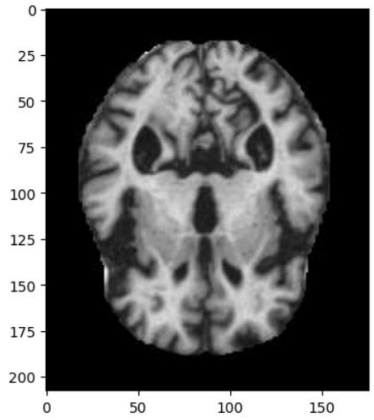
Image Read and Visualize

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

```
: image = cv2.imread('E:\Dataset\Alzheimers Dataset ( 4 class of Images)\ModerateDemented\moderateDem2.jpg')
```

```
plt.imshow(image)
plt.show()
```



Contrast and Brightness Adjustment

Brightness refers to the overall lightness or darkness of an image. It is the amount of light in the image. Increasing the brightness makes the image lighter, and decreasing the brightness makes the image darker.

Contrast refers to the difference between the light and dark areas of an image. High contrast means there's a large difference between the lightest and darkest parts of the image, making the image appear more vivid. Low contrast means that the light and dark parts are closer in value, making the image appear more dull or flat.

Why Needed?

- **Visual Enhancement:** Images might not be clear or visually appealing due to poor lighting conditions, which can lead to too dark or too bright areas. Adjusting brightness and contrast helps enhance visibility and make images more aesthetically pleasing.
- **Highlight Details:** In many cases, important details of an image might be lost due to improper lighting or poor contrast. Adjusting these settings can reveal hidden information, making images more useful for analysis or interpretation.
- **Medical Imaging:** In medical imaging, such as X-rays or MRIs, fine details may be hard to see if contrast is too low. Adjusting the contrast helps medical professionals view and diagnose conditions more clearly.

- **Image Preprocessing for Analysis:** In computer vision, especially when working with algorithms like object detection or edge detection, proper contrast and brightness can improve the performance of these algorithms by making features more distinct and easier to process.

Calculation of Brightness: Calculated as the average pixel intensity of the image.

```
: brightness = np.mean(gray_image)
  print(f"Brightness (Average pixel intensity): {brightness}")
```

```
Brightness (Average pixel intensity): 61.85494973776224
```

Calculation of Contrast: Calculated as the standard deviation of the pixel values in the image.

```
: contrast = np.std(gray_image)
  print(f"Contrast (Standard deviation of pixel values): {contrast}")
```

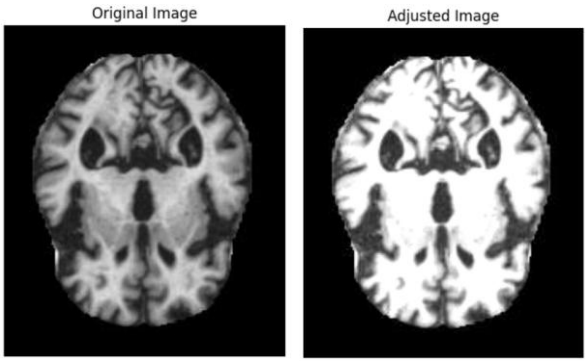
```
Contrast (Standard deviation of pixel values): 72.17346849658134
```

To adjust the brightness and contrast of an image, we Generally use the following linear transformation:

$$Output\ Pixel = \alpha * input\_pixel + \beta$$

- α is the contrast factor (also called gain).
- β is the brightness factor (also called bias or offset).

```
alpha = 1.5
beta = 40
adjusted_image = cv2.convertScaleAbs(gray_image, alpha=alpha, beta=beta)
```



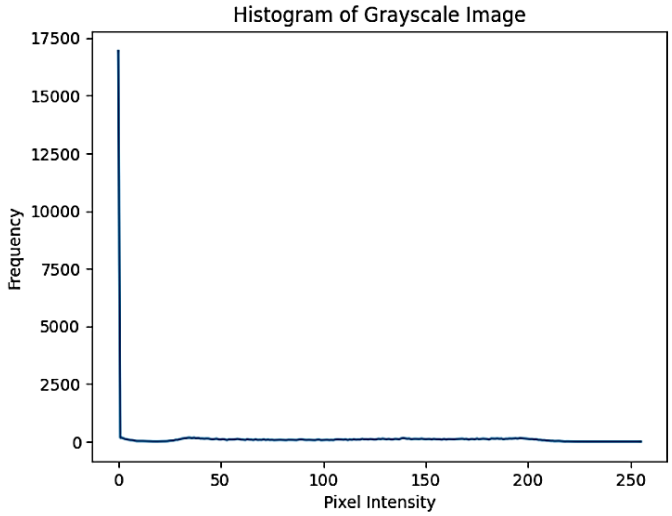
Histogram Equalization

A histogram in the context of image processing is a graphical representation of the distribution of pixel intensities (values) in an image. It shows the frequency of occurrence of each pixel intensity in the image.

- **X-axis:** Represents the pixel intensity values (ranging from 0 to 255 for an 8-bit grayscale image, or 0 to 1 for normalized values).

- **Y-axis:** Represents the frequency or count of how many pixels in the image have that particular intensity.

The histogram of an image shows how many pixels there are in each intensity level (dark or light).



So,

Histogram equalization is a contrast adjustment technique in image processing used to improve the visual appearance of an image. The technique involves redistributing the image's pixel intensity values across the entire range of available intensities (from 0 to 255 for an 8-bit image).

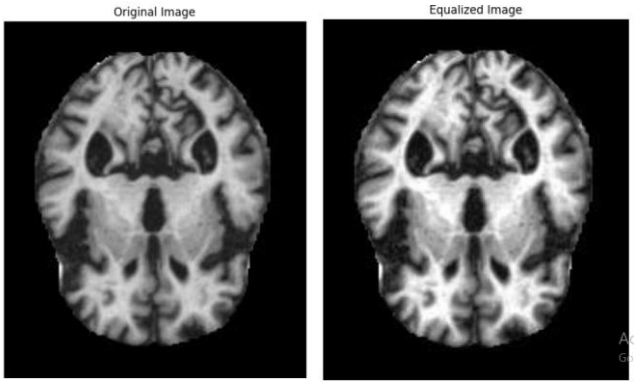
Why is Histogram Equalization Needed?

- **Enhancing Contrast:** Many images, especially those taken in poor lighting conditions, may have a limited range of pixel intensities. This can make it hard to distinguish features or details. Histogram equalization helps by spreading pixel intensities more evenly across the entire range, improving contrast.
- **Improving Visibility:** It makes dark regions darker and bright regions brighter, making the image clearer and more readable.
- **Improving Image Analysis:** In computer vision, better contrast can help algorithms (e.g., edge detection, object recognition) perform better by making key features more distinguishable.

How Histogram Equalization Works?

- Step 1: Compute the cumulative distribution function (CDF) of the histogram.
- Step 2: Normalize the CDF to map pixel values to the full range (0 to 255).
- Step 3: Map the original pixel values to new values using the CDF.

```
equalized_image = cv2.equalizeHist(gray_image)
```



Linear and Non-linear Transformations

In image processing, transformations are used to manipulate pixel values in images to improve the quality of the image or to extract useful features. These transformations can be categorized into linear and non-linear types based on how the pixel values are modified.

Brightness and Contrast Adjustment together is also known as linear transformation.

Non-linear Transformations

A non-linear transformation applies more complex functions to the pixel values. Unlike linear transformations, non-linear transformations don't maintain a direct proportionality between the input and output pixel values. Instead, they might apply more complex mathematical functions (e.g., logarithmic, exponential, etc.) to transform the image.

What is Gamma Correction?

- Gamma correction is a non-linear transformation applied to an image's pixel values to adjust the brightness in a way that corrects for display devices or lighting conditions.

- It is based on the gamma function and is commonly used because human vision perceives changes in brightness non-linearly. That is, the perceived brightness of an image is not directly proportional to its pixel values.

Why is Gamma Correction Needed?

- **To correct for display characteristics:** Display devices like monitors, cameras, and scanners often do not exhibit linear behavior when capturing or displaying images.
- **To match human vision:** Human vision is more sensitive to changes in dark areas than in bright areas, so gamma correction can make the image appear more natural to the human eye.

$$P'(x,y) = 255 \times \left(\frac{P(x,y)}{255}\right)^{\gamma}$$

```
gamma = 2.2
gamma_corrected_image = np.array(255 * (gray_image / 255) ** gamma, dtype='uint8')
```