```
import cv2
import os
import matplotlib.pyplot as plt

# Define the input and output directories
input_dir = 'PATH TO INPUT FILE/DIRECTORY'
output_dir = 'PATH TO OUTPUT FILE/DIRECTORY'

# Create the output directory
os.makedirs(output_dir, exist_ok=True)

# Process each image in the input directory
for filename in sorted(os.listdir(input_dir)):
    if not filename.lower().endswith(('.png', '.jpg', '.jpeg', '.tif',
'.tiff')):  # Filter non-image files
        continue

    img = cv2.imread(os.path.join(input_dir, filename))

    # Checking if the image was loaded correctly
    if img is None:
        print(f"Error loading image: {filename}")
        continue

    # Extract the green channel
    green_channel = img[:, :, 1]

    # Save the green channel image
    green_channel_path = os.path.join(output_dir, f'g-
channel_{filename}')
    cv2.imwrite(green_channel_path, green_channel)

    # Display the original and green channel images
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title(f'Original Image: {filename}')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(green_channel, cmap='gray')
    plt.title(f'Green Channel: {filename}')
    plt.axis('off')

    plt.show()
```


```
import cv2
import os
import matplotlib.pyplot as plt
```


```
# Define directories
```

```
input_dir = 'PATH TO INPUT FILE/DIRECTORY'
output_dir = 'PATH TO OUTPUT FILE/DIRECTORY'

# Create output directory
os.makedirs(output_dir, exist_ok=True)

# Initialize image counter
image_count = 1

# Process each image
for filename in sorted(os.listdir(input_dir)):
    if not filename.lower().endswith(('.png', '.jpg', '.jpeg', '.tif',
'.tiff')):  # Filter non-image files
        continue

    img = cv2.imread(os.path.join(input_dir, filename),
cv2.IMREAD_GRAYSCALE)

    # Check if the image was loaded correctly
    if img is None:
        print(f"Error loading image: {filename}")
        continue

    # Apply adaptive thresholding
    adaptive_thresh = cv2.adaptiveThreshold(img, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                            cv2.THRESH_BINARY_INV, 5,
2)  # Inverted thresholding

    # Save the thresholded image
    thresholded_path = os.path.join(output_dir,
f'adaptive_thresholding_{filename}')
    cv2.imwrite(thresholded_path, adaptive_thresh)

    # Display the images with numbering
    plt.imshow(adaptive_thresh, cmap='gray')
    plt.title(f'Adaptive Thresholding: {image_count}')
    plt.axis('off')
    plt.show()

    # Increment the counter
    image_count += 1

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

# Define directories
input_dir = 'PATH TO INPUT FILE/DIRECTORY'
```

```
output_dir = 'PATH TO OUTPUT FILE/DIRECTORY'

# Create output directory
os.makedirs(output_dir, exist_ok=True)

# Initialize image counter
image_count = 1

# Process each image
for filename in sorted(os.listdir(input_dir)):
    if not filename.lower().endswith(('.png', '.jpg', '.jpeg', '.tif',
'.tiff')):  # Filter non-image files
        continue

    img = cv2.imread(os.path.join(input_dir, filename))

    # Check if the image was loaded correctly
    if img is None:
        print(f"Error loading image: {filename}")
        continue

    # Convert the image from RGB to Lab color space
    lab_img = cv2.cvtColor(img, cv2.COLOR_BGR2Lab)

    # Split the channels
    L, a, b = cv2.split(lab_img)

    # Apply sharpening filter to the L channel
    sharpening_kernel = np.array([[0, -1, 0],
                                  [-1, 9, -1],
                                  [0, -1, 0]])
    L_sharpened = cv2.filter2D(L, -1, sharpening_kernel)

    # Merge the sharpened L channel with the original a and b channels
    lab_sharpened = cv2.merge((L_sharpened, a, b))

    # Convert back to RGB color space
    sharpened_img = cv2.cvtColor(lab_sharpened, cv2.COLOR_Lab2BGR)

    # Unsharp Masking
    blurred = cv2.GaussianBlur(sharpened_img, (5, 5), 1.0)  # Adjust
the kernel size and sigma as needed
    unsharp_masked = cv2.addWeighted(sharpened_img, 1.5, blurred, -
0.5, 0)  # Adjust weights as needed

    # Save the final image
    final_path = os.path.join(output_dir, f'sharpened_{filename}')
    cv2.imwrite(final_path, unsharp_masked)

    # Display the images with numbering
```

```
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title(f'Adaptive Thresholding: {image_count}')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(unsharp_masked, cv2.COLOR_BGR2RGB))
    plt.title(f'Unsharp Masked: {image_count}')
    plt.axis('off')

    plt.show()

    # Increment the counter
    image_count += 1

import cv2
import os
import numpy as np
import matplotlib.pyplot as plt
from skimage.restoration import denoise_wavelet

# Define directories
input_dir = 'PATH TO INPUT FILE/DIRECTORY'
output_dir = 'PATH TO OUTPUT FILE/DIRECTORY'
# Create output directory
os.makedirs(output_dir, exist_ok=True)

# Initialize image counter
image_count = 1  # Update this with the last count from the previous
code if needed

# Process each image
for filename in sorted(os.listdir(input_dir)):
    if not filename.lower().endswith(('.png', '.jpg', '.jpeg', '.tif',
'.tiff')):  # Filter non-image files
        continue

    img = cv2.imread(os.path.join(input_dir, filename),
cv2.IMREAD_GRAYSCALE)

    # Check if the image was loaded correctly
    if img is None:
        print(f"Error loading image: {filename}")
        continue

    # Denoise the sharpened image using Wiener filter
    denoised = cv2.fastNlMeansDenoising(img, None, h=10,
templateWindowSize=7, searchWindowSize=21)

    # Save the denoised images
```

```
    denoised_path = os.path.join(output_dir, f'denoised_{filename}')
    cv2.imwrite(denoised_path, denoised)

    # Display the images with numbering
    plt.subplot(1, 2, 1)
    plt.imshow(img, cmap='gray')
    plt.title(f'Original: {image_count}')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(denoised, cmap='gray')
    plt.title(f'Denoised: {image_count}')
    plt.axis('off')

    plt.show()

    # Increment the counter
    image_count += 1

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

# Define directories
input_dir = 'PATH TO INPUT FILE/DIRECTORY'
output_dir = 'PATH TO OUTPUT FILE/DIRECTORY'

# Create output directory
os.makedirs(output_dir, exist_ok=True)

# Initialize image counter
image_count = 1  # Update this with the last count from the previous
code if needed

# Process each image
for filename in sorted(os.listdir(input_dir)):
    if not filename.lower().endswith(('.png', '.jpg', '.jpeg', '.tif',
'.tiff')):  # Filter non-image files
        continue

    img = cv2.imread(os.path.join(input_dir, filename),
cv2.IMREAD_GRAYSCALE)

    # Check if the image was loaded correctly
    if img is None:
        print(f"Error loading image: {filename}")
        continue

    # Apply Otsu's thresholding
    _, otsu_thresh = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY +
```

```
cv2.THRESH_OTSU)

    # Save the Otsu thresholded images
    otsu_path = os.path.join(output_dir, f'otsu_{filename}')
    cv2.imwrite(otsu_path, otsu_thresh)

    # Display the images with numbering
    plt.subplot(1, 2, 1)
    plt.imshow(img, cmap='gray')
    plt.title(f'Denoised: {image_count}')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(otsu_thresh, cmap='gray')
    plt.title(f'Otsu Thresholding: {image_count}')
    plt.axis('off')

    plt.show()

    # Increment the counter
    image_count += 1

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

# Define directories
input_dir = 'PATH TO INPUT FILE/DIRECTORY'
output_dir = 'PATH TO OUTPUT FILE/DIRECTORY'

# Create output directory
os.makedirs(output_dir, exist_ok=True)

# Initialize image count
image_count = 1

# Process each image
for filename in sorted(os.listdir(input_dir)):
    if not filename.lower().endswith(('.png', '.jpg', '.jpeg', '.tif',
'.tiff')):  # Filter non-image files
        continue

    img = cv2.imread(os.path.join(input_dir, filename),
cv2.IMREAD_GRAYSCALE)

    # Check if the image was loaded correctly
    if img is None:
        print(f"Error loading image: {filename}")
        continue
```

```
    # Apply binary thresholding to emphasize the vessels
    _, thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

    # Define a small structuring element for morphological opening
    kernel = np.ones((2, 1), np.uint8)  # Small kernel for opening

    # Apply morphological opening to remove noise
    morph_opened = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)


    # Save the morphological images
    morph_opened_path = os.path.join(output_dir,
f'morphological_opening_{filename}')
    cv2.imwrite(morph_opened_path, morph_opened)

    # Display the images with numbering
    plt.subplot(1, 2, 1)
    plt.imshow(thresh, cmap='gray')
    plt.title(f'Thresholded: {image_count}')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(morph_opened, cmap='gray')
    plt.title(f'Morphological Opening: {image_count}')
    plt.axis('off')

    plt.show()

    # Increment the counter
    image_count += 1

from PIL import Image
import os

# Define directories
input_dir = 'PATH TO INPUT FILE/DIRECTORY'
output_dir = 'PATH TO OUTPUT FILE/DIRECTORY'

# Create the output directory
os.makedirs(output_dir, exist_ok=True)

# Process each GIF mask image
for filename in sorted(os.listdir(input_dir)):
    if not filename.lower().endswith('.gif'):  # Filter only GIF files
        continue

    # Construct the full file path
    gif_path = os.path.join(input_dir, filename)

    # Read the GIF image using PIL
    try:
```

```
        mask = Image.open(gif_path).convert('L')  # Convert to
grayscale (mode 'L')
    except Exception as e:
        print(f"Error loading mask: {filename}. {e}")
        continue

    # Save the mask as a TIFF file in the output directory
    tiff_filename = filename.replace('.gif', '.tif')
    tiff_path = os.path.join(output_dir, tiff_filename)
    mask.save(tiff_path, format='TIFF')

    print(f"Converted {filename} to {tiff_filename}")

print("Conversion completed.")
```

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

# Define directories
input_dir = 'PATH TO INPUT FILE/DIRECTORY'
mask_dir = 'PATH TO MASK FILE/DIRECTORY'
output_dir = 'PATH TO OUTPUT FILE/DIRECTORY'

# Create output directory
os.makedirs(output_dir, exist_ok=True)

# Process each image
for i in range(1, 41):
    if i <= 20:
        img_filename = 'FILE NAME'
        mask_filename = 'FILE NAME'
    else:
        img_filename = 'FILE NAME'
        mask_filename = 'FILE NAME'

    # Construct full file paths
    img_path = os.path.join(input_dir, img_filename)
    mask_path = os.path.join(mask_dir, mask_filename)

    # Read the image and the mask
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)

    # Check if the image and mask were loaded correctly
    if img is None:
        print(f"Error loading image: {img_filename}")
        continue
    if mask is None:
```

```
        print(f"Error loading mask: {mask_filename}")
        continue

    # Binarize the mask to ensure it is completely filled
    _, mask_binary = cv2.threshold(mask, 1, 255, cv2.THRESH_BINARY)

    # Optionally, you can apply morphological operations to the mask
to fill small holes
    kernel = np.ones((5, 5), np.uint8)
    mask_filled = cv2.morphologyEx(mask_binary, cv2.MORPH_CLOSE,
kernel)

    # Apply the mask to the image
    img_masked = cv2.bitwise_and(img, mask_filled)

    # Save the result
    output_path = os.path.join(output_dir,
f'circle_removed_{img_filename}')
    cv2.imwrite(output_path, img_masked)

    # Display the result
    plt.imshow(img_masked, cmap='gray')
    plt.title(f'Circle Removed: {img_filename}')
    plt.axis('off')
    plt.show()

print("Circle removal completed.")
```

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

# Define directories
input_dir = 'PATH TO INPUT FILE/DIRECTORY'
output_dir = 'PATH TO OUTPUT FILE/DIRECTORY'
os.makedirs(output_dir, exist_ok=True)

# Process each image
for filename in sorted(os.listdir(input_dir)):
    if not filename.lower().endswith(('.png', '.jpg', '.jpeg', '.tif',
'.tiff')):
        continue

    # Load image in grayscale
    img = cv2.imread(os.path.join(input_dir, filename),
cv2.IMREAD_GRAYSCALE)
    if img is None:
        print(f"Error loading image: {filename}")
        continue
```

```
    # Apply Non-Local Means Denoising to reduce noise while preserving
vessel structures
    denoised = cv2.fastNlMeansDenoising(img, None, h=30,
templateWindowSize=7, searchWindowSize=21)

    #  Detect circles using Hough Circle Transform
    circles = cv2.HoughCircles(denoised, cv2.HOUGH_GRADIENT, dp=1.2,
minDist=1000,
                               param1=50, param2=30, minRadius=100,
maxRadius=300)

    # Create a mask with a ring around the detected circle
    mask = np.ones_like(denoised, dtype=np.uint8) * 255  # Start with
a white mask

    if circles is not None:
        circles = np.round(circles[0, :]).astype("int")
        largest_circle = max(circles, key=lambda x: x[2])  # Find the
circle with the largest radius
        x, y, r = largest_circle

        # Draw a filled circle (black) slightly smaller than the
detected circle radius
        cv2.circle(mask, (x, y), r - 10, 255, thickness=cv2.FILLED)  #
Inner part of the ring is white
        # Draw a larger circle (black ring) to cover the outer
boundary
        cv2.circle(mask, (x, y), r, 0, thickness=10)  # Outer boundary
ring is black

    #  Apply the ring mask to remove only the outer boundary
    circle_removed = cv2.bitwise_and(denoised, denoised, mask=mask)

    #  Apply CLAHE to enhance vessel visibility while keeping
background black
    clahe = cv2.createCLAHE(clipLimit=2.5, tileGridSize=(8, 8))
    vessels_enhanced = clahe.apply(circle_removed)

    # Save the image with the circle removed and noise reduced, using
the specified naming convention
    base_name = os.path.splitext(filename)[0]
    processed_path = os.path.join(output_dir,
f'postprocessed_{base_name}.tif')
    cv2.imwrite(processed_path, vessels_enhanced)

    # Display the result
    plt.imshow(vessels_enhanced, cmap='gray')
    plt.title(f"Post-Processed Image: {base_name}")
```

```
    plt.axis('off')
    plt.show()

import cv2
import numpy as np
import os
from skimage.morphology import skeletonize
from skimage.measure import label, regionprops
import pandas as pd

# Define the directory containing the processed images
input_dir = 'PATH TO INPUT FILE/DIRECTORY'
# Lists to store calculated parameters
vessel_densities = []
vessel_tortuosities = []
vessel_lengths = []
vessel_widths = []

# Function to calculate vessel density
def calculate_vessel_density(binary_image):
    vessel_pixels = np.sum(binary_image == 255)
    total_pixels = binary_image.size
    density = vessel_pixels / total_pixels
    return density

# Function to calculate vessel tortuosity and length
def calculate_vessel_tortuosity_and_length(skeleton):
    labeled_skeleton = label(skeleton)
    tortuosity_sum = 0
    total_length = 0

    for region in regionprops(labeled_skeleton):
        coords = region.coords
        if len(coords) < 2:
            continue

        # Calculate tortuosity as path length / euclidean distance
        path_length = len(coords)
        start, end = coords[0], coords[-1]
        euclidean_distance = np.linalg.norm(start - end)
        if euclidean_distance > 0:
            tortuosity = path_length / euclidean_distance
        else:
            tortuosity = 1

        tortuosity_sum += tortuosity
        total_length += path_length

    average_tortuosity = tortuosity_sum /
len(regionprops(labeled_skeleton)) if
```

```
len(regionprops(labeled_skeleton)) > 0 else 0
    return average_tortuosity, total_length

# Function to calculate average vessel width
def calculate_vessel_width(binary_image, skeleton):
    distance_map = cv2.distanceTransform(binary_image, cv2.DIST_L2, 5)
    widths = distance_map[skeleton > 0] * 2  # Distance to edge,
multiplied by 2 for diameter
    average_width = np.mean(widths) if len(widths) > 0 else 0
    return average_width

# Process each image in the directory
for filename in sorted(os.listdir(input_dir)):
    if not filename.lower().endswith(('.png', '.jpg', '.jpeg', '.tif',
'.tiff')):
        continue

    # Loading image in grayscale
    img_path = os.path.join(input_dir, filename)
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        print(f"Could not load image {filename}")
        continue

    # Binarize the image
    _, binary_image = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

    # Skeletonize the binary image
    skeleton = skeletonize(binary_image // 255).astype(np.uint8)

    # Calculate vessel density
    vessel_density = calculate_vessel_density(binary_image)

    # Calculate vessel tortuosity and length
    vessel_tortuosity, vessel_length =
calculate_vessel_tortuosity_and_length(skeleton)

    # Calculate average vessel width
    vessel_width = calculate_vessel_width(binary_image, skeleton)

    # Store the calculated values
    vessel_densities.append(vessel_density)
    vessel_tortuosities.append(vessel_tortuosity)
    vessel_lengths.append(vessel_length)
    vessel_widths.append(vessel_width)

# Create a DataFrame with the results
df_results = pd.DataFrame({
    "Vessel Density": vessel_densities,
    "Average Tortuosity": vessel_tortuosities,
```

```
    "Total Vessel Length": vessel_lengths,
    "Average Vessel Width": vessel_widths
})

# Display the DataFrame in Jupyter Notebook
df_results
```