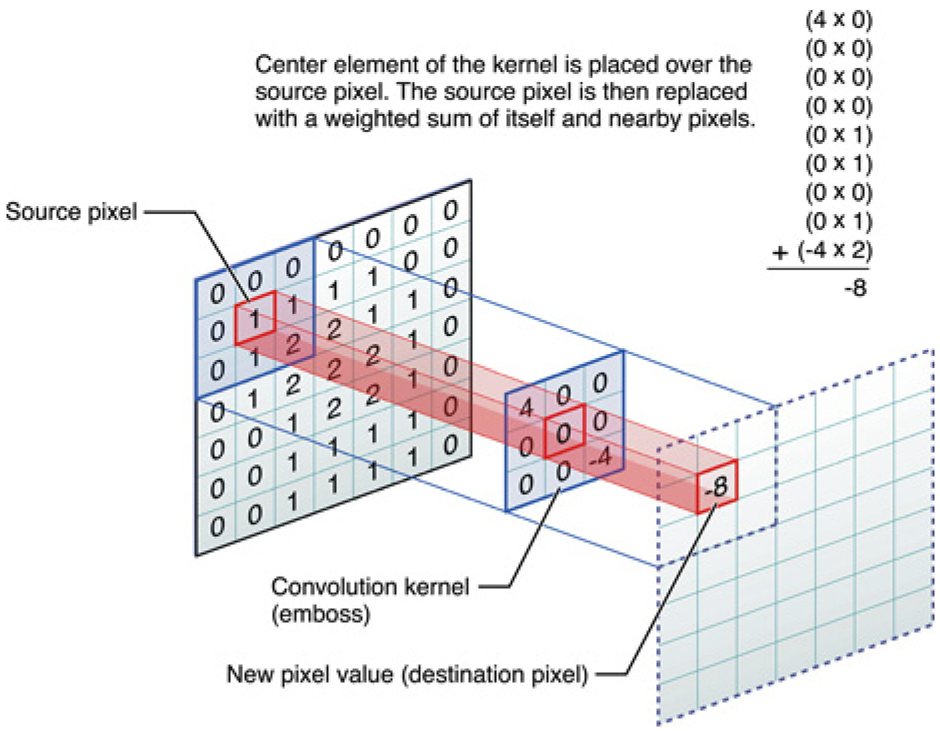
IMAGE ENHANCEMENT TECHNIQUES USING OPENCV AND NUMPY

By: M Nikitha
Rakshit S Rudragoudar

BASIC CONCEPTS:

Convolution:

1. It is an operation in image processing


Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution kernel (emboss)
New pixel value (destination pixel)

2. Input and output matrix are of same dimensions.
3. Mathematical operation is applied to each pixel to change its value in some way.
4. To apply this mathematical operator Kernel(matrix) is used.
5. Kernel is fixed with its center on each pixel and corresponding pixels are multiplied.
6. The pixel value is replaced with sum of all multiplication.

COMPUTER VISION TERMINOLOGY:

1. Kernel: image filter
2. Image filtering: applying kernel to given image
3. Filtered image: output obtained after applying kernel to the image
4. Frequency: rate of change of pixel values

5. Low pass filter/kernel: attenuates the high frequency components and preserves low frequency components.
6. High pass filter/kernel: attenuates low frequency components and preserves high frequency components.

IMAGE ENHANCEMENT TECHNIQUES:

1. Blurring

   Types of blurring:

   - Kernel blurring:

$$\texttt{dst}(x,y) = \sum_{\substack{0 \le x' < \texttt{kernel.cols} \\ 0 \le y' < \texttt{kernel.rows}}} \texttt{kernel}(x',y') * \texttt{src}(x+x'-\texttt{anchor.x}, y+y'-\texttt{anchor.y})$$

Syntax:
cv.filter2D(src, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]])

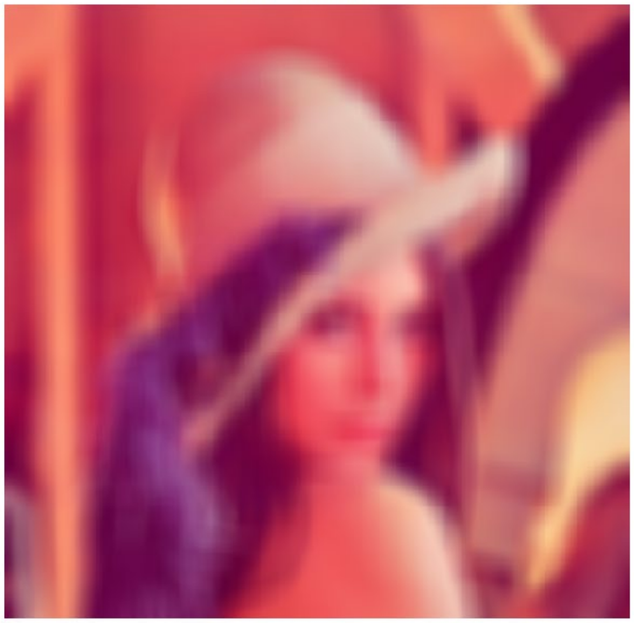
Src: source of the image
Ddepth: the amount of colour information contained in each pixel of an image.
-1 will give the output image depth as same as the input image.
Kernel: a single-channel floating point matrix.
Dst: output image of the same size and the same number of channels as src
Anchor: anchor of the kernel that indicates the relative position of a filtered point within the kernel; the anchor should lie within the kernel; default value (-1,-1) means that the anchor is at the kernel center.
Delta: optional value added to the filtered pixels before storing them in dst.
borderType: pixel extrapolation method

Code snippet:

```
#kernel blurring
kernel_25 = np.ones((25,25), np.float32)/625.0
output_kernel = cv2.filter2D(img, -1, kernel_25)
```

BEFORE

AFTER



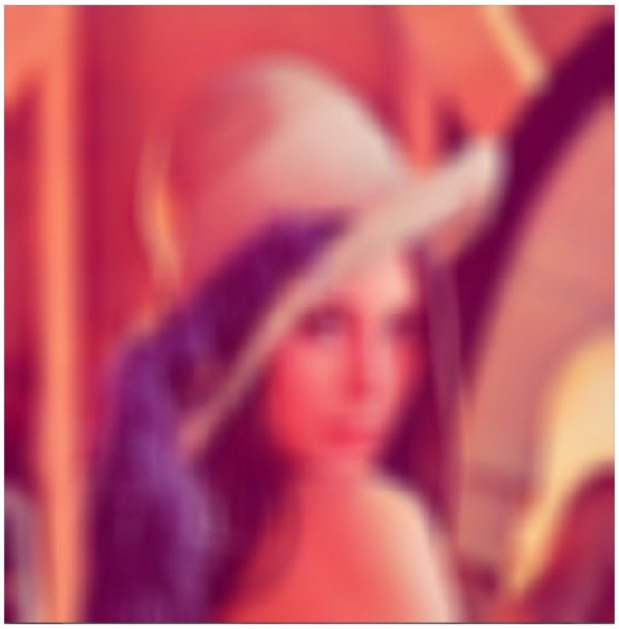
- Blur function:

Syntax:
cv.blur(src, ksize[, dst[, anchor[, borderType]]])

Code snippet:

```
output_blur = cv2.blur(img,(25,25))
```

BEFORE **AFTER**



- **Box filter:**

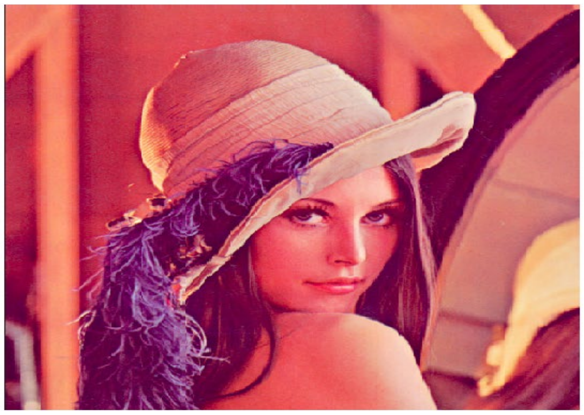
  Syntax:
  cv.boxFilter(src, ddepth, ksize[, dst[, anchor[, normalize[, borderType]]]])

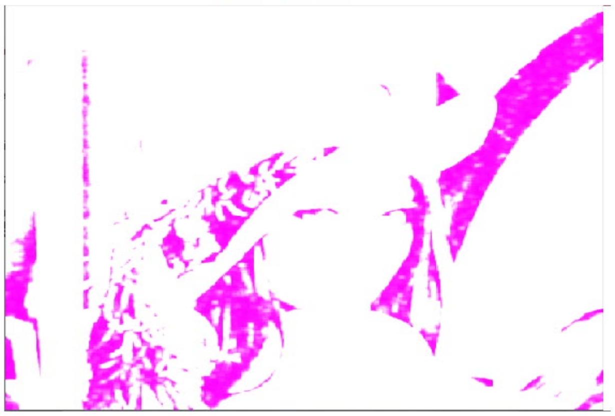
  Normalize: flag, specifying whether the kernel is normalized by its area or not. Used when some image is too dark.

  Code snippet:

```
output_box= cv2.boxFilter(img, -1, (5,5), normalize=False)
```

BEFORE **AFTER**





- **Gaussian blurring:**
  Syntax:
  cv.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]])
  sigmaX: Gaussian kernel standard deviation in X direction.

sigmaY: Gaussian kernel standard deviation in Y direction; if sigmaY is zero, it is set to be equal to sigmaX, if both sigmas are zeros, they are computed from ksize.width and ksize.height, respectively

Code snippet:

```
output_gauss = cv2.GaussianBlur(img,(5,5), 1)
```

Rule of thumb for Gaussian filter design:

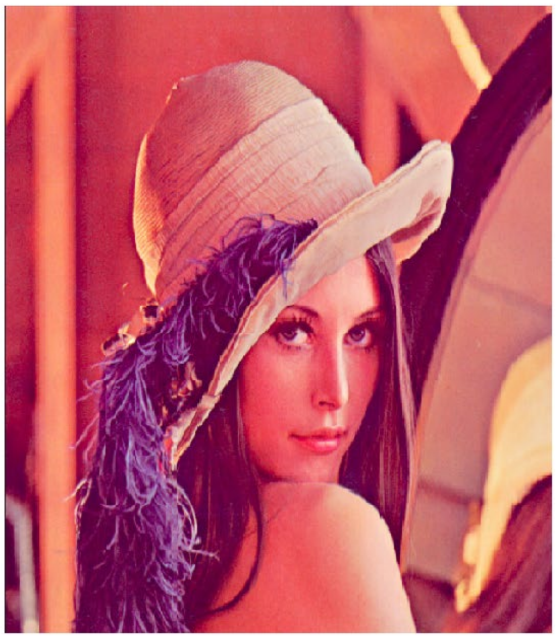
- Choose filter size to be about 3 times the standard deviation(sigma X) in each direction.
- Total filter size is about 6* sigma rounded to odd integer value.
- Gaussian image has nice contrasting images because in Gaussian the influence of outer pixels is less.

- **Median blur(Noise reduction):**
  Square kernel so only one value is given.
  Smoothens the image
  Syntax:
  cv.medianBlur(src, ksize[, dst])

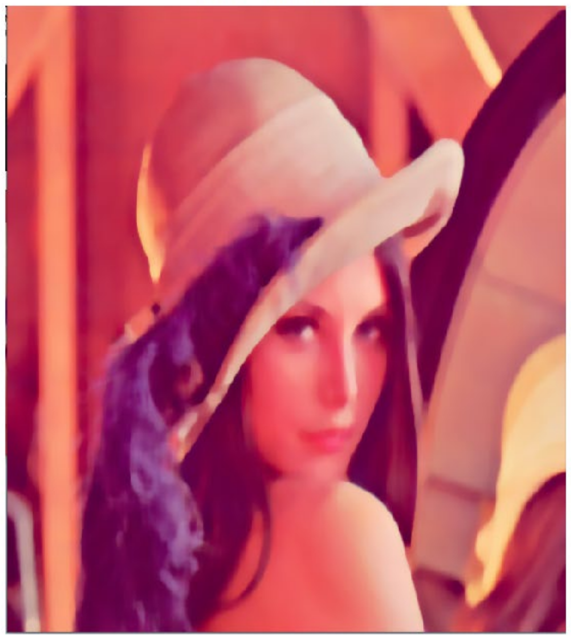
Code snippet:

```
output_med = cv2.medianBlur(img, 5)
```

BEFORE



AFTER



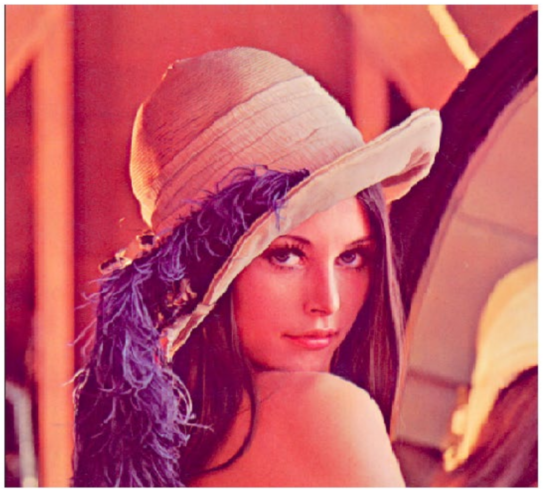
- **Bilateral Filter(Reduction of noise+Preserving of edges)**
  Bilateral filtering makes sure that only those pixels that only those pixels having intensity almost same as target pixel are considered.
  Syntax:
  cv.bilateralFilter(src, d, sigmaColor, sigmaSpace[, dst[, borderType]])

sigmaColor	**Filter sigma in the color space. A larger value of the parameter means that farther colors within the pixel neighborhood will be mixed together, resulting in larger areas of semi-equal color.**
sigmaSpace	**Filter sigma in the coordinate space. A larger value of the parameter means that farther pixels will influence each other as long as their colors are close enough. When d>0, it specifies the neighborhood size regardless of sigmaSpace. Otherwise, d is proportional to sigmaSpace.**

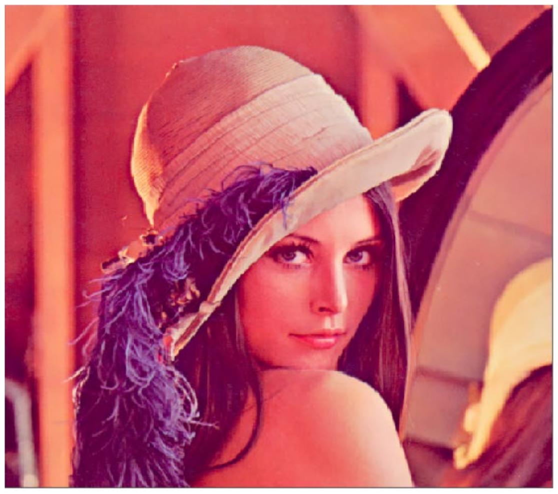
Code snippet:

```
output_bi = cv2.bilateralFilter(img, 10, 6,6)
```

BEFORE



AFTER



2. **Sharpening:**

Code Snippet:

```
gausian_blur = cv2.GaussianBlur(img, (7,7),
2)

#sharpening using add weighted
sharpened = cv2.addWeighted(img, 5.5,
gausian_blur, -4.5, 0)

cv2.imshow('final', sharpened)
```

addWeighted:
The function addWeighted calculates the weighted sum of two arrays as follows:

$$dst(I)=saturate(src1(I)*alpha+src2(I)*beta+gama)$$

Syntax:cv.addWeighted(src1, alpha, src2, beta, gamma[, dst[, dtype]])

Alpha- weight of the first array elements.
Beta-weight of the second array elements.
Gamma-scalar added to each sum.
dtype-optional depth of the output array; when both input arrays have the same depth, dtype can be set to -1.

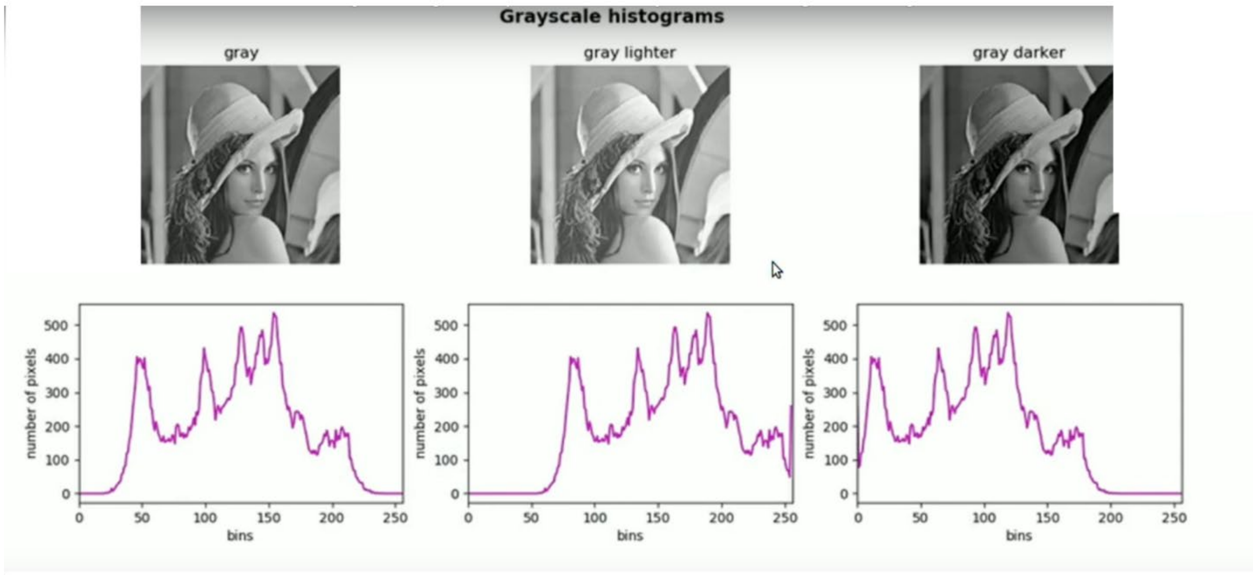
3. **Image Histogram:**
    - An image histogram is a type of histogram that reflects the intensity(tonal) distribution of the image, plotting the number of pixels for each intensity.

Terminologies for Image histogram:

1. **Frequency:** number of pixels for each intensity value.
2. **Bins:** Histograms show the number of pixels(frequency) for every intensity value, ranging from 0 to 255. Each of these 256 values is called a bin in histogram terminology.
3. **Brightness:** The brightness of a grayscale image can be defined as the average intensity of all the pixels of the image given by the following formulation:

$$Brightness = \frac{1}{m \cdot n} \sum_{x=1}^{m} \sum_{y=1}^{n} I(x, y)$$

Grayscale Histograms(only 1 channel):



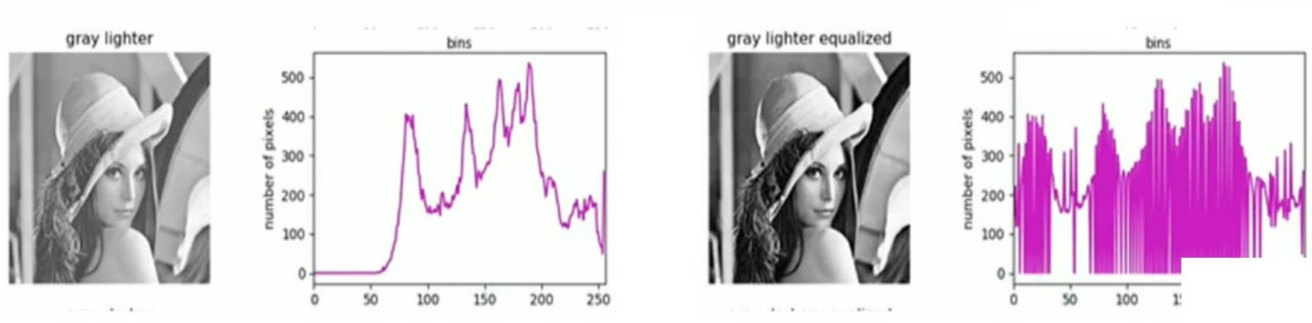
- **For lighter,** graph shifts towards right.
- **For darker,** graph shifts towards left.

Color Histograms:

- In case of multi channel image the process of calculating color histogram involves calculating the histogram in each of the channels.

Histogram equalization:

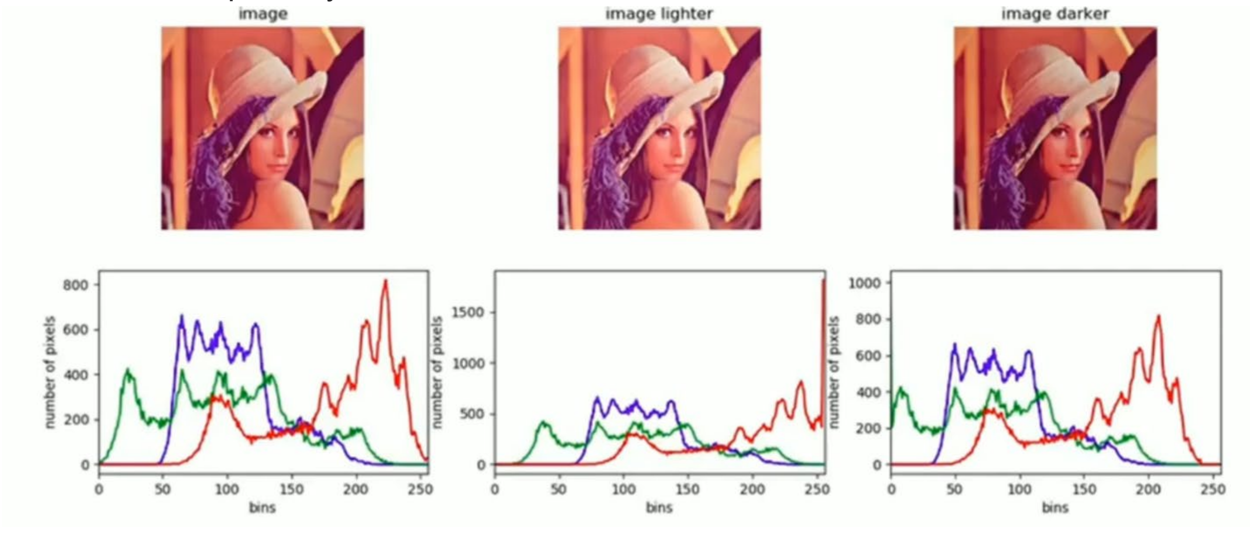
- cv2.equalizeHist() function is used to equalize the image histogram which normalizes the brightness and also increases the contrast.



Color Histogram Equalization:

- We separate channels of image and apply equalization to each of them.

- Instead of BGR we use HSV color space to get light and color information separately.



Problems in histogram equalization:

- The impurities are magnified in global histogram equalization
- Fixed by adaptive histogram equalization(AHE).

Adaptive histogram equalization:

- Image is broken into small blocks called “tiles”.
- Then each of these blocks are equalized as usual.

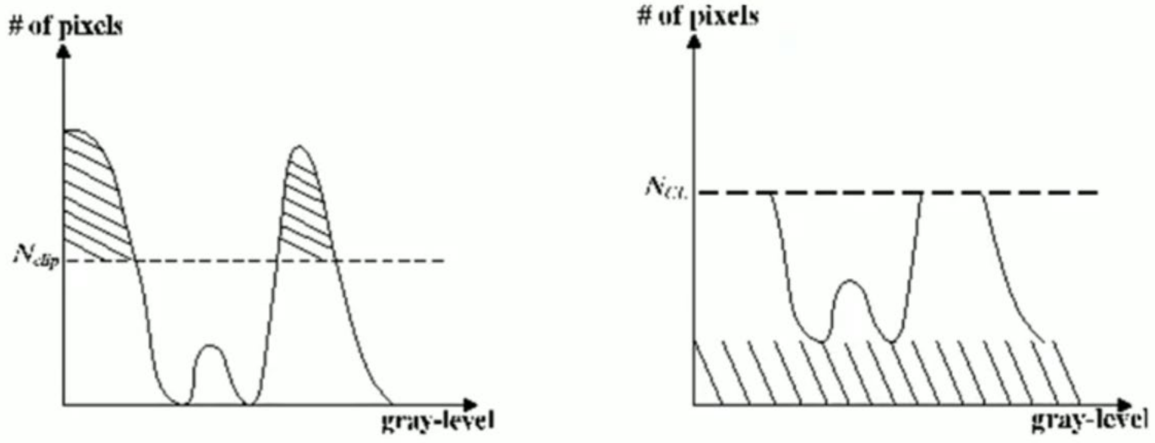
Problems in AHE:

- The noise in relatively homogeneous regions of the image is overamplified by AHE.
- Fixed by CLAHE.

Contrast Limited AHE(CLAHE):

- CLAHE solves the problem of impurity maximization by clipping the extra values.

- If any histogram bin is above the specified contrast limit, those pixels are clipped and distributed uniformly to other bins before applying histogram equalization.



(a) Original histogram (b) Clipped histogram