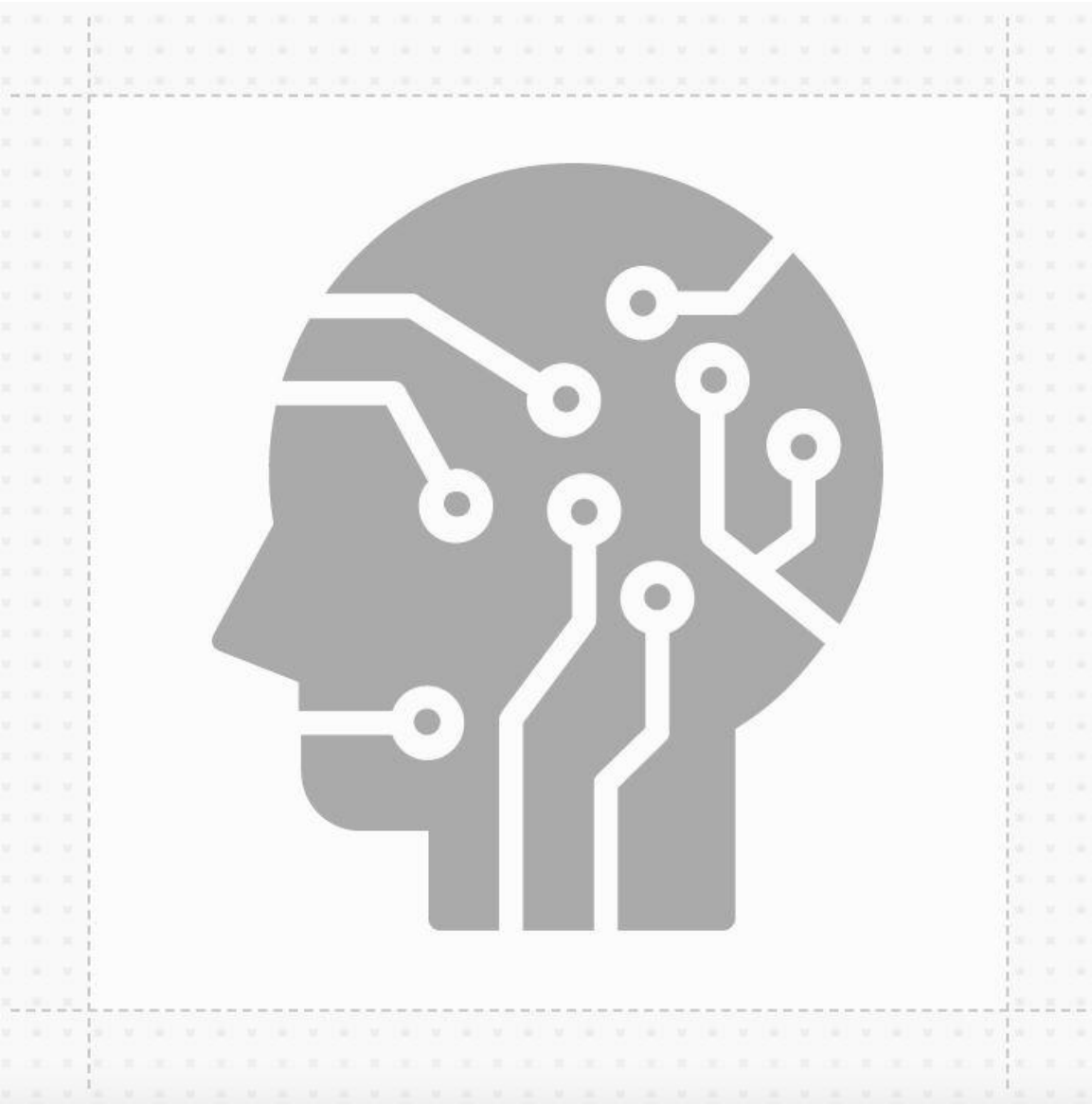


Image Processing project

FEE -CSE DEPT.

Ola E-Shiekh | DIP | May 23, 2023

AI-ROS

OLA EL-SHIEKH

Digital Image processing project

Digital Image Processing (DIP) is a rapidly evolving field that specializes in developing algorithms to improve the quality and clarity of digital images. One of the key objectives in DIP projects is image enhancement, where the goal is to apply various techniques to improve visual quality, enhance details, and make images more visually appealing.

In this project, I will focus on image enhancement and denoising algorithms. Image enhancement techniques involve adjusting brightness, contrast, and color balance to enhance the overall appearance of an image. Additionally, I will explore advanced algorithms such as histogram equalization, adaptive filtering, and contrast stretching, which aim to improve specific aspects of an image, such as enhancing details in low-contrast regions or reducing noise artifacts.

Denoising algorithms, on the other hand, are designed to remove unwanted noise from images. Noise can degrade the visual quality and make it difficult to extract useful information from the image. I will investigate various denoising techniques, such as spatial filters, frequency domain filters, and wavelet-based methods, to effectively reduce noise while preserving important image details.

Throughout the project, I will implement and evaluate these image enhancement and denoising algorithms on different types of images, including photographs, medical scans, or digital artwork. By experimenting with these algorithms, I aim to gain a deeper understanding of their strengths, limitations, and their impact on image quality.

Ultimately, this project will equip me with practical skills in implementing and evaluating image enhancement and denoising algorithms. By mastering these techniques, I will be able to contribute to the field of Digital Image Processing by improving image quality, enabling better visual analysis, and facilitating image-based decision-making in various domains.

For this Project, I will Enhance some photos by denoising them. I'm not going to use the built-in functions. I will build a user defined functions for the mentioned functionalities :

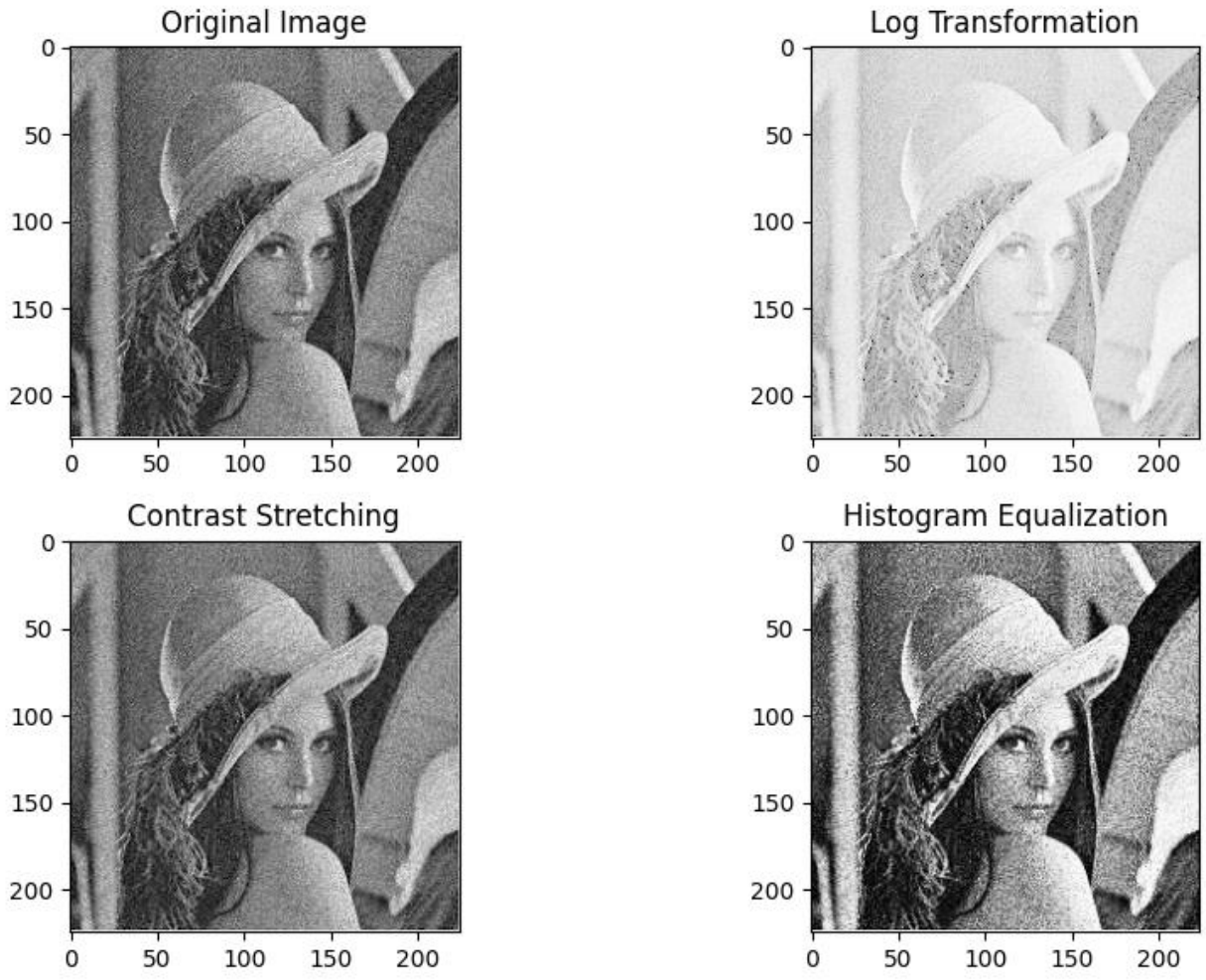
- Log transformation
- Contrast Stretching
- Histogram Equalization

1- Using the built- in functions:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('download.jpeg', 0)  # Load as grayscale

# Log Transformation
log_image = np.log1p(image)

# Contrast Stretching
stretched_image = cv2.normalize(image, None, 0, 255, cv2.NORM_MINMAX)

# Histogram Equalization
equalized_image = cv2.equalizeHist(image)

# Display the images
plt.figure(figsize=(10, 6))

plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')

plt.subplot(2, 2, 2)
plt.imshow(log_image, cmap='gray')
plt.title('Log Transformation')

plt.subplot(2, 2, 3)
plt.imshow(stretched_image, cmap='gray')
plt.title('Contrast Stretching')

```

```
32. plt.subplot(2, 2, 4)
33. plt.imshow(equalized_image, cmap='gray')
34. plt.title('Histogram Equalization')
35.
36. plt.tight_layout()
37. plt.show()
```

Applying this code will result on Enhancing the Images by denoising them



But this results is out of using the built-in functions of python. In the upcoming steps I will implement these function using python without using any built-in functions. Just pure programming and Mathmatics.

2- Applying the Log transformation manually

In this code, we manually apply the log transformation to each pixel of the image. First, we create an output image log_image with the same size and data type as the original image, initialized with zeros.

We calculate the scaling factor c based on the maximum pixel value in the original image. Then, we iterate over each pixel in the image and compute the log transformation using the formula c * np.log(1 + pixel_value).

Note that we add 1 to the pixel value before taking the logarithm to avoid errors when dealing with values close to zero.

After applying the transformation, we convert the log_image array to the uint8 data type to ensure it is in the range [0, 255] for proper display.

Finally, we display the original image and the log-transformed image using matplotlib.

```
log_image = np.zeros_like(image, dtype=np.float32)
c = 255 / np.log(1 + np.max(image))  # Scaling factor

for i in range(image.shape[0]):
    for j in range(image.shape[1]):
        log_image[i, j] = c * np.log(1 + image[i, j])

log_image = np.uint8(log_image)

# Display the images
plt.figure(figsize=(10, 6))

plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
```

```
15. plt.title('Original Image')
16.
17. plt.subplot(1, 2, 2)
18. plt.imshow(log_image, cmap='gray')
19. plt.title('Log Transformation')
20.
21. plt.tight_layout()
22. plt.show()
23.
```

And this is result of applying that code on the example image we have

3- Applying the Contrast Stretch manually

In this code, we manually apply the contrast stretching operation to each pixel of the image. First, we create an output image stretched_image with the same size and data type as the original image, initialized with zeros.

We calculate the minimum and maximum pixel values in the original image using np.min and np.max functions, respectively. Then, we define the desired output range as a (minimum value) and b (maximum value), typically 0 and 255 for an 8-bit image.

Next, we iterate over each pixel in the image and apply the contrast stretching formula:

stretched_pixel = (pixel_value - min_val) * ((b - a) / (max_val - min_val)) + a

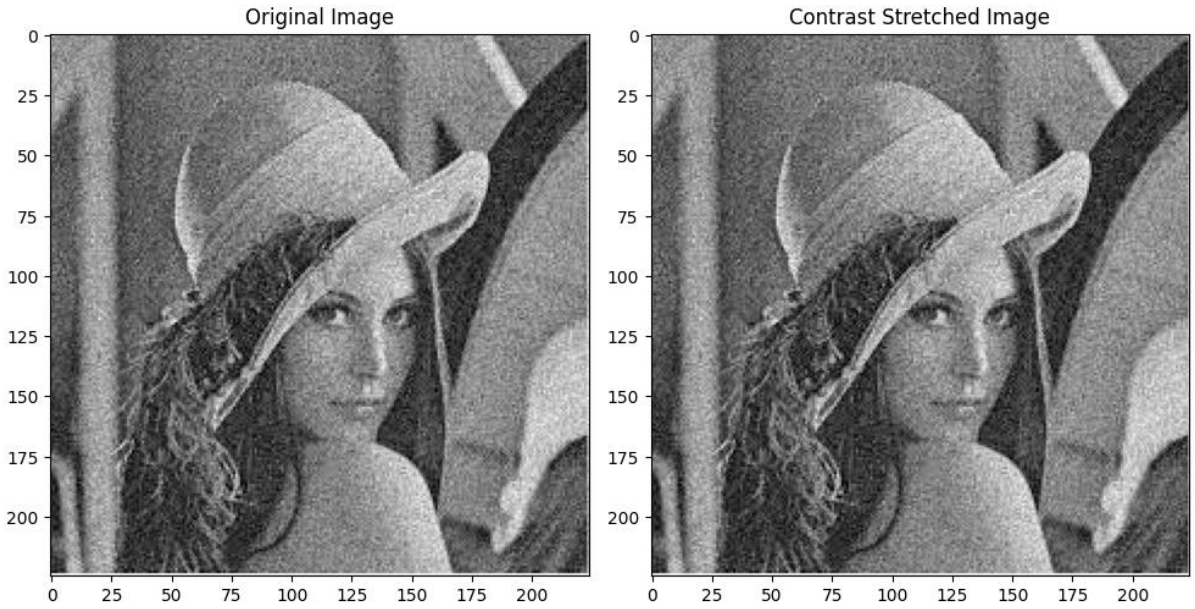
This formula linearly maps the input range [min_val, max_val] to the desired output range [a, b].

Finally, we display the original image and the contrast-stretched image using matplotlib.

```
# Contrast Stretching
stretched_image = np.zeros_like(image, dtype=np.uint8)

# Calculate minimum and maximum pixel values
min_val = np.min(image)
max_val = np.max(image)

# Calculate scaling factors
a = 0
b = 255

# Apply contrast stretching
for i in range(image.shape[0]):
    for j in range(image.shape[1]):
```

```
15.         stretched_image[i, j] = int((image[i, j] - min_val) * ((b - a) /
(max_val - min_val)) + a)
16.
17. # Display the images
18. plt.figure(figsize=(10, 6))
19.
20. plt.subplot(1, 2, 1)
21. plt.imshow(image, cmap='gray')
22. plt.title('Original Image')
23.
24. plt.subplot(1, 2, 2)
25. plt.imshow(stretched_image, cmap='gray')
26. plt.title('Contrast Stretched Image')
27.
28. plt.tight_layout()
29. plt.show()
30.
```

And this is result of applying that code on the example image we have

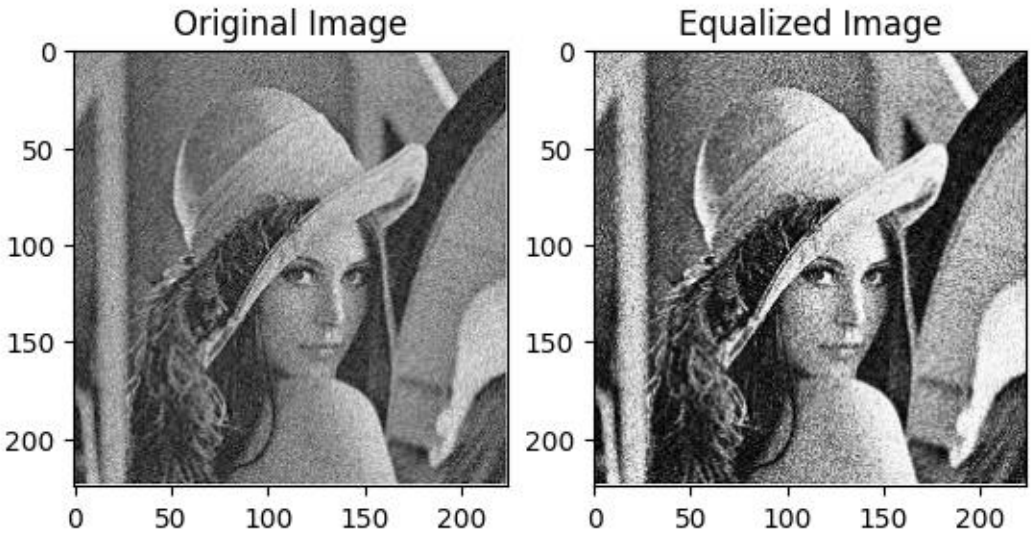


4- Applying the Histogram Equalization algorithm manually

This implementation calculates the histogram, CDF, and performs the equalization manually without relying on any built-in functions. It demonstrates the underlying process of histogram equalization by accessing the pixel intensities and performing the necessary calculations step by step.

```
#Calculate the histogram of the image:
hist, bins = np.histogram(image.flatten(), 256, [0,256])

#Calculate the cumulative distribution function (CDF) of the histogram:
cdf = hist.cumsum()
cdf_normalized = cdf * hist.max() / cdf.max()

#Calculate the equalized histogram:
equalized_image = np.interp(image.flatten(), bins[:-1], cdf_normalized)
equalized_image = equalized_image.reshape(image.shape)

#Display the original and equalized images:
plt.subplot(121)
plt.imshow(image, cmap='gray')
plt.title('Original Image')

plt.subplot(122)
plt.imshow(equalized_image, cmap='gray')
plt.title('Equalized Image')

plt.show()

```

And this is result of applying that code on the example image we have



Project Summary

After implementing the log transform, contrast stretching, and histogram equalization techniques manually in my Digital Image Processing (DIP) project, I have gained valuable insights into image enhancement algorithms and their impact on image quality.

- The log transform is a powerful technique that enhances the details in low-intensity regions of an image while compressing the high-intensity regions. By applying the log transform, I was able to improve the visibility of darker areas and enhance the overall dynamic range of the image.
- Contrast stretching, another image enhancement technique, aims to increase the contrast between different intensity levels in an image. By stretching the intensity values to cover a wider range, I successfully enhanced the image's visual appearance and made the details more distinguishable.
- Histogram equalization, a popular technique in image processing, helps in redistributing the intensity levels of an image to achieve a more balanced histogram. By equalizing the histogram, I effectively improved the overall contrast and made the image appear more vibrant and visually appealing.

Throughout the implementation of these techniques, I encountered challenges and made observations regarding their limitations. For instance, the log transform may introduce artifacts in high-intensity regions, leading to a loss of detail. Similarly, contrast stretching can lead to over-enhancement or noise amplification if not carefully controlled.

By manually implementing these image enhancement techniques, I gained a deeper understanding of their underlying principles and the trade-offs involved. I also learned about the importance of parameter selection and the need for careful evaluation to achieve the desired results.

Moving forward, I will leverage this knowledge and experience to explore additional image enhancement techniques and further refine the implemented algorithms. By continuing to refine and evaluate these techniques, I aim to contribute to the advancement of image processing and its applications in various domains.

This is a document reporting all the steps I implanted in this DIP project

You can find the source code I wrote in this colab notebook :

https://colab.research.google.com/drive/1SP2U9wb2jlx7rmflUfEuC4exqakiL6jU?usp=sharing

and on github via this link :

https://github.com/olaelshiekh/DIP_project.git

Under supervision of Dr. Mohamed Azzam.

AI-ROS

OLA EL-SHIEKH