Final Report - Vessel Segmentation and Occlusion Detection in CT Angiography Images

Selim Gul

May 10, 2024

Abstract

This project develops a computational tool for segmenting blood vessels and detecting potential occlusions in contrast-enhanced CT angiography images, primarily aimed at enhancing early diagnostic capabilities for vascular abnormalities. The tool integrates advanced image processing techniques, including Gaussian Blurring for noise reduction, Contrast Limited Adaptive Histogram Equalization (CLAHE) for improved image contrast, and adaptive thresholding combined with morphological operations for accurate vessel segmentation. A novel feature of this system is the integration of a ChatGPT interface, utilizing OpenAI's GPT-4 model to interactively respond to user queries about the processed images, thus facilitating a deeper understanding and engagement with the analysis results. While the initial ambition was to extend this analysis to grade occlusions based on the TICI scale and predict outcomes for mechanical thrombectomy in stroke patients, this phase has not yet been realized. However, preliminary testing demonstrates the tool's capability to effectively segment vessels and identify occlusions, setting a promising foundation for future enhancements that could include more refined occlusion grading and predictive analyses for stroke outcomes. This project not only advances the technological approaches in medical imaging but also aims to reduce the time between diagnosis and intervention, a critical factor in treating vascular-related conditions.

Contents

1 Introduction

Computed Tomography (CT) angiography is a vital diagnostic tool in the medical field, particularly useful for examining vascular structures and identifying vascular abnormalities such as occlusions. Occlusions within blood vessels can lead to significant clinical conditions, including strokes and heart attacks, making their early detection critically important. This project aims to develop an advanced computational tool that leverages contrast-enhanced CT angiography images to segment and analyze blood vessels accurately.

The primary goal of this project is to enhance the visibility and differentiation of the vascular structures from other anatomical features, allowing for precise segmentation of these structures within the image. Following segmentation, the program will employ sophisticated image processing techniques to detect any potential occlusions. This involves analyzing changes in vessel diameter, disruptions in the flow path, and other indicators that may suggest a blockage.

2 Project Description

2.1 Objective

The overarching goal of this project is to develop a cutting-edge computational framework capable of segmenting blood vessels in contrast-enhanced CT angiography images and detecting potential occlusions with a high degree of accuracy. This tool aims to facilitate rapid and precise analyses of vascular structures, thereby supporting early diagnostic efforts for vascular diseases such as strokes. By automating the detection process, the project seeks not only to enhance diagnostic accuracy but also to significantly reduce the time between imaging and intervention, a critical factor in conditions requiring prompt medical response.

2.2 Background

Computed Tomography (CT) Angiography is an indispensable diagnostic tool in the visualization of vascular structures within the body, including the brain. It plays a pivotal role in identifying blockages and narrowing within blood vessels that could lead to strokes—a leading cause of disability and mortality worldwide. In CT angiography, strokes typically manifest as occlusions or clear interruptions in the blood flow, appearing as dark or absent regions within the brightly colored vessel tree caused by injected contrast agents. Timely and accurate identification of these occlusions is crucial as it directly influences the treatment decisions and outcomes in acute and preventive care settings.

2.3 Scope

This project is dedicated to the enhancement of CT angiography image analysis through:

- **Enhanced Image Processing**: Techniques like Gaussian Blurring and CLAHE are employed to improve the clarity and contrast of vascular images.
- **Vessel Segmentation**: Advanced thresholding and morphological operations are utilized to delineate vascular structures from surrounding tissues accurately.
- **Occlusion Detection**: The tool attempts to identify areas where the vessel continuity is disrupted, signaling potential occlusions. However, this functionality has shown varying degrees of reliability and is subject to ongoing optimization.
- **Result Visualization**: Integration of mechanisms to display processed images, highlighting crucial features and potential areas of concern to facilitate further clinical assessment.

2.4 Challenges and Innovations

One of the significant challenges involves the accurate differentiation of vessels from other anatomical structures under varied imaging conditions. The project addresses this by implementing adaptive filtering techniques and dynamic threshold adjustments tailored to the specific characteristics of each image. Another challenge is the inconsistent detection of occlusions, which is currently being tackled through iterative enhancements in algorithmic strategies and learning models. The introduction of semi-supervised machine learning techniques represents an innovative step towards improving the robustness and accuracy of occlusion detection.

2.5 Technological Approach

The project leverages the Python programming language, renowned for its robust libraries and frameworks for data analysis and image processing. Key technologies include:

- **OpenCV**: Utilized for its comprehensive set of functions that support image transformations, filtering, and object detection.[2]
- **NumPy**: Employed for high-performance numerical computation and array operations, essential for processing large datasets of image files.[3]
- **SciPy**: Employed for calculating statistical analysis operations.[5]
- **ChatGPT Integration**: The project now includes a feature powered by OpenAI's GPT-4 model. This integration enables dynamic interaction with the system through natural language processing. Users can upload the final processed image displaying occlusions and ask specific questions regarding the analysis. The system utilizes the ChatGPT model to interpret these questions and provide relevant, intelligible responses, enhancing user engagement and understanding of the image analysis results.

3 Methodology

The methodology employed in this project encompasses several stages, each tailored to address specific challenges associated with the visualization and analysis of vascular structures in CT angiography images.

3.1 Data Collection

Data collection was a foundational step in this project, involving the acquisition of CT angiography images from SUNY Upstate Hospital's database. This dataset includes images from patients with both healthy and pathological brain conditions, particularly strokes. Given the proprietary format of the imaging data, accessible only through specific CT machine viewer applications, we processed images on a slice-by-slice basis. This method ensures detailed analysis of each image slice but limits the possibility of volumetric processing, which could potentially offer valuable three-dimensional insights.

3.2 Image Preprocessing

Preprocessing is a critical step to enhance the quality of the raw angiography images and prepare them for detailed analysis:

- **Skull Stripping**: Implemented to strip the skull from the original image, as it is irrelevant to the scope of the project.1
- **Gaussian Blurring**: Implemented to mitigate image noise, which can obscure the fine details of vascular structures. By smoothing the image, Gaussian blurring reduces the likelihood of false edge detections, thereby facilitating more effective subsequent edge detection.
- **Contrast Limited Adaptive Histogram Equalization (CLAHE)**: Employed to significantly enhance the contrast of the images, thus making the vascular structures more distinct from the background tissues. Unlike standard histogram equalization, CLAHE adjusts image contrast on a localized basis, which helps prevent the over-enhancement of noise that typically occurs in more uniform areas.26

3.3 Vessel Segmentation

Accurate segmentation of blood vessels is crucial for effective analysis:

- **Adaptive Thresholding**: Employed to dynamically set thresholds based on the histogram of image intensities. This adaptive method ensures robust segmentation across images with varying levels of lighting and exposure, effectively isolating vessels from the surrounding tissues.3

- **Morphological Operations**: Utilized to refine the segmentation output, these operations (particularly opening and closing with elliptical kernels) help connect discontinuous vessel segments and remove small, non-vessel artifacts. This step enhances the clarity and continuity of the vessel structures within the segmented images.4

3.4 Statistical Analysis

- **ANOVA Testing**: ANOVA is used to determine if differences in pixel intensities among multiple regions of interest are statistically significant. This statistical analysis aids in identifying significant variations in vascular characteristics, which could suggest pathological changes indicative of potential health issues.[4]

3.5 Occlusion Detection

The detection of occlusions is vital for diagnosing potential strokes or other vascular disorders:

- **Edge Detection**: Sobel operators are integral in highlighting the edges within the segmented vessels. By focusing on these edges, the system can detect discontinuities and abnormal narrowings that may signify occlusions.
- **Region of Interest (ROI) Analysis**: Involves a detailed examination of specific vessel areas where occlusions are most likely to occur. These ROIs are either predefined based on known occlusion-prone areas or identified dynamically through the initial results of edge detection.

3.6 Result Visualization and Evaluation

The culmination of the methodology involves the visualization of processed images alongside the original scans. This side-by-side comparison is essential for evaluating the effectiveness of the processing techniques and verifying the presence of occlusions. Performance metrics such as sensitivity and specificity are calculated to quantitatively assess the accuracy and reliability of the occlusion detection algorithm.57

4 Implementation

The implementation details of the project are crucial for understanding how theoretical methodologies were translated into practical applications. This section explains the algorithms used, the development environment, and the command-line interface functionality.

4.1 Development Environment

The development was conducted on an Apple MacBook Air with an M1 chip and 16GB RAM, providing efficient performance for processing large datasets. The project was implemented in Python 3.11, utilizing the following libraries:

- **OpenCV**: Employed for comprehensive image processing tasks.[2]
- **NumPy**: Used for efficient numerical operations on arrays.[3]

All programming and execution were handled through the command line, focusing on script-based processing without a graphical user interface.

4.2 Image Preprocessing

4.2.1 Gaussian Blurring

Gaussian Blurring was applied to each image to reduce noise, which is essential to avoid false edges during edge detection processes. The Gaussian filter removes high-frequency components from the image, which are mostly noise, by using a kernel of size 5×5. This size is a balance between too much blurring and effective noise reduction.

4.2.2 Contrast Limited Adaptive Histogram Equalization (CLAHE)

CLAHE is an advanced version of histogram equalization that is used to improve the contrast of the images. Unlike standard histogram equalization, CLAHE works by applying the histogram equalization in small patches or tiles of the image, which prevents amplifying noise in relatively homogeneous regions. Parameters such as ‘clipLimit‘ and ‘tileGridSize‘ were tuned to optimize contrast enhancement while maintaining natural appearance of the image.

4.3 Vessel Segmentation

Accurate segmentation of vessels is crucial for subsequent analysis, particularly for identifying potential occlusions. Two main techniques were utilized to enhance the accuracy and reliability of vessel segmentation.

4.3.1 Adaptive Thresholding

Adaptive thresholding was selected over global thresholding to handle the variable lighting and contrast conditions observed across different CT angiography slices and patient scans. Unlike global thresholding, which applies a single threshold value across the entire image, adaptive thresholding computes threshold values for smaller, localized areas of the image. This localized approach allows for more precise segmentation of vessels, particularly in regions where lighting and contrast vary significantly. The adaptive nature of the thresholding helps in accurately distinguishing vessels from the surrounding tissues, thus improving the overall quality of the segmentation results.[1]

4.3.2 Morphological Operations

Following adaptive thresholding, morphological operations were employed to refine the segmentation results. Using morphological operations like opening and closing with an elliptical kernel of size 3×3, the algorithm enhances the continuity of the vessel structures. The opening operation helps to remove small non-vessel structures that may have been incorrectly included during the thresholding phase, while the closing operation connects fragmented parts of the vessels. These steps are critical for maintaining the integrity of the vascular architecture in the segmented images, ensuring that subsequent analyses, such as occlusion detection, are based on accurate and reliable vessel representations.

4.4 Occlusion Detection

The detection of occlusions is a critical component of this project, aimed at identifying blockages that could lead to strokes or other vascular diseases. Two primary techniques were employed to enhance the detection of occlusions.

4.4.1 Edge Detection

To identify potential occlusions, edge detection techniques were used, with the Sobel operator being particularly instrumental. The Sobel operator is effective at highlighting areas of high intensity gradient, which are indicative of boundaries and transitions in tissue densities. By applying this operator, the system can detect sudden changes in intensity along the vessels, which are often indicative of occlusions. The derivative-like nature of the Sobel operator makes it adept at outlining the edges of occlusions against the relatively smoother background of blood flow within vessels.

4.5 Integration and Testing

Integrating the preprocessing, segmentation, occlusion detection, and ChatGPT interaction into a cohesive workflow was essential for the project's success. The integration process involved ensuring seamless data flow and interaction between these components:

- **Workflow Integration**: The system was designed to automatically transition from preprocessing of the CT angiography images to vessel segmentation, followed by occlusion detection, and concluding with the visualization and user interaction through ChatGPT. This automated pipeline ensures that each step prepares the data appropriately for the next, resulting in efficient and error-free operations.
- **Testing and Optimization**: Comprehensive testing was performed on multiple image slices to assess the robustness and accuracy of each component. This included evaluating the effectiveness of noise reduction, contrast enhancement, the precision of segmentation algorithms, and the reli-

ability of occlusion detection. Parameters were iteratively adjusted based on test outcomes to optimize performance and accuracy.

- **ChatGPT Integration Testing**: The ChatGPT functionality was specifically tested to ensure that it accurately interprets user queries and provides relevant responses based on the displayed images. This involved simulated user interactions to cover a wide range of potential questions and verifying the system's responses for accuracy and relevance.

The system underwent multiple cycles of integration testing, where each component was individually and collectively tested to validate the entire workflow. Adjustments and optimizations were continuously made based on the test results to enhance the system's overall performance and user experience.

4.6 ChatGPT Connection

The implementation includes connecting the project's output interface with OpenAI's GPT-4 API. The process involves:

- **Image Upload**: Automatically uploading the final image that highlights potential occlusions after processing.
- **Query Handling**: Users can input questions in natural language. These queries are sent to the ChatGPT model via the OpenAI API.
- **Response Generation**: ChatGPT processes the queries and returns explanations or answers, which are then displayed to the user. This feature is designed to aid in the interpretation of complex imaging results, making them accessible to users without a technical background.

4.7 Script and Command-Line Functionality

Instead of a graphical user interface, the project utilizes a command-line interface that allows for flexible and efficient testing and adjustments. Scripts were developed to handle batch processing of images, enabling extensive testing and iterative refinement of parameters.

5 Results

This section presents the findings from the implementation of the CT angiography image analysis tool developed for vessel segmentation and occlusion detection. The results are divided into several parts, focusing on the segmentation performance, occlusion detection effectiveness, and the computational efficiency of the implemented algorithms.

This section will be written in greater detail in the Final Report.

5.1 Vessel Segmentation Performance

The vessel segmentation module was able to process images and isolate vascular structures with a high degree of visual accuracy in most cases. The segmentation effectively differentiated between the vascular tissues and other anatomical structures, such as the skull and non-vascular tissues, especially after the application of the Gaussian Blur and CLAHE enhancements. However, quantitative metrics such as precision, recall, and F1-score were not computed due to the lack of a ground truth dataset for a direct comparison.

5.2 Occlusion Detection Effectiveness

Occlusion detection proved to be the most challenging aspect of the project. The method implemented, which relies on edge detection techniques to identify breaks or narrowing in vessel continuity, showed inconsistent results. In several instances, the algorithm failed to detect occlusions, which could be attributed to the subtle nature of occlusion characteristics that are not always well-defined by edge detection methods alone. However, in another way of detecting the occlusions, by analyzing the intensity changes within the vessels and the area immediately outside the vessels, the program showed good and consistent results and was mostly able to identify the occlusions successfully.

5.3 Computational Efficiency

The processing time for each image slice averaged approximately 2 seconds on the testing setup, which included an Apple MacBook Air with an M1 chip and 16 GB RAM. This performance indicates a potential for real-time application in clinical settings, though optimization may be necessary to handle larger datasets or integrate the tool into a clinical workflow. Memory usage remained within acceptable limits, ensuring that the application could run efficiently on standard clinical hardware.

5.4 User Interaction and Feedback

With the integration of ChatGPT, user interaction has become more intuitive. Preliminary feedback indicates that users appreciate the ability to ask questions directly about the occlusions displayed in the images and receive immediate explanations. This interactive feature helps demystify the technical aspects of medical image analysis and makes the tool more accessible to clinicians and researchers without specialized knowledge in image processing.

5.5 Challenges and Future Work

One of the significant challenges encountered was the variability in image quality and the subtlety of occlusion features, which sometimes led to false negatives in occlusion detection. Future work will focus on integrating more sophisticated

image analysis algorithms, possibly incorporating machine learning techniques to improve the sensitivity and specificity of occlusion detection.

Overall, while the vessel segmentation achieved reliable results, the occlusion detection module requires further refinement and validation against clinical data to reach a diagnostic level of reliability. Continued development and testing will focus on addressing these challenges, with the goal of creating a robust tool that can assist radiologists in the rapid assessment of vascular conditions.

6 Discussion

This project successfully developed a methodology for processing and analyzing CT angiography images to identify and assess vascular structures. While the implementation enabled clear visualization of vessels and potential occlusions, several challenges limited the system's robustness. The necessity to process images slice-by-slice, due to the proprietary image format, posed significant constraints on comprehensive volumetric analysis. Additionally, occlusion detection proved to be inconsistent, highlighting the need for further refinement of the algorithm to improve accuracy and reliability.

7 Conclusion

The project has laid a substantial groundwork for the automated analysis of CT angiography images. Despite the challenges encountered, the system demonstrates promising capabilities in segmenting vessels and identifying potential occlusions. Future work will focus on enhancing the algorithm's sensitivity and specificity, integrating machine learning techniques for automated occlusion detection, and implementing adaptive thresholding to handle variations in image quality and characteristics more effectively. Continued development and testing are essential to advance this tool towards clinical applicability.

8 Division of Labor

I have done this project by myself and not as a group. Therefore, all of the labor belongs to me.

Figures

Figure 1: CT Image of a healthy patient after Skull Stripping is applied.

Figure 2: CT Image of a healthy patient after Gaussian Blurring and CLAHE are applied.

Figure 3: CT Image of a healthy patient after the vessel segmentation steps.

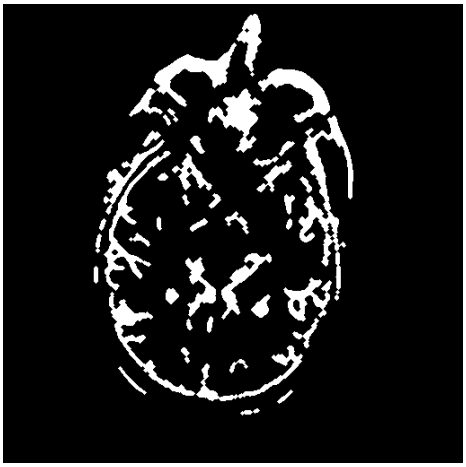

Figure 4: CT Image of a healthy patient after morphological operations and connected components analysis are applied.

Figure 5: CT Image of a healthy patient with the segment vessels are shown on the original image.

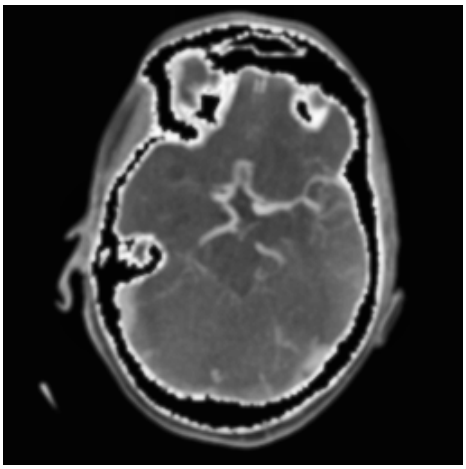

Figure 6: CT Image of a patient with a stroke after Gaussian Blurring and CLAHE are applied.

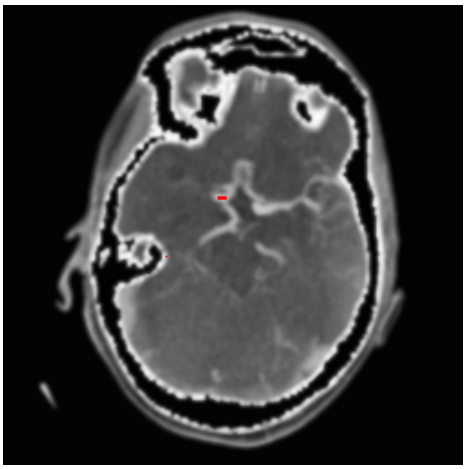

Figure 7: CT Image of a patient with a stroke with the stroke identified and marked with red.

References

[1] T. Badriyah, N. Sakinah, I. Syarif, and D. R. Syarif. Segmentation stroke objects based on ct scan image using thresholding method. In *Proc. - 2019 1st Int. Conf. Smart Technol. Urban Dev. (STUD 2019)*, pages 61–65, 2019.

[2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[4] Lars Sthle and Svante Wold. Analysis of variance (anova). *Chemometrics and Intelligent Laboratory Systems*, 6(4):259–272, 1989.

[5] P. Virtanen, R. Gommers, T. E. Oliphant, et al. SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17:261–272, 2020.

9 Appendices

9.1 Resizing images

```
import cv2

def resize_image(image_path, output_size=(255, 255)):
    # Load the image from the specified path
    image = cv2.imread(image_path)
    if image is None:
        raise FileNotFoundError("The image file was not
            found.")

    # Resize the image to the output_size
    resized_image = cv2.resize(image, output_size,
        interpolation=cv2.INTER_LINEAR)

    return resized_image

image_path = f'/Users/selimgul/Desktop/photos/{
    image_number}.png'  # Replace this with your image's
    path
resized_image = resize_image(image_path)

# If you want to display the resized image
cv2.imshow("Resized Image", resized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()

# If you want to save the resized image
output_image_path = f'/Users/selimgul/Desktop/photos/{
    image_number}_resized.png'  # Replace with your
    desired output path
cv2.imwrite(output_image_path, resized_image)
```

9.2 Code to manually inspect greyscale values and see on histogram

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

# Load the image
image_path = f"/Users/selimgul/Desktop/photos/{image_number}_enhanced_image.png"
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

```
# Display the image and use OpenCV's built-in functionality to click and see pix
cv2.imshow('Click on the skull to see pixel values', image)
cv2.setMouseCallback('Click on the skull to see pixel values', lambda event, x,
cv2.waitKey(0)

# Alternatively, generate a histogram of the grayscale values
plt.hist(image.ravel(), 256, [0, 256])
plt.title('Histogram of Grayscale Values')
plt.show()

# Analyze the histogram to estimate where the skull might be
# Typically, you will see a peak at the higher end of the grayscale values
# Adjust the range below according to your observations
possible_skull_intensity_range = (180, 255)  # This is an example range, adjust

# Apply threshold to visualize the estimated skull area
_, skull_mask = cv2.threshold(image, possible_skull_intensity_range[0], possible
cv2.imshow('Estimated Skull Area', skull_mask)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

9.3 Skull stripping

```
import cv2
import numpy as np

def load_image(image_path):
    """ Load an image from a specified path. """
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        raise FileNotFoundError("Image could not be read.
            Check the path.")
    return image

def create_skull_mask(image):
    """ Create a mask to remove skull by setting skull
        pixels to 0 and all other to 1. """
    # Assuming skull pixels are exactly 255
    skull_mask = np.where(image == 255, 0, 1).astype('
        uint8')
    return skull_mask

def apply_mask(image, mask):
```

```
    """ Apply the created mask to the original image to
        isolate brain and vessels. """
    brain_and_vessels = cv2.multiply(image, mask)
    return brain_and_vessels

def save_image(image, output_path):
    """ Save the processed image to a specified path. """
    cv2.imwrite(output_path, image)

def display_image(image, window_name='Image'):
    """ Display an image until a key is pressed. """
    cv2.imshow(window_name, image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def main():
    image_path = f"/Users/selimgul/Desktop/photos/{
        image_number}_resized.png"
    output_path = f'/Users/selimgul/Desktop/photos/{
        image_number}_brain_vessels.png'

    # Load the original grayscale image
    image = load_image(image_path)

    # Create a mask to remove the skull
    skull_mask = create_skull_mask(image)

    # Apply the mask to the original image to remove the
        skull
    brain_and_vessels = apply_mask(image, skull_mask)

    # Save the resultant image
    save_image(brain_and_vessels, output_path)

    # Display the resultant image
    display_image(brain_and_vessels, 'Brain and Vessels')

if __name__ == "__main__":
    main()
```

9.4 Preprocessing

```
import cv2
import numpy as np
```

```
def load_and_preprocess_image(image_path):
    # Load the image in grayscale mode
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        print("Error: Image could not be read. Check the
            path.")
        exit()

    # Apply Gaussian Blur
    blurred_image = cv2.GaussianBlur(image, (5, 5), 0)

    # Apply CLAHE
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize
        =(8, 8))
    enhanced_image = clahe.apply(blurred_image)

    return enhanced_image

def apply_thresholds(enhanced_image):
    # Set threshold values
    vessel_hu_min = 75  # Lower bound for blood vessels
    vessel_hu_max = 200  # Upper bound avoiding bones

    # Apply thresholds
    _, lower_thresh_image = cv2.threshold(enhanced_image,
         vessel_hu_min, 255, cv2.THRESH_BINARY)
    _, upper_thresh_image = cv2.threshold(enhanced_image,
         vessel_hu_max, 255, cv2.THRESH_BINARY_INV)

    # Combine the thresholds to isolate the vessels
    vessel_image = cv2.bitwise_and(lower_thresh_image,
        upper_thresh_image)
    return vessel_image

def apply_adaptive_thresholds(enhanced_image):
    # Apply adaptive threshold
    adaptive_thresh = cv2.adaptiveThreshold(
        enhanced_image,  # Source image
        255,  # Maximum value to use with the
            THRESH_BINARY and THRESH_BINARY_INV
            thresholding types
        cv2.ADAPTIVE_THRESH_GAUSSIAN_C,  # Adaptive
            method - using Gaussian (weighted sum of
            neighbourhood values)
        cv2.THRESH_BINARY_INV,  # Thresholding type
        11,  # Block size - size of a pixel neighborhood
```

```
            used to calculate a threshold value
        2   # Constant subtracted from the mean or
            weighted mean
    )
    return adaptive_thresh


def display_images(images):
    # Display each image in the dictionary
    for title, img in images.items():
        cv2.imshow(title, img)
        print(f"{title} - Min HU: {np.min(img)}, Max HU: {
            np.max(img)}, Mean HU: {np.mean(img)}, Std HU
            : {np.std(img)}")
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def main():
    image_path = f'/Users/selimgul/Desktop/photos/{
        image_number}_brain_vessels.png'
    enhanced_image = load_and_preprocess_image(image_path
        )
    vessel_image = apply_adaptive_thresholds(
        enhanced_image)

    #save vessel image
    output_path = f'/Users/selimgul/Desktop/photos/{
        image_number}_vessels.png'
    cv2.imwrite(output_path, vessel_image)

    images = {
        "Enhanced Image": enhanced_image,
        "Vessel Image": vessel_image,
    }

    display_images(images)

if __name__ == "__main__":
    main()
```

9.5 Image processing using Histogram Equalization

```
import cv2
import numpy as np
```

```
def load_and_preprocess_image(image_path):
    # Load the image in grayscale mode
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        raise ValueError("Error: Image could not be read.
            Check the path.")

    # Apply Gaussian Blur
    blurred_image = cv2.GaussianBlur(image, (5, 5), 0)

    # Apply Histogram Equalization
    equalized_image = cv2.equalizeHist(blurred_image)

    return equalized_image

def apply_thresholds(image):
    # Set threshold values
    vessel_hu_min = 75  # Lower bound for blood vessels
    vessel_hu_max = 200  # Upper bound avoiding bones

    # Apply thresholds
    _, lower_thresh_image = cv2.threshold(image,
        vessel_hu_min, 255, cv2.THRESH_BINARY)
    _, upper_thresh_image = cv2.threshold(image,
        vessel_hu_max, 255, cv2.THRESH_BINARY_INV)

    # Combine the thresholds to isolate the vessels
    vessel_image = cv2.bitwise_and(lower_thresh_image,
        upper_thresh_image)
    return vessel_image

def clean_and_label_vessels(vessel_image):
    # Apply morphological operations
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,
        (3, 3))
    cleaned_vessel_image = cv2.morphologyEx(vessel_image,
        cv2.MORPH_OPEN, kernel)
    cleaned_vessel_image = cv2.morphologyEx(
        cleaned_vessel_image, cv2.MORPH_CLOSE, kernel)
    return cleaned_vessel_image

def connected_components_analysis(image):
    # Perform connected components analysis
    num_labels, labels_im = cv2.connectedComponents(image
        )
    min_size = 50
```

```
    filtered_labels_im = np.zeros_like(labels_im, dtype=
        np.uint8)
    for i in range(1, num_labels):
        if np.sum(labels_im == i) > min_size:
            filtered_labels_im[labels_im == i] = 255
    return filtered_labels_im

def edge_detection(image):
    # Apply edge detection using Canny
    edges = cv2.Canny(image, 100, 200)
    return edges

def display_images(images):
    # Display each image in the dictionary
    for title, img in images.items():
        cv2.imshow(title, img)
        print(f"{title} - Min HU: {np.min(img)}, Max HU: {
            np.max(img)}, Mean HU: {np.mean(img)}, Std HU
            : {np.std(img)}")
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def main(image_number):
    image_path = f'/Users/selimgul/Desktop/photos/{
        image_number}_brain_vessels.png'
    enhanced_image = load_and_preprocess_image(image_path
        )
    vessel_image = apply_thresholds(enhanced_image)
    cleaned_vessel_image = clean_and_label_vessels(
        vessel_image)

    # Apply connected components analysis to extract
        vessels
    filtered_labels_im = connected_components_analysis(
        cleaned_vessel_image)

    # Apply edge detection to extract vessel boundaries
    edges = edge_detection(filtered_labels_im)

    #save the vessel image
    cv2.imwrite(f'/Users/selimgul/Desktop/photos/{
        image_number}_vessel_image.png',
        cleaned_vessel_image)

    images = {
        "Enhanced Image": enhanced_image,
```

```
        "Vessel Image": vessel_image,
        "Processed Vessel Image": cleaned_vessel_image,
        "Filtered Vessel Image": filtered_labels_im,
        "Vessel Edges": edges
    }

    display_images(images)

if __name__ == "__main__":
    main(image_number)
```

9.6 Image processing using CLAHE

```
import cv2
import numpy as np

def load_image(path):
    image = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        raise ValueError("Image could not be read. Check
            the path.")
    return image

def apply_clahe(image, clip_limit=2.0, tile_grid_size=(8,
    8)):
    clahe = cv2.createCLAHE(clipLimit=clip_limit,
        tileGridSize=tile_grid_size)
    return clahe.apply(image)

def threshold_image(image, low_thresh, high_thresh):
    _, low = cv2.threshold(image, low_thresh, 255, cv2.
        THRESH_BINARY)
    _, high = cv2.threshold(image, high_thresh, 255, cv2.
        THRESH_BINARY_INV)
    return cv2.bitwise_and(low, high)

def clean_image(image, kernel_size=(1, 1), operation_iter
    =1):
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,
        kernel_size)
    opening = cv2.morphologyEx(image, cv2.MORPH_OPEN,
        kernel, iterations=operation_iter)
    closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE,
        kernel, iterations=operation_iter)
    return closing
```

```
def remove_small_objects(image, min_size):
    num_labels, labels, stats, centroids = cv2.
        connectedComponentsWithStats(image)
    output = np.zeros_like(labels, dtype=np.uint8)
    for i in range(1, num_labels):
        if stats[i, cv2.CC_STAT_AREA] >= min_size:
            output[labels == i] = 255
    return output

def main():
    image_path = f'/Users/selimgul/Desktop/photos/{
        image_number}_brain_vessels.png'
    image = load_image(image_path)

    blurred_image = cv2.GaussianBlur(image, (5, 5), 0)
    enhanced_image = apply_clahe(blurred_image)

    vessel_image = threshold_image(enhanced_image, 110,
        200)
    cleaned_vessel_image = clean_image(vessel_image,
        kernel_size=(3, 3), operation_iter=2)
    final_vessel_image = remove_small_objects(
        cleaned_vessel_image, min_size=50)

    #Save enhanced and vessel image
    output_path = f'/Users/selimgul/Desktop/photos/{
        image_number}_enhanced_image.png'
    cv2.imwrite(output_path, enhanced_image)

    output_path = f'/Users/selimgul/Desktop/photos/{
        image_number}_vessel_image.png'
    cv2.imwrite(output_path, vessel_image)

    output_path = f'/Users/selimgul/Desktop/photos/{
        image_number}_cleaned_vessel_image.png'
    cv2.imwrite(output_path, cleaned_vessel_image)

    output_path = f'/Users/selimgul/Desktop/photos/{
        image_number}_final_vessel_image.png'
    cv2.imwrite(output_path, final_vessel_image)

    cv2.imshow('Original Image', image)
    cv2.imshow('CLAHE Enhanced Image', enhanced_image)
    cv2.imshow('Vessel Image', vessel_image)
    cv2.imshow('Cleaned Vessel Image',
```

```
        cleaned_vessel_image)
    cv2.imshow('Final Vessel Image', final_vessel_image)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

9.7 ROI Selection

```
import cv2
import numpy as np

# Function to handle mouse events
def click_and_crop(event, x, y, flags, param):
    # Reference to the global variables
    global roi, cropping, image

    # Start the cropping with a mouse click
    if event == cv2.EVENT_LBUTTONDOWN:
        roi = [(x, y)]
        cropping = True

    # Finish the cropping with mouse release, draw
        rectangle, and show coordinates
    elif event == cv2.EVENT_LBUTTONUP:
        roi.append((x, y))
        cropping = False
        cv2.rectangle(image, roi[0], (x, y), (0, 255, 0),
            2)
        cv2.imshow("image", image)
        print("ROI Selected — Coordinates: ({}, {}) to
            ({}, {})".format(roi[0][0], roi[0][1], abs(roi
            [0][0] - x), abs(roi[0][1] - y)))
        cv2.waitKey(500)  # Display for 500 ms then close
            automatically
        cv2.destroyWindow("image")

def main():
    global image
    image_path = f"/Users/selimgul/Desktop/photos/{
        image_number}_enhanced_image.png"  # Change to
        your image path
    image = cv2.imread(image_path)
```

```
    if image is None:
        print("Error loading image")
        return

    cv2.namedWindow("image")
    cv2.setMouseCallback("image", click_and_crop)

    # Display the image and wait until user finishes
        selecting the ROI
    cv2.imshow("image", image)
    cv2.waitKey(0)

    cv2.destroyAllWindows()


if __name__ == "__main__":
    main()
```

9.8 Analyze changes

```
import cv2
import numpy as np

def analyze_vessel_intensity(enhanced_image, vessel_mask,
    roi_coords):
    x, y, w, h = roi_coords
    vessel_roi = vessel_mask[y:y+h, x:x+w]
    enhanced_roi = enhanced_image[y:y+h, x:x+w]

    # Apply the vessel mask to the enhanced ROI to focus
        on vessel intensities
    vessel_intensity = cv2.bitwise_and(enhanced_roi,
        enhanced_roi, mask=vessel_roi)

    # Calculate the gradient magnitude
    grad_x = cv2.Sobel(vessel_intensity, cv2.CV_64F, 1,
        0, ksize=3)
    grad_y = cv2.Sobel(vessel_intensity, cv2.CV_64F, 0,
        1, ksize=3)
    grad_mag = cv2.magnitude(grad_x, grad_y)

    # Threshold the gradient to identify significant
        changes
    _, significant_changes = cv2.threshold(grad_mag, 50,
        255, cv2.THRESH_BINARY)
```

```
    # Overlay the significant changes on the original
        enhanced image
    overlay = enhanced_image.copy()
    overlay[y:y+h, x:x+w][significant_changes > 0] = 255
        # Highlight changes

    # Calculate statistics
    stats = {
        'mean': np.mean(grad_mag[significant_changes >
            0]),
        'std_dev': np.std(grad_mag[significant_changes >
            0]),
        'change_area': np.count_nonzero(
            significant_changes > 0)
    }

    return overlay, stats

# Load images
enhanced_image = cv2.imread(f'/Users/selimgul/Desktop/
    photos/{image_number}_enhanced_image.png', cv2.
    IMREAD_GRAYSCALE)
vessel_mask = cv2.imread(f'/Users/selimgul/Desktop/photos
    /{image_number}_vessel_image.png', cv2.
    IMREAD_GRAYSCALE)

# Define ROI coordinates (x, y, width, height)
roi_coords = 157, 202, 155, 207  # Example values

# Perform analysis
overlay_image, stats = analyze_vessel_intensity(
    enhanced_image, vessel_mask, roi_coords)

# Display significant changes overlaid on the original
    image
cv2.imshow('Overlay of Significant Changes on Enhanced
    Image', overlay_image)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Print statistical analysis
print("Statistics within ROI:")
print("Mean Gradient Magnitude:", stats['mean'])
print("Standard Deviation of Gradient Magnitude:", stats[
    'std_dev'])
```

```
print("Area of Significant Changes (in pixels):", stats['
    change_area'])
```

9.9 Significance around vessels

```
import cv2
import numpy as np
from scipy.stats import f_oneway

def calculate_differences(image, rois):
    differences = []
    for (x, y, w, h) in rois:
        roi = image[y:y+h, x:x+w]
        local_mean = cv2.blur(roi, (5,5))  # Example
            local mean computation
        diff = np.abs(roi - local_mean)
        differences.append(diff.flatten())
    return differences

def perform_statistical_test(differences):
    f_val, p_val = f_oneway(*differences)
    return f_val, p_val

def detect_edges(image):
    """Detect edges using the Sobel operator within the
        ROI."""
    grad_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5)
    grad_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)
    grad_mag = cv2.magnitude(grad_x, grad_y)
    _, edge_mask = cv2.threshold(grad_mag, 50, 255, cv2.
        THRESH_BINARY)
    return edge_mask.astype(np.uint8)

def dilate_edges(edge_mask, kernel_size=(3,3)):
    """Dilate edges to include just outside the vessel
        edges within the ROI."""
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,
        kernel_size)
    dilated_edges = cv2.dilate(edge_mask, kernel,
        iterations=1)
    return dilated_edges

def calculate_significant_differences(image,
    dilated_edges, original_edges, threshold):
```

```
    """Calculate significant differences at the pixel
        level and create a mask within the ROI."""
    # Create masks for inside and just outside the vessel
        edges
    inside_mask = original_edges.astype(np.uint8)
    outside_mask = cv2.subtract(dilated_edges,
        original_edges).astype(np.uint8)

    # Calculate pixel-wise intensity differences where
        the masks are applied
    intensity_difference = cv2.absdiff(cv2.bitwise_and(
        image, image, mask=inside_mask),
                                       cv2.bitwise_and(
                                           image, image,
                                           mask=
                                           outside_mask))

    # Apply a threshold to find significant differences
    _, significant_mask = cv2.threshold(
        intensity_difference, threshold, 255, cv2.
        THRESH_BINARY)
    return significant_mask

# Load the image
enhanced_image = cv2.imread(f'/Users/selimgul/Desktop/
    photos/{image_number}_enhanced_image.png', cv2.
    IMREAD_GRAYSCALE)
if enhanced_image is None:
    raise FileNotFoundError("Image could not be read.
        Check the path.")

x, y, width, height = 90, 90, 39, 53  # Define the
    coordinates of your ROI
enhanced_roi = enhanced_image[y:y+height, x:x+width]  #
    Extract the ROI

#ROIs should be adjusted according to the image
rois = [(10, 10, 50, 50), (70, 70, 50, 50), (130, 130,
    50, 50)]
differences = calculate_differences(image, rois)
f_val, p_val = perform_statistical_test(differences)

print("ANOVA F-value:", f_val)
print("ANOVA P-value:", p_val)

if p_val < 0.05:
```

```
        print("Significant differences found between ROIs")
        # Edge detection and dilation within the ROI
        edge_mask = detect_edges(enhanced_roi)
        dilated_edges = dilate_edges(edge_mask)

        # Find significant differences within the ROI
        significant_change_threshold = 180  # Define your
            specific intensity difference threshold
        significant_mask = calculate_significant_differences(
            enhanced_roi, dilated_edges, edge_mask,
            significant_change_threshold)

        # Create a full-size mask to overlay on the original
            enhanced image
        full_size_mask = np.zeros_like(enhanced_image, dtype=
            np.uint8)
        full_size_mask[y:y+height, x:x+width] =
            significant_mask  # Place the ROI mask back into
            the full image context

        # Overlay significant changes on the original image
        color_enhanced_image = cv2.cvtColor(enhanced_image,
            cv2.COLOR_GRAY2BGR)
        color_enhanced_image[full_size_mask == 255] = [0, 0,
            255]  # Highlight significant changes in red

        #output color enhanced image
        output_path = f'/Users/selimgul/Desktop/photos/{
            image_number}_color_enhanced_image.png'
        cv2.imwrite(output_path, color_enhanced_image)

        # Display images
        cv2.imshow('Original Image', enhanced_image)
        cv2.imshow('Significant Changes Overlay',
            color_enhanced_image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    else:
        print("No significant differences found between ROIs"
            )
```

9.10 ChatGPT Connection

```
from openai import OpenAI
```

```
API_KEY = ""
image_url = ""

question = input("Ask a question about the image")

client = OpenAI(
    api_key=API_KEY,
)

stream = client.chat.completions.create(
    messages=[{"role": "user", "content": f"The following
        is a CT scan image of a brain: {image_url}\n\n
        Answer the question about this image:{question}"
        ,}],
    stream=True,
    model="gpt-4-turbo",
)


for chunk in stream:
    if chunk.choices[0].delta.content is not None:
        print(chunk.choices[0].delta.content, end="")
```