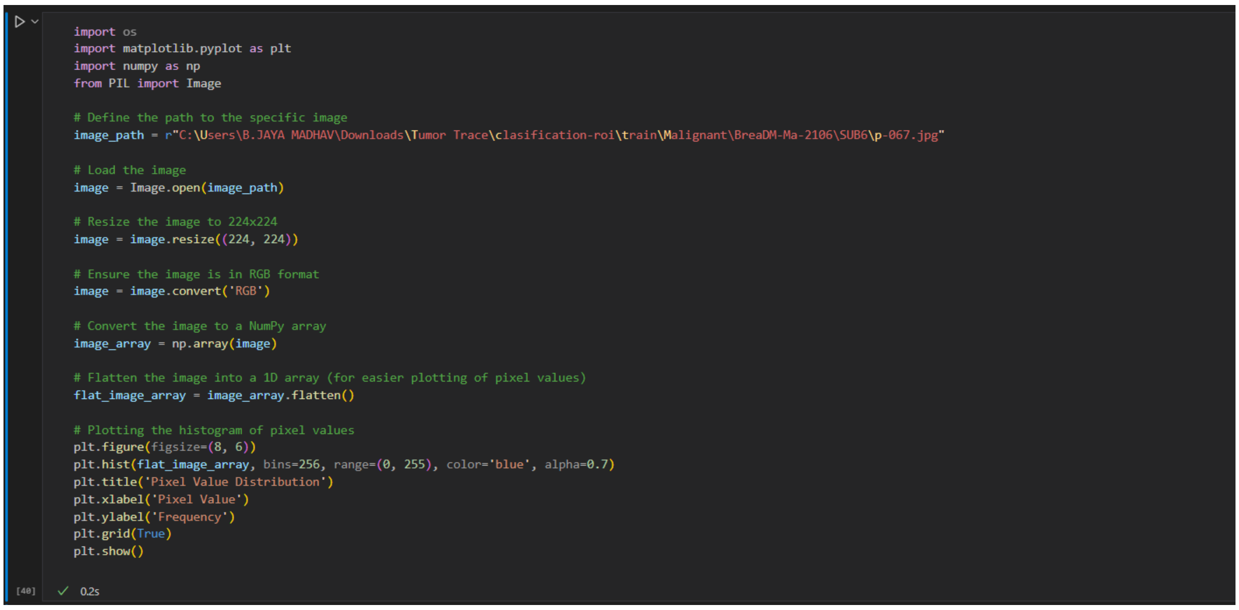
Project: TumorTrace: MRI-Based AI for Breast Cancer Detection

Name: Jaya Madhav B

Email: madhavjaya4@gmail.com

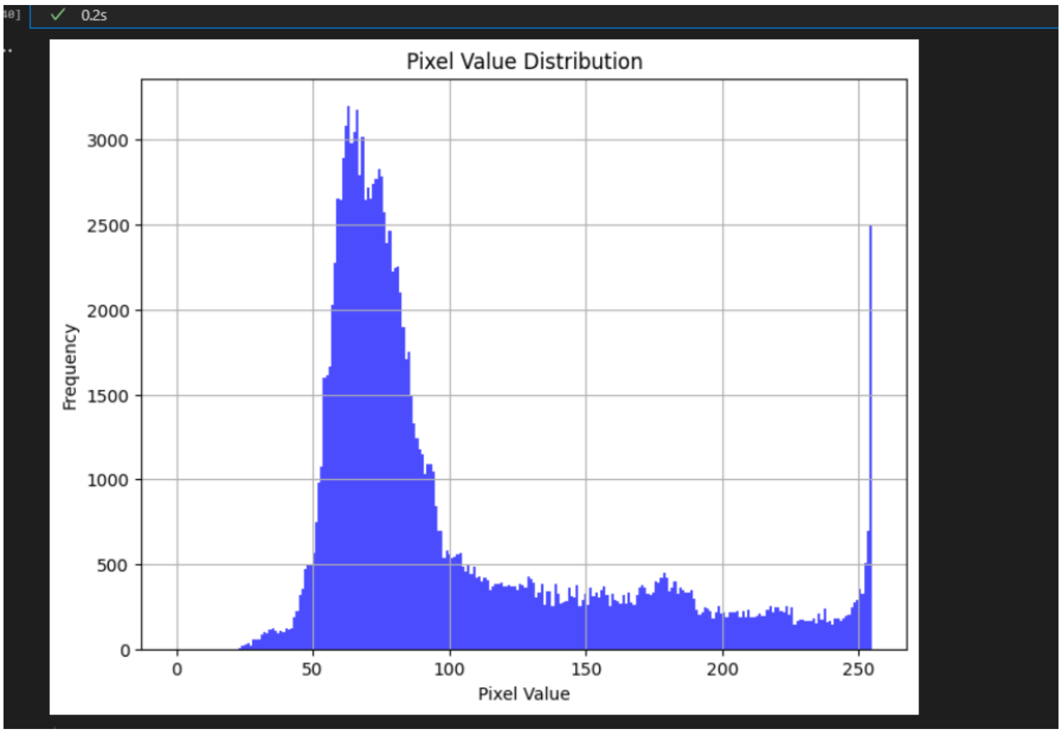
CONTENTS:

PIXEL VALUE DISTRIBUTION:

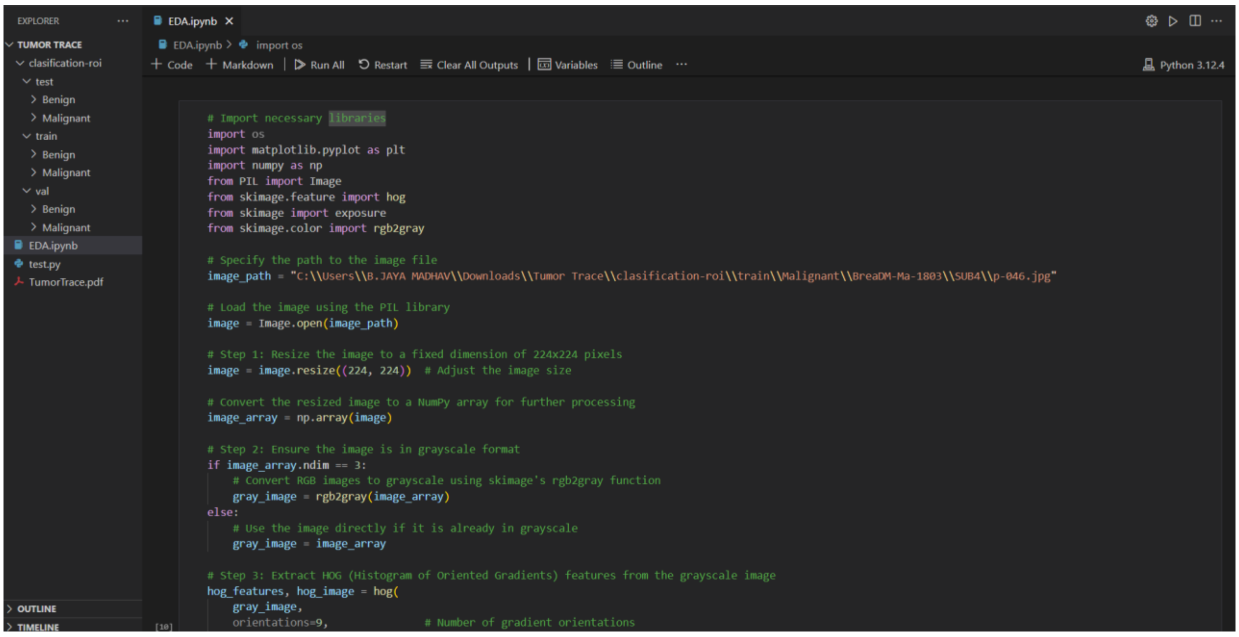


```python
import os
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image

# Define the path to the specific image
image_path = r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-2106\SUB6\p-067.jpg"

# Load the image
image = Image.open(image_path)

# Resize the image to 224x224
image = image.resize((224, 224))

# Ensure the image is in RGB format
image = image.convert('RGB')

# Convert the image to a NumPy array
image_array = np.array(image)

# Flatten the image into a 1D array (for easier plotting of pixel values)
flat_image_array = image_array.flatten()

# Plotting the histogram of pixel values
plt.figure(figsize=(8, 6))
plt.hist(flat_image_array, bins=256, range=(0, 255), color='blue', alpha=0.7)
plt.title('Pixel Value Distribution')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

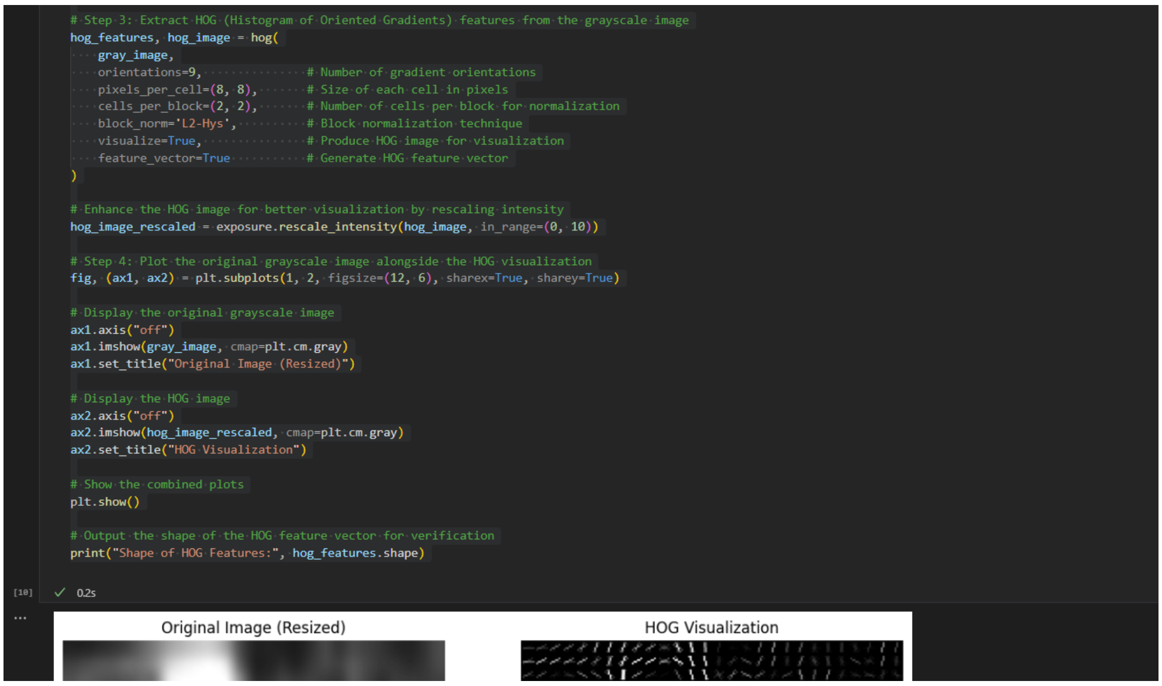
```
[40]  ✓  0.2s
```



HOG:

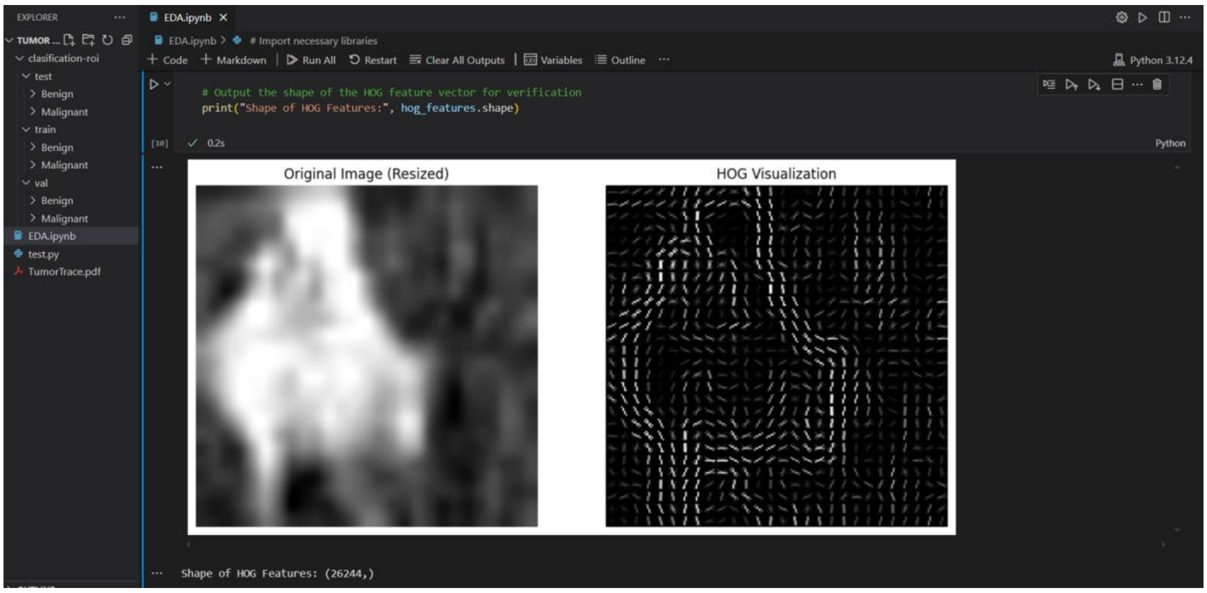


```python
# Import necessary libraries
import os
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
from skimage.feature import hog
from skimage import exposure
from skimage.color import rgb2gray

# Specify the path to the image file
image_path = "C:\\Users\\B.JAYA MADHAV\\Downloads\\Tumor Trace\\clasification-roi\\train\\Malignant\\BreaDM-Ma-1803\\SUB4\\p-046.jpg"

# Load the image using the PIL library
image = Image.open(image_path)

# Step 1: Resize the image to a fixed dimension of 224x224 pixels
image = image.resize((224, 224))  # Adjust the image size

# Convert the resized image to a NumPy array for further processing
image_array = np.array(image)

# Step 2: Ensure the image is in grayscale format
if image_array.ndim == 3:
    # Convert RGB images to grayscale using skimage's rgb2gray function
    gray_image = rgb2gray(image_array)
else:
    # Use the image directly if it is already in grayscale
    gray_image = image_array

# Step 3: Extract HOG (Histogram of Oriented Gradients) features from the grayscale image
hog_features, hog_image = hog(
    gray_image,
    orientations=9,            # Number of gradient orientations
```



```python
# Step 3: Extract HOG (Histogram of Oriented Gradients) features from the grayscale image
hog_features, hog_image = hog(
    gray_image,
    orientations=9,              # Number of gradient orientations
    pixels_per_cell=(8, 8),      # Size of each cell in pixels
    cells_per_block=(2, 2),      # Number of cells per block for normalization
    block_norm='L2-Hys',         # Block normalization technique
    visualize=True,              # Produce HOG image for visualization
    feature_vector=True          # Generate HOG feature vector
)

# Enhance the HOG image for better visualization by rescaling intensity
hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))

# Step 4: Plot the original grayscale image alongside the HOG visualization
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6), sharex=True, sharey=True)

# Display the original grayscale image
ax1.axis("off")
ax1.imshow(gray_image, cmap=plt.cm.gray)
ax1.set_title("Original Image (Resized)")

# Display the HOG image
ax2.axis("off")
ax2.imshow(hog_image_rescaled, cmap=plt.cm.gray)
ax2.set_title("HOG Visualization")

# Show the combined plots
plt.show()

# Output the shape of the HOG feature vector for verification
print("Shape of HOG Features:", hog_features.shape)
```

[10] ✓ 0.2s

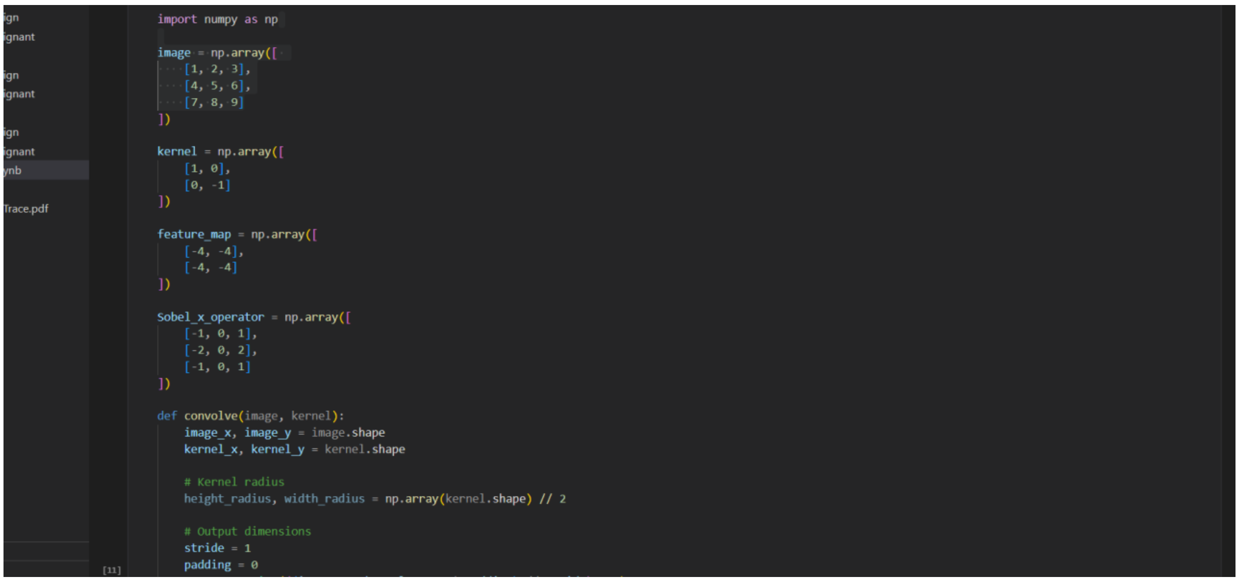


```python
# Output the shape of the HOG feature vector for verification
print("Shape of HOG Features:", hog_features.shape)
```

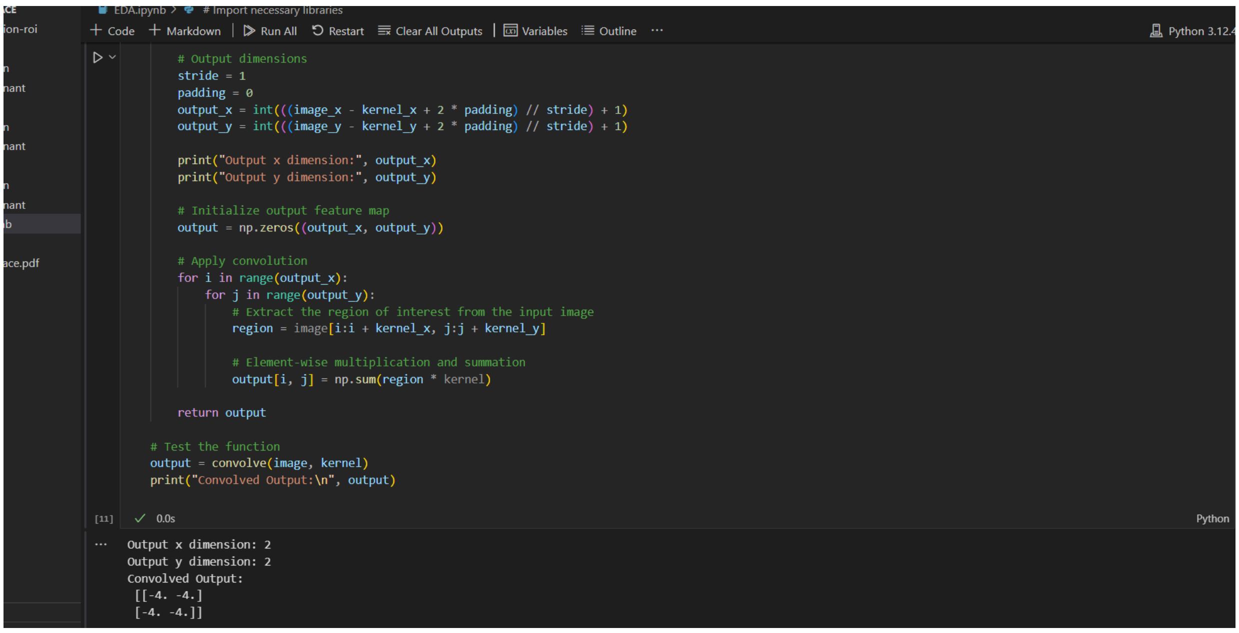
[10] ✓ 0.2s

```
Shape of HOG Features: (26244,)
```

SOBEL OPERATOR, SOBEL EDGE DETECTION:



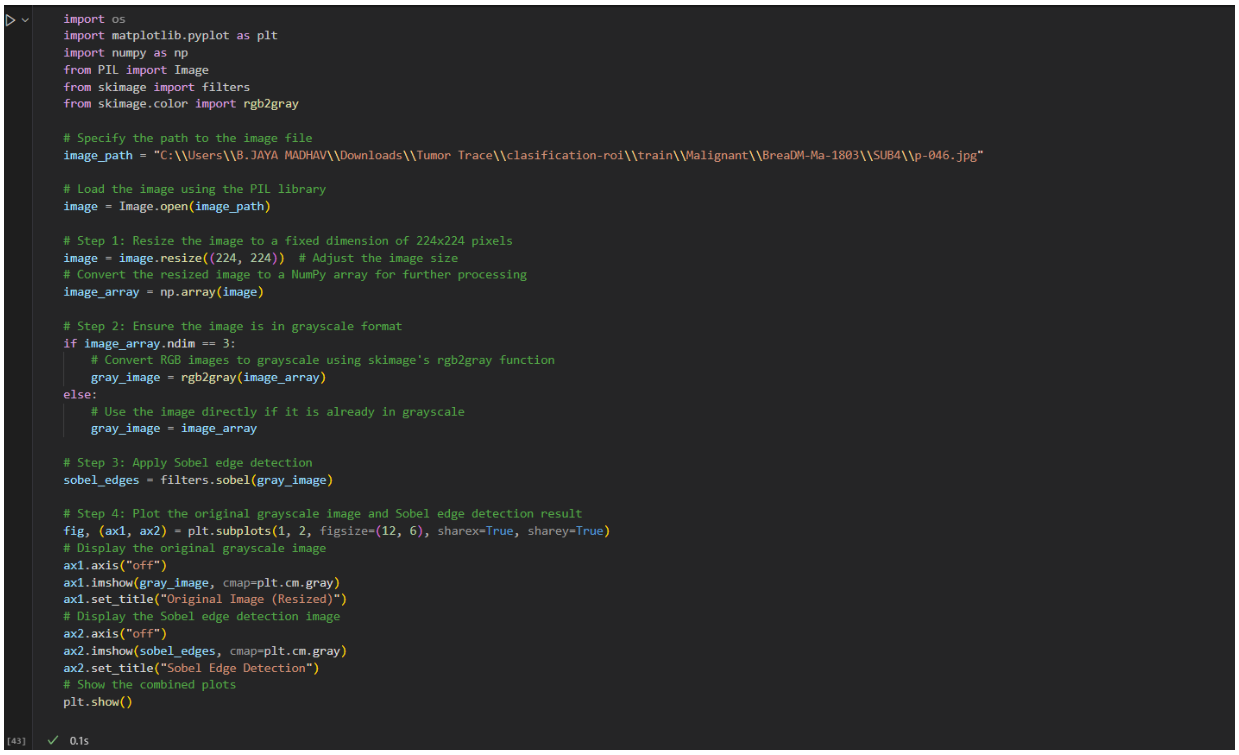
```python
import numpy as np

image = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])

kernel = np.array([
    [1, 0],
    [0, -1]
])

feature_map = np.array([
    [-4, -4],
    [-4, -4]
])

Sobel_x_operator = np.array([
    [-1, 0, 1],
    [-2, 0, 2],
    [-1, 0, 1]
])

def convolve(image, kernel):
    image_x, image_y = image.shape
    kernel_x, kernel_y = kernel.shape

    # Kernel radius
    height_radius, width_radius = np.array(kernel.shape) // 2

    # Output dimensions
    stride = 1
    padding = 0
```


EDA.ipynb > # Import necessary libraries
+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ···
Python 3.12.4

```python
    # Output dimensions
    stride = 1
    padding = 0
    output_x = int(((image_x - kernel_x + 2 * padding) // stride) + 1)
    output_y = int(((image_y - kernel_y + 2 * padding) // stride) + 1)

    print("Output x dimension:", output_x)
    print("Output y dimension:", output_y)

    # Initialize output feature map
    output = np.zeros((output_x, output_y))

    # Apply convolution
    for i in range(output_x):
        for j in range(output_y):
            # Extract the region of interest from the input image
            region = image[i:i + kernel_x, j:j + kernel_y]

            # Element-wise multiplication and summation
            output[i, j] = np.sum(region * kernel)

    return output

# Test the function
output = convolve(image, kernel)
print("Convolved Output:\n", output)
```

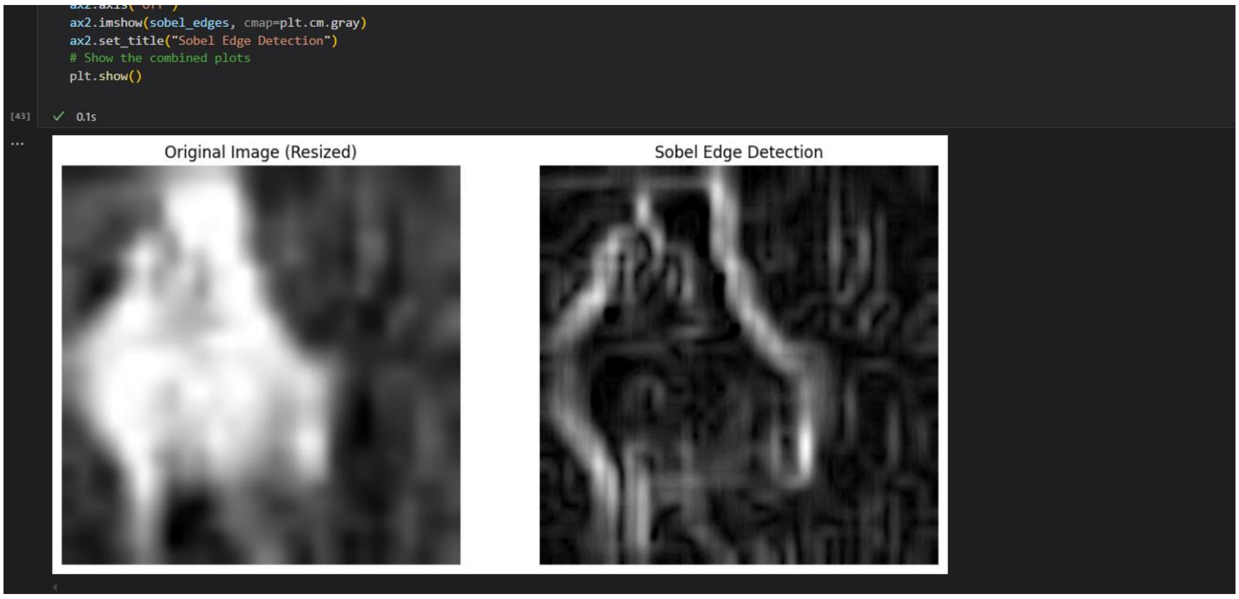
[11] ✓ 0.0s

```
Output x dimension: 2
Output y dimension: 2
Convolved Output:
 [[-4. -4.]
 [-4. -4.]]
```



```python
import os
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
from skimage import filters
from skimage.color import rgb2gray

# Specify the path to the image file
image_path = "C:\\Users\\B.JAYA MADHAV\\Downloads\\Tumor Trace\\clasification-roi\\train\\Malignant\\BreaDM-Ma-1803\\SUB4\\p-046.jpg"

# Load the image using the PIL library
image = Image.open(image_path)

# Step 1: Resize the image to a fixed dimension of 224x224 pixels
image = image.resize((224, 224))  # Adjust the image size
# Convert the resized image to a NumPy array for further processing
image_array = np.array(image)

# Step 2: Ensure the image is in grayscale format
if image_array.ndim == 3:
    # Convert RGB images to grayscale using skimage's rgb2gray function
    gray_image = rgb2gray(image_array)
else:
    # Use the image directly if it is already in grayscale
    gray_image = image_array

# Step 3: Apply Sobel edge detection
sobel_edges = filters.sobel(gray_image)

# Step 4: Plot the original grayscale image and Sobel edge detection result
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6), sharex=True, sharey=True)
# Display the original grayscale image
ax1.axis("off")
ax1.imshow(gray_image, cmap=plt.cm.gray)
ax1.set_title("Original Image (Resized)")
# Display the Sobel edge detection image
ax2.axis("off")
ax2.imshow(sobel_edges, cmap=plt.cm.gray)
ax2.set_title("Sobel Edge Detection")
# Show the combined plots
plt.show()
```

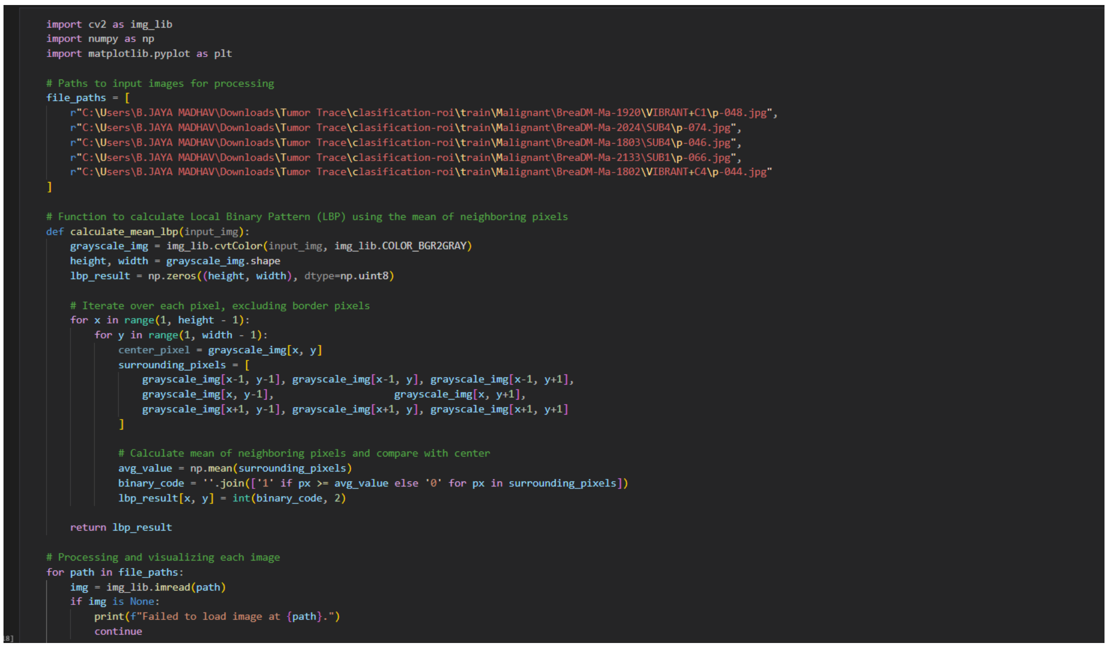
[43] ✓ 0.1s



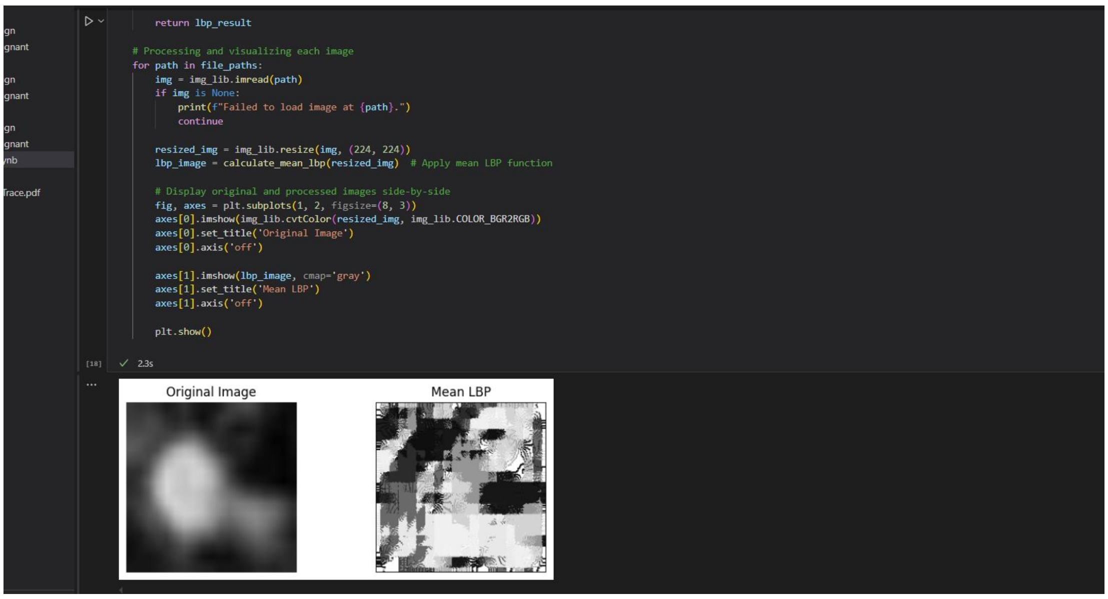
```python
ax2.imshow(sobel_edges, cmap=plt.cm.gray)
ax2.set_title("Sobel Edge Detection")
# Show the combined plots
plt.show()
```

[43] ✓ 0.1s

MEAN LBP:

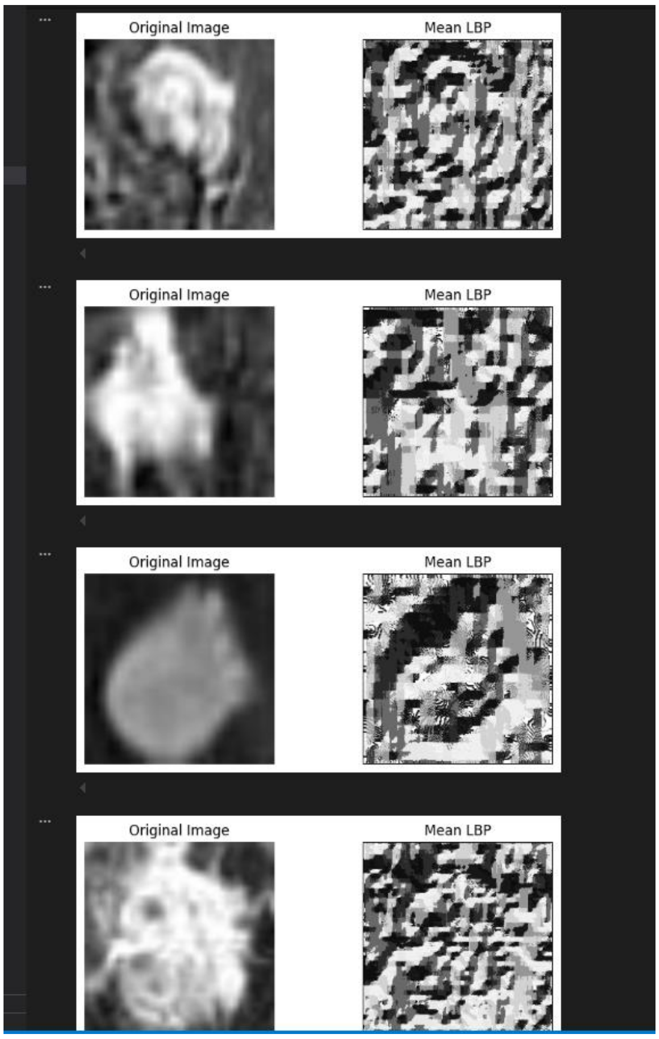


```python
import cv2 as img_lib
import numpy as np
import matplotlib.pyplot as plt

# Paths to input images for processing
file_paths = [
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-1920\VIBRANT+C1\p-048.jpg",
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-2024\SUB4\p-074.jpg",
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-1803\SUB4\p-046.jpg",
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-2133\SUB1\p-066.jpg",
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-1802\VIBRANT+C4\p-044.jpg"
]

# Function to calculate Local Binary Pattern (LBP) using the mean of neighboring pixels
def calculate_mean_lbp(input_img):
    grayscale_img = img_lib.cvtColor(input_img, img_lib.COLOR_BGR2GRAY)
    height, width = grayscale_img.shape
    lbp_result = np.zeros((height, width), dtype=np.uint8)

    # Iterate over each pixel, excluding border pixels
    for x in range(1, height - 1):
        for y in range(1, width - 1):
            center_pixel = grayscale_img[x, y]
            surrounding_pixels = [
                grayscale_img[x-1, y-1], grayscale_img[x-1, y], grayscale_img[x-1, y+1],
                grayscale_img[x, y-1],                          grayscale_img[x, y+1],
                grayscale_img[x+1, y-1], grayscale_img[x+1, y], grayscale_img[x+1, y+1]
            ]

            # Calculate mean of neighboring pixels and compare with center
            avg_value = np.mean(surrounding_pixels)
            binary_code = ''.join(['1' if px >= avg_value else '0' for px in surrounding_pixels])
            lbp_result[x, y] = int(binary_code, 2)

    return lbp_result

# Processing and visualizing each image
for path in file_paths:
    img = img_lib.imread(path)
    if img is None:
        print(f"Failed to load image at {path}.")
        continue
```

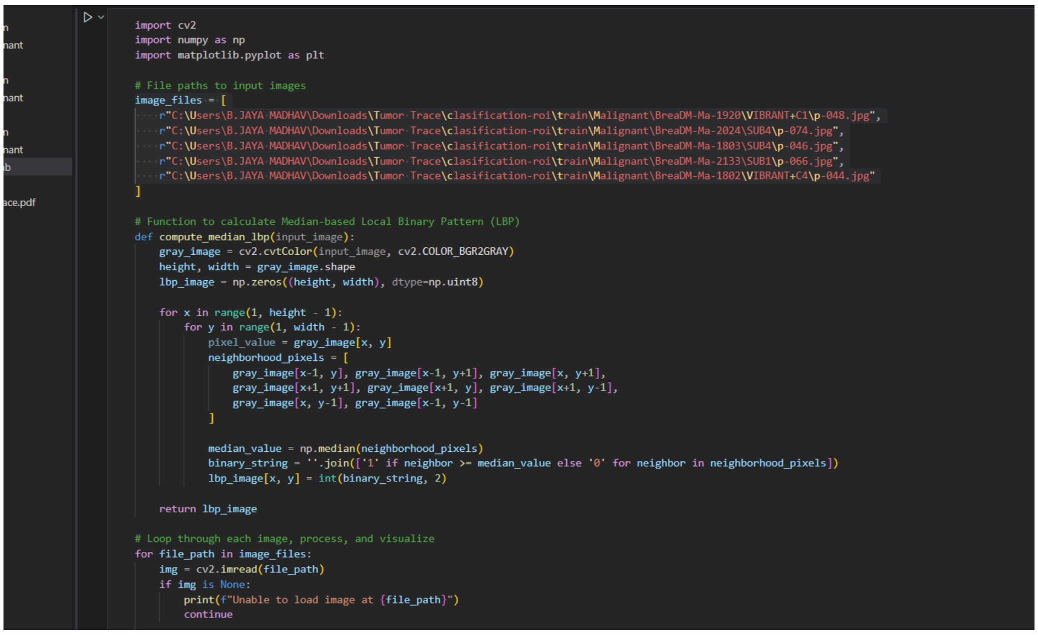


```python
    return lbp_result

# Processing and visualizing each image
for path in file_paths:
    img = img_lib.imread(path)
    if img is None:
        print(f"Failed to load image at {path}.")
        continue

    resized_img = img_lib.resize(img, (224, 224))
    lbp_image = calculate_mean_lbp(resized_img)  # Apply mean LBP function

    # Display original and processed images side-by-side
    fig, axes = plt.subplots(1, 2, figsize=(8, 3))
    axes[0].imshow(img_lib.cvtColor(resized_img, img_lib.COLOR_BGR2RGB))
    axes[0].set_title('Original Image')
    axes[0].axis('off')

    axes[1].imshow(lbp_image, cmap='gray')
    axes[1].set_title('Mean LBP')
    axes[1].axis('off')

    plt.show()
```

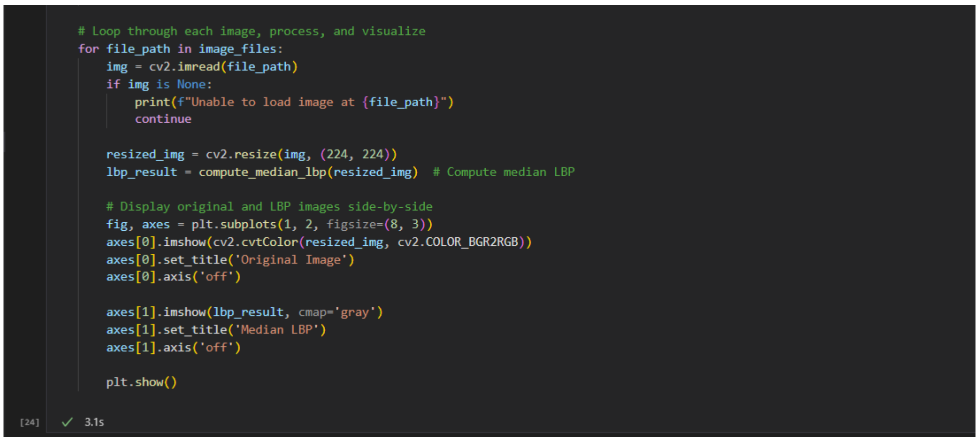
```
[18]  ✓ 2.3s
```



MEDIAN LBP:

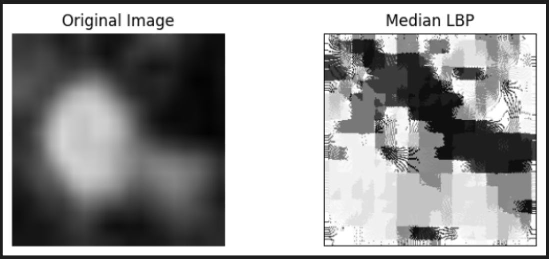


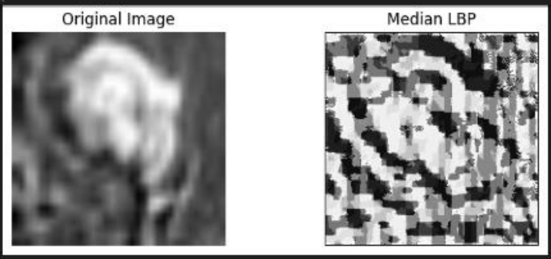
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# File paths to input images
image_files = [
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-1920\VIBRANT+C1\p-048.jpg",
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-2024\SUB4\p-074.jpg",
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-1803\SUB4\p-046.jpg",
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-2133\SUB1\p-066.jpg",
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-1802\VIBRANT+C4\p-044.jpg"
]

# Function to calculate Median-based Local Binary Pattern (LBP)
def compute_median_lbp(input_image):
    gray_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2GRAY)
    height, width = gray_image.shape
    lbp_image = np.zeros((height, width), dtype=np.uint8)

    for x in range(1, height - 1):
        for y in range(1, width - 1):
            pixel_value = gray_image[x, y]
            neighborhood_pixels = [
                gray_image[x-1, y], gray_image[x-1, y+1], gray_image[x, y+1],
                gray_image[x+1, y+1], gray_image[x+1, y], gray_image[x+1, y-1],
                gray_image[x, y-1], gray_image[x-1, y-1]
            ]

            median_value = np.median(neighborhood_pixels)
            binary_string = ''.join(['1' if neighbor >= median_value else '0' for neighbor in neighborhood_pixels])
            lbp_image[x, y] = int(binary_string, 2)

    return lbp_image

# Loop through each image, process, and visualize
for file_path in image_files:
    img = cv2.imread(file_path)
    if img is None:
        print(f"Unable to load image at {file_path}")
        continue
```

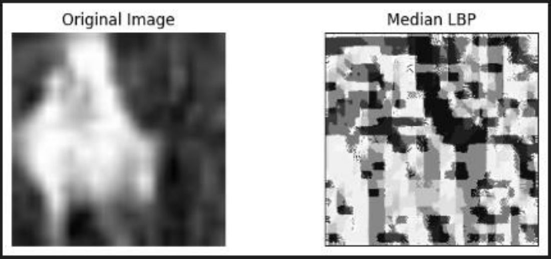


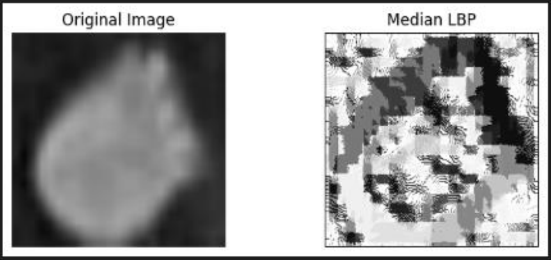
```
# Loop through each image, process, and visualize
for file_path in image_files:
    img = cv2.imread(file_path)
    if img is None:
        print(f"Unable to load image at {file_path}")
        continue

    resized_img = cv2.resize(img, (224, 224))
    lbp_result = compute_median_lbp(resized_img)  # Compute median LBP

    # Display original and LBP images side-by-side
    fig, axes = plt.subplots(1, 2, figsize=(8, 3))
    axes[0].imshow(cv2.cvtColor(resized_img, cv2.COLOR_BGR2RGB))
    axes[0].set_title('Original Image')
    axes[0].axis('off')

    axes[1].imshow(lbp_result, cmap='gray')
    axes[1].set_title('Median LBP')
    axes[1].axis('off')

    plt.show()
```

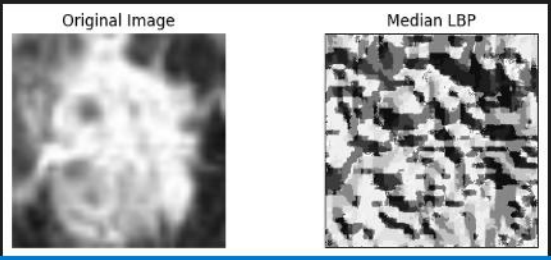
[24] ✓ 3.1s



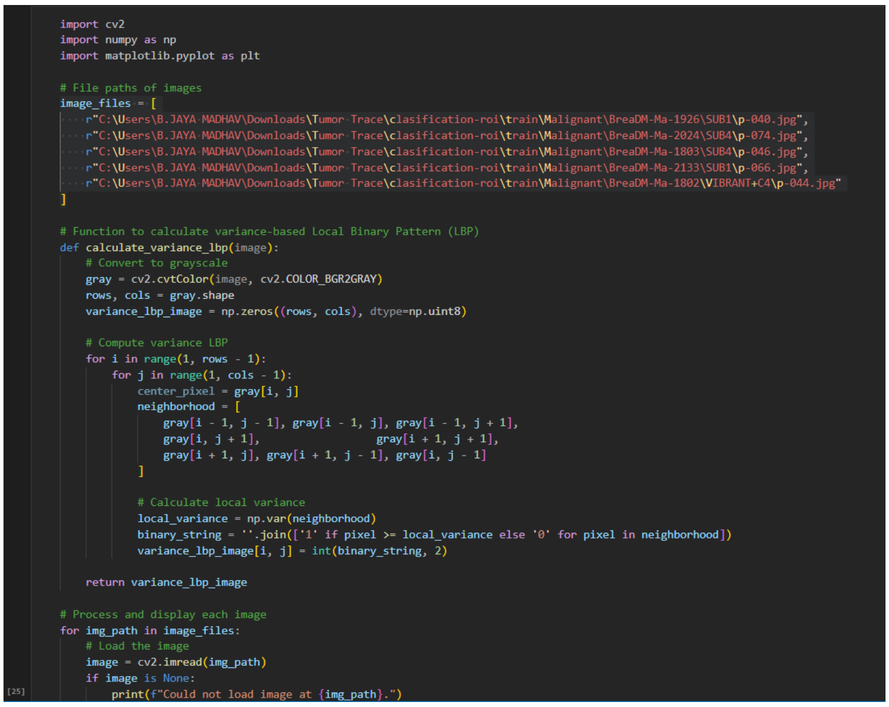




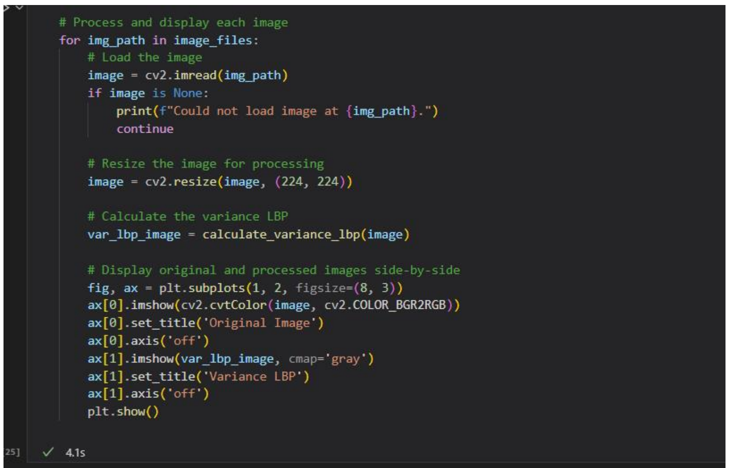




VARIANCE LBP:

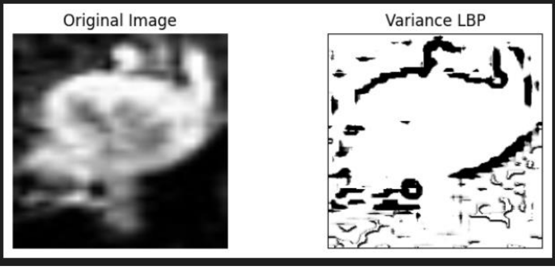


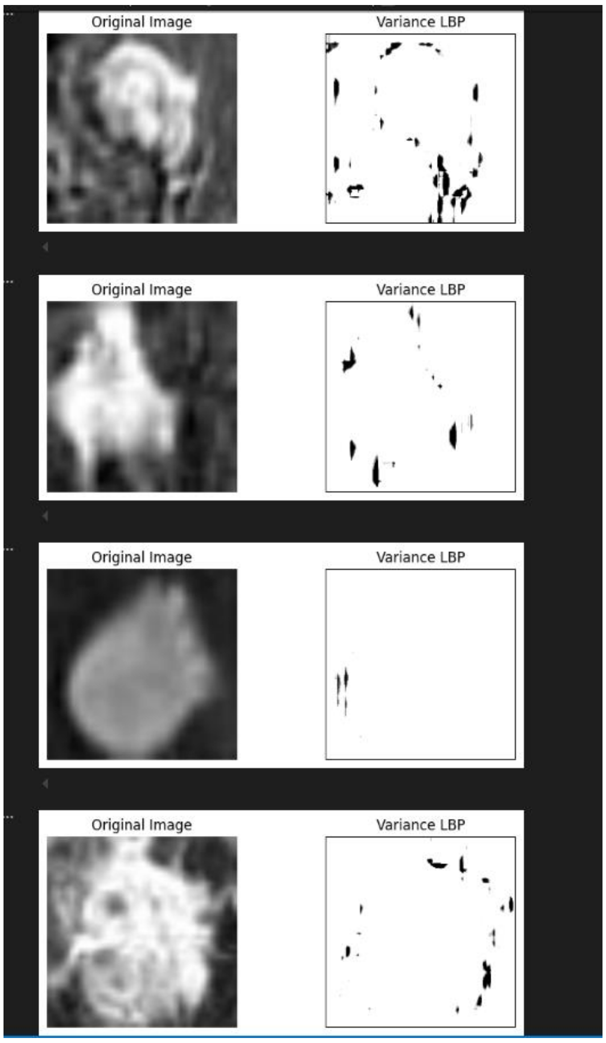
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# File paths of images
image_files = [
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-1926\SUB1\p-040.jpg",
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-2024\SUB4\p-074.jpg",
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-1803\SUB4\p-046.jpg",
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-2133\SUB1\p-066.jpg",
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-1802\VIBRANT+C4\p-044.jpg"
]

# Function to calculate variance-based Local Binary Pattern (LBP)
def calculate_variance_lbp(image):
    # Convert to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    rows, cols = gray.shape
    variance_lbp_image = np.zeros((rows, cols), dtype=np.uint8)

    # Compute variance LBP
    for i in range(1, rows - 1):
        for j in range(1, cols - 1):
            center_pixel = gray[i, j]
            neighborhood = [
                gray[i - 1, j - 1], gray[i - 1, j], gray[i - 1, j + 1],
                gray[i, j + 1],                 gray[i + 1, j + 1],
                gray[i + 1, j], gray[i + 1, j - 1], gray[i, j - 1]
            ]

            # Calculate local variance
            local_variance = np.var(neighborhood)
            binary_string = ''.join(['1' if pixel >= local_variance else '0' for pixel in neighborhood])
            variance_lbp_image[i, j] = int(binary_string, 2)

    return variance_lbp_image

# Process and display each image
for img_path in image_files:
    # Load the image
    image = cv2.imread(img_path)
    if image is None:
        print(f"Could not load image at {img_path}.")
```



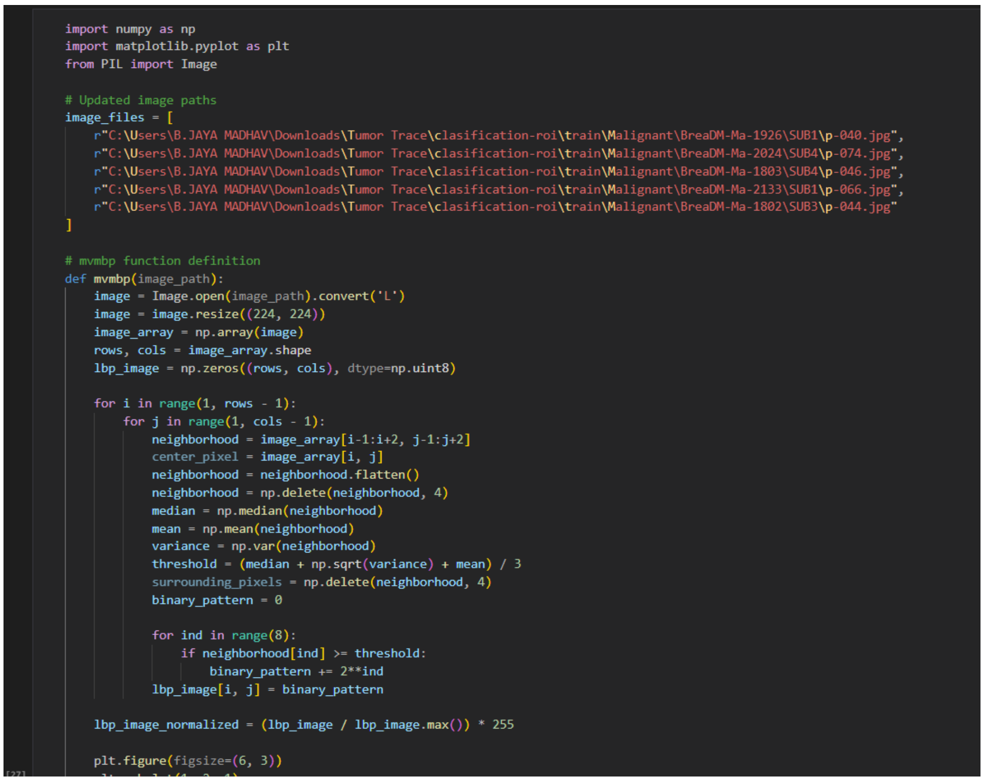
```python
# Process and display each image
for img_path in image_files:
    # Load the image
    image = cv2.imread(img_path)
    if image is None:
        print(f"Could not load image at {img_path}.")
        continue

    # Resize the image for processing
    image = cv2.resize(image, (224, 224))

    # Calculate the variance LBP
    var_lbp_image = calculate_variance_lbp(image)

    # Display original and processed images side-by-side
    fig, ax = plt.subplots(1, 2, figsize=(8, 3))
    ax[0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    ax[0].set_title('Original Image')
    ax[0].axis('off')
    ax[1].imshow(var_lbp_image, cmap='gray')
    ax[1].set_title('Variance LBP')
    ax[1].axis('off')
    plt.show()
```

```
✓ 4.1s
```

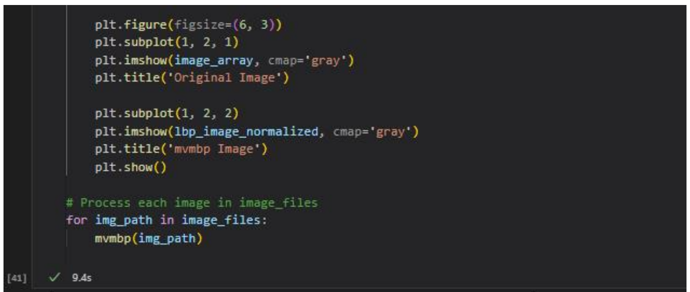




MVMBP:



```python
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

# Updated image paths
image_files = [
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-1926\SUB1\p-040.jpg",
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-2024\SUB4\p-074.jpg",
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-1803\SUB4\p-046.jpg",
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-2133\SUB1\p-066.jpg",
    r"C:\Users\B.JAYA MADHAV\Downloads\Tumor Trace\clasification-roi\train\Malignant\BreaDM-Ma-1802\SUB3\p-044.jpg"
]

# mvmbp function definition
def mvmbp(image_path):
    image = Image.open(image_path).convert('L')
    image = image.resize((224, 224))
    image_array = np.array(image)
    rows, cols = image_array.shape
    lbp_image = np.zeros((rows, cols), dtype=np.uint8)

    for i in range(1, rows - 1):
        for j in range(1, cols - 1):
            neighborhood = image_array[i-1:i+2, j-1:j+2]
            center_pixel = image_array[i, j]
            neighborhood = neighborhood.flatten()
            neighborhood = np.delete(neighborhood, 4)
            median = np.median(neighborhood)
            mean = np.mean(neighborhood)
            variance = np.var(neighborhood)
            threshold = (median + np.sqrt(variance) + mean) / 3
            surrounding_pixels = np.delete(neighborhood, 4)
            binary_pattern = 0

            for ind in range(8):
                if neighborhood[ind] >= threshold:
                    binary_pattern += 2**ind
            lbp_image[i, j] = binary_pattern

    lbp_image_normalized = (lbp_image / lbp_image.max()) * 255

    plt.figure(figsize=(6, 3))
```



```
        plt.figure(figsize=(6, 3))
        plt.subplot(1, 2, 1)
        plt.imshow(image_array, cmap='gray')
        plt.title('Original Image')

        plt.subplot(1, 2, 2)
        plt.imshow(lbp_image_normalized, cmap='gray')
        plt.title('mvmbp Image')
        plt.show()

    # Process each image in image_files
    for img_path in image_files:
        mvmbp(img_path)
```

[41] ✓ 9.4s