**INFOSYS SPRINGBOARD INTERNSHIP**

**Problem Statement:**

Breast cancer is one of the leading causes of mortality among women worldwide. Early and accurate detection is critical to improving survival rates, yet existing diagnostic methods can be prone to false positives or negatives, which can delay treatment and lead to patient anxiety. Magnetic Resonance Imaging (MRI) has emerged as a powerful imaging technique for identifying abnormalities in breast tissue. However, accurately interpreting MRI scans remains challenging due to the complex nature of the scans and the potential for human error. TumorTrace aims to leverage artificial intelligence (AI) to enhance the precision and efficiency of breast cancer detection by analysing MRI scans. By employing advanced machine learning algorithms, TumorTrace seeks to reduce misdiagnosis, accelerate detection, and improve overall patient outcomes, making early intervention more feasible and effective.

**OUR APPROACH:**

**1. Image Enhancement:** Apply Sobel edge detection to highlight tumour boundaries and enhance image clarity.

**2. Intensity Mapping:** Use histograms to map pixel intensity and frequency, improving contrast and highlighting subtle variations in MRI scans.

**3. Model Training:** Train a deep learning model on the enhanced images to distinguish between benign and malignant tissues.

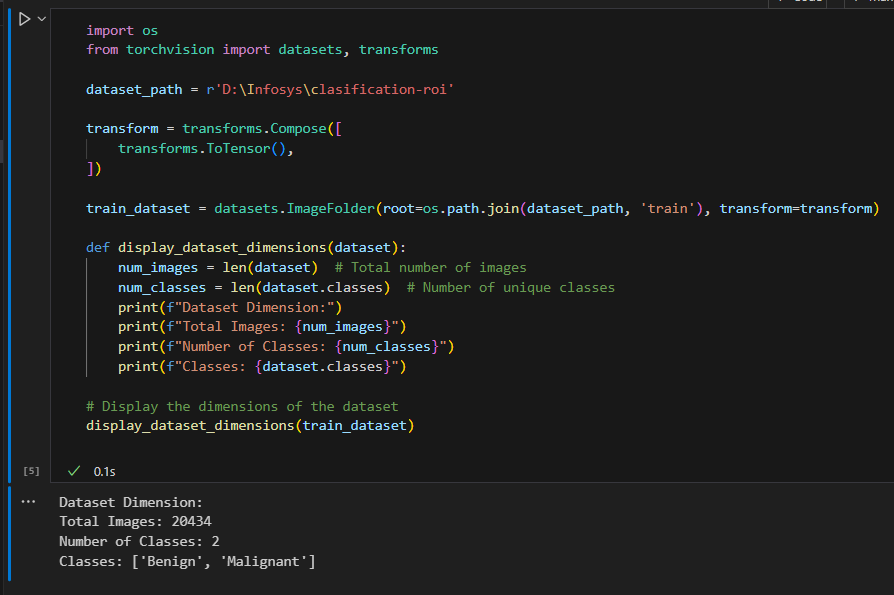
**4. Accuracy Optimization:** Use supervised learning techniques to achieve high sensitivity and specificity in detecting abnormalities.

**5. Deployment:** Deploy the trained model to analyze new MRI scans in real-time, supporting healthcare professionals in early breast cancer detection.

**-Shreya Singh**

**DATA INSIGHTS**

Task-1 EDA of the dataset:



**Basic Description**

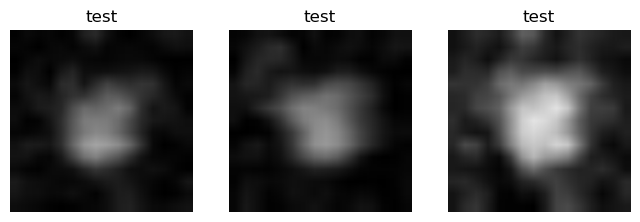
1. **Dataset Path:** The dataset is stored in the directory 'D:\Infosys\classification-roi', structured for automatic label assignment based on folder names ('Benign' and 'Malignant')
2. **Image Transformation:** The images are converted from PIL format to PyTorch tensors using transforms.ToTensor(), enabling compatibility for model training.
3. **Loading the Dataset:** The dataset is loaded using datasets.ImageFolder, which automatically assigns labels based on the

subfolder names.

1. **Dataset Dimensions Function:** The display\_dataset\_dimensions () function calculates and displays the total number of images,

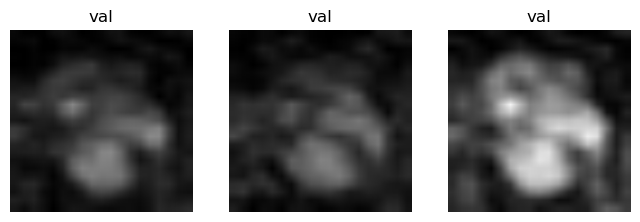
the number of unique classes, and the list of classes ('Benign' and 'Malignant').

1. **Output Summary:** The dataset contains 20,434 images across 2 classes: 'Benign' and 'Malignant'.

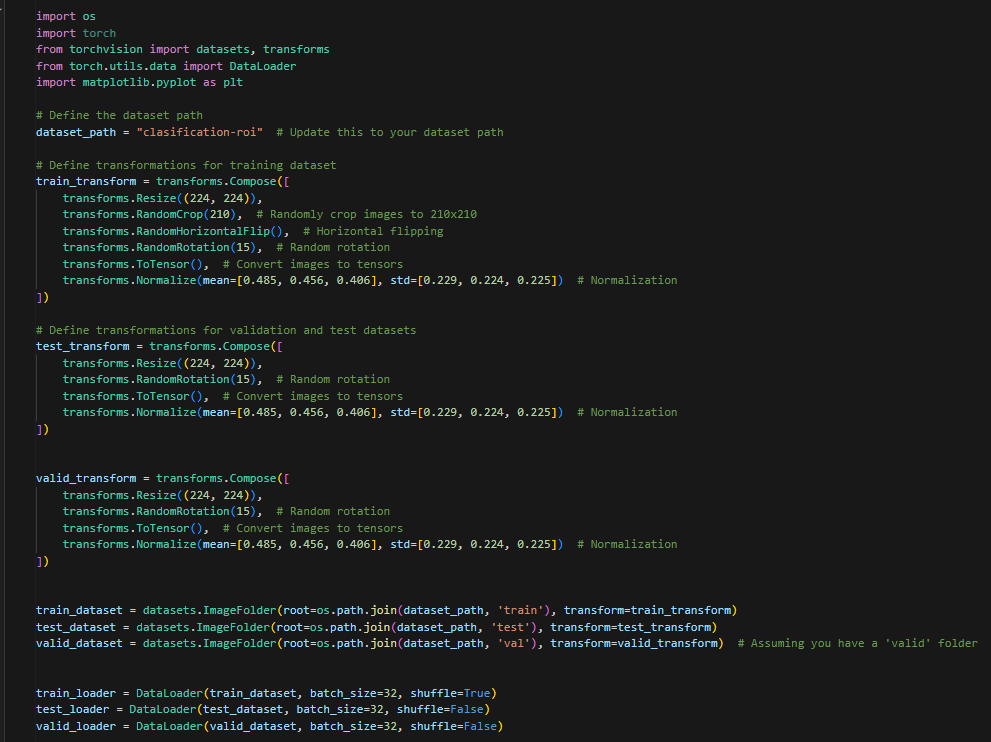
**SAMPLE IMAGES**



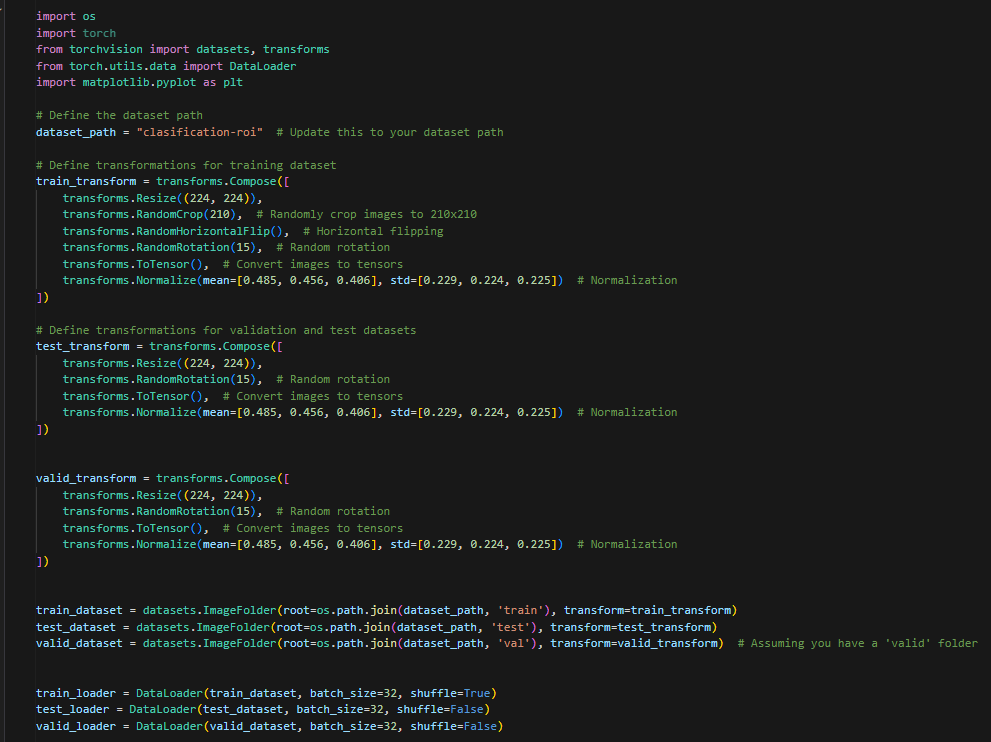


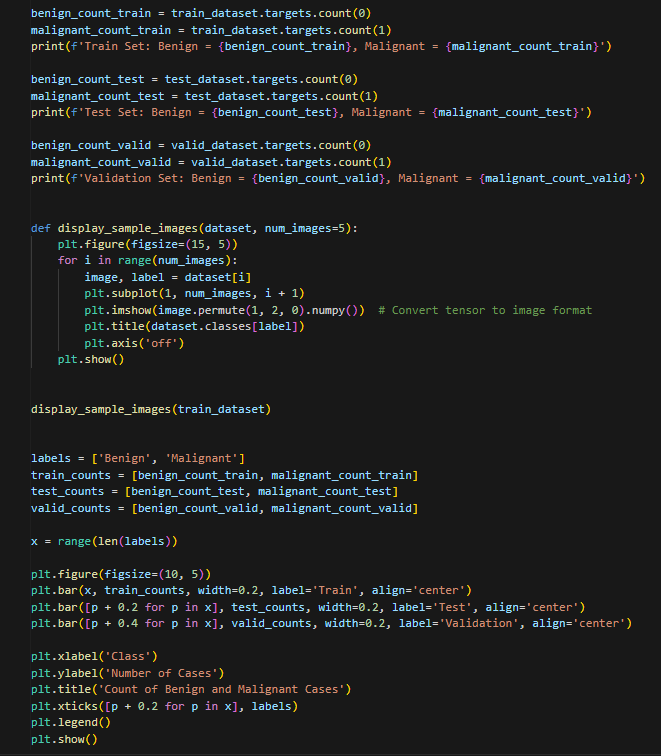


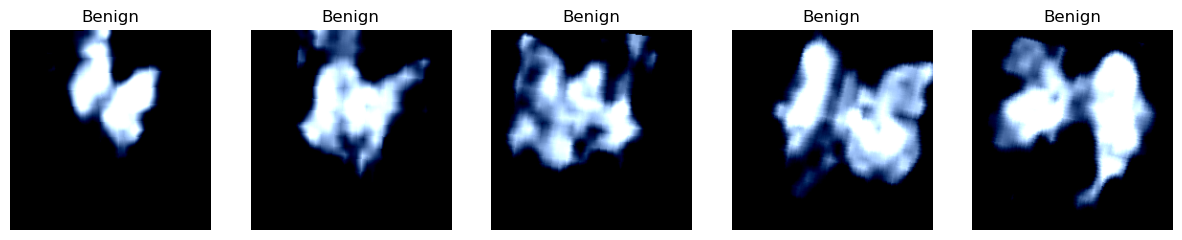
Task-2 Normalisation and Data Augmentation

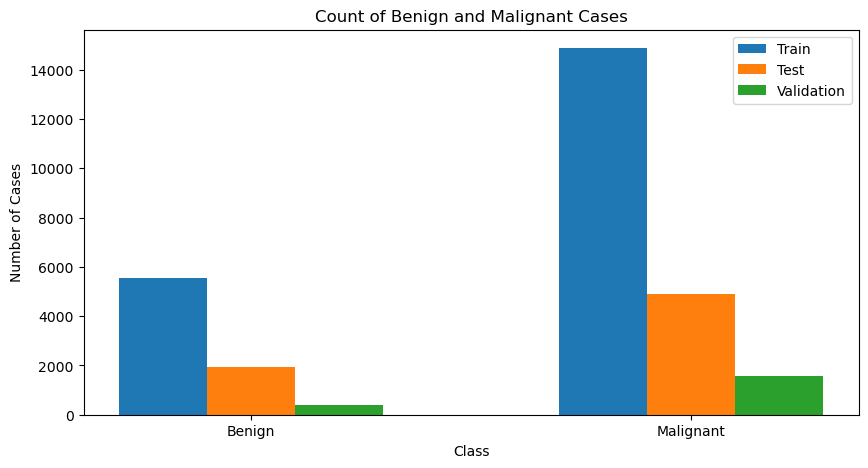


COUNT OF BENIGN AND MALIGNANT IMAGES









## **1. Data Preparation:**

## The code resizes images, applies random crops, flips, and rotations, and normalizes them to make them ready for training.

## **2. Loading Datasets:**

## It loads three datasets (train, test, validation) from the specified directory, automatically labelling images as 'Benign' or 'Malignant' based on their folder names.

## **3. Data Handling:**

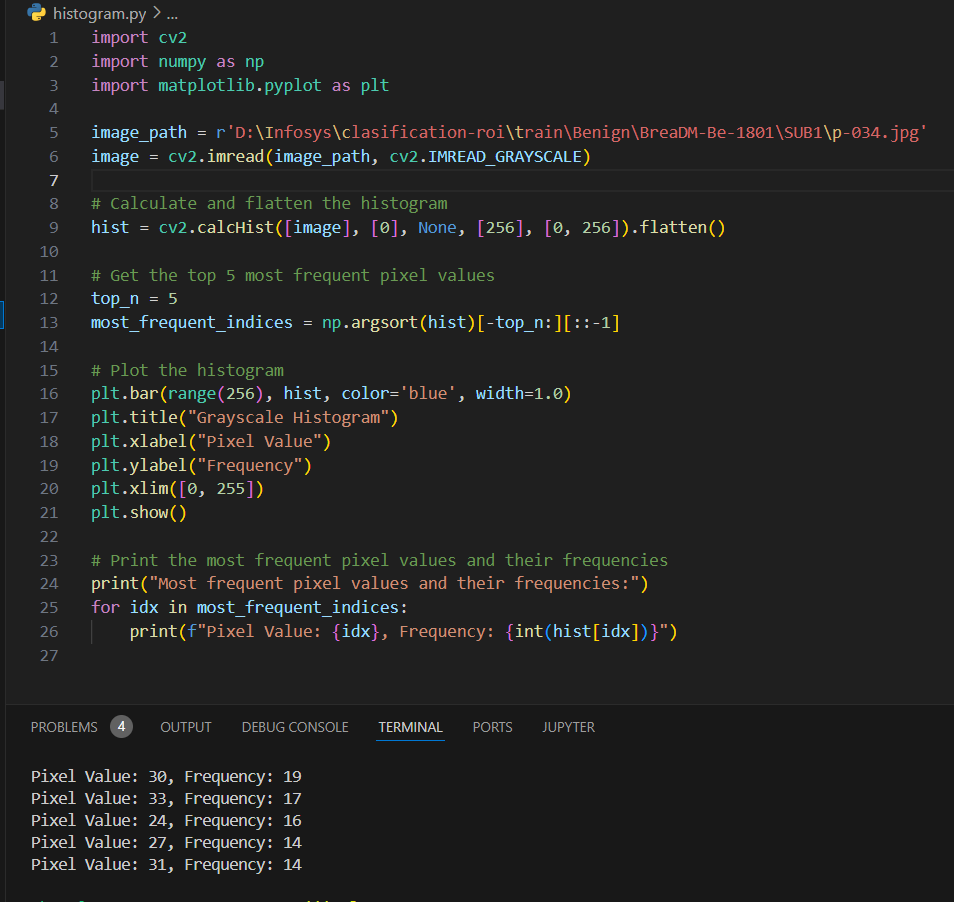
## DataLoaders are created for each dataset to load images in batches of 32. The training data is shuffled to add randomness, while test and validation data are not.

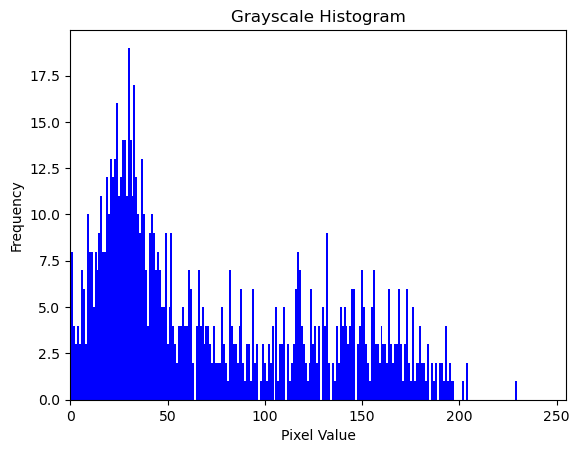
## **4. Data Distribution:**

## The code counts and displays the number of benign and malignant images in each dataset, then plots a bar chart to compare the distribution.

## **5. Image Display:**

## A function shows a few sample images from the training set, allowing you to see what the images look like along with their labels.

**3 Histogram of the pixel value and it’s the frequency in an image**

****

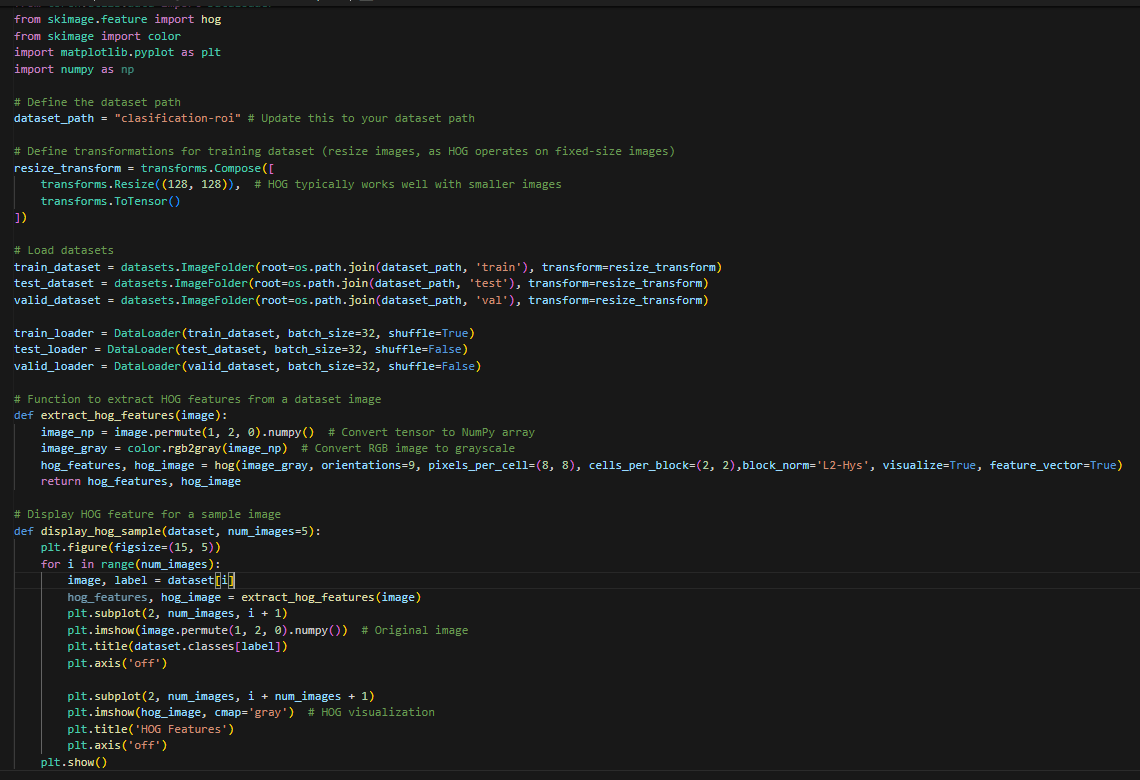
**Basic Description:**

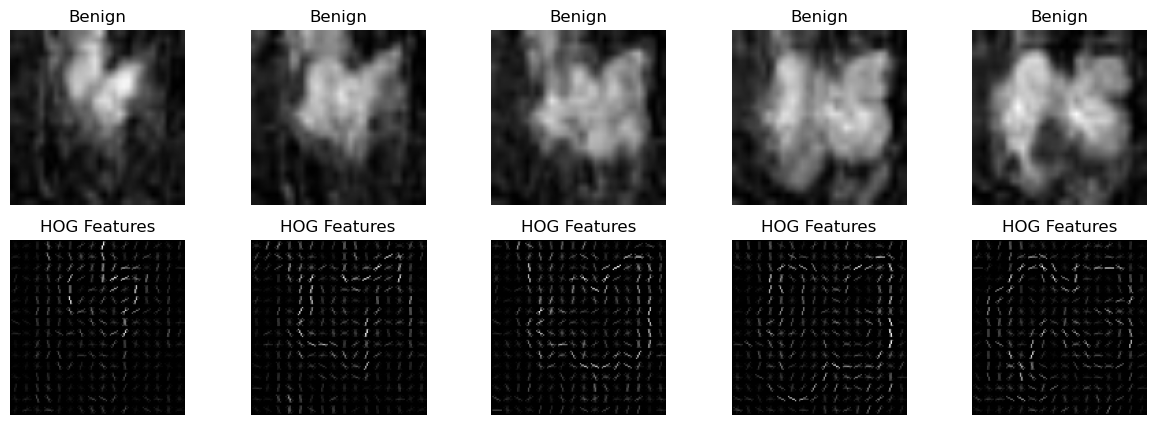
**Grayscale Histogram Calculation and Plotting:**

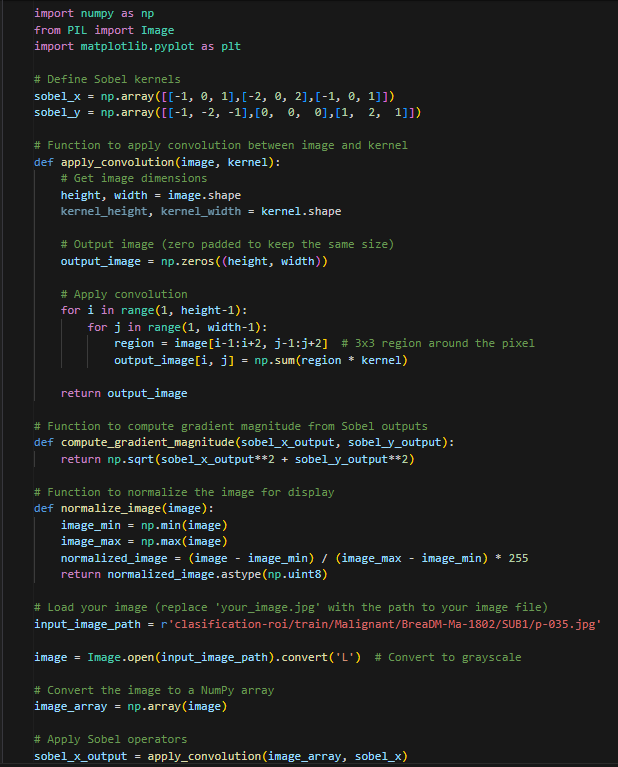
* **Histogram Calculation**:
  + A histogram of pixel values is computed using cv2.calcHist(), which counts the frequency of each grayscale pixel value.
  + The histogram is flattened to a 1D array to simplify subsequent operations.
* **Top 5 Frequent Pixel Values**:
  + The top 5 most frequent pixel values are extracted using np.argsort() on the histogram values.
* **Histogram Plotting**:
  + A bar chart is created to visualize the frequency of grayscale pixel values (range 0–255) using plt.bar().
  + The histogram helps in analyzing the distribution of pixel intensities in the grayscale image, indicating image contrast and texture characteristics.
* **Output**:
  + The most frequent pixel values are printed alongside their frequencies.

**2. Grayscale Histogram Output Visualization:**

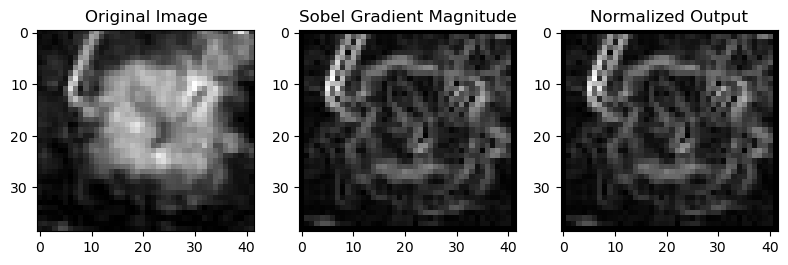
* **Histogram Plot**:
  + The output figure shows a **Grayscale Histogram**, plotting pixel values (x-axis) against their frequencies (y-axis).

HOG FEATURES









**1. Sobel Kernel Definition:**

* Two 3x3 Sobel kernels are defined to detect edges in horizontal (sobel\_x) and vertical (sobel\_y) directions. These kernels help in calculating gradients in the x and y directions, which are essential for detecting edges in the image.

**2. Applying Convolution:**

* The apply\_convolution() function convolves the Sobel kernels with the input image. It calculates the intensity gradients by sliding the kernel over the image, emphasizing areas with rapid intensity changes (edges).

**3. Gradient Magnitude Calculation:**

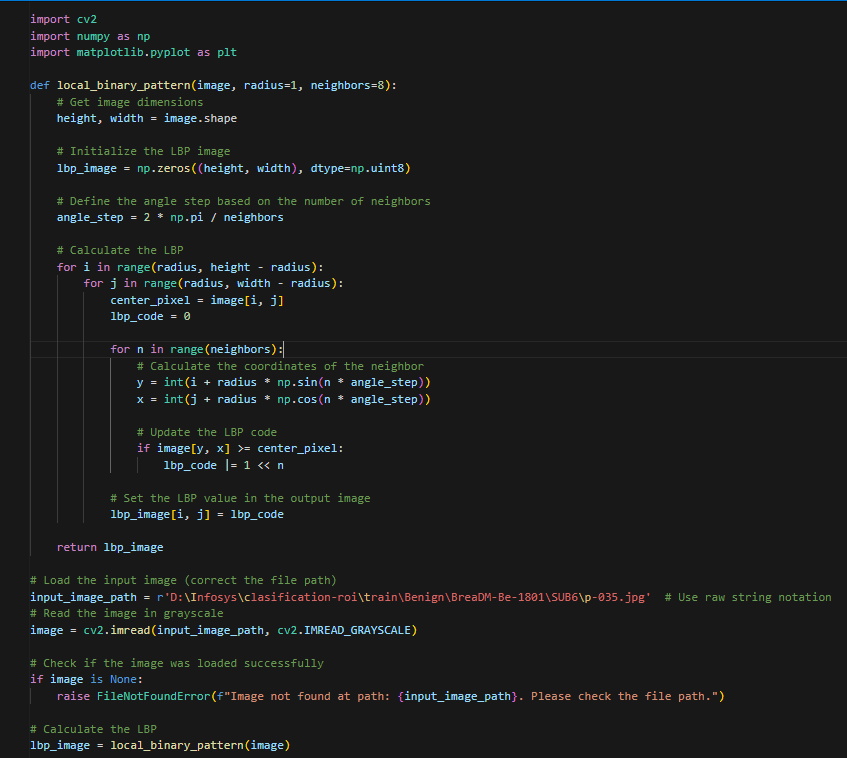
* The compute\_gradient\_magnitude() function calculates the overall edge magnitude by combining the gradients from the x and y directions using the Euclidean norm. This provides a clear representation of edge strength across the image.

**4. Normalization for Visualization:**

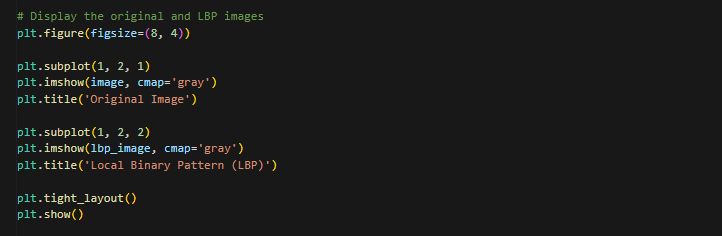
* The normalize\_image() function scales the gradient magnitudes to a range of 0-255, making the edges visible as a grayscale image. This step ensures better visualization of the detected edges.

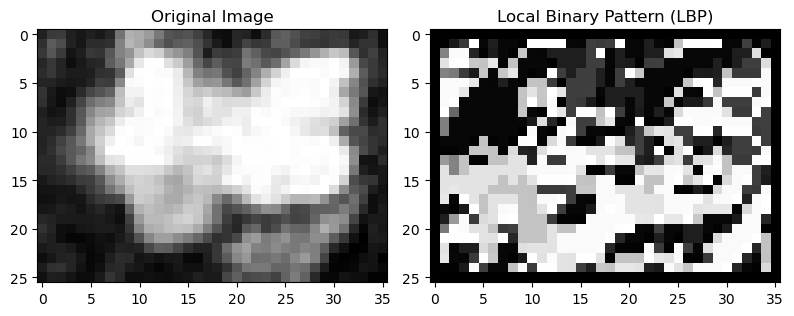
**5. Plotting the Results:**

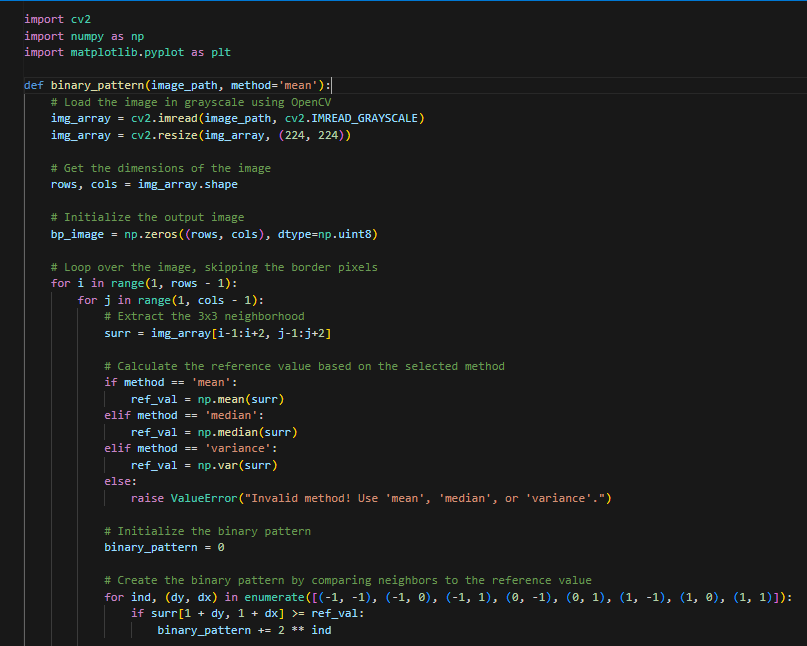
* The final section displays the original image, gradient magnitudes from the x and y directions, and the combined edge magnitude using matplotlib. The visualization helps in understanding how the Sobel operator detects edges and enhances the image for analysis.

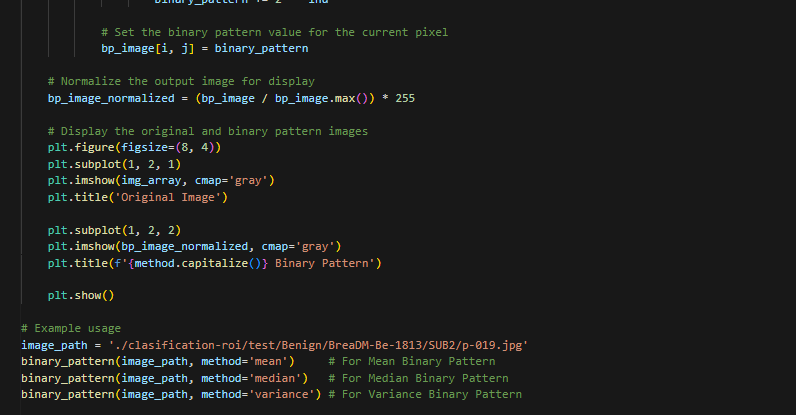
L

LOCAL BINARY PATTERN

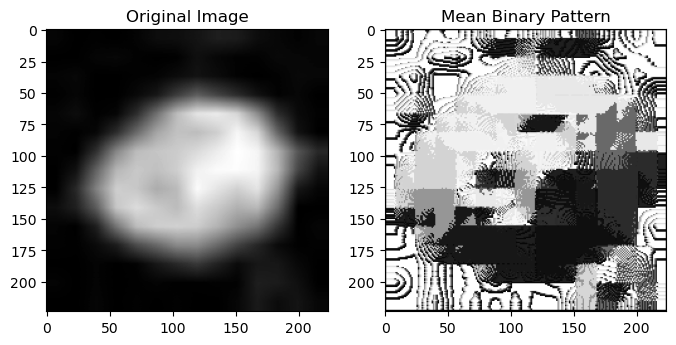


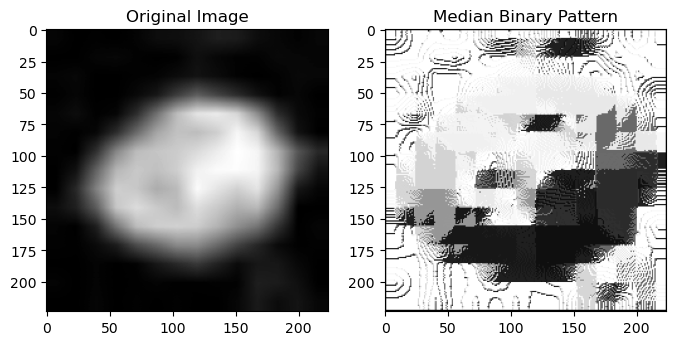


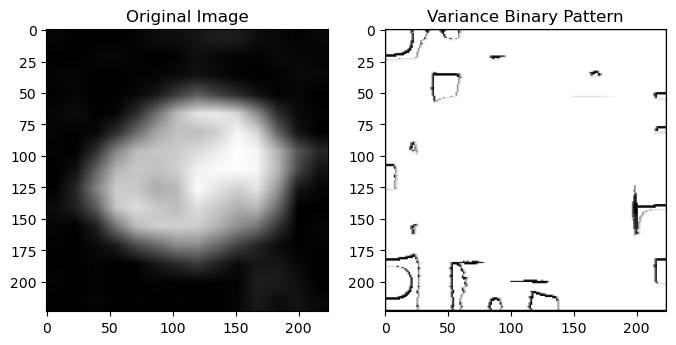




K





 q