Scenario no 01 : Medical Image Analysis (X-ray & MRI Enhancement)

Doctors rely on **X-ray and MRI scans** to diagnose diseases, but sometimes these images have **low contrast** or **noise** that makes it difficult to detect abnormalities. Your task is to apply **image enhancement** techniques to improve visibility.

Tasks:

1. Compute mean and variance to analyze brightness and contrast of the image.
2. Apply log and inverse log transformation to enhance faint details.
3. Use contrast stretching to improve bone or tissue structures.
4. Use thresholding to segment bones from the background.

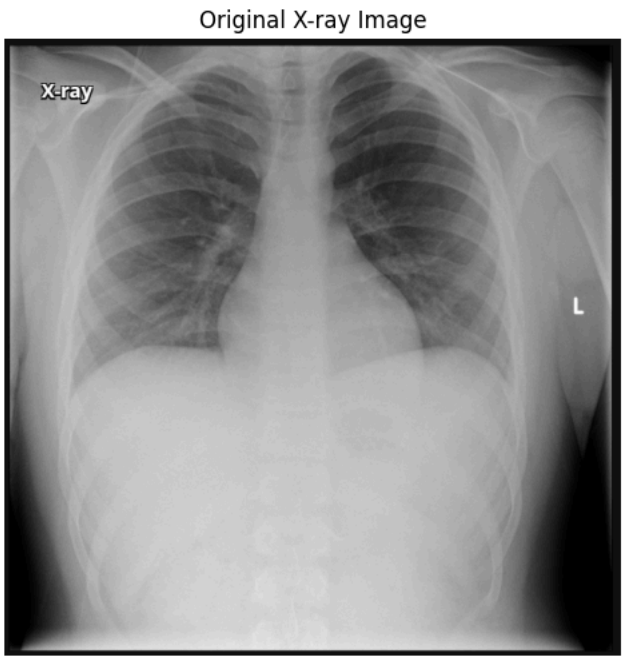
Hint:

- Low variance = low contrast (image needs enhancement).
- Log transformation highlights low-intensity pixels (useful for MRI scans).
- Power-law transformation (gamma correction) can adjust brightness dynamically.

Step 1: Load and Display the Image

```
import cv2
from google.colab.patches import cv2_imshow
import numpy as np
import matplotlib.pyplot as plt
```

```
# Load the image
image = cv2.imread('Scanerio 01.png', cv2.IMREAD_GRAYSCALE)

# Display the original image
plt.figure(figsize=(6,6))
plt.imshow(image, cmap='gray')
plt.title("Original X-ray Image")
plt.axis('off')
plt.show()
```



Step 2: Compute Mean and Variance for Brightness and Contrast Analysis

```
# Compute mean and variance
mean_intensity = np.mean(image)
variance_intensity = np.var(image)

print(f"Mean Intensity: {mean_intensity:.2f}")
print(f"Variance Intensity: {variance_intensity:.2f}")
```

```
Mean Intensity: 138.44
Variance Intensity: 3417.82
```

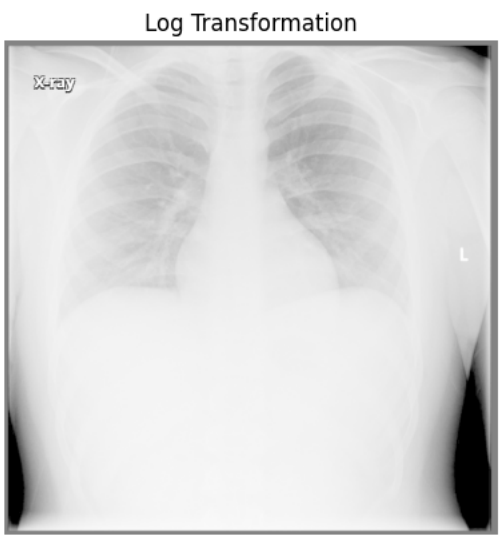
High variance means the image has high contrast and doesnt requires enhancement

Step 3: Apply Log and Inverse Log Transformations

Log transformation enhances low-intensity details.

Log transformation

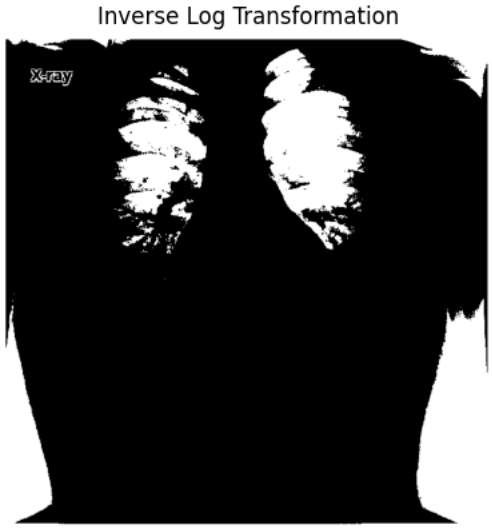
```
c = 255 / np.log(1 + np.max(image))  # Compute scaling factor
log_transformed = c * np.log(1 + image.astype(np.float32))
log_transformed = np.array(log_transformed, dtype=np.uint8)

plt.imshow(log_transformed, cmap='gray')
plt.title("Log Transformation")
plt.axis('off')
plt.show()
```



Inverse log transformation

```
inverse_log = np.exp(image.astype(np.float32)) - 1
inverse_log = cv2.normalize(inverse_log, None, 100, 255, cv2.NORM_MINMAX)
inverse_log = np.array(inverse_log, dtype=np.uint8)

plt.imshow(inverse_log, cmap='gray')
plt.title("Inverse Log Transformation")
plt.axis('off')
plt.show()
```

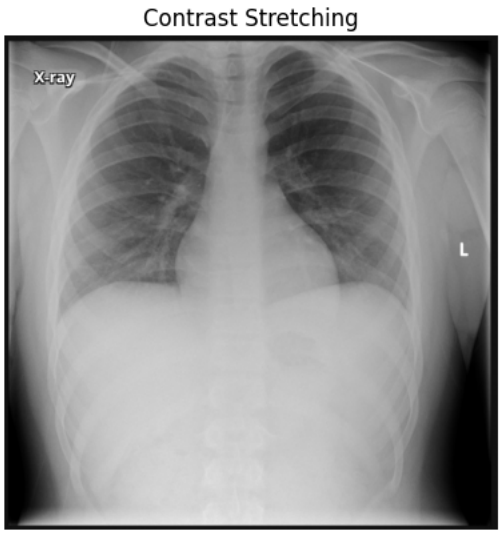
```
<ipython-input-10-413023859ef7>:1: RuntimeWarning: overflow encountered in exp
  inverse_log = np.exp(image.astype(np.float32)) - 1
<ipython-input-10-413023859ef7>:3: RuntimeWarning: invalid value encountered in cast
  inverse_log = np.array(inverse_log, dtype=np.uint8)
```



Step 4: Contrast Stretching

This enhances details by scaling pixel intensities.

```
min_val = np.min(image)
max_val = np.max(image)

contrast_stretched = (image - min_val) * (255.0 / (max_val - min_val))
contrast_stretched = np.array(contrast_stretched, dtype=np.uint8)

plt.imshow(contrast_stretched, cmap='gray')
plt.title("Contrast Stretching")
plt.axis('off')
plt.show()
```

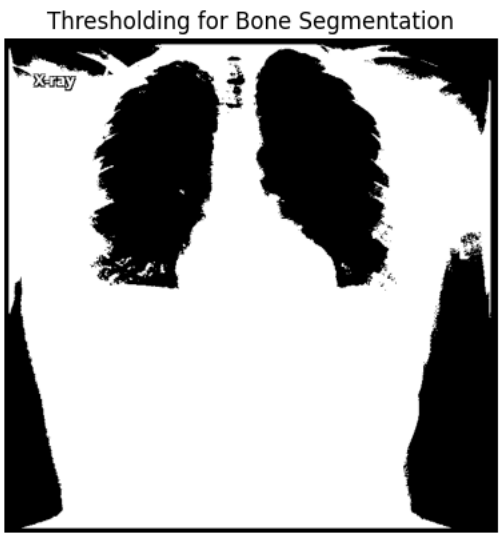


Step 5: Apply Thresholding for Bone Segmentation

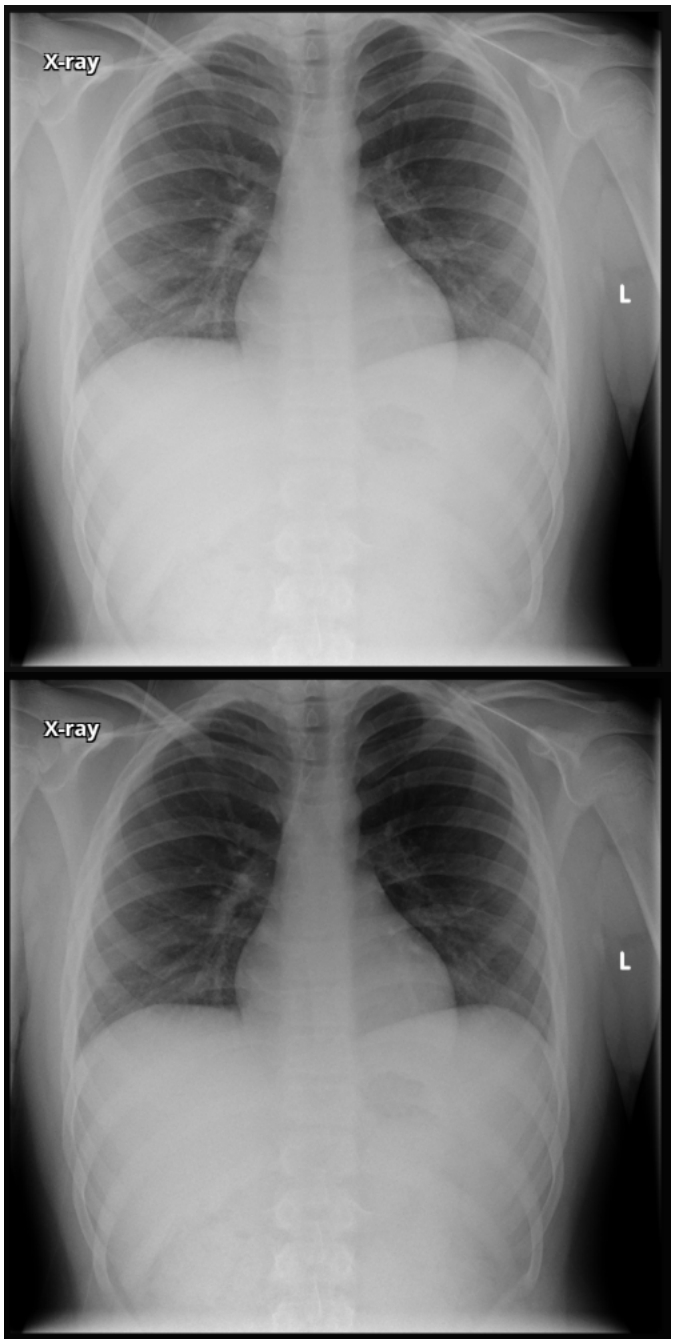
separate bones from the background.

```
_, thresholded = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)

plt.imshow(thresholded, cmap='gray')
```

```
plt.title("Thresholding for Bone Segmentation")
plt.axis('off')
plt.show()
```




```
gamma = 1.5
gamma_corrected = np.array(255 * (image / 255) ** gamma, dtype=np.uint8)

resized_image = cv2.resize(image, (500, 500))
resized_gamma_corrected = cv2.resize(gamma_corrected, (500, 500))

cv2_imshow(resized_image)
cv2_imshow(resized_gamma_corrected)

cv2.waitKey(0)
cv2.destroyAllWindows()
```


X-ray
L
X-ray
L

Scenario no 02 : License Plate Recognition in Traffic Monitoring

A traffic monitoring system needs to **automatically recognize license plates** from surveillance footage. However, due to varying lighting conditions, some license plates appear **too dark or too bright**.

Tasks:

1. Convert image to **grayscale**.
2. Apply **gray level transformation** to improve visibility.

3. Use **thresholding function** to separate text from the background.
4. Apply **log transformation** to enhance low-light images.
5. Compute **mean and variance** to adjust brightness dynamically.

Hint:

- **Adaptive thresholding** (cv2.adaptiveThreshold) works well for different lighting conditions.
- **Mean intensity value** determines whether an image needs **brightness correction**.

Step 1: Load and Display the Image

```
# Load the image
image = cv2.imread("Scanerio 02.png")
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7a13038751d0>
```

Convert to grayscale

We start by converting the image to grayscale. This simplifies the image to a single intensity channel, making it easier to process.

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

plt.imshow(gray_image, cmap='gray')
plt.axis('off')
plt.title('Grayscale Image')
```

```
Text(0.5, 1.0, 'Grayscale Image')
```

Apply gray level transformation

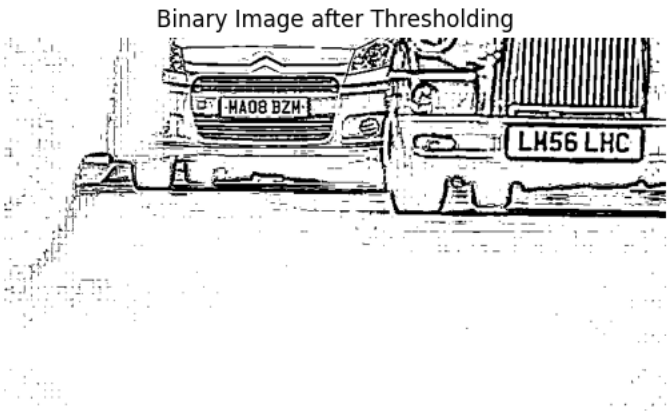
Gray level transformation can be used to adjust the brightness and contrast of the image. We will use a linear transformation to enhance the visibility.

```
alpha = 1.2
beta = 50
gray_transformed = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

plt.imshow(gray_transformed,cmap='grey')
plt.axis('off')
plt.title('Gray Level Transformed Image')
plt.show()
```

Apply adaptive thresholding

Thresholding is used to separate the text from the background. We will use adaptive thresholding to handle varying lighting conditions.

```
binary_image = cv2.adaptiveThreshold(gray_image, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                     cv2.THRESH_BINARY, 11, 2)

plt.imshow(binary_image, cmap='gray')
plt.axis('off')
plt.title('Binary Image after Thresholding')
plt.show()
```



log transformation on gray scale image

Log transformation is useful for enhancing details in darker regions of the image.

```
log_transformed = np.uint8(255 * (np.log(1 + gray_image) / np.log(1 + np.max(gray_image))))

plt.imshow(log_transformed, cmap='gray')
plt.axis('off')
plt.title('Log Transformed Image')
plt.show()
```

```
<ipython-input-45-6477311ea5bf>:1: RuntimeWarning: divide by zero encountered in log
  log_transformed = np.uint8(255 * (np.log(1 + gray_image) / np.log(1 + np.max(gray_image))))
<ipython-input-45-6477311ea5bf>:1: RuntimeWarning: invalid value encountered in cast
  log_transformed = np.uint8(255 * (np.log(1 + gray_image) / np.log(1 + np.max(gray_image))))
```

log transformation on original image

```
log_transformed = np.uint8(255 * (np.log(1 + image) / np.log(1 + np.max(image))))

plt.imshow(log_transformed, cmap='gray')
plt.axis('off')
plt.title('Log Transformed Image')
plt.show()
```

```
<ipython-input-46-919b45d37066>:1: RuntimeWarning: divide by zero encountered in log
  log_transformed = np.uint8(255 * (np.log(1 + image) / np.log(1 + np.max(image))))
<ipython-input-46-919b45d37066>:1: RuntimeWarning: invalid value encountered in cast
  log_transformed = np.uint8(255 * (np.log(1 + image) / np.log(1 + np.max(image))))
```

Compute mean and variance

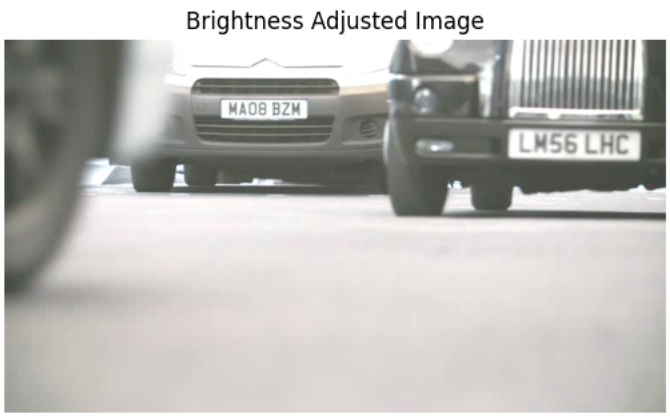
```
mean, stddev = cv2.meanStdDev(image)
print(f"Mean: {mean}, Variance: {stddev}")
```

```
Mean: [[92.85926348]
 [91.40007716]
 [89.42735761]], Variance: [[47.27883222]
 [45.4959386 ]
 [46.8146344 ]]
```

Explanation:

As the original image is having 3 Kernels and so 3 Different Mean Intensity Values , here if condition is applied upon the Average of all the Mean Intensity Values of the original image.

```
mean_of_image = np.mean(image)

if mean_of_image < 100:  # If the image is too dark
    bright_image = cv2.convertScaleAbs(image, alpha=1.5, beta=50)
else:
    bright_image = image

# Display the final adjusted image
plt.imshow(bright_image, cmap='gray')
plt.axis('off')
plt.title('Brightness Adjusted Image')
plt.show()
```



Scenario 03: Satellite Image Enhancement for Disaster Management

After a **flood or wildfire**, satellite images are used to assess damage. However, raw images may have **poor contrast** or **too much noise** due to atmospheric interference.

Tasks:

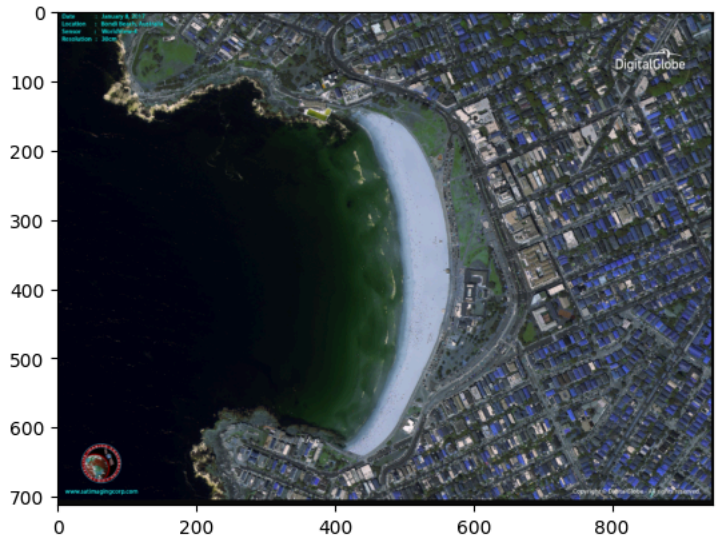
1. Apply **Fourier Transform (FFT)** to remove noise.
2. Use **contrast stretching** to highlight affected regions.
3. Apply **power-law transformation (gamma correction)** to enhance different land types.
4. Compute **mean and variance** to detect **high-reflectance areas (e.g., water bodies after a flood)**.
5. Apply **thresholding function** to separate affected regions from normal land.

Hint:

- **Low-brightness areas** (water, burned land) can be **enhanced using log transformation**.
- **Power-law transformation** helps highlight **vegetation, buildings, and water bodies** differently.

```
# Load image
image = cv2.imread("Scanerio 03.png")
plt.imshow(image)
```

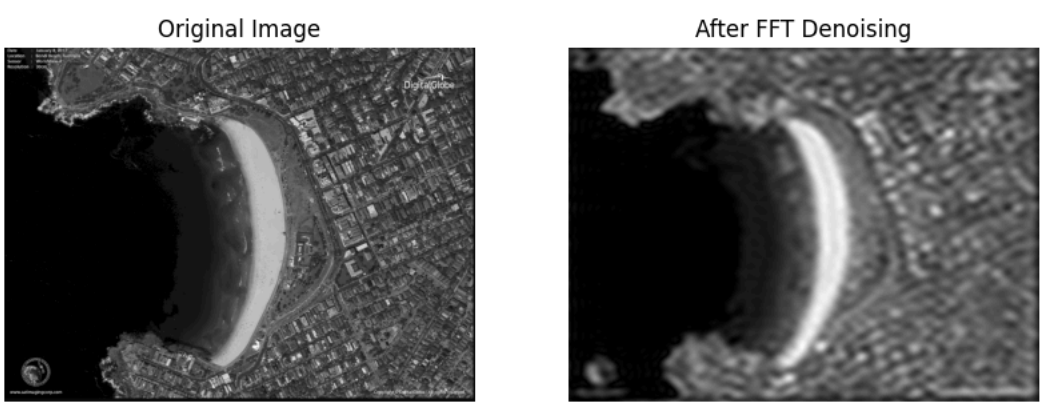
```
<matplotlib.image.AxesImage at 0x7a13081f1350>
```



Step 1: Apply Fourier Transform (FFT) to Remove Noise

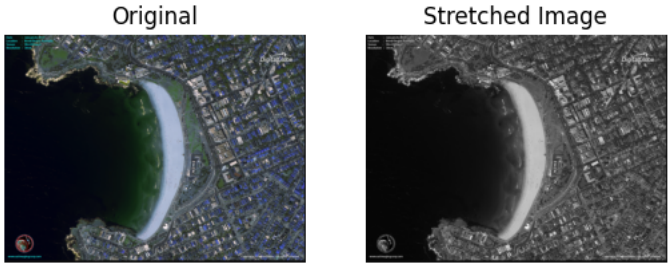
```
image = cv2.imread("Scanerio 03.png", cv2.IMREAD_GRAYSCALE)

# Compute the FFT
f_transform = np.fft.fft2(image)
f_shift = np.fft.fftshift(f_transform)

# Create a mask for the low-pass filter
rows, cols = image.shape
crow, ccol = rows // 2, cols // 2
mask = np.zeros((rows, cols), np.uint8)
mask[crow-30:crow+30, ccol-30:ccol+30] = 1

# Apply mask and inverse FFT
f_shift_filtered = f_shift * mask
f_ishift = np.fft.ifftshift(f_shift_filtered)
image_filtered = np.abs(np.fft.ifft2(f_ishift))

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image, cmap="gray")
plt.title("Original Image")
plt.axis("off")
```

```
plt.subplot(1, 2, 2)
plt.imshow(image_filtered, cmap="gray")
plt.title("After FFT Denoising")
plt.axis("off")

plt.show()
```



Step 2: Contrast Stretching

```
image = cv2.imread("Scanerio 03.png", cv2.IMREAD_COLOR)
grey = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply contrast stretching
min_val = np.min(grey)
max_val = np.max(grey)
stretched = (grey - min_val) * (255.0 / (max_val - min_val))

# Display images
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.axis('off')
plt.title('Original')

plt.subplot(1, 2, 2)
plt.imshow(stretched, cmap='gray')
plt.axis('off')
plt.title('Stretched Image')

plt.show()
```



What Does Contrast Stretching Do?

Contrast stretching (also called **normalization**) improves the **contrast** of an image by **expanding the range of pixel intensity values**.

How It Works:

1. Finds the **minimum** and **maximum** intensity values in the grayscale image.
2. Scales pixel values to spread across the full intensity range **(0 to 255)**.
3. Enhances visibility of details in **low-contrast images**.

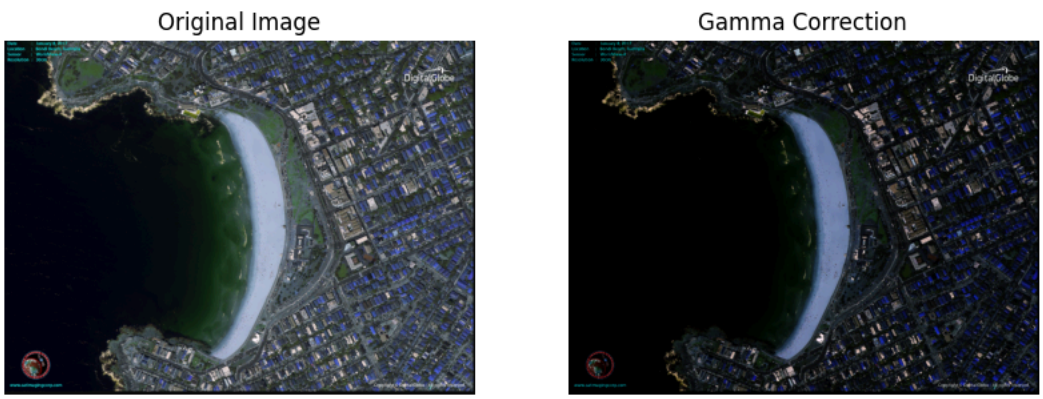
Step 3: Power-Law (Gamma) Transformation

Gamma Value > 1 = Enhances Darker regrions

Gamma Value < 1 = Enhances Brighter regrions

Gamma - Value > 1

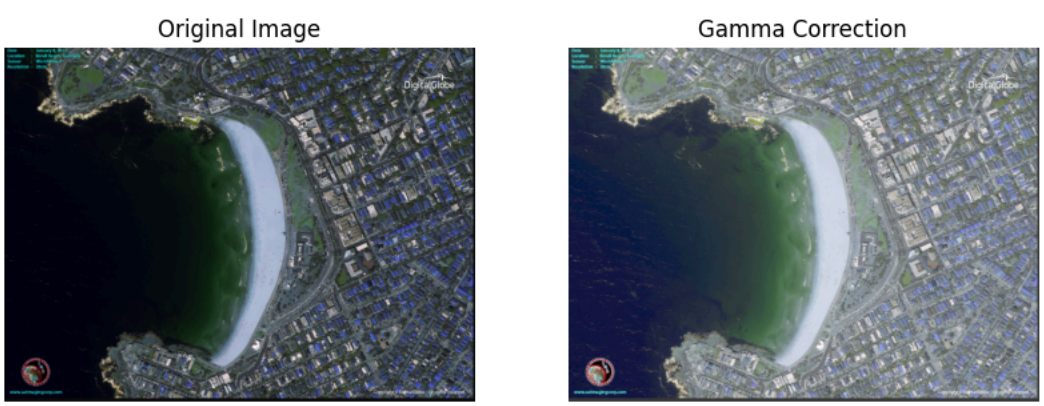
```
# Apply power-law transformation
gamma = 2
gamma_corrected = np.array(255 * (image / 255) ** gamma, dtype="uint8")


plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(gamma_corrected)
plt.title("Gamma Correction")
plt.axis("off")

plt.show()
```



Gamma - Value < 1

```
# Apply power-law transformation
gamma = 0.5
gamma_corrected = np.array(255 * (image / 255) ** gamma, dtype="uint8")


plt.figure(figsize=(10, 5))
```

```
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(gamma_corrected)
plt.title("Gamma Correction")
plt.axis("off")

plt.show()
```

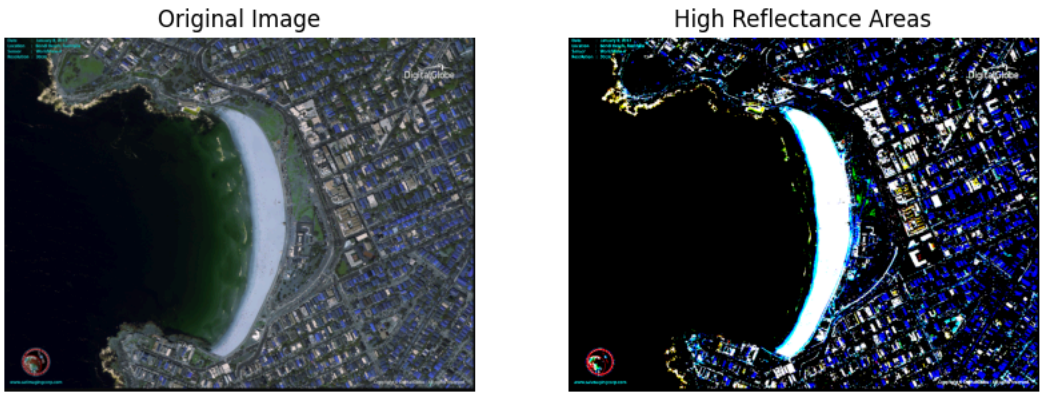


Step 4: Compute Mean and Variance

1. Threshold Calculation

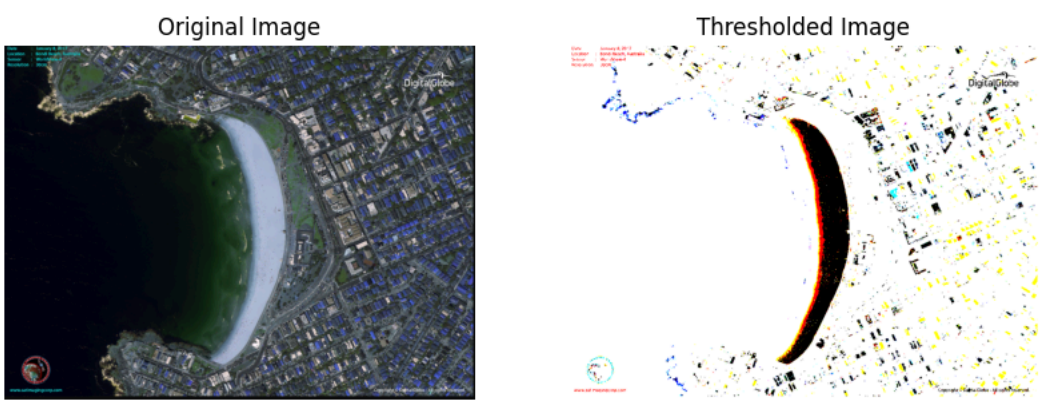
- `mean_val + variance_val ** 0.5`
  - This calculates a threshold using the mean (`mean_val`) and standard deviation (`sqrt(variance_val)`).
  - **Standard deviation** (`sqrt(variance_val)`) measures how much pixel intensities deviate from the mean.
  - Pixels **above this threshold** are considered **high-reflectance areas**.

```
mean_val = np.mean(image)
variance_val = np.var(image)

# Highlight high-reflectance areas (thresholding based on mean)
high_reflectance = np.where(image > mean_val + variance_val ** 0.5, 255, 0).astype(np.uint8)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(high_reflectance)
plt.title("High Reflectance Areas")
plt.axis("off")

plt.show()
```



Step 5: Thresholding Function

```
# Apply binary thresholding
_, thresholded_image = cv2.threshold(255-(image), 100, 255, cv2.THRESH_BINARY)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(thresholded_image)
plt.title("Thresholded Image")
plt.axis("off")

plt.show()
```



Scenario 04: Enhancing Old Manuscripts for Digital Archiving

Libraries and museums often **digitize old manuscripts** for preservation. However, these documents may have **faded ink, uneven lighting, and noise** that make them hard to read.

Tasks:

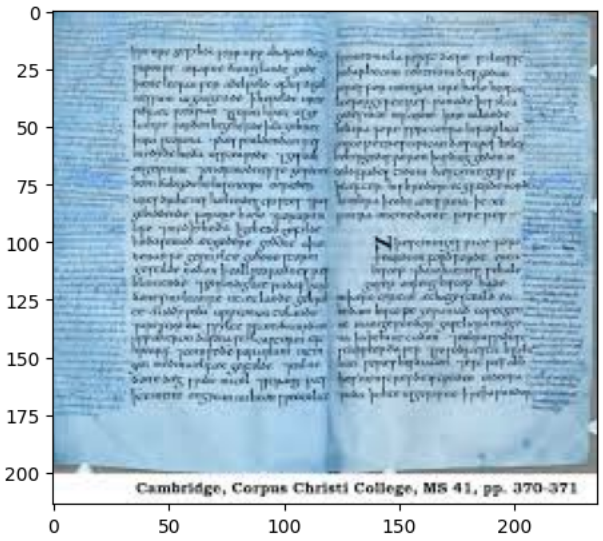
1. Compute **mean and variance** to check overall brightness.
2. Use **log transformation** to enhance faded text.
3. Apply **gray-level transformation** to balance lighting.
4. Apply **contrast stretching** to improve readability.
5. Use **thresholding** to remove the background and extract text.

Hint:

- **Log transformation is useful when the document has faded ink.**
- **Adaptive histogram equalization (cv2.equalizeHist) can enhance text visibility.**

```
image = cv2.imread("Scanerio 04.png")
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7a1303974350>
```



Step 1: Load Image & Compute Mean & Variance

```
image = cv2.imread("Scanerio 04.png")

# Compute mean and variance
mean, stddev = cv2.meanStdDev(image)
print(f"Mean: {mean}, Variance: {stddev}")
```

```
Mean: [[154.68612387]
 [185.47843735]
 [206.21202281]], Variance: [[38.65001918]
 [35.718644  ]
 [35.29796165]]
```

Step 2: Apply Log Transformation

Purpose of Log Transformation

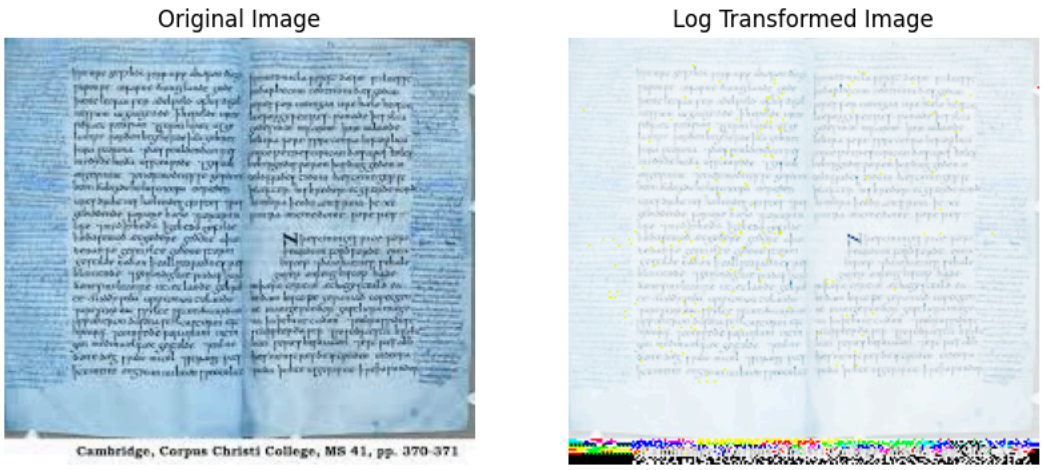
- Enhances details in **dark regions** of the image.
- Reduces the **dominance of bright regions.**
- Helps in cases where images have **faded text** or **low contrast.**

$$\text{log\_transformed} = 255 \times \frac{\log(1 + \text{image})}{\log(1 + \max(\text{image}))}$$

```
log_transformed = np.uint8(255 * (np.log(1 + image) / np.log(1 + np.max(image))))

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(log_transformed)
plt.title("Log Transformed Image")
```

```
plt.axis("off")

plt.show()
```

```
<ipython-input-116-3c5c7fffa17f>:1: RuntimeWarning: divide by zero encountered in log
  log_transformed = np.uint8(255 * (np.log(1 + image) / np.log(1 + np.max(image))))
<ipython-input-116-3c5c7fffa17f>:1: RuntimeWarning: invalid value encountered in cast
  log_transformed = np.uint8(255 * (np.log(1 + image) / np.log(1 + np.max(image))))
```



Step 3: Apply Gray-Level Transformation

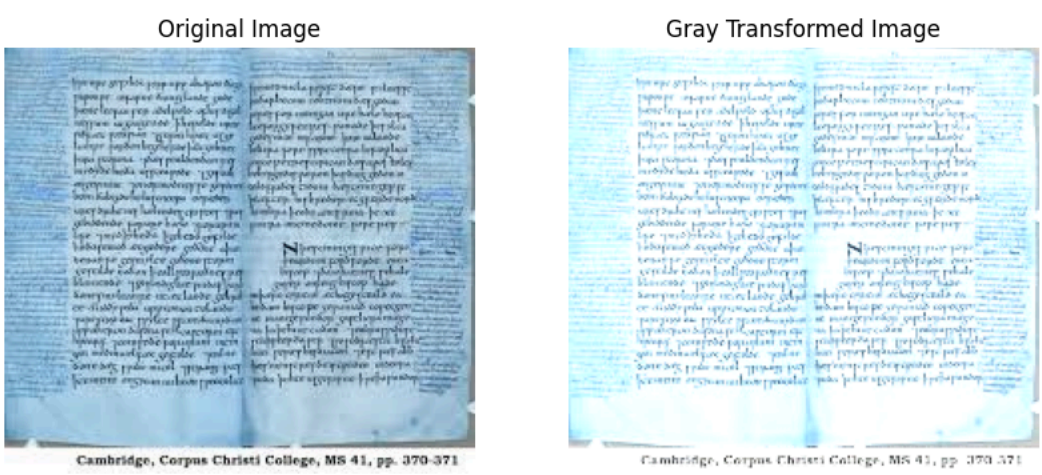
This operation **adjusts the brightness and contrast** of an image using the following transformation:

$$I_{\text{new}} = \alpha \cdot I_{\text{original}} + \beta$$

where:

- I_{new} = Output pixel intensity.
- I_{original} = Input pixel intensity.
- α = **Contrast factor** (here, **1.2** increases contrast by 20%).
- β = **Brightness adjustment** (here, **50** increases brightness by 50 intensity units).
- `cv2.convertScaleAbs()` ensures that pixel values remain within **valid 8-bit range** $[0, 255]$.

```
gray_transformed=cv2.convertScaleAbs(image,alpha=1.2,beta=50)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(gray_transformed)
plt.title("Gray Transformed Image")
plt.axis("off")

plt.show()
```



Step 4: Apply Contrast Stretching

Mathematical Interpretation of Contrast Stretching

The formula:

$$I_{\text{new}} = \left(\frac{I_{\text{original}} - I_{\text{min}}}{I_{\text{max}} - I_{\text{min}}} \right) \times 255$$

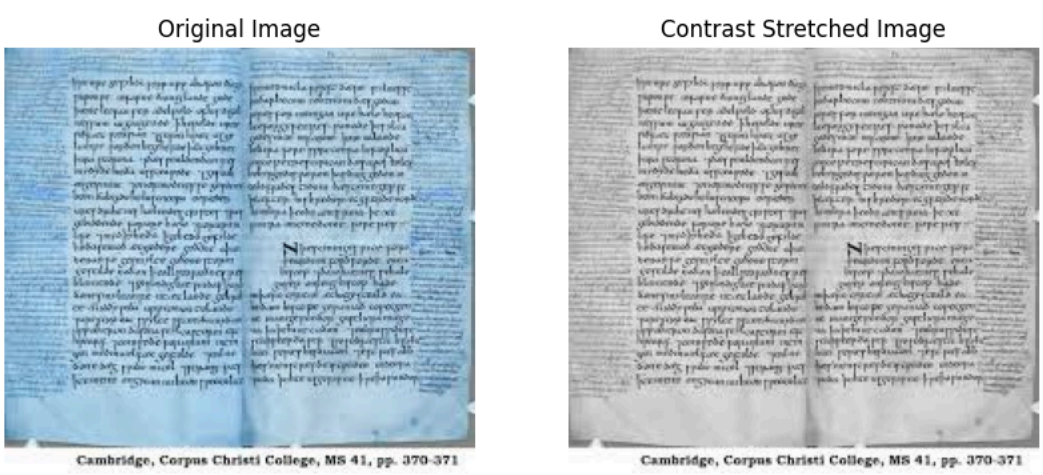
where:

- I_{new} = Output pixel intensity after contrast stretching.
- I_{original} = Input pixel intensity.
- I_{min} = Minimum intensity value in the image.
- I_{max} = Maximum intensity value in the image.
- 255 = The maximum possible intensity value in an 8-bit image.

Effect of Contrast Stretching

- **Maps pixel values from $[I_{\text{min}}, I_{\text{max}}]$ to $[0, 255]$.**
- **Enhances contrast** by making the darkest pixel **black (0)** and the brightest pixel **white (255)**.
- Increases visibility of faint or faded details.

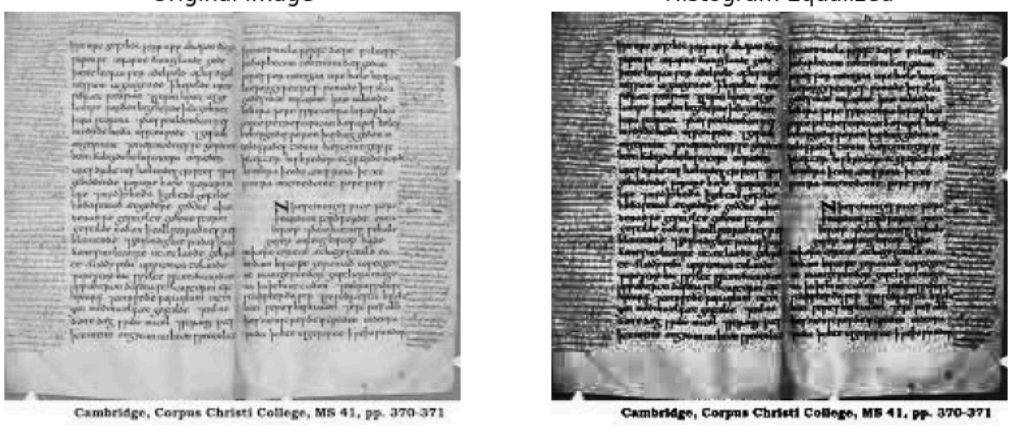
```
image_str = cv2.imread("Scanerio 04.png", cv2.IMREAD_GRAYSCALE)
# Contrast Stretching
min_val = np.min(image_str)
max_val = np.max(image_str)
contrast_stretched = ((image_str - min_val) * (255.0 / (max_val - min_val))).astype(np.uint8)


plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(contrast_stretched, cmap='gray')
plt.title("Contrast Stretched Image")
plt.axis("off")

plt.show()
```



Step 5: Apply Thresholding

The function `cv2.equalizeHist(image)` applies **Histogram Equalization**, a technique to improve image contrast by redistributing pixel intensities.

```
image = cv2.imread("Scanerio 04.png", cv2.IMREAD_GRAYSCALE)

# Apply Histogram Equalization
equalized = cv2.equalizeHist(image)


plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.imshow(image, cmap="gray")
plt.title("Original Image")
plt.axis("off")

plt.subplot(1, 3, 2)
plt.imshow(equalized, cmap="gray")
plt.title("Histogram Equalized")
plt.axis("off")

plt.show()
```



Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.