Int. j. adv. multidisc. res. stud. 2024; 4(4):332-339

Received: 29-05-2024
Accepted: 09-07-2024

International Journal of Advanced Multidisciplinary Research and Studies

ISSN: 2583-049X

The Performance of Histogram Processing in Real Time

Panca Mudjirahardjo
Department of Electrical Engineering, Faculty of Engineering, Universitas Brawijaya, Indonesia

Corresponding Author: **Panca Mudjirahardjo**

Abstract
This research we evaluate the performance of histogram processing, i.e. histogram equalization, CLAHE and histogram matching both for static image and real time frame captured by a camera. For static image, we evaluate objective measures i.e. intensity mean, entropy, PSNR and SSIM. For real time frame, we evaluate the average execution time. The experiment result shows the CLAHE image is closer to the original in terms of pixel values (higher PSNR) and higher similarity between the images in terms of structure, contrast, and luminance (higher SSIM) and has a shorter execution time. The experiment is conducted using programming language python and openCV library.

Keywords: Histogram Equalization, CLAHE, Histogram Matching, Real Time

Introduction

Image processing is the main task in pattern recognition and computer vision. One of image processing is histogram processing to enhance the image contrast. The image contrast is required to distinguish background and objects on it. It is required also in feature extraction.

Histogram processing refers to techniques used to analyze and manipulate the histogram of an image. The histogram of an image represents the frequency distribution of pixel intensities. It plots the number of pixels for each intensity level (from 0 to 255 in an 8-bit grayscale image).

Here are some common histogram processing techniques [1]:

1. **Histogram Equalization:** This technique improves the contrast of an image by redistributing the pixel intensities. It spreads out the intensity values so that they cover the entire range more evenly. This can be particularly useful when an image has a lot of pixels concentrated in a narrow range of intensities.
2. **Histogram Matching/Specification:** This technique allows you to specify a desired histogram that you want your image to have. It adjusts the pixel intensities of the image so that its histogram matches the specified histogram. This is useful for matching the color distribution of images or for performing color transfer between images.
3. **Histogram Stretching:** This technique stretches the range of intensity values in an image to make the image appear sharper and more detailed. It is particularly useful when an image has intensities that are concentrated in a narrow range and need to be spread out to cover the full intensity range.
4. **Histogram Modification:** This includes techniques such as histogram shifting or scaling, which adjust the histogram by adding or multiplying pixel intensities by a constant value. These techniques can be used for brightness or contrast adjustment.
5. **Histogram Clipping:** This technique involves clipping the histogram at certain intensity levels to remove extreme pixel values. It can be used to remove noise or to enhance specific features in an image.
6. **Histogram Smoothing:** This involves applying a smoothing function to the histogram to reduce noise and produce a smoother distribution of pixel intensities.

Histogram processing is widely used in image enhancement, preprocessing for image analysis tasks, and in applications where adjusting the contrast, brightness, or color balance of images is important. It helps in improving image quality and making images more suitable for subsequent processing or visual inspection.

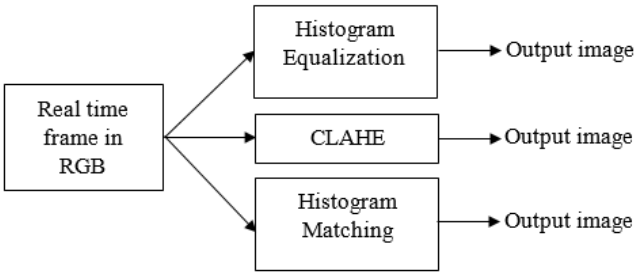


Fig 1: The proposed method of this study

CLAHE stands for Contrast Limited Adaptive Histogram Equalization. It is an extension of histogram equalization that improves the local contrast of an image while limiting the amplification of noise in flat regions of the image.

The Proposed Method

In this section, we briefly explain the proposed method to evaluate the performance of histogram processing in real time. We evaluate histogram equalization, CLAHE and histogram matching in real time. The method is shown in Fig 1.

In Fig 1, the input image from real time frame captured by a camera. Then the RGB image is processed by histogram equalization, CLAHE and histogram matching processing. We evaluate the output image and computation time.

Histogram Equalization

Histogram equalization is a technique used in image processing to adjust the contrast of an image by redistributing the intensity levels. The main goal of histogram equalization is to obtain an image where the pixel intensities are spread out evenly across the entire range of possible intensities (usually 0-255 for 8-bit images), which can make details more visible and improve the overall appearance of the image [1, 2].

Steps in Histogram Equalization

- Compute Histogram: Calculate the histogram of the input image, which shows how many pixels have each intensity value.

$$p_r(r_k) = \frac{n_k}{n} \qquad k = 0,1,2,\dots,L-1 \tag{1}$$

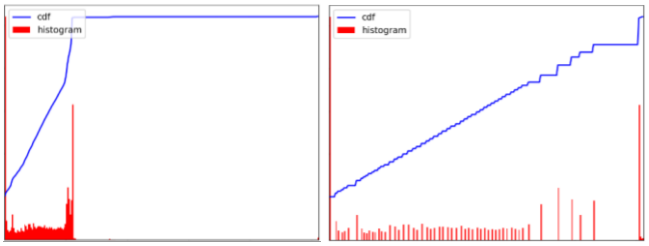
Where, n is the total number of pixels in the image, n_k is the number of pixels that have gray level r_k, and L is the total number of possible gray level in the image [1].

- **Calculate CDF:** Compute the cumulative distribution function (CDF) from the histogram. The CDF gives cumulative sum of histogram values up to each intensity level.
- **Normalize CDF:** Normalize the CDF to map it to the range [0, 1].
- **Create Transformation Function:** Create a transformation function by mapping each intensity value from the original image to a new intensity value based on the normalized CDF.

$$s_k = T(r_k) = \sum_{j=0}^{k} p_r(r_j) = \sum_{j=0}^{k} \frac{n_j}{n} \qquad k = 0,1,2,\dots,L-1 \tag{2}$$

The transformation (mapping) given in equation (2) is called *histogram equalization* or *histogram linearization*.

- **Apply Transformation:** Apply the transformation function to each pixel in the original image to get the equalized image.
- **Mapping:** Each pixel intensity value in the original image is mapped to a new intensity value using the transformation function derived from the CDF.



(a) (b)

Fig 2: Example of histogram and cdf (a) original image (b) equalized image

Advantages and Considerations

- **Enhanced Contrast:** Histogram equalization can improve the contrast of an image by stretching out intensity levels that are concentrated in a narrow range.
- **Noise Amplification:** It can also amplify noise in regions of low contrast. Adaptive techniques like adaptive histogram equalization (AHE) are used to mitigate this issue.
- **Computational Complexity:** The method is relatively simple and computationally inexpensive.
- **Applications:** Histogram equalization is widely used in image enhancement tasks, especially where contrast improvement is critical for visual inspection or analysis.

Overall, while histogram equalization is effective for enhancing global contrast, care must be taken to avoid over-amplifying noise and artifacts, particularly in regions with little detail.

Program-1:

```
# --- import packages --
import cv2
import numpy as np
import time

# --- image capture from camera --
cam = cv2.VideoCapture(0)

img_counter = 0
frameKe = 0

while True:
    start = time.time()

    ret, frame = cam.read()

    # --- create the histogram of original image
    hist,bins = np.histogram(frame.flatten(),256,[0,256])

    # --- calculate cdf, cumulative density function
    cdf = hist.cumsum()
    cdf_normalized = cdf * float(hist.max()) / cdf.max()
```

```
    # --- processing histogram equalization
    cdf_m = np.ma.masked_equal(cdf,0)
     cdf_m = (cdf_m - cdf_m.min())*255/(cdf_m.max()-
cdf_m.min())
    cdf = np.ma.filled(cdf_m,0).astype('uint8')

    img2 = cdf[frame]

    end = time.time()

    # --- displaying the original and equalized image
    cv2.imshow("original img ",frame)
    cv2.imshow("equalized img ",img2)

    if not ret:
        break
    k = cv2.waitKey(1)

    if k%256 == 27:
        # ESC pressed
        print("Escape hit, closing...")
        break
    elif k%256 == 32:
        # SPACE pressed
                                        img_name =
"opencv_frame_{}.jpg".format(img_counter)
        cv2.imwrite(img_name, frame)
        print("{} written!".format(img_name))
        img_counter += 1

    frameKe += 1

          print('frame : {}, execution time: {}
sec.'.format(frameKe,end-start))

cam.release()
cv2.destroyAllWindows()

print()
local_time = time.ctime(start)
print("Local time:", local_time)
print()
```

Contrast Limited Adaptive Histogram Equalization (CLAHE)

Adaptive histogram equalization (AHE) is an image pre-processing technique used to improve contrast in images. It computes several histograms, each corresponding to a distinct section of the image, and uses them to redistribute the luminance values of the image. It is therefore suitable for improving the local contrast and enhancing the definitions of edges in each region of an image. However, AHE has a tendency to over amplify noise in relatively homogeneous regions of an image. A variant of adaptive histogram equalization called contrast-limited adaptive histogram equalization (CLAHE) prevents this effect by limiting the amplification.

Ordinary AHE tends to overamplify the contrast in near-constant regions of the image, since the histogram in such regions is highly concentrated. As a result, AHE may cause noise to be amplified in near-constant regions. Contrast Limited AHE (CLAHE) is a variant of adaptive histogram equalization in which the contrast amplification is limited, so as to reduce this problem of noise amplification [3-7].

In CLAHE, the contrast amplification in the vicinity of a given pixel value is given by the slope of the transformation function. This is proportional to the slope of the neighborhood cumulative distribution function (CDF) and therefore to the value of the histogram at that pixel value. CLAHE limits the amplification by clipping the histogram at a predefined value before computing the CDF. This limits the slope of the CDF and therefore of the transformation function. The value at which the histogram is clipped, the so-called clip limit, depends on the normalization of the histogram and thereby on the size of the neighborhood region. Common values limit the resulting amplification to between 3 and 4.

It is advantageous not to discard the part of the histogram that exceeds the clip limit but to redistribute it equally among all histogram bins [3].

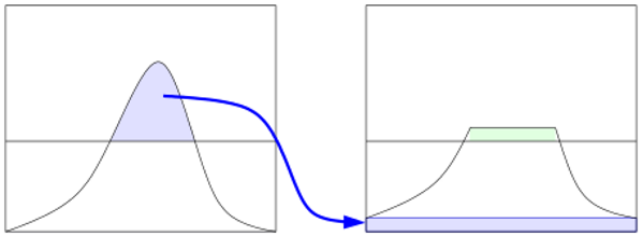


Fig 3: The histogram distribution in CLAHE [3]

The redistribution will push some bins over the clip limit again (region shaded green in the Fig 3), resulting in an effective clip limit that is larger than the prescribed limit and the exact value of which depends on the image. If this is undesirable, the redistribution procedure can be repeated recursively until the excess is negligible.

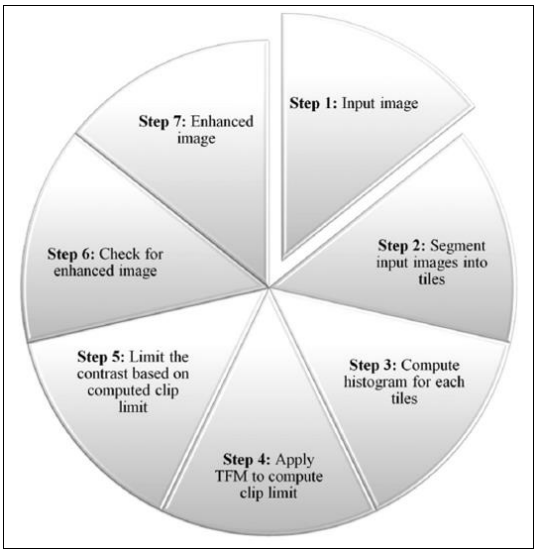


Fig 4: Steps followed in CLAHE algorithm [7]

The CLAHE algorithm has three major parts: Tile generation, histogram equalization, and bilinear interpolation. The input image is first divided into sections. Each section is called a tile. Histogram equalization is then performed on each tile using a pre-defined clip limit. Histogram equalization consists of five steps: Histogram computation, excess calculation, excess distribution, excess redistribution, and scaling and mapping using a cumulative distribution function (CDF). The histogram is computed as a set of bins for each tile. Histogram bin values higher than

the clip limit are accumulated and distributed into other bins. CDF is then calculated for the histogram values. CDF values of each tile are scaled and mapped using the input image pixel values. The resulting tiles are stitched together using bilinear interpolation, to generate an output image with improved contrast.

To increase image contrast, use the CLAHE algorithm as below. Grayscale and color photos can both be processed using this approach.

CLAHE algorithm steps are as follows [7]:

Step 1: Input image
Step 2: Segment input images into tiles
Step 3: Compute histogram for each tiles
Step 4: Apply TFM to compute clip limit
Step 5: Limit the contrast based on computed clip limit
Step 6: Check for enhanced image
Step 7: Enhanced image

Fig 4 illustrates the steps to be followed in the CLAHE algorithm. Prior to creating a histogram for each context region, a given input image is first separated into context regions. So that various portions of the image may be easily linked, a mapping function is used to produce an image mapping. The image noise is subsequently reduced using an interpolation approach. This enables us to lessen the noise in particular regions of the image. Although the method de-noises the image, it does not do so fully.

Program-2:

```
# --- import packages --
import cv2
import time

# --- image capture from camera --
cam = cv2.VideoCapture(0)

img_counter = 0
frameKe = 0

while True:
    start = time.time()

    ret, frame = cam.read()

    img = cv2.cvtColor(frame, cv2.COLOR_RGB2Lab)

    # --- configure CLAHE
                                    clahe                    =
cv2.createCLAHE(clipLimit=10,tileGridSize=(8,8))

     # --- 0 to 'L' channel, 1 to 'a' channel, and 2 to 'b'
channel
    img[:,:,0] = clahe.apply(img[:,:,0])

    img = cv2.cvtColor(img, cv2.COLOR_Lab2RGB)

    end = time.time()

    # --- Showing the two images
    cv2.imshow("ORIGINAL frame", frame)
    cv2.imshow("CLAHE image", img)
 if not ret:
        break
    k = cv2.waitKey(1)

    if k%256 == 27:
        # ESC pressed
        print("Escape hit, closing...")
        break
    elif k%256 == 32:
        # SPACE pressed
                                                  img_name          =
"opencv_frame_{}.jpg".format(img_counter)
        cv2.imwrite(img_name, frame)
        print("{} written!".format(img_name))
        img_counter += 1

    frameKe += 1

            print('frame : {}, execution time: {}
sec.'.format(frameKe,end-start))

cam.release()
cv2.destroyAllWindows()

print()
local_time = time.ctime(start)
print("Local time:", local_time)
print()
```

Histogram Matching

Histogram matching, also known as histogram specification or histogram equalization, is a technique used in image processing to adjust the contrast of an image by modifying its intensity distribution [1, 8, 9]. Here's a concise explanation of how histogram matching works:

Process of Histogram Matching:

1. Compute Histograms:
- Calculate the histogram of the input image and the histogram of the reference image (or the desired histogram).

2. Cumulative Distribution Function (CDF):
- Compute the cumulative distribution function (CDF) for both histograms. The CDF gives the cumulative probability of intensity values.

3. Mapping Function:
- Create a mapping function that maps each intensity value in the input image to a new intensity value. This mapping function is typically derived from the relationship between the CDFs of the input and reference histograms.

4. Adjust Intensity Values:
- Apply the mapping function to every pixel in the input image. This transforms the intensity values such that the histogram of the output image matches the desired histogram (reference histogram).

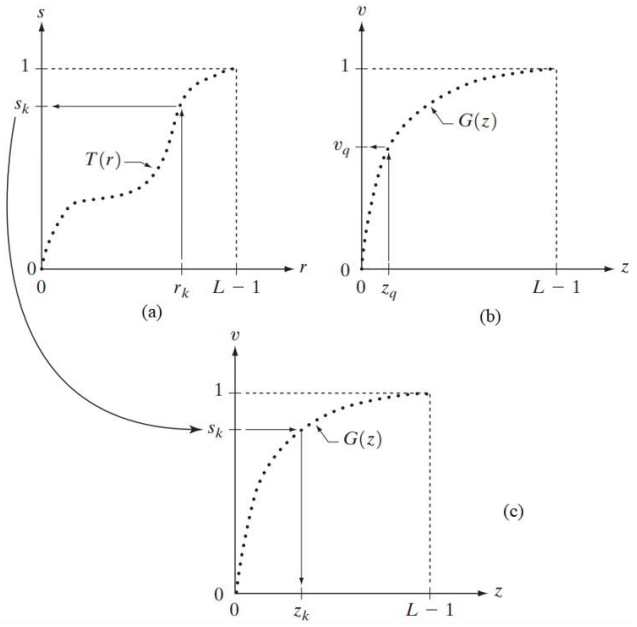


Fig 5: (a) Graphical interpretation of mapping from r_k to s_k via $T(r)$ in original image (b) mapping of z_q to its corresponding value v_q via $G(z)$ in reference image (c) inverse mapping from s_k to its corresponding value of z_k [1]

Benefits and Applications:

- **Enhancing Image Contrast:** Histogram matching can improve the contrast of an image by spreading out the intensity values more uniformly.
- **Standardization:** It can be used to standardize images so that they have similar intensity distributions, which is useful in applications like medical imaging or remote sensing.
- **Image Restoration:** In some cases, histogram matching can help in restoring details in an image that might be lost due to poor lighting conditions or improper exposure.

Example Scenario:
Imagine you have a photograph that is underexposed, making it too dark. By using histogram matching with a reference histogram of a well-exposed photograph (with a desired intensity distribution), you can adjust the intensities of the pixels in the underexposed photograph to match those of the well-exposed photograph. This process would result in an image with improved brightness and contrast, making details more discernible.

Considerations:

- **Local vs. Global Matching:** Histogram matching can be applied globally (to the entire image) or locally (in patches or regions of the image), depending on the application and the desired effect.
- **Effect on Image Information:** While histogram matching can enhance visual quality, it can also potentially alter the original information content of an image, so it should be used judiciously in applications where preserving exact pixel values is crucial.

Histogram matching is a powerful tool in image processing, often used in conjunction with other techniques to enhance the quality and interpretability of digital images across various domains.

Program-3:

```
# --- import packages --
import cv2
import matplotlib.pyplot as plt
from skimage.exposure import match_histograms
import time

# --- image capture from camera --
cam = cv2.VideoCapture(0)

img_counter = 0
frameKe = 0

# --- reading reference image
imgREF = cv2.imread("REFimage.jpg")

while True:
    start = time.time()

    ret, frame = cam.read()

    h,w,ch=frame.shape
            print("height: {}, width: {}, channel: {}".format(h,w,ch))

    matched = match_histograms(frame, imgREF,
                    channel_axis=-1)

    end = time.time()

    cv2.imshow("original", frame)
    cv2.imshow("matched", matched)

    if not ret:
        break
    k = cv2.waitKey(1)

    if k%256 == 27:
        # ESC pressed
        print("Escape hit, closing...")
        break
    elif k%256 == 32:
        # SPACE pressed
                                        img_name = "opencv_frame_{}.jpg".format(img_counter)
        cv2.imwrite(img_name, frame)
        print("{} written!".format(img_name))
        img_counter += 1

    frameKe += 1

            print('frame : {}, execution time: {} sec.'.format(frameKe,end-start))

cam.release()
cv2.destroyAllWindows()

print()
local_time = time.ctime(start)
print("Local time:", local_time)
print()
```

The Evaluation

To evaluate the output image, we use objective measures i.e. *intensity mean, entropy, peak signal-to-noise ratio* (*PSNR*) and *Structural Similarity Index Measure* (*SSIM*) [9].

Intensity mean for an image refers to the average pixel intensity across the entire image. It's a straightforward measure that provides a general sense of the brightness of the image as perceived by the human eye.

Here's how intensity mean is typically calculated:

1. **Convert Image to Grayscale:** If the image is in color (RGB format), it's usually converted to grayscale to simplify the calculation. This is done because grayscale images have a single intensity value per pixel, making it easier to compute the mean intensity.
2. **Calculate Mean Intensity:** Once in grayscale, the mean intensity I_{mean} can be calculated by averaging the intensity values of all pixels in the image. Mathematically, it can be represented as:

$$I_{mean} = \frac{1}{N}\sum_{i=1}^{N} I_i \quad (3)$$

Where: N is the total number of pixels in the image, I_i is the intensity value of the i-th pixel.

3. **Interpretation:** The resulting I_{mean} gives a single number that represents the average brightness of the image. Higher values indicate a brighter image, while lower values indicate a darker image.

Intensity mean is a basic but useful metric in image processing and analysis. It can be used for tasks such as image normalization, contrast adjustment, and as a component in more complex image analysis algorithms.

Entropy in the context of images refers to a measure of randomness or uncertainty in the pixel intensity values across the image. It quantifies the amount of information or disorder present in the image.

Calculation of Entropy:

1. **Convert Image to Grayscale:** Similar to intensity mean calculation, the image is typically converted to grayscale if it is in color.
2. **Histogram Calculation:** Compute the histogram of the grayscale image, which represents the frequency distribution of pixel intensity values.
3. **Probability Distribution:** Normalize the histogram to obtain the probability distribution $p(i)$, where i represents the intensity levels ranging from 0 (black) to 255 (white) in an 8-bit grayscale image.
4. **Entropy Calculation:** Calculate the entropy H using the formula:

$$H = -\sum_{i=0}^{255} p(i)\log_2(p(i)) \quad (4)$$

Where: $p(i)$ is the normalized probability of intensity level i, $\log_2$ denotes the base-2 logarithm.

Interpretation:

- **High Entropy:** Indicates a high degree of randomness or complexity in the distribution of pixel intensities. Images with high entropy may appear noisy or textured.
- **Low Entropy:** Indicates a more predictable or ordered distribution of pixel intensities. Images with low entropy may appear more uniform or smooth.

Peak Signal-to-Noise Ratio (PSNR) is a widely used objective metric for evaluating the quality of reconstructed or compressed images. It measures the quality degradation by computing the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. PSNR is expressed in decibels (dB) and higher values indicate better image quality.

Calculation of PSNR:

1. Mean Squared Error (MSE): First, compute the Mean Squared Error between the original image I and the reconstructed or compressed image $\hat{I}$:

$$\text{MSE} = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - \hat{I}(i,j)]^2 \quad (5)$$

Where m and n are the dimensions of the image, $I(i,j)$ is the pixel value of the original image at position (i,j), and $\hat{I}(i,j)$ is the corresponding pixel value of the reconstructed or compressed image.

2. **Peak Value (typically 255 for 8-bit images):** Calculate the maximum possible pixel value (often 255 for 8-bit grayscale images or 255 for each channel in 24-bit color images).

3. **PSNR Calculation:** Finally, PSNR is calculated using the MSE and peak value:

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{\text{Peak Value}^2}{\text{MSE}}\right) \quad (6)$$

The result is typically expressed in decibels (dB).

Interpretation:

- **Higher PSNR:** Indicates higher image quality, meaning the reconstructed or compressed image is closer to the original in terms of pixel values.
- **Lower PSNR:** Indicates lower image quality, suggesting more noticeable differences between the original and reconstructed/compressed images.

Considerations:

- **Subjectivity:** PSNR is an objective measure but may not always correlate perfectly with human perception of image quality. Some artifacts that are noticeable to humans may not significantly affect PSNR.
- **Dynamic Range:** PSNR can vary depending on the dynamic range of the image and the choice of peak value. For different bit depths or color spaces, appropriate adjustments may be needed.

In summary, PSNR provides a quantitative measure of image quality degradation and is valuable for comparing the effectiveness of different image processing or compression techniques.

SSIM (Structural Similarity Index Measure) is indeed a well-known metric used to assess the similarity between two images. SSIM takes into account three components of similarity between images: luminance, contrast, and structure.

Calculation of SSIM:

SSIM is calculated by comparing local patterns of pixel intensities that have been normalized for luminance and contrast. Here's how it's typically computed:

1. **Luminance Comparison:** Calculate the mean luminance μ_x and μ_y and the variance of luminance σ_x^2 and σ_y^2 for the two images *X* and *Y*.
2. **Contrast Comparison:** Calculate the covariance of *X* and *Y* (σ_{xy}).
3. **Structure Comparison:** Calculate the SSIM index for each pixel, combining the three comparisons into a single value using the formula:

$$\text{SSIM}(X,Y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \quad (7)$$

Where C_1 and C_2 are constants to stabilize the division with weak denominator.

4. **Average SSIM:** To get the average SSIM score over the entire image, compute the mean SSIM value of all pixels.

Interpretation:

- **SSIM Value:** The SSIM index ranges from -1 to 1, where 1 indicates perfect similarity between two images.
- **Higher SSIM:** Indicates higher similarity between the images in terms of structure, contrast, and luminance.
- **Lower SSIM:** Indicates greater differences between the images.

Considerations:

- **Perceptual Relevance:** SSIM correlates well with human perception of image quality, making it a valuable tool in assessing image fidelity.
- **Computational Efficiency:** Calculating SSIM can be computationally intensive, especially for large images or video sequences, but efficient algorithms have been developed to handle this.

In summary, SSIM is a powerful metric for evaluating the structural similarity between images and is widely used in various fields where accurate image quality assessment is critical.

Code program to calculate the intensity mean, entropy, PSNR and SSIM in this study is written in Program-4.

Program-4:

```
import numpy as np
import cv2 as cv
from scipy.stats import entropy
from skimage.metrics import structural_similarity as ssim

# --- read the input image
imgIN = cv.imread('inputImg.jpg', cv.IMREAD_GRAYSCALE)

# --- create the original histogram
histIN,bins = np.histogram(imgIN.flatten(),256,[0,256])
prob_hist_in = histIN/histIN.sum()

# --- processing HE, CLAHE, HM -------
    { ---- }
    return imgOUT

# --- create the output histogram
histOUT,bins = np.histogram(imgOUT.flatten(),256,[0,256])
prob_hist_out = histOUT/histOUT.sum()

# --- calculate the intensity mean
meanIN = np.mean(imgIN)
meanOut = np.mean(imgOUT)

print('input: intensity mean: {}'.format(meanIN))
print('output: intensity mean: {}'.format(meanOut))

# --- calculate entropy
image_entropy_in = entropy(prob_hist_in, base=2)
image_entropy_out = entropy(prob_hist_out, base=2)

print(f"Image Entropy input: {image_entropy_in}")
print(f"Image Entropy output: {image_entropy_out}")

# --- calculate PSNR
mse = np.mean((imgIN - imgOUT) ** 2)
max_pixel = 255.0
psnr = 20 * np.log10(max_pixel / np.sqrt(mse))

print('PSNR: ',psnr)

# --- calculate SSIM
(score, diff) = ssim(imgIN, imgOUT, full=True)
diff = (diff * 255).astype("uint8")
print("SSIM: {}".format(score))
```

The Experimental Result

In this section, the experimental procedure and result are briefly explain. This experiment is performed using programming language python and openCV library. The programming codes in real time to process the histogram equalization, CLAHE and histogram matching are written in Program-1, Program-2, Program-3, respectively.

We use images in Fig 6 to evaluate using objective measures and the evaluation result is summarized in Table 1. For real time processing, the output images for histogram equalization, CLAHE and histogram matching processing are depicted in Figure 7, 8 and 9 respectively. For frame size 480×640 pixels, the average execution time is summarized in Table 2.

(a) (b) (c)

(d) (e)

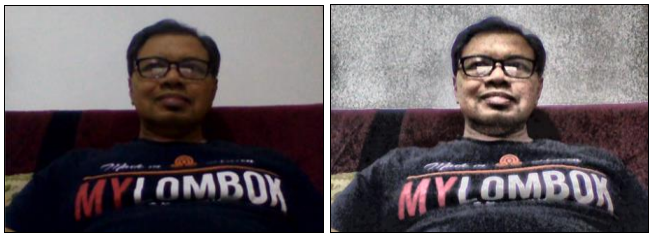
Fig 6: (a) original image (b) equalized image (c) CLAHE image (d) matched image (e) reference image used for (d)

Table 1: The evaluation using objective measures for Figure 6

Image	Mean	Entropy	PSNR (dB)	SSIM
Original	17	5.13		
Hist. Equalization	130	**5.05**	27.75	0.125
CLAHE	51	6.65	**27.78**	**0.309**
Hist. Matching	158	7.31	27.51	0.105

(a) (b)

Fig 7: (a) original image (b) equalized image



(a) (b)

Fig 8: (a) original image (b) CLAHE image

(a) (b)

(c)

Fig 9: (a) original image (b) matched image (c) reference image

Table 2: The average execution time each processing

Process	Average execution time (ms)
Histogram equalization	46,9
CLAHE	**14,9**
Histogram matching	42,8

Conclusion

From the above study, we evaluate the performance of histogram processing, i.e. histogram equalization, CLAHE and histogram matching both for static image and real time frame. For static image, we evaluate objective measures i.e. intensity mean, entropy, PSNR and SSIM. Table 1 shows the CLAHE image is closer to the original in terms of pixel values and higher similarity between the images in terms of structure, contrast, and luminance.

For real time frame, we evaluate the average execution time. Table 2 shows the CLAHE process has a shorter execution time.

References

1. Gonzalez RC, Woods RE. Digital Image Processing, second edition. Prentice Hall, 2001.
2. Xie Y, Ning L, Wang M, Li C. Image Enhancement Based on Histogram Equalization. Journal of Physics: Conference Series, Volume 1314, 3rd International Conference on Electrical, Mechanical and Computer Engineering 9–11 August, 2019.
3. Pizer SM, Amburn EP, Austin JD, *et al*. Adaptive Histogram Equalization and Its Variations. Computer Vision, Graphics, and Image Processing. 1987; 39:355-368.
4. Zuiderveld K. Contrast Limited Adaptive Histogram Equalization. In: P. Heckbert: Graphics Gems IV, Academic Press, 1994. ISBN 0-12-336155-9.
5. Sund T, Moystad A. Sliding window adaptive histogram equalization of intra-oral radiographs: Effect on diagnostic quality. Dentomaxillofac Radiol. 2006; 35(3):133-138.
6. Vidhya GR, Ramesh H. Effectiveness of contrast limited adaptive histogram equalization technique on multispectral satellite imagery, Proc. Int. Conf. Video Image Process., Dec, 2017, 234-239.
7. Venkatesh S, John De Britto C, Subhashini P, Somasundaram K. Image Enhancement and Implementation of CLAHE Algorithm and Bilinear Interpolation. Cybernetics and systems: an International Journal, 2022.
8. Zhang X, Feng Z. New Development of the Image Matching Algorithm. Conference paper in lecture notes of the Institute for Computer Sciences, July, 2017.
9. Mim T-E-J, Mony SA. Image Enhancement using Local Histogram Matching with Normal Distribution. International Journal for Multidisciplinary Research (IJFMR). 2023; 5(4).