IMAGE ENHANCEMENT

1. Write a program to demonstrate histogram equalization.
2. Do the same using adaptive histogram equalization.
3. Perform different image smooth/blur method using opencv.

```
import cv2
from google.colab.patches import cv2_imshow
```

Histogram Equilization

Histogram equalization is a technique used in image processing to enhance the contrast of an image by redistributing the intensity values. It works by mapping the input image histogram to a new histogram that is more evenly distributed.

The formula for histogram equalization is as follows:

$$G(z) = round\left(\frac{(L-1)}{(M \times N)}\sum_{i=0}^{z} n_i\right)$$

where:

- $G(z)$ is the intensity mapping function
- L is the number of intensity levels in the image
- M is the number of rows in the image
- N is the number of columns in the image
- n_i is the frequency of occurrence of intensity level i

The steps involved in histogram equalization are:

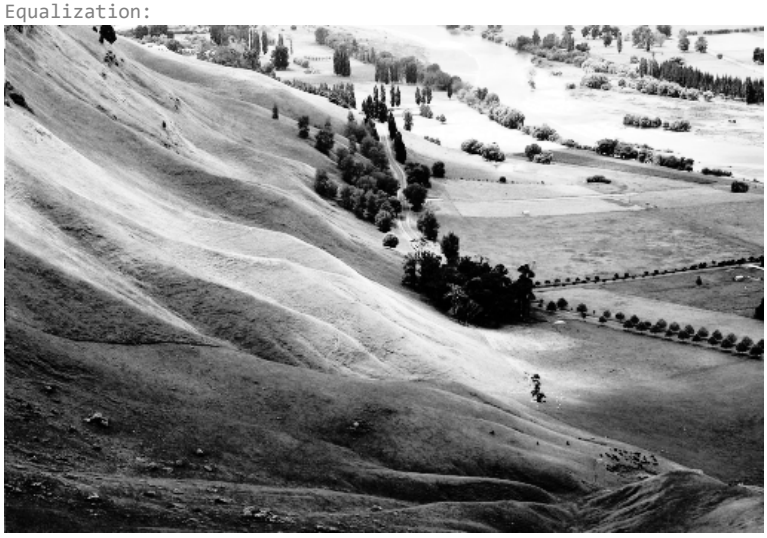
1. Compute the histogram of the input image
2. Calculate the cumulative distribution function (CDF) of the histogram
3. Map the input intensity levels to new intensity levels using the formula above
4. Generate the equalized image using the mapped intensity values

Histogram equalization is a simple yet effective method for improving the contrast of an image, making it a widely used technique in computer vision and image processing applications.

```
img = cv2.imread('/content/drive/MyDrive/CVIP/LAB/Images/Unequalized.jpg',1)
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img = cv2.resize(img, (0, 0), fx = 0.5, fy = 0.5)


equi = cv2.equalizeHist(img)


print('Source image:')
cv2_imshow(img)

print('Equalization:')
cv2_imshow(equi)
```

Source image:

Equalization:



Adaptive Image Equalization

Adaptive histogram equalization (AHE) is an extension of histogram equalization that operates on small regions of an image instead of the entire image. This allows for local contrast enhancement, making it particularly effective for images with varying illumination levels or regions of interest.

The formula for adaptive histogram equalization is as follows:

$$G(x,y) = round\left(\frac{(L-1)}{N}\sum_{i=0}^{f(x,y)} n_i\right)$$

where:

- $G(x,y)$ is the intensity mapping function for a pixel at location (x,y)
- L is the number of intensity levels in the image
- N is the size of the local region around pixel (x,y)
- $f(x,y)$ is the intensity value of the pixel at location (x,y)
- n_i is the frequency of occurrence of intensity level i in the local region

The steps involved in adaptive histogram equalization are:

1. Divide the image into small overlapping regions
2. Compute the histogram and cumulative distribution function (CDF) for each region
3. Map the intensity values of each pixel in the region using the formula above
4. Combine the enhanced regions to generate the final equalized image

Adaptive histogram equalization is a powerful technique for improving the contrast and details of an image, especially in scenarios where local contrast enhancement is required. It is commonly used in medical imaging, satellite imagery, and other applications where enhancing specific regions of an image is crucial for analysis and interpretation.

```
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
adatpive_equalization = clahe.apply(img)


print('Source Image:')
cv2_imshow(img)

print('Adaptive Equalization:')
cv2_imshow(adatpive_equalization)
```

```
Source Image:
```

```
Adaptive Equalization:
```

Image Smoothing/Blurring

Image smoothing or blurring is a common technique in image processing used to reduce noise and sharp transitions in an image, resulting in a more visually pleasing and easier-to-analyze image. One of the widely used methods for image smoothing is Gaussian smoothing.

The formula for Gaussian smoothing is given by:

$$I_{smoothed}(x,y) = \frac{1}{2\pi\sigma^2} \sum_{i=-k}^{k} \sum_{j=-k}^{k} I(x+i, y+j) \cdot e^{-\frac{i^2+j^2}{2\sigma^2}}$$

where:

- $I_{smoothed}(x,y)$ is the smoothed intensity value at pixel location (x,y)
- $I(x+i,y+j)$ is the intensity value of the neighboring pixel at offset (i,j)
- σ is the standard deviation of the Gaussian kernel
- k is the size of the Gaussian kernel

The steps involved in Gaussian smoothing are:

1. Define a Gaussian kernel based on the desired smoothing effect and standard deviation
2. Convolve the image with the Gaussian kernel to compute the smoothed intensity values for each pixel
3. Replace the original intensity values with the smoothed values to obtain the blurred image

Gaussian smoothing is effective in reducing noise and preserving important image features while achieving a visually pleasing result. It is commonly used in applications such as image denoising, edge detection, and feature extraction in computer vision and image processing tasks.

Gaussian blur is a popular technique used in image processing to reduce noise and sharp transitions in an image by convolving the image with a Gaussian kernel. This process results in a smoothed version of the image, which is visually more pleasing and easier to analyze.

The formula for Gaussian blur is given by:

$$I_{smoothed}(x, y) = \frac{1}{2\pi\sigma^2} \sum_{i=-k}^{k} \sum_{j=-k}^{k} I(x+i, y+j) \cdot e^{-\frac{i^2+j^2}{2\sigma^2}}$$

where:

- $I_{smoothed}(x, y)$ is the smoothed intensity value at pixel location (x, y)
- $I(x+i, y+j)$ is the intensity value of the neighboring pixel at offset (i, j)
- σ is the standard deviation of the Gaussian kernel
- k is the size of the Gaussian kernel

The steps involved in Gaussian blur are:

1. Define a Gaussian kernel based on the desired blurring effect and standard deviation
2. Convolve the image with the Gaussian kernel to compute the smoothed intensity values for each pixel
3. Replace the original intensity values with the smoothed values to obtain the Gaussian-blurred image

Gaussian blur is widely used in various image processing tasks such as image denoising, edge detection, and feature extraction. It provides a balance between noise reduction and preservation of important image details, making it a fundamental tool in computer vision and image processing applications.

```
#Averaging
avg_blur = cv2.blur(adatpive_equalization,(5,5))


print('Source Image:')
cv2_imshow(adatpive_equalization)

print('Average Blur:')
cv2_imshow(avg_blur)
```

Source Image:

Average Blur:

```
#Gaussian Blur
gaussian_blur = cv2.GaussianBlur(adatpive_equalization,(5,5),0)


print('Source Image:')
cv2_imshow(adatpive_equalization)

print('Gaussian Blur:')
cv2_imshow(gaussian_blur)
```