

Impact of Image pre-processing on Image Segmentation in Deep Learning for breast cancer detection

CS39440 Major Project Report

Author: Wojciech Sowinski (wos2@aber.ac.uk)

Supervisor: Prof Reyer Zwiggelaar (rrz@aber.ac.uk)

29th April 2021

Version 1.0 (Approved version)

This report is submitted as partial fulfilment of a BSc degree in
Computer Science And Artificial Intelligence (GG4R)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work, I understand and agree to abide by the University's regulations governing these issues.

Name Wojciech Sowinski

Date 27.04.2021

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name Wojciech Sowinski

Date 27.04.2021

Acknowledgements

I am grateful to Reyer Zwiggelaar for his guidance and patience that played a huge role in accomplishing the project.

I would like to thank Sreenivas Bhattiprolu for sharing knowledge, without which this project would not be possible.

Abstract

This paper presents an investigation into the impact of pre-processing on Breast Cancer Image Segmentation using a deep learning network on Ultrasound images only. In 2020, breast cancer became the world's most commonly diagnosed cancer, and early detection is a significant key in fighting that. To detect cancer using computer-aided-detection (CAD) software, it needs first to perform segmentation as it is a part of the staging process and a standard step before doing classification. Breast ultrasound (BUS) images are challenging due to the noise which can make segmentation inaccurate. The project aimed to perform a series of image pre-processing techniques on the BUS images to minimize noise's negative effect or enhance the image and use the pre-processed images as input for a deep-learning network. Different pre-processing methods were applied to the original dataset and used as input for the deep-learning model (U-Net). The project compares the output of the network using IoU, Dice-Coefficient measures the impact of many pre-processing techniques to find the most efficient one among them, for BUS images to achieve the most accurate results of image segmentation and to highlight the unique characteristic of the BUS data. The analysis of results could be a valuable insight in developing CAD systems for breast cancer detection and classification.

Contents

Table of Contents

1	BACKGROUND, ANALYSIS & PROCESS.....	7
1.1	Background	8
1.2	Analysis Process	9
1.3	Process	10
2	EXPERIMENT METHODS	12
2.1	Median Filter	13
2.2	Normalization.....	13
2.3	Multi-Otsu Thresholding	14
2.4	Canny (canny edge detector)	14
2.5	Blob detection using Laplacian of Gaussian	14
2.6	3-channel combination of images	15
2.7	Sørensen-Dice coefficient	15
2.8	Geometric Active Contour (GAC)	16
2.9	U-Net (Convolutional Neural Network)	16
3	SOFTWARE DESIGN	17
3.1	Data pre-processing.....	18
3.2	Image Segmentation using U-Net	19
3.3	Comparison and results saving.....	20
4	IMPLEMENTATION	21
4.1	Implementing Issues.....	21
4.2	Programming Language	22
4.3	Experimental results of implemented methods.....	23
4.3.1	Median Filter	23
4.3.2	Normalization.....	24
4.3.3	Multi-Otsu Thresholding	24
4.3.4	Blob Detection (LoG – Laplacian of Gaussian)	25
4.3.5	Canny-Edge Detection	26

4.3.6	3-channel images.....	27
4.4	U-Net architecture	28
4.4.1	U-Net implementation in Python.....	30
4.4.2	U-Net Training.....	31
5	TESTING – EMPTY FOR NOW	31
5.1	Overall Approach to Testing.....	31
5.2	Functionality testing.....	32
5.2.1	Automated Testing.....	32
	Integration Testing.....	32
5.2.2	32
6	RESULTS.....	32
6.1	Pre-processing comparison	40
6.1.1	Median filter and normalization results.....	40
6.1.2	Multi-Otsu thresholding results	41
6.1.3	Canny-edge detection results.....	41
6.1.4	Blob detection (LoG) results.....	42
6.1.5	3-channel combinations of images	42
7	CONCLUSIONS.....	44
8	CRITICAL EVALUATION	45
9	ANNOTATED BIBLIOGRAPHY	47
10	APPENDICES	50
10.1	Third-Party Code and Libraries.....	50
10.2	Ethics Submission	51
10.3	Code Samples	54

1 Background, Analysis & Process

In 2020 the breast cancer became the world's most commonly diagnosed cancer. Every day in the UK, there are new 150 breast cancer cases: 55,200 every year. Breast cancer kills 11,500 women and 85 men in the UK year by year and is a leading cause of death in women under 50 in the UK. However, early detection of cancer or abnormal tissue reduces the chance of dying from breast cancer and may make the treatment more accessible. Detection of breast cancer is available thanks to screening tests such as mammography, MRI. Ultrasound screening is not a good tool for cancer screening itself, but it often complements other screening. According to the American Cancer Society, the rate of 5-year relative

survival is 99% when breast cancer is detected early. Many computer-aided-detection (CAD) software has been developed and continues to develop as a tool for doctors to help detect and classify breast cancer worldwide using various technologies, machine learning, and deep learning. Despite the wide distribution of these kinds of software and the work put into this field, there still occurs false-negative results or false-positive results (Berg W.A. et al., 2012) (O'Flynn E. A et al., 2015), overdiagnosis - therefore, work in this area continues to improve cancer detection. The purpose of this project is to reveal the importance of image pre-processing in a deep learning pipeline. Project will focus on comparison of segmentations in deep learning performed on breast ultrasound images, exposing the difference in results caused by image pre-processing. The image pre-processing is a necessity before approaching to deep learning segmentation; however, the research reveals which are the most suitable for BUS images. The most challenging problems were ultrasound artifacts and noisiness of images from some of datasets. The dataset that's been used for learning purposes came from 4 different origin datasets: RODTOOK, BUSI, BUS_UDIAT and OASBUD. The BUS images vary in terms of noise and quality as shown in figures 1-4.

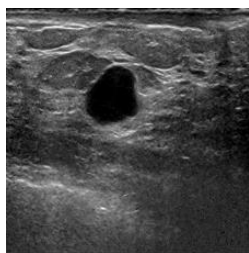


Fig.1 BUSI_UDIAT's example BUS image



Fig.2 BUSI's example BUS image

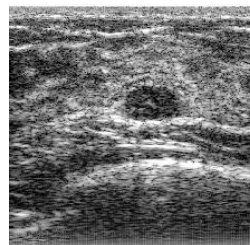


Fig.3 OASBUD's example BUS image



Fig.4 RODTOOK's example BUS image

1.1 Background

The motivation for this project was based on my interest and course, which is Artificial Intelligence. I wanted to find a project that would involve deep learning as it dynamically develops every year. The project involves technologies that are an excellent opportunity to acknowledge before finding a job in a similar field, Artificial Intelligence. Social I knew a bit about the ever-growing problem and danger of breast cancer, so I thought I could somehow contribute to developing a technology that fights this type of cancer. I must admit that this thought motivated me the most throughout the entire period of creating this project. At the first meeting with my supervisor, Prof. Reyer Zwiggelaar, we talked about potential topics, and he suggested a few, including the 'comparison of pre-processing in segmentation.' Research exposed a gap in this field as there is only a little research on this topic, which is essential, as pre-processing is used in almost every project involving segmentation and deep learning. The motivation became filling this gap. Besides, due to the growing popularity and use of machine learning and mainly deep learning in today's technology, which will probably shape the future, the project also involved the CNN network for segmentation, whose output will reveal its impact pre-processing techniques. Research about breast cancer shows that the most popular breast screening is mammography, while ultrasound screening is usually complimentary screening. However, ultrasound may be more sensitive in case of detecting invasive cancer in dense breasts (Constantini M. et al. 2006) (Drukker K. et al. 2002). In the beginning of the project, I was not aware of that disproportion in screening results. Nevertheless, still so many false-negatives and false-positives cases occur, even in mammography screening. Finally, the choice fell on ultrasound images that will eventually eliminate the uncertainty. The project's fundamental research involved studying more

deeply Computer Vision, which is the foundation for that project, as image pre-processing is nothing else than just manipulation on images to enhance the image to perform better segmentation. From CS36220: Machine Learning module, I learned a lot about data manipulation and machine learning model adjustment, but I knew very little about deep learning. It was an invaluable opportunity to learn about that field and practically use it to do the segmentation. Although CNN networks are an enormous field to study, I learned about TensorFlow, Keras, various architectures, and modelling my own network. Additionally, the project required better knowledge of the Python programming language, which I did not know that well, so it made me catch up.

1.2 Analysis Process

The main problem of breast cancer classification or detection, and so in fact, segmentation as the standard step before approaching them, is inaccuracy. The research and analysis of the problem showed that multiple reasons cause the inaccuracy in ultrasound screening, and the pre-processing performed on images before segmentation reduces the impact of main faults in images such as ultrasound artifacts and noise. The analysis of the problems reveals the first task in the project, which is pre-processing. The choice of technique was oriented towards eliminating noise since the avoidance of ultrasound artifacts lies mainly with the radiologists, as ultrasound screening is very operator-dependent modality. Of course, for a deep learning approach for segmentation, it is crucial that pre-processing provides as much information as possible from which the model can learn. An essential element was the understanding of the dataset. Abnormalities, which the model wanted to segment, were often characterized by a black background and a rounded shape. Depending on the photo, they were surrounded by tissue, which was a light colour, approximating white in the photos. Unfortunately, some of the photos, mainly from the OASBUD dataset, had a lot of white noise inside the abnormalities. On the other hand, in the BUS_UDIAT dataset, while it was easy to distinguish the centre of the abnormality from the tissue, it was hard to see the boundaries of the abnormalities in some of the photos. In summary, the technicians had to level out the noise, highlighting the boundaries, while being careful not to even threshold, lest they lose valuable information that would have helped segment the abnormalities. In projecting a deep learning model, it is necessary to have lots of data; otherwise, the traditional machine learning algorithms perform better. My supervisor provided me a dataset containing 1154 images from different origin datasets of size 224 x 224 pixels: each with corresponding mask. That amount of data was enough to train the model. The next major step, after pre-processing, naturally was to implement the deep learning model that will take images and corresponding masks as input for training purposes and output the segmented image. Saved output images are then used to analyse the results, using various methods, such as dice-coefficient, intersection over the union, or simply comparing the output image with the corresponding mask used as a ground truth. The analysis results expose the advantages and disadvantages of using specific pre-processing techniques according to the type of image or lesion and provide the most optimal ones. The research project, based on the performed experiments, answers what impact on performance of deep learning segmentation have following methods alone, or in combination:

- Median filter
- Canny-edge detection
- Blob detection using Laplacian of Gaussian
- Normalization

- Multi-Otsu thresholding
- Combination of each technique in 3-channel image:
 - Median filter + Normalization = red channel, Blob Detection = green channel, and Original image = blue channel
 - Original image, canny-edge detection and Blob Detection
 - Original image, median filter + normalization and Blob Detection
 - Canny-edge detection, multi-Otsu thresholding and median + normalization

Results will reveal the pros and cons of using each of the pre-processing techniques. They will show the segmented image and the corresponding mask, calculations of dice-coefficient from output and masks, and expose the difference between baseline output from a model trained on the original dataset and output from a model trained on a pre-processed dataset, highlighting the pre-processing as a necessary step in image segmentation.

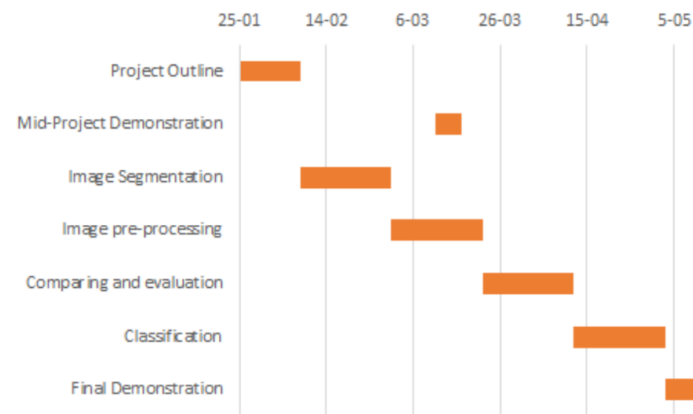
Segmentation of BUS images using a deep learning network required research of numerous CNN architectures that would efficiently and accurately segment the images. Almost every day, new architectures are invented, and so it became challenging to choose one of them. The most popular architectures are: LeNet, AlexNet, VGG, InceptionNet/GoogleNet, ResNet, FCN-AlexNet, U-net, and many others. "Automated Breast Ultrasound Lesions Detection Using Convolutional Neural Networks" [6] study finds that in BUS lesions detection where the best performance is achieved by U-Net and FCN-AlexNet when training and testing on a single dataset. Ultimately, I decided to use U-Net architecture for the implementation of the deep convolutional network due to the fact that U-net is fast and efficient, was primarily invented for biomedical segmentation, but other research proved (Siddique N. 2020) that the network and its variations are very efficient in segmenting medical images such as brain, liver, breasts or other.

1.3 Process

When preparing for this project, I meticulously approached its planning to be prepared for most of the problems and systematically progress the work. I was sure that the iterative and incremental development approach would be the perfect solution for me and this project from the very beginning. The problem analysis identified the main tasks, which in a result, divided the time from the start of the project to the deadline into the main segments. In the analysis mentioned above, the following main tasks emerged:

- Pre-processing of images
- Segmentation of images
- Comparing and evaluation

Additionally, according to the program, I had to prepare a project outline, mid-project demonstration, and final demonstration. Weekly meetings with my supervisor were extremely helpful in the whole process of creation the research project. Initially, my plan was to do breast cancer classification, as the fourth of the main tasks, using the previously segmented area, I would extract features that would be used to classify lesions. The classification results could be used as an additional measure to evaluate and compare pre-processing techniques and their impact on eventual classification. Classification, however, from the beginning was planned as an additional task that would be completed if time allowed. As usual, not everything went according to plan and the task had to be abandoned due to lack of time. As shown in figure 5, Gantt chart was designed in accordance with the above assumptions for the project.



[Fig. 5 Gantt Chart of project plan including main tasks and other.

After receiving these three main tasks, I decided to break them down into smaller ones to include smaller tasks in the plan. As a result, it allowed me to estimate the project's possibilities and time and make the project continuously progressing. The order of tasks changed right after completing Project Outline. Image pre-processing has swapped from Image Segmentation. From the first task, which is "pre-processing of images," I extracted the following methods:

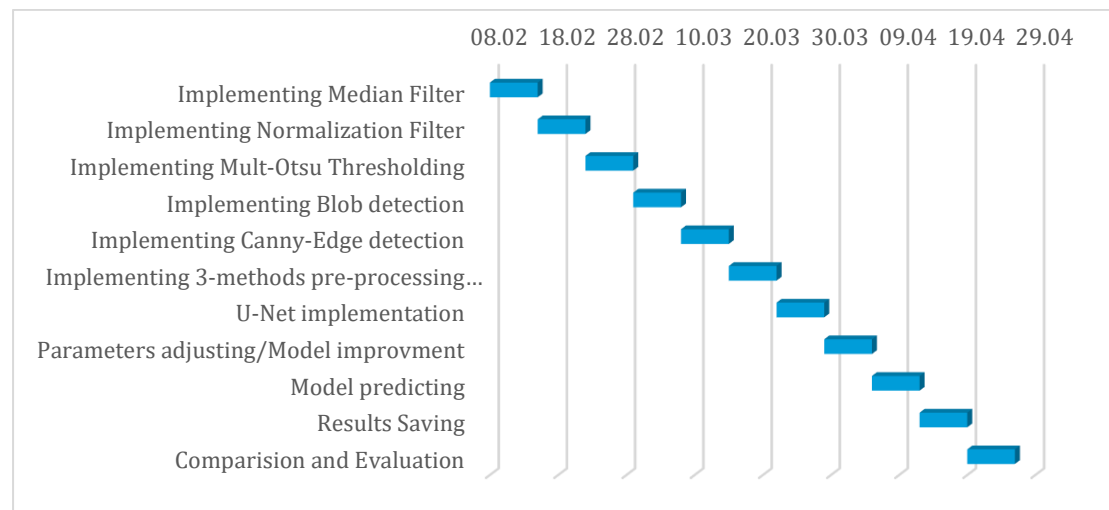
- Median filter using kernel 5x5
- Normalization
- Multi-Otsu thresholding
- Canny-edge detection
- Blob-detection using Laplacian of Gaussian
- Combination of methods in 3-channel images

Then, in planning "Image segmentation," I had to do research that would allow me to plan the appropriate steps. Naturally, the first step was to implement the network. Initially, I thought about using the library with segmentation models, but as a result, I decided to independently implement U-Net architecture from scratch, which I found more exciting and demanding on my part while providing valuable learning and experience. This and the following steps allowed me to divide "Image Segmentation" into the following segments:

- U-Net implementation
- Parameters adjusting
- Model training
- Model predicting
- Results saving

The last of the three main tasks was "Comparison and evaluation." In order to accomplish this task, I decided to use Dice-coefficient or "statistic used to gauge the similarity of two samples," Intersection over Union and resulted image to visualize the difference in segmented areas simply. The time initially allocated to classifications was used to write the final report. From 8th Feb to 15th Apr, it was ten weeks. The tasks differed in the level of difficulty, but I decided that assigning one week to the task would be a good solution, and tasks that would be completed faster than they would take would allow for a deeper focus

on others, such as U-Net implementation or model training which we know can be time-consuming. The figure 6 shows Gantt Chart with the described plan of completing the tasks.



[Fig.6 Gantt Chart of detailed plan including 11 main tasks to complete.

The plan did not include writing the final report, which would be created along with the project progressing. In practice, however, a significant part of the report was created in the last two weeks of work and was based on notes, recorded results, as well as the last and most crucial part of the project, i.e., comparison and evaluation. Overall, the iterative approach has perfectly adapted to this type of project. The analysis of the problem and the tasks that come out of it, allowed me to construct a progressive plan in which the cycles lasted a week and were conditioned by individual tasks. This approach was, in my opinion, as realistic as possible and reduced the chances of failure; or not completing most of the tasks due to the fact that each task was given one cycle, which in fact was timely appropriate for the most demanding of these tasks. Working this way, I could realistically complete most of the planned tasks at the beginning of the project and be prepared for problems that happened or could have happened during the project's creation.

2 Experiment methods

The overall hypothesis being tested is a statement that pre-processing, and augmentation of these pre-processed results can improve deep learning models for BUS data. The likelihood of this hypothesis being true may also be emphasized by the fact that in almost every scientific work dealing with segmentation - not necessarily using only the convolutional neural network - pre-processing was an applied step before segmentation was performed. However, there are many pre-processing methods, and some of them do not always have a positive effect on image segmentation results. In addition to examining the hypothesis, the conclusions of this project will also be supported by the results and insights regarding various pre-processing techniques and their exact impact on image segmentation results.

Project is based on true experimental research design. The images are pre-processed using various techniques and then used as input in U-net model. The model is trained on datasets that have been pre-processed using different techniques. The output of the model is segmented image that is used for comparison with its corresponding masks. The similarity is

calculated using Sørensen–Dice coefficient – standard statistic used for calculating the similarity between two samples – in this case segmented image and corresponding mask.

Original dataset of images containing 1154 images was divided into two, with proportion: 1079 images for training purposes and 75 images for model testing. Pre-processing made 10 different datasets that are following:

1. Original dataset
2. Blob detection using Laplacian of Gaussian dataset
3. Canny-edge detection dataset
4. Multi-Otsu thresholding preceded by normalization & median filtering dataset
5. Median filter and normalization dataset
6. 3-channel image where for each channel is used different method
 - a. Canny-edge detection, Blob detection, Median&Normalization
 - b. Canny-edge detection, Original, Blob detection
 - c. Canny-edge detection, Multi-Otsu thresholding, Median&Normalization
 - d. Median&Normalization, Blob detection, Original

2.1 Median Filter

Efficient, a non-linear method of image processing used to remove noise or signal from an image. The median filter is the simplest non-linear spatial filter, reducing noise and preserving edges better than convolutional filters (Connistraci, C.V. et al. 2009) The median filter works by going through the input signal upon input (for photos through a 2D array), where each entry encounters changes using the median of adjacent entries. The "window," as we call the pattern of neighbours, moves continuously throughout the signal, changing its values.

$$y[m,n] = \text{median}\{x[i,j], (i,j) \in w\}$$

w represents a neighbourhood (*window*), centered around location $[m,n]$ in the image

2.2 Normalization

The next step in image processing is also very often normalization, which I also decided to use in this project. It is a process that changes pixel values within a given range (Zhany Y. et al. 2011). Images are normalized by subtracting the mean value from each gray level. The results are divided by the standard deviation. This step is important as it keeps the image pixel intensity values in the range of 0 - 255. The purposes behind applying normalization are usually bringing the image into a normal range to senses and bringing consistency among other images - in this project's dataset - for training and testing purposes.

$$x' = \frac{x - x_{\text{mean}}}{x_{\text{max}} - x_{\text{min}}}$$

where X_{max} is highest intensity values of image and X_{min} is the lowest.

X mean is mean intensity value of image

During data processing on this project, normalization is only used in tandem with Median Filter.

2.3 Multi-Otsu Thresholding

In image processing, a standard and the simplest method for image segmentation is thresholding. This method simply replaces each pixel in an image with a white pixel if the intensity of image $I_{i,j}$ is greater than fixed constant T (that is $I_{i,j} > T$) it is replaced by black if intensity is less (Dim and Takamura, 2013). There are more than one thresholding methods from various categories, but I decided to use the multi-Otsu Thresholding. This type of thresholding varies from ordinary, binary thresholding in a way, that instead of dividing image into two groups of pixels (black and white) its thresholding algorithm separates pixels of image into several different classes accordingly to the intensity of the gray levels within the image. In my project the number of determined classes for thresholding was 5: for obtaining three classes; the algorithm returns 4 thresholds.

2.4 Canny (canny edge detector)

Edge detection is the process of marking points in a digital image where luminance changes rapidly. Sharp edges usually mark important elements in the photo. The changes in luminance are caused, among other things, by changes in the depth or properties of the material, which in the case of abnormalities in the breast tissue is desirable to detect. Many edge detection methods are based on the first derivative of luminance, which gives us the increment (gradient) of the original data value (W.K. Pratt, 2001). Using this method, we can find the extremes of the first derivative. If $I(x)$ represents the luminance of a pixel x , and $I'(x)$ is the first derivative of pixel x , you can write:

$$I'(x) = -\frac{1}{2} \times I(x-1) + 0 \times I(x) + \frac{1}{2} \times I(x+1)$$

One of the operators used in edge detection using the first derivative is Canny edge detection, a multi-step algorithm developed by John F. Canny. The algorithm consists of 4 main steps. The first step is noise reduction (due to the use of the first derivative, which is noise sensitive) using a Gaussian filter. The next step is to look for the intensity of the image gradient, the next step is to remove non-maximum pixels, and the last step is to threshold with hysteresis, which is to remove irrelevant edges with a specific slope (Zhou P et al. 2011)

2.5 Blob detection using Laplacian of Gaussian

Blob detection methods in computer vision are used to detect regions in a digital image with different properties, such as light or color, from the surrounding regions. Convolution is the most popular approach for blob detection. Blob detectors are divided into two categories: (1) differential methods, which are focused on the function's derivatives with respect to

position, and (2) methods based on local extrema, which are based on finding the function's local maxima and minima (Han and Uyyanonvara, 2016) In this project, the Laplacian of Gaussian computations will be used as it is the most common method for blob detection, successively increasing standard deviation and stacking images up in a cube, where blobs are local maximas this cube. This approach is very accurate and very slow compared to other methods, as it struggles with detecting larger blobs due to the larger kernel size during convolution (scikit-image.org, 2021).

2.6 3-channel combination of images

Deep learning network (U-Net) implemented in this project has RGB images at the input. My project supervisor, prof. Zwiggelaar suggested to take advantage of that and use the three channels for different images. One channel would take the original image for the R channel, Canny edge image for G channel, and the blobs for the B channel. The idea behind that is to provide image with more information together that the network would take into account, the same as a color images uses red, green and blue information. I have created 4 different methods with different combinations:

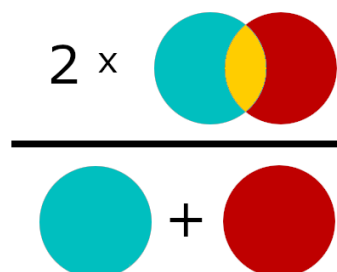
1. Canny edge detection, Median filtering & normalization, Blob detection
2. Canny edge detection, Original image, Blob detection
3. Canny edge detection, Multi-Otsu thresholding, Median filtering & normalization
4. Median & normalization, Blob detection, original image

2.7 Sørensen-Dice coefficient

The final analysis based on the comparison resulting in images from the U-Net model, a ground truth, or provided masks of the abnormalities, is mainly based on the Sørensen-Dice coefficient, a well-known statistic to measure the similarity between two samples introduced by botanist Thorvald Sørensen. The coefficient is the result of the formula:

$$DSC = \frac{2 \times |X \cap Y|}{|X| + |Y|}$$

Where $|X|$ and $|Y|$ are the cardinalities of the two sets. Dice coefficient is widely used as a metric to evaluate **semantic** segmentation model. To simply explain it, Dice coefficient is *area of overlap* indicated by yellow color in figure below, divided by the total number of pixels in both images. The good visual explanation of the formula is shown in figure 7.



[Fig.7 Visualization of formula used to calculate Dice-Coefficient. Yellow color indicates area of overlap, blue and red indicate two different images.

Simply put, the Dice coefficient is just counting pixels in the two images. It is worth mentioning that the segmented image from deep learning network trained and tested on the original dataset only - without any interference with these images- will be used as the baseline used to evaluate the effectiveness of various pre-processing methods. Such a baseline will give objectivity to the entire study and its results. Thanks to this, I was able to determine which of the methods realistically and to what extent each method improves the deep learning model for BUS data.

2.8 Geometric Active Contour (GAC)

Additionally, to reinforce the factual nature of the results, I decided to perform additional original image segmentation using the Morphological Snake of the image segmentation method family, and the exact segmentation method used was Morphological Geodesic Active Contours. In computer vision, GAC models are used to outline a noisy image. As a result, they are used, among others, for edge detection, shape recognition, or image segmentation. GAC, which was used in employing in medical image in the past, uses Euclidean curve shortening evolution, where contours split and merge. The equation of GAC using the gradient descent curve is:

$$\frac{\partial C}{\partial t} = g(I)(c + \kappa)\vec{N} - \langle \nabla g, \vec{N} \rangle \vec{N}$$

where $g(I)$ is a halting function, c is a Lagrange multiplier, k is the curvature, and \vec{N} is the unit inward normal.

GAC with number of iterations in the evolution that has been carried out is 230, using the skimage library, on a sample image without prior pre-processing, returns us a segmented image which is then compared to its mask. The dice coefficient will be then calculated between these images and use for comparison of the results at the final stage.

2.9 U-Net (Convolutional Neural Network)

In this project the image segmentation is done by using an architecture called 'U-Net'. U-Net, a fully convolutional network, has special type of architecture for image segmentation, and by saying 'architecture' I mean the arrangements of the deep learning tools that are already known, for example convolutional layers and maxpooling. The tools are arranged in such a way that the result would be image segmentation. It is worth mentioning that U-Net architecture is designed for semantic segmentation, which is different from instance segmentation. The figure 8 visualizes the difference between the segmentations. As shown in figure 8, semantic segmentation simply uses two colour values to separate background from foreground, or in other words, to separate human from non-human. The instance segmentation is extension of semantic segmentation, where each person is separated and become a distinct object. In my project, the semantic segmentation is the most suitable, as in every image is only 1 abnormality we are looking for, so there is no need for instance segmentation. We separate abnormality from background. The architecture of this deep learning network looks like 'U' letter, that's why it's called U-Net. The details of U-Net's architecture are explained in Implementation Chapter.

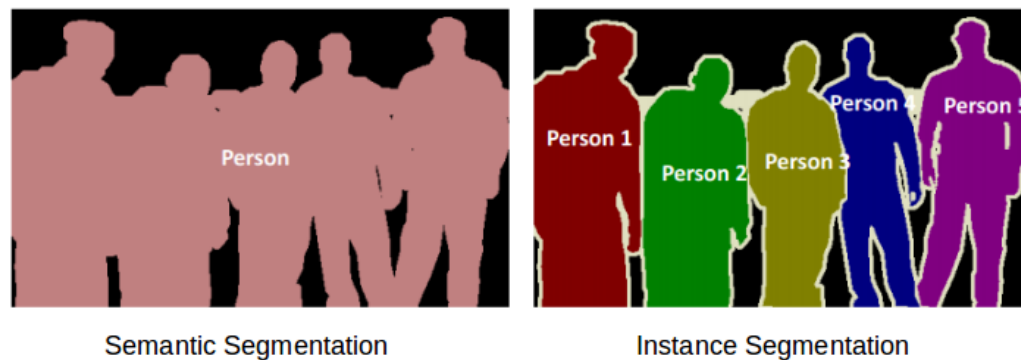


Fig.8 Visual explanation of the difference between semantic and instance segmentation. Source: <https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/>

3 Software Design Software Design

Due to the nature of this project, i.e., research area, also based on machine learning, which requires an efficient GPU, which my laptop does not have, the project's design was, I would say, primarily forced on me. Naturally, I could have chosen a different approach, but in the end, I decided that the design discussed above would have a positive effect on my work. Working on the Google Colab notebook, as well as following the prescribed plan, my software design had to be a procedural programming design. The program is procedural with the following steps: implementing libraries, pre-processing methods, reading the datasets, operations on the dataset, implementing U-Net but also training and testing. All in one place. In the case of object-oriented design, the UML Diagram of classes or use cases are often used to describe design. Due to the software in which the software was implemented (Google Colab), as well as its procedural programming approach, I created three main steps to describe the design that describe the entire course of the project, as well as software development and the sequence of its operation. These steps are:

1. Data pre-processing
2. Image segmentation using U-Net
3. Comparison and evaluation (results saving)

Google Colab allows us to create "cells", where we place the code. We can run specified cells without forcing every other cell to run. This way, we can save loads of time for loading and performing operations we have already performed, or we simply do not need it anymore. Due to this, I have created 3 maps of operations, consisting of the main methods, indicating the sequence of their execution, allowing each of the 3 main steps to be completed. In this way, we can perform selected operations based on the selected target, and the program is structured to make any modification very simple. Assuming that we want to add a new pre-processing method, it is enough to define it in the newly created cell and run it. According to the maps presented below, repeat all the steps required to achieve the adopted goal, considering the newly created method in the already programmed function as one of the accepted arguments. I considered rewriting the program to the form of object-oriented programming but found it superfluous. The program in Google Colab allows you to run it and intervene or change the input datasets with great ease. The methods are universal, and I think that even for a person unfamiliar with deep learning or even computer vision, or even Python, it will be easy for them to read and manipulate the program. Most functions are

programmed in a way, to take arguments - directory path of dataset or pre-processing method, making program easy to operate with.

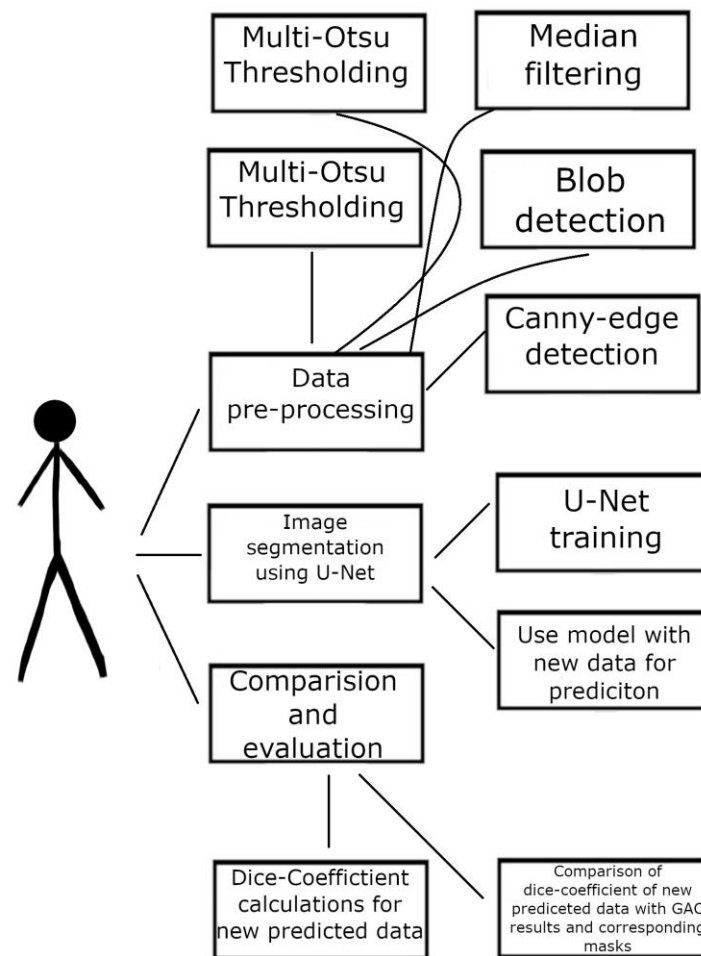


Fig.9 Use-case diagram of the project's software.

It is worth mentioning that a transfer of this software from procedural programming to object-oriented programming design would be fairly easy to achieve due to the 3-steps division. The program would have four classes: three for each step and fourth for running the program. The other advantage of Google Colab is updated Python to the newest version (3.7) and TensorFlow is already pre-installed and optimized for the hardware being used. This is huge help, as when I worked with PyCharm on my laptop I encountered many problems relating to the nature of Python compatibility with TensorFlow and Keras. Again, it becomes easy to run the program for someone without ready and set environment. The data is imported to the Google Colab through Google Drive. It is possible to set the Colab to run it on a local computer and use it on the hard drive. However, code can be run in any IDE supporting .py files, but preferable to run the code is Google Colab using .ipynb files. The following sections provide brief descriptions of the design and each of the three main steps and their operation and purpose.

3.1 Data pre-processing

The data pre-processing step is fundamental for this project. This step I performed multiple times, each time for different time of pre-processing technique. While I had pre-processed

data, I could skip this stage to start Image segmentation, and later evaluate and compare the results. The program needs to implement libraries with methods to manipulate the data.

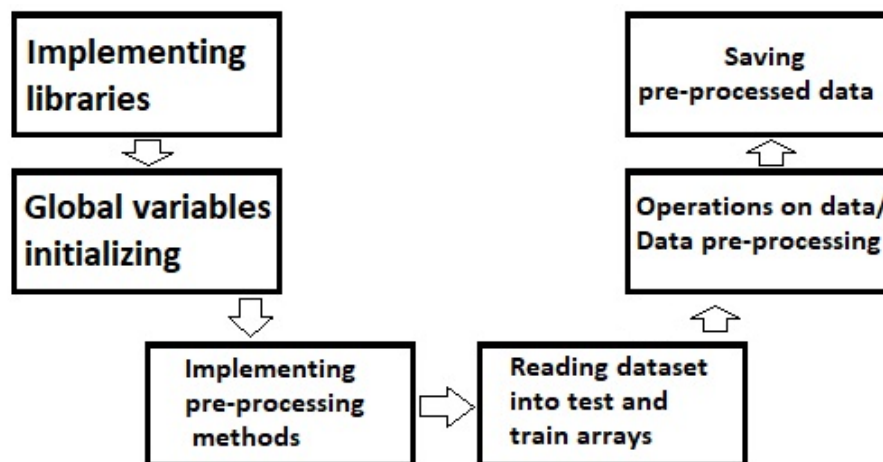


Fig.9.2 Diagram for data pre-processing with indicated, sequenced steps.

The program needs to implement libraries with methods to manipulate the data. Then it initializes the global variables such as arrays for train and testing images. Using various libraries, it implements pre-processing methods. Program reads the specified dataset from given directory – usually original dataset. Along with reading, the program is pre-processing the image currently read and saves it in suitable array. The array of pre-processed images is then saved in specified directory. The pre-processed dataset is then used in Image Segmentation. Most of the project work was preparing the data for image segmentation, thus dividing the project into 3 different steps seem reasonable since this design can save loads of time. It would be senseless and extremely time and resource consuming to pre-process the dataset using various techniques every time the program starts. For example, the pre-processing using multi-Otsu thresholding on the original dataset (1079 train images) took almost 5 hours using free CPU – as GPU has 12hours limit which I reached that day. Multi-Otsu thresholding took less than 2 hours while using the free GPU. This proves and ensures that this design approach is completely reasonable.

3.2 Image Segmentation using U-Net

Performing Image Segmentation step design also is requiring the implementation of libraries and initializing global variables. Program loads specified dataset (train and test data), that has been pre-processed – except the segmentation of original dataset for future comparison, and also the corresponding masks that are used as one of parameters when the program compiles the model for training purposes as a ground truth image that the network tries to achieve the most close results.

The deep learning implementation is described in future chapter; however, it is worth mentioning that in the program in Google Colab it requires running only few cells to implement, train and predict the images. Running the Image Segmentation takes usually less than 0.5h as pre-processing step is skipped and it simply loads.

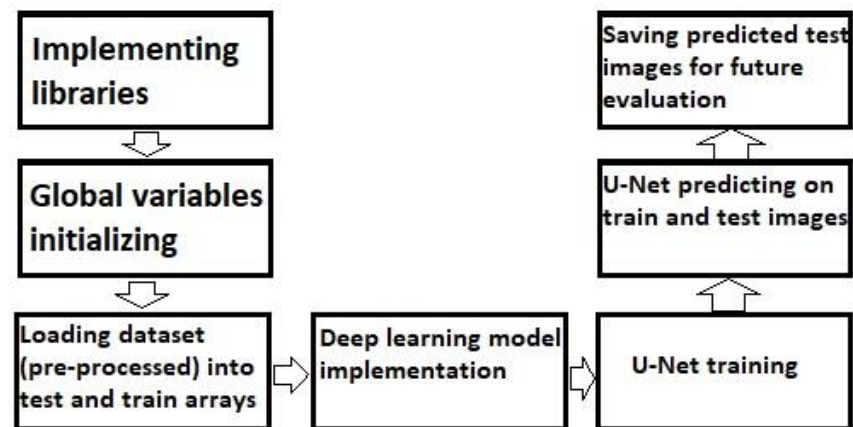


Fig.10 Diagram for data Image Segmentation using U-Net with indicated, sequenced steps.

3.3 Comparison and results saving

Comparison can be done separately without implementing U-Net model or pre-processing methods. Having saved, segmented images the program can simply load the data such as:

1. U-Net Segmented images from original dataset
2. U-Net Segmented images from specified pre-processed dataset
3. GAC Segmented images from original dataset
4. Corresponding masks

The program allows to calculate dice-coefficient of provided 2 images. It is suggested to first perform calculations using U-Net Segmented images from original dataset and dataset of corresponding masks. Secondly, it is advised to repeat the process with using U-Net Segmented images from specified pre-processed dataset and dataset of corresponding masks. At this stage, the initial conclusion of the performed image segmentation using specified pre-processing technique can be drawn. The segmented images of original dataset without any pre-processing are a baseline for the segmentation. Thirdly, to enhance the results, the software allows to load dataset of segmented images from original dataset using GAC segmentation and use the images in dice-coefficient method as a parameter. GAC is used a second baseline.

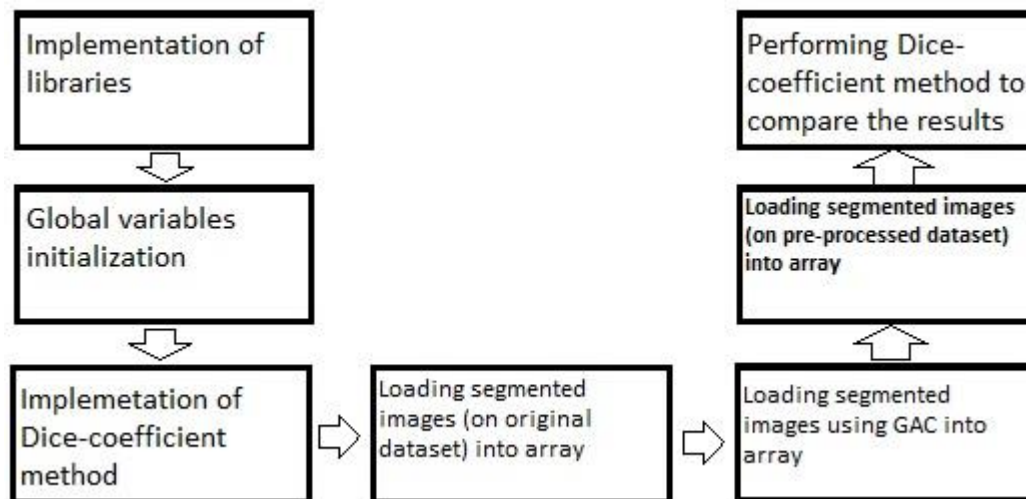


Fig.10 Diagram for comparison and results saving with sequenced steps.

4 Implementation

The implementation of the software was highly dictated by the web IDE for python. Due to required resources for a deep learning, the software was written in Google Colaboratory that allows to write and execute python code in a cloud. It provides free GPU and CPU and is suited to machine learning, data analysis and education. [28] The software uses many libraries made for computer vision and machine learning purposes.

4.1 Implementing Issues

Implementing the methods mentioned above, such as Median Filter, Multi-Otsu thresholding, and others, was the first thing in my project. Most of the methods were available to implement using skimage or cv2 libraries. As the dataset is made of images, sometimes it complicated the managing the data, due to its various types after performing the method, e.g., normalization changes the image dtype from uint8 to float64. There were many muddy points throughout the whole project like this, or sometimes even more complicated.

The implementation of a method that read the images and masks from directories was complex. The problem with that was to read images and the masks in the same order, while when I was performing an automatic method for reading every file in the directory, although the names of images and masks files were the same, the program was reading the files in different order. As a result, it took me some time to figure out why the results from U-Net segmentation are very poor. The reason was the different order in arrays where images were loaded to. To solve this, I had to use glob method to sort images' names and then read the images and their names to a separated array of names. I read the corresponding masks to the array in the same order as BUS images were loaded, based on the array of names. I have read somewhere that implementing and training the deep learning model is 20% of work, while 80% is about preparing the data. I believe that this

thought ideally reflects my work. The big issue was pre-processing the data using the various techniques, especially because I have been using a free version of Google Colab, which was also a complication due to the 12h free GPU usage limits. In addition, pre-processing 1079 photos using the Multi-Otsu thresholding sometimes took several hours, which combined with other techniques to get a 3-channel image where each channel corresponds to a different technique was really time-consuming. Google Colab also often caused problems that weren't necessarily my fault. There were frequent errors, pre-processing, or U-Net training interrupted for unknown reasons, so I had to reset the runtime and repeat the steps all over again, often wasting a lot of time. Next time I would choose the paid version of Google Colab or look for a different solution. Certainly, a computer with a powerful GPU using the Anaconda environment would be an easier and more stable solution.

Blob detection using Laplacian of Gaussian also made me a bit of trouble as I had problems with understanding how this method works, as well as its implementation. I think the problem was my deficiencies in the field of image data types. With the help of prof. Reyer, I managed to understand the operation of the method from the skimage library, which in effect allowed me to draw blobs on the newly created image. The whole problem was also a valuable lesson related to image data types, which came in handy later.

The implementation of the dice coefficient also caused me many problems, and I spent about two days solving them. While the formula and code appear to be easy to understand and implement, similar to the blob detection, shape, and image data type of masks and segmented images, they were different. Although the problem seems to be trivial because it turned out that the segmented image is not a binary image when read again into the program, its view did not give this impression. I had to apply thresholding to get a black & white version of the segmented image and then convert it into binary type. The dice-coefficient method, which previously returned incorrect results ($\text{dice} > 1$ or $\text{dice} < 0$), finally started working correctly.

4.2 Programming Language Programming Language

The language in which all software is programmed, including methods and all architecture deep learning network, is Python language of version 3.7. It is the most popular language used for this type of project, where machine learning is the key element. Although I did not feel comfortable using this language at first - I took it as a right moment to learn, which in fact did not take me that long because Python is very readable, intuitive and has many libraries that speed up work. The primary IDE where most of the code was implemented was in PyCharm from JetBrains. This is where I implemented most of the image pre-processing methods. Image operations would have failed if not for specially created libraries such as OpenCV or Skimage. The "Numpy" package allows operations on multidimensional arrays and matrices, so in this project, it was used to manipulate values in images or convert between arrays. Another library used in software implementation is Matplotlib - a library for creating graphs for the Python programming language and its numerical extension NumPy, which helped in debugging the code by, for example, creating graphs like the one above regarding Blob detection, and also allowed me to save images in selected folders. Of course, system libraries were also helpful, for example, the random function or provided function glob (), which allowed me to program smooth reading of images based on file names in the provided folder, which, as a result, allowed me to match images to their corresponding masks for training purposes.

Finally, we come to the tools used in the implementation of a deep learning network. Thanks to the open-source platform by Google Brain team, full of tools, libraries, and community resources, I implemented my network. However, to make things even easier, I

used the Keras library, which is also open-source and provides a Python interface for artificial neural networks. Keras acts as an interface to the TensorFlow library. It has many tools that allow you to implement the network in an effortless way. At some stage, as I mentioned, I wanted to use a ready segmentation_models [x] library, which allows the segmentation model to be implemented using just a few lines of code and additionally provides pre-trained models. Finally, for the purposes of learning, I tried to create my network from the first layer to the last. When the network was fully programmed, and I started the first test training, it turned out that the computing power of my graphic card is low, while the model training the network is expensive. As a result, training on a small number of epochs took a long time. While doing research or watching tutorials on different platforms, I found out about Google Colab, a product from Google Research, which is based on a well-known Jupyter notebook that allows users, in simple words, to run code in a web browser. It uses data cleaning and transformation, machine learning, numerical simulation, and much more [x]. Google Colaboratory is specially designed for creating machine learning models or data analysis. However, in my case, the most significant advantage that convinced me of Google Colab was the fact that Google Colab provides a single 12GB NVIDIA Tesla K80 GPU for free that can be used for up to 12 hours continuously. So finally, I made a transfer from PyCharm to Google Colab, which significantly accelerated deep learning training.

4.3 Experimental results of implemented methods

Following sections provide the results and brief description of implementation of the methods used for pre-processing.

4.3.1 Median Filtering

In this project, the method from the skimage (docs.scipy.org, 2021) library was used, in which the "window" was 5x5. As a result, a processed image had a much smaller noise level, with ensured by adequate edges. The code for performing in can be seen in section 10.3.1. The result of median filter operation is shown in figure 11.

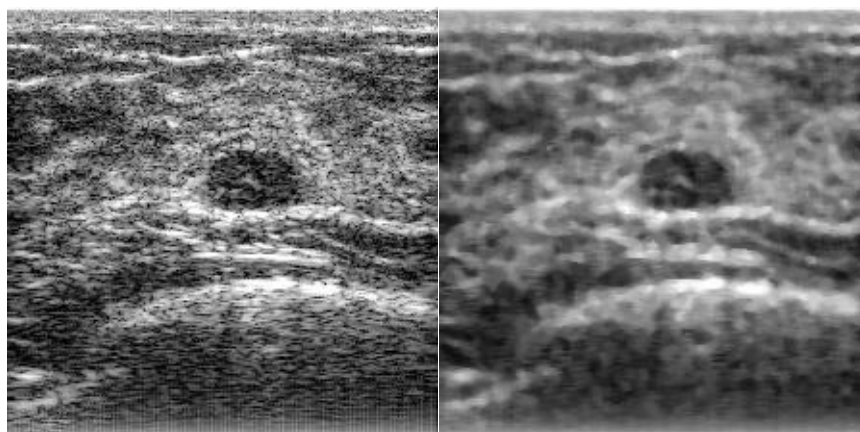


Fig.11 – The effects of median filtering (left image original, right image filtered)

4.3.2 Normalization

As described in experiment methods chapter, normalization is done by subtracting the mean value from each grey level and dividing the results by the standard deviation – as shown in code in section 10.3.2. NumPy packages contains methods for calculating the mean value from image, as well as standard deviation. The normalization methods takes input image and returns normalized image. During data processing on this project, normalization is only used in tandem with Median Filter. The effect of combining these two methods in the example image used to present the effect of median filtering is shown in figure 12.

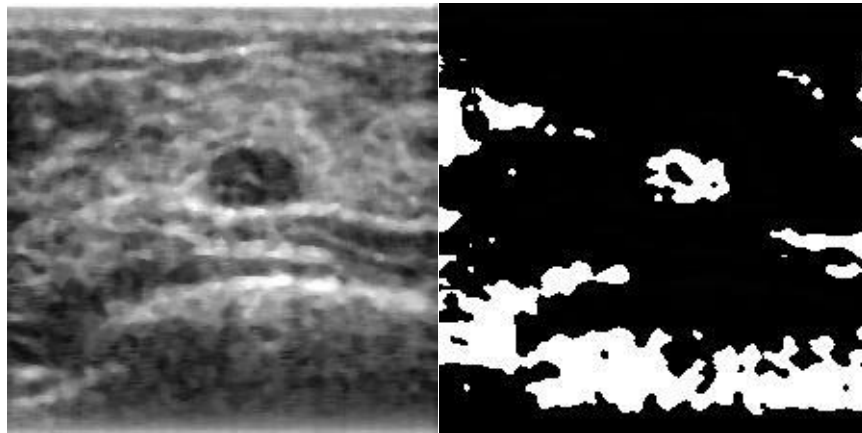


Fig.12 – The effects of normalization on median filtered image (left image with median filter only, right image normalized)

4.3.3 Multi-Otsu Thresholding

Performed method on the example image, preceded by median filtering and normalization, is presented below. The method was implemented using `skimage.filters` library. According to scikit-image documentation, Multi-Otsu generates N number of classes of threshold values. According to the scikit-image documentation, “The threshold values are chosen to maximize the total sum of pairwise variances between the thresholded gray-level classes.” In my project I defined number of classes 5, which gives 4 different values of gray-level. I preceded multi-Otsu thresholding with median filtering and normalization to enhance the action of thresholding as shown in code in section 10.3.3. The method `threshold_multiotsu()` returns array with threshold values that are later digitized using NumPy arrays and its type changed to unsigned format with values [0,255]. The result of this pre-processing is shown in figure 13.

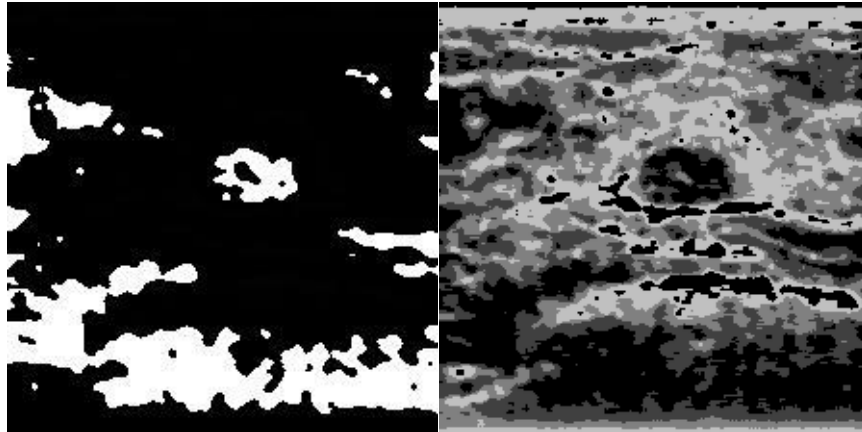


Fig.13 – The effects of multi-Otsu thresholding (left image after median filtering and normalization, right image presents the left image but with additional pre-processing using multi-Otsu thresholding).

4.3.4 Blob Detection (LoG – Laplacian of Gaussian)

The figure 14 presents how blob detection works on the original image using Laplacian of Gaussian and others for comparison.

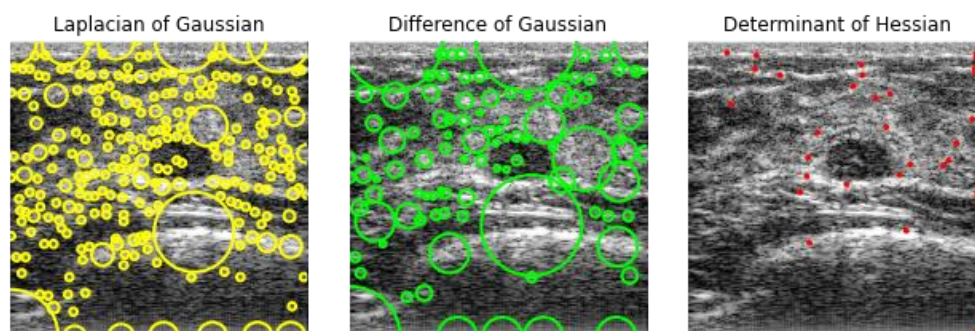


Fig.14 – The effects of blob detection on example BUS image (left image detection using Laplacian of Gaussian, middle image using difference of Gaussian, right image using determinant of Hessian).

For the project, as mentioned, only Laplacian of Gaussian is used on images preceded by inverting colors as blob_log method from skimage library looks for bright blobs – where our abnormalities are darker. The method blob_log from mentioned library returns an array with locations of blobs (x,y) and their seize (sigma). That information is used then for drawing blobs on an empty image of the size of the original image. Figure 14.1 presents the results of performing the steps on original image.

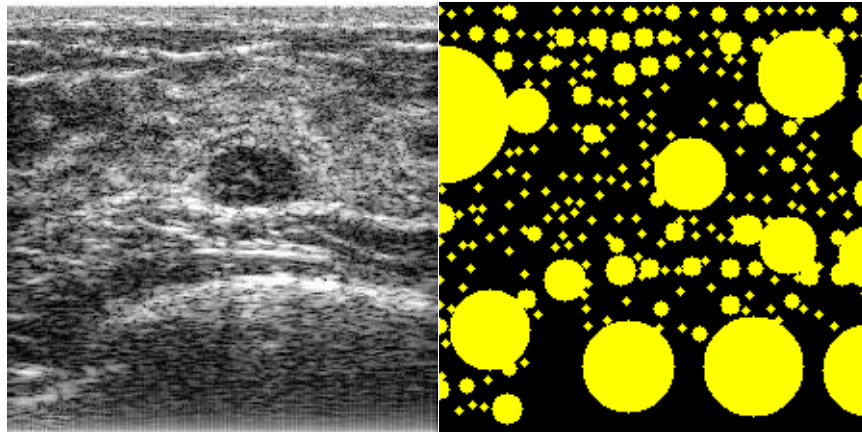


Fig.14.1 – The effects of blob detection using Laplacian of Gaussian (on left original image, on right created image with drawn blobs detected on original image).

The implemented code for blob detection can be found in section 10.3.4. First it loads image, resize, inverts color as `blob_log` method from `skimage` looks for white blobs and breast abnormalities are indicated on images with black color. The image is converted into gray image and `blob_log` method performed. Then, the radius for sigma is computed and saved in 3rd column of `blobs_log` variable. The new, empty, black image, where the blobs will be drawn. The for loop changes coordinates type of blob center and draws a circle with specified sigma's size (`each[2]`) that is converted into Integer type.

4.3.5 Canny-Edge Detection

OpenCV library contains method `cv.Canny()` that does all mathematic equations. It requires minimum 3 parameters: first is an image, second and third are `minVal` and `maxVal` respectively of Hysteresis thresholding. After some of experiments I decided to use `minVal = 100` and `maxVal = 200`, because higher `minValue` miss some of important edges and lower `minVal` draws some useless edges (no abnormality, noise edges). Similar consequences of changing the `maxVal`, and empiric experiment lead me to that conclusion and these final parameters. The implemented code is easy to understand, however I perform Canny-Edge detection on pre-processed images with Median filter and normalization. As normalization changes the type of image, that `cv.Canny()` cannot preprocess, in my implementation the image type is changed to `uint8` and then edge detection is performed. The three methods are then combined into one, "preprocessing" method can be found in section 10.3.5. The result of performing the canny edge detection preceded with median filtering and normalization shows figure 15.

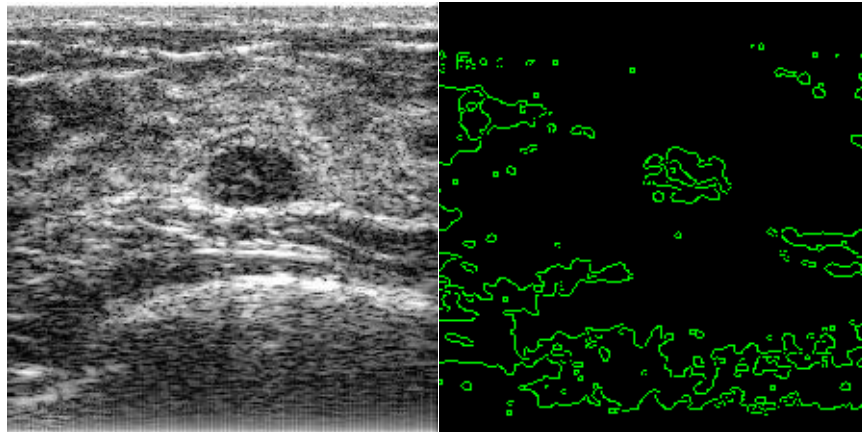


Fig.15 – The effects of canny-edge detection using OpenCV (on left original image, on right image with applied median filter, normalization and canny-edge detection at the ..

4.3.6 3-Channel Images

Combining 3 or even 4 preprocessing techniques is not that hard using RGB channels to do so. The example of implemented method “preprocess3channel_COB” (COB – Canny, Original, Blob detection) is an example of that approach and the all the code remains the same for other 3 methods but the step before extracting channels differs with using different processing techniques as mentioned in experiment methods. The method takes input image, creates 3 variables in which it saves preprocessed images. Then extracts red channel from original image, blue channel from canny, and green from blob image. The code for this implementation can be seen in section 10.3.6.

As mentioned in *Experiment Methods* chapter, I created 4 different combinations:

1. Canny-edge detection, Blob detection, Median&Normalization (Fig.16)
2. Canny-edge detection, Original, Blob detection (Fig.17)
3. Canny-edge detection, Multi-Otsu thresholding, Median&Normalization (Fig.18)
4. Median&Normalization, Blob detection, Original (Fig.19)

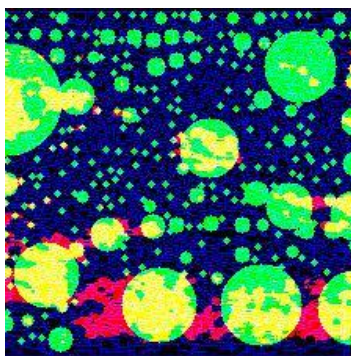


Fig. 16 – 3-channel image combined of images with applied canny-edge detection, blob detection and median&normalization

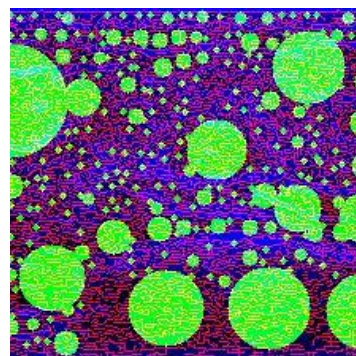


Fig.17 – 3-channel image combined of original image and 2 images with applied canny-edge detection and blob detection

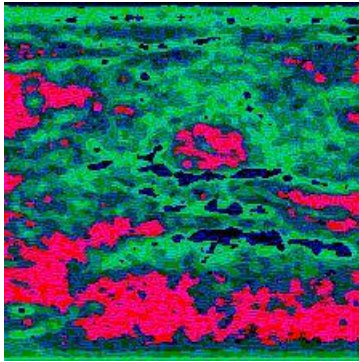


Fig. 18– 3-channel image combined of images with applied canny-edge detection, multi-Otsu thresholding and median&normalization

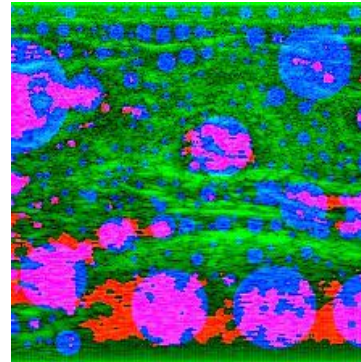


Fig. 19 – 3-channel image combined of original image and 2 images with applied blob detection and median&normalization

4.4 U-Net architecture

Fig. 20 shows original U-Net architecture that comes from original U-net paper (Ronneberger et al. 2015), and the second one shown in Fig. 21 is especially suited for our problem and type of input data. The architecture itself contains two paths. The path on the left is called contraction path (also called encoder path), and path on the right is called expansion path (also called decoder path). In between we can see arrows that indicate that data from left is concatenated with data from the right side that is connected. This concatenation of feature maps helps give localization information and in fact makes the semantic segmentation possible (Ronneberger et al. 2015). U-Net was primarily designed for biomedical segmentation. On image from original paper, we can see that their input for their model has size of $572 \times 572 \times 1$ and they use 64 features, but I modified my model for my needs, so my model takes input of size $224 \times 224 \times 3$ (shape of my original images from BUS dataset) and at the beginning I use 16 features. The blue arrow shown on Fig. 20, which is also in legend in right-bottom corner, indicates convolutional operations that are 3×3 kernel size. Padding equals same which means to add extra pixels on the edges, as when we run 3×3 convolution we miss edges, so the output image's dimensions are same as input image. The next step, indicated by green arrow is Maxpooling step using 2×2 and stride = 2, which in simple words, puts a 2×2 matrix and within that, selects a maximum value and replace the 2×2 matrix by the maximum value. In a result we get dimensions half of the input's feature channels as we use 2×2 kernel, so from $224 \times 224 \times 16$ and two convolutional operators of feature space 32 so it become $112 \times 112 \times 32$. We repeat this process all the way to P4. As we can see the dimensions of P4 are $14 \times 14 \times 128$ and $14 \times 14 \times 256$ of C5. The following steps is called upsampling with very similar parameters (2×2). The C4 ($28 \times 28 \times 128$) is concatenated with U6 ($28 \times 28 \times 128$) and the result is $28 \times 28 \times 256$. After it goes through the couple of convolutional steps, so the result is $28 \times 28 \times 128$. It again upsamples that, concatenates with left side, performs convolutional steps and repeat all the way to the top. The last convolutional layer is with parameters 1, so the final output is $224 \times 224 \times 1$. Between convolutional layers (blue arrows on image) throughout the whole segmentation process, the model dropouts. Dropout means that we randomly select specified number of pixels and just dropout these. The dropout is specified with numbers:

C1 = 0.1, C2 = 0.1, C3 = 0.2, C4 = 0.2, C5 = 0.3, C6 = 0.2, C7 = 0.2, C8 = 0.1, C9 = 0.1. The dropout is used typically to prevent model from overfitting. In code section 10.3.12 can be found the model summary describing that briefly talks about each layer and their parameters, its connections and numbers of total, trainable and non-trainable params.

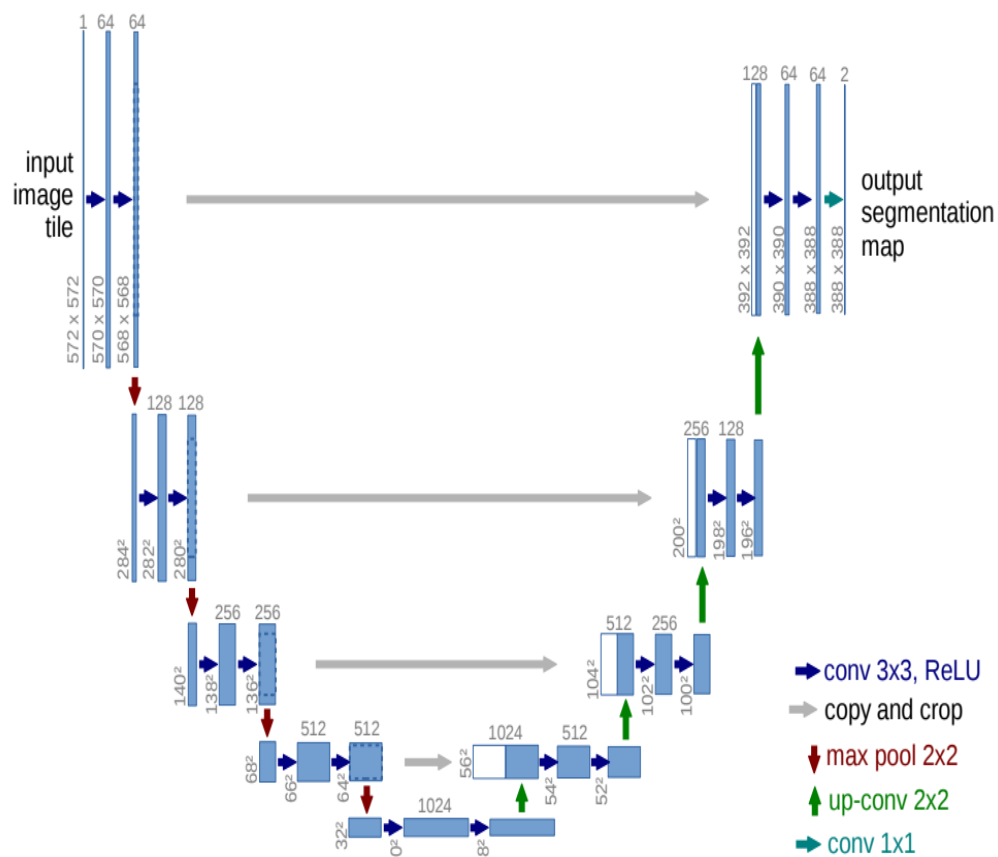


Fig. 20 – Original U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. Source: *U-Net: Convolutional Networks for Biomedical Image Segmentation* <https://arxiv.org/abs/1505.04597>

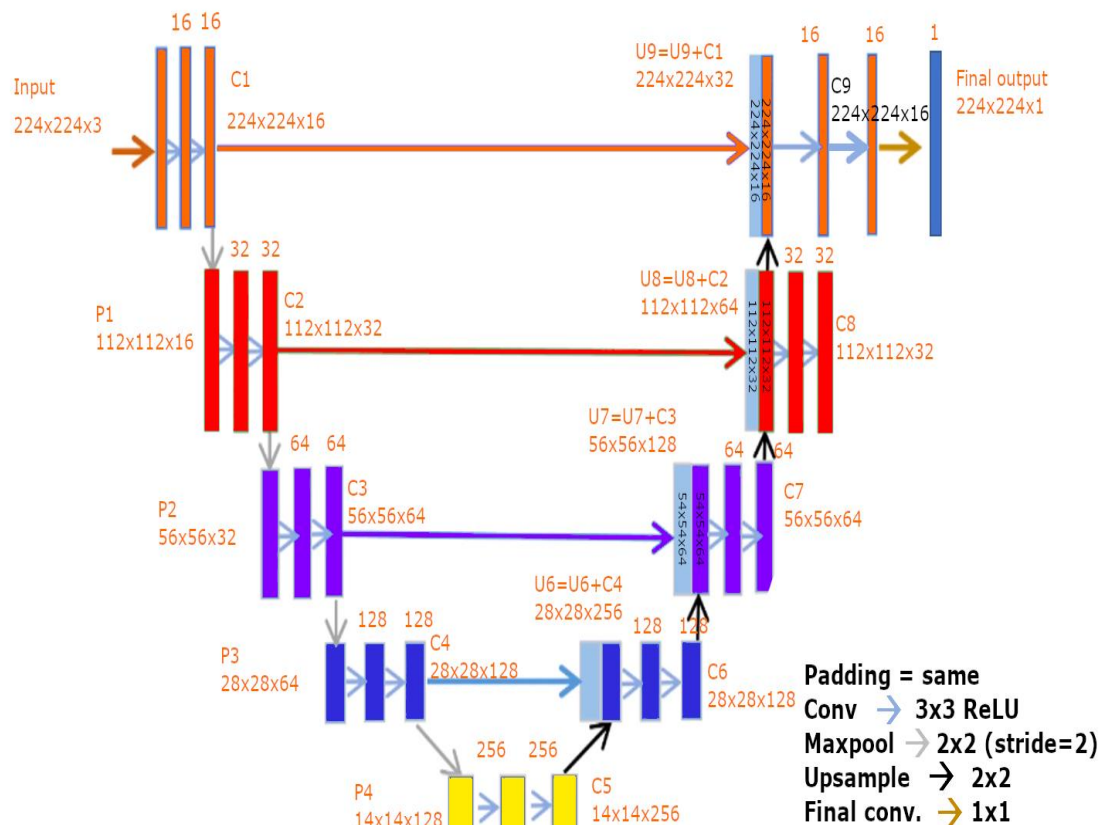


Fig.21 U-net architecture fitted for the project's purposes.

4.4.1 U-Net implementation in Python

To implement project's U-Net model were used TensorFlow and Keras libraries. The input layer takes input, which is an image of specified image width, height, and a number of channels. The first convolutional layer will be a good example shown below to visualize how the implementation looks for every other layer, as the code is similar. However, before implementing C1, the software uses Lambda Keras Layer that converts each input's pixel (int values) to floating points. The C1 layer implementation can be seen in section 10.3.7. "Inputs" indicates input layers, where "s" is the inputs layer but with pixel values converted into floating points. The activation function is a function by which the value of the output of neurons of a neural network is calculated. In my implementation, I use the ReLU activation function that, unlike the also popular Sigmoid function, ReLU reduces the likelihood of the gradient to vanish and provides sparsity that seems to be better than dense representations (Krizhevsky et. Al. 2012).

Additionally, the sigmoid function shows worse convergence performance than ReLU. Kernel initializer is set on 'he_normal', which defines the initial values that are updated while training. As mentioned, 'Padding' equals the same, which means to add extra pixels on the edges, so the output image's dimensions are the same as the input image. In the code snippet (10.3.8) is also defined dropout function and maxpooling. Using the same structures in implementing encoding layers is basically the same but with different feature values. Decoding layers looks almost the same as encoding with the difference of lines of code, where we perform concatenation as shown in the code snippet in section

10.3.8.Conv2DTranspose is an opposite function to Conv2D. Conv2DTranspose is upsampling, where Conv2D is downsampling. As we perform semantic segmentation or so-called binary segmentation, the output layer has kernel of size 1x1 and outputs the image with 1 channel thus the last line of implementation of U-Net architecture has such parameters as shown in section 10.3.9. The activation function is “sigmoid” in this case, as later this value is taken and used in binary_crossentropy loss. This combination only works for binary segmentation.

Model is finally defined by tf.keras.Model method that groups layers into an object with training and inference features (tensorflow.org, 2021). In model.compile as shown in section 10.3.10 it was defined ‘adam’ as optimizer which is the most common optimizer, and since this is binary classification that we will work with – either it is a lesion or not, ‘binary_crossentropy’ was used, which is efficient loss function. Optimizer does nothing more than trying to minimize the loss function. Binary Cross-Entropy formula is:

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i * \log(p(y_i)) + (1 - y_i) * \log(1 - p(y_i))$$

Where y is the label (1 for lesion and 0 for not lesion) and $p(y)$ is the predicted probability of the point being lesion for all N points.

4.4.2 U-Net Training

For training purposes, the accuracy is set as a parameter defining metrics, which is the percentage of correct predictions. The results are from calculations of dividing the correct predictions by the total number of predictions. After that implementation we get total number of trainable parameters equals 1,941,105. Additionally, the implemented callbacks force early stopping when results are not improving in a period of 10 epochs (called patience) based on monitored ‘val_loss’. The callbacks contain check pointer that keeps the model with best trained accuracy and allows TensorBoard for further analysis in that tool. The code for implemented checkpoints are shown in section 10.3.11. Finally, the last line of code for U-Net implementation performs the actual model training. The method fit() takes the parameters such as training images with corresponding masks, disjunction values for validation which is set to 0.2 (80% training images, 20% testing). The code for model fitting is shown in section 10.3.12, where the number of epochs is set on 35 with batch size of 16, and callbacks are the one we implemented before. More details from testing could be found in chapter 6 that describes results – including training results of how the training did go for every dataset, how many epochs it managed to run, and other.

5 Testing

Testing for this project was complicated issue as due to its nature as research project – not a strictly software engineering project.

5.1 Overall Approach to Testing

It was hard to define the good testing strategy. Using the Google Colaboratory that has the form of notebook/Jupyter. It allows to test the methods by executing specified cells of code. It simplifies the debugging and significantly speed up the process of implementation,

as it does not require to load all code. For example, after median filtering was implemented, it was applied on loaded image. The resulted image was printed, so when something went wrong, the Colab pointed it out and the bug was fixed. The methods, the data loading, the U-Net model, and others, everything can be executed independently. This was treated as a big advantage of the Google Colab as it minimized the number of tests. The software does not have any unit or automated tests and does not use any testing software. The idea behind that project is to draw a conclusion about the impact of pre-processing, so the software is not made for user, thus it does not have user interface testing.

5.2 Functionality testing

The project testing strategy was however highly based on functionality testing. The output of performing methods mentioned in section 4.3 on images could be predicted in some way thanks to the adequate, preceded research. After implementation of code for the specific methods, the Google Colab allowed to run chosen cell, so the method was testing based on black-box technique – the image was used as input and the generated output was analysed in terms of required behaviour. This approach seemed the most suitable for this research project that was highly focused on data preparation and analysis rather than creating complicated software made of multiple classes and objects that required testing. The tests were based on the three main software's goal. Firstly, the project was testing the output of pre-processing methods. Secondly, the U-Net model segmentation results were tested in terms of getting the output image of same size and segmented region. The third tests were focus on dice coefficient results where they had to meet the specific conditions, such as absolute value and be lower than zero.

5.3 Stress testing

The results from this form of testing were highly dependent on the resources provided from Google Colab, like GPU or RAM when it comes to hardware. The tests achieved different results as the free version of Google Colab requires some level of attention as Google tries to share its resources evenly so when user is not using from the Colab but is connected – it very often disconnects the user. However, the software did not show any signs of problems with loading large dataset (more than 3300 images) or to perform complicated operations like Multi-Otsu thresholding on more than 1100 images which took sometimes from two, up to five hours.

6 Results

Pre-processing of original dataset using methods described in experiment method chapter 2 resulted in having eight datasets, divided into train (1079) and test (75 images) sets:

1. Median filter & normalization (Fig.12)
2. Canny-edge detection (Fig.15)
3. Blob detection using Laplacian of Gaussian (Fig.14.1)

4. Multi-Otsu thresholding (Fig. 13)
5. Four different datasets of 3-channel pre-processed data:
 - A. Canny-edge detection, Blob detection, Median&Normalization (Fig.16)
 - B. Canny-edge detection, Original, Blob detection (Fig.17)
 - C. Canny-edge detection, Multi-Otsu thresholding, Median&Normalization (Fig.18)
 - D. Median&Normalization, Blob detection, Original (Fig.19)

The images have been used as input for U-Net model described in section 4.4 in training along with corresponding masks – same for each dataset. For every dataset, the following steps were made:

1. The U-Net model was trained on train dataset, achieving different accuracy and loss values while training.
2. The trained model was then used for prediction on the test dataset.
3. Predicted images were saved.

DATASET	ORIGINAL	1	2	3	4	A	B	C	D
VAL_ ACCURACY	0.9345	0.9489	0.9285	0.9099	0.9489	0.9131	0.9434	0.9455	0.9367
VAL_LOSS	0.1893	0.1418	0.1910	0.2347	0.1418	0.2258	0.1488	0.1532	0.1851
LOSS	0.0908	0.1490	0.1770	0.1294	0.1490	0.1130	0.1136	0.1194	0.1752
ACCURACY	0.9672	0.9459	0.9352	0.9498	0.9459	0.9589	0.9582	0.9556	0.9350
NUM OF EPOCHS	23	31	22	27	16	29	27	22	12

Table 1

Table shows validation metrics achieved while training model on the given datasets. 'Val_accuracy' and 'val_loss' indicate loss value and accuracy achieved on validation set, while 'loss' and 'accuracy' indicate the metrics achieved on training set. 'Num of epochs' specifies the number of epochs model had run with patience set on 10 as mentioned in section 4.4.2. and stopped if model was not improved in a period of 10 epochs.

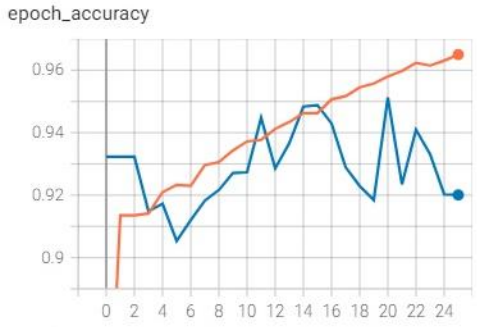
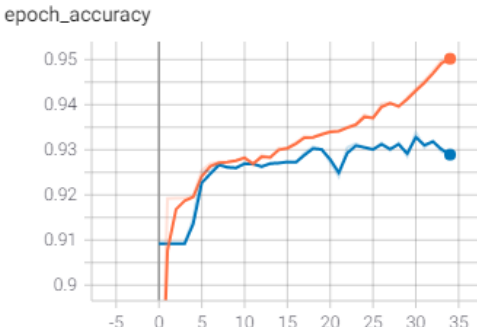
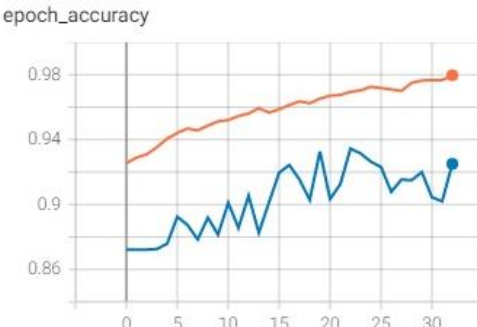

Multi-Otsu thresholding (preceded by Median filtering and Normalization)	Median and Normalization
	
Original	Blob detection
	

Table 2
Table shows visualization graphs from TensorBoard of model training using example datasets. As mentioned before, one of the model’s callbacks had a parameter patience set on 10, which means that model stops running if there was not any improvement in a period of 10 epochs from the last improvement. The model was trained and saved using val_loss indicated with a blue line on images, and orange line indicates the val_accuracy.

The training of the model was highly influent by pre-processed dataset. By looking at table 2, it can be noticed that the accuracy line (orange color) of Multi-Otsu thresholding rises very similarly to the Median and Normalization line, as both pre-processing approaches contain the same methods – median and normalization. To reduce overfitting, the patience was set on 10, so by looking at graphs and Table 1, it can be seen when the model improved and saved based on val_loss drops. Even though accuracy was rising, the model did not save changes since val_loss was rising along.

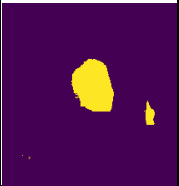




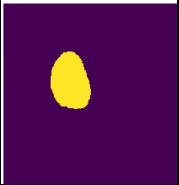




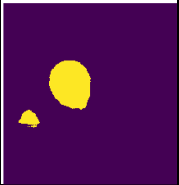

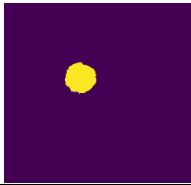







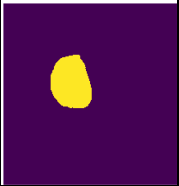

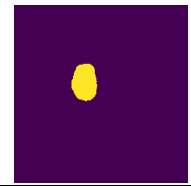


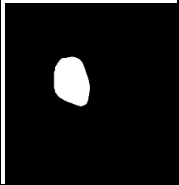



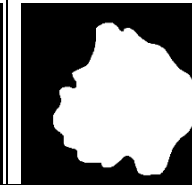
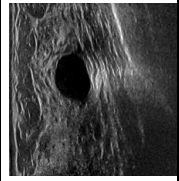
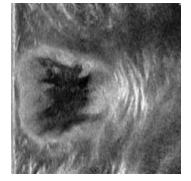
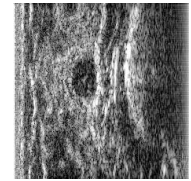
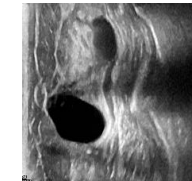
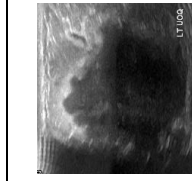
Canny-edge detection					
Multi-Otsu					
Blob detection					
Median and normalization					
Original					
Mask					
Image					
Dataset	BUS_UDIAT	RODTOOK	OASBUD	BUSI	BUSI

Table 3

Segmentation results obtained by the U-Net using indicated pre-processing techniques in comparison to original image and corresponding masks. The “original” column shows segmented images using original dataset without any pre-processing. Dataset column inform of images’ origin.

Median and Normalization, Blob Detection, original					
Canny, Multi-Otsu, median and normalization					
Canny, original, Blob detection					
Canny, Median and Normalization, Blob detection					
Original					
Mask					
Image					
Dataset	1.BUS_UDIAT	2.RODTOK	3.OASBUD	4.BUSI	5.BUSI

Table 4

Segmentation results obtained by the U-Net using indicated pre-processing techniques in comparison to original image and corresponding masks. This table contains segmented images using 3-channel processing techniques in comparison to the segmented images using original dataset only.

image	Original	Median and normalization	Blob detection	Multi-Otsu thresholding	Canny-edge detection
1	0.34	0.89	0.32	0.93	0.32
2	0.36	0.49	0.41	0.57	0.53
3	0.93	0.00	0.76	0.90	0.71
4	0	0.38	0.15	0.82	0.12
5	0.58	0.68	0.74	0.84	0.75

Table 5

Table shows results from calculating dice coefficient of images presented in **Table 3**. The dice similarity coefficient was the highest for Multi-Otsu thresholding (preceded by Median and Normalization). Segmented image no. 4 (BUSI dataset) using original dataset obtained dice = 0 because the segmented abnormality was far to right, so there was no area of overlap between image and mask.

image	Original	Canny, Median and Normalization, Blob detection	Canny, original, Blob detection	Canny, Multi-Otsu, median and normalization	Median and Normalization, Blob Detection, original
1	0.34	0.45	0	0.54	0.31
2	0.36	0.43	0.33	0.25	0.42
3	0.93	0.84	0.00	0.10	0.79
4	0	0.17	0.02	0.28	0.12
5	0.10	0.69	0.14	0.64	0.10

Table 6

Table shows results from calculating dice coefficient of images presented in **Table 4**. The dice similarity coefficient was the highest for combining Canny edge detection, Median and Normalization, and Blob detection. Although it can be hardly seen, on image 1 from COB dataset (Canny-edge detection, Original, Blob detection) there is a segmented lesion, but it is very small, and does not overlap with corresponding mask, thus the result equals 0.

When investigating the pre-processed results, it was noticed that some of the predictions (segmented images) were empty. In results from segmenting original images, three images are empty – all from the BUSI dataset, each of them of malignant type. The other segmented datasets were also struggling to segment those images. The original images that the model struggled with are presented in figure 22. The dice coefficient was 0.00 when the model was trying to predict another image from the BUS_UDIAT dataset for every single pre-processing technique except Multi-Otsu thresholding (preceded by Median filtering and Normalization) where it achieved dice = 0.8.

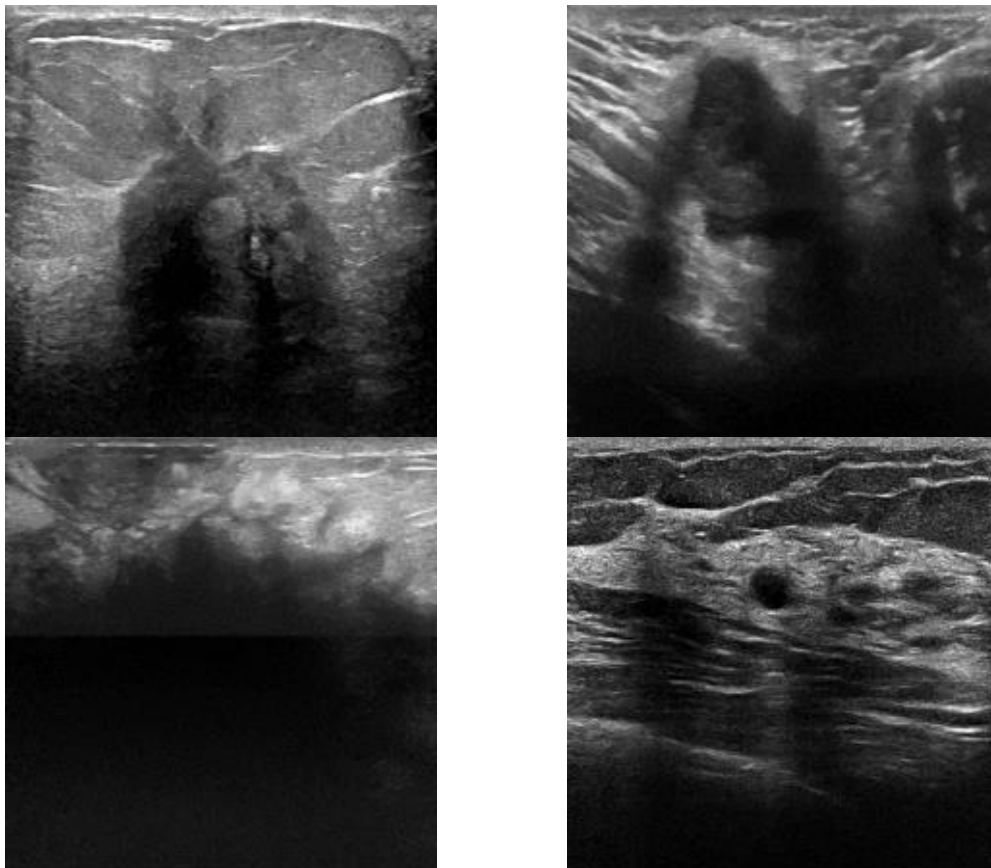


Fig.22. Top-left, Top-right and bottom-left images are originally from BUSI dataset. The bottom-right image is from BUSI_UDIAT dataset. The segmentation model, regardless the pre-processing technique used on original data, have done poorly for the images.

The dice coefficient average was calculated for a dataset with segmented images using the dice coefficient calculated for each image included in the datasets and presented in Table 6. For 75 images, the average dice coefficient was calculated, including the values equal to 0, where the segmentation model did extremely poorly. For comparison, the results from GAC segmentation on pre-processed images (Median filtering and Normalization) are presented next to results achieved from segmentation using original data for training and testing, as these results determine the baseline. Naturally, the segmentation results from original data are more reliable, and so the baseline is. This step was crucial for the project as the results and comparison were fundamental for the future conclusion and verification or falsification of the project's hypothesis. In table 6, the results included there come from the following dataset of segmented images:

1. Original
2. Median filter & normalization
3. Canny-edge detection
4. Blob detection using Laplacian of Gaussian
5. Multi-Otsu thresholding
6. Canny-edge detection, Blob detection, Median&Normalization
7. Canny-edge detection, Original, Blob detection
8. Canny-edge detection, Multi-Otsu thresholding, Median&Normalization
9. Median&Normalization, Blob detection, Original

Segmented images (test data)	Average of Dice coefficient
GAC (230 evolution) segmentation on images (pre-processed using Median filtering and normalization)	0.22
Original	0.44
Median filtering and normalization	0.34
Canny-edge detection	0.43
Multi-Otsu thresholding (preceded by Median filtering and normalization)	0.70
Blob detection using Laplacian of Gaussian	0.36
Canny-edge detection, Blob detection, Median and Normalization	0.41
Canny-edge detection, Original, Blob detection	0.33
Canny-edge detection, Multi-Otsu thresholding, Median and normalization	0.17
Median and normalization, Blob detection, Original	0.42

Table 6

Results achieved by calculating the average of dice coefficients from every image in segmented datasets.

As shown in Table 6, segmentation results from every pre-processing technique used in this project achieved worse results than segmentation results from the original dataset used for training. The only exception is the dataset pre-processed using Multi-Otsu thresholding, preceded by performing median filtering and normalization, as its average dice coefficient was 0.70, which is 0.26 better than results from segmenting original images. Although the first assumptions about how the 3-channel combination could be efficient thanks to providing more information that the model considers, image pre-processing using specified techniques in combination is not sufficient for image segmentation on ultrasound images. The worst results were achieved by COTM (Canny-edge detection, Multi-Otsu thresholding, Median and normalization), that did significantly worse than other techniques, as 15 out of 75 test images got 0.00 dice coefficient, and another 16 images got dice coefficient less than 0.10. The COTB results were even worse than GAC segmentation on images pre-processed using Median filtering and normalization for about 0.05.

Similar results gained Blob detection using LoG and 3-channel images of COB (Canny-edge detection, Original, Blob detection): 0.33 and 36. Both pre-processing techniques were using Blob detection that is fundamental behind the similarity in results. Even though they were in saved different channels, the combination of canny-edge detection and blob detection was doing unfavourably. The original dataset achieved an average dice coefficient of 0.44, but with the additional pre-processing, it gained worse results. A similar situation is with COTM processing that achieved 0.17, although one of the channels contained the image of Multi-Otsu threshold image that alone achieved much better results than other datasets.

The result of the segmentation highly depends on image quality. Some of the pre-processing techniques, although that in general did extremely poor, it achieved better result than for example Multi-Otsu thresholding. The reason is that the images come from different datasets, have different quality and different among of noise. Types of pictured breast cancers vary, and some are more suitable for specific pre-processing, some are less. The following section tries to draw the difference between the achieved results between the pre-processing approaches.

6.1 Pre-processing comparison

For the purpose of the following section, we will assume that the “significance” rate equals 0.05, which means that every time one model or result is “significantly better” it means that it achieved better result for more than 5% (usually 0.05 of dice coefficient). The section reveals the differences in results of segmented images, highlighting the advantages and disadvantages of specific pre-processing techniques.

6.1.1 Median filter and normalization results

Median filtering and normalization achieved results worse original dataset for about 0.10 of average dice coefficient. Most of the images from this dataset were very similar to segmented images from original dataset, and the model in general was doing almost the same, and due to this fact, it is hard to spot the difference and the pattern of when this pre-processing approach did better than the baseline. Significant difference can be seen on figure 23 of malignant breast cancer. The pre-processing significantly improved result for 0.69 of dice coefficient. However, as shown in Table 6, in general the pre-processing did better than the baseline and 4 out of 75 segmented images are empty and in figure 24 can be seen images that segmented images using original dataset achieved better results for 0.5 or more of dice coefficient.



Fig. 23 The first image (from left) is the original image of malignant. The next image is corresponding mask, third image is result of segmenting image pre-processed using median&normalization. The last image is segmented image using original dataset.

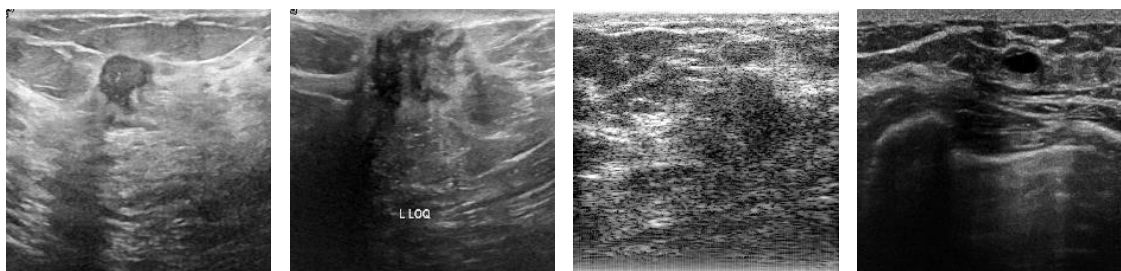


Fig. 24 The first two images (from left) are the original image of malignant cancers (BUSI dataset). The next, highly noisy image comes from OASBUD dataset. The last image (on right) comes from BUS_UDIAT dataset.

6.1.2 Multi-Otsu thresholding results

Multi-Otsu thresholding combined with preceded median filtering and normalization achieved the best results as shown in Table 6. Mostly, the model trained on that dataset performed accurate segmentation where other techniques failed. For example, four images shown in figure 25 were segmented with accuracy of ~ 0.80 (dice coefficient), while for other segmentations the results were less than 0.05 (mostly 0.00). However, there were some images (Fig. 26) where median and normalization itself achieved significantly better results, than in combination with Multi-Otsu thresholding. In short, although model trained and tested on dataset preceded by pre-processing using this technique achieved the best results, this method still got some weak points, and sometimes even Multi-Otsu thresholding have bad influence on the images in order to perform accurate segmentation.

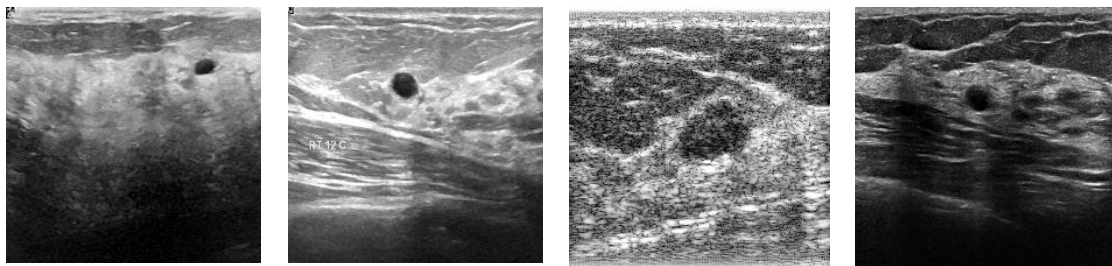


Fig.25

The figure shows example images where Multi-Otsu thresholding combined with median filtering and normalization improved the results for ~ 0.80 (dice coefficient). The images come from 3 originally different datasets – first two images (from left) are benign type of cancer, the next two are unknown type and come from OASBUD (middle-right) and BUSI_UDIAT datasets.

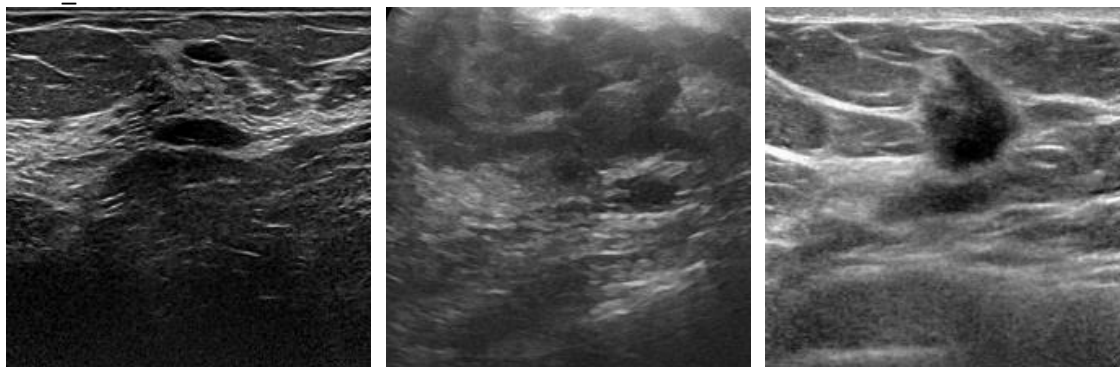


Fig. 26

The figure shows images where median filtering and normalization achieved better results than in combination with Multi-Otsu thresholding. Each comes from originally different datasets and the middle image is known type cancer – malignant.

6.1.3 Canny-edge detection results

As shown in Table 6, the Canny-edge detection as pre-processing method achieved poor results – 0.43, worse but not significantly than the baseline. It did better on some images than original, even then multi-Otsu approach as shown in figure 24. The reason that canny did better than others is that, as shown on original image in Fig. 27, at first glance there can be found two lesions, but in fact the corresponding mask contains only one lesion. The model trained using canny-edge dataset focused more on the smaller, top lesion while the other models focused mainly on the bigger, lower lesion that does not exist in corresponding mask.



Fig. 27 The first image (from left) is the original image from BUSI_UDIAT dataset. The next image is corresponding mask, third image is result of segmenting image pre-processed using Canny-edge detection. The last image is segmented image using original dataset.

6.1.4 Blob detection (LoG) results

The average dice coefficient that pre-processing technique blob detection using Laplacian of Gaussian achieved equals 0.36, which is worse than the baseline. In all segmented images using this approach only one image (Fig.28) has significantly better dice coefficient than any other technique (except CMB combination with Blob detection included), of malignant type cancer. 18 results out of 75 are equal 0.00 or less than 0.10. The highest dice coefficient (0.86) achieved on image (Fig. 28) from RODTOOK dataset, however other techniques got similar or better results.

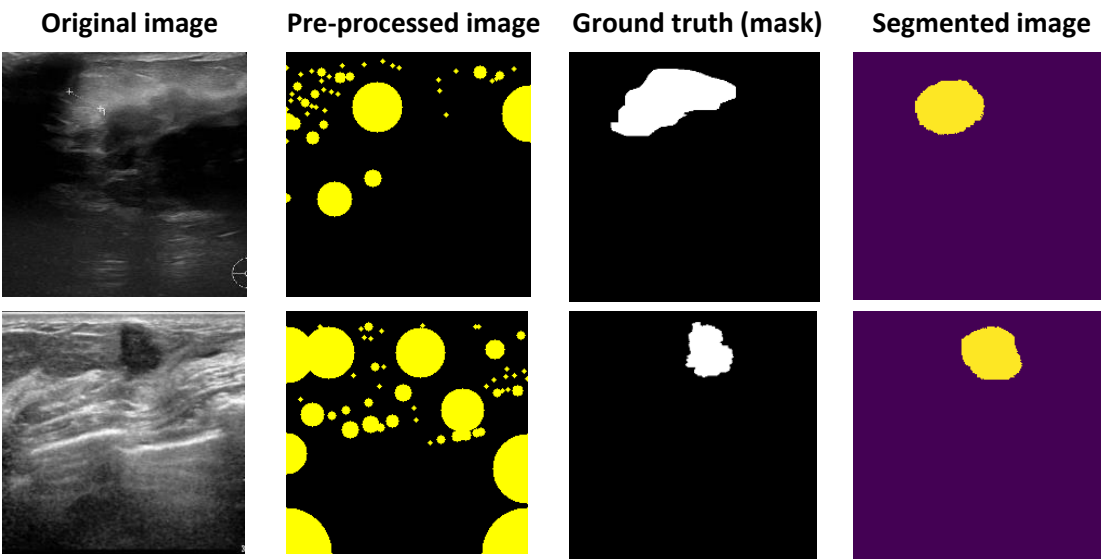


Fig.28 The figure shows (first row) the only image (malignant cancer), where segmentation of pre-processed image using blob detection achieved significantly better results than other techniques. The second row shows original BUS image, pre-processed image (middle-left), corresponding mask and segmented image, where blob detection achieved the highest accuracy = 0.86 (dice coefficient).

6.1.5 3-channel combinations of images

Before approaching to combination of 3-channel pre-processing, the project assumed that this approach would perform significantly better as it focuses multiple techniques, and so more information that U-Net model can take advantage of. As shown in Table 6 the assumption was completely wrong and none of the 3-channel combination improved the segmentation results. Combinations that contained pre-processing techniques that achieved average results (median and normalization or canny-edge detection) or even

COTM that contained channel of Multi-Otsu pre-processed image achieved worse results than the combined techniques alone.

6.1.5.1 COTM

The COTM (Canny-OTsu-Median-normalization) achieved the worst results (0.17 average dice coefficient) among all pre-processing methods, and even worse than GAC segmentation. This is caused by combining the Multi-Otsu thresholding and Canny-edge detection on one image, where in results that combination significantly decreased the accuracy in segmentation. The results have not exposed any advantage among other techniques that would point the COTM pre-processing as a good and suitable technique for preparing data for image segmentation.

6.1.5.2 MBO

The MBO (Median and normalization, Blob detection, Original) achieved 0.42 that was not significantly worse than the baseline and did the best among the 3-channels combinations pre-processing. Again, the segmentation results of the data were dictated by the quality of the segmented image – when the segmentation on original image was accurate – then MBO did similarly accurate as it contains the original image in one of the channels. The pattern can be seen when we look closer at the dice coefficient results – when the model trained on that pre-processed dataset did good – it usually means that slightly (sometimes even significantly) better results were achieved by model trained on original data only.

6.1.5.3 COB

The COB (Canny-Original-Blob detection) achieved average dice coefficient equal to 0.33, which is worse than original data for about 0.11. The similar pattern as in MBO can be spot in this technique; however, in MBO, the original channel had bigger impact on the final segmentation. In case of COB, even though the each of combined pre-processing method achieved more than 0.7 dice coefficient alone, when combined, the result was 0 – as shown in Table 4 on image no. 3 from dataset of OASBUD origin. The combination of Canny-edge detection and Blob detection has bad impact on BUS image segmentation using U-Net.

6.1.5.4 CBM

The CBM (Canny-edge detection, Blob detection, Median and normalization) worse average dice coefficient result than the baseline, yet the difference was not significant. The pattern from two previous discussed pre-processing approach seems to be not present when it comes to CBM. The dice coefficient results show that even though results from Median and Normalization dataset were extremely inaccurate (less than 0.05), the segmented images from CBM dataset achieved sometimes better results than the pre-processing techniques separately. The results seem to be predicted by the results of techniques separately, but all taken into consideration, which lead to very unstable accuracy – if each of the pre-processing technique did significantly better than original dataset then it was more likely that the result was significantly better than original dataset and sometimes better than the each of the method separately. However, this pattern works in both ways which in fact very often results in decreasing the result when one of the pre-processing techniques achieved bad result alone but the other two were very accurate.

7 Conclusions

Image segmentation in medical research still requires constant improvement. Pre-processing of data is crucial in a deep learning pipeline, especially when it comes to image segmentation using Convolutional Neural Networks. Currently, CAD systems do cancer detection/classification/segmentation mainly rely on mammographic images, and ultrasound screening is done most often for a complimentary screening. When detecting invasive cancer in dense breasts, ultrasound screening may be more sensitive than mammographic screening (Costanitini et al. 2006, Drukker K. et al. 2002). In the sheer volume of research using mammographic images, this project focused on a less crowded topic, increasing the benefits of ultrasound screening using modern tools. The project used the original dataset to create nine other sets, consisting of pre-processed in multiple ways images. The hypothesis assumed that pre-processing could improve deep learning models for BUS data. The results presented in chapter 6 verified that assumption regarding specific methods; however, it also exposed other methods that can worsen the accuracy of the U-Net model. The results showed that the only dataset used to train the model influenced its performance in a way that was able to segment 26% better than the model trained on the original dataset without any pre-processing interference, was the pre-processing method combined with Median filtering and normalization with additional Multi-Otsu thresholding. Any other pre-processing method used in this project could not achieve better results than the established baseline - that is, the dataset without prior pre-processing. However, as shown in Chapter 6, the project does not show the binary conclusions. Many factors influenced accuracy achieved in the process of image segmentation. Such factors were the quality of the photo, origin dataset, type of cancer, amount of noise, or a number of abnormalities on the image. The results indicated that each method had big problems with the photos of malignant type cancer, and the results of the methods used significantly varied depending on the image. Some of the methods, such as Canny-Edge detection or Blob detection using Laplacian of Gaussian, did better with the segmentation of problematic malignant cancers than Multi-Otsu supported by median filtering and Normalization.

The results indicate that, in general, the use of Multi-Otsu, Median filtering, and Normalization will statistically perform segmentation with an accuracy of 70%, but in practice, it may prove to be better or much worse. Image segmentation is strongly influenced by the quality of data, while the quality of data depends on the person doing the screening (Sassaroli et al. 2019 and Chauhan et. al. 2010). Combinations of pre-processing methods using 3-channel images were a bad idea in this project, as the previous chapter shows. Usually, the selected methods in combination had a negative effect on each other, producing more noise than the original photo had. In summary, potentially successful segmentation models should be trained on pre-processed data using Median filtering, normalization, and Multi-Otsu thresholding. Thus, the other described and observed techniques might be more suitable for some images. The noise in these photos is a factor that should condition the selection of pre-processing methods. When it comes to ultrasound screening and its potential, there is still much room for improvement, and the advancing technology of deep learning can accelerate research in this area. The main reason for the low accuracy in segmentation of the images is the high range of quality of images and the way of taking these images. Indeed, the homogeneity of the images, improved quality, and greater access to them would improve image segmentation and cancer detection and classification.

Although the project obtained improved results when using a specific pre-processing method, there is still a significant factor of inaccuracy, which can cost a lot in the medical area, so the research work in this direction must be continued.

8 Critical Evaluation

When planning the requirements at the beginning of this project, I was not aware of the amount of information that I had to process during its creation and the work that I had to do to achieve most of the goals. Lack of experience in computer vision and designing deep learning software resulted in planning too many tasks, which as a result, I did not manage to perform, such as classifications of the segmented lesion. However, the project's main goal was to measure the impact of specific techniques of pre-processing data in the form of images.

The initial design decision changed during the project due to its nature and requirements. The initial design assumed the creation of several classes in the PyCharm environment. The changes came when designing a deep learning network (CNN), which requires appropriate GPU resources and a setup environment, in which libraries created for machine learning (TensorFlow, Keras) will work with the Python version without any problems. Initial attempts of setting up the environment on windows failed after many tries, so it fell on the Linux system. However, when it came to training the model and previously pre-processing data in large amounts using various techniques, I decided to switch to the Google Colaboratory, which by its form, dictated the procedural programming design, which, due to the nature of the project, was very efficient. Additionally, this environment is set up for machine learning projects and provides a free GPU that, unfortunately, my laptop does not have.

I believe that developing the software and achieving the project goals using the iterative approach was perfect, and together with the weekly meetings with my supervisor Reyer Zwiggerlaar, helped me a lot in completing this project, and it worked well together. Project completion, however, during its inception, several external and unrelated factors contributed to its delay. As a result of the delay, the spare time was almost used up and the project was completed very close to the deadline.

The software made for this project does not fully satisfy me. Due to the lack of experience in creating a research project, the software, although relatively simple to understand in its form, lacks full reusability and flexibility in transferring it to a different environment than Google Colab. The software itself would need some effort to extend the functionality; however, I believe that the program's simple design would help achieve this. The software does not have a good class structure, and as this seems new to me - it also makes me feel insecure about this.

When it comes to the aims achieved, I need to say that I am satisfied with the progress and achievements in general. The main aim of this project was to measure the impact of pre-processing on image segmentation using deep learning network, and that was completed. The original dataset was extended to 9 different datasets with various images that made the results more comprehensive, which led to a reliable conclusion based on the performed experiments. The U-Net model did not achieve high accuracy in image segmentation; however, in my opinion, that fact does not determine the credibility of the project's results as the aim was not to create the best possible BUS image segmentation software but to measure the impact of pre-processing on the final segmentation. I sincerely believe that this project's conclusions could help develop CAD systems for segmenting / detecting/classification of breast cancer or creating a deep learning network pipeline for medical purposes using ultrasound images. If I were starting again, I would work in a more organized way, as it happened that I lost or overwrite some data or results because of my lack of skills in creating a research project. Additionally, I would take care of the appropriate technical facilities that would allow me to work on a local machine while limiting the use of tools that force the user to a specific design, style, or pace of work. I would put more work into proper research that would allow me to define the right amount and clearly defined

goals, rather than assuming more than I can do in a timely manner. Of course, most of the goals were achieved anyway, but as I mentioned, the plan also included classification, a very first draw of aims contained a user interface that would allow to choose the filters and perform segmentation on the chosen image. I would choose a few different pre-processing methods that would focus on eliminating the noise and dealing with poor quality while instead of focusing primarily on edge or blob detection. Using more than one deep learning architecture could be valuable insight for the conclusion and attention to tune the network's parameters to achieve higher accuracy.

Nevertheless, the software and the results seem to be a good foundation for development research in that area. In the future, I will keep working on this project and try to extend its functionality to the classification and detection of breast cancer and improve U-Net's results.

9 Bibliography

1. American Cancer Society. (2019). Breast cancer facts & figures 2019–2020. *Am. Cancer Soc*, 1-44.
<https://www.cancer.org/content/dam/cancer-org/research/cancer-facts-and-statistics/breast-cancer-facts-and-figures/breast-cancer-facts-and-figures-2019-2020.pdf>
2. Berg, W. A., Zhang, Z., Lehrer, D., Jong, R. A., Pisano, E. D., Barr, R. G., ... & ACRIN 6666 Investigators. (2012). Detection of breast cancer with addition of annual screening ultrasound or a single screening MRI to mammography in women with elevated breast cancer risk. *Jama*, 307(13), 1394-1404.
<https://jamanetwork.com/journals/jama/article-abstract/1148330>
3. Cannistraci, C. V., Montevicchi, F. M., & Alessio, M. (2009). Median-modified Wiener filter provides efficient denoising, preserving spot edge and morphology in 2-DE image processing. *Proteomics*, 9(21), 4908-4919.
<https://analyticalsciencejournals.onlinelibrary.wiley.com/doi/abs/10.1002/pmic.200800538>
4. Costantini, M., Belli, P., Lombardi, R., Franceschini, G., Mulè, A., & Bonomo, L. (2006). Characterization of solid breast masses: use of the sonographic breast imaging reporting and data system lexicon. *Journal of ultrasound in medicine*, 25(5), 649-659.
<https://onlinelibrary.wiley.com/doi/abs/10.7863/jum.2006.25.5.649>
5. Dim, J. R., & Takamura, T. (2013). Alternative approach for satellite cloud classification: edge gradient application. *Advances in Meteorology*, 2013.
<https://www.hindawi.com/journals/amete/2013/584816/>
6. Drukker, K., Giger, M. L., Horsch, K., Kupinski, M. A., Vyborny, C. J., & Mendelson, E. B. (2002). Computerized lesion detection on breast ultrasound. *Medical physics*, 29(7), 1438-1446.
<https://aapm.onlinelibrary.wiley.com/doi/abs/10.1118/1.1485995>
7. Han, K. T. M., & Uyyanonvara, B. (2016, March). A survey of blob detection algorithms for biomedical images. In *2016 7th International Conference of Information and Communication Technology for Embedded Systems (IC-ICTES)* (pp. 57-60). IEEE.
<https://ieeexplore.ieee.org/abstract/document/7467122>
8. Huang, T., Yang, G. J. T. G. Y., & Tang, G. (1979). A fast two-dimensional median filtering algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 27(1), 13-18.
<https://ieeexplore.ieee.org/abstract/document/1163188>
9. Jalalian, A., Mashohor, S., Mahmud, R., Karasfi, B., Saripan, M. I. B., & Ramli, A. R. B. (2017). Foundation and methodologies in computer-aided diagnosis systems for breast cancer detection. *EXCLI journal*, 16, 113
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5379115/>

10. Karimov, A., Razumov, A., Manbatchurina, R., Simonova, K., Donets, I., Vlasova, A., ... & Ushenin, K. (2019, October). Comparison of UNet, ENet, and BoxENet for Segmentation of Mast Cells in Scans of Histological Slices. In *2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)* (pp. 0544-0547). IEEE.
<https://ieeexplore.ieee.org/abstract/document/8958121>
11. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 1097-1105.
<https://kr.nvidia.com/content/tesla/pdf/machine-learning/imagenet-classification-with-deep-convolutional-nn.pdf>
12. Noble, J. A., & Boukerroui, D. (2006). Ultrasound image segmentation: a survey. *IEEE Transactions on medical imaging*, 25(8), 987-1010.
<https://ieeexplore.ieee.org/abstract/document/1661695>
13. O'Flynn, E. A., Ledger, A. E., & deSouza, N. M. (2015). Alternative screening for dense breasts: MRI. *American Journal of Roentgenology*, 204(2), W141-W149.
<https://www.ajronline.org/doi/full/10.2214/AJR.14.13636>
14. Oliver, A., Freixenet, J., Marti, J., Perez, E., Pont, J., Denton, E. R., & Zwiggelaar, R. (2010). A review of automatic mass detection and segmentation in mammographic images. *Medical image analysis*, 14(2), 87-110.
<https://www.sciencedirect.com/science/article/abs/pii/S1361841509001492>
15. Rodriguez-Cristerna, A., Guerrero-Cedillo, C. P., Donati-Olvera, G. A., Gómez-Flores, W., & Pereira, W. C. A. (2017, October). Study of the impact of image preprocessing approaches on the segmentation and classification of breast lesions on ultrasound. In *2017 14th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)* (pp. 1-4). IEEE.
<https://ieeexplore.ieee.org/abstract/document/8108826>
16. Ronneberger, O., Fischer, P., & Brox, T. (2015, October). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention* (pp. 234-241). Springer, Cham.
https://link.springer.com/chapter/10.1007/978-3-319-24574-4_28
17. Saini, K., Dewal, M. L., & Rohit, M. (2010). Ultrasound imaging and image segmentation in the area of ultrasound: a review. *International Journal of advanced science and technology*, 24.
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.446.5224&rep=rep1&type=pdf>
18. Sassaroli, E., Crake, C., Scorza, A., Kim, D. S., & Park, M. A. (2019). Image quality evaluation of ultrasound imaging systems: advanced B-modes. *Journal of applied clinical*

- medical physics*, 20(3), 115-124.
<https://aapm.onlinelibrary.wiley.com/doi/full/10.1002/acm2.12544>
19. Siddique, N., Sidike, P., Elkin, C., & Devabhaktuni, V. (2020). U-Net and its variants for medical image segmentation: theory and applications. *arXiv preprint arXiv:2011.01118*.
<https://arxiv.org/abs/2011.01118>
 20. Tsallis, C. (2011). The nonadditive entropy S_q and its applications in physics and elsewhere: Some remarks. *Entropy*, 13(10), 1765-1804.
<https://www.mdpi.com/1099-4300/13/10/1765>
 21. Wang, X. S., Huang, X. Y., & Fu, H. (2009, October). The Study of Color Tree Image Segmentation. In *2009 Second International Workshop on Computer Science and Engineering* (Vol. 2, pp. 303-307). IEEE.
<https://ieeexplore.ieee.org/abstract/document/5403294>
 22. W. K. Pratt, *Digital Image Processing*, 4th Edn., 2nd Ed., New York, NY, USA: Wiley, 2001.
https://www.researchgate.net/publication/31735099_Digital_Image_Processing_PIKS_Inside_WK_Pratt
 23. Yap, M. H., Pons, G., Marti, J., Ganau, S., Sentis, M., Zwigelaar, R., ... & Marti, R. (2017). Automated breast ultrasound lesions detection using convolutional neural networks. *IEEE journal of biomedical and health informatics*, 22(4), 1218-1226.
<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8003418>
 24. Zhou, P., Ye, W., Xia, Y., & Wang, Q. (2011). An improved canny algorithm for edge detection. *Journal of Computational Information Systems*, 7(5), 1516-1523.
<https://easystudy.info/attachment.php?aid=23321>
 25. https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.median_filter.html
 26. <https://research.google.com/colaboratory/faq.html>
 27. https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_blob.html
 28. <https://stats.stackexchange.com/questions/126238/what-are-the-advantages-of-relu-over-sigmoid-function-in-deep-neural-networks>
 29. https://www.tensorflow.org/api_docs/python/tf/keras/Model

10 Appendices

10.1 Third-Party Code and Libraries

Scikit-image – open source library, a collection of algorithms for image processing.

Tensorflow – end-to-end open source platform for machine learning that provides library for models implementation.

OpenCV2 – Computer Vision library for image processing.

SciPy – open source library for mathematics, science and engineering.

NumPy – python library for operation on big tables, arrays, matrices etc.

matplotlib – comprehensive library for plotting figures, creating static, visualization etc.

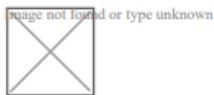
PIL – Python extension for image processing, reading, modifying, saving.

UNET_NUCLEI_TUTORIAL

(source: https://github.com/bnsreenu/python_for_microscopists/blob/master/076-077-078-Unet_nuclei_tutorial.py) - GitHub code that was big inspiration in creating the U-Net model implemented in this project.

10.2 Ethics Submission

This appendix includes a copy of the ethics submission for the project. After you have completed your Ethics submission, you will receive a PDF with a summary of the comments. That document should be embedded in this report, either as images, an embedded PDF or as copied text. The content should also include the Ethics Application Number that you receive.



09/02/2021

For your information, please find below a copy of your recently completed online ethics assessment

Next steps

Please refer to the email accompanying this attachment for details on the correct ethical approval route for this project. You should also review the content below for any ethical issues which have been flagged for your attention

Staff research - if you have completed this assessment for a grant application, you are not required to obtain approval until you have received confirmation that the grant has been awarded.

Please remember that collection must not commence until approval has been confirmed.

In case of any further queries, please visit www.aber.ac.uk/ethics or contact ethics@aber.ac.uk quoting reference number **18576**.

Assesment Details

AU Status

Undergraduate or PG Taught

Your aber.ac.uk email address

wos2@aber.ac.uk

Full Name

Wojciech Sowinski

Please enter the name of the person responsible for reviewing your assessment.

Neil Taylor

Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment

nst@ber.ac.uk

Supervisor or Institute Director of Research Department

cs

Module code (Only enter if you have been asked to do so)

CS39440

Proposed Study Title

Impact of image pre-processing on Image Segmentation in Deep Learning for breast cancer detection

Proposed Start Date

25 January 2021

Proposed Completion Date

1 June 2021

Are you conducting a quantitative or qualitative research project?

Mixed Methods

Does your research require external ethical approval under the Health Research Authority?

No

Does your research involve animals?

No

Does your research involve human participants?

No

Are you completing this form for your own research?

Yes

Does your research involve human participants?

No

Institute

IMPACS

Please provide a brief summary of your project (150 word max)

The purpose of this project is to reveal the importance of image pre-processing in a deep learning pipeline. Project will focus on comparison of two segmentations in deep learning performed on breast ultrasound images, exposing the difference in results caused by image pre-processing.

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?

Not applicable

Will appropriate measures be put in place for the secure and confidential storage of data?

Yes

Does the research pose more than minimal and predictable risk to the researcher?

No

Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?

No

Please include any further relevant information for this section here:

Is your research study related to COVID-19?

No

If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one.

Yes

Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change.

Yes

Please include any further relevant information for this section here:

10.3 Code Samples

10.3.1

```
# median filter
def medianFilter(image):
    median = ndimage.median_filter(image, 5)
    return median
```

10.3.2

```
def normalization(image):
    mean = np.mean(image)
    # print(mean)
    std = np.std(image)
    # print(std)
    image = (image - mean) / std
    return image
```

10.3.3

```
def multiOtsuThresholding(image):
    n = image
    n = medianFilter(n)
    n = normalization(n)
    thresholds = threshold_multiotsu(n, classes=5)
    regions = np.digitize(n, thresholds)
    output = img_as_ubyte(exposure.rescale_intensity(regions))
    return output
```

10.3.4

```
def blob_detection(img):

    image = img

    image = resize(img, (224,224,3))
    image = util.invert(image)
    #image = difference_of_gaussians(image, 2, 10, multichannel=True)
    #image = invert(img)
    image_gray = rgb2gray(image)

    blobs_log = blob_log(image_gray, min_sigma=2, max_sigma=30, num_sigma=30, threshold=.1)

    # Compute radii in the 3rd column.
    blobs_log[:, 2] = blobs_log[:, 2] * sqrt(2)

    w, h = 224, 224
    data = np.zeros((h, w, 3), dtype=np.uint8)
    data[0:224, 0:224] = [0, 0, 0] # make image black
    img = Image.fromarray(data, 'RGB')
    img = np.uint8(img)

    for each in blobs_log:
        x = np.float32(each[0])
        y = np.float32(each[1])
        cv2.circle(img, (y,x), each[2].astype("int"), (255,255, 0), -1)
    return img
```

10.3.5

```
def preprocessing(img):
    median = medianFilter(img)
    normalized = normalization(median)
    normalizedCopy = np.uint8(normalized)
    img = cv2.Canny(normalizedCopy, 100, 100)
    return img
```

10.3.6

```
def preprocess3channels_COB(img):
    img_original = img
    img_blob = blob_detection(img)
    img_canny = canny_preprocessing(img)
    img_canny = resize(img_canny, (IMG_HEIGHT, IMG_WIDTH, 3), mode='constant', preserve_range=True)

    red = img[:, :, 2]
    blue = img_canny[:, :, 1]
    green = img_blob[:, :, 0]
    new_img = np.zeros((224, 224, 3), dtype=np.uint8)

    new_img[:, :, 0] = blue
    new_img[:, :, 1] = green
    new_img[:, :, 2] = red

    return new_img
```

10.3.7

```
[ ] inputs = tf.keras.layers.Input((IMG_WIDTH, IMG_HEIGHT, IMG_CHANNELS))
```

```
[ ] s = tf.keras.layers.Lambda(lambda v: v / 255)(inputs)
```

```
[ ] c1 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(s)
    c1 = tf.keras.layers.Dropout(0.1)(c1)
    c1 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c1)
    p1 = tf.keras.layers.MaxPooling2D((2, 2))(c1)
```

10.3.8

```
#Expansive path
u6 = tf.keras.layers.Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(c5)
u6 = tf.keras.layers.concatenate([u6, c4]) # concatenation
c6 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u6)
c6 = tf.keras.layers.Dropout(0.2)(c6)
c6 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c6)
```

10.3.9

```
outputs = tf.keras.layers.Conv2D(1, (1, 1), activation='sigmoid')(c9)
```

10.3.10

```
model = tf.keras.Model(inputs=[inputs], outputs=[outputs])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

10.3.11

```
#Model Checkpoint
checkpointer = tf.keras.callbacks.ModelCheckpoint('UNET_median_blob_original.h5', verbose=1, save_best_only=True)
callbacks_original = [
    tf.keras.callbacks.EarlyStopping(patience=10, monitor='val_loss'),
    checkpointer,
    tf.keras.callbacks.TensorBoard(log_dir='logs')]

```

10.3.12

10.3.13

Model: "model"

Layer (type) Connected to	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 224, 224, 3)	0
=====		
lambda (Lambda) input_1[0][0]	(None, 224, 224, 3)	0
=====		
conv2d (Conv2D) lambda[0][0]	(None, 224, 224, 16)	448
=====		
dropout (Dropout) conv2d[0][0]	(None, 224, 224, 16)	0
=====		
conv2d_1 (Conv2D) dropout[0][0]	(None, 224, 224, 16)	2320
=====		
max_pooling2d (MaxPooling2D) conv2d_1[0][0]	(None, 112, 112, 16)	0
=====		
conv2d_2 (Conv2D) max_pooling2d[0][0]	(None, 112, 112, 32)	4640
=====		
dropout_1 (Dropout) conv2d_2[0][0]	(None, 112, 112, 32)	0
=====		
conv2d_3 (Conv2D) dropout_1[0][0]	(None, 112, 112, 32)	9248
=====		
max_pooling2d_1 (MaxPooling2D) conv2d_3[0][0]	(None, 56, 56, 32)	0
=====		
conv2d_4 (Conv2D) max_pooling2d_1[0][0]	(None, 56, 56, 64)	18496
=====		
dropout_2 (Dropout) conv2d_4[0][0]	(None, 56, 56, 64)	0
=====		

conv2d_5 (Conv2D) dropout_2[0][0]	(None, 56, 56, 64)	36928
max_pooling2d_2 (MaxPooling2D) conv2d_5[0][0]	(None, 28, 28, 64)	0
conv2d_6 (Conv2D) max_pooling2d_2[0][0]	(None, 28, 28, 128)	73856
dropout_3 (Dropout) conv2d_6[0][0]	(None, 28, 28, 128)	0
conv2d_7 (Conv2D) dropout_3[0][0]	(None, 28, 28, 128)	147584
max_pooling2d_3 (MaxPooling2D) conv2d_7[0][0]	(None, 14, 14, 128)	0
conv2d_8 (Conv2D) max_pooling2d_3[0][0]	(None, 14, 14, 256)	295168
dropout_4 (Dropout) conv2d_8[0][0]	(None, 14, 14, 256)	0
conv2d_9 (Conv2D) dropout_4[0][0]	(None, 14, 14, 256)	590080
conv2d_transpose (Conv2DTranspose) conv2d_9[0][0]	(None, 28, 28, 128)	131200
concatenate (Concatenate) conv2d_transpose[0][0] conv2d_7[0][0]	(None, 28, 28, 256)	0
conv2d_10 (Conv2D) concatenate[0][0]	(None, 28, 28, 128)	295040
dropout_5 (Dropout) conv2d_10[0][0]	(None, 28, 28, 128)	0
conv2d_11 (Conv2D) dropout_5[0][0]	(None, 28, 28, 128)	147584

conv2d_transpose_1 (Conv2DTrans	(None, 56, 56, 64)	32832
conv2d_11[0][0]		

concatenate_1 (Concatenate)	(None, 56, 56, 128)	0
conv2d_transpose_1[0][0]		

conv2d_5[0][0]

conv2d_12 (Conv2D)	(None, 56, 56, 64)	73792
concatenate_1[0][0]		

dropout_6 (Dropout)	(None, 56, 56, 64)	0
conv2d_12[0][0]		

conv2d_13 (Conv2D)	(None, 56, 56, 64)	36928
dropout_6[0][0]		

conv2d_transpose_2 (Conv2DTrans	(None, 112, 112, 32)	8224
conv2d_13[0][0]		

concatenate_2 (Concatenate)	(None, 112, 112, 64)	0
conv2d_transpose_2[0][0]		

conv2d_3[0][0]

conv2d_14 (Conv2D)	(None, 112, 112, 32)	18464
concatenate_2[0][0]		

dropout_7 (Dropout)	(None, 112, 112, 32)	0
conv2d_14[0][0]		

conv2d_15 (Conv2D)	(None, 112, 112, 32)	9248
dropout_7[0][0]		

conv2d_transpose_3 (Conv2DTrans	(None, 224, 224, 16)	2064
conv2d_15[0][0]		

concatenate_3 (Concatenate)	(None, 224, 224, 32)	0
conv2d_transpose_3[0][0]		

conv2d_1[0][0]

conv2d_16 (Conv2D)	(None, 224, 224, 16)	4624
concatenate_3[0][0]		

dropout_8 (Dropout)	(None, 224, 224, 16)	0
conv2d_16[0][0]		
<hr/>		
conv2d_17 (Conv2D)	(None, 224, 224, 16)	2320
dropout_8[0][0]		
<hr/>		
conv2d_18 (Conv2D)	(None, 224, 224, 1)	17
conv2d_17[0][0]		
=====		
=====		
Total params: 1,941,105		
Trainable params: 1,941,105		
Non-trainable params: 0		
<hr/>		
<hr/>		