**matplotlib.pyplot**: A module in the matplotlib library that provides a collection of functions for creating plots, figures, and visualizations.

**plt**: The conventional alias used for matplotlib.pyplot to make the code cleaner and easier to read.

**matplotlib.image**: A module used for loading and displaying image files.

**image**: The variable stores the image data as a **NumPy array**.

* For **JPEG images**, the resulting array typically has a shape of (height, width, 3) for RGB images or (height, width, 4) for RGBA images (with an alpha channel for transparency).
* The pixel values are usually stored as floating-point numbers between 0 and 1 (for normalized images) or integers between 0 and 255.

**numpy**: A powerful library for numerical and scientific computing. It provides support for working with large, multi-dimensional arrays and matrices, along with a wide range of mathematical functions to operate on these arrays efficiently.

When an image is read using matplotlib.image.imread() (assuming it’s an RGB image), it is typically represented as a **3-dimensional NumPy array** with the shape:

(height, width, 3)

* **Height**: The number of rows (pixels) in the image.
* **Width**: The number of columns (pixels) in the image.
* **3**: The third dimension represents the three color channels:
  + **Channel 0**: Red (R)
  + **Channel 1**: Green (G)
  + **Channel 2**: Blue (B)

For example, if an image has a shape of (100, 200, 3), it means:

* The image is **100 pixels tall**.
* The image is **200 pixels wide**.
* Each pixel has 3 values: one for **Red**, one for **Green**, and one for **Blue**.

 **r, g, b = image[:, :, 0], image[:, :, 1], image[:, :, 2]**:

* Separates the Red, Green, and Blue channels from the image array.

 This is useful for tasks like:

* **Color analysis**: Processing each color channel separately.
* **Image processing**: Applying filters or transformations to specific channels.
* **Visualization**: Displaying individual color channels

Raw pixel values do not linearly correspond to perceived brightness. Gamma correction adjusts the pixel values to ensure accurate reproduction of brightness.

**Luminance Contribution Constants** are weights used to calculate the perceived brightness (or **luminance**) of a color image by combining the Red, Green, and Blue (RGB) channels. These constants account for the fact that the human eye perceives different colors with varying sensitivities:

* **We are most sensitive to green**, which appears brighter.
* **We are moderately sensitive to red**.
* **We are least sensitive to blue**, which appears darker.

When converting a color image to grayscale, these constants ensure that the resulting image closely matches how humans perceive brightness.

These constants ensure that the resulting grayscale intensity represents how bright the color **appears** to humans, not just the numerical average of the RGB channels.

 Pixels with intensity **above 127** will become **white (1)**.

 Pixels with intensity **127 or below** will become **black (0)**.

 **Checks each pixel** in a grayscale image.

 **Compares** the pixel's intensity to a **threshold (127)**.

 Converts pixels to either **white (1)** or **black (0)** to create a **binary image**.

**plt.subplots(1, 3)** creates a **1 row × 3 columns** grid of subplots (3 side-by-side images).

**fig** represents the overall figure.

**axes** is an array of the 3 individual subplot axes, allowing you to control each subplot separately.

**figsize=(10, 7)** sets the overall figure size to **10 units wide** and **7 units high**.

**imshow(grayscale\_image, cmap='gray')** displays the grayscale image using the gray colormap, which ensures the image appears in shades of black, white, and gray.

plt.bar(bin\_edges[:-1], hist, width=np.diff(bin\_edges), edgecolor="black", align="edge")

* **plt.bar** creates a bar chart.
* **bin\_edges[:-1]** gives the starting edge of each bin (to position the bars correctly).
* **width=np.diff(bin\_edges)** calculates the width of each bar based on the bin edges.
* **edgecolor="black"** adds a black border around the bars.
* **align="edge"** aligns the bars to the left edge of the bins.

| **Filter** | **Function** | **Best Use Case** |
| --- | --- | --- |
| **Gaussian Filter** | Blurs the image by averaging pixel values | Reducing general noise |
| **Median Filter** | Replaces pixels with the median value | Removing salt-and-pepper noise |

scipy.ndimage is a **submodule** in the **SciPy** library that provides functions for **multi-dimensional image processing**.

**binary\_image.astype(float)**

**Converts the binary image** (with pixel values 0 and 1) to **floating-point numbers** (0.0 and 1.0).

**Applies a Gaussian blur** to the image.

The parameter **sigma=2** controls the **amount of blurring**:

A **larger sigma** value means **more blurring**.

A **smaller sigma** value means **less blurring**.

**What the Gaussian Filter Does**:

It **smooths the image** by averaging pixel values using a **Gaussian distribution**.

The pixels closer to the center have **higher weights**, while those farther away contribute **less** to the average.

This helps to **soften edges** and **reduce noise**.

**Example of How It Works**

Imagine you have a simple binary image like this:

Copy code

1 0 0 0 0

0 0 1 0 0

0 0 0 0 0

0 1 0 0 0

0 0 0 0 1

After applying gaussian\_filter with sigma=2, the image might look like this:

Copy code

0.4 0.3 0.2 0.1 0.0

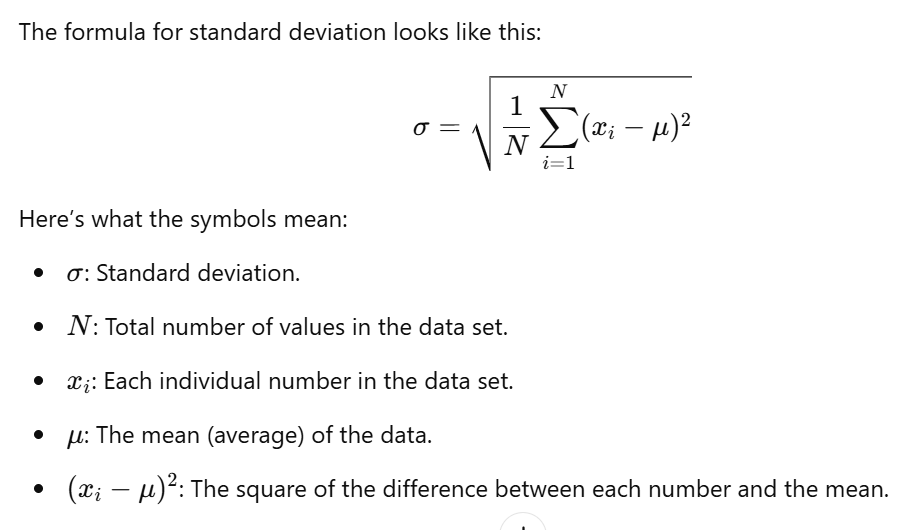
0.3 0.3 0.3 0.2 0.1

0.2 0.2 0.2 0.2 0.2

0.2 0.3 0.3 0.2 0.1

0.1 0.2 0.2 0.2 0.3

**Standard deviation** is a measure of how spread out the numbers in a data set are. In simpler terms, it tells you how much the numbers differ from the average (mean) of the data.



astype(float)

 The **astype(float)** part converts the image data to a floating-point format (like decimals).

 This is because some operations, like smoothing, work better with precise decimal values instead of just integers.

**gaussian\_filter**:

This is a function used to smooth an image, meaning it reduces noise and makes the image less "sharp."

It works by applying a **Gaussian kernel**, which is like a mathematical "blurring brush" that spreads out pixel values to their neighbors in a specific way.