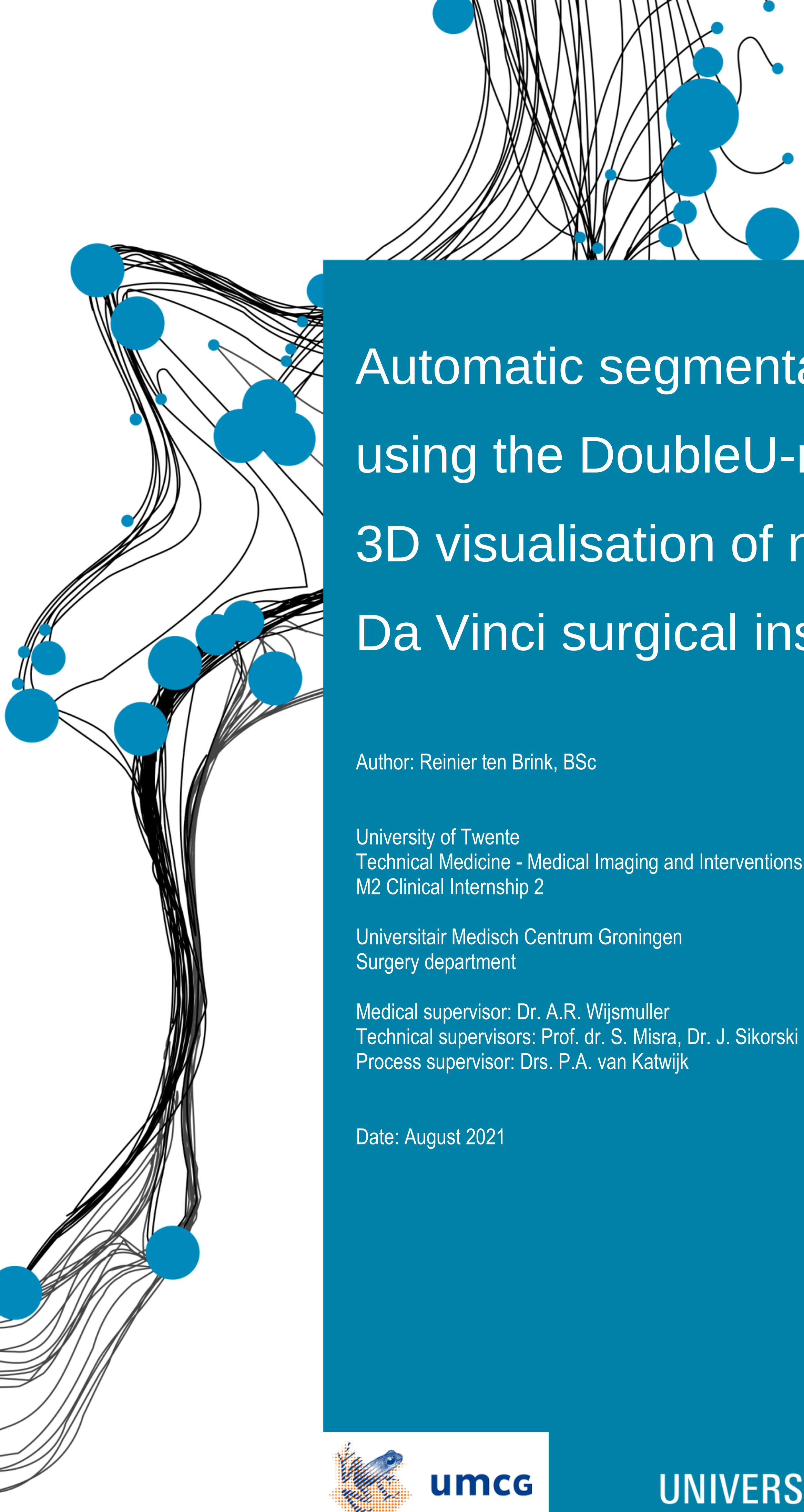

# Automatic segmentation using the DoubleU-net and 3D visualisation of non-rigid Da Vinci surgical instruments

Author: Reinier ten Brink, BSc

University of Twente
Technical Medicine - Medical Imaging and Interventions
M2 Clinical Internship 2

Universitair Medisch Centrum Groningen
Surgery department

Medical supervisor: Dr. A.R. Wijsmuller
Technical supervisors: Prof. dr. S. Misra, Dr. J. Sikorski
Process supervisor: Drs. P.A. van Katwijk

Date: August 2021

## Abstract

**Introduction:** Locally advanced primary and recurrent rectal carcinoma have a large risk of not being radically removed during surgery. Image guided robot assisted surgery might help increase radicality. However, since these instruments are non-rigid, stereotactic navigation alone can not offer a solution alone. This study proposes a method of combining computer vision techniques with stereotactic navigation to achieve image guided robot assisted surgery. As a first step, the study aims to automatically segment a Da Vinci instrument using a machine learning model and visualise this instrument in 3D from stereotactic endoscope images.
**Method:** Using the 2015 MICCAI dataset and the DoubleU-net algorithm, a model is trained to semantically segment a Da Vinci needle driver from ex-vivo 2D image sequences.
Stereo images of a Da Vinci needle driver in front of a printed surgical scene area taken using an endoscope. Rectification, disparity maps and triangulation were used for 3D instrument reconstruction.
**Results:** Several performance scores are reported for the DoubleU-Net on the dataset.
Rectification of stereo images is done successfully. The needle driver is not visible when stereo images are turned into disparity maps. Therefore 3D visualisation of the instrument could not be done.
**Discussion:** Automatic segmentation of Da Vinci surgical instruments can be done accurately. However, many hurdles need to be overcome before it can be used in a live setting.
The setup of the experiment needs to be improved and/or other algorithms need to be used in order to successfully visualise a surgical instrument in a 3D space.

## Introduction

Last year, about 3300 people in the Netherlands got the diagnosis for rectal carcinoma. Around 95% of those diagnosed with stadium I-III will need surgery as curative treatment.[1] However, 15-20% of locally advanced primary rectal carcinomas are also not removed radically.[2] Despite surgery, the recurrence of locally recurrent rectal cancer is 5-10%. Since local recurrent carcinomas are often not limited to one surgical compartment, one of the major challenges in removing local recurrence is achieving clear resection margins and removing the tumor radically. Despite the best of intentions, over 50% of resections of locally recurrent rectal carcinomas are not removed radically.[3]

More and more locally advanced primary and recurrent rectal tumours are treated surgically in a minimally-invasive approach, often assisted by the Da Vinci Surgical robot (Intuitive Surgical, Sunnyvale, California).[4] By making anatomical planes and landmarks more easy to detect, radicality could be increased. One method for this has been the use of surgical navigation based on preoperative imaging, such as MRI or CT. It has been established that surgical navigation increases precision and minimizes invasiveness. A condition for the use of this technique is that the target organ does not move too much. This would alter the position compared to the preoperative scan, which would decrease accuracy significantly. For the rectum it has been proven that the organ does not move very much because of its attachment to the bony pelvis.[4] Therefore image guided surgical navigation is feasible.

Unfortunately, robot joint encoder data can not be used to determine the position of the surgical instruments. This is because it uses a cable driven system and therefore calibration is very difficult, time consuming and errors accumulate over time.[5] At the University Medical Center Groningen (UMCG), a proposed method for surgical navigation is stereotactic navigation, which is currently used often for neurosurgery. In the UMCG, the navigation system of Brainlab® is used. This form of navigation is based on a stereotactic camera that tracks infrared (IR) markers that are mounted on a surgical tool. After calibration, the position of the tool with respect to the patient can be calculated. When using this method in combination with the Da Vinci Robot, there are two major problems. The first problem is that the IR markers have to be in the line of sight of the camera at all times. This is however not always the case due to e.g. sterile draping or movement of the robotic arm. The second problem is the localisation of the tip of the surgical instrument. The tip of the arm consists of endowrists that move relative to the IR markers during surgery, while the calibration is based on rigid instruments.[6]

Therefore, this study suggests placing the IR markers of the Brainlab® system on the Da Vinci stereo camera, so after calibration the position of the camera can be known relative to the patient. Then using a machine learning model to automatically segment the surgical instrument in both video feeds. Next, using computer vision techniques, the position of the instrument relative to the camera can be known. By combining this info, the position of the instrument relative to the patient can be known. A visual representation of this design is presented in figure 1. This study focuses on investigating whether a machine learning model can accurately segment a Da Vinci surgical instrument automatically. Besides that, the study aims to show that it is possible to visualise a surgical instrument from stereo images taken by the Da Vinci stereo endoscope.

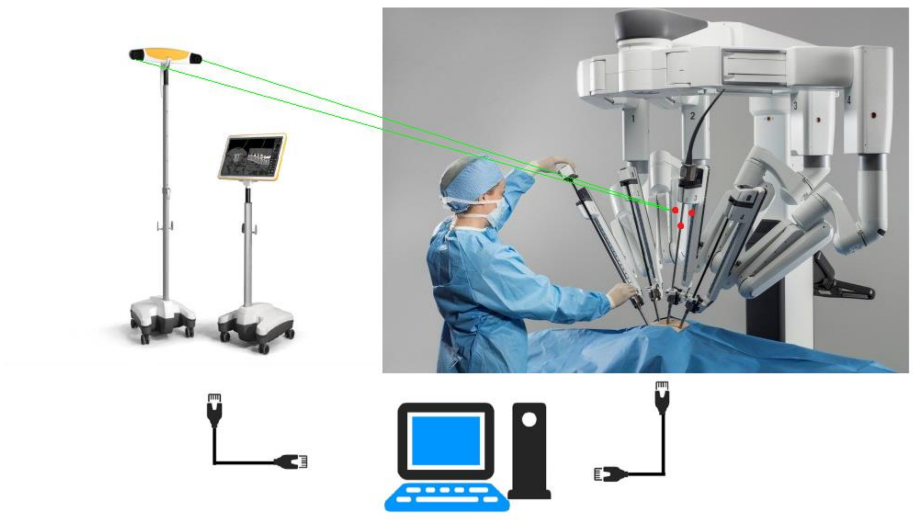

Figure 1: A visual representation of the proposed design. Left: the conventional Brainlab navigation system registers the IR markers on the Da Vinci stereo camera (right). Below: a computer runs an application that combines information from Brainlab and the video feed from the camera.

## Method

This study has of two objectives: 1) segmenting a non-rigid Da Vinci instrument using a machine learning algorithm and 2) visualising a Da Vinci instrument in a 3D space from stereo images using computer vision. This section describes the methods used to achieve those objectives.

### Machine Learning

For segmenting a Da Vinci instrument using machine learning, there are two things necessary: an algorithm architecture and a training dataset. Many different algorithms for segmenting medical images. One of the most used algorithms is the U-Net. It labels each pixel of an image with its corresponding class in a process named semantic image segmentation.[7] While this algorithm produces accurate results, new algorithms such as the DoubleU-Net have produced even more accurate results on several datasets, such as the CVC-ClinicDB dataset of colonoscopy images for polyp detection (Sørensen–Dice coefficient (SDC) = 0.9239). The basic architecture of this algorithm consists of two U-Net stacked on top of each other. A more detailed description will be given in the next paragraph.[8] Because of these encouraging results, this study will use the DoubleU-Net as a segmentation algorithm.

#### The DoubleU-net algorithm

The DoubleU-net uses semantic segmentation where it labels each pixel in an image with a class. There are already many medical applications that use semantic segmentation such as: detection and segmentation of lesions, improving administration of radiation and improving medical images. [9] In order to describe the architecture of the DoubleU-Net, the architecture of its predecessor, the U-Net will be described.

The U-Net was created by Ronneberger et al. [7] in 2015 and is built on the architecture of the Fully Convolutional Network (FCN). This means it only performs convolution and is able to classify each pixel in the input instead of generating a discrete label from an output image, which was the case with previous Convolutional Neural Networks (CNN).
Figure 2 shows the shape of the U-Net, where it is visible where the network gets its name. In the left leg of the U-Net, called the encoder, the input image gets downsampled and a lot is learned about what is in the image. FCNs are great at understanding the features in an image, but unfortunately have trouble with understanding where these features come from.

This is because in downsampling the image, a lot of spatial information is lost. In the right leg of the U-Net, called the decoder, the image is upsampled to localize the detected features and create a mask with the same size as the input image.
Next, is where the big advantage of the U-Net comes in. Between the two legs of the U-Net, grey arrows are visible, which are called skip connections. These skip connections connect the feature map of the encoder to the corresponding layer in the decoder. This gives the network extra information about the location of the features, which greatly increases segmentation accuracy.[10]
While this U-Net improved semantic segmentation, a lot of researchers still tried to make improvements. This is where the DoubleU-Net comes in.

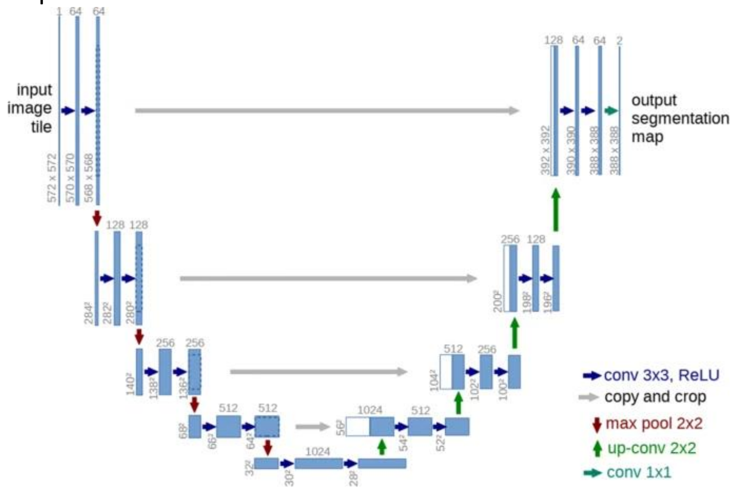

Figure 2: The structure of the U-Net [7]

The main goal of creating the DoubleU-Net was to improve the performance of the U-Net. In order to achieve this, Jha et al. [8] proposed an architecture which put two U-net architectures in sequence, where the encoder of the first U-net is a pre-trained VGG-19. A model trained on the ImageNet dataset. A block diagram of the DoubleU-Net is presented in figure 3. As visible, both the first and second U-net produce an output image.
Since the second network increases the number of layers, more features can be learned, which should improve the output segmentation of the second output compared to the first.
Furthermore, squeeze and excite blocks are added to reduce redundant information and Atrous Spatial Pyramid Pooling are added to improve the resolution of the output mask.
Python code for generating the DoubleU-Net architecture can be found in Appendix A.

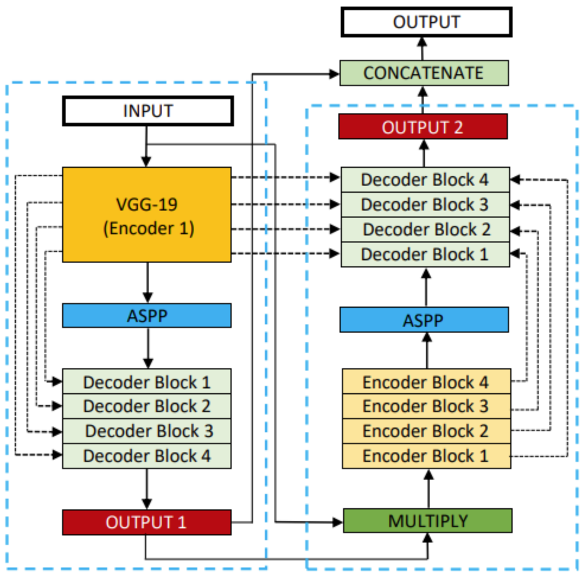

Figure 3: Block diagram of the DoubleU-net architecture [8]

## Dataset

To check whether the DoubleU-Net algorithm can accurately segment Da Vinci instruments, a dataset is needed. Having a good dataset that is large enough, is often a challenge in machine learning. In 2015, a medical image computing and computer assisted intervention challenge (MICCAI), released a dataset consisting of three 45 second 2D images sequences of one Large Needle Driver instrument in an ex-vivo setup and a ground truth where this instrument is segmented manually.[11] In order to be able to put this data into the DoubleU-Net, all the frames had to be extracted from the videos and saved as jpg files. This was done using the code in Appendix B. For 3D visualisation of the instrument, the entire instrument needs to be segmented. Since the ground truth is divided into the different parts of the instrument, all the ground truth frames had to be altered to make the entire instrument white. This was done using a simple threshold. The code for this can be found in Appendix C.

## Training the model

The model is trained using a GPU from Google Collab. The training code can be found in Appendix D. Dependencies for this code can be found in Appendix E. The data is randomly split into 80% training, 10% validation and 10% testing (Appendix F) and uploaded to Google Drive. A batch size of 4 is used with a learning rate of 1e-4. The model is trained for 8 epochs. To prevent overfitting, the model uses the callback: EarlyStopping. Multiple consecutive decreases in loss function are a sign of overfitting. Therefore, if the loss function of the validation set is dropping for 3 epochs in a row, the model stops training and restores the weights from the model with the smallest loss function so far. There are four metrics used to evaluate the performance of the model:

- Sørensen Dice Coefficient (DSC), which is a measurement of the similarity between two samples. The closer this score is to 1, the better the performance.
- Loss function, which can be calculated by 1- DSC. This is often shown separately in research. The closer this score is to 0, the better the performance.
- Interjection over union (IOU), which calculates the overlap between the predicted output and the ground truth. The closer this score is to 1, the better the performance.
- Recall or sensitivity, which measures the number of true positives divided by all positive pixels. The closer this score is to 1, the better the performance.
- Precision, which measures the number of true positives divided by all true pixels. The closer this score is to 1, the better the performance.

These performance scores will be calculated for the train, validation and test set individually.
Lastly, the code from Appendix G is used to make a prediction and segmentation on a 2D image sequence which was used to judge participants of the challenge. This prediction segmented the Da Vinci instrument from each frame and saved it. From all these frames, a video was generated using the code from Appendix H.

# Stereo 3D visualisation

Stereo images were taken of a Da Vinci needle driver in front of a printed surgical scene using the Da Vinci stereo endoscope. This setup is displayed in figure 4). This resulted in images as seen in figure 5. Due to time constraints, the instrument was segmented manually from the scene using Microsoft Paint.
To visualise the Da Vinci surgical instrument, it needs to be reconstructed. Stereo reconstruction follows three basic steps: rectification, disparity calculation and triangulation. But before these steps can be taken, the stereo camera needs to be calibrated to estimate the camera parameters.[12] To do this, several images need to be taken of a rectangle shaped checkerboard. These images need to be taken at different angles and distances from the camera. For this study, 16 image pairs of a 7 x 9 checkerboard with 16 mm checkers were taken using the Da Vinci stereo endoscope. In this study, the Matlab Camera Calibration Toolbox was then used to estimate the camera parameters.[13] These parameters were then exported to the Matlab workspace.

The first step in stereo reconstruction is rectifying the stereo images. Rectification uses the camera calibration parameters to align the left and right image. This way, when looking for matching points between the left and right image, only a 1D space needs to be considered. [12] The function rectifyStereoImages is used.
Next, the disparity between the images is calculated. This means that the distance between matching points is calculated. There are several methods of calculating disparity, but this study tried using the functions disparityBM [14], which uses block matching, and disparitySGM [15], which uses semi global matching from Matlab
The last step is to use triangulation to project the pixel points in a 3D space. Because only visualisation of the instrument is desired, this step is applied to a segmented image of the target.

The steps described are applied to the described stereo images. The results of this will be described in the next chapter.

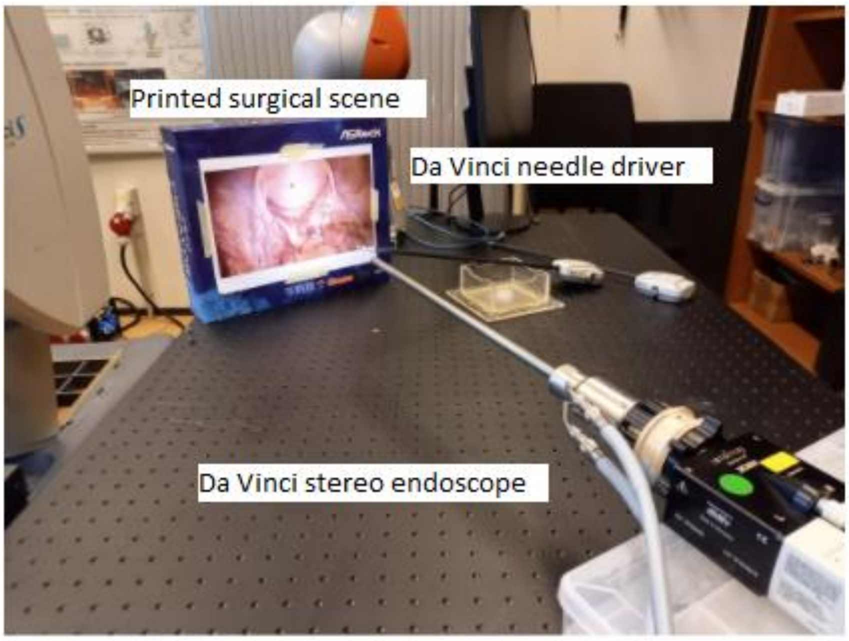

Figure 4: The experimental setup.

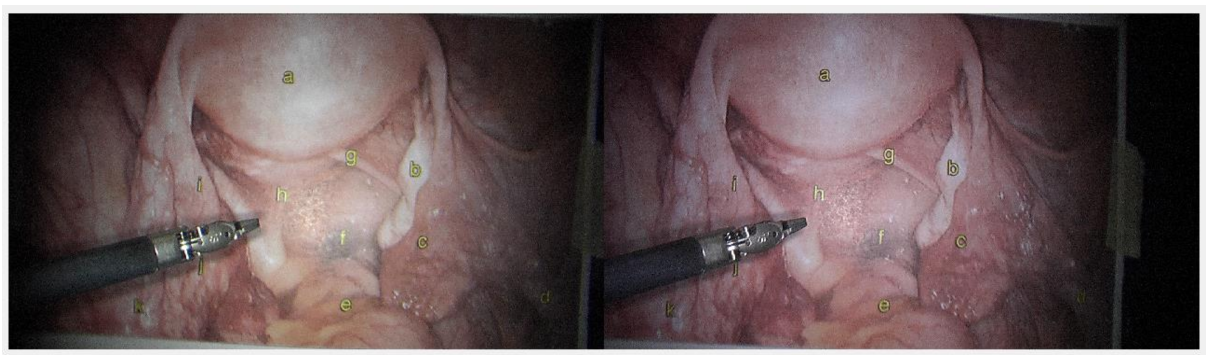

Figure 5: Example of a pair of images of the needle driver in front of a surgery frame.

## Results

This section gives an overview of the results obtained from training the machine learning model and the results from the 3D visualisation.

## Machine learning model

The performance scores and qualitative results of the DoubleU-Net on the 2015 MICCAI dataset are presented below, as well as two frames from the video from the test dataset. The full video can be found in this link: https://drive.google.com/drive/folders/1qsf-8YKOZqE0mbyaxRzUn964LWXTLVyu?usp=sharing

| Train Dice | Train Loss | Train IOU | Train Recall | Train Precision |
|---|---|---|---|---|
| 0.9687 | 0.0313 | 0.9399 | 0.9399 | 0.9783 |

Table [1] Performance scores of the model on the training dataset

| Validation Dice | Validation Loss | Validation IOU | Validation Recall | Validation Precision |
|---|---|---|---|---|
| 0.8726 | 0.1274 | 0.7789 | 0.8530 | 0.9000 |

Table [2] Performance scores of the model on the validation dataset

| Test Dice | Test Loss | Test IOU | Test Recall | Validation Precision |
|---|---|---|---|---|
| 0.9066 | 0.0934 | 0.8306 | 0.8868 | 0.9338 |

Table [3] Performance scores of the model on the test dataset

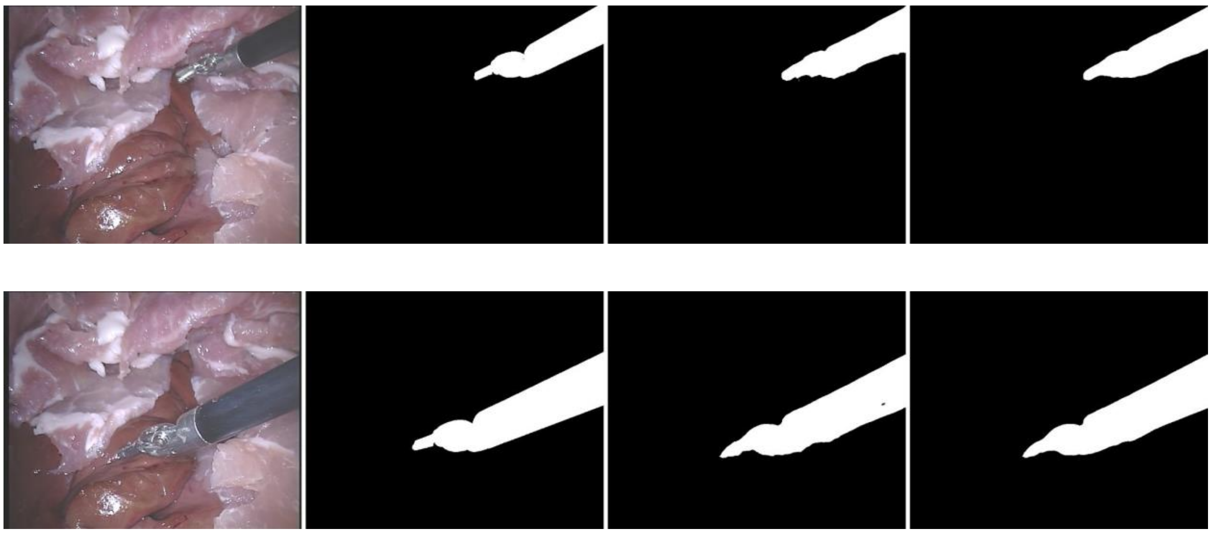

Figure 6: Results of the model on two frames. From left to right: the original frames, the ground truth masks, the output masks from network 1, the output masks from network 2, the instrument segmented from the original image using output mask 2.

# 3D Visualisation

To calibrate the Da Vinci stereo endoscope, 16 image pairs of a 7 x 9 checkerboard with 16 mm checkers were taken. An example of these image pairs are displayed in figure 7. For calibrating, 5 image pairs were removed to reduce the mean calibration error. The final mean error was 1.08 pixels (figure 8). A visualisation of the camera parameters is displayed in figure 9. After exporting the camera parameters were exported into the Matlab workspace the image pairs of the Da Vinci needle driver in front of a printed surgery frame were loaded. These images were then rectified, as visible in figure 10.

After rectifying, a disparity map was created using block matching and semi global matching, as visible in figure 11. Unfortunately, the instrument could not be visualised in 3D space, because the disparity maps are noisy and no structures are recognizable. Several attempts to improve the disparity map, such as filtering the images to reduce noise, different disparity algorithms, altering disparity range and uniqueness threshold all made did not make the needle driver structure visible.

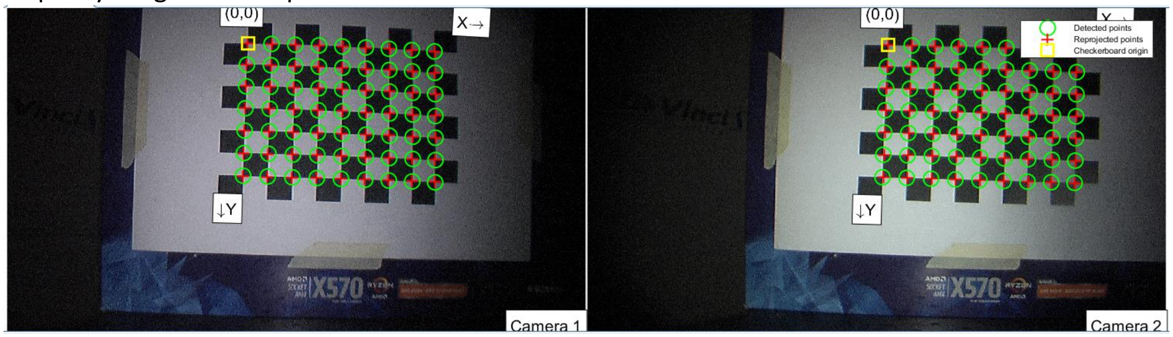

Figure 7: Example of a pair of calibration images

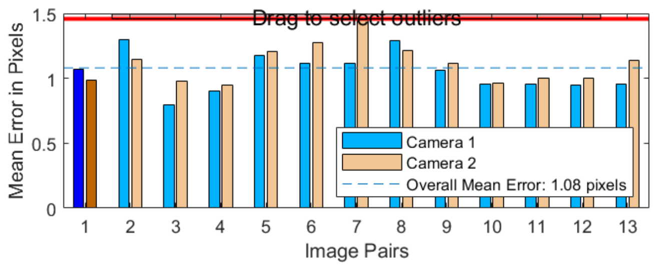

Figure 8: Plot of the reprojection errors of all image pairs

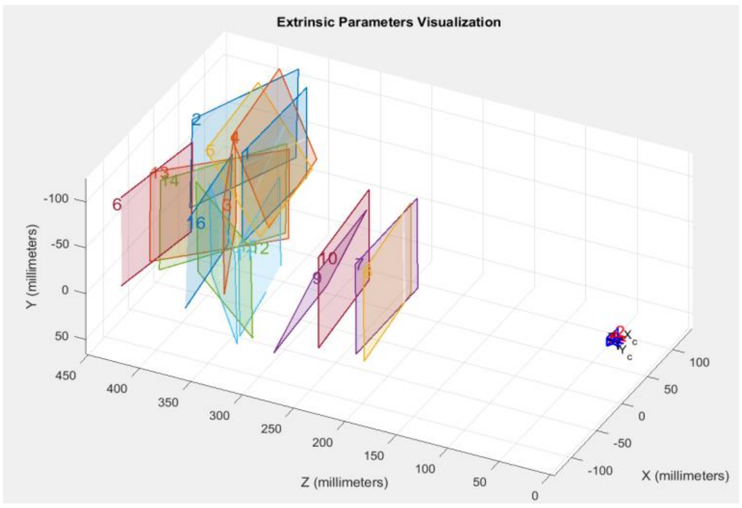

Figure 9: A visualisation of the camera parameters. Left displays the position of the checkerboard in all the calibration images. Right displays the position of the cameras within the endoscope.

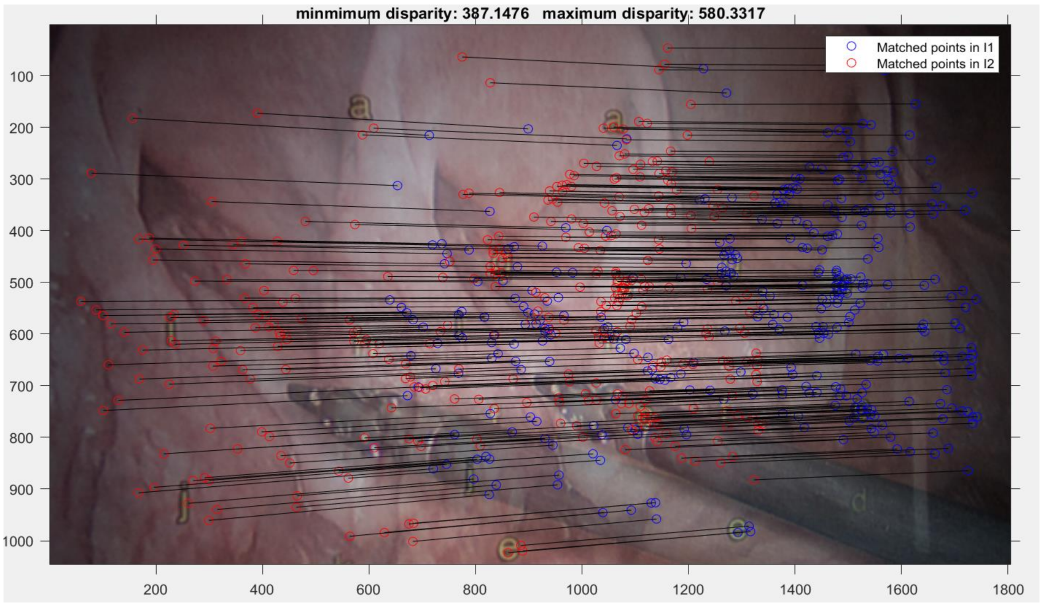

Figure 10: Overlay of the rectified images of the needle driver in front of a surgery frame. The blue and red dots display matching points between both images and the distance and direction between them.

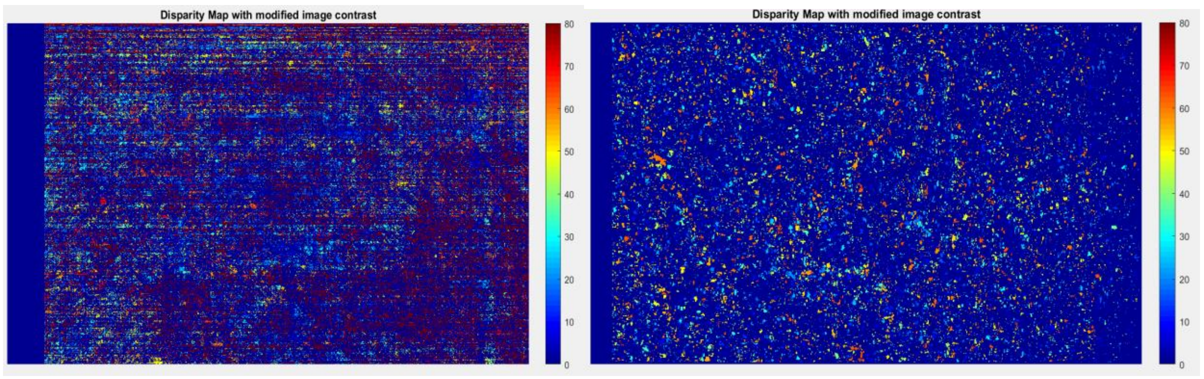

Figure 11: Disparity maps of the instrument in front of a surgery frame using semi global matching (left) and block matching (right). The colors display the distance of each pixel from the camera. Red is near the camera, while blue is far away from the camera.

# Discussion

The DoubleU-Net can accurately segment a Da Vinci surgical instrument from a set of 2D monocular images. There are however several steps that need to be taken before clinical use, since there are several limitations in this study. This section describes these limitations, and what could be improved in future research to improve results.

There are several limitations with regards to the dataset. First of which is the fact that all data was gathered ex vivo. Therefore it is not yet known how the DoubleU-Net would perform in vivo. During real surgery, it may be harder to detect and segment the instrument due to occlusion, smoke and bleeding. Besides this, this study decided to train the model on segmenting just one instrument. A needle driver was used because this was the instrument used in the dataset. However, in the surgery phase where navigation would be needed, the diathermic hook is the preferred instrument. A future model should be trained on a dataset containing this instrument. Also, during real surgery, multiple instruments are used. Perhaps the model could even be trained to recognize these instruments. But since this would require instance segmentation, another algorithm should be used.
To improve the dataset, footage of a real surgery would make the model more useful in a clinical setting. Unfortunately, such a dataset was not found. So data would need to be gathered and ground truth masks would need to be made. But this is a very time-consuming task.
Also, the dataset consisted of mono-images, while the end goal is to apply the model to stereo images. But since stereo-images are basically two mono-images, this would probably not influence the performance of the model. It could however influence the speed of the model in a live setting, because twice as many calculations are needed.
This brings us to the next limitation, which is the fact that the model was not tested on live footage. In order to be used clinically, the model needs to be able to segment the surgical instrument in live footage from two inputs with a reasonable number of frames per second. In order to do this, a GPU is probably needed. A future study could try to segment instruments from live footage from the Da Vinci stereo endoscope. It might also be interesting to look at other algorithms that do not aim to segment every pixel to save computing power, or the use of temporal CNNs.
As is visible in the segmented video, the model has trouble segmenting the instrument while the jaws are open. This could be due to the fact that the model was not trained for many epochs. Unfortunately, running more epochs might not improve performance. As is clear from the difference in performance scores between the training and validation set, there is already slight overfitting. So more and better data might have more effect on the performance of the model. It could also be useful to try different settings for e.g. batch size and learning rate.

The end goal is to create a system where the position of surgical instruments in the patient is known based on preoperative scans. However, the requirements, such as the accuracy, frames/second and latency of such a system have not been established yet. Future studies should set these requirements.
This system should combine segmented footage from the Da Vinci stereo camera with positional information of the camera from the Brainlab® system. Combining this information could be a challenge in future studies. This software should probably be written in C++ because of speed issues. There is already open-source software available that is able to extract information from Brainlab® called IGTLink. Using an ethernet cable, Brainlab can be linked to a computer with modified IGTLink software. In the UMCG there is already expertise on how to work with IGTLink. Glas et al. linked information of the Brainlab® system to the Microsoft HoloLens.[16] It would be useful to use this expertise in further developing this system. Calibration of the Brainlab® system could perhaps be done on the sacrum.
All of this should be finalized in a user-friendly interface, which uses one programming language that can be used in the operating room.

Anatomy including tumor are generally segmented to facilitate peroperative stereotactic navigation. On MRI images this is usually performed manually. Preferably, this segmentation is performed automatically to the extent possible. Deep learning for automated segmentation of pelvic anatomy by 3D U-net training on MRI scans seems feasible. [17]. It might also be possible to use other algorithms. A ground truth standard for segmentations of pelvic anatomy already exists [18]. A preliminary test has already been initiated to investigate whether a machine learning model can be trained on MRI scans with manually segmented pelvic visceral and neural anatomy as ground truth.

This study has not yet been able to successfully reconstruct a surgical instrument in 3D from the Da Vinci stereo endoscope. There are multiple reasons why this might be.
First, it might be that not the right algorithm for detecting disparity has been used, due to time constraints. A future study could try more algorithms or methods to determine the location of the surgical instrument. Lu et al. have deployed several methods to reconstruct a scene from Da Vinci stereotactic video.[19] Parachami et al. also proved that it is feasible to construct a scene from stereotactic Da Vinci images.[12]
Other reasons might be more data related. For example, as is visible in figure 8, the rectification is not the same everywhere and not as straight as perhaps is necessary.
This could be because there is slight distortion in the images. Using the function undistortImage, attempts have been made to remove this distortion.[20] Unfortunately without success.
It might also be that noise in the images make it more difficult for the algorithm to detect disparity between the images. An attempt has been made to remove noise using a gaussian filter, but more methods could be investigated.
As visible in figure 9, there is a difference in colour between the left and right image. How this is possible is unknown, but perhaps the room should have been darkened, with the endoscope as the only lightsource. This difference in colour could prevent the algorithm from detecting disparity. Perhaps histogram matching could equalize the colours. But this has not been tried.
It might also be that the colour of the arm of the surgical instrument is too uniform. Therefore it might be difficult to find matching points.
This study used a flat printed surface as a background. This background suggests depth, which might confuse the algorithm. This also means the algorithm has only two surfaces it can compare. Perhaps a setup with a more irregular surface might help the algorithm.
If a setup is found which is capable of reconstructing a surgical instrument, the accuracy of this method needs to be determined and, if necessary, improved.

Using surgical navigation with instruments containing wrists remains a challenge. The method described in this report needs a lot of development before it can be used clinically. But it has a lot of potential to improve navigation, make it easier for surgeons to determine the anatomical plane and hopefully increase radical removal of locally advanced and  recurrent rectal carcinomas. It might also be useful in other fields where Robot Assisted surgery is used and the target organ does not move much relative to the preoperative scan or where intraoperative imaging is used to navigate.

# References

[1] Integraal kankercentrum Nederland. (2020). *Behandeling van darmkanker*. Last visited at 09-06-2021 from https://iknl.nl/kankersoorten/darmkanker/registratie/behandeling
[2] Mariathasan, A.B., et al., *Beyond total mesorectal excision in locally advanced rectal cancer with organ or pelvic side-wall involvement.* Eur J Surg Oncol, 2018. 44(8): p. 1226-1232.
[3] van Zoggel, D. M. G. I., Bosman, S. J., Kusters, M., Nieuwenhuijzen, G. A. P., Cnossen, J. S., Creemers, G. J., van Lijnschoten, G., & Rutten, H. J. T. (2017). *Preliminary results of a cohort study of induction chemotherapy-based treatment for locally recurrent rectal cancer.* British Journal of Surgery, 105(4), 447–452. https://doi.org/10.1002/bjs.10694
[4] Kwak, J.-M., Romagnolo, L., Wijsmuller, A., Gonzalez, C., Agnus, V., Lucchesi, F. R., Melani, A., Marescaux, J., & Dallemagne, B. (2019). *Stereotactic Pelvic Navigation With Augmented Reality for Transanal Total Mesorectal Excision*. Diseases of the Colon & Rectum, 62(1), 123–129. https://doi.org/10.1097/dcr.0000000000001259
[5] Du, X., Kurmann, T., Chang, P.-L., Allan, M., Ourselin, S., Sznitman, R., Kelly, J. D., & Stoyanov, D. (2018). *Articulated Multi-Instrument 2-D Pose Estimation Using Fully Convolutional Networks*. IEEE Transactions on Medical Imaging, 37(5), 1276–1287. https://doi.org/10.1109/tmi.2017.2787672
[6] Atallah, S., Parra-Davila, E., Melani, A. G. F., Romagnolo, L. G., Larach, S. W., & Marescaux, J. (2019). *Robotic-assisted stereotactic real-time navigation: initial clinical experience and feasibility for rectal cancer surgery*. Techniques in Coloproctology, 23(1), 53–63. https://doi.org/10.1007/s10151-018-1914-y
[7] Ronneberger, O., Fischer, P., & Brox, T. (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation.* In Lecture Notes in Computer Science (pp. 234–241). Springer International Publishing. https://doi.org/10.1007/978-3-319-24574-4_28
[8] Jha, D., Riegler, M. A., Johansen, D., Halvorsen, P., & Johansen, H. D. (2020). *DoubleU-Net: A Deep Convolutional Neural Network for Medical Image Segmentation.* 2020 IEEE 33rd International Symposium on Computer-Based Medical Systems (CBMS).https://doi.org/10.1109/cbms49503.2020.00111
[9] Asgari Taghanaki, S., Abhishek, K., Cohen, J. P., Cohen-Adad, J., & Hamarneh, G. (2020). *Deep semantic segmentation of natural and medical images: a review.* Artificial Intelligence Review, 54(1), 137–178. https://doi.org/10.1007/s10462-020-09854-1
[10] Hesamian, M. H., Jia, W., He, X., & Kennedy, P. (2019). *Deep Learning Techniques for Medical Image Segmentation: Achievements and Challenges*. Journal of Digital Imaging, 32(4), 582–596. https://doi.org/10.1007/s10278-019-00227-x
[11] MICCAI (2015). *Instrument segmentation and tracking*. Retrieved June 7 2021 from https://endovissub-instrument.grand-challenge.org/Data/
[12] Parchami, M., Cadeddu, J. A., & Mariottini, G.-L. (2014). *Endoscopic stereo reconstruction: A comparative study. 2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society.* 2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). https://doi.org/10.1109/embc.2014.6944115
[13] J.Y. Bouguet.(2015) *Camera Calibration Toolbox for MATLAB*. http://www.vision.caltech.edu/bouguetj/calib_doc/index.html
[14] MathWorks, (2021). *Compute disparity map using block matching (2014a)*. Retrieved august 11, 2021 from https://nl.mathworks.com/help/vision/ref/disparitybm.html
[15] MathWorks, (2021). *Compute disparity map through semi-global matching (2019a)*. Retrieved august 11, 2021 from https://nl.mathworks.com/help/vision/ref/disparitysgm.html
[16] Glas, H. H., Kraeima, J., van Ooijen, P. M. A., Spijkervet, F. K. L., Yu, L., & Witjes, M. J. H. (2021). *Augmented Reality Visualization for Image-Guided Surgery: A Validation Study Using a Three-Dimensional Printed Phantom*. Journal of Oral and Maxillofacial Surgery. https://doi.org/10.1016/j.joms.2021.04.001

[17] Zhu, H., Zhang, X., Shi, Y., Li, X., & Sun, Y. (2021). *Automatic segmentation of rectal tumor on diffusion-weighted images by deep learning with U-Net.* Journal of Applied Clinical Medical Physics. https://doi.org/10.1002/acm2.13381
[18] Wijsmuller, A. R., Giraudeau, C., Leroy, J., Kleinrensink, G. J., Rociu, E., Romagnolo, L. G., Melani, A. G. F., Agnus, V., Diana, M., Soler, L., Dallemagne, B., Marescaux, J., & Mutter, D. (2018). *A step towards stereotactic navigation during pelvic surgery: 3D nerve topography.* Surgical Endoscopy, 32(8), 3582–3591. https://doi.org/10.1007/s00464-018-6086-3
[19] Lu, J., Jayakumari, A., Richter, F., Li, Y., & Yip, M.C. (2020). *SuPer Deep: A Surgical Perception Framework for Robotic Tissue Manipulation using Deep Learning for Feature Extraction*. *ArXiv, abs/2003.03472*.
[20] MathWorks, (2021). *Correct image for lens distortion (2014a)*. Retrieved august 11, 2021 from https://nl.mathworks.com/help/vision/ref/undistortimage.html

# Appendix A: DoubleU-Net model

```python
import tensorflow as tf
from tensorflow.keras.layers import *
from tensorflow.keras.models import Model
from tensorflow.keras.applications import *
from keras.models import Sequential

def squeeze_excite_block(inputs, ratio=8):
    init = inputs
    channel_axis = -1
    filters = init.shape[channel_axis]
    se_shape = (1, 1, filters)

    se = GlobalAveragePooling2D()(init)
    se = Reshape(se_shape)(se)
    se = Dense(filters // ratio, activation='relu',
kernel_initializer='he_normal', use_bias=False)(se)
    se = Dense(filters, activation='sigmoid',
kernel_initializer='he_normal', use_bias=False)(se)

    x = Multiply()([init, se])
    return x

def conv_block(inputs, filters):
    x = inputs

    x = Conv2D(filters, (3, 3), padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(filters, (3, 3), padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = squeeze_excite_block(x)

    return x

def encoder1(inputs):
    skip_connections = []

    model = VGG19(include_top=False, weights='imagenet',
input_tensor=inputs)
    names = ["block1_conv2", "block2_conv2", "block3_conv4",
"block4_conv4"]
    for name in names:
        skip_connections.append(model.get_layer(name).output)

    output = model.get_layer("block5_conv4").output
    return output, skip_connections

def decoder1(inputs, skip_connections):
    num_filters = [256, 128, 64, 32]
    skip_connections.reverse()
    x = inputs

    for i, f in enumerate(num_filters):
        x = UpSampling2D((2, 2), interpolation='bilinear')(x)
        x = Concatenate()([x, skip_connections[i]])
        x = conv_block(x, f)
    return x

# def encoder2(inputs):
#     skip_connections = []

#     output = DenseNet121(include_top=False, weights='imagenet')(inputs)
#     model = tf.keras.models.Model(inputs, output)

#     names = ["input_2", "conv1/relu", "pool2_conv", "pool3_conv"]
#     for name in names:
#         skip_connections.append(model.get_layer(name).output)
#     output = model.get_layer("pool4_conv").output

#     return output, skip_connections

def encoder2(inputs):
    num_filters = [32, 64, 128, 256]
    skip_connections = []
    x = inputs

    for i, f in enumerate(num_filters):
        x = conv_block(x, f)
        skip_connections.append(x)
        x = MaxPool2D((2, 2))(x)

    return x, skip_connections

def decoder2(inputs, skip_1, skip_2):
    num_filters = [256, 128, 64, 32]
    skip_2.reverse()
    x = inputs

    for i, f in enumerate(num_filters):
        x = UpSampling2D((2, 2), interpolation='bilinear')(x)
        x = Concatenate()([x, skip_1[i], skip_2[i]])
        x = conv_block(x, f)

    return x

def output_block(inputs):
    x = Conv2D(num_labels, (1, 1), padding="same")(inputs)
    x = Activation('sigmoid')(x)
    return x

def Upsample(tensor, size):
    """Bilinear upsampling"""
    def _upsample(x, size):
        return tf.image.resize(images=x, size=size)
    return Lambda(lambda x: _upsample(x, size), output_shape=size)(tensor)

def ASPP(x, filter):
    shape = x.shape

    y1 = AveragePooling2D(pool_size=(shape[1], shape[2]))(x)
    y1 = Conv2D(filter, 1, padding="same")(y1)
    y1 = BatchNormalization()(y1)
    y1 = Activation("relu")(y1)
    y1 = UpSampling2D((shape[1], shape[2]), interpolation='bilinear')(y1)

    y2 = Conv2D(filter, 1, dilation_rate=1, padding="same",
use_bias=False)(x)
    y2 = BatchNormalization()(y2)
    y2 = Activation("relu")(y2)

    y3 = Conv2D(filter, 3, dilation_rate=6, padding="same",
use_bias=False)(x)
    y3 = BatchNormalization()(y3)
    y3 = Activation("relu")(y3)

    y4 = Conv2D(filter, 3, dilation_rate=12, padding="same",
use_bias=False)(x)
    y4 = BatchNormalization()(y4)
    y4 = Activation("relu")(y4)

    y5 = Conv2D(filter, 3, dilation_rate=18, padding="same",
use_bias=False)(x)
    y5 = BatchNormalization()(y5)
    y5 = Activation("relu")(y5)

    y = Concatenate()([y1, y2, y3, y4, y5])

    y = Conv2D(filter, 1, dilation_rate=1, padding="same",
use_bias=False)(y)
    y = BatchNormalization()(y)
    y = Activation("relu")(y)

    return y

def build_model(shape):
    inputs = Input(shape)
    x, skip_1 = encoder1(inputs)
    x = ASPP(x, 64)
    x = decoder1(x, skip_1)
    outputs1 = output_block(x)

    x = inputs * outputs1

    x, skip_2 = encoder2(x)
    x = ASPP(x, 64)
    x = decoder2(x, skip_1, skip_2)
    outputs2 = output_block(x)
    outputs = Concatenate()([outputs1, outputs2])

    model = Model(inputs, outputs)
    return model

if __name__ == "__main__":
    num_labels=1
    model = build_model((576, 720, 3))
    model.summary()
    #print(model)
```

## Appendix B: Extract frames from video

```
import cv2
import os

# Opens the Video file
cap =
cv2.VideoCapture(r'C:\Users\reini\PycharmProjects\Groningen2\Data\Segmentat
ion_Robotic_Testing_GT\Dataset4\Segmentation.avi')
i = 0
while (cap.isOpened()):
    ret, frame = cap.read()
    if ret == False:
        break
    path = r'C:\Users\reini\PycharmProjects\Groningen2\Data\test\mask'
    cv2.imwrite(os.path.join(path, '3_' + str(i) + '.jpg'), frame)
    i += 1

cap.release()
cv2.destroyAllWindows()
```

## Appendix C: Alter grey values masks

```
from glob import glob
import cv2
import os

img_mask = r'C:\Users\reini\PycharmProjects\Groningen2\Data\mask\*.jpg'
img_names = glob(img_mask)
print(len(img_names))
i=0
for fn in img_names:
    print('processing %s...' % fn,)
    im_gray = cv2.imread(fn, 0)
    (thresh, im_bw) = cv2.threshold(im_gray, 60, 255, cv2.THRESH_BINARY |
cv2.THRESH_OTSU)
    thresh = 60
    im_bw = cv2.threshold(im_gray, thresh, 255, cv2.THRESH_BINARY)[1]
    path = r'C:\Users\reini\PycharmProjects\Groningen2\mask'
    cv2.imwrite(os.path.join(path, '1_' + str(i) + '.jpg'), im_bw)
    i += 1
```

## Appendix D: Model training

```
import os
import numpy as np
import cv2
import tensorflow as tf
from tensorflow.keras.layers import *
from tensorflow.keras.callbacks import *
from tensorflow.keras.optimizers import Adam, Nadam
from tensorflow.keras.metrics import *
from glob import glob
from sklearn.model_selection import train_test_split
# from model import build_model
# from utils import *
# from metrics import *

from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession

config = ConfigProto()
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)


def read_image(x):
    x = x.decode()
    image = cv2.imread(x, cv2.IMREAD_COLOR)
    image = np.clip(image - np.median(image) + 127, 0, 255)
    image = image / 255.0
    image = image.astype(np.float32)
    return image


def read_mask(y):
    y = y.decode()
    mask = cv2.imread(y, cv2.IMREAD_GRAYSCALE)
    mask = mask / 255.0
    mask = mask.astype(np.float32)
    mask = np.expand_dims(mask, axis=-1)
    return mask


def parse_data(x, y):
    def _parse(x, y):
        x = read_image(x)
        y = read_mask(y)
        y = np.concatenate([y, y], axis=-1)
        return x, y

    x, y = tf.numpy_function(_parse, [x, y], [tf.float32, tf.float32])
    x.set_shape([576, 720, 3])  # Dit aanpassen aan grootte van
afbeeldingen
    y.set_shape([576, 720, 2])  # 6 was eerst 2, even 6 van gemaakt als
test voor multiclass
    return x, y


def tf_dataset(x, y, batch=8):
    dataset = tf.data.Dataset.from_tensor_slices((x, y))
    dataset = dataset.shuffle(buffer_size=32)
    dataset = dataset.map(map_func=parse_data)
    dataset = dataset.repeat()
    dataset = dataset.batch(batch)
    return dataset


if __name__ == "__main__":
    np.random.seed(42)
    tf.random.set_seed(42)
    create_dir("files")

    train_path = "/content/gdrive/MyDrive/Machine_Learning_II/Double_U-
net/Reinier_stage/train"
    valid_path = "/content/gdrive/MyDrive/Machine_Learning_II/Double_U-
net/Reinier_stage/valid"

    ## Training
    train_x = sorted(glob(os.path.join(train_path, "image", "*.jpg")))
    train_y = sorted(glob(os.path.join(train_path, "mask", "*.jpg")))

    print(len(train_x))
    print(len(train_y))

    ## Validation
    valid_x = sorted(glob(os.path.join(valid_path, "image", "*.jpg")))
    valid_y = sorted(glob(os.path.join(valid_path, "mask", "*.jpg")))

    print(len(valid_x))
    print(len(valid_y))

    model_path = "/content/gdrive/MyDrive/Machine_Learning_II/Double_U-
net/files/model.h5"

    batch_size = 4
    epochs = 50
    lr = 1e-4
    shape = (576, 720, 3)  # Change this according to size of input image
    model = build_model(shape)

    metrics = [
        dice_coef,
        iou,
        Recall(),
        Precision()
    ]

    train_dataset = tf_dataset(train_x, train_y, batch=batch_size)
    valid_dataset = tf_dataset(valid_x, valid_y, batch=batch_size)

    # model.compile(loss='binary_crossentropy', optimizer=Adam(lr),
metrics=metrics)
    model.compile(loss=dice_loss, optimizer=Adam(lr), metrics=metrics)

    callbacks = [
        ModelCheckpoint(model_path),
        ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=4),
        CSVLogger("files/data.csv"),
        TensorBoard(),
        EarlyStopping(monitor='val_loss', patience=4,
restore_best_weights=True)
    ]
    train_steps = (len(train_x) // batch_size)
    valid_steps = (len(valid_x) // batch_size)

    if len(train_x) % batch_size != 0:
        train_steps += 1

    if len(valid_x) % batch_size != 0:
        valid_steps += 1

    model.load_weights(model_path)

    model.fit(train_dataset,
              epochs=epochs,
              validation_data=valid_dataset,
              steps_per_epoch=train_steps,
              validation_steps=valid_steps,
              callbacks=callbacks,
              shuffle=False)
```

## Appendix E: Training dependencies

```
import os
import numpy as np
import cv2
import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras.losses import binary_crossentropy

smooth = 1e-15
def dice_coef(y_true, y_pred):
    y_true = tf.keras.layers.Flatten()(y_true)
    y_pred = tf.keras.layers.Flatten()(y_pred)
    intersection = tf.reduce_sum(y_true * y_pred)
    return (2. * intersection + smooth) / (tf.reduce_sum(y_true) +
tf.reduce_sum(y_pred) + smooth)

def dice_loss(y_true, y_pred):
    return 1.0 - dice_coef(y_true, y_pred)

def iou(y_true, y_pred):
    def f(y_true, y_pred):
        intersection = (y_true * y_pred).sum()
        union = y_true.sum() + y_pred.sum() - intersection
        x = (intersection + smooth) / (union + smooth)
        x = x.astype(np.float32)
        return x
    return tf.numpy_function(f, [y_true, y_pred], tf.float32)

def bce_dice_loss(y_true, y_pred):
    return binary_crossentropy(y_true, y_pred) + dice_loss(y_true, y_pred)

def focal_loss(y_true, y_pred):
    alpha=0.25
    gamma=2
    def focal_loss_with_logits(logits, targets, alpha, gamma, y_pred):
        weight_a = alpha * (1 - y_pred) ** gamma * targets
        weight_b = (1 - alpha) * y_pred ** gamma * (1 - targets)
        return (tf.math.log1p(tf.exp(-tf.abs(logits))) + tf.nn.relu(-
logits)) * (weight_a + weight_b) + logits * weight_b

    y_pred = tf.clip_by_value(y_pred, tf.keras.backend.epsilon(), 1 -
tf.keras.backend.epsilon())
    logits = tf.math.log(y_pred / (1 - y_pred))
    loss = focal_loss_with_logits(logits=logits, targets=y_true,
alpha=alpha, gamma=gamma, y_pred=y_pred)
    # or reduce_sum and/or axis=-1
    return tf.reduce_mean(loss)
import os
import numpy as np
import cv2
import json
from glob import glob
#from metrics import *
from sklearn.utils import shuffle
from tensorflow.keras.utils import CustomObjectScope
from tensorflow.keras.models import load_model
#from model import build_model, Upsample, ASPP

def create_dir(path):
    """ Create a directory. """
```

```python
    try:
        if not os.path.exists(path):
            os.makedirs(path)
    except OSError:
        print(f"Error: creating directory with name {path}")

def read_data(x, y):
    """ Read the image and mask from the given path. """
    image = cv2.imread(x, cv2.IMREAD_COLOR)
    mask = cv2.imread(y, cv2.IMREAD_COLOR)
    return image, mask

def read_params():
    """ Reading the parameters from the JSON file."""
    with open("params.json", "r") as f:
        data = f.read()
        params = json.loads(data)
        return params

def load_data(path):
    """ Loading the data from the given path. """
    images_path = os.path.join(path, "image/*")
    masks_path  = os.path.join(path, "mask/*")

    images = glob(images_path)
    masks  = glob(masks_path)

    return images, masks

def shuffling(x, y):
    x, y = shuffle(x, y, random_state=42)
    return x, y

def load_model_weight(path):
    with CustomObjectScope({
        'dice_loss': dice_loss,
        'dice_coef': dice_coef,
        'bce_dice_loss': bce_dice_loss,
        'focal_loss': focal_loss,
        'iou': iou
        }):
        model = load_model(path)
    return model
    # model = build_model(256)
    # model.load_weights(path)
    # return model
```

## Appendix F: Split data

```python
from skimage import io
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import os
import random
import numpy as np
import cv2
from tqdm import tqdm
from glob import glob
import tifffile as tif
from sklearn.model_selection import train_test_split
# from utils import *

from albumentations import (
    PadIfNeeded,
    HorizontalFlip,
    VerticalFlip,
    CenterCrop,
    Crop,
    Compose,
    Transpose,
    RandomRotate90,
    ElasticTransform,
    GridDistortion,
    OpticalDistortion,
    RandomSizedCrop,
    OneOf,
    CLAHE,
    RandomBrightnessContrast,
    RandomGamma,
    HueSaturationValue,
    RGBShift,
    RandomBrightness,
    RandomContrast,
    MotionBlur,
    MedianBlur,
    GaussianBlur,
    GaussNoise,
    ChannelShuffle,
    CoarseDropout
)


def create_dir(path):
    """ Create a directory. """
    try:
        if not os.path.exists(path):
            os.makedirs(path)
    except OSError:
        print(f"Error: creating directory with name {path}")


def read_data(x, y):
    """ Read the image and mask from the given path. """
    image = cv2.imread(x, cv2.IMREAD_COLOR)
    mask = cv2.imread(y, cv2.IMREAD_COLOR)
    return image, mask
```

```
def augment_data(images, masks, save_path, augment=True):
    """ Performing data augmentation. """
    crop_size = (576 - 32, 720 - 32)
    size = (720, 576)

    for image, mask in tqdm(zip(images, masks), total=len(images)):
        image_name = image.split("/")[-1].split("\\")[0]
        mask_name = mask.split("/")[-1].split("\\")[0]
        folder_name = image.split("\\")[-1].split(".")[0]
        x, y = read_data(image, mask)

        try:
            h, w, c = x.shape
        except Exception as e:
            image = image[:-1]
            x, y = read_data(image, mask)
            h, w, c = x.shape

        if augment == True:
            ## Center Crop
            aug = CenterCrop(p=1, height=crop_size[0], width=crop_size[1])
            augmented = aug(image=x, mask=y)
            x1 = augmented['image']
            y1 = augmented['mask']

            ## Crop
            x_min = 0
            y_min = 0
            x_max = x_min + size[0]
            y_max = y_min + size[1]

            aug = Crop(p=1, x_min=x_min, x_max=x_max, y_min=y_min,
y_max=y_max)
            augmented = aug(image=x, mask=y)
            x2 = augmented['image']
            y2 = augmented['mask']

            ## Random Rotate 90 degree
            aug = RandomRotate90(p=1)
            augmented = aug(image=x, mask=y)
            x3 = augmented['image']
            y3 = augmented['mask']

            ## Transpose
            aug = Transpose(p=1)
            augmented = aug(image=x, mask=y)
            x4 = augmented['image']
            y4 = augmented['mask']

            ## ElasticTransform
            aug = ElasticTransform(p=1, alpha=120, sigma=120 * 0.05,
alpha_affine=120 * 0.03)
            augmented = aug(image=x, mask=y)
            x5 = augmented['image']
            y5 = augmented['mask']

            ## Grid Distortion
            aug = GridDistortion(p=1)
            augmented = aug(image=x, mask=y)
            x6 = augmented['image']
```

```python
        y6 = augmented['mask']

        ## Optical Distortion
        aug = OpticalDistortion(p=1, distort_limit=2, shift_limit=0.5)
        augmented = aug(image=x, mask=y)
        x7 = augmented['image']
        y7 = augmented['mask']

        ## Vertical Flip
        aug = VerticalFlip(p=1)
        augmented = aug(image=x, mask=y)
        x8 = augmented['image']
        y8 = augmented['mask']

        ## Horizontal Flip
        aug = HorizontalFlip(p=1)
        augmented = aug(image=x, mask=y)
        x9 = augmented['image']
        y9 = augmented['mask']

        ## Grayscale
        x10 = cv2.cvtColor(x, cv2.COLOR_RGB2GRAY)
        y10 = y

        ## Grayscale Vertical Flip
        aug = VerticalFlip(p=1)
        augmented = aug(image=x10, mask=y10)
        x11 = augmented['image']
        y11 = augmented['mask']

        ## Grayscale Horizontal Flip
        aug = HorizontalFlip(p=1)
        augmented = aug(image=x10, mask=y10)
        x12 = augmented['image']
        y12 = augmented['mask']

        ## Grayscale Center Crop
        aug = CenterCrop(p=1, height=crop_size[0], width=crop_size[1])
        augmented = aug(image=x10, mask=y10)
        x13 = augmented['image']
        y13 = augmented['mask']

        ##
        aug = RandomBrightnessContrast(p=1)
        augmented = aug(image=x, mask=y)
        x14 = augmented['image']
        y14 = augmented['mask']

        aug = RandomGamma(p=1)
        augmented = aug(image=x, mask=y)
        x15 = augmented['image']
        y15 = augmented['mask']

        aug = HueSaturationValue(p=1)
        augmented = aug(image=x, mask=y)
        x16 = augmented['image']
        y16 = augmented['mask']

        aug = RGBShift(p=1)
        augmented = aug(image=x, mask=y)
        x17 = augmented['image']
```

```
            y17 = augmented['mask']

            aug = RandomBrightness(p=1)
            augmented = aug(image=x, mask=y)
            x18 = augmented['image']
            y18 = augmented['mask']

            aug = RandomContrast(p=1)
            augmented = aug(image=x, mask=y)
            x19 = augmented['image']
            y19 = augmented['mask']

            aug = MotionBlur(p=1, blur_limit=7)
            augmented = aug(image=x, mask=y)
            x20 = augmented['image']
            y20 = augmented['mask']

            aug = MedianBlur(p=1, blur_limit=9)
            augmented = aug(image=x, mask=y)
            x21 = augmented['image']
            y21 = augmented['mask']

            aug = GaussianBlur(p=1, blur_limit=9)
            augmented = aug(image=x, mask=y)
            x22 = augmented['image']
            y22 = augmented['mask']

            aug = GaussNoise(p=1)
            augmented = aug(image=x, mask=y)
            x23 = augmented['image']
            y23 = augmented['mask']

            aug = ChannelShuffle(p=1)
            augmented = aug(image=x, mask=y)
            x24 = augmented['image']
            y24 = augmented['mask']

            aug = CoarseDropout(p=1, max_holes=8, max_height=32,
max_width=32)
            augmented = aug(image=x, mask=y)
            x25 = augmented['image']
            y25 = augmented['mask']

            images = [
                x, x1, x2, x3, x4, x5, x6, x7, x8, x9, x10,
                x11, x12, x13, x14, x15, x16, x17, x18, x19, x20,
                x21, x22, x23, x24, x25
            ]
            masks = [
                y, y1, y2, y3, y4, y5, y6, y7, y8, y9, y10,
                y11, y12, y13, y14, y15, y16, y17, y18, y19, y20,
                y21, y22, y23, y24, y25
            ]

        else:
            images = [x]
            masks = [y]

        idx = 0

        for i, m in zip(images, masks):
            i = cv2.resize(i, size)
            m = cv2.resize(m, size)

            tmp_image_name = f"{image_name}_{folder_name}_{idx}.jpg"
            tmp_mask_name = f"{mask_name}_{folder_name}_{idx}.jpg"

            image_path = os.path.join(save_path, "image/",
tmp_image_name).replace("\\", "/")
            mask_path = os.path.join(save_path, "mask/",
tmp_mask_name).replace("\\", "/")

            cv2.imwrite(image_path, i)
            cv2.imwrite(mask_path, m)

            idx += 1


def load_data(path, split=0.1):
    """ Load all the data and then split them into train and valid dataset.
"""
    img_path = glob(os.path.join(path, "Data/image", "*.jpg"))
    print(img_path)
    msk_path = glob(os.path.join(path, "Data/mask", "*.jpg"))

    total_train_x = []
    total_train_y = []
    total_test_x = []
    total_test_y = []
    total_valid_x = []
    total_valid_y = []
    idx = 0

    len_ids = len(img_path)
    train_size = int((80 / 100) * len_ids)
    valid_size = int((10 / 100) * len_ids)  ## Here 10 is the percent of
mask used for validation
    test_size = int((10 / 100) * len_ids)  ## Here 10 is the percent of
mask used for testing

    train_x, test_x = train_test_split(img_path, test_size=test_size,
random_state=42)
    train_y, test_y = train_test_split(msk_path, test_size=test_size,
random_state=42)

    train_x, valid_x = train_test_split(train_x, test_size=test_size,
random_state=42)
    train_y, valid_y = train_test_split(train_y, test_size=test_size,
random_state=42)

    total_train_x += train_x
    total_train_y += train_y
    total_test_x += test_x
    total_test_y += test_y
    total_valid_x += valid_x
    total_valid_y += valid_y

    return (total_train_x, total_train_y), (total_valid_x, total_valid_y),
(total_test_x, total_test_y)


## Skin lesion segmentation
```

```
def get_skin_lesion_data(path, split=0.1):
    total_train_x = glob(os.path.join(path, "New_Data/train/image/*"))
    total_train_y = glob(os.path.join(path, "New_Data/train/mask/*"))

    total_valid_x = glob(os.path.join(path, "New_Data/valid/image/*"))
    total_valid_y = glob(os.path.join(path, "New_Data/valid/mask/*"))

    total_test_x = glob(os.path.join(path, "New_Data/test/image/*"))
    total_test_y = glob(os.path.join(path, "New_Data/test/mask/*"))

    return (total_train_x, total_train_y), (total_valid_x, total_valid_y),
(total_test_x, total_test_y)


def main():
    np.random.seed(42)
    path = "/content/gdrive/MyDrive/Machine_Learning_II/Double_U-net"
    (total_train_x, total_train_y), (total_valid_x, total_valid_y),
(total_test_x, total_test_y) = load_data(path,

split=0.1)

    create_dir("New_Data/train/image/")
    create_dir("New_Data/train/mask/")
    create_dir("New_Data/valid/image/")
    create_dir("New_Data/valid/mask/")
    create_dir("New_Data/test/image/")
    create_dir("New_Data/test/mask/")

    augment_data(total_train_x, total_train_y, "new_data/train/",
augment=False)
    augment_data(total_valid_x, total_valid_y, "new_data/valid/",
augment=False)
    augment_data(total_test_x, total_test_y, "new_data/test/",
augment=False)


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

## Appendix G: Predict and evaluate segmentation

```
def evaluate_normal(model, x_data, y_data):
    THRESHOLD = 0.5
    total = []
    # i=0
    for i, (x, y) in tqdm(enumerate(zip(x_data, y_data)),
total=len(x_data)):
        x = read_image(x)
        y = read_mask(y)
        print(x.shape)
        _, h, w, _ = x.shape

        y_pred1 = parse(model.predict(x)[0][..., -2])
        y_pred2 = parse(model.predict(x)[0][..., -1])

        line = np.ones((h, 10, 3)) * 255.0

        all_images = [
            mask_to_3d(y_pred2) * 255
        ]
        # x[0] * 255.0, line,
        # mask_to_3d(y) * 255.0, line,
        # mask_to_3d(y_pred1) * 255.0, line
        mask = np.concatenate(all_images, axis=1)

        cv2.imwrite(f"results/{i}.jpg", mask)


def read_image(x):
    image = cv2.imread(x, cv2.IMREAD_COLOR)
    image = np.clip(image - np.median(image) + 127, 0, 255)
    image = image / 255.0
    image = image.astype(np.float32)
    image = np.expand_dims(image, axis=0)
    return image


def read_mask(y):
    mask = cv2.imread(y, cv2.IMREAD_GRAYSCALE)
    mask = mask.astype(np.float32)
    mask = mask / 255.0
    mask = np.expand_dims(mask, axis=-1)
    return mask


create_dir("results/")
evaluate_normal(model, test_x, test_y)

from PIL import Image, ImageChops
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

i = 0
create_dir("Concatinated/")
create_dir("Segmented/")
for n in range(0, 1122):
    im1path = "/content/gdrive/MyDrive/Groningen_test/Filmpje/image/"
    im1 = cv2.imread(os.path.join(im1path, '3_' + str(i) + '.jpg')) / 255
    print(os.path.join(im1path, '3_' + str(i) + '.jpg'))
    im2path = "/content/gdrive/MyDrive/Groningen_test/Filmpje/results/"
    im2 = cv2.imread(os.path.join(im2path, str(i) + '.jpg')) / 255
    im3 = im1 * im2
    im3 = im3 * 255

    line = np.ones((576, 10, 3)) * 255.0
    im1 = im1 * 255
    Concatinated_images = [im1, line, im3]
    Concat = np.concatenate(Concatinated_images, axis=1)
    cv2.imwrite(f"Concatinated/{i}.jpg", Concat)
    i += 1

    cv2.waitKey(0)
```

## Appendix H: Generate video from prediction

```
import cv2
import numpy as np
import glob

i = 0
img_array = []
path = '/content/gdrive/MyDrive/Groningen_test/Filmpje/Concatinated/'

for n in range(0, 1122):
    i += 0
    img = cv2.imread(os.path.join(path, f"{i}.jpg"))
    height, width, layers = img.shape
    size = (width, height)
    img_array.append(img)
    i += 1

out = cv2.VideoWriter('project.mp4', cv2.VideoWriter_fourcc(*'DIVX'), 15,
size)

for i in range(len(img_array)):
    out.write(img_array[i])
out.release()
```