Digital Image Processing
Maria Eleni Chochlidaki
Michail Samaritakis

Project Purpose

The primary objective of this project is to develop a method for detecting the quantity of stitches from images of surgical operations. This task involves using computer vision and digital image processing techniques to accurately identify and count the number of stitches present in each image.

Methodology

Two attempts were made to achieve the project's purpose, documented in files `test1.py` and `test2.py`. The methodology for each attempt involved a series of steps to process the images, evaluate their quality, detect stitches, and visualize the results.

- Analysis of `test1.py`

Code Content and Functionality:

1. **Image Processing:**
   process_image()
   - **Purpose:** Convert the image to grayscale, apply Gaussian blur, perform Otsu's thresholding, and execute morphological operations to prepare the image for contour detection.
   - **Reason:**
     i. Grayscale conversion simplifies the image by removing color information,
     ii. Gaussian blur reduces noise,
     iii. Otsu's thresholding binarize the image effectively, and
     iv. Morphological operations help remove small noise and connect disjointed regions to enhance contour detection.
2. **Image Quality Evaluation:**
   evaluate_image_quality()
   - **Purpose:** Assess the quality of the image by detecting lines using the Hough Line Transform.
   - **Reason:** Ensures that only good quality images (those with detectable features) are processed further. If no lines are detected, the image is deemed to be of poor quality and is flagged accordingly.
3. **Visualization:**
   visualize_processing_bad_data()
   visualize_processing()
   - Create and save visualizations of each step in the image processing pipeline. Provides a visual representation of the processed images at different stages

(grayscale, thresholded, morphological operations, and contours). This helps in understanding the effectiveness of the processing steps and the accuracy of stitch detection.

4. **Main Function:**
    - Orchestrates the processing of images, evaluates their quality, processes them to detect stitches, visualizes the processing steps, and records the results. Manages the entire workflow from reading images, processing them, visualizing results, to saving the final output. This ensures a systematic and automated approach to handle multiple images and record their analysis in a structured format.

Output Analysis:

CSV File (`results_test1.csv`):

- Records the number of stitches detected for each image. Images of poor quality are marked with a `-1`.

Output Directory (`output_images_from_test1`):

- Saves visualizations of each processed image, including grayscale, thresholded, morphologically processed images, and images with detected contours.

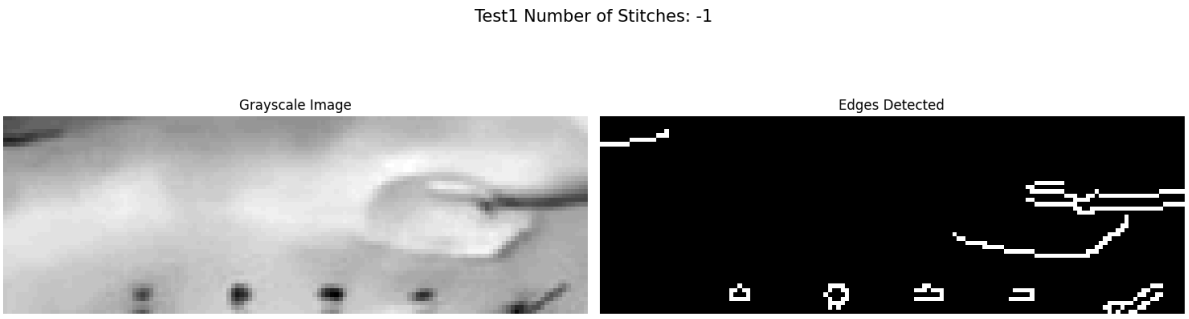
Bad Data Directory (`output_images_with_bad_data_from_test1`):

- Saves visualizations for images deemed to be of poor quality, showing the grayscale and edges detected.

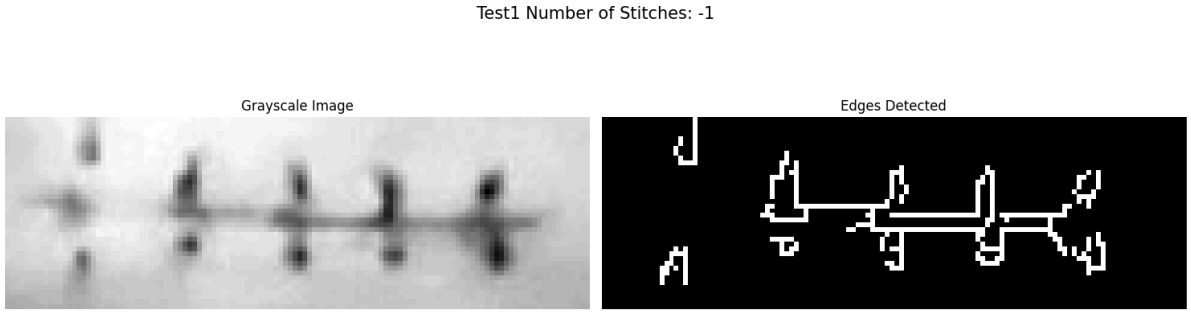
Examples of what is working and what is not working:

Bad Data: From 134 images 5 of them were detected with bad data and only 1 out of those 5 was incorrectly marked as bad data. However, there were more images with bad data that could not be detected. Success rate: 16/134 images were bad data, so the success rate of our script is: 25% (excluding the incorrectly flagged image).

Correct result:

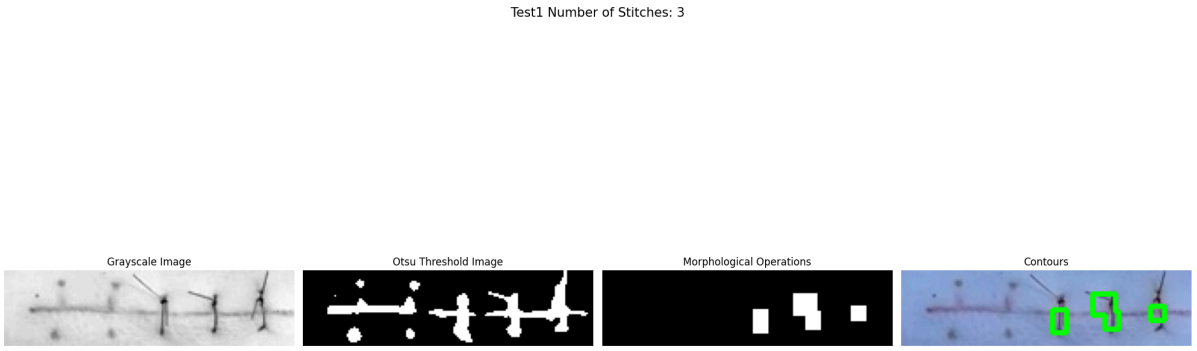


Incorrect result:

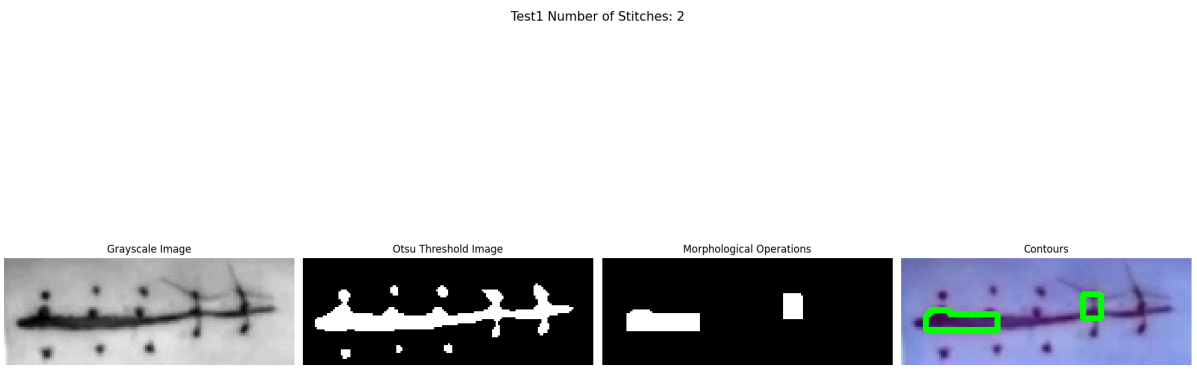


Data: From 129 images 23 were correct which corresponds to 17.8% success rate.

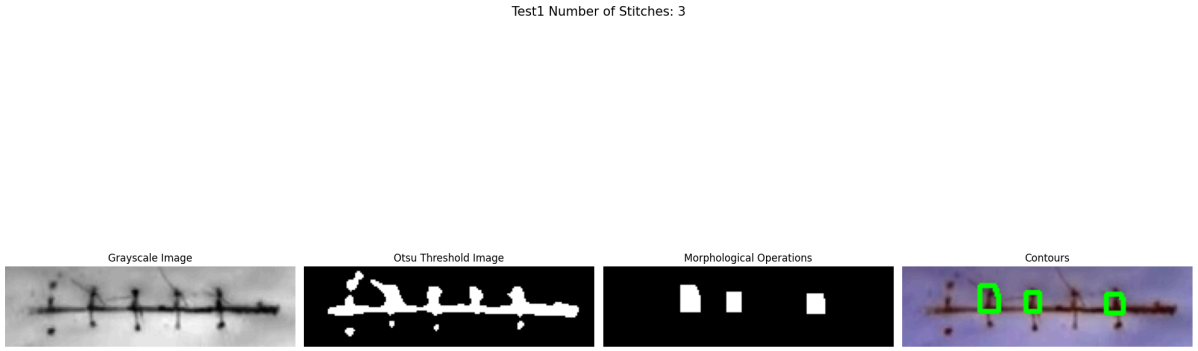
Correct result:



However, sometimes the stitches that were detected based on the intermediate results were wrong but the result was correct. For example:



Incorrect result:



- Analysis of `test2.py`

Code Content and Functionality:

1. **Image Processing:**

   process_image()
   process_image_with_morphological_operations()

   - **Purpose:** Convert the image to grayscale, apply Gaussian blur, detect edges, and perform morphological operations.
   - **Reason:**
     i. Grayscale conversion simplifies the image for further processing.
     ii. Gaussian blur reduces noise,
     iii. Edge detection highlights significant boundaries, and
     iv. Morphological operations clean up the image by removing noise and connecting disjointed elements to facilitate contour detection.
2. **Image Quality Evaluation:**

   evaluate_image_quality()

   - **Purpose:** Assess the quality of the image by detecting lines using the Hough Line Transform.
   - **Reason:** Ensures that only good quality images with detectable features are processed further. If no lines are detected, the image is flagged as poor quality to prevent further processing of low-quality images.
3. **Visualization:**

   visualize_processing_bad_data()
   visualize_processing()

   - Create and save visualizations of the image at different processing stages. Provides a clear visual representation of the image at various stages (grayscale, blurred, edges detected, and morphological operations result). This helps in understanding the effectiveness of the processing steps and the accuracy of stitch detection.
4. **Main Function:**

- Manages the workflow for processing images, evaluating their quality, detecting stitches, visualizing results, and recording the output. Ensures a systematic and automated approach to handle multiple images. It reads images from a directory, processes them, saves visualizations, and records the results in a structured format.

Output Analysis:

CSV File (`results_test2.csv`):

- Records the number of stitches detected for each image. Images of poor quality are marked with a `-1`.

Output Directory (`output_images_from_test2`):

- Saves visualizations of each processed image, including grayscale, blurred, edge-detected, and morphologically processed images.

Bad Data Directory (`output_images_with_bad_data_from_test2`):

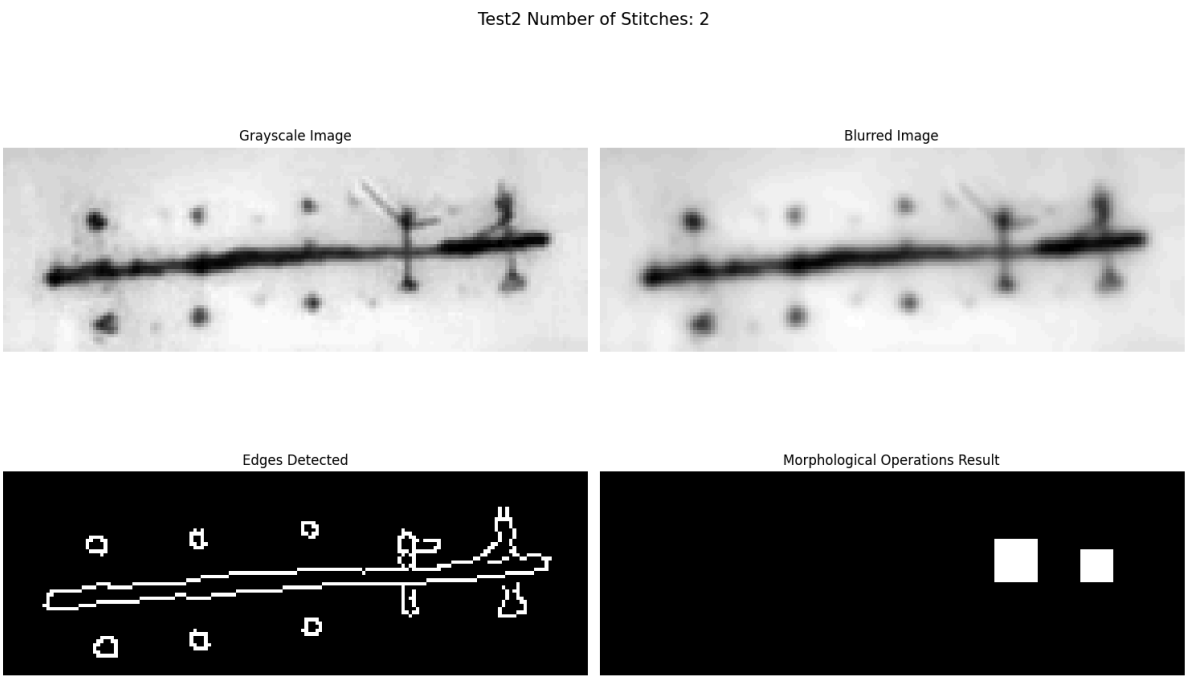
- Saves visualizations for images deemed to be of poor quality, showing the grayscale and edges detected.

Examples of what is working and what is not working:

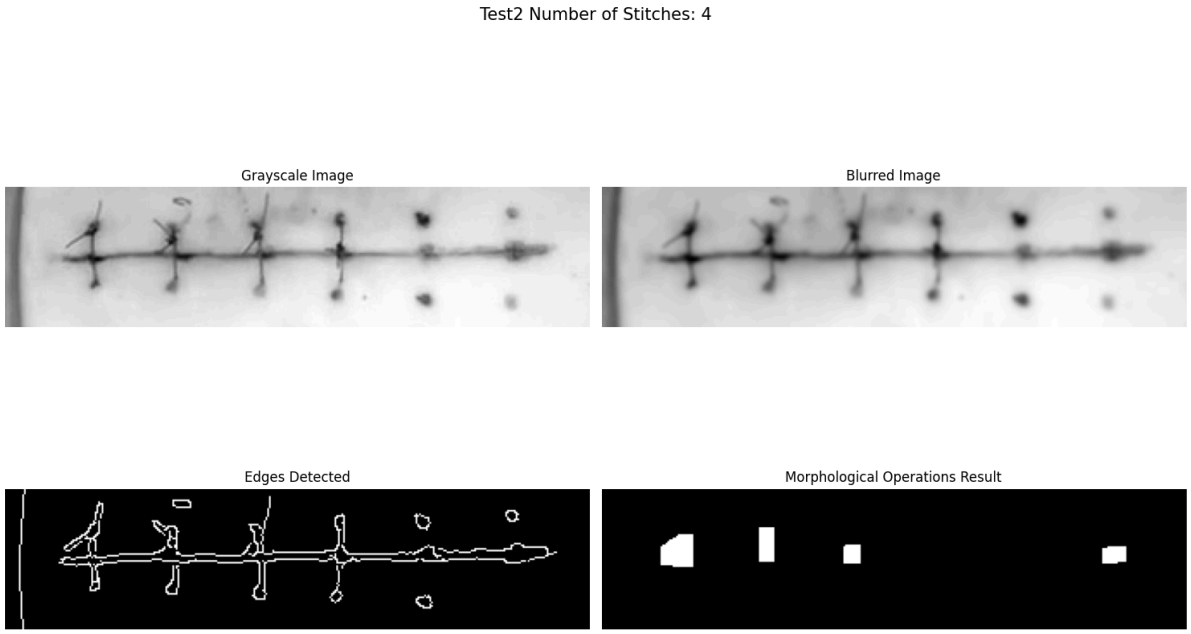
Bad Data: We used the same function as the function in 'test1.py' because we could not find a way to make it work better. So the results are the same.

Data: From 129 images 14 were correct which corresponds to 14.7% success rate.

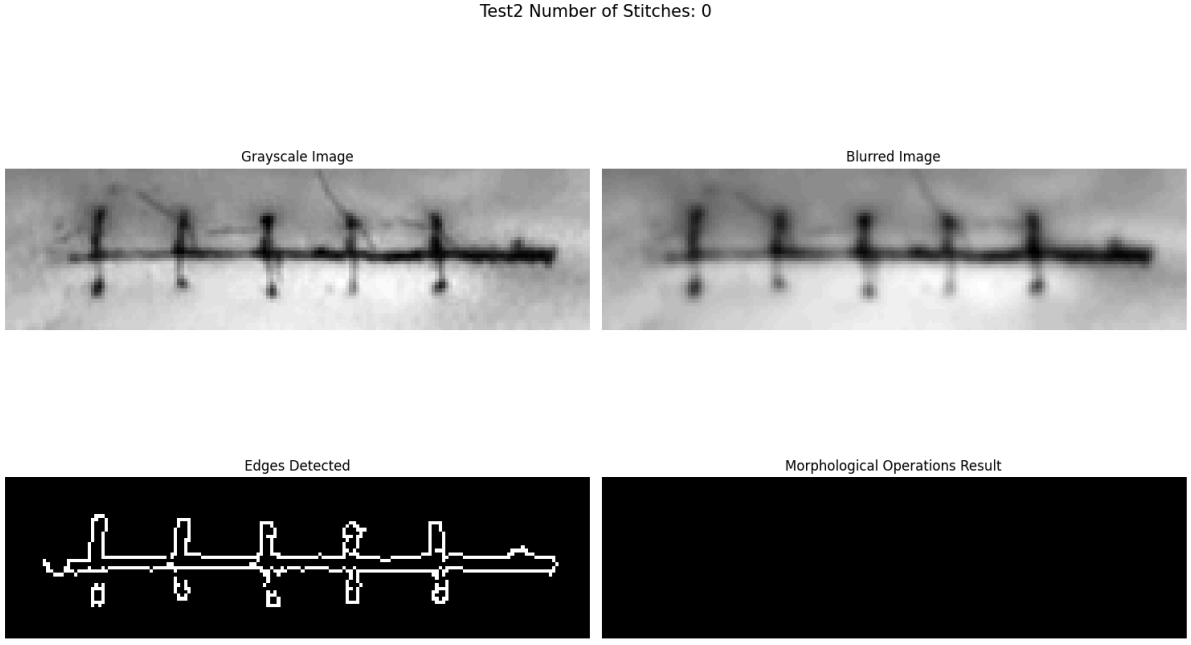
Correct result:



However, sometimes the stitches that were detected based on the intermediate results were wrong but the result was correct. For example:



Incorrect result:



Conclusion

Both *`test1.py`* and *`test2.py`* have been evaluated for their ability to detect and count the number of stitches in surgical images. However, test1 works better than test2 based on the success rate, as found above.

General Observations

1. Accuracy Issues: Both methods show inaccuracies in detecting the correct number of stitches. Contours and lines detected sometimes do not correspond well with the actual stitches.
2. Image Quality Checks: The quality evaluation steps help filter out poor-quality images but do not improve stitch detection accuracy.
3. Visualization: The visual outputs help in understanding the steps but reveal that the methods need refinement.

Overall, while the current scripts provide a foundation for stitch detection, improvements are needed to achieve reliable and accurate results in various image conditions.

How to run the project

In order to run the project in an IDE of your choice or in the terminal, type the command: *python test1.py* or try the command: *python test2.py* .
After successfully running one of the scripts, a new folder is going to be created with all the intermediate results named '*output_images_from_test1*' or '*output_images_from_test2*', respectively, also the folder containing the bad data will be named '*output_images_with_bad_data_from_test1' or* '*output_images_with_bad_data_from_test2'* respectively. The source of the images is the folder named *'test_images' .*