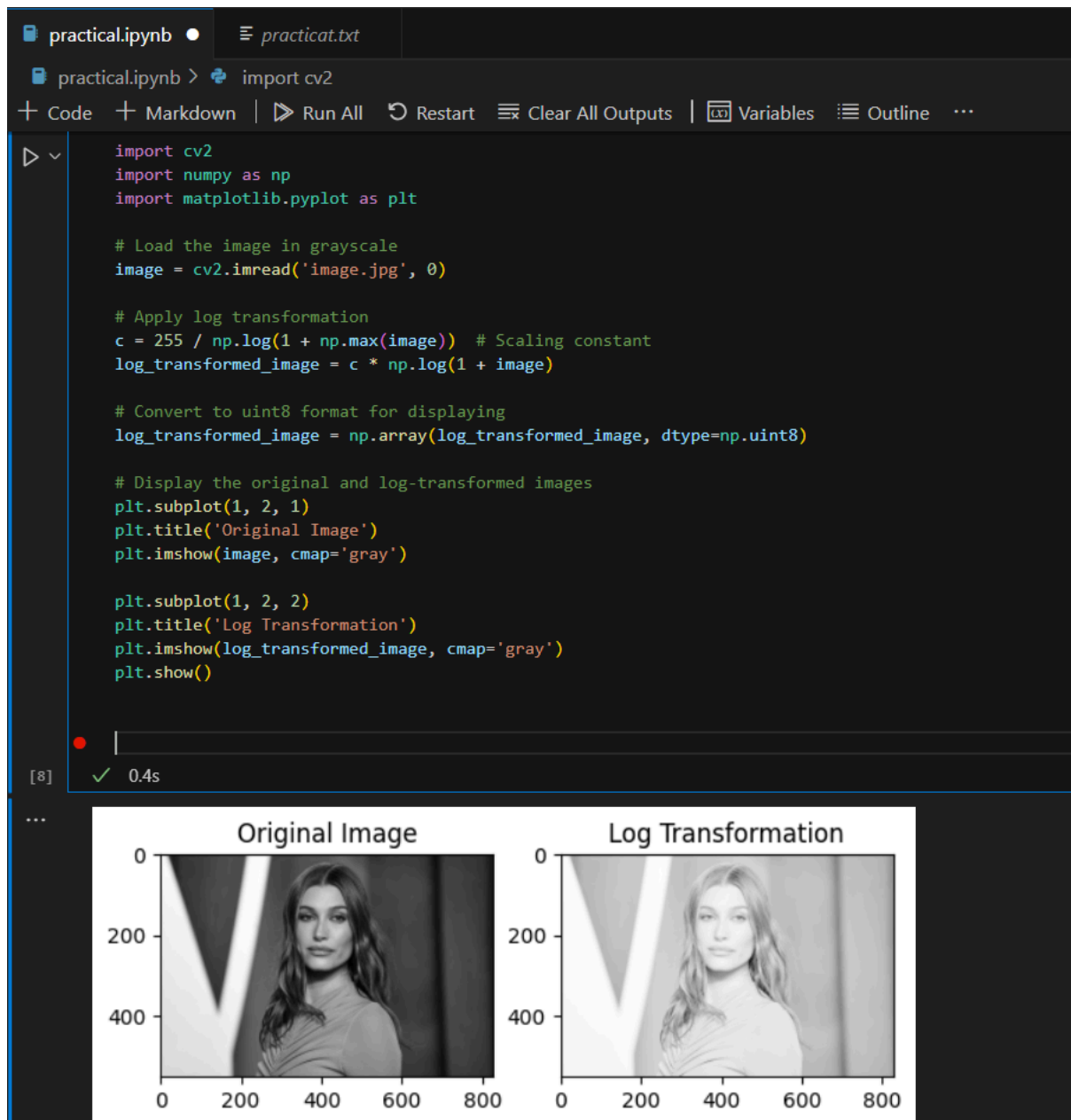


Practical - 07

Aim: Write a program to perform log transformation of an image.

Theory: Log transformation enhances low-intensity pixels in an image, making darker areas brighter. It's defined by the formula $s = c * \log(1 + r)$, where c is a constant, and r is the pixel intensity.

Code and Output :



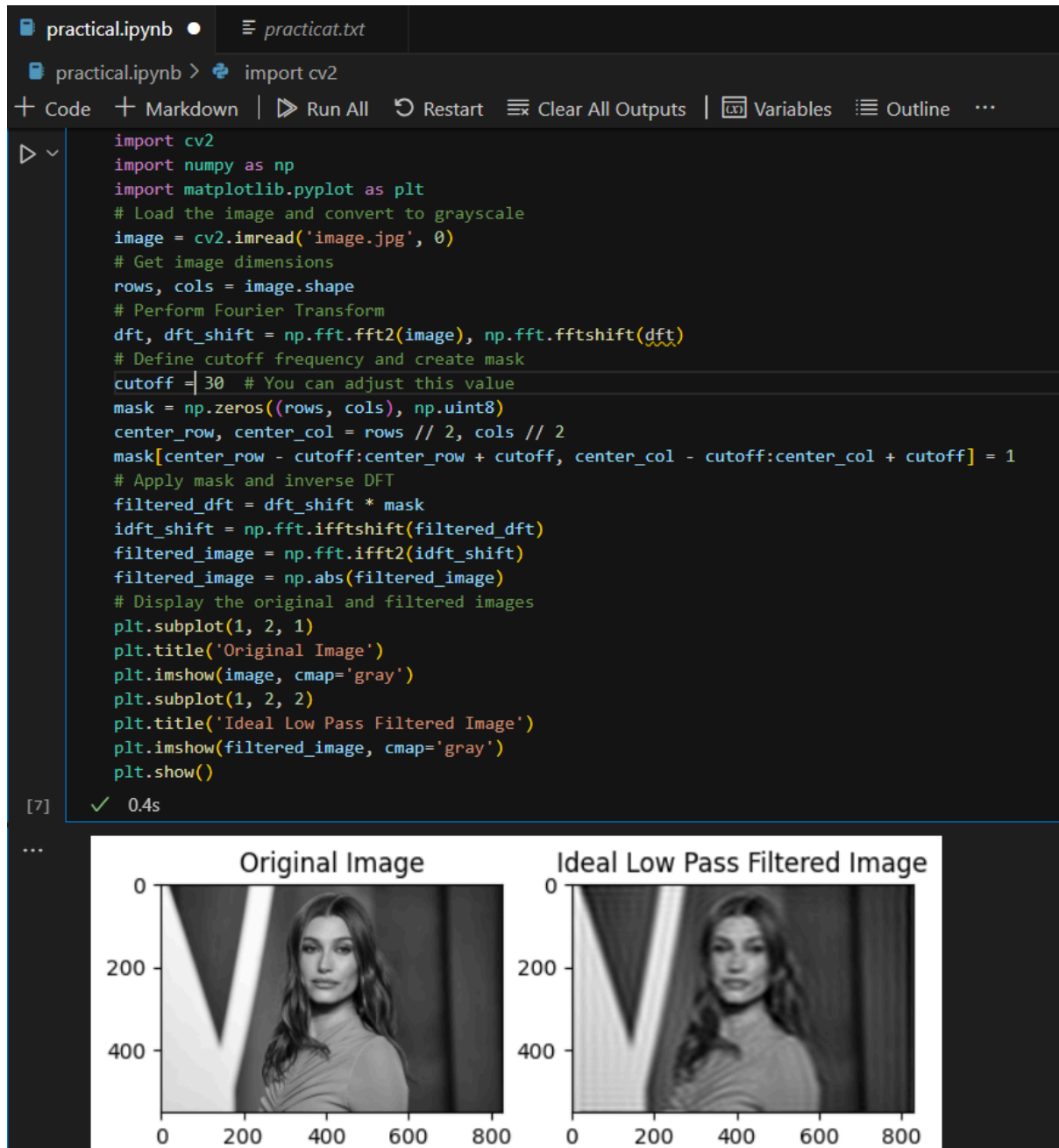
Practical - 08

Aim: Write a program to implement ideal low pass filter

Theory:

An Ideal Low-Pass Filter (ILPF) blocks high frequencies beyond a certain cutoff, retaining only low-frequency components. This sharp cutoff creates a ringing effect, blurring the image while losing high-frequency details like edges.

Code and Output :



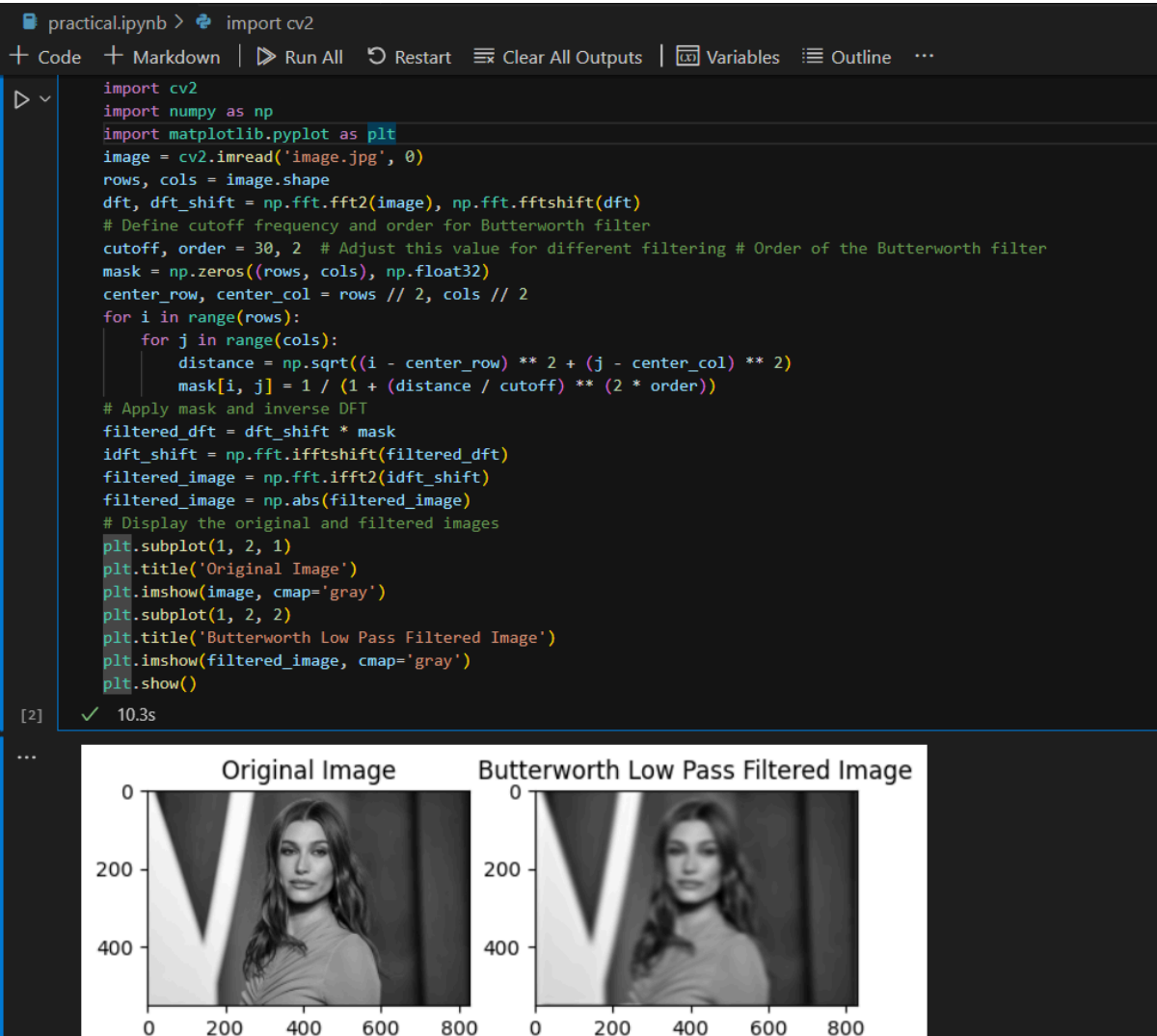
Practical - 09

Aim: Write a program to implement Butterworth low pass filter

Theory:

The Butterworth Low-Pass Filter provides a gradual cutoff to reduce high frequencies smoothly, defined by its cutoff frequency and filter order. It balances sharpness and smoothness without harsh transitions, preserving more details than an ideal low-pass filter.

Code and Output :



Practical - 10

Aim: Write a program to implement gaussian low pass filter

Theory:

A Gaussian Low-Pass Filter applies a Gaussian function to attenuate high frequencies in a smooth, bell-shaped curve, avoiding abrupt changes. This filter minimizes the ringing effect and is commonly used to blur images while maintaining a natural appearance.

Code and Output :


```
practical.ipynb > import cv2
+ Code + Markdown | ▶ Run All ⏮ Restart ⌵ Clear All Outputs | 📄 Variables 📄 Outline ...
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('image.jpg', 0)
rows, cols = image.shape
dft = np.fft.fft2(image)
dft_shift = np.fft.fftshift(dft)
cutoff = 30 # Adjust this value as needed
# Create Gaussian filter mask
mask = np.zeros((rows, cols), np.float32)
center_row, center_col = rows // 2, cols // 2
for i in range(rows):
    for j in range(cols):
        distance = np.sqrt((i - center_row) ** 2 + (j - center_col) ** 2)
        mask[i, j] = np.exp(-(distance ** 2) / (2 * (cutoff ** 2)))
filtered_dft = dft_shift * mask
idft_shift = np.fft.ifftshift(filtered_dft)
filtered_image = np.fft.ifft2(idft_shift)
filtered_image = np.abs(filtered_image)
# Display the original and filtered images
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.subplot(1, 2, 2)
plt.title('Gaussian Low Pass Filtered Image')
plt.imshow(filtered_image, cmap='gray')
plt.show()
```

[3] ✓ 4.0s


...

Original Image



0 200 400 0 200 400 0 200 400 600 800

Gaussian Low Pass Filtered Image



0 200 400 0 200 400 0 200 400 600 800

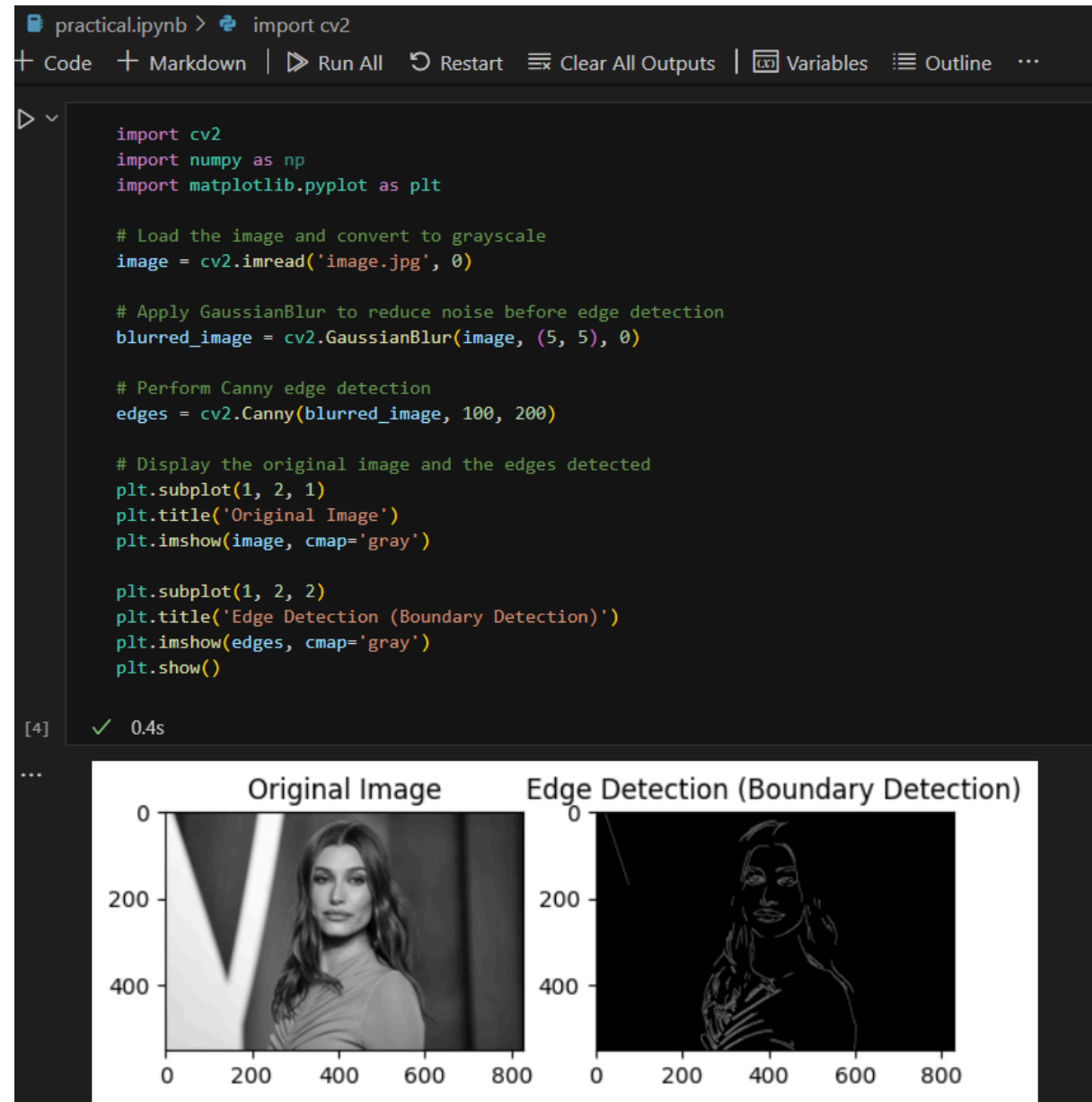
Practical - 11

Aim: Write a program to boundary detection in python

Theory:

Boundary detection focuses on identifying the edges or contours within an image by analyzing changes in pixel intensity. It emphasizes regions with significant intensity shifts, isolating edges that represent object boundaries, enabling accurate segmentation.

Code and Output :



Practical - 12

Aim: Write a program to neural network edge detection

Theory:

Neural network edge detection employs convolutional neural networks (CNNs) trained to detect edges. The network learns complex patterns and captures fine-grained edges by recognizing spatial structures and intensity gradients, offering more accuracy than traditional filters.

Code and Output :

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

# Load the HED neural network model (prototxt and caffemodel paths)
net = cv2.dnn.readNetFromCaffe('deploy.prototxt', 'hed_pretrained_bsds.caffemodel')

# Load and preprocess the image
image = cv2.imread('image.jpg')
image = cv2.resize(image, (500, 500)) # Resize for faster processing
(h, w) = image.shape[:2]

# Preprocess the image for the neural network
blob = cv2.dnn.blobFromImage(image, scalefactor=1.0, size=(w, h),
                               mean=(104.00698793, 116.66876762, 122.67891434),
                               swapRB=False, crop=False)

# Set the blob as input to the network
net.setInput(blob)

# Perform forward pass to get the edge map
edge_map = net.forward()
edge_map = edge_map[0, 0] # Extract the single channel output
edge_map = cv2.resize(edge_map, (w, h)) # Resize to match input image size

# Convert edge map to displayable format
edge_map = (255 * edge_map).astype("uint8")

# Display the original image and the edge detection result
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

plt.subplot(1, 2, 2)
plt.title('Edge Detection (Neural Network)')
plt.imshow(edge_map, cmap='gray')
plt.show()
```

OUTPUT :

