so that features in the image may be teased out by the sophisticated algorithms in the brain which match features to known models and identify significance.

The human visual system is very sensitive to contrast in the image, where the intensity of a color or grayscale channel in a scene sees a large change over a small distance. Image enhancement techniques which enhance contrast in a select region make small differences in intensity between pixels larger in order for those differences to be more obvious to the human eye. Many simple methods which act on a per pixel scale have been developed such as gamma correction or exponential transforms.

A more computationally expensive approach which acts on entire regions of an image – and as such can act that much more intelligently – is histogram equalization. Histogram equalization takes statistics on the frequency of every level of intensity in an image and attempts to map these pixels to new intensity values. This is frequently done on images where most of the pixel intensities concentrate over a small range. In these cases, there is likely low contrast in the features of the image and any features in the image would be difficult to pick out. This can occur in photographs where a large object obscures a light source causing a shadow to appear over the scene. Objects in this shadow would have less light reflecting from their surface back to the camera's sensor which causes low contrast on surfaces under this shadow. The contrast can be increased by making each level of intensity in this region of the image more spread out. Histogram equalization and the algorithm's various permutations provide a mathematical way to describe how the intensity of each pixel should change.

Research

Adaptive histogram equalization was originally developed as an image enhancement technique for use in the cockpits of military aircraft by David Ketcham, Roger Lowe, and William Weber who worked in the Display Systems Laboratory for the Hughes Aircraft Company [1]. The goal of the algorithm was to overcome two problems in the imaging systems which made it more difficult for the aircraft's operator to detect features on the display in the cockpit. The first was that the dynamic range of the display was often lower than some imaging sensors, particularly infrared imagery in their case. The second problem was that the slow response time of some sensors caused blurring of edges in the displayed image. By increasing contrast in these images, features could be more readily identified by the pilot of the aircraft.

Adaptive histogram equalization makes strides forward from the more simple form of histogram equalization due to splitting the input image up into multiple regions which have their histograms mapped to new histograms individually. This change makes the method much more computationally expensive, but achieves superior results in images where the global histogram of the image is relatively well distributed, but smaller regions of the image have low contrast. In their work, Ketcham, et al., noted that in an image where there was a relatively high concentration of bright pixels and of very dark pixels, but with few in the medium gray range, that splitting the image up into equal bins across the range for display on the screen did not accurately show the amount of information contained at each level of intensity. Their goal was for their adaptive algorithm to quantize the gray levels in a higher resolution input image such that when it was displayed on the lower resolution CRT display that each histogram bin would have approximately the same frequency.

They achieved this by use of a sliding window and experimented with various sizes of this window in order to optimize their computation time while still avoiding problems with the global histogram equalization. By taking a histogram of only a small sample of pixels and applying equalization based on their distribution, it correctly took into account the amount of information represented at each gray level, but taking these statistics was time consuming and their application was for real-time imaging systems.

In order to improve the computation time, a group working on biomedical images at the University of North Carolina Chapel Hill lead by Stephen Pizer developed a change to the original adaptive histogram equalization method which allowed for faster computation time by doing away with the sliding window and using interpolation [2]. Instead of recomputing a series of histograms for overlapping windows of the image, the image was instead split up into defined tiles and the statistics of gray level distribution were taken for each one. This was not done before over the sliding window because taking non-overlapping windows caused edges to appear at the boundaries of the windows. In order to overcome this, the group working at Chapel Hill used bilinear interpolation. A pixel in the image was mapped using the gray level mapping of the four closest tiles and then the final gray level was interpolated based on its distance to those tiles' centers. This removed the edges being created at the boundary between tiles. A frequency limit was also imposed on each tile's histogram in order limit the algorithm's tendency to amplify noise in the tile. This final form of the algorithm is called contrast-limited adaptive histogram equalization (CLAHE).

Methodology

The C++ implementation leverages OpenCV for reading and writing images from the filesystem to a data structure in memory for manipulation. OpenCV implements an interface very similar to MATLAB for reading images into a matrix structure where individual pixels can be accessed just by their xy-coordinates. All of the functionality for the CLAHE algorithm written for this project was written from scratch.

First, the image is passed as a *cv::Mat* object and the width and height of the tiles is calculated for use throughout the rest of the algorithm. For this implementation, the number of tiles in both the vertical and horizontal directions are fixed to eight, leaving a total of 64 tiles across the image. When the image's dimensions were not perfect integer multiples of the tile dimensions, the remaining rows and columns of the image are considered part of the closest tile.

A lookup table for each tile is stored in a 3-dimensional array where the first two dimensions are the tile coordinates and the third dimension is the gray level intensity of the input image. Next for each tile the histogram is computed by iterating over each pixel in the tile's subregion of the image and incrementing a corresponding bin's frequency over the whole range of gray levels in the image. This algorithm assumes inputs are 8-bit resolution so there are 256 possible bins. After the actual histogram is computed, the histogram is clipped so that all pixels above a certain frequency are "redistributed". This clipping limit is specified by the caller of the function as its effectiveness can be dependent on the size of the image or how much noise is in the input image, but if the caller does not supply a clipping limit it is defaulted to a value of 40.

The frequencies of each bin are set to the limit and the excess is accumulated. Then this excess is divided among every bin in the histogram.

With the histogram of the tile computed, the gray level mapping can be performed in order to populate the lookup table. The CLAHE function is implemented such that the caller can supply a function which will be called to perform the gray level mapping. This function takes the histogram as a data structure defined by the module and a pointer to the tile's lookup table as an array. If the user does not supply a gray level mapping function, the function falls back on a default which behaves like the method originally described in Ketcham, et al [1] whereby it tries to bring the image's cumulative distribution function closer to a slope of 1. This method was also used in OpenCV's implementation [4]. For each bin, the number of pixels in all of the bins thus far is accumulated and taken as a ratio of the total pixels is multiplied by the number of bins to find the new bin for the pixels to go to. For example, if the image has a high concentration of pixels at low gray levels such that as bin 63 is reached one half of the pixels in the image are accounted for, the gray level mapping for an input intensity of 63 will 255 / 2 = 127.

The gray level mapping described above can be more succinctly described with Equation 1 below. In the equation $h_{i,j}(k)$ is the histogram function of the tile where for an input intensity k the function returns the frequency of that intensity in the image, M is the number of pixels in the tile, and N is the number of grayscales.

$$f_{i,j}(n) = \frac{(N-1)}{M} * \sum_{k=0}^{n} h_{i,j}(k);\ n = 0,\ 1,\ 2, \ldots,\ N-1$$

Equation 1. Standard Gray Level Mapping for CLAHE

With the lookup table for each tile in the image populated, the histogram transformation for the entire image can be performed. Each tile in the image is divided into four quadrants. For each quadrant in the image, it can be classified as existing in one of three types of regions. There are four corner regions (CR) in the image which are the rectangular subsections of the image beyond the corner most tile-center. In the top left corner, all pixels above and to the left of the tile (0, 0) are in the corner region. In the bottom right, all pixels with greater x and y coordinates than tile (7, 7) are in a corner region. This extends to the other two corners. Along the edge of the image between the four corner images are the border regions (BR) which exist in the quadrants directly adjacent to the image border, but before the first tile center pixel encountered as distance from the center decreases. All remaining regions are interior regions (IR). This is illustrated well with a figure from a paper by Ali M. Reza on implementation of CLAHE [3] below (Figure 1).

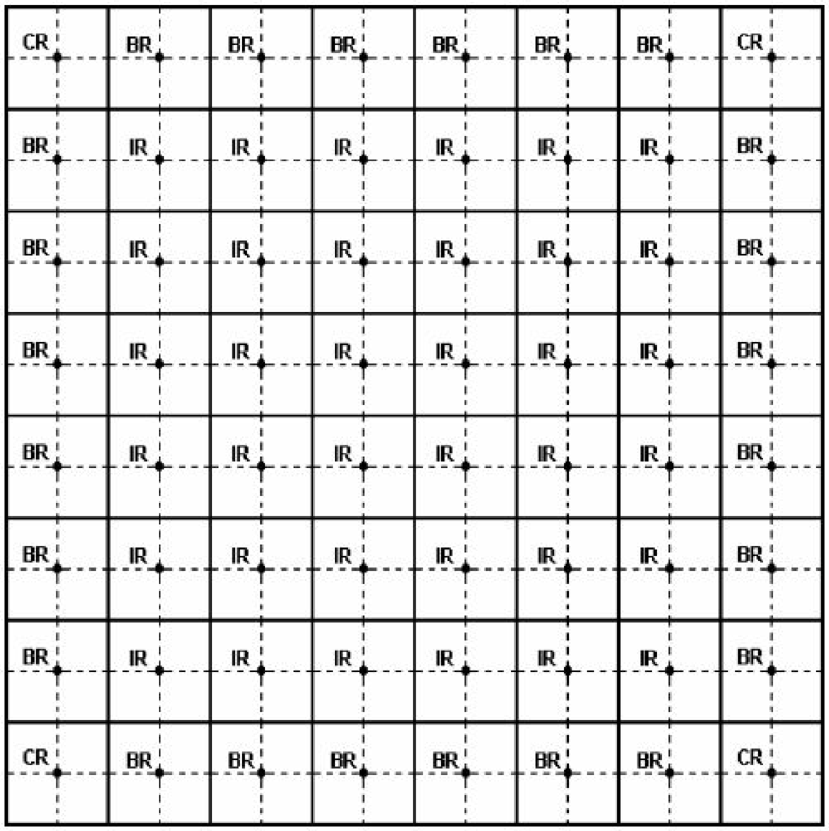

Figure 1. Classification of Regions of Pixels in Image

The classification of each pixel determines how the output image's pixel is mapped from the input image pixel intensity. In the case of corner regions, the pixel is simply plugged into the lookup table for the current tile and the resulting value is written in position to the output. For border regions, the input intensity is plugged into the lookup tables of the two closest tiles and a new value is interpolated based on the distance along the x-axis for border regions on the top and bottom or along the y-axis for border regions on the left and right of the image. Finally, all interior regions are interpolated similarly to the border regions, but with the output of the four closest tiles' lookup tables. The interpolation in the border and interior regions ensures that pixels on either side of an edge between tiles have a smooth change in intensity and there are no resulting new edges added from the process.

This concludes the discussion of the implementation of the CLAHE algorithm. After this series of steps a new pixel intensity value is calculated for each pixel in the final image and the image is returned to the caller as a *cv::Mat*.

Learnings

Adaptive histogram equalization is a well known algorithm and image enhancement technique. Despite originally being implemented in it's most common form as far back as 1987, there is a lot of ongoing work on tweaking this technique to fit various applications. There is a lot of material published on either speeding up the algorithm by modifying it to run as a hardware realization on field-programmable gate arrays (FPGAs) in order to allow it to enhance high resolution images in near real-time or to alter a mapping function to change the way it enhances different features or to allow it to reject noise in the image during its grayscale amplification.

There is a common opinion that almost all problems in computer vision can be solved by throwing a graphics processor and a neural network or deep learning at it. This approach requires a lot of data however, or it will not be reach accuracy rates required for manufacturing where false-positives can dip into profit margins. On the other hand the error rate makes these techniques unsuitable for biomedical imaging due to the cost of an error, whether that be a high monetary cost treating a false-positive or the cost of a life. Part of the issue is that it can be difficult to get training data for these systems, as it takes a highly trained individual to detect issues in these images. This will become easier as more specialized variations of CLAHE make it easier to detect by humans. Until this comes to pass, this technique will serve as an aid to human specialists to make them better at their job.

As for the actual implementation of the algorithm in C++, the advantages of a flexible interpreted environment like MATLAB became obvious as issues with managing memory and properly moving ownership of that memory between functions caused headaches or slowdowns in the runtime of the program. It was rewarding to see the results of the implementation of a more complex image processing algorithm implemented at the lowest level as the input images and their histograms were compared with the outputs and the new resulting histogram.

In testing the algorithm, a limitation in the implementation became apparent in running CLAHE on larger images. As the image's size increases, so does the size of each tile. This results in the clipping of the histogram having an outsized effect. A relatively small clipping limit during the lookup table generation can cause almost all bins to be clipped causing a nearly flat histogram. With a nearly flat histogram, almost all pixel intensities end up being mapped back onto themselves in the lookup table and the resulting image ends up almost unchanged. A future improvement would be to dynamically size the default clipping limit with the size of the image to prevent this.

Results

The implemented adaptive histogram equalization appears to increase contrast in images with concentrated histograms quite well. Here is an image side-by-side with the output of the implemented CLAHE function.

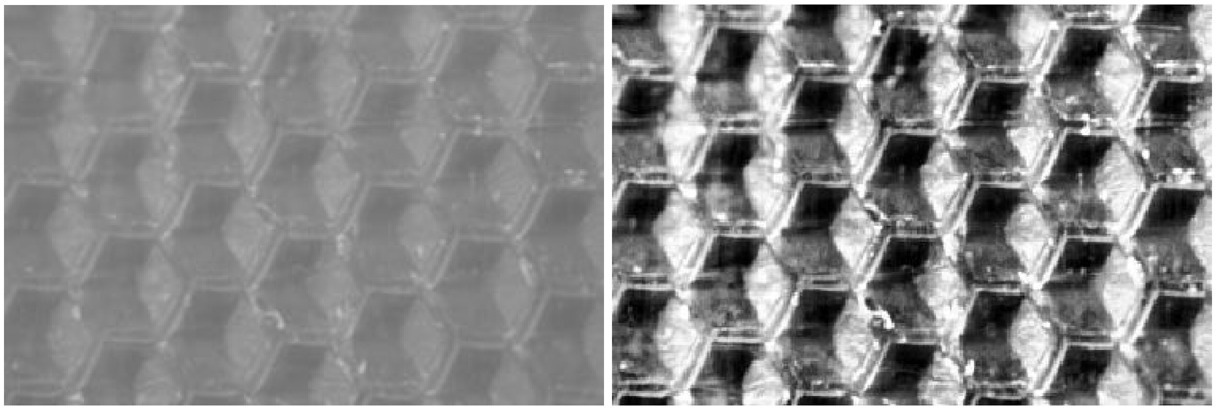

Figure 2. Honeycomb.png Before and After CLAHE

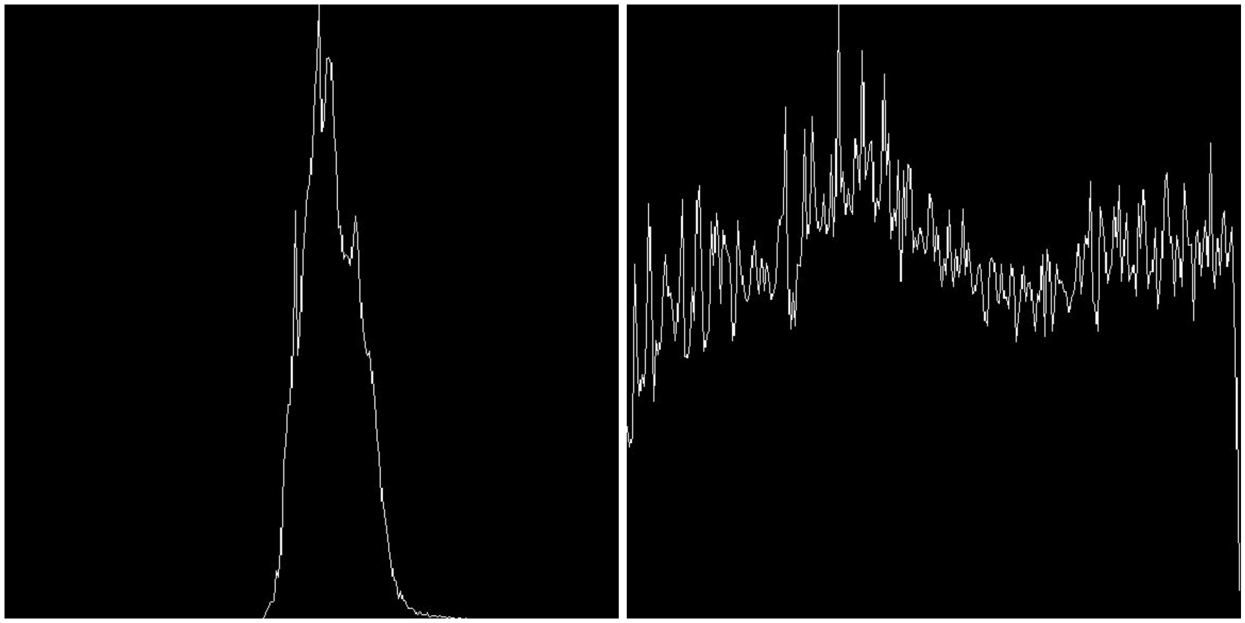

Figure 3. Histograms of Image Before and After CLAHE

The image *honeycomb.png* was run through the function as an example of an image with very low contrast and a histogram with a concentrated distribution. It is clear from the histogram in Figure 3a that all of the information in the image is concentrated in a small range of gray levels. This histogram is essentially expanded over the whole range of the 8-bit grayscale to heavily increase the contrast of the output image. This was run with the default clipping limit of 40.

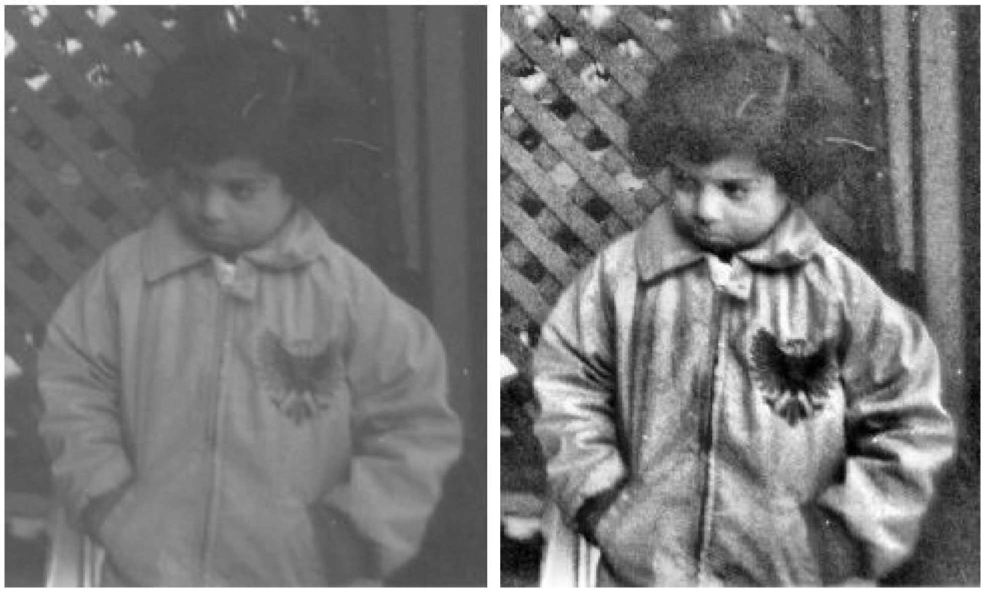

Figure 4. MATLAB's pout.tif Before and After CLAHE

In Figure 4, an image included with MATLAB's Image Processing Toolbox is shown after transformation through the CLAHE function with a clipping limit set at 30. Here again the increased contrast makes especially visible features which are more hidden in shadow in the original image. The effect of the noise amplification begins to show here as well with parts of the image have a slightly fuzzy appearance.

The performance of this CLAHE implementation, OpenCV's *clahe*, and MATLAB's *adapthisteq* function were run on the *honeycomb.png* image from Figure 2 since this could be done with the default clipping limit in OpenCV and MATLAB's clipping limit can be configured to be the same. Each function was run ten times and the average was taken for the running time. The Snover implementation and OpenCV 4.0 were built in release mode which results in the flag "-O3" being passed to GCC. The results in the runtime are provided in Table 1 below.

Implementation	Snover (-O3)	OpenCV (-O3)	MATLAB
Runtime (ms)	5.8	1.6	32.3

Table 1. Comparison of CLAHE Runtimes on honeycomb.png

The results of the performance comparison highlight MATLAB's role as a great prototyping platform, but not as useful for real-time applications. While the resulting algorithm could be run in MATLAB on 30 frames per second video assuming not much other processing was required of the image, the C++ implementations scale well beyond that and ran an order of magnitude more quickly. A quick check of runtimes in Debug mode of the Snover implementation with "-O0" being passed resulted in runtimes which were on the same order of magnitude as MATLAB's function.

Conclusion

Contrast limited adaptive histogram equalization is a classical image processing technique for enhancing the contrast in the luminosity channel of images. The algorithm has been implemented in many different contexts to make an image more useful to a human interpreter which must find relevant information in the image such that a defect can be found in manufacturing or a malignant tumor can be spotted in a CT scan.

After studying some of the larger papers detailing the process for implementing adaptive histogram equalization and why each step came to be, the source code for OpenCV's version of the algorithm was stepped through to further understand how different aspects of the algorithm could be realized as data structures and where steps could be sped up on a GPU. There were many areas discovered in OpenCV's code which were made to be highly customizable such that it could be easily used as a library function for a wide variety of use cases including images of different grayscale resolutions, or massaging the data into a form where it could easily be used in a parallelized form on GPUs. By constraining the possible inputs to the function, many of these distractions were taken away in order to further speed up the operation on images of interest: grayscale images with 256 gray levels. The result is an implementation which, while a little more specialized than OpenCV's, can be run easily by the user and can serve as a well documented reference for individuals who wish to implement the algorithm for themselves for a different set of constraints.

The algorithm itself was sufficiently challenging in complexity to require learning the low level details of a number of sub-problems and how to run them efficiently to limit the amount of memory required for the program to run. While components like decoding and encoding images to a file and showing them on the screen were glossed over for the benefit of working strictly on the CLAHE algorithm, many components needed to be implemented from scratch before work on the actual image processing technique of concern could be started. This included splitting the image up into smaller regions, interpolating between both 2 and 4 points, and taking different statistical computations of regions of the image. While MATLAB provides some very convenient interfaces that allows the programmer to not have to be concerned about these low level details, it is immensely satisfying to have a high degree of knowledge about how every line in the source actually works with very few abstractions in the way. This knowledge was especially helpful during debugging when a bug would be noticeable in the final image and the problem could be traced to what would have been an abstraction in MATLAB, but was new code written for this project and could be tinkered with instead of viewed as a black box with documentation. Finally, it was great to see the CLAHE implementation outrun the MATLAB function by an order of magnitude.

There are definitely more areas where the runtime performance of the algorithm could be improved and more gray level mapping functions which could be implemented and experimented with for various image types. The hope is that this project will serve as a reference for those who wish to extend CLAHE further in this regard. This project required the writing of many utility functions which can be used going forward to continue implementation of other image processing techniques using the power of a low-level compiled language.

Bibliography

1. Ketcham, D. J., Lowe, R. W., & Weber, J. W. (1976). Image Enhancement Techniques for Cockpit Displays. Ft. Belvoir: Defense Technical Information Center.
2. Pizer, S. M. (1987). Adaptive histogram equalization and its variations. Computer Vision, Graphics and Image Processing,39(3), 355-368. Retrieved November 17, 2018.
3. Reza, A. M. (1997). Realization of the Contrast Limited Adaptive Histogram Equalization (CLAHE) for real-time image enhancement. Journal of VLSI Signal Processing,38(1), 35-44. Retrieved November 17, 2018.
4. NVIDIA Corporation, Itseez Inc. (2014) OpenCV CLAHE source code (version 4.0.0) [Source code]. https://github.com/opencv/opencv/blob/master/modules/imgproc/src/clahe.cpp