Dynamic Histogram Equalization and Color Histogram Equalization

Hritam Basak
ID: 114783055

Sai Rachana Yerram
ID: 115401413

Part 1: Dynamic Histogram Equalization

Introduction

Histogram equalization is a commonly used technique in image processing that helps in enhancing the contrast and brightness of an image. It is a non-linear method that involves altering the intensity distribution of an image in order to make the dark and bright regions more distinguishable. This technique has numerous applications in various fields such as medical imaging, computer vision, and satellite imagery.

The goal of histogram equalization is to redistribute the intensity values of an image such that the resulting image has a uniform distribution of pixel intensities. This is achieved by computing the histogram of the image and then applying a transformation function to the pixel values. This works by mapping the original intensity values of an image to new values such that the new distribution of intensities is uniform. This process involves increasing the contrast of an image by spreading the pixel intensities over the entire range of the image. The result is a higher dynamic range of pixel intensities that allow for greater contrast and detail in the image.

Why is GHE not the best solution?

Gray histogram equalization (GHE) is a useful technique for enhancing the contrast of grayscale images. However, it has some limitations that can make it less effective in certain situations. One of the main limitations of traditional histogram equalization is that it operates globally on the entire image, which can lead to over-enhancement of some areas and under-enhancement of others. This can result in the generation of artificial noise in the image and can also lead to the loss of important details in the image. This is evident, especially in the images where the histogram is much skewed across the intensity values, i.e. the image has both very high and low pixel intensity values. Following is an example of this scenario.

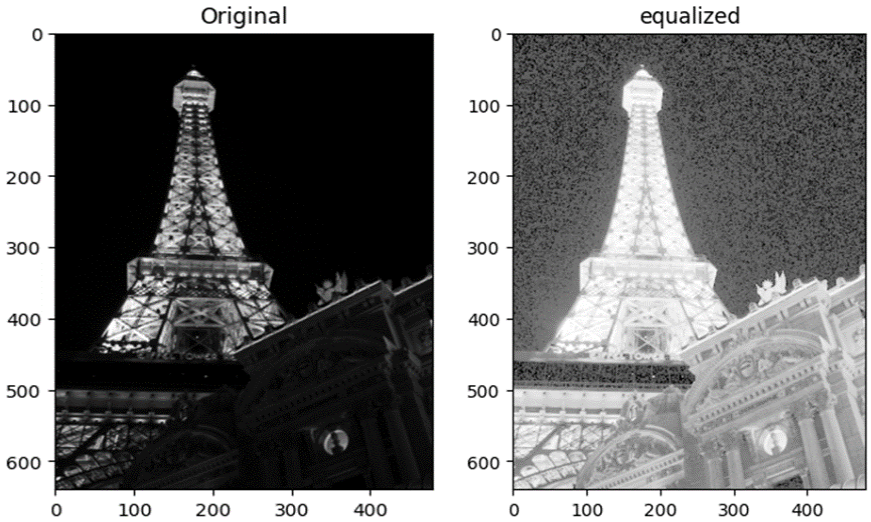

Figure: Original image and resultant image after traditional gray-level histogram equalization. Clearly, this points out the drawbacks of GHE.

Dynamic histogram equalization, on the other hand, is a better solution because it addresses some of the shortcomings of traditional histogram equalization. Dynamic histogram equalization is a local image processing technique that operates on small regions of an image rather than on the entire image. By doing so, it can adjust the enhancement of each region independently, resulting in a more natural and visually pleasing output.

Another advantage of dynamic histogram equalization is that it can adapt to changes in lighting conditions within an image. This is particularly useful in situations where the lighting conditions vary significantly across different parts of the image. For example, in medical imaging, dynamic histogram equalization can be used to enhance the contrast of X-ray images, which often have significant variations in lighting across different regions.

Dynamic histogram equalization also includes a constraint on the maximum and minimum pixel values in each region, which helps to prevent over-amplification of pixel values. This can result in the generation of artificial noise in the image and can also lead to the loss of important details in the image.

Additionally, dynamic histogram equalization can adapt to changes in lighting conditions within an image and includes constraints on the maximum and minimum pixel values in each region to prevent over-amplification of pixel values. If we look at the histograms and CDF of the above images, we can identify the problem clearly.

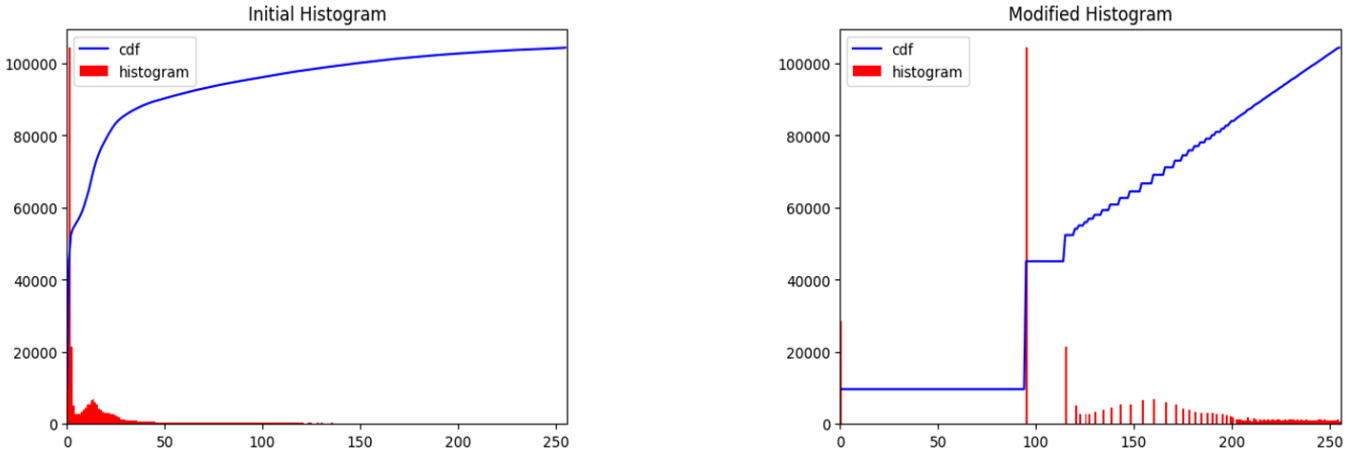

Figure: Histogram and CDF of normal image and equalized image. The CDF is not uniform in the resultant image.

Methodology

We start from the basic equation for histogram equalization, and then we extend it for any given distribution. Let $I(x,y)$ be a gray-level image defined on the square domain [0, M]^2. $CDF(0,r)$ is defined as:

$$P(r) := \int_0^r p(s)ds$$

Then we define the following flow:

$$\frac{\partial I(x,y,t)}{\partial t} = [M^2 - P(I(x,y,t))] - A^{(t)}(I(x,y,t)),$$

Where $A^{(t)}(s) = Area\ of\ set\{(w,z): I(w,z,t) \geq s\}$. Now let I_{SS} denotes the steady state value of the above evolution equation as $t \to \infty$ and $A(s) = Area\ of\ set\{(w,z): I_{SS}(w,z) \geq s\}$. Then the LHS of this equation is 0, resulting to the following state:

$$M^2 - P(I_{ss}(x,y)) = A(I_{ss}(x,y)),$$

Which implies the following condition:

$$Area\ of\ the\ set\ \{(w,z) : s+\varepsilon \geq I_{ss}(w,z) \geq s\} = P(s+\varepsilon) - P(s), \quad s = I_{ss}(x,y).$$

Now dividing both sides by ϵ and assuming $\epsilon \to 0$, we perform Taylor's expansion on the RHS and we reach the differentiation of CDF, which is nothing byt PDF. If we take p to be constant, then we have precisely a uniform distribution and we have recovered histogram equalization.

Local Contrast Enhancement

For local contrast enhancement, we define the following flow:

$$\frac{\partial I(x,y,t)}{\partial t} = [M^2 - P(I(x,y,t))] - A^{(t)}_{B(\delta)}(I(x,y,t))$$

Where $A^{(t)}_{B(\delta)}(s) = Area\ of\ set\ \{(w,z) \in B(\delta) : I(w,z,t) \geq s\}$, where $B(\delta)$ denotes a ball of radius $r = \delta$ around coordinate (x,y). So, instead of considering the whole image, we consider a small circular region of the image.

Further, to improve the performance, we can perform anisotropic diffusion (Perona-Malik) or Laplacian of the image to generate a smoothened version. This, thereafter, can be combined to the equalized image for a better output. This takes the following form:

$$\frac{\partial I(x,y,t)}{\partial t} = \alpha\ div\left(\frac{\nabla I(x,y,t)}{\|\nabla I(x,y,t)\|}\right)^{1/3} \|\nabla I(x,y,t)\| + [M^2 - P(I(x,y,t))] - A^{(t)}(I(x,y,t)).$$

Where $\alpha \geq 0$ is a parameter controlling the smoothing contribution to the final output.

OKAY, ENOUGH THEORY, LET US IMPLEMENT SOMETHING. Here we show the implementation of the above algorithm.

Implementation and Results

```
import cv2
import numpy as np

# Load the input image
img = cv2.imread('input_image.jpg', 0)

# Define the size of the local windows for contrast limited adaptive histogram equalization
```

```
dhe = cv2.createCLAHE(clipLimit=2.0)

# Apply DHE to the image
img_dhe = dhe.apply(img)

# Display the output image
cv2.imshow('Dynamic Histogram Equalization', img_dhe)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Here is a step-by-step explanation of the code:

1. First, we import important image processing libraries like cv2, numpy, etc.

3. The third line loads the input image from the file 'input_image.jpg' using the `cv2.imread()` function. The second argument (0) indicates that we want to load the image in grayscale mode.

4. The fourth line defines the size of the local windows for dynamic histogram equalization using the `cv2.createCLAHE()` function. Here, we set the clip limit to 2.0, which determines the maximum amount of contrast enhancement that can be applied to a pixel value, i.e. the radius of the local region circle is 2.0

5. The fifth line applies DHE to the input image using the `apply()` function of the `dhe` object. This function applies the DHE algorithm to the image, enhancing the contrast of the image.

6. The sixth line displays the output image in a window with the title 'Dynamic Histogram Equalization' using the `cv2.imshow()` function.

Overall, the code demonstrates how to perform dynamic histogram equalization using the DHE algorithm in OpenCV, which can be useful for enhancing the contrast of grayscale images.

Now, we also perform Laplacian of the image to generate the smoothened version of the image.

```
import cv2
import numpy as np

# Load the input image
img = cv2.imread('input_image.jpg')
```

```
# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Apply the Laplacian operator to the grayscale image
laplacian = cv2.Laplacian(gray, cv2.CV_64F)

# Convert the Laplacian output to a normalized 8-bit image
laplacian = cv2.normalize(laplacian, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)

# Display the smoothed image
cv2.imshow('Smoothed Image', laplacian)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Here's a step-by-step explanation of the code:

1. The first two lines import the necessary libraries, OpenCV and NumPy.

2. The third line loads the input image from the file 'input_image.jpg' using the cv2.imread() function.

3. The fourth line converts the image to grayscale using the cv2.cvtColor() function. This is necessary because the Laplacian operator operates on grayscale images.

4. The fifth line applies the Laplacian operator to the grayscale image using the cv2.Laplacian() function. The second argument (cv2.CV_64F) specifies the data type of the output image.

5. The sixth line normalizes the output of the Laplacian operator to the range 0-255 using the cv2.normalize() function. This is done to ensure that the output image has values in the range that can be displayed as an 8-bit image.

6. The seventh line displays the smoothed image in a window with the title 'Smoothed Image' using the cv2.imshow() function.

Finally, we combine the output of Laplacian smoothening along with the contrast-enhanced version using the following code:

```
alpha = 0.7
output = (1-alpha) * img_dhe + alpha * laplacian
plt.imshow(output, 'gray')
plt.title('Smoothened DHE')
plt.show()
```

Here alpha = 0.7 determines the smoothing coefficient. So, after performing all the steps here are the result of original image, dynamic contrast-enhanced, and smoothened version:

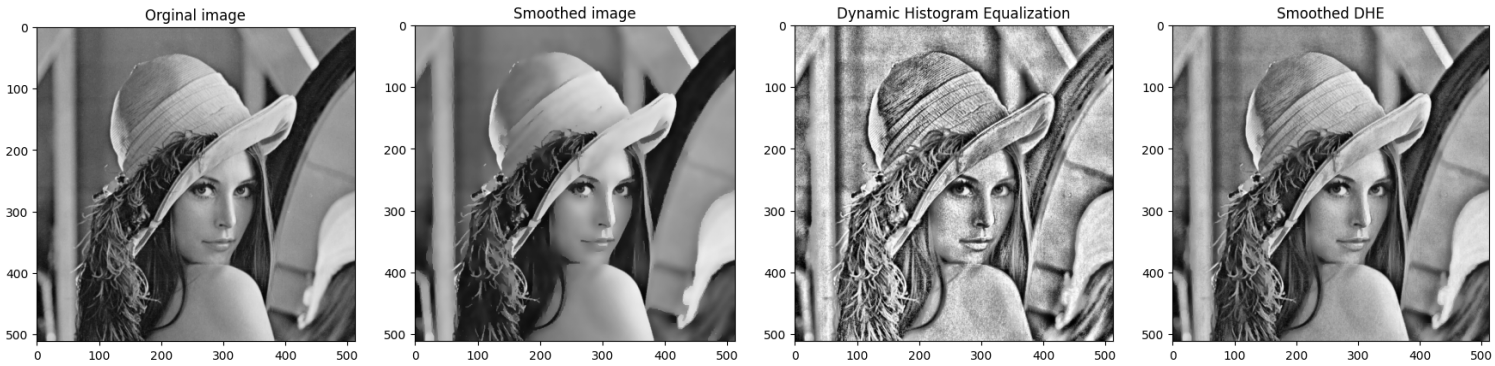

Figure: Original image, smoothened version using Laplacian, Dynamic Histogram Equalization result and smoothened DHE result.

Following the same steps, we have the other following outputs with different values of radius of circle.

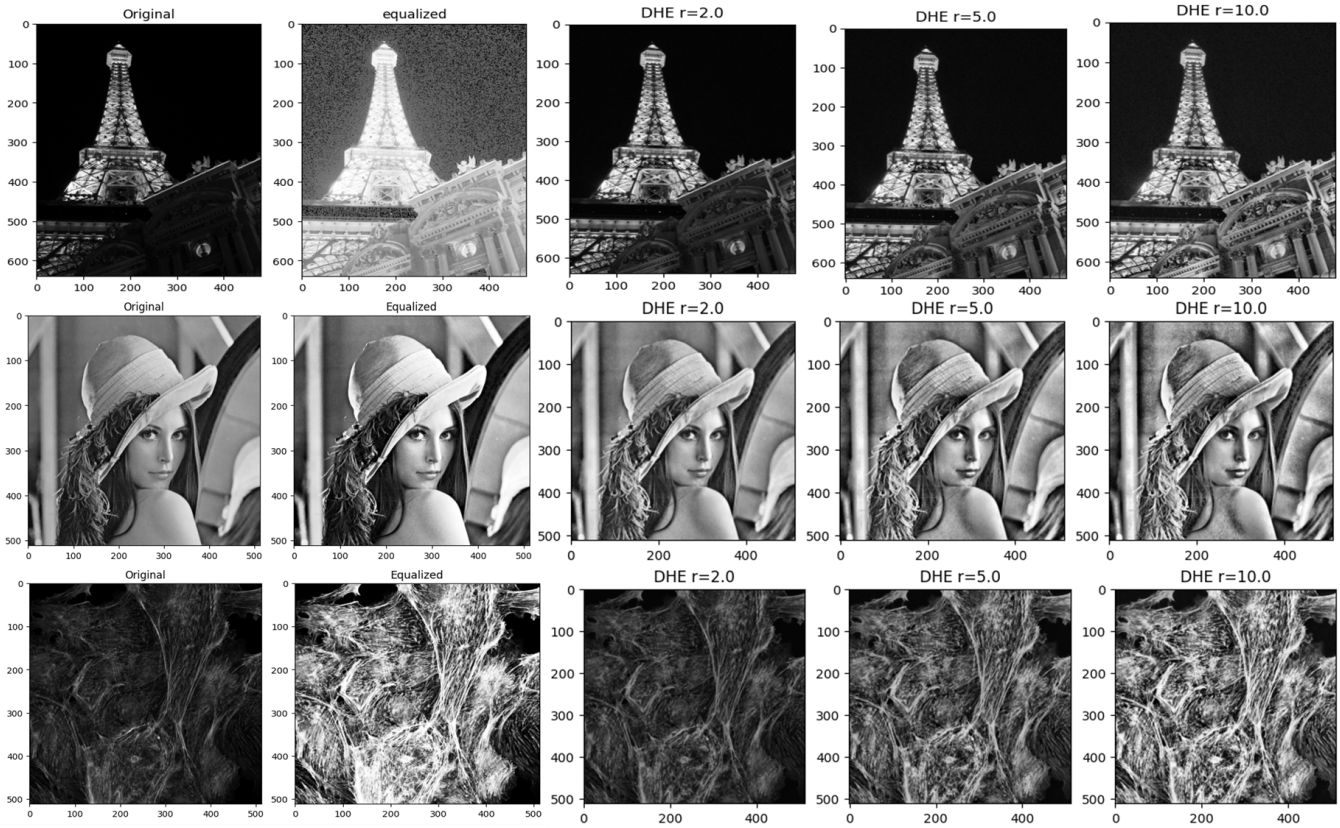

Figure: Image, traditional GHE, and dynamic histogram equalization with multiple neighborhood radius ® values.

Extension to Color Space:

Further, we extend the experimentation to color images also. Our idea is to transform the image from RGB to HSV space, like this:

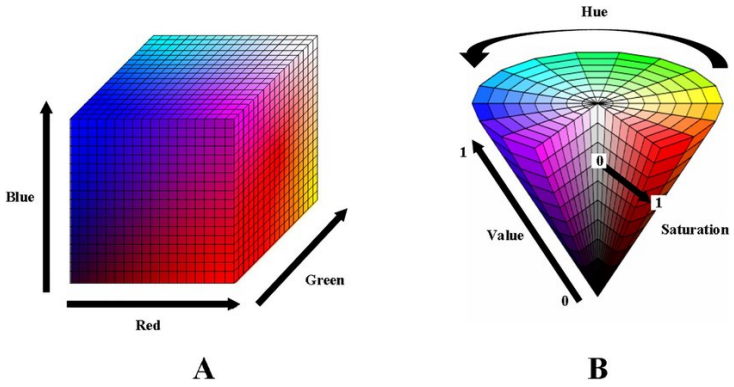

Figure: Visualization of RGB and HSV space.

HSV (Hue-Saturation-Value) is a color space that is widely used in image processing and computer vision applications. The HSV color space is designed to represent colors in a way that is more intuitive and perceptually meaningful than other color spaces such as RGB or CMYK.

In the HSV color space, a color is represented as a combination of three values:

1. **Hue**: Hue is the attribute that distinguishes one color from another, based on the dominant wavelength of the light. Hue is represented as an angle in degrees, with 0 degrees corresponding to red, 60 degrees to yellow, 120 degrees to green, 180 degrees to cyan, 240 degrees to blue, 300 degrees to magenta, and 360 degrees back to red.

2. **Saturation**: Saturation is the intensity or purity of the color. A fully saturated color is one where the hue is at its maximum and the value is at its maximum. In other words, the color is as intense as possible. A desaturated color is one where the saturation is zero, resulting in a grayscale color.

3. **Value**: Value is the brightness or luminance of the color. It determines how much light is reflected by the color. A value of 0 corresponds to black, while a value of 1 corresponds to white.

Together, these three values define a color in the HSV color space. Unlike other color spaces, such as RGB, where colors are defined as a combination of red, green, and blue values, the HSV color space separates the color information into these three perceptually meaningful attributes. This makes it easier to manipulate colors in image processing tasks, such as adjusting the brightness, contrast, or color balance of an image.

So, we perform DHE on the V channel of the HSV image. We then replace the old V space with the transformed V-space and then bring the transformed HSV image to RGB space. The implementation looks like this:

```
def equalize_dhe_color_hsv(img):
    """Equalize the image splitting it after conversion to HSV and applying DHE
    to the V channel and merging the channels and convert back to BGR
    """

    dhe = cv2.createCLAHE(clipLimit=4.0)
    H, S, V = cv2.split(cv2.cvtColor(img, cv2.COLOR_BGR2HSV))
    eq_V = dhe.apply(V)
    eq_image = cv2.cvtColor(cv2.merge([H, S, eq_V]), cv2.COLOR_HSV2BGR)
    return eq_image
```

So, we have the following output for this implementation:

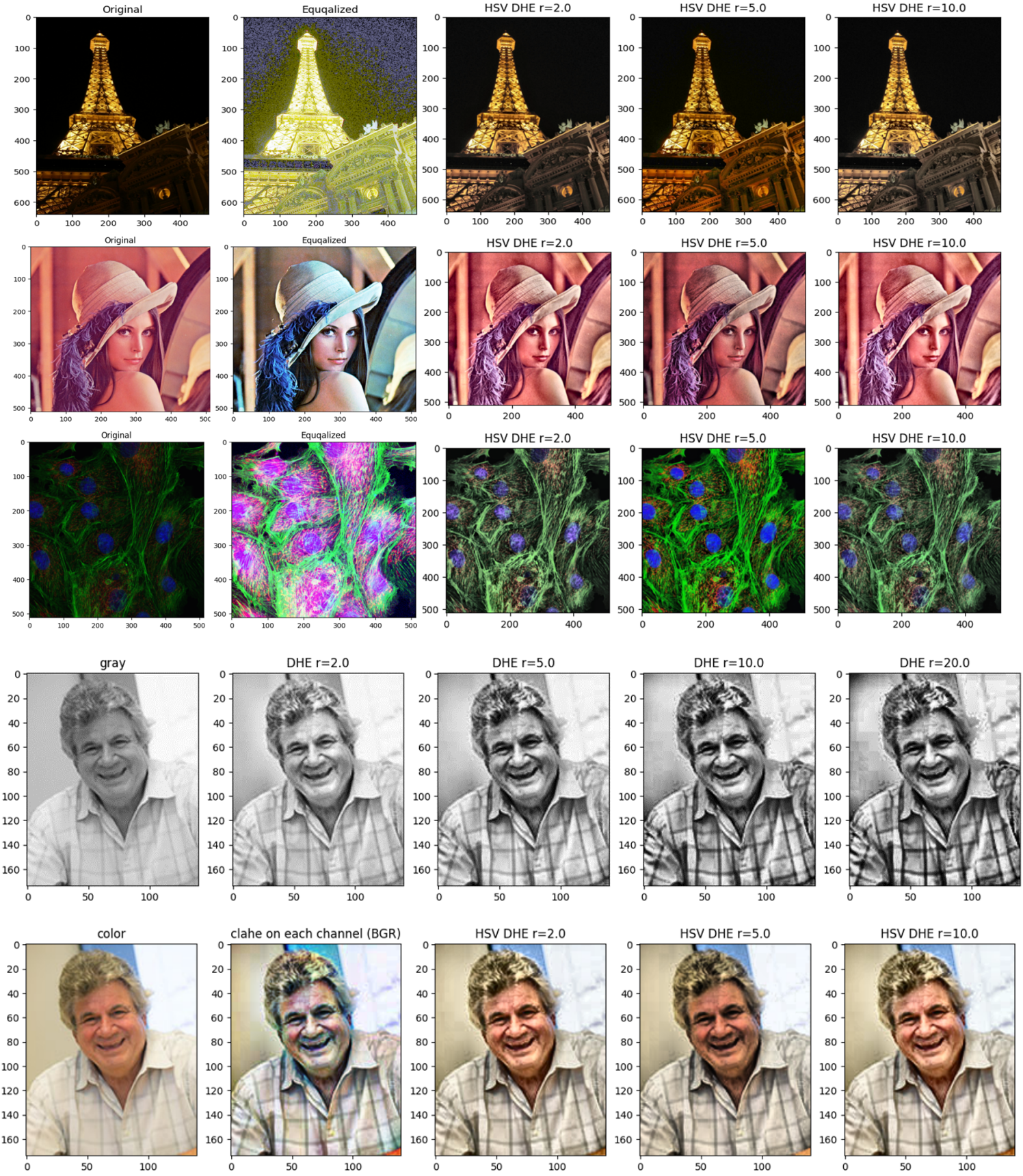

Figure: Visualization of images, normal 3-channel histogram equalization result, and DHE on HSV space with multiple neighborhood radius (r) values.

Conclusion:

Dynamic Histogram Equalization (DHE) is a variation of the traditional Histogram Equalization (HE) technique that is commonly used in image processing and computer vision applications. While DHE has several advantages over traditional HE, it also has some disadvantages that should be taken into account when deciding whether to use it in a particular application.DHE can produce intensity inconsistencies in an image, resulting in a non-uniform appearance. This is because DHE does not take into account the global intensity of an image, which can lead to the over-enhancement of certain regions. It can amplify noise in an image, especially in regions with low contrast. This can result in an increase in noise artifacts and a reduction in image quality. Moreover, it can be computationally intensive, especially for large images or when used with high-resolution cameras. This can result in longer processing times and increased resource requirements.

Part 2: Color Histogram Equalization

Introduction

Color histogram equalization is a technique used to adjust the contrast and brightness of an image by redistributing the pixel values in its color channels. The purpose of the color histogram equalization is to improve the visibility of the image details that may be difficult to see due to low contrast or brightness. Hence, when color histogram equalization is performed, the images' contrast and brightness improves which helps to view these fine image details clearly. Traditionally, color histogram equalization is performed by calculating the histogram of each color channel (such as red, green, and blue) separately and then stretching the histogram to increase the contrast of the image. The stretching is done by mapping the minimum and maximum pixel values of the histogram to the minimum and maximum pixel values of the output image. Ultimately, the three channels after performing histogram equalization are merged to get the equalized output image.

Figure: Original image and resultant image after traditional color histogram equalization. Clearly, this points out the drawbacks of traditional CHE.

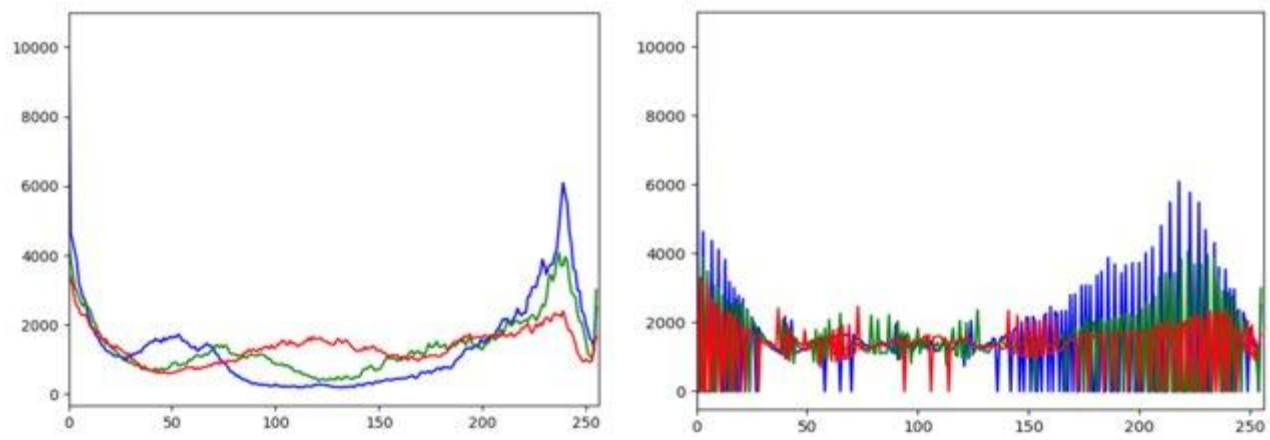

Figure: Histograms of three channels before and after equalization on three channels independently

From the above plots, we can see that the histograms of three channels are calculated and equalized separately. We can also observe that there is an unnatural color contrast in the resulting equalized image. Thus, we understand that there are certain drawbacks for the traditional color histogram equalization method.

Why is Traditional CHE not the best solution?

The traditional color histogram equalization (CHE) has certain drawbacks as mentioned below:

1. Performing CHE on each of the three color channels separately can lead to an image with unnatural colors and contrast.
2. This technique doesn't take into account the correlation between different color channels.
3. If a particular color is dominant in one channel, equalizing that channel can lead to a shift in overall color balance.
4. Also, some image features can be better represented in a particular color channel. For example, fine details can be better represented in the blue channel while skin tone can be better represented in the red channel. Equalizing all the channels individually can result in loss of these important features.
5. Hence, there is a need for an approach that preserves the color balance of the image while improving overall contrast.

On the brighter note, Color Histogram Equalization using Mesh Deformation addresses all the above mentioned drawbacks. This method aims to improve the quality of color images by adjusting the distribution of color values in an image. The proposed method uses a mesh deformation technique to modify the image's histogram while preserving its color information.

Idea behind CHE using Mesh Deformation

The idea is to deform a mesh in color space to fit the existing histogram and then map it to a uniform histogram. So, the initial histogram is calculated for the input image. A NxN mesh is deformed onto the colorspace to fit this initial histogram such that the number of pixels in all the cells of the grid are equal. This deformed mesh is mapped to an uniform regular mesh i.e., onto a uniform output histogram. From this histogram, an output image is computed which will be equalized.

Methodology

Given a vector-valued image I = [I_1,..., I_n] and its histogram $H_I(i_1,..., i_n) : R^n -> R$ is defined as a relative frequency of occurrence of color [i_1,..., i_n].

Now, we want to create a transform T T : R^n –> R^n such that J(x,y) = T(I(x,y)) is equalized i.e., $H_J(i_1,\ldots, i_n)$ is constant i.e., pixels across various color intensities are constant.

Now, we construct some equations which will be useful in:

1. Mapping a mesh M_I to the uniform regular mesh M_J
2. To construct suitable mesh on the initial histogram or the initial color space.

For creating a mapping between the cells of the mesh M_I to the uniform regular mesh M_J, we need to find the transform equation T. Good way to describe T is the way it maps from one mesh to another. It is possible to construct a piecewise linear approximation T_{PW} of T by considering that all the triangular half cells of vertices p_I, q_I, r_I will be linearly mapped to the corresponding triangle of vertices $p_J = T(p_I)$, $q_J = T(q_I)$ and $r_J = T(r_I)$.

Hence, according to barycentric coordinates, we can define T_{PW} as:

$$T_{PW}(c_I) = \alpha p_J + \beta q_J + \gamma r_J \quad \text{–>} \quad \text{(equation 1)}$$

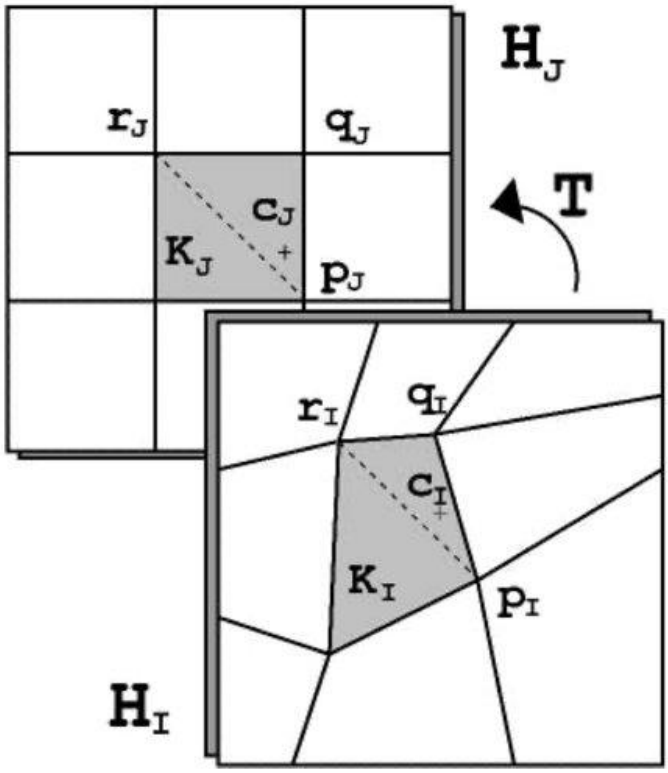

Figure: The transformation T can be represented by a mesh

Background:

Consider a pixel (x,y). Its color is a vector $c_I = I(x,y)$. M_I is a mesh of color space, c_I is within a certain cell K_I. By definition of M_I and M_J, all points c_I of K_I are mapped to points $c_J = T(c_I)$ of the cell $K_J = T(K_I)$ of M_J through transform function T. J is an equalized image and M_J is uniform, therefore, all cells of M_J contain exactly the same number of points. Thus, this must hold for M_I as well.

Based on the above assumption, we find a suitable mesh M_I. We start with NxN mesh, we converge on a solution iteratively by,

1. Expanding all the cells that don't contain enough points i.e., if this condition $(\frac{n_X n_Y}{N^2} - n_{K_I}) > 0$ is satisfied.
2. Contract cells that already contain too many points i.e., if this condition $(\frac{n_X n_Y}{N^2} - n_{K_I}) < 0$ is satisfied.

This is done by moving the vertices p_I, q_I, r_I away or towards the centroid o_{K_I}. Thus, this process is replicated by the following equation:

$$\frac{\partial p_I}{\partial t} = (\frac{n_X n_Y}{N^2} - n_{K_I})(p_I - o_{K_I}) \quad \text{->} \quad \text{(equation 2)}$$

Where $n_X n_Y$ are the dimensions of the image, NxN is the size of the grid, n_{K_I} is the number of pixels of the image whose colors are within the cell K_I.

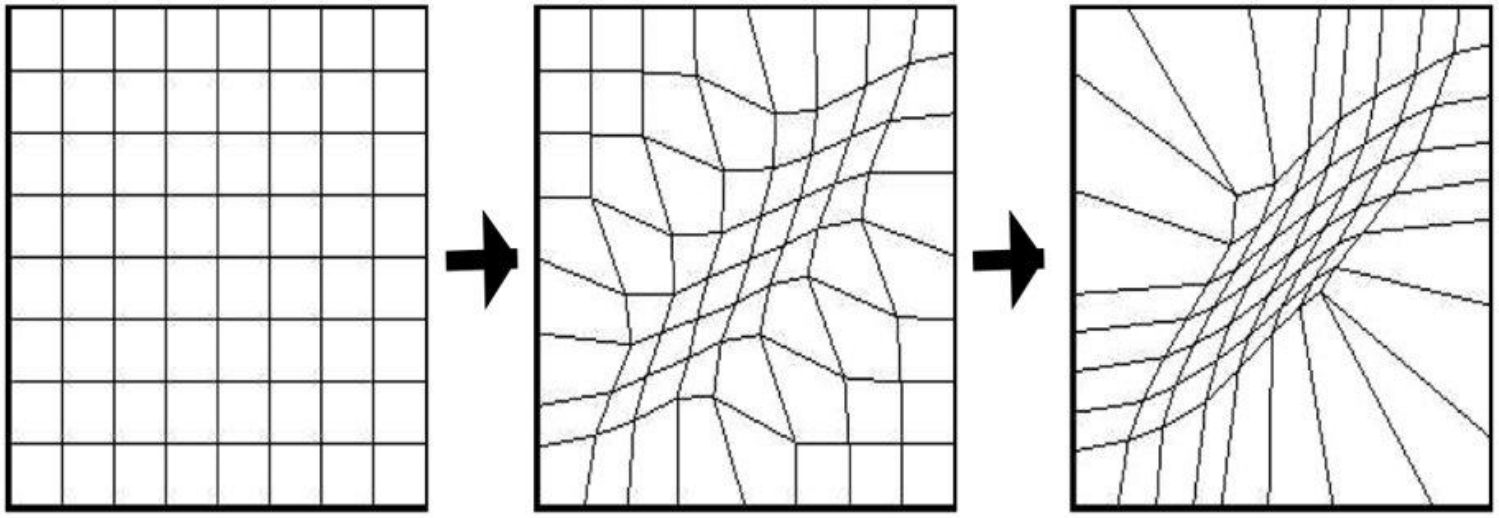

Figure: Mesh deforming to fit histogram on some figure

At steady state,

$$\frac{n_X n_Y}{N^2} = n_{K_I} = \text{constant}$$

It means that all the cells contain the same number of points.

Algorithm

1. Determine color histogram H_I of the original image I.
2. Deform a mesh M_I on the color space to fit the histogram H_I based on equation 2.
3. Use M_I to define a piecewise linear deformation T_{PW} of color space. All cells of M_I are linearly mapped to corresponding cells of uniform mesh M_J based on equation 1.
4. Compute equalized image J = T_{PW}(I)

Implementation and Results

Here is the implementation of the algorithm:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from scipy import interpolate

def mesh_deformation(img, mesh_rows=256, mesh_cols=256, num_iterations=100):
    # Resize the input image to match the desired mesh size
    img = cv2.resize(img, (mesh_cols, mesh_rows), interpolation=cv2.INTER_LINEAR)

    # Convert the input image to LAB color space
    lab_img = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

    # Initialize the mesh coordinates
    rows, cols, _ = lab_img.shape
    # print(rows,cols)
    x_coords = np.linspace(0, cols, mesh_cols+1, dtype=np.float32)
    y_coords = np.linspace(0, rows, mesh_rows+1, dtype=np.float32)
    x_mesh, y_mesh = np.meshgrid(x_coords, y_coords)
    mesh = np.dstack([x_mesh, y_mesh])

    luminance = lab_img[:,:,0].astype(np.float32) / 255.
    sobel_x = cv2.Sobel(luminance, cv2.CV_32F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(luminance, cv2.CV_32F, 0, 1, ksize=3)

    grad_mag = np.sqrt(sobel_x**2 + sobel_y**2)
    grad_x = sobel_x / (grad_mag + 1e-8)
    grad_y = sobel_y / (grad_mag + 1e-8)

    # Perform mesh deformation to achieve contrast enhancement
    for iteration in range(num_iterations):
        # Compute the average color value in each mesh region
        avg_colors = np.zeros((mesh_rows, mesh_cols, 3), dtype=np.float32)
        for i in range(mesh_rows):
            for j in range(mesh_cols):
                x_start, y_start = mesh[i,j]
                x_end, y_end = mesh[i+1,j+1]
                                avg_colors[i,j]  =  np.mean(lab_img[int(y_start):int(y_end),
int(x_start):int(x_end)], axis=(0,1))



          # Update the mesh coordinates based on the average color values and gradient
direction
```

```
        mesh_step_size = 0.3
        mesh_delta = np.zeros((mesh_rows+1, mesh_cols+1, 2), dtype=np.float32)
        for i in range(mesh_rows+1):
            for j in range(mesh_cols+1):
                if i > 0 and j > 0 and i < mesh_rows and j < mesh_cols:
                                grad_direction  =  np.arctan2(grad_y[int(mesh[i-1,j-1][1]),
int(mesh[i-1,j-1][0])],
                                                              grad_x[int(mesh[i-1,j-1][1]),
int(mesh[i-1,j-1][0])])
                        mesh_delta[i,j] = mesh_step_size * np.array([np.cos(grad_direction),
np.sin(grad_direction)])
                else:
                    mesh_delta[i,j] = [0, 0]
        mesh += mesh_delta

    # Warp the input image based on the updated mesh coordinates
    out_img = cv2.remap(img, mesh[:,:,0], mesh[:,:,1], cv2.INTER_LINEAR)

    # Convert the output image to LAB color space
    out_lab_img = cv2.cvtColor(out_img, cv2.COLOR_BGR2LAB)


    # Convert the output image back to BGR color space
    out_img = cv2.cvtColor(out_lab_img, cv2.COLOR_LAB2BGR)

    return out_img


# Load the input image
img = cv2.imread('color_2.png')

# Apply color histogram equalization through mesh deformation
out_img = mesh_deformation(img)


cv2.imshow('Input Image', img)
cv2.imshow('Equalized Image', out_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Here is a step-by-step explanation of the code:

1. Compute the color histogram of the input image.
2. Generate a mesh grid over the image, which will be used to deform the image's histogram.
3. Deform the mesh grid using a specified deformation function we make sure that the number of pixels in each cell of the grid will be constant.

4. Update the mesh coordinates based on the average color values and gradient direction using nested for loops and np.arctan2() function.
5. Warp the input image based on the updated mesh coordinates using cv2.remap() function.
6. Using the transform equation, create a mapping between the input and output mesh.
7. Compute the new color histogram based on the uniform regular output mesh.
8. Map the color values of the modified image to obtain the final output image.

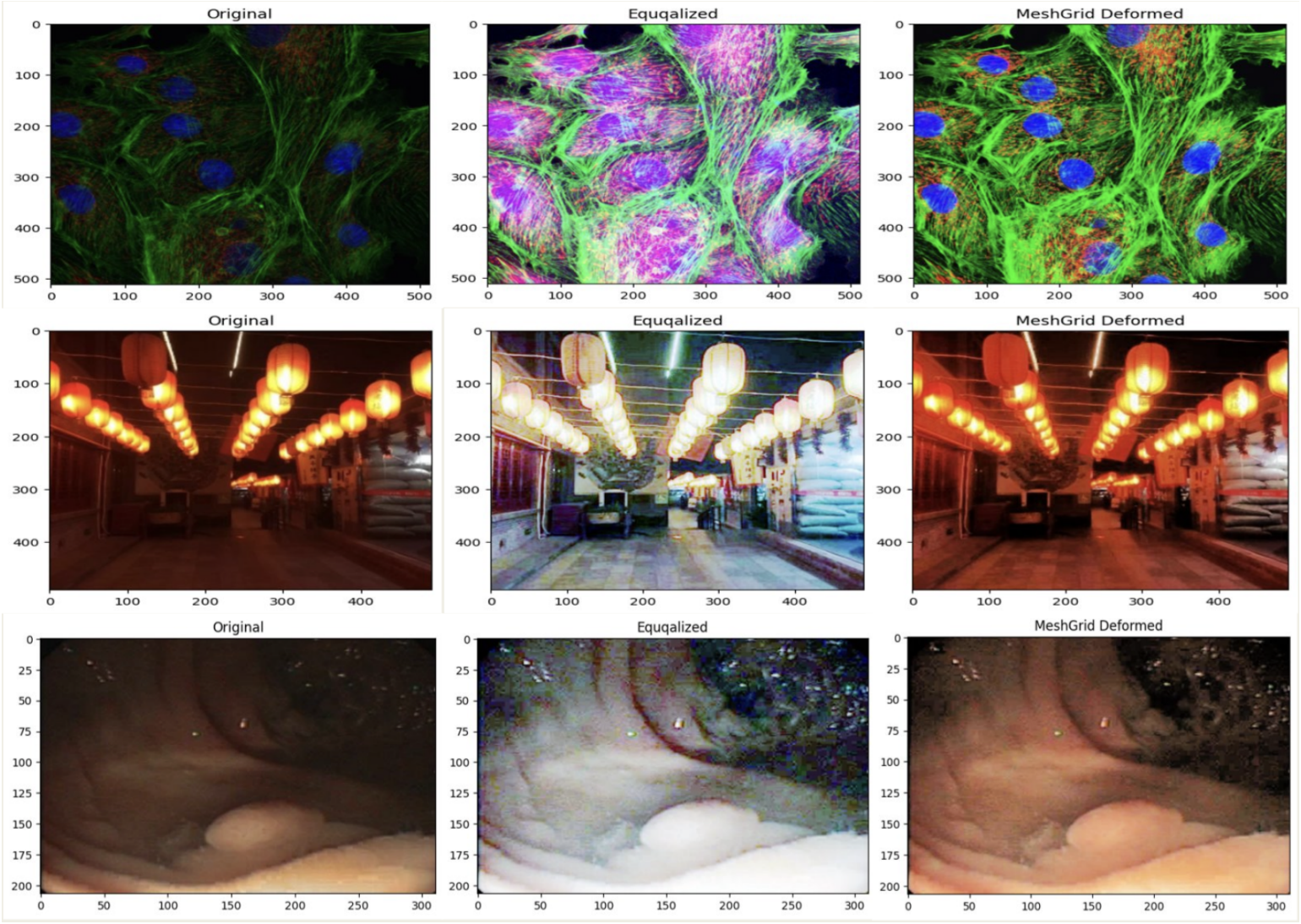

Figure: Original image, normal 3-channel histogram equalization result, and mesh deform CHE result

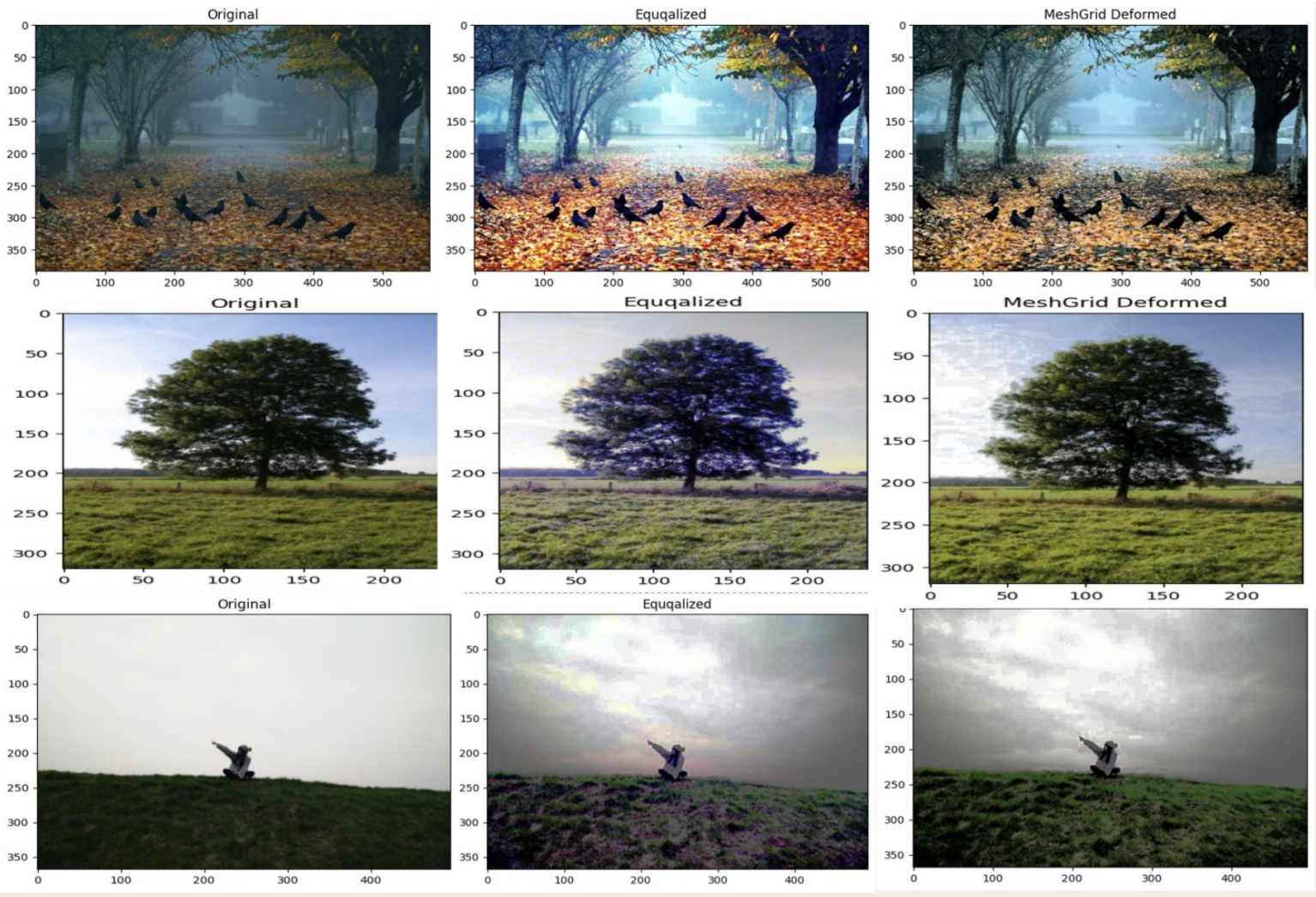

Figure: Original image, normal 3-channel histogram equalization result, and mesh deform CHE result

Conclusion

Color Histogram Equalization using Mesh Deformation has several advantages like increased flexibility, better preservation of local features and more accurate histogram equalization. Higher flexibility in the equalization process is achieved by enabling the user to specify a custom deformation function that can be used to warp the image in a non-linear way too. This can be especially useful in cases where the input image has a complex or non-uniform distribution of pixel intensities. Also, since mesh deformation can be used to selectively warp different regions of the image in different ways, it can be more effective at preserving local features such as edges, corners, and other salient image features. This is particularly important in cases where such features are critical to the interpretation or analysis of the image. Overall, mesh deformation provides a powerful and flexible tool for color histogram equalization that can be tailored to the specific needs and requirements of different applications and use cases. By allowing for greater customization and flexibility in the equalization process, mesh deformation can help to achieve better results and preserve important image features more effectively.