Bachelor Thesis

Automated vessel detection for fetal surgery

Jonas Bohn

Jonas Lussi and Simone Gervasoni

Adviser

Prof. Dr. Bradley J. Nelson

Institute of Robotics and Intelligent Systems

Swiss Federal Institute of Technology Zurich (ETH)

2021

MSRL
MULTI-SCALE ROBOTICS LAB

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Preface

Through the lecture on microrobotics held by Prof. Dr. Nelson, I became attentive to the medical applications of robotics. I was quickly fascinated by the possibilities in this field of research and the broad variety of technologies used, that I wanted to get more experience in it.
When looking for a thesis, the search for a lab covering my interests was not very long. The MSRL was my first and single choice. Unfortunately, the virus delayed the start of my thesis, but I think it was worth the wait. The project fit my interest very well, as I am very interested in the possibilities of deep learning methods and wanted to get more proficient with coding. I am very glad to have worked on this exciting project and hope that my contributed work will be useful for the future and possibly help increase the life quality of twins affected by Twin-to-twin transfusion syndrome.

Special thanks go to:

- Prof. Dr. Bradley Nelson, who introduced me to the fascinating field of medical robotics and gave me the opportunity to work on this project at the MSRL.
- Jonas Lussi and Simone Gervasoni for all the valuable feedback and support.
- Everyone who proofread my thesis.

Keywords:
Deep learning, Vessel segmentation, Computer vision, Twin-to-twin transfusion syndrome, Fetal surgery, Video segmentation, Computer assisted interventions (CAI)

Abstract

Twin-to-twin transfusion syndrome is a potentially severe condition in pregnancies, where twins share a single placenta. The most promising treatment is fetoscopic laser coagulation, where the clinician localises and ablates abnormal vessels to balance the blood flow between the twin fetuses again. The surgical procedure is particularly challenging due to poor visibility, challenging lighting conditions, a limited field of view of the fetoscopic camera and low overall image quality.
Automatically detecting blood vessels can help to identify all abnormalities, which facilitates the surgery. A solution using deep-learning models that creates binary segmentation is proposed. The binary segmentation can be recombined with the fetoscopic video frames to enhance the surgeons view on the placental blood vessels directly. As part of this thesis, a new dataset of 270 fetoscopic video frames is created of data acquired from a human placenta in an ex vivo setting. The video frames were annotated by an undergraduate student and used to evaluate 26 different state-of-the-art segmentation networks through cross-validation.
The best performing deep learning models resulting from the cross-validations are optimised and qualitatively evaluated on a test set. The obtained architectures can produce segmentations with a high similarity to the ground-truth annotations even outperforming the annotations of the undergraduate student in some cases.
The optimised neural networks are further used to create segmentations on video sequences, where different modalities to enhance the view on the placental blood vessel are implemented.

Zusammenfassung

Das Fetofetale Transfusionssyndrom ist ein potenziell schwerwiegender Zustand bei Schwangerschaften, bei denen sich Zwillinge eine Plazenta teilen. Die vielversprechendste Behandlung ist die fetoskopische Laserkoagulation, bei der der Kliniker abnormale Gefäße lokalisiert und ablatiert, um den Blutfluss zwischen den Zwillingsfeten auszugleichen. Der chirurgische Eingriff ist aufgrund der schlechten Sichtverhältnisse, der schwierigen Lichtverhältnisse, des eingeschränkten Sichtfelds der fetoskopischen Kamera und der geringen Gesamtbildqualität eine besondere Herausforderung.
Die automatische Erkennung von Blutgefäßen kann helfen, alle Anomalien zu identifizieren, was die Operation erleichtert. Es wird eine Lösung mit Deep-Learning-Modellen vorgeschlagen, die eine binäre Segmentierung erstellt. Die binäre Segmentierung kann mit den fetoskopischen Videobildern rekombiniert werden, um dem Chirurgen einen direkten Blick auf die Blutgefäße der Plazenta zu ermöglichen. Ein neuer Datensatz von 270 fetoskopischen Videobildern wird aus Daten erstellt, die von einer menschlichen Plazenta in einer Ex-vivo-Einstellung erfasst wurden. Die Videobilder wurden von einem Studenten annotiert und werden zur Evaluierung von 26 verschiedenen Segmentierungsnetzwerken, die dem neuesten Stand der Technik entsprechen, durch Kreuzvalidierung verwendet.
Die leistungsfähigsten Deep-Learning-Modelle, die sich aus den Kreuzvalidierungen ergeben, werden optimiert und qualitativ auf einem Testset evaluiert. Die erhaltenen Architekturen können Segmentierungen mit einer hohen Ähnlichkeit zu den Ground-Truth-Annotationen erzeugen und übertreffen in einigen Fällen sogar die Annotationen des Studenten.
Die optimierten neuronalen Netze werden weiter verwendet, um Segmentierungen auf Videosequenzen zu erstellen, wobei verschiedene Modalitäten zur Verbesserung der Sicht auf das plazentare Blutgefäß implementiert werden.

Contents

Abstract **ii**

Zusammenfassung **iv**

List of Tables **vii**

List of Figures **viii**

Notation **ix**

1 Introduction **1**
1.1 Motivation 2
1.2 Goals 2

2 Medical Background **3**
2.1 Twin-to-Twin Transfusion Syndrome 3
2.2 Treatment 4

3 Related Work **6**

4 Method **8**
4.1 Preliminaries 8
4.2 Network Architectures 12

5 Experiments and Results **18**
5.1 Datasets 18
5.2 Data Augmentation 21
5.3 Evaluation Metrics 22
5.4 Model Evaluation 25
5.5 Optimisation 34
5.6 Results 39
5.7 Video Segmentation 41

6 Discussion **44**
6.1 Dataset 44

6.2 Networks 45
6.3 Video Segmentation 45

7 Conclusion **46**
7.1 Summary of the Results 46
7.2 Future Work 47

References **48**

A How-To Guide **55**

B Cross-Validation Tables **60**

C Optimisation Tables **68**

List of Tables

1 Cross-validation U-Net on UCL dataset 28
2 Cross-validation LinkNet on UCL dataset 29
3 Cross-validation PSPNet on UCL dataset 29
4 Cross-validation FPN on UCL dataset 30
5 Top results from cross-validation on UCL dataset 31
6 Cross-validation U-Net on MSRL dataset 32
7 Cross-validation LinkNet on MSRL dataset 32
8 Cross-validation PSPNet on MSRL dataset 33
9 Cross-validation FPN on MSRL dataset 33
10 Top results from cross-validation on MSRL dataset 34
11 Evaluation of loss functions 37
12 Evaluation of batch sizes 37
13 Evaluation of initial learning rates 39
14 Summary of the optimised models 40

List of Figures

1 Fetoscopic laser coagulation procedure for TTTS 1
2 Challenges in fetoscopic video 2
3 Solomon technique performed on chorionic plate 4
4 An example of semantic segmentation 8
5 Process of a convolution operation 9
6 Example of feature maps 9
7 Different pooling methods 10
8 Example of a transpose convolution operation 11
9 Transformation of a classification CNN to a FCN by replacing fully connected layers with convolutional ones 11
10 U-Net architecture 13
11 LinkNet architecture 14
12 Encoder and Decoder blocks of the LinkNet architecture 15
13 Feature Pyramid Network for semantic and instance segmentation 16
14 Decoder block of the FPN 16
15 PSPNet architecture 17
16 UCL placental dataset 19
17 Setup for placental image acquisition 20
18 MSRL placental dataset 20
19 Data augmentation on MSRL dataset 21
20 Prediction classes for segmentation metrics 22
21 Visualization of the IoU score 24
22 First segmentation results on the UCL dataset 25
23 Segmentation of UCL model on MSRL data 26
24 Effect of different learning rates 38
25 Segmentation results of the optimised networks 41
26 Qualitative comparison of FPN-EfficientNetB5 with the U-Net-ResNet101 41
27 Comparison of FPN-EfficientNetB5 with the U-Net-ResNet101 as a baseline 42
28 Segmentation results of FPN-EfficientNetB5 on test images compared to a baseline architecture 42
29 Segmentation with FPN-EfficientNetB5 on a video frame 43

30 Visualisation methods of predictions on fetoscopic video 43
31 Pixel annotation interface 55
32 Binary segmentation 60

Notation

Acronyms and Abbreviations

MSRL	Multi-Scale Robotics Lab
BCE	Binary Cross-Entropy
CNN	Convolutional Neural Network
FCN	Fully Convolutional Network
FPN	Feature Pyramid Network
GPU	Graphics Processing Unit
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoU	Intersection over Union
PSPNet	Pyramid Scene Parsing Network
ReLU	Rectified Linear Unit
TTTS	Twin-to-Twin Transfusion Syndrome
UCL	University College London

1 Introduction

Twin-to-twin transfusion syndrome (TTTS) is a rare complication during monochorionic twin pregnancies, affecting identical twins that share a placenta [1]. It is caused by abnormal vascular connections on the placenta that disproportionately transfer blood from one twin, the donor to the other, the recipient. If left untreated, this chronic imbalance in blood flow can lead to severe problems for both twins, including cardiac complications, long-term neurodevelopmental impairment, and death in either twin [1, 2]. Several options for TTTS treatment exist, but fetoscopic laser coagulation has proven to be superior to other methods in most cases [3]. Fetoscopic laser coagulation is a minimally invasive procedure that uses a specialised endoscopic camera, a fetoscope, with a laser tool shown in Figure 1. The surgeon inserts the fetoscope into the amniotic cavity, visualizes the chorionic plate of the placenta, and coagulates the twins abnormal vascular connections on the plate [1].

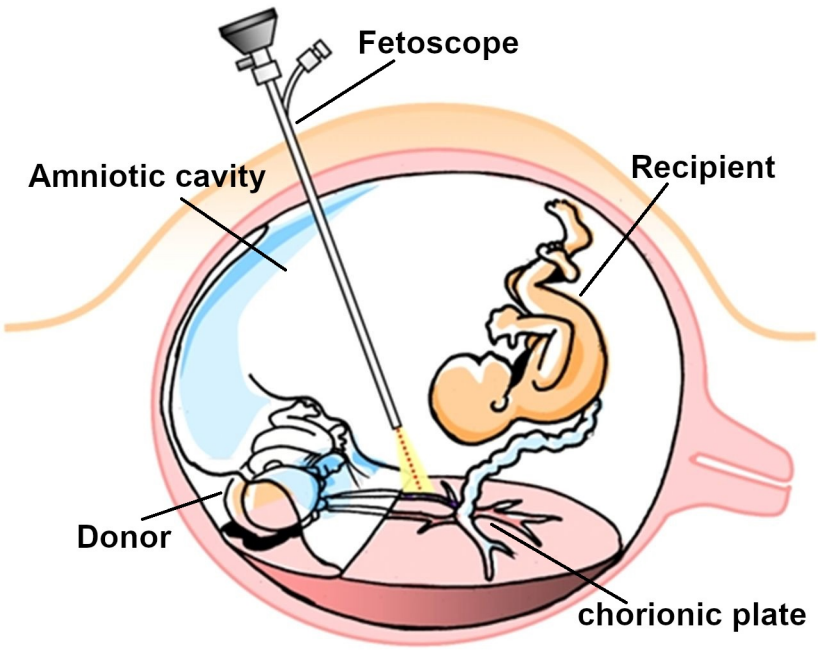

Figure 1: Fetoscopic laser coagulation procedure for TTTS [4]

1.1 Motivation

Even though fetoscopic laser coagulation decreases the risks of complications, several challenges can occur during surgery.
Today, there is no possibility of imaging the problematic blood vessels preoperatively, which means that the surgeon must identify them during the operation [5]. Limited field-of-view, poor visibility due to turbidity in the amniotic fluid, occlusion, and illumination changes impede the surgeons ability to identify all blood vessels accurately on the image provided by the fetoscope, as seen in Figure 2. Not identifying all blood vessels can lead to incomplete ablations of the problematic vascular connections, which can further result in persistent TTTS [1].

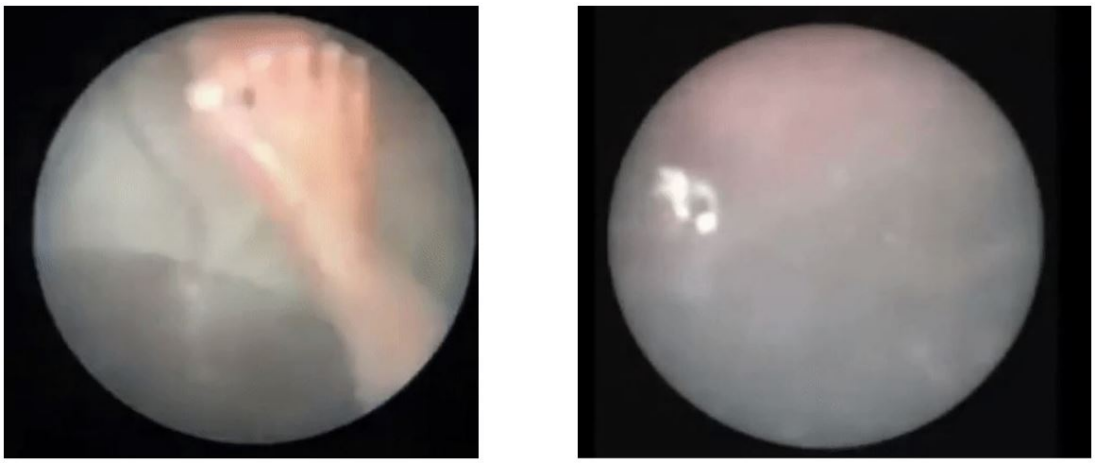

Figure 2: Challenges in fetoscopic video [6]

1.2 Goals

This thesis aims to come up with an approach to identify blood vessels during fetal surgery automatically. It should assist the surgeon during the treatment to ensure that all important blood vessels are detected and can be ablated. The main approach of this thesis is the usage of deep convolutional neural networks to increase the safety of the fetoscopic laser coagulation treatment.

2 Medical Background

2.1 Twin-to-Twin Transfusion Syndrome

Twin-to-twin transfusion syndrome (TTTS) occurs in monochorionic diamniotic pregnancies, which means that the twins share a single placenta and are in different amniotic sacs. Monochorionic diamniotic (MoDi) pregnancies account for around 70% of identical twin pregnancies, where 15-20 % of them are affected by TTTS [8, 7].
The anatomical prerequisite of TTTS development is the intertwin vascular connections resulting from the monochorionic placenta. Those anastomoses connect the two fetal circulations and allow blood to pass from one twin to another with a naturally balanced blood flow interchange [1]. There exist three types of placental anastomoses that exchange blood between the twins:

- Arterio-venous (AV), the most numerous group with a unidirectional blood flow existing in both directions from either fetus to the other [9].
- Veno-venous (VV) normally work as bidirectional vessels but can be unidirectional if differences in blood pressure between the fetuses occur [1].
- Arterio-arterial (AA) work the same way as VV anastomoses, but they can additionally compensate blood flow imbalances caused by AV anastomoses [1].

The number and size of those anastomotic vessels can vary between different placentas and influence the twins blood flow interchange. A predominance in unidirectional AV anastomoses from the donor to the recipient twin with a shortage of AA vessels leads to a chronic imbalance in blood flow known as TTTS [1]. This imbalance results in slower growth and development of the donor twin, reducing the urinary output due to dehydration leading to a lower amount of amniotic fluid [8]. The recipient twin experiences the opposite effects, with faster growth and a higher amount of amniotic fluid due to increased urinary output. The difference in sizes of the fetuses and amniotic sac is illustrated in Figure 1.
If left untreated, the condition results in death for both fetuses in most cases due to cardiac failure, kidney failure, or extremely preterm delivery [3, 1].

To avoid severe complications during pregnancy, early detection of TTTS is crucial to allow for timely treatment [1]. Therefore, all monochorionic twin pregnancies are recommended to schedule an ultrasound in biweekly intervals, where a list of diagnostic criteria is checked to precisely distinguish whether or not TTTS is present [1].

2.2 Treatment

TTTS usually diagnosed between the 16th and 26th week of pregnancy, where the condition is classified into a severeness stage proposed by Quintero et al. [3, 10]. Five stages exist, where the risk of death in either twin increases within each stage [1]. From stage two upwards minimally invasive laser treatment, called fetoscopic laser coagulation, has shown to be superior to other existing methods and increased the survival rate for at least one twin significantly [3]. The goal is to coagulate all the placental anastomoses to achieve a separate vascular system for each of the twins. The state of the art laser technique was initially proposed by Slaghekke et al. [11], the Solomon technique, which reduced complications after surgery compared to alternative laser techniques. This method focuses on coagulating the whole vascular equator on the chorionic plate [11] pictured in Figure 3.

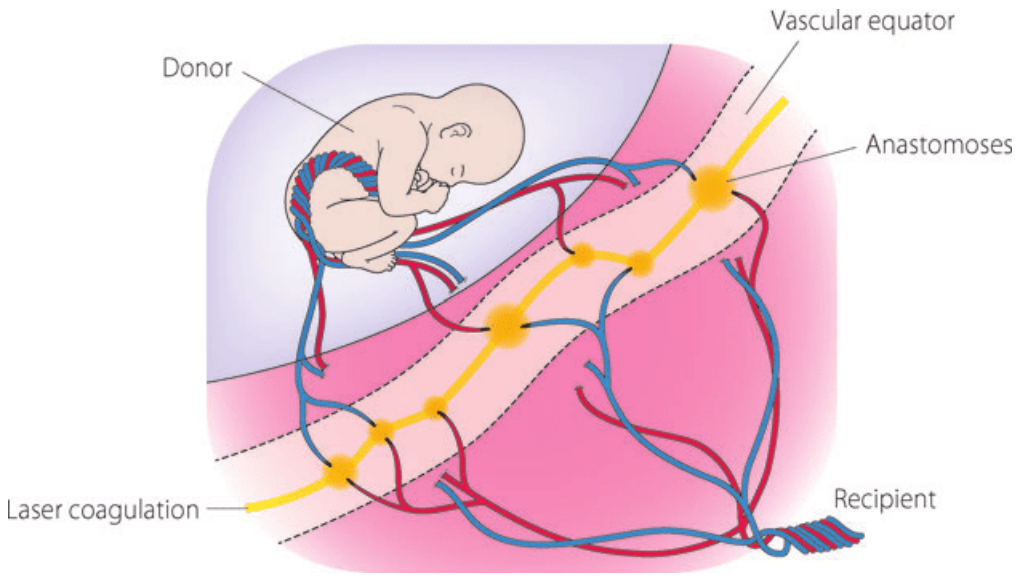

Figure 3: Solomon technique performed on chorionic plate [12]

Bamberg et al. [3] express concerns on lasering placental tissue without any anastomoses resulting in an increased placental injury and a higher risk of placental abruption. They propose to use a partial Solomon technique by only coagulating areas along the placental equator with detectable anastomoses, to reduce the damage of placental tissue. This again increases the probability of missing out an important blood vessels, which makes exact detection vital.
Even though the mean survival rate of both twins has increased from 35% to 65% and for at least one twin from 70% to 88% [13], fetoscopic laser treatment remains an invasive surgery with some non-negligible risks that may be improved by technical progress in the field of visualization and coagulation of all abnormal anastomoses [3].

3 Related Work

The analysis of blood vessels is important in multiple medical fields such as ophthalmology [14], oncology and neurosurgery, both for diagnosis and treatment [15]. While manual segmentation and detection of blood vessel remain expensive in terms of time and efficiency, automatic or semi-automatic blood vessel segmentation could optimize the procedure and assist clinicians in their daily work [15].

Moccia et al. [15] review multiple states of the art approaches for automatic or semi-automatic vessel segmentation categorised in vessel enhancement through filtering, machine learning, model-based segmentation and gradient-based approaches. With increasing computational power through the use of Graphics Processor Units (GPU), deep learning algorithms are becoming strongly popular and leading in several segmentation tasks [15]. Although the technology is up-and-coming in medical applications, the field of use is still limited to some specific anatomical regions due to the lack of publicly available image datasets with Gold Standard (GS) [15]. Especially the retinal region has multiple high-quality databases including STARE [16] and DRIVE [17], resulting in an increased scientific output on retinal blood vessel segmentation [15].

However, the region of interest of this thesis is the segmentation of blood vessels in the placental region during TTTS treatment, where the scientific output is lower than in the retinal field. Gaisser et al. [18] proposed to apply deep convolutional neural networks, which outperformed standard feature extraction algorithms for in-vivo images by far. The used network architecture is based on the Single Shot Multibox Detector (SSD), a region-based convolutional neural network proposed by Liu et al. [19]. This network performed especially well in good visibility conditions, but could hardly detect any vessels in difficult lighting conditions [18], demonstrating that the network was not robust to variations during surgery.

Based on the findings in [18], Sadda et al. [5] proposed a new network architecture that builds upon the U-Net described by Ronneberger et al. [20]. The network is trained on 345 non publicly available fetoscopic video frames se-

lected from ten surgeries performed at the Yale New Haven Hospital. An expert clinician manually labelled the selected video frames for training to ensure correctness of the training data. All the videos have been recorded with a Storz miniature 11540AA endoscope and add up to approximately five hours of in vivo video [5]. The authors compare their deep-learned segmentation method with the performance of a state of the art vesselness filter, the Frangi filter [21], and the segmentation of a novice human rater. The results demonstrate that deep-learned models perform better than regular filtering methods or novice human raters, in detecting blood vessels from fetoscopic videos [5].

While in [5] only one network architecture was tested, Bano et al. [6] searched for a better-performing architecture in segmenting blood vessels. With their work, they also released the first publicly available placenta dataset including 483 frames with ground-truth vessel segmentation extracted from six different fetoscopic procedures. The dataset is used to compare different U-Net [20] architectures with the baseline network proposed by Sadda et al. [5]. This proposed network is strongly outperformed in cross-validation by all of the networks used by Bano et al. [6], where the U-Net with ResNet101 [22] backbone performs best.

4 Method

In this chapter, the preliminaries are presented, and different network architectures that were implemented are described.

4.1 Preliminaries

The task of detecting placental blood vessels can be categorised under semantic segmentation, one of the key problems in the field of computer vision. Semantic segmentation consists of labelling each pixel of an image with a class of what is represented as shown in Figure 4.

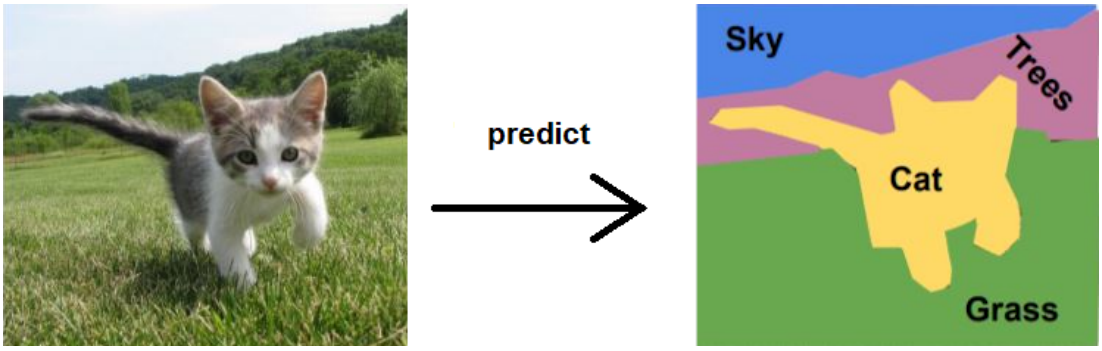

Figure 4: An example of semantic segmentation [24]

This task has gained importance in autonomous driving, augmented reality and medical image analysis, where deep learning methods are outperforming others [23]. Currently, the most successful technique in semantic segmentation is the application of convolutional neural networks (CNNs). Most of the best-performing CNNs in segmentation problems are based on the idea of fully convolutional networks (FCN) by Long et al. [25, 23] which was a milestone in semantic segmentation and is described in Section 4.1.4. In principle, all CNNs, including the FCN, are built upon a set of different layers and tools. The most important ones are briefly explained in Section 4.1.1 - 4.1.3.

4.1.1 Convolutional Layer

The convolutional layer is the core building block for all CNNs. The element used to carry out the convolution operation is called a kernel or filter illustrated

in Figure 5. This operation aims to extract features such as edges or lines with repeated application of filters on the input data, mostly RGB images, which results in multiple new feature maps.

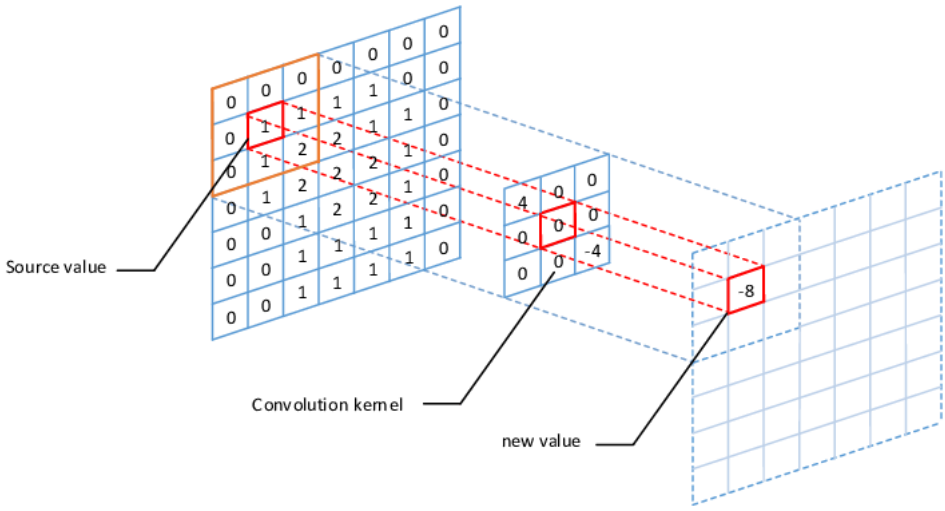

Figure 5: Process of a convolution operation [31]

Usually, more than one convolutional layer is used in CNNs, where the first few layers of such CNNs are responsible for extracting low-level features such as edges or colour gradients. With additional layers, the architecture adapts to high-level features as animals, faces or blood vessels [32]. Some examples of how the resulting feature maps after each convolutional layer look like are illustrated in Figure 6, where in reality multiple of those feature maps exist at each layer of the network. It can be observed that the feature maps lose resolution at each layer, which is done by pooling layers described in Section 4.1.2.

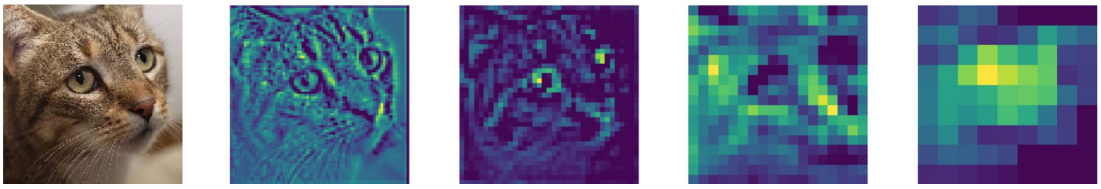

Figure 6: Example of feature maps [58]

4.1.2 Pooling Layer

Pooling layers are often applied between convolutional layers reducing the dimensionality of the input data, also known as downsampling. Reducing the spatial

size of the images decreases the number of parameters which leads to less required computational power to process the data [24]. Additionally, it is useful to robustly extract dominant higher-level features from images [32].
Different pooling methods exist, where the max pooling and average pooling operations are the most commonly used [24]. Max pooling returns the maximum value inside the predefined kernel, while the average pooling operation is returning the mean value of all the values inside the kernel as shown in Figure 7. The max-pooling currently is the favoured method over the average pooling due to its noise-suppressing functionality and overall better performance [32].

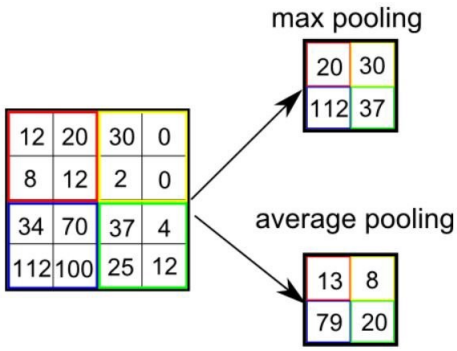

Figure 7: Different pooling methods [32]

4.1.3 Transpose Convolutional Layer

Producing high-resolution segmentation maps requires the network to upsample the low-resolution feature maps resulting from the pooling operations. Several predefined interpolation methods exist for this task such as Nearest Neighbor, Bed of Nails, bilinear transformation, or Max Unpooling [24]. As these methods are predefined mathematical operations, they do not learn and adapt from data to optimize the output. However, transpose convolution is a method that can learn how to upsample optimally with learnable parameters. As the name indicates, the transposed convolution essentially does the opposite to the typical convolution explained in Section 4.1.1 and the pooling layers. For a transpose convolution, a single value in a low-resolution feature map is taken and multiplied with a kernel projecting the weighted values to a higher-resolution output feature map [24]. Figure 8 shows an example of how the transpose convolution operation works to upsample a specific feature map.

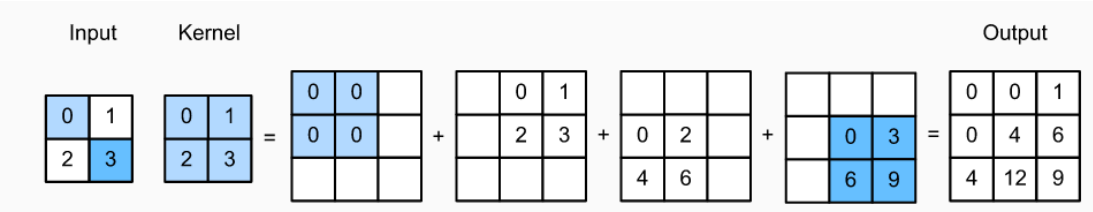

Figure 8: Example of a transpose convolution operation [33]

4.1.4 Fully Convolutional Networks

The success of regular CNNs in image classification or object detection tasks has motivated researchers to progress from a coarse to a fine prediction. While existing CNNs were outputting classification scores for the image, Long et al. [25] wanted the produced output to be the same size as the input with pixel-wise predictions. The proposed fully convolutional network takes advantage of powerful classification networks like AlexNet [27], GoogLeNet [28] and VGG16 [29], that serve as the feature extractor of the network [25], where any classification network available could be used. The authors transformed those networks into FCNs by replacing the last layer with convolutional ones resulting in a low-resolution feature map instead of classification scores [25, 23] illustrated in Figure 9.

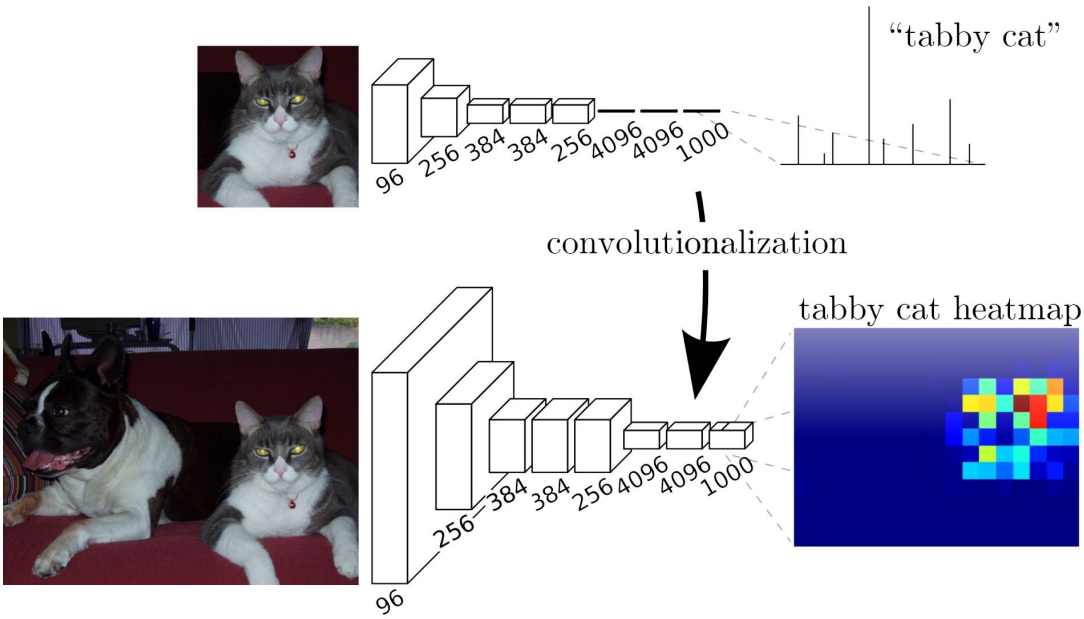

Figure 9: Transformation of a classification CNN to a FCN by replacing fully connected layers with convolutional ones [25]

The map is then upsampled using transposed convolutions, which results in a FCN that can be trained end to end to produce a segmentation map for the whole input data [25]. This approach is considered a milestone in semantic segmentation since it showed how CNNs could be trained end to end and make predictions on inputs with arbitrary size [23]. Despite its improvements on segmentation datasets, it is still known to mainly produce low-resolution segmentation maps due to the downsampling while encoding the image. The extensive downsampling is done to reduce the computational power needed and is done by pooling layers described in Section 4.1.2. The difficulty in producing high-resolution segmentation maps is learning how to decode the low-resolution maps to achieve high pixel-level accuracy [23].

4.2 Network Architectures

Based on the findings in Long et al. [25], multiple advanced architectures have been created over the past years, that present different ways of decoding the feature maps. This section presents some of the most successful network architectures in image segmentation of medical or natural images. These architectures act as the base for all the networks used in this thesis.

4.2.1 U-Net

In 2015 Ronneberger et al. [20] proposed a new architecture, the U-Net, that was built upon the FCN [25]. The U-Net is a traditional encoder-decoder style network, where the encoder captures the context of the image, and the decoder enables precise localization with pixel-wise classification. Compared to the FCN, the most important modification was to expand the capacity of the decoder by extending the numbers of feature channels. This alteration leads to a symmetric path to the encoder, which gives the network its characteristic u-shaped architecture displayed in Figure 10.
The encoder follows a simple CNN architecture, where a specific form is implemented in the original paper. However, as with the FCN, any classification model could be used to encode the image. The original work proposed a U-Net with repeated application of two convolutional layers with a 3x3 sized kernel each followed by a rectified linear unit (ReLU) and a 2x2 max pooling layer for down-

sampling [20]. The vanilla U-Net is five stages deep, where the number of feature channels is doubled in every stage until the bottleneck is reached.

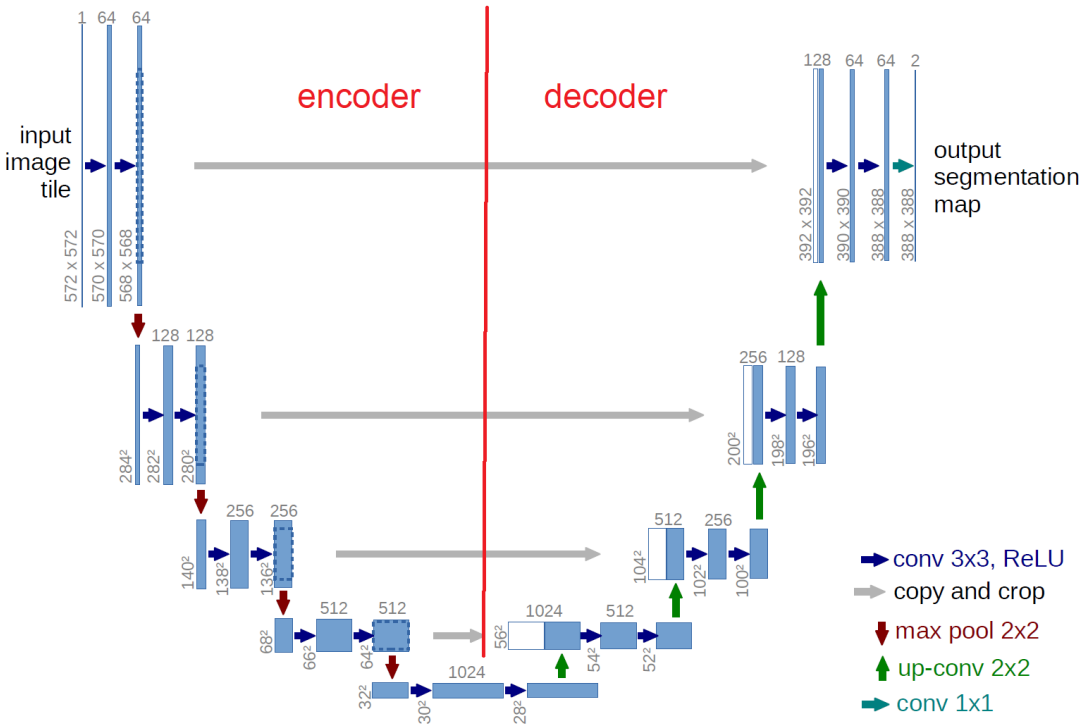

Figure 10: U-Net architecture [20]

The decoder part of the U-Net consist of repeated application of a 2x2 transpose convolution, concatenation with the same sized feature map of the encoder, and two 3x3 convolutional layers, each followed by a ReLU [20]. The concatenation, marked by the white boxes in the decoder, is also referred to as a skip connection. This skip connection is used to reconstruct segmentation boundaries and finer details that have been lost while downsampling [23]. At the final layer of the network is a 1x1 convolution that reduces the feature channels and outputs the predicted segmentation map.

4.2.2 LinkNet

The LinkNet is built very similar to the U-Net. Chaurasia et al. [34] proposed this novel architecture to use in real-time applications, where processing time is an important factor. The best segmentation models generally use extensive models as the encoder, which are winners of the ImageNet Large Scale Visual Recognition

Challenge (ILSVRC) [26, 34]. These models perform very well in segmentation task in terms of accuracy but are mostly too slow to actually be used in real-time applications [34]. The LinkNet, illustrated in Figure 11, addresses this problem with the application of a much smaller encoder, the ResNet18 [22], which uses significantly fewer parameters while achieving state-of-the-art accuracy on popular segmentation dataset [34].

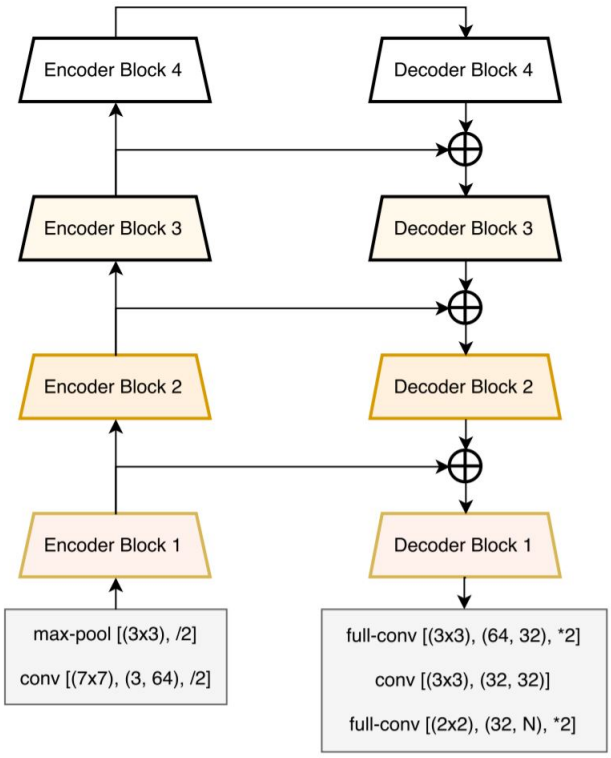

Figure 11: LinkNet architecture [34]

The initial block of the LinkNet architecture performs a 7x7 convolution followed by a 3x3 max pooling layer on the input data. Four equivalent encoder blocks then follow this block. Each encoder block uses residual blocks [22] which perform downsampling with convolutions [34] and are represented in Figure 12 on the left. The variables m and n are indicating the input and output feature maps size.

The decoding block consists of 1x1 convolutions and a 3x3 transposed convolution for upsampling the image illustrated in Figure 12 on the right, which is basically the same way as the U-Net does the encoding. The novelty in the LinkNet lies in how the encoder and decoder are linked, thus the name. The LinkNet uses the

addition operation instead of concatenation as skip connections, which reduces the processing time and increases the efficiency [34].

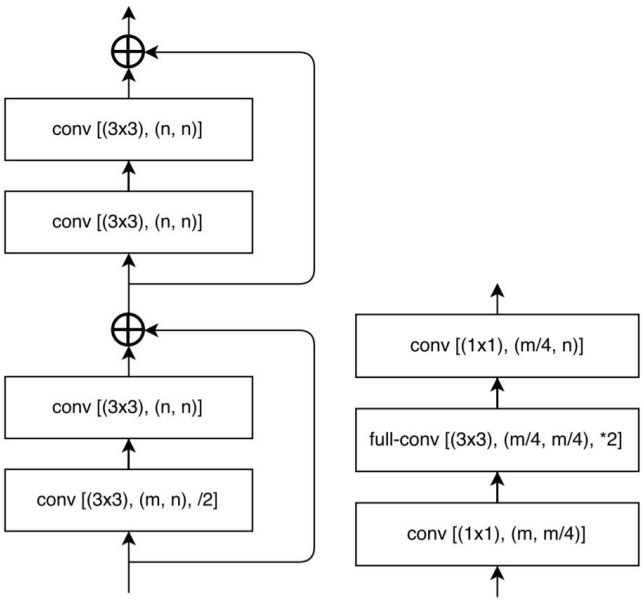

Figure 12: Encoder and Decoder blocks of the LinkNet architecture [34]

4.2.3 Feature Pyramid Network

The Feature Pyramid Network (FPN) was proposed by researchers from the Facebook AI Research to solve both detection and segmentation tasks [35]. While the U-Net [20] and the LinkNet [34] only produce semantic feature maps at the final stage, Lin et al. [35] built a network that makes predictions on all stages of the network. Based on this structure Kirillov et al. [38] developed a FPN especially for semantic or instance segmentation illustrated in Figure 13.

The basic structure of the FPN is very similar to the U-Net or LinkNet, as it has a type of encoder-decoder style, where the encoder can be basically any classification network (e.g. VGG [29], ResNet [22], EfficientNet [36]). The backbone used by the authors is the ResNet architecture in different sizes, which achieves state-of-the-art accuracy on the biggest segmentation and detection challenges [35]. To decode the low-resolution feature map, the nearest neighbour method is used on each stage which upsamples the maps by two. The upsampled map is then merged with the encoder's corresponding feature map by addition after the encoder map dimension is reduced through a 1x1 convolution, which can be

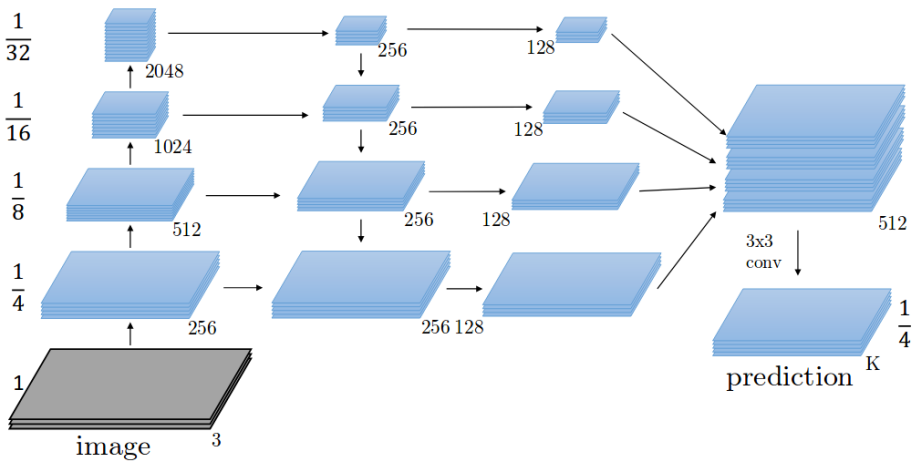

Figure 13: Feature Pyramid Network for semantic and instance segmentation [38]

seen in Figure 14. Unlike other encoder-decoder architectures, two 3x3 convolutional layers are then applied to the merged feature maps in each stage, shown in Figure 13. This convolution operation helps minimize aliasing effects during upsampling the feature maps [35]. The stages are then upsampled to concatenate the maps with low-resolution and strong features with the high-resolution and weak features maps [38]. Another convolution and upsampling are applied to the concatenated maps, which outputs the final segmentation map with K classes as shown in Figure 13.

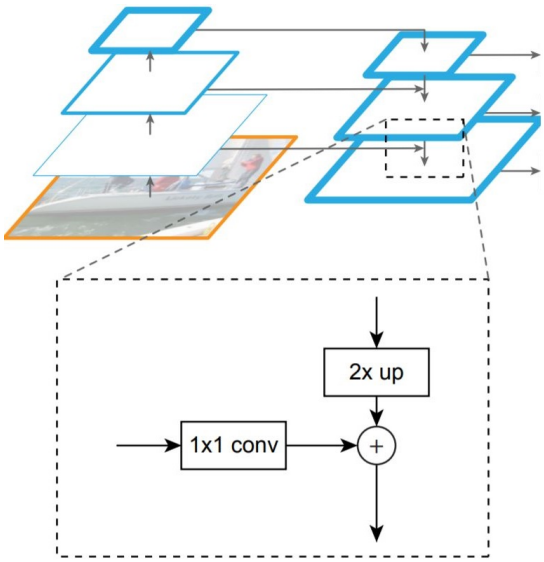

Figure 14: Decoder block of the FPN [35]

4.2.4 Pyramid Scene Parsing Network

The Pyramid Scene Parsing Network (PSPNet) [39] was built upon observations of failure cases when applying the FCN [25] for scene understanding. Zhao et al. [39] observed that the FCN had trouble collecting contextual information on the global scene. It did not understand visual patterns that coexist. For example, a boat is likely to be in a port or piloting on the water, while not flying in the sky. The main goal of the PSPNet is to capture global contextual information and transfer it into a more accurate pixel-wise prediction [39]. The network uses FCN [25], and DeepLab [40] as baseline architectures and introduces the characteristic pyramid pooling module for global scene understanding [39]. The proposed PSPNet is illustrated in Figure 15, which shows the workflow of the network end-to-end. In a first step, as shown in Figure 15(a), an encoder network creates feature maps that are 1/8 of the size of the input image, where the authors used a pre-trained ResNet with the application of dilated convolutions to extract the feature maps [39]. Figure 15(b) shows the core of the net, where the feature map is pooled and separated into different sub-regions of the feature map. The authors used the average pooling methods, which led to better results in their test. The kernels used for pooling cover the whole, half of, and small portions of the image which generates the pyramidal form [39]. A 1x1 convolution is used on each level to reduce the dimension, which ensures that the weights of the global features are maintained. All pyramidal feature maps are then upsampled, using bilinear interpolation, to the original feature map size in Figure 15(a). The different feature maps are then concatenated with the original map followed by a convolutional layer that creates the final segmentation map as pictured in Figure 4.2.4(c).

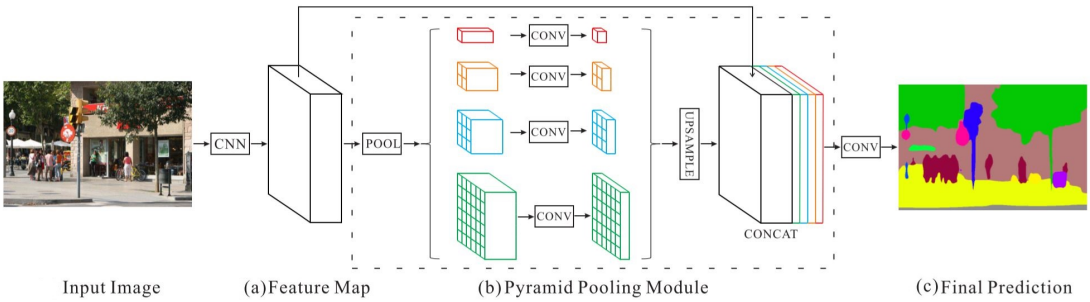

Figure 15: PSPNet architecture [39]

5 Experiments and Results

To evaluate the performance of different state-of-the-art segmentation models, experiments were conducted. All experiments are implemented in Keras [60] with Tensorflow backend using a single Nvidia TITAN Xp 12 GB. In this chapter, the experimental setup of the network evaluation is explained, and results are shown.

5.1 Datasets

The key to getting better in most fields of life is practice. This applies to deep learning, as well. Deep learning methods learn with datasets that are built to train a particular problem, where this thesis focuses on placental blood vessel segmentation. As mentioned in Section 3, there is work that uses placental datasets, but most of this data is not released to the general public or does not adapt to the problem well.
Section 5.1.1 and 5.1.2 discuss the available fetoscopic placental datasets used to perform the experiments, published by the University College in London (UCL) and a self-created dataset for the specific problem of this thesis.

5.1.1 UCL Dataset

As part of their work on fetoscopic mosaicking, Bano et al. [6] published the first publicly available placental vessel segmentation dataset. The dataset consists of 482 annotated video frames from six different in vivo fetoscopic laser surgeries to treat TTTS. The different surgeries video frames varied greatly in terms of quality and overall image appearance due to variation in intra-operative environment, which resulted in increased variability in the data as can be seen on the sample images in Figure 16. To focus solely on identifying placental blood vessels, a separate fetoscopic event identification network [43] selects all the video frames with a clear view on the blood vessel without any occlusions or tools present. To avoid including very similar video frames, the clear view sequences have been down-sampled from 25 to 1 fps, and each selected frame was resized to 448x448 pixel resolution. The resized video frames have then been used to create binary ground-truth masks created with a pixel annotation tool [44] and verified by a fetal medicine specialist [6].

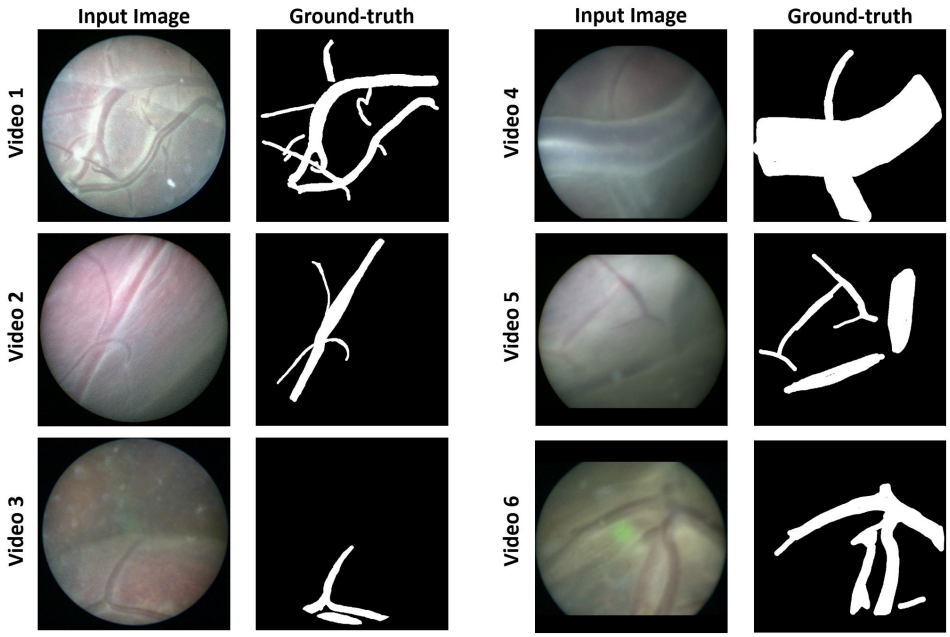

Figure 16: UCL placental dataset [6]

5.1.2 MSRL Dataset

The blood vessel segmentation created in this thesis is part of a larger project by the Multi-Scale Robotics Lab (MSRL) and has to work on the newly created fetoscope, which uses different imaging techniques than the ones used in [6]. Building a deep-learning model that works on the new fetoscope also requires training data obtained by the new fetoscope, which resulted in a new dataset built for this project. Due to the limited quantities available by the University Hospital in Zurich, only a single placenta was used to create the new dataset. The data was acquired in an ex vivo setting, where an opaque box was used to simulate the dark environment of the TTTS surgery, illustrated in Figure 17. Approximately 20 minutes of video with RGB colour channels were collected filming different regions on the placenta to get as much variety in the data as possible. The video frames were acquired through screen recording from six videos of different sizes and different fps rates. The actual used fetoscopic camera creates images with a resolution of 400x380 pixels, that are upsampled by the computer to much larger images. For the final dataset, video frames have been extracted at a rate of 3 fps, and the far left and right of the frames were cropped resulting in a square of resolution 784x784 pixels using the FFmpeg software [45].

A total of 270 video frames were then hand-selected to create a new dataset with different shaped blood vessels. Unlike the dataset in Section 5.1.1, the ablation tool is omnipresent on the left bottom corner in all extracted video frames, which made exact identification of the blood vessel harder. The uniformly resized video frames were then used to create binary ground-truth masks using the PixelAnnotationTool [44]. The created ground-truth masks have been hand-labelled by the author of this thesis, a novice in segmenting placental blood vessels, using a tablet with the matching stylus pen. All the extracted and labelled video frames from the six recorded videos resulted in this new dataset, with some sample images shown in Figure 18.

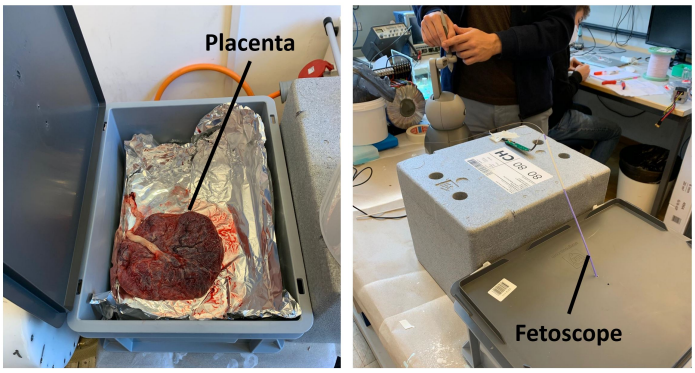

Figure 17: Setup for placental image acquisition

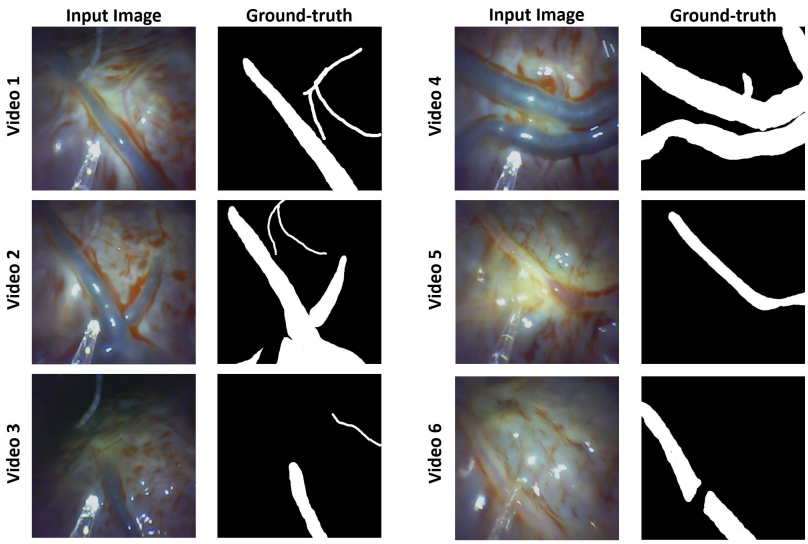

Figure 18: MSRL placental dataset

5.2 Data Augmentation

Having a significant variability of images to learn from lets the model adapt much better to new data. Overfitting is a problem caused by having too few samples to learn from, which means that the model only works well on the training data. Data augmentation takes the approach of generating more training data with the existing, by applying image transformations that result in reasonable training data. Augmenting the data achieves that the model never sees the same picture twice, which helps the model learn more aspects and generalize better to new data and reduce overfitting to the dataset. Additionally a variety of conditions can be recreated which may not exist in the dataset but could occur in the target application such as different lighting conditions, orientations or scales.
The image transformations applied are central and random zooming, random rotation, distortion, horizontal and vertical flip, illustrated in Figure 19.

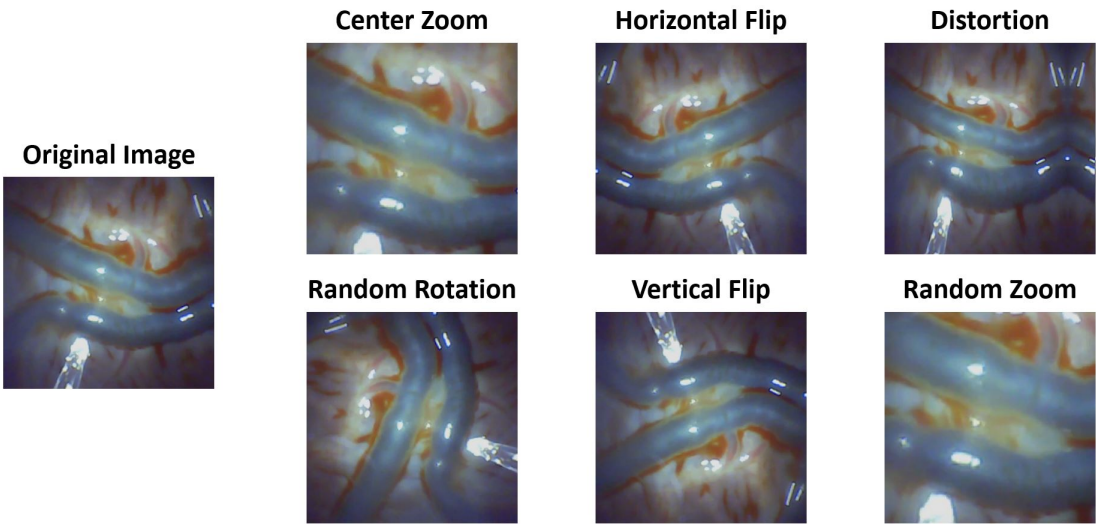

Figure 19: Data augmentation on MSRL dataset

There exist two different ways of augmenting the data in deep learning applications. The first is the augmentation of the whole dataset that increases the amount before training, and the second is an in-place method that continuously transforms images during training. In this thesis, both techniques have been applied where no significant change in metrics could be observed in any experiment performed. The chosen method in the following experiments is the augmentation before training, as it is easier to visualise the results after the transformations.

5.3 Evaluation Metrics

Metrics are used to evaluate and compare the performance of segmentation models, but not all metrics indicate how well the models actually perform. A right selection of metrics can save much time evaluating and tuning deep learning models to achieve the best results possible.
The pixel-wise predictions in segmentation tasks can be categorised into four classes, which are used to calculate all the metrics: true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN) illustrated in Figure 20.

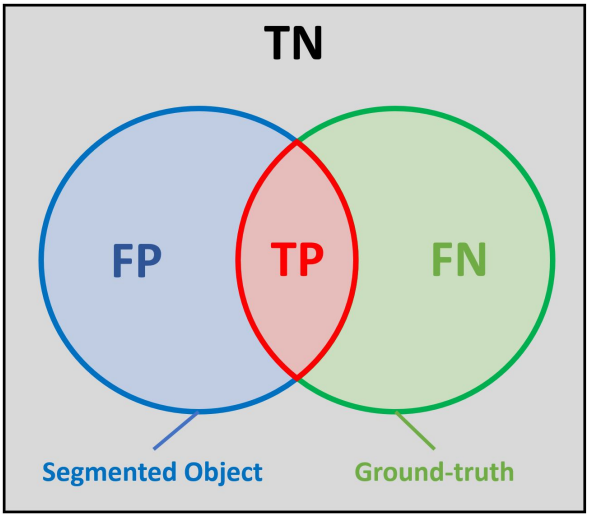

Figure 20: Prediction classes for segmentation metrics

5.3.1 Accuracy

Accuracy is the most popular metric in all kinds of deep learning tasks as it is straightforward to understand. In image segmentation, it gives out the percentage of pixels that are classified correctly and is defined as:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{5.1}$$

While the accuracy is easy to understand, it faces some problems, especially in binary segmentation tasks. Because the metric evaluates the overall percentage of correctly classified pixels, misleading results can occur when the classes are extremely imbalanced. This means that accuracy scores of above 95% can be achieved by predicting the whole image as a single class, which does not imply

good segmentation ability. Therefore, other metrics indicate the actual performance of segmentation models more robustly.

5.3.2 Precision

Precision is defined as the proportion of automatically detected pixels, that actually correspond to the ground-truth annotation. It is an attractive measurement for purity of the predicted segmentation and is defined as:

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

The precision metric is a good indicator for over-segmentation, leading to low precision scores and measuring the actual performance of the segmentation [46].

5.3.3 Recall

While precision measures the purity, the recall describes the completeness of the segmentation compared to the ground-truth [46] and is defined as:

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

Recall, also known as sensitivity, captures the under-segmentation very well, which means that a good performing segmentation model requires both recall and precision to be high. There exists a metric that combines both of them, which is known as the Dice coefficient, explained in Section 5.3.4.

5.3.4 Dice

The Dice score also known as F1-score is the harmonic mean of precision and recall defined as:

$$Dice = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{5.4}$$

Combining those measurements makes the dice score one of the most used metrics to measure the quality of segmentation models [47].

5.3.5 Intersection Over Union

The most popular metric used in semantic segmentation tasks is the Intersection over Union (IoU), also called the Jaccard index, which is defined as:

$$IoU = \frac{TP}{TP + FP + FN} \tag{5.5}$$

The IoU is a straightforward metric that measures all the pixels that overlap between the segmentation, and the ground-truth divided by the total number of pixels, which can easily be visualised, as seen in Figure 21.

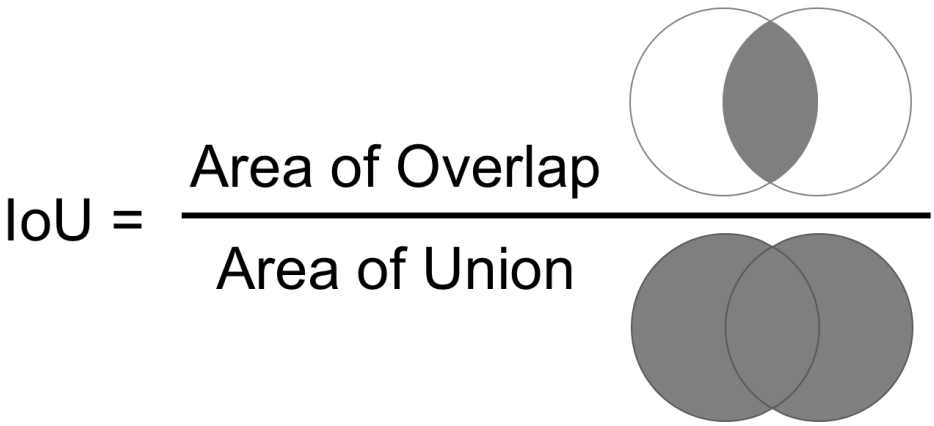

Figure 21: Visualization of the IoU score [48]

The IoU positively correlates with the Dice score, which can be seen in the similar way both metrics are calculated. This means that if model A has a better IoU score than model B, it also has a better Dice score. The difference between the metrics lies in the way of penalizing the wrong classification. The Dice score values accurate classifications more, which can be seen in the factor two in Equation (5.4). This results in the Dice score measuring the actual segmentation performance better, while the IoU measures the worst-case scenario with equally weighting every segmentation class [49]. Meaning that the Dice gives more information in quantifying the overall performance of different models but is less intuitive than the IoU, which is why both are used in the evaluation protocol in Section 5.4.

5.4 Model Evaluation

Before any data from the MSRL fetoscopic system was available, different segmentation models were tested to achieve state-of-the-art results as reported from Bano et al. [6]. Ideally, the pre-trained networks could be used directly in the MSRL system whenever it was ready, if the fetoscopic frameworks are similar. In a first step, small segmentation models based on the U-Net architecture were tested on the existing UCL dataset. Even though the achieved metrics were not yet comparable with the models proposed in [6], the predicted images already looked promising, as shown in Figure 22.

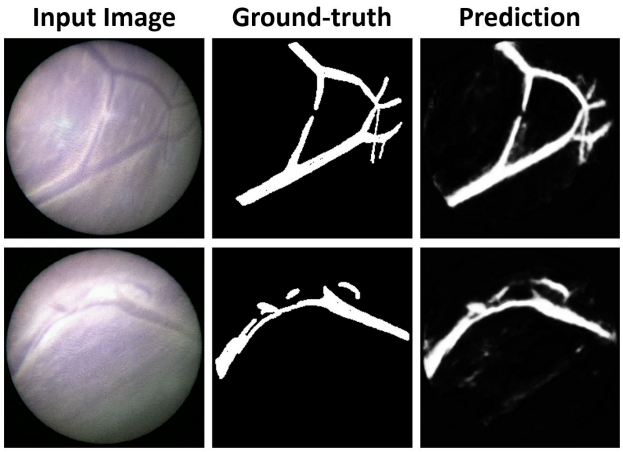

Figure 22: First segmentation results on the UCL dataset

Shortly after the first tests, images gathered with the MSRL fetoscopic system were available. The currently best performing network, a modified version of the U-Net, which achieved near state-of-the-art performance on the UCL dataset, was applied to the new data resulting in poor segmentation results illustrated in Figure 23.

Since the blood vessel detection in this thesis is built for the MSRL fetoscopic system, it needs to perform well on these obtained images and not on the UCL dataset. The poor performance of the UCL trained models then led to the creation of a new dataset presented in Section 5.1.2, which can be used to train models, specifically to the MSRL fetoscopic system.
Several differences between the fetoscopic systems exist, potentially explaining why the UCL models could not be adapted onto the MSRL data. As mentioned

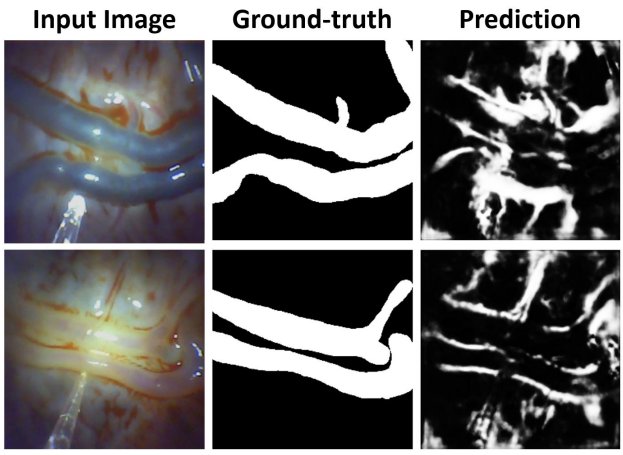

Figure 23: Segmentation of UCL model on MSRL data

in Section 5.1.2, another imaging technique is used, which leads to a different appearance of the blood vessels on the fetoscopic images. Additionally, the fetoscope lighting is brighter in the MSRL images, which leads to more reflections and thus not so easily detectable blood vessels. More difficult lighting conditions also occur due to the appearance of the ablation tool on every video frame of the MSRL data, while the UCL only uses frames with a clear view of the vessels. Another difference worth mentioning is the ex-vivo setup used to test the MSRL system, while the UCL data is recorded in vivo.

Even though the UCL models can not directly be used in the MSRL system, they could still be relevant in the future, primarily because of the in vivo data and the overall more extensive variety in placentas. Additionally, it is beneficial to have a baseline to see that the proposed models achieve state-of-the-art metrics on other datasets, which is why the segmentation models have been evaluated on both datasets.

Before the results are shown, Section 5.4.1 discusses the experimental setup used to evaluate performance of different segmentation models. In Section 5.4.2 and 5.4.3, the obtained evaluation results are presented for the UCL and MSRL datasets, while the focus lies on the MSRL dataset since it is more important for this thesis at the current time.

5.4.1 Experimental Setup

To evaluate which network architecture results in the best performance, current state-of-the-art segmentation architectures, described in Section 4.2, are compared. Six-fold cross-validation is performed using Intersection over Union (IoU) and Dice scores to measure and evaluate the performance of the segmentation models.
Besides the vanilla encoders of the U-Net [20], the LinkNet [34], the PSPNet [39] and the FPN architecture [38], different state-of-the-art classification models, that performed well in the ILSVRC [26], are tested as feature extractors, also referred to as encoder or backbone. The encoders tested are the VGG16 [29], DenseNet201 [50], EfficientNetB5 [36], InceptionResNetV2 [52], InceptionV3 [51] and ResNet101 [22] with pre-trained Imagenet weights for faster and better convergence [53].
Adam optimizer [54] and dice loss are used in all performed experiments, where patch-based training is done with a batch size of eight. For each fold, training is performed for 1000 epochs with early stopping, i.e., if the validation loss does not decrease in the latest ten epochs, the training is terminated to prevent overfitting of the model. An initial learning rate of $3e^{-4}$ is used, where an additional learning rate reduction was implemented, which automatically reduces the learning rate by a factor ten if the validation loss did not decrease in the latest four epochs. This is an effective strategy used in deep-learning tasks to get out of local minima [53]. After the training is terminated, the best performing weights are used to evaluate performance on the left out validation images of each fold. The obtained evaluation metric was once calculated with the built-in method from Keras [60] and once by a self-made function that returned the mean value and standard deviation of all the evaluated images. The self-made function was created to clearly interpret the meaning of the obtained metrics, since this is not given with the built-in method. The new method led to results, where the mean differed ($+/-10\%$) compared to the Keras [60] evaluation, mostly being smaller. The tables presented in the thesis only contain the self-calculated values with the standard deviation, while all complete tables can be found in the Appendix B in a bigger size.

5.4.2 UCL Dataset

To compare performance with the current state-of-the art models proposed by Bano et. al [6], the six folds of the UCL dataset were split the same way. Each fold belongs to a different in vivo TTTS surgery, where the intra-operative environment varies in each surgery as seen in Section 5.1.1. The images are augmented with rotation, zoom, horizontal, vertical flip, and illumination intensity changes before training. The augmentation increased the number of training images by five in each fold, which resulted in around 2000 training images per fold. After the augmentation is done, each image was resized to 256x256 pixels or 288x288 for the PSPNet, because the network requires the input image to be divisible by 48.

U-Net The cross-validation results obtained for the U-Net architecture can be seen in Table 1. It can be observed that the overall best performance is achieved by the network using the EfficientNetB5 [36] as an encoder showing an improvement of 4% Dice and 4% IoU over the Vanilla U-Net. Even though the same dataset is used as in [6], the obtained results can not directly be compared. The reason is that the experimental setup could not be precisely recreated due to lack of information available for an exact recreation. But it is safe to say that, that the obtained results achieve state-of-the-art performance on this dataset, as the metrics only differ minimally which can be seen in Table 1, where the metrics obtained in [6] are reported as a reference.

U-Net		**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Fold 6**	**Overall**
No. training images		1810	1905	2215	1970	2225	1925	
No. validation images		120	101	39	88	37	97	482
Bano et al. Vanilla [6]	Dice	0.82 (+/- 0.12)	0.73 (+/- 0.17)	0.82 (+/- 0.09)	0.70 (+/- 0.20)	0.67 (+/- 0.19)	0.72 (+/- 0.13)	0.75 (+/- 0.15)
	IoU	0.71 (+/- 0.13)	0.60 (+/- 0.18)	0.70 (+/- 0.19)	0.57 (+/- 0.22)	0.53 (+/- 0.19)	0.58 (+/- 0.15)	0.62 (+/- 0.17)
Vanilla [20]	Dice	0.80 (+/- 0.11)	0.68 (+/- 0.17)	0.78 (+/- 0.10)	0.62 (+/- 0.22)	0.68 (+/- 0.13)	0.65 (+/- 0.16)	0.70 (+/- 0.15)
	IoU	0.67 (+/- 0.13)	0.54 (+/- 0.17)	0.65 (+/- 0.13)	0.48 (+/- 0.23)	0.53 (+/- 0.14)	0.50 (+/- 0.17)	0.56 (+/- 0.17)
VGG16 [29]	Dice	0.81 (+/- 0.09)	0.69 (+/- 0.17)	0.79 (+/- 0.10)	0.57 (+/- 0.24)	0.68 (+/- 0.14)	0.53 (+/- 0.17)	0.67 (+/- 0.15)
	IoU	0.69 (+/- 0.12)	0.55 (+/- 0.18)	0.66 (+/- 0.13)	0.44 (+/- 0.24)	0.53 (+/- 0.15)	0.38 (+/- 0.16)	0.54 (+/- 0.17)
DenseNet201 [50]	Dice	0.81 (+/- 0.08)	**0.73 (+/- 0.16)**	0.80 (+/- 0.11)	0.73 (+/- 0.18)	0.73 (+/- 0.14)	0.57 (+/- 0.16)	0.72 (+/- 0.14)
	IoU	0.68 (+/- 0.10)	**0.59 (+/- 0.17)**	0.68 (+/- 0.14)	0.61 (+/- 0.19)	0.59 (+/- 0.16)	0.42 (+/- 0.15)	0.59 (+/- 0.15)
EfficientNetB5 [36]	Dice	0.79 (+/- 0.10)	0.70 (+/- 0.16)	0.79 (+/- 0.13)	**0.75 (+/- 0.15)**	**0.70 (+/- 0.16)**	**0.72 (+/- 0.13)**	**0.74 (+/- 0.13)**
	IoU	0.67 (+/- 0.11)	0.55 (+/- 0.17)	0.66 (+/- 0.15)	**0.62 (+/- 0.18)**	**0.56 (+/- 0.17)**	**0.57 (+/- 0.15)**	**0.61 (+/- 0.15)**
InceptionResNetV2 [52]	Dice	**0.82 (+/- 0.08)**	0.65 (+/- 0.20)	**0.82 (+/- 0.09)**	0.71 (+/- 0.19)	0.69 (+/- 0.16)	0.71 (+/- 0.17)	0.73 (+/- 0.15)
	IoU	**0.70 (+/- 0.10)**	0.51 (+/- 0.20)	**0.70 (+/- 0.12)**	0.58 (+/- 0.20)	0.55 (+/- 0.17)	0.57 (+/- 0.18)	0.60 (+/- 0.16)
InceptionV3 [51]	Dice	0.80 (+/- 0.11)	0.64 (+/- 0.18)	0.81 (+/- 0.10)	0.73 (+/- 0.14)	0.72 (+/- 0.14)	0.65 (+/- 0.18)	0.72 (+/- 0.15)
	IoU	0.68 (+/- 0.13)	0.49 (+/- 0.17)	0.69 (+/- 0.13)	0.60 (+/- 0.17)	0.58 (+/- 0.15)	0.51 (+/- 0.18)	0.58 (+/- 0.16)

Table 1: Cross-validation U-Net on UCL dataset

LinkNet Table 2 shows the results acquired with the LinkNet architecture, where all the tested models resulted in very similar Dice and IoU scores as for the U-Net. The best performance is again achieved with the EfficientNetB5 [36] that showed an overall robust performance on every fold. In comparison, the DenseNet201 [50] performed exceptionally well on the first few folds but was one of the worst performing in fold six, which decreased the overall performance of the network.

LinkNet		**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Fold 6**	**Overall**
No. training images		1810	1905	2215	1970	2225	1925	
No. validation images		120	101	39	88	37	97	482
Vanilla [34]	Dice	0.80 (+/- 0.08)	0.69 (+/- 0.15)	0.78 (+/- 0.10)	0.61 (+/- 0.23)	0.69 (+/- 0.13)	0.52 (+/- 0.19)	0.67 (+/- 0.15)
	IoU	0.67 (+/- 0.10)	0.54 (+/- 0.16)	0.65 (+/- 0.13)	0.48 (+/- 0.23)	0.54 (+/- 0.14)	0.37 (+/- 0.17)	0.54 (+/- 0.16)
VGG16	Dice	0.82 (+/- 0.09)	0.69 (+/- 0.13)	0.78 (+/- 0.12)	0.66 (+/- 0.21)	0.68 (+/- 0.14)	0.48 (+/- 0.18)	0.68 (+/- 0.14)
	IoU	0.71 (+/- 0.11)	0.54 (+/- 0.14)	0.65 (+/- 0.15)	0.53 (+/- 0.22)	0.53 (+/- 0.15)	0.33 (+/- 0.16)	0.55 (+/- 0.15)
DenseNet201	Dice	**0.84 (+/- 0.07)**	0.71 (+/- 0.20)	**0.81 (+/- 0.10)**	**0.74 (+/- 0.16)**	0.68 (+/- 0.14)	0.61 (+/- 0.18)	0.73 (+/- 0.14)
	IoU	**0.73 (+/- 0.10)**	0.57 (+/- 0.19)	**0.70 (+/- 0.12)**	**0.60 (+/- 0.18)**	0.53 (+/- 0.14)	0.47 (+/- 0.17)	0.6 (+/- 0.15)
EfficientNetB5	Dice	0.80 (+/- 0.08)	0.70 (+/- 0.18)	0.78 (+/- 0.15)	0.73 (+/- 0.18)	0.71 (+/- 0.15)	**0.70 (+/- 0.13)**	**0.74 (+/- 0.14)**
	IoU	0.68 (+/- 0.10)	0.56 (+/- 0.18)	0.66 (+/- 0.17)	0.60 (+/- 0.19)	0.57 (+/- 0.16)	**0.56 (+/- 0.15)**	**0.61 (+/- 0.15)**
InceptionResNetV2	Dice	0.80 (+/- 0.11)	0.67 (+/- 0.18)	0.81 (+/- 0.10)	0.73 (+/- 0.18)	0.66 (+/- 0.16)	0.67 (+/- 0.17)	0.72 (+/- 0.15)
	IoU	0.68 (+/- 0.13)	0.53 (+/- 0.18)	0.69 (+/- 0.13)	0.60 (+/- 0.19)	0.51 (+/- 0.16)	0.52 (+/- 0.17)	0.59 (+/- 0.16)
InceptionV3	Dice	0.81 (+/- 0.10)	**0.71 (+/- 0.17)**	0.80 (+/- 0.09)	0.71 (+/- 0.17)	**0.72 (+/- 0.12)**	0.63 (+/- 0.20)	0.73 (+/- 0.15)
	IoU	0.69 (+/- 0.12)	**0.57 (+/- 0.18)**	0.68 (+/- 0.12)	0.57 (+/- 0.19)	**0.58 (+/- 0.14)**	0.48 (+/- 0.20)	0.59 (+/- 0.16)

Table 2: Cross-validation LinkNet on UCL dataset

PSPNet The worst overall performance was observed with the PSPNet architecture, where the achieved metrics for each backbone are around 4% lower than the values obtained for the U-Net or LinkNet architectures, as seen in Table 3. In terms of encoding the image, it can again be observed that the EfficientNetB5 [36] leads to the best performance for the PSPNet.

PSPNet		**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Fold 6**	**Overall**
No. training images		1810	1905	2215	1970	2225	1925	
No. validation images		120	101	39	88	37	97	482
VGG16 [29]	Dice	0.79 (+/- 0.09)	0.66 (+/- 0.17)	0.73 (+/- 0.16)	0.59 (+/- 0.22)	0.67 (+/- 0.12)	0.56 (+/- 0.16)	0.67 (+/- 0.15)
	IoU	0.67 (+/- 0.11)	0.51 (+/- 0.17)	0.60 (+/- 0.18)	0.45 (+/- 0.22)	0.51 (+/- 0.13)	0.41 (+/- 0.16)	0.53 (+/- 0.16)
DenseNet201 [50]	Dice	0.80 (+/- 0.10)	0.65 (+/- 0.18)	0.76 (+/- 0.14)	0.65 (+/- 0.19)	**0.69 (+/- 0.13)**	0.55 (+/- 0.20)	0.68 (+/- 0.16)
	IoU	0.67 (+/- 0.12)	0.51 (+/- 0.18)	0.62 (+/- 0.16)	0.51 (+/- 0.20)	**0.54 (+/- 0.15)**	0.40 (+/- 0.18)	0.54 (+/- 0.16)
EfficientNetB5 [36]	Dice	0.79 (+/- 0.10)	0.65 (+/- 0.19)	0.71 (+/- 0.20)	**0.67 (+/- 0.18)**	0.66 (+/- 0.13)	**0.68 (+/- 0.15)**	**0.70 (+/- 0.15)**
	IoU	0.67 (+/- 0.11)	0.51 (+/- 0.18)	0.58 (+/- 0.21)	**0.54 (+/- 0.19)**	0.51 (+/- 0.14)	**0.54 (+/- 0.16)**	**0.57 (+/- 0.16)**
InceptionResNetV2 [52]	Dice	**0.81 (+/- 0.09)**	**0.70 (+/- 0.17)**	**0.77 (+/- 0.13)**	0.67 (+/- 0.20)	0.66 (+/- 0.14)	0.54 (+/- 0.20)	0.69 (+/- 0.16)
	IoU	**0.69 (+/- 0.11)**	**0.56 (+/- 0.17)**	**0.64 (+/- 0.16)**	0.53 (+/- 0.21)	0.51 (+/- 0.15)	0.40 (+/- 0.19)	0.56 (+/- 0.16)
InceptionV3 [51]	Dice	0.78 (+/- 0.10)	0.64 (+/- 0.20)	0.73 (+/- 0.15)	0.66 (+/- 0.19)	0.61 (+/- 0.16)	0.66 (+/- 0.16)	0.69 (+/- 0.16)
	IoU	0.65 (+/- 0.12)	0.50 (+/- 0.19)	0.59 (+/- 0.17)	0.52 (+/- 0.20)	0.46 (+/- 0.16)	0.51 (+/- 0.17)	0.55 (+/- 0.17)

Table 3: Cross-validation PSPNet on UCL dataset

FPN Whilst the PSPNet performed the worst, the best results on nearly every fold were achieved by an FPN architecture. Table 4 shows that especially the

encoders using Inception Modules perform superior on the UCL dataset with the FPN encoder. Besides the InceptionResNetV2 [52] and the InceptionV3 [51], the DenseNet201 [50] is performs well, achieving the second-highest overall performance with both the Keras [60] and the self-calculated metrics.

FPN		**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Fold 6**	**Overall**
No. training images		1810	1905	2215	1970	2225	1925	
No. validation images		120	101	39	88	37	97	482
VGG16 [29]	Dice	0.78 (+/- 0.08)	0.67 (+/- 0.18)	0.78 (+/- 0.11)	0.67 (+/- 0.21)	0.69 (+/- 0.14)	0.59 (+/- 0.17)	0.69 (+/- 0.15)
	IoU	0.65 (+/- 0.10)	0.53 (+/- 0.18)	0.66 (+/- 0.14)	0.54 (+/- 0.22)	0.55 (+/- 0.15)	0.44 (+/- 0.17)	0.56 (+/- 0.16)
DenseNet201 [50]	Dice	**0.84 (+/- 0.09)**	0.73 (+/- 0.15)	0.79 (+/- 0.13)	0.72 (+/- 0.18)	**0.73 (+/- 0.14)**	0.63 (+/- 0.20)	0.74 (+/- 0.15)
	IoU	**0.73 (+/- 0.11)**	0.59 (+/- 0.16)	0.68 (+/- 0.16)	0.59 (+/- 0.20)	**0.59 (+/- 0.16)**	0.49 (+/- 0.19)	0.61 (+/- 0.16)
EfficientNetB5 [36]	Dice	0.81 (+/- 0.07)	0.69 (+/- 0.18)	0.80 (+/- 0.13)	**0.74 (+/- 0.18)**	0.48 (+/- 0.17)	**0.70 (+/- 0.14)**	0.72 (+/- 0.14)
	IoU	0.69 (+/- 0.09)	0.55 (+/- 0.18)	0.68 (+/- 0.15)	**0.61 (+/- 0.19)**	0.33 (+/- 0.15)	**0.56 (+/- 0.16)**	0.59 (+/- 0.15)
InceptionResNetV2 [52]	Dice	0.83 (+/- 0.09)	**0.74 (+/- 0.14)**	0.80 (+/- 0.11)	0.73 (+/- 0.18)	0.72 (+/- 0.13)	0.70 (+/- 0.17)	**0.75 (+/- 0.14)**
	IoU	0.72 (+/- 0.12)	**0.60 (+/- 0.16)**	0.68 (+/- 0.14)	0.60 (+/- 0.20)	0.58 (+/- 0.15)	0.56 (+/- 0.18)	**0.63 (+/- 0.16)**
InceptionV3 [52]	Dice	0.83 (+/- 0.09)	0.73 (+/- 0.13)	**0.81 (+/- 0.10)**	0.72 (+/- 0.17)	0.70 (+/- 0.18)	0.68 (+/- 0.19)	0.74 (+/- 0.14)
	IoU	0.71 (+/- 0.11)	0.59 (+/- 0.15)	**0.69 (+/- 0.13)**	0.59 (+/- 0.18)	0.57 (+/- 0.18)	0.54 (+/- 0.19)	0.62 (+/- 0.15)

Table 4: Cross-validation FPN on UCL dataset

Summary When comparing the results obtained at each fold, it is noticeable that the segmentation results for some folds are significantly lower than others. This is mainly because some of the vessels in the validation sets are thin or have been wrongly annotated, which led to slightly scewed results in the metrics [6]. The top performances for each network architecture are reported in Table 5, where it can be observed that the overall best performance is achieved by the FPN network with the InceptionResnetV2 [52] backbone. Also noteworthy is the performance of the EfficientNetB5 [36] backbone, which is among the top performers on almost every architecture type, indicating an ability to identify placental blood vessels well. Even though the best overall performance was achieved with the FPN InceptionResnetV2 [52] network, the differences in the obtained metrics are small. This means that multiple top performers should be compared to get to the best network for this dataset for further optimisation. Since the models did not transfer well to the MSRL data, as mentioned in Section 5.4, the optimisation of these networks is not part of this thesis.

5.4.3 MSRL Dataset

The MSRL dataset described in Section 5.1.2 is split into six different folds to evaluate performance with cross-validation. Since only one placenta was available, the folds are split into different regions of the placenta that have been obtained in the videos. Fold one, two, three and four cover a region where each

Top networks		**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Fold 6**	**Overall**
No. training images		1810	1905	2215	1970	2225	1925	
No. validation images		120	101	39	88	37	97	482
U-Net EfficientNetB5	Dice	0.79 (+/- 0.10)	0.70 (+/- 0.16)	0.79 (+/- 0.13)	**0.75 (+/- 0.15)**	0.70 (+/- 0.16)	**0.72 (+/- 0.13)**	0.74 (+/- 0.13)
	IoU	0.67 (+/- 0.11)	0.55 (+/- 0.17)	0.66 (+/- 0.15)	**0.62 (+/- 0.18)**	0.56 (+/- 0.17)	**0.57 (+/- 0.15)**	0.61 (+/- 0.15)
LinkNet EfficientNetB5	Dice	0.80 (+/- 0.08)	0.70 (+/- 0.18)	0.78 (+/- 0.15)	0.73 (+/- 0.18)	0.71 (+/- 0.15)	0.70 (+/- 0.13)	0.74 (+/- 0.14)
	IoU	0.68 (+/- 0.10)	0.56 (+/- 0.18)	0.66 (+/- 0.17)	0.60 (+/- 0.19)	0.57 (+/- 0.16)	0.56 (+/- 0.15)	0.61 (+/- 0.15)
FPN InceptionResNetV2	Dice	**0.83 (+/- 0.09)**	**0.74 (+/- 0.14)**	**0.80 (+/- 0.11)**	0.73 (+/- 0.18)	**0.72 (+/- 0.13)**	0.70 (+/- 0.17)	**0.75 (+/- 0.14)**
	IoU	**0.72 (+/- 0.12)**	**0.60 (+/- 0.16)**	**0.68 (+/- 0.14)**	0.60 (+/- 0.20)	**0.58 (+/- 0.15)**	0.56 (+/- 0.18)	**0.63 (+/- 0.16)**
PSPNet EfficientNetB5	Dice	0.79 (+/- 0.10)	0.65 (+/- 0.19)	0.71 (+/- 0.20)	0.67 (+/- 0.18)	0.66 (+/- 0.13)	0.68 (+/- 0.15)	0.7 (+/- 0.15)
	IoU	0.67 (+/- 0.11)	0.51 (+/- 0.18)	0.58 (+/- 0.21)	0.54 (+/- 0.19)	0.51 (+/- 0.14)	0.54 (+/- 0.16)	0.57 (+/- 0.16)

Table 5: Top results from cross-validation on UCL dataset

fold focuses on other blood vessel structures while fold five and six cover another region, where the different placental regions can be seen in Figure 18. The folds covering different regions are used for validation, where the training images do not contain similar blood vessel structures as the validation set. The training images in each fold are first augmented using the transformations described in Section 19 and then resized to 288x288 pixels for the PSPNet and 256x256 pixels for all other networks used. As the video sequences of the different regions are not equally long, resulting in different sized validation folds since fewer video frames of those regions have been extracted.

U-Net The cross-validation results with the U-Net architecture on the MSRL dataset can be seen in Table 6, where it can be observed that a new backbone is used for evaluation, the ResNet101 [22]. This backbone architecture is used, because it has shown to extract blood vessels well in the work published by Bano et al. [6], and acts as a guideline for the performance of other networks. As shown in the Table, the ResNet101 [22] backbone was not as good on the MSRL dataset as on the UCL dataset, resulting in lower Dice and IoU scores than the Vanilla U-Net. The EfficientNetB5 [36] and the InceptionV3 [51] encoders achieved the best performance of all the different backbones tested with the U-Net. These networks resulted in the same mean Iou and Dice scores, but the EfficientNetB5 [36] backbone had a smaller standard deviation, making it the best performing backbone for the U-Net architecture.

LinkNet While for the U-Net, two networks were the best performing, there are three for the LinkNet where the results are reported in Table 7. The EfficientNetB5 [36], InceptionResNetV2 [52] and DenseNet201 [50] encoders resulted in the same Dice scores, where the LinkNet InceptionResNetV2 [52] achieved a

U-Net		**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Fold 6**	**Overall**
No. training images		1526	1232	1722	1729	1631	1575	
No. validation images		52	93	23	22	36	44	270
Vanilla [20]	Dice	0.69 (+/- 0.13)	0.68 (+/- 0.24)	0.88 (+/- 0.04)	0.91 (+/- 0.04)	0.58 (+/- 0.20)	0.67 (+/- 0.16)	0.7 (+/- 0.17)
	IoU	0.55 (+/- 0.15)	0.56 (+/- 0.25)	0.79 (+/- 0.07)	0.83 (+/- 0.06)	0.44 (+/- 0.19)	0.53 (+/- 0.18)	0.58 (+/- 0.18)
VGG16 [29]	Dice	0.70 (+/- 0.12)	0.68 (+/- 0.25)	0.88 (+/- 0.06)	0.91 (+/- 0.04)	0.57 (+/- 0.20)	0.67 (+/- 0.18)	0.7 (+/- 0.17)
	IoU	0.56 (+/- 0.14)	0.56 (+/- 0.26)	0.79 (+/- 0.09)	0.84 (+/- 0.06)	0.42 (+/- 0.19)	0.53 (+/- 0.18)	0.58 (+/- 0.18)
DenseNet201 [50]	Dice	0.58 (+/- 0.22)	**0.72 (+/- 0.22)**	0.88 (+/- 0.05)	0.91 (+/- 0.04)	0.58 (+/- 0.26)	**0.73 (+/- 0.12)**	0.71 (+/- 0.18)
	IoU	0.44 (+/- 0.21)	**0.60 (+/- 0.23)**	0.79 (+/- 0.08)	0.84 (+/- 0.07)	0.46 (+/- 0.25)	**0.59 (+/- 0.14)**	0.58 (+/- 0.19)
EfficientNetB5 [36]	Dice	0.67 (+/- 0.19)	0.72 (+/- 0.19)	**0.90 (+/- 0.05)**	**0.91 (+/- 0.03)**	**0.66 (+/- 0.16)**	0.73 (+/- 0.14)	**0.73 (+/- 0.15)**
	IoU	0.53 (+/- 0.20)	0.59 (+/- 0.21)	**0.82 (+/- 0.07)**	**0.84 (+/- 0.05)**	**0.51 (+/- 0.17)**	0.59 (+/- 0.16)	**0.61 (+/- 0.17)**
InceptionResNetV2 [52]	Dice	0.68 (+/- 0.17)	0.68 (+/- 0.24)	0.89 (+/- 0.04)	0.91 (+/- 0.03)	0.56 (+/- 0.23)	0.70 (+/- 0.13)	0.70 (+/- 0.17)
	IoU	0.53 (+/- 0.18)	0.57 (+/- 0.25)	0.80 (+/- 0.06)	0.84 (+/- 0.06)	0.42 (+/- 0.21)	0.55 (+/- 0.15)	0.58 (+/- 0.18)
InceptionV3 [51]	Dice	**0.73 (+/- 0.13)**	0.71 (+/- 0.22)	0.89 (+/- 0.05)	0.91 (+/- 0.04)	0.59 (+/- 0.22)	0.72 (+/- 0.13)	0.73 (+/- 0.16)
	IoU	**0.59 (+/- 0.16)**	0.59 (+/- 0.23)	0.81 (+/- 0.07)	0.84 (+/- 0.06)	0.45 (+/- 0.20)	0.58 (+/- 0.15)	0.61 (+/- 0.17)
ResNet101 [22]	Dice	0.58 (+/- 0.23)	0.69 (+/- 0.24)	0.88 (+/- 0.05)	0.91 (+/- 0.04)	0.55 (+/- 0.23)	0.70 (+/- 0.16)	0.69 (+/- 0.19)
	IoU	0.45 (+/- 0.23)	0.57 (+/- 0.25)	0.80 (+/- 0.08)	0.84 (+/- 0.06)	0.42 (+/- 0.20)	0.55 (+/- 0.18)	0.57 (+/- 0.2)

Table 6: Cross-validation U-Net on MSRL dataset

slightly higher IoU score making it the best performing network according to the self-calculated metrics. The metrics obtained through the Keras [60] evaluation resulted in the EfficientNetB5 [36] as the best performing encoder for the LinkNet architecture, which can be observed in Appendix B.

LinkNet		**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Fold 6**	**Overall**
No. training images		1526	1232	1722	1729	1631	1575	
No. validation images		52	93	23	22	36	44	270
Vanilla [34]	Dice	**0.73 (+/- 0.10)**	0.68 (+/- 0.24)	0.87 (+/- 0.05)	0.90 (+/- 0.04)	0.63 (+/- 0.19)	0.65 (+/- 0.17)	0.71 (+/- 0.16)
	IoU	**0.59 (+/- 0.13)**	0.56 (+/- 0.25)	0.78 (+/- 0.08)	0.83 (+/- 0.07)	0.48 (+/- 0.19)	0.50 (+/- 0.17)	0.59 (+/- 0.18)
VGG16 [29]	Dice	0.71 (+/- 0.11)	0.69 (+/- 0.23)	0.88 (+/- 0.04)	0.90 (+/- 0.05)	0.53 (+/- 0.22)	0.71 (+/- 0.12)	0.71 (+/- 0.16)
	IoU	0.57 (+/- 0.13)	0.57 (+/- 0.24)	0.79 (+/- 0.06)	0.83 (+/- 0.08)	0.39 (+/- 0.20)	0.56 (+/- 0.14)	0.58 (+/- 0.17)
DenseNet201 [50]	Dice	0.69 (+/- 0.15)	0.71 (+/- 0.23)	**0.89 (+/- 0.03)**	**0.91 (+/- 0.04)**	0.65 (+/- 0.18)	0.70 (+/- 0.16)	0.73 (+/- 0.16)
	IoU	0.55 (+/- 0.17)	0.59 (+/- 0.24)	**0.81 (+/- 0.05)**	**0.84 (+/- 0.06)**	0.50 (+/- 0.19)	0.56 (+/- 0.18)	0.6 (+/- 0.18)
EfficientNetB5 [36]	Dice	0.71 (+/- 0.15)	**0.71 (+/- 0.21)**	0.89 (+/- 0.04)	0.91 (+/- 0.04)	0.58 (+/- 0.22)	**0.74 (+/- 0.13)**	0.73 (+/- 0.16)
	IoU	0.56 (+/- 0.17)	**0.59 (+/- 0.22)**	0.81 (+/- 0.07)	0.83 (+/- 0.06)	0.44 (+/- 0.21)	**0.60 (+/- 0.15)**	0.6 (+/- 0.17)
InceptionResNetV2 [52]	Dice	0.72 (+/- 0.13)	0.70 (+/- 0.22)	0.88 (+/- 0.05)	**0.91 (+/- 0.04)**	**0.67 (+/- 0.18)**	0.71 (+/- 0.15)	**0.73 (+/- 0.16)**
	IoU	0.58 (+/- 0.14)	0.58 (+/- 0.23)	0.78 (+/- 0.08)	**0.84 (+/- 0.06)**	**0.53 (+/- 0.19)**	0.57 (+/- 0.17)	**0.61 (+/- 0.17)**
InceptionV3 [51]	Dice	0.68 (+/- 0.17)	0.70 (+/- 0.23)	0.89 (+/- 0.04)	0.91 (+/- 0.04)	0.64 (+/- 0.18)	0.70 (+/- 0.14)	0.72 (+/- 0.17)
	IoU	0.54 (+/- 0.19)	0.57 (+/- 0.24)	0.80 (+/- 0.07)	0.84 (+/- 0.06)	0.49 (+/- 0.18)	0.56 (+/- 0.15)	0.59 (+/- 0.18)
ResNet101 [22]	Dice	0.62 (+/- 0.21)	0.68 (+/- 0.25)	0.88 (+/- 0.06)	0.91 (+/- 0.04)	0.50 (+/- 0.27)	0.71 (+/- 0.17)	0.69 (+/- 0.2)
	IoU	0.48 (+/- 0.21)	0.56 (+/- 0.25)	0.79 (+/- 0.09)	0.84 (+/- 0.07)	0.37 (+/- 0.24)	0.57 (+/- 0.18)	0.56 (+/- 0.2)

Table 7: Cross-validation LinkNet on MSRL dataset

PSPNet While the U-Net and LinkNet models performed very similarly, the models obtained from the PSPNet performed around 3% worse on each tested network, as reported in Table 8. The best metrics were achieved with the VGG16 [29] encoder, which showed a consistent performance on all folds. The PSPNet with DenseNet201 [50] backbone performed better on most folds but showed a lack of robustness in fold five, which led to a lower overall IoU score.

FPN It is noticeable in Table 9 that the overall performance of the FPN architectures is mostly higher than other tested networks with an increase of Dice and

PSPNet		Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Overall
No. training images		1526	1232	1722	1729	1631	1575	
No. validation images		52	93	23	22	36	44	270
VGG16 [29]	Dice	**0.71 (+/- 0.12)**	0.69 (+/- 0.23)	0.87 (+/- 0.06)	0.91 (+/- 0.04)	0.61 (+/- 0.19)	0.67 (+/- 0.14)	**0.71 (+/- 0.16)**
	IoU	**0.56 (+/- 0.14)**	0.57 (+/- 0.23)	0.78 (+/- 0.09)	0.83 (+/- 0.06)	0.47 (+/- 0.19)	0.52 (+/- 0.16)	**0.59 (+/- 0.17)**
DenseNet201 [50]	Dice	0.70 (+/- 0.13)	0.70 (+/- 0.21)	0.88 (+/- 0.05)	**0.91 (+/- 0.03)**	0.57 (+/- 0.19)	**0.68 (+/- 0.12)**	0.71 (+/- 0.15)
	IoU	0.55 (+/- 0.15)	0.58 (+/- 0.22)	0.78 (+/- 0.07)	**0.83 (+/- 0.05)**	0.42 (+/- 0.17)	**0.53 (+/- 0.14)**	0.58 (+/- 0.16)
EfficientNetB5 [36]	Dice	0.67 (+/- 0.15)	0.69 (+/- 0.22)	0.88 (+/- 0.06)	0.90 (+/- 0.05)	0.53 (+/- 0.21)	0.65 (+/- 0.16)	0.69 (+/- 0.17)
	IoU	0.52 (+/- 0.17)	0.56 (+/- 0.22)	0.79 (+/- 0.08)	0.82 (+/- 0.08)	0.39 (+/- 0.19)	0.49 (+/- 0.16)	0.56 (+/- 0.17)
InceptionResNetV2 [52]	Dice	0.62 (+/- 0.21)	0.69 (+/- 0.22)	0.88 (+/- 0.04)	0.91 (+/- 0.05)	**0.64 (+/- 0.19)**	0.67 (+/- 0.14)	0.7 (+/- 0.17)
	IoU	0.49 (+/- 0.22)	0.56 (+/- 0.22)	0.79 (+/- 0.06)	0.83 (+/- 0.08)	**0.49 (+/- 0.19)**	0.52 (+/- 0.15)	0.57 (+/- 0.18)
InceptionV3 [51]	Dice	0.60 (+/- 0.22)	**0.70 (+/- 0.20)**	0.87 (+/- 0.06)	0.91 (+/- 0.04)	0.62 (+/- 0.18)	0.68 (+/- 0.13)	0.7 (+/- 0.16)
	IoU	0.47 (+/- 0.22)	**0.58 (+/- 0.22)**	0.78 (+/- 0.08)	0.83 (+/- 0.06)	0.47 (+/- 0.18)	0.53 (+/- 0.15)	0.57 (+/- 0.18)
ResNet101 [22]	Dice	0.56 (+/- 0.23)	0.66 (+/- 0.25)	**0.89 (+/- 0.04)**	0.89 (+/- 0.06)	0.44 (+/- 0.25)	0.63 (+/- 0.17)	0.64 (+/- 0.2)
	IoU	0.43 (+/- 0.22)	0.54 (+/- 0.25)	**0.80 (+/- 0.06)**	0.80 (+/- 0.09)	0.32 (+/- 0.21)	0.48 (+/- 0.17)	0.52 (+/- 0.2)

Table 8: Cross-validation PSPNet on MSRL dataset

IoU score in nearly every model. The overall best performance was again achieved by an FPN architecture, as with the UCL dataset. While in the cross-validation for the UCL dataset the InceptionResNetV2 [52] resulted in the best IoU and Dice scores, the EfficientNetB5 [36] backbone achieves the highest metrics on the cross-validation of the MSRL dataset.

FPN		Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Overall
No. training images		1526	1232	1722	1729	1631	1575	
No. validation images		52	93	23	22	36	44	270
VGG16 [29]	Dice	**0.72 (+/- 0.12)**	0.67 (+/- 0.25)	0.89 (+/- 0.04)	0.91 (+/- 0.04)	0.62 (+/- 0.20)	0.71 (+/- 0.13)	0.72 (+/- 0.16)
	IoU	**0.58 (+/- 0.14)**	0.55 (+/- 0.26)	0.80 (+/- 0.06)	0.83 (+/- 0.07)	0.48 (+/- 0.20)	0.57 (+/- 0.15)	0.59 (+/- 0.18)
DenseNet201 [50]	Dice	0.72 (+/- 0.13)	0.71 (+/- 0.20)	0.89 (+/- 0.04)	0.91 (+/- 0.04)	0.62 (+/- 0.19)	**0.72 (+/- 0.14)**	0.73 (+/- 0.15)
	IoU	0.58 (+/- 0.15)	0.59 (+/- 0.21)	0.80 (+/- 0.06)	0.84 (+/- 0.06)	0.47 (+/- 0.20)	**0.57 (+/- 0.16)**	0.61 (+/- 0.16)
EfficientNetB5 [36]	Dice	0.72 (+/- 0.13)	**0.73 (+/- 0.21)**	**0.91 (+/- 0.04)**	0.91 (+/- 0.03)	**0.69 (+/- 0.15)**	0.65 (+/- 0.17)	**0.74 (+/- 0.15)**
	IoU	0.58 (+/- 0.16)	**0.61 (+/- 0.22)**	**0.83 (+/- 0.06)**	0.84 (+/- 0.06)	**0.54 (+/- 0.16)**	0.51 (+/- 0.18)	**0.62 (+/- 0.17)**
InceptionResNetV2 [52]	Dice	0.67 (+/- 0.18)	0.70 (+/- 0.23)	0.88 (+/- 0.06)	0.91 (+/- 0.03)	0.59 (+/- 0.23)	0.70 (+/- 0.16)	0.71 (+/- 0.18)
	IoU	0.53 (+/- 0.19)	0.58 (+/- 0.24)	0.79 (+/- 0.08)	0.84 (+/- 0.06)	0.45 (+/- 0.21)	0.56 (+/- 0.17)	0.59 (+/- 0.19)
InceptionV3 [51]	Dice	0.65 (+/- 0.19)	0.72 (+/- 0.20)	0.90 (+/- 0.05)	**0.91 (+/- 0.03)**	0.58 (+/- 0.23)	0.71 (+/- 0.14)	0.72 (+/- 0.17)
	IoU	0.51 (+/- 0.20)	0.59 (+/- 0.22)	0.81 (+/- 0.08)	**0.84 (+/- 0.05)**	0.45 (+/- 0.22)	0.57 (+/- 0.16)	0.59 (+/- 0.18)
ResNet101 [22]	Dice	0.60 (+/- 0.21)	0.65 (+/- 0.28)	0.89 (+/- 0.04)	0.91 (+/- 0.04)	0.50 (+/- 0.23)	0.66 (+/- 0.18)	0.66 (+/- 0.2)
	IoU	0.46 (+/- 0.21)	0.54 (+/- 0.27)	0.81 (+/- 0.07)	0.84 (+/- 0.06)	0.37 (+/- 0.21)	0.52 (+/- 0.19)	0.55 (+/- 0.2)

Table 9: Cross-validation FPN on MSRL dataset

Summary Table 10 summarizes the overall best performing models for each architecture type resulting from the cross-validation on the MSRL dataset. When comparing the achieved Dice and IoU score on each fold, it can be noticed that the scores on fold three and four are much higher than on the residual folds. This behaviour can be explained when comparing the different folds. The main reason for the better performance lies in the minimal diversity in the validation images, where only bigger and easy to identify blood vessels are present. Additionally, the validation sets in fold three, and four are small, with around 20 validation images, which results in larger training set. When comparing the validation set with the training images, one can observe that some structures are present in both

sets, which means that these higher scores could show signs of overfitting on those particular folds. Because of those observations, the folds are only minimally taken into account when evaluating the performance of the models.

Top networks		**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Fold 6**	**Overall**
No. training images		1526	1232	1722	1729	1631	1575	
No. validation images		52	93	23	22	36	44	270
U-Net EfficientNetB5	Dice	0.67 (+/- 0.19)	0.72 (+/- 0.19)	0.90 (+/- 0.05)	0.91 (+/- 0.03)	0.66 (+/- 0.16)	0.73 (+/- 0.14)	0.73 (+/- 0.15)
	IoU	0.53 (+/- 0.20)	0.59 (+/- 0.21)	0.82 (+/- 0.07)	0.84 (+/- 0.05)	0.51 (+/- 0.17)	0.59 (+/- 0.16)	0.61 (+/- 0.17)
U-Net InceptionV3	Dice	**0.73 (+/- 0.13)**	0.71 (+/- 0.22)	0.89 (+/- 0.05)	0.91 (+/- 0.04)	0.59 (+/- 0.22)	0.72 (+/- 0.13)	0.73 (+/- 0.16)
	IoU	**0.59 (+/- 0.16)**	0.59 (+/- 0.23)	0.81 (+/- 0.07)	0.84 (+/- 0.06)	0.45 (+/- 0.20)	0.58 (+/- 0.15)	0.61 (+/- 0.17)
LinkNet EfficientNetB5	Dice	0.71 (+/- 0.15)	0.71 (+/- 0.21)	0.89 (+/- 0.04)	0.91 (+/- 0.04)	0.58 (+/- 0.22)	**0.74 (+/- 0.13)**	0.73 (+/- 0.16)
	IoU	0.56 (+/- 0.17)	0.59 (+/- 0.22)	0.81 (+/- 0.07)	0.83 (+/- 0.06)	0.44 (+/- 0.21)	**0.60 (+/- 0.15)**	0.6 (+/- 0.17)
LinkNet InceptionResNetV2	Dice	0.72 (+/- 0.13)	0.70 (+/- 0.22)	0.88 (+/- 0.05)	0.91 (+/- 0.04)	0.67 (+/- 0.18)	0.71 (+/- 0.15)	0.73 (+/- 0.16)
	IoU	0.58 (+/- 0.14)	0.58 (+/- 0.23)	0.78 (+/- 0.08)	0.84 (+/- 0.06)	0.53 (+/- 0.19)	0.57 (+/- 0.17)	0.61 (+/- 0.17)
FPN DenseNet201	Dice	0.72 (+/- 0.13)	0.71 (+/- 0.20)	0.89 (+/- 0.04)	0.91 (+/- 0.04)	0.62 (+/- 0.19)	0.72 (+/- 0.14)	0.73 (+/- 0.15)
	IoU	0.58 (+/- 0.15)	0.59 (+/- 0.21)	0.80 (+/- 0.06)	0.84 (+/- 0.06)	0.47 (+/- 0.20)	0.57 (+/- 0.16)	0.61 (+/- 0.16)
FPN EfficientNetB5	Dice	0.72 (+/- 0.13)	**0.73 (+/- 0.21)**	**0.91 (+/- 0.04)**	**0.91 (+/- 0.03)**	**0.69 (+/- 0.15)**	0.65 (+/- 0.17)	**0.74 (+/- 0.15)**
	IoU	0.58 (+/- 0.16)	**0.61 (+/- 0.22)**	**0.83 (+/- 0.06)**	**0.84 (+/- 0.06)**	**0.54 (+/- 0.16)**	0.51 (+/- 0.18)	**0.62 (+/- 0.17)**
PSPNet VGG16	Dice	0.71 (+/- 0.12)	0.69 (+/- 0.23)	0.87 (+/- 0.06)	0.91 (+/- 0.04)	0.61 (+/- 0.19)	0.67 (+/- 0.14)	0.71 (+/- 0.16)
	IoU	0.56 (+/- 0.14)	0.57 (+/- 0.23)	0.78 (+/- 0.09)	0.83 (+/- 0.06)	0.47 (+/- 0.19)	0.52 (+/- 0.16)	0.59 (+/- 0.17)
PSPNet DenseNet201	Dice	0.70 (+/- 0.13)	0.70 (+/- 0.21)	0.88 (+/- 0.05)	0.91 (+/- 0.03)	0.57 (+/- 0.19)	0.68 (+/- 0.12)	0.71 (+/- 0.15)
	IoU	0.55 (+/- 0.15)	0.58 (+/- 0.22)	0.78 (+/- 0.07)	0.83 (+/- 0.05)	0.42 (+/- 0.17)	0.53 (+/- 0.14)	0.58 (+/- 0.16)

Table 10: Top results from cross-validation on MSRL dataset

When comparing the backbones among all top networks, it becomes clear that the EfficientNetB5 [36] is again performing best in numerous cases. The same observations were also made in the top networks on the UCL dataset in Table 5, emphasising the ability to reliably and robustly identify placental blood vessels. The overall best performance on both evaluation methods was also achieved with the EfficientNetB5 [36] backbone, used with the FPN architecture. Even though the FPN-EfficientNetB5 model performed best on the cross-validation, very similar metrics were achieved with the other models using U-Net, LinkNet or FPN architectures. Only the PSPNet performed worse on all validated folds; therefore, it is not the optimal model for the MSRL fetoscopic system.
The best performing models of the U-Net, LinkNet and FPN are further optimised and evaluated in Section 5.5 to get to the network that best identifies the placental blood vessels with the MSRL fetoscopic images.

5.5 Optimisation

When optimising a deep-learning model, many seemingly arbitrary decisions have to be made since no rules exist which guarantee the best performance of a network. How many layers should be stacked? Which loss function should be used? How much dropout should be applied? And so on. Thousands of ways a deep-learning models can be tuned and optimised exist, which can be an endless procedure.

This thesis focuses on optimising the best performing networks from Section 5.4.3 by tuning three of the most influential parameters, the loss function, the batch size and the initial learning rate. The evaluation through cross-validation used a fixed set of those parameters with the dice loss, batch size of eight and an initial learning rate of $3e^{-4}$, to obtain the most robust networks out of many. But arbitrary choices rarely lead to the limit of what can be achieved on a given task, even with a good intuition and lots of experience [53].

5.5.1 Experimental Setup

The experimental setup for fine-tuning the top-performing models obtained through cross-validation is very similar to the setup described in Section 5.4.1. Training is performed over 1000 epochs using Adam optimiser with early stopping and learning rate reduction implemented, where the best-performing weights are saved and used for evaluation. Unlike Section 5.4, no cross-validation is performed to optimise the networks. The MSRL dataset has been randomly split into a training, validation and test set to evaluate the performance of the network. A split of 60-20-20 is used, which means that the dataset is split into 60% training and 20% validation and test set each. This resulted in validation and test set with 53 images each and a training set of 1141 images after augmenting the images. Since the dataset is relatively small, the data is split into a larger validation and test set, reducing overfitting. If more data is available, the usual split is 80-10-10. In the following sections, the best networks from Section 5.4.3 are optimised on the validation set using different parameters. In the end the optimised models are evaluated on the test set, which then resulted in the final segmentation models presented in Section 5.6. All tables with the metrics from both evaluation methods can be found in Appendix C, while the following sections only show metrics of the self-calculated evaluation method.

5.5.2 Loss Function

The loss function represents the quantity that is minimised during the training of deep-learning models and acts as another measure of success in segmentation tasks. In the performed tests, different state-of-the-art loss functions for segmentation tasks are used to evaluate which loss function results in the best IoU and

Dice scores on the tested networks. The tested loss functions are the Dice loss, the Jaccard or IoU loss, the binary cross-entropy (BCE) and combinations of these losses. The mathematical definition of the losses are shown in Equations (5.6) - (5.8). While the Dice loss and Jaccard loss are straight forward to understand given the definition of their metric in Section 5.3, the binary cross-entropy is defined differently using the ground-truth (gt) and the predictions (pr) and can be used directly within the Keras API [60].

$$L_{Dice} = 1 - Dice = 1 - \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = 1 - \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{5.6}$$

$$L_{Jacc} = 1 - IoU = 1 - \frac{TP}{TP + FP + FN} \tag{5.7}$$

$$L_{BCE} = -[gt \cdot log(pr) + (1 - gt) \cdot log(1 - pr)] \tag{5.8}$$

Since the optimal parameter for the batch size and the learning rate have not been evaluated, the initial values used for the cross-validation are used. The results obtained for the different loss functions using a batch size of eight and an initial learning rate of $3e^{-4}$ are reported in Table 11. The obtained results show that the Jaccard loss leads to the best results for the FPN networks and the LinkNet with InceptionResNetV2 [52] backbone while the remaining networks perform best with a combined loss function of the Jaccard loss and the binary cross-entropy. The best performances were again achieved by the networks using EfficientNetB5 [36] as backbone, underlining the results obtained through the cross-validation in Section 5.4.3, with the FPN architecture achieving the best overall results. The best performing loss functions are now fixed for each model and used in the experiments in Section 5.5.3 and 5.5.4.

5.5.3 Batch Size

Performing training with deep learning models is mostly done in batches, which increases convergence and performance. The batch size is defined as the number of images used for a single training iteration of a deep learning model. Different batch sizes influence the performance and convergence speed of deep learning

Loss functions		**Dice**	**Jaccard**	**BCE**	**Jaccard + BCE**	**Dice + BCE**
No. training images		1141	1141	1141	1141	1141
No. validation images		53	53	53	53	53
FPN EfficientNetB5	Dice	0.7825 (+/- 0.1627)	**0.7922 (+/- 0.1590)**	0.7589 (+/- 0.1560)	0.7736 (+/- 0.1622)	0.7790 (+/- 0.1563)
	IoU	0.6662 (+/- 0.1800)	**0.6784 (+/- 0.1740)**	0.6327 (+/- 0.1720)	0.6542 (+/- 0.1807)	0.6595 (+/- 0.1722)
U-Net EfficientNetB5	Dice	0.7625 (+/- 0.1794)	0.7724 (+/- 0.1717)	0.7436 (+/- 0.1579)	**0.7855 (+/- 0.1584)**	0.7642 (+/- 0.1701)
	IoU	0.6438 (+/- 0.1944)	0.6552 (+/- 0.1901)	0.6129 (+/- 0.1708)	**0.6690 (+/- 0.1742)**	0.6434 (+/- 0.1855)
LinkNet EfficientNetB5	Dice	0.7624 (+/- 0.1669)	0.7840 (+/- 0.1640)	0.7684 (+/- 0.1551)	**0.7877 (+/- 0.1545)**	0.7764 (+/- 0.1606)
	IoU	0.6406 (+/- 0.1860)	0.6686 (+/- 0.1805)	0.6447 (+/- 0.1682)	**0.6709 (+/- 0.1701)**	0.6572 (+/- 0.1765)
U-Net InceptionV3	Dice	0.7688 (+/- 0.1719)	0.7569 (+/- 0.1954)	0.7508 (+/- 0.1879)	**0.7722 (+/- 0.1761)**	0.7563 (+/- 0.1794)
	IoU	0.6506 (+/- 0.1904)	0.6413 (+/- 0.2094)	0.6310 (+/- 0.2028)	**0.6562 (+/- 0.1933)**	0.6354 (+/- 0.1926)
LinkNet InceptionResNetV2	Dice	0.7597 (+/- 0.1948)	**0.7783 (+/- 0.1814)**	0.7688 (+/- 0.1756)	0.7717 (+/- 0.1707)	0.7669 (+/- 0.1831)
	IoU	0.6455 (+/- 0.2130)	**0.6662 (+/- 0.1997)**	0.6516 (+/- 0.1942)	0.6543 (+/- 0.1903)	0.6511 (+/- 0.1995)
FPN DenseNet201	Dice	0.7669 (+/- 0.1854)	**0.7768 (+/- 0.1623)**	0.7732 (+/- 0.1714)	0.7256 (+/- 0.2138)	0.7379 (+/- 0.1869)
	IoU	0.6516 (+/- 0.2003)	**0.6589 (+/- 0.1834)**	0.6563 (+/- 0.1901)	0.6059 (+/- 0.2211)	0.6145 (+/- 0.2051)

Table 11: Evaluation of loss functions

models [55], where a larger batch size increases convergence speed. The increase in batch size also demands more processing power which is again limited by the available GPU memory. While trying different batch sizes, this exact limitation could be observed. The limited availability of GPU memory resulted in error messages because the processor could not do all of the required computations. Table 12 shows the models affected by those limitations, which are marked with *overflow*.

Batch sizes		**8**	**16**	**32**	**64**
No. training images		1141	1141	1141	1141
No. validation images		53	53	53	53
FPN EfficientNetB5	Dice	**0.7922 (+/- 0.1590)**	overflow	overflow	overflow
	IoU	**0.6784 (+/- 0.1740)**	overflow	overflow	overflow
U-Net EfficientNetB5	Dice	**0.7855 (+/- 0.1584)**	overflow	overflow	overflow
	IoU	**0.6690 (+/- 0.1742)**	overflow	overflow	overflow
LinkNet EfficientNetB5	Dice	**0.7877 (+/- 0.1545)**	overflow	overflow	overflow
	IoU	**0.6709 (+/- 0.1701)**	overflow	overflow	overflow
U-Net InceptionV3	Dice	**0.7722 (+/- 0.1761)**	0.7655 (+/- 0.1779)	0.7525 (+/- 0.1762)	overflow
	IoU	**0.6562 (+/- 0.1933)**	0.6471 (+/- 0.1907	0.6303 (+/- 0.1958)	overflow
LinkNet InceptionResNetV2	Dice	**0.7783 (+/- 0.1814)**	0.7377 (+/- 0.2023)	overflow	overflow
	IoU	**0.6662 (+/- 0.1997)**	0.6180 (+/- 0.2130)	overflow	overflow
FPN DenseNet201	Dice	**0.7768 (+/- 0.1623)**	0.7384 (+/- 0.1947)	overflow	overflow
	IoU	**0.6589 (+/- 0.1834)**	0.6167 (+/- 0.2064)	overflow	overflow

Table 12: Evaluation of batch sizes

Those models to overflow are the larger models, which already take up more processing power, rendered increasing the batch size impossible. The models, where a larger batch size could be tested, did not show an increase in Dice or IoU scores, resulting in a final batch size of eight for all networks, which was further used in Section 5.5.4.

5.5.4 Learning Rate

The learning rate controls how much the model weights are updated in each training iteration using information of the loss function. Choosing the right learning rate is challenging and crucial for a good and efficient performance of any deep learning model as a value too large can cause the system to diverge and a value too small can lead to slow learning [56]. This means that the goal is to choose the learning rate as high as possible and as small as necessary, leading to an optimal minimisation of the loss function. Figure 24 illustrates the influence of the learning rate in minimising the loss function.

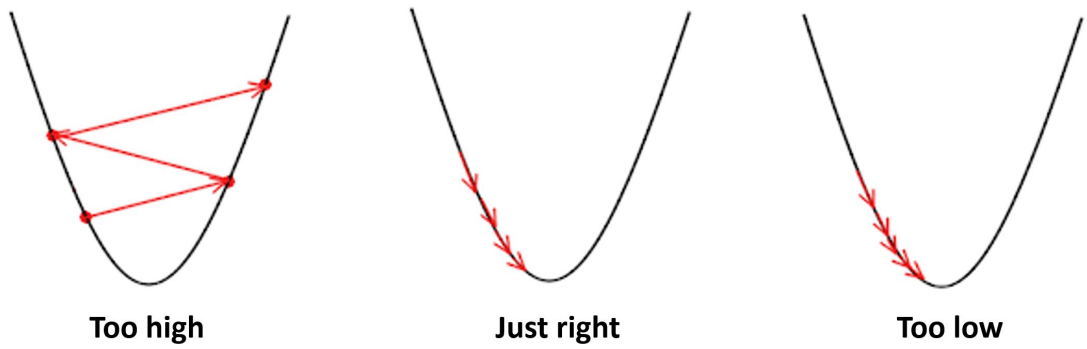

Figure 24: Effect of different learning rates [57]

Since early stopping is implemented, the training is terminated when the progress is too slow, which leads to non-optimal results. Additionally, the learning rate reduction implemented can prevent the model from diverging because the learning rate is reduced if no improvement is made. However, sometimes even this preventative method can not stop the model from diverging, as can be observed in Table 13, where the initial learning rate of $3e^{-1}$ was too high and led to bad results. The opposite effect of a learning rate that is too slow can also be observed in Table 13, where the training terminated early due to minimal progress for an initial rate of $3e^{-5}$.

The best performance out of all the tested initial learning rates was achieved with a value of $3e^{-4}$ on every tested model. The evaluation of the different learning rates completes the optimisation process for the networks used, which can now be evaluated on the never seen test set.

Initial learning rates		**3e-1**	**3e-2**	**3e-3**	**3e-4**	**3e-5**
No. training images		1141	1141	1141	1141	1141
No. validation images		53	53	53	53	53
FPN EfficientNetB5	Dice	0.2861 (+/- 0.1523)	0.7272 (+/- 0.2044)	0.7864 (+/- 0.1619)	**0.7922 (+/- 0.1590)**	0.6943 (+/- 0.2126)
	IoU	0.1766 (+/- 0.1095)	0.6045 (+/- 0.2106)	0.6715 (+/- 0.1794)	**0.6784 (+/- 0.1740)**	0.5671 (+/- 0.2201)
U-Net EfficientNetB5	Dice	0.1670 (+/- 0.0598)	0.7188 (+/- 0.1906)	0.7779 (+/- 0.1634)	**0.7855 (+/- 0.1584)**	0.6731 (+/- 0.1765)
	IoU	0.0923 (+/- 0.0356)	0.5901 (+/- 0.1988)	0.6595 (+/- 0.1759)	**0.6690 (+/- 0.1742)**	0.5316 (+/- 0.1847)
LinkNet EfficientNetB5	Dice	0.1672 (+/- 0.0599)	0.7271 (+/- 0.1903)	0.7564 (+/- 0.1605)	**0.7877 (+/- 0.1545)**	0.6524 (+/- 0.1893)
	IoU	0.0924 (+/- 0.0357)	0.6002 (+/- 0.1975)	0.6307 (+/- 0.1773)	**0.6709 (+/- 0.1701)**	0.5108 (+/- 0.1916)
U-Net InceptionV3	Dice	0.1671 (+/- 0.0599)	0.6835 (+/- 0.1997)	0.7226 (+/- 0.1872)	**0.7722 (+/- 0.1761)**	0.6485 (+/- 0.1997)
	IoU	0.0923 (+/- 0.0356)	0.5501 (+/- 0.2066)	0.5942 (+/- 0.1981)	**0.6562 (+/- 0.1933)**	0.5094 (+/- 0.2024)
LinkNet InceptionResNetV2	Dice	0.2861 (+/- 0.1523)	0.6437 (+/- 0.2526)	0.7485 (+/- 0.1729)	**0.7783 (+/- 0.1814)**	0.5978 (+/- 0.2252)
	IoU	0.1766 (+/- 0.1095)	0.5205 (+/- 0.2504)	0.6236 (+/- 0.1889)	**0.6662 (+/- 0.1997)**	0.4604 (+/- 0.2150)
FPN DenseNet201	Dice	0.2861 (+/- 0.1523)	0.2861 (+/- 0.1523)	0.6712 (+/- 0.2396)	**0.7768 (+/- 0.1623)**	0.6968 (+/- 0.2139)
	IoU	0.1766 (+/- 0.1095)	0.1766 (+/- 0.1095)	0.5479 (+/- 0.2405)	**0.6589 (+/- 0.1834)**	0.5702 (+/- 0.2200)

Table 13: Evaluation of initial learning rates

5.6 Results

The final models, after optimising the parameters of each network are summarised in Table 14. The table shows the best performance on the validation set, where the weights are saved and further used to evaluate performance on a test set with never before seen test images. It can be observed that the models perform similarly to the validation set and mostly even better on both calculated evaluation methods, where the Dice and IoU scores of the Keras [60] evaluation are around 5% higher than the self-calculated mean values. The performance of the optimised networks is compared to the U-Net with ResNet101 [22] backbone that performed best on the placental dataset in Bano et al. [6] and acts as a baseline architecture. The overall best performance on the validation images was achieved by the FPN-EfficientNetB5, which showed an overall improvement of 5% Dice and 6% IoU over the U-Net-ResNet101. This network also performed best on the test images where an improvement of 3% Dice and 4% IoU can be observed.

While Table 14 compares the different models through the obtained metrics, it is also of high interest on how the actual segmentation results look like on the placental images. Figure 25 shows the performance of all optimised models on an arbitrary input image of the test set, with the ground-truth segmentation in the bottom left corner.

While the metrics of each model are different, the actual segmentation results look very similar. The segmentations in Figure 25 show that the bigger blood vessels are reliably identified with all the optimised models; the differences appear when comparing smaller details like the thin blood vessel present at the top.

Final networks		**Loss function**	**Batch size**	**Learning rate**	**Validation set**		**Test set**	
No. training images					1141		1141	
No. validation images					53		53	
FPN EfficientNetB5	Dice	Jaccard	8	3.00E-04	**0.835**	**0.7922 (+/- 0.1590)**	**0.8628**	**0.7954 (+/- 0.1423)**
	IoU				**0.7179**	**0.6784 (+/- 0.1740)**	**0.7592**	**0.6801 (+/- 0.1698)**
U-Net EfficientNetB5	Dice	Jaccard + BCE	8	3.00E-04	0.8289	0.7855 (+/- 0.1584)	0.8399	0.7762 (+/- 0.1685)
	IoU				0.7086	0.6690 (+/- 0.1742)	0.7249	0.6591 (+/- 0.1839)
LinkNet EfficientNetB5	Dice	Jaccard + BCE	8	3.00E-04	0.8244	0.7877 (+/- 0.1545)	0.8435	0.7736 (+/- 0.1709)
	IoU				0.7022	0.6709 (+/- 0.1701)	0.7299	0.6559 (+/- 0.1836)
U-Net InceptionV3	Dice	Jaccard + BCE	8	3.00E-04	0.8279	0.7722 (+/- 0.1761)	0.8568	0.7843 (+/- 0.1498)
	IoU				0.7083	0.6562 (+/- 0.1933)	0.7501	0.6665 (+/- 0.1754)
LinkNet InceptionResNetV2	Dice	Jaccard	8	3.00E-04	0.8346	0.7783 (+/- 0.1814)	0.8492	0.7834 (+/- 0.1467)
	IoU				0.7176	0.6662 (+/- 0.1997)	0.7387	0.665 (+/- 0.176)
FPN DenseNet201	Dice	Jaccard	8	3.00E-04	0.8186	0.7768 (+/- 0.1623)	0.8584	0.7889 (+/- 0.1494)
	IoU				0.6937	0.6589 (+/- 0.1834)	0.7527	0.6729 (+/- 0.1768)
U-Net ResNet101	Dice	Jaccard + BCE	8	3.00E-04	0.8019	0.74 (+/- 0.1965)	0.8429	0.7646 (+/- 0.1581)
	IoU				0.6717	0.6196 (+/- 0.2104)	0.7295	0.6424 (+/- 0.1857)

Table 14: Summary of the optimised models

Identifying thinner blood vessels is essential since the eye can hardly see them, making even a small improvement in Dice and IoU scores significant. Another difference compared to the ground-truth is that the predicted segmentations have much smoother edges on the blood vessels, which corresponds more to the truth in nature.

To visualise characteristic segmentation performance of the optimised network, more results of the FPN-EfficientNetB5 are shown in Figure 26 and compared to the ground-truth and the performance of the U-Net-ResNet101, which acts as a baseline to show improvements made. Figure 26 shows that the FPN-EfficientNetB5 managed to detect blood vessels, depicted with a blue circle, which the human annotator missed or not labeled correctly. Even though the model outperformed the human annotator in some cases, it had trouble identifying the smaller vessels that are hardly visible by eye, indicated with a red circle on Figure 26. When comparing the FPN-EfficientNetB5 predictions with the baseline architecture it can clearly be seen that performance improved. First of all, the optimised model detects blood vessels with more certainty, which can be seen in the sharp predictions compared to the blurry ones on the baseline. Additionally, it can generalise better and, thus, more robust, where the U-Net-ResNet101 had trouble identifying the blood vessel missed in the ground-truth annotation.

Figure 27 illustrates that the FPN-EfficientNetB5 is also able to detect very small blood vessels if the lighting is a bit better. Furthermore it can be seen that the networks sometimes have trouble in correctly segmenting the edges of the image frame indicated with the red circle in Figure 27. This can be explained

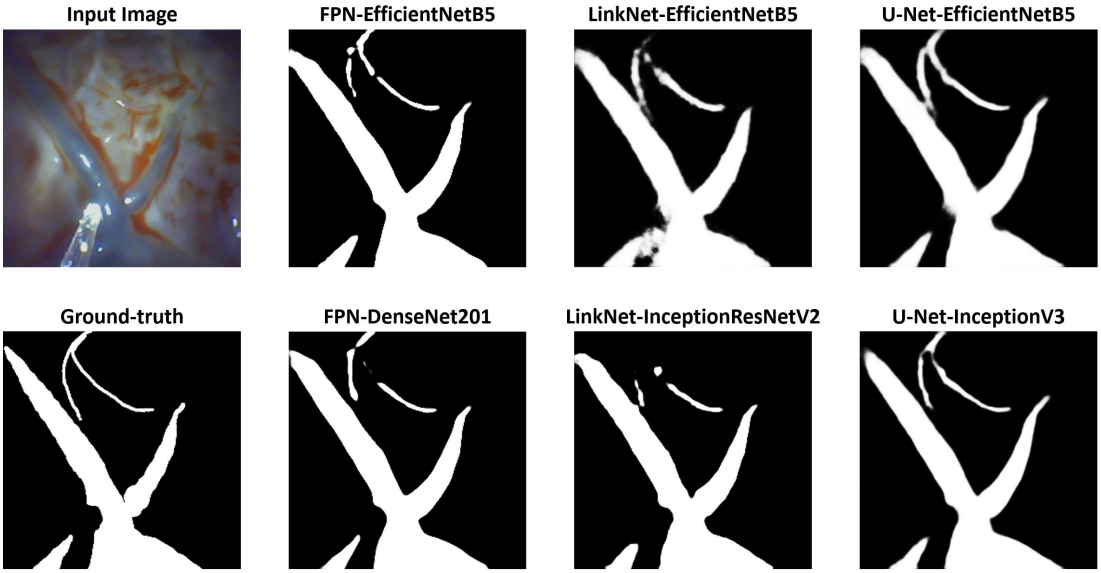

Figure 25: Segmentation results of the optimised networks

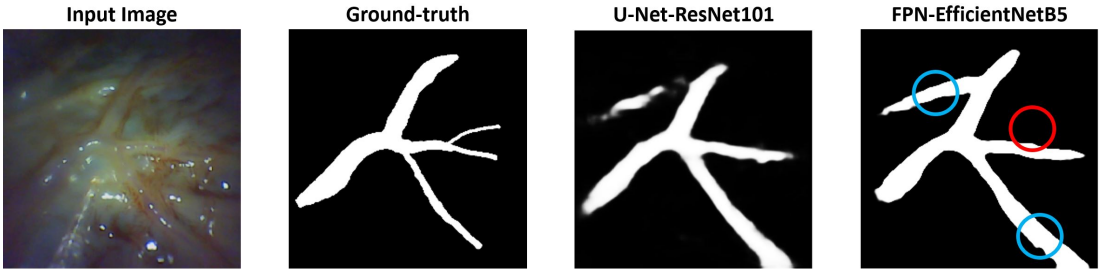

Figure 26: Qualitative comparison of FPN-EfficientNetB5 with the U-Net-ResNet101

through the bad lighting on the edges, but should not be a big problem since the surgeon can move the fetoscope to centre the position.

To get a better overall picture of the performance on the test set, Figure 28 visualizes the segmentation on two more test images. When looking at the segmentation performance of the best performing network, the FPN-EfficientNetB5, it can be observed that it can reliably detect blood vessels in different sizes under different lighting conditions on images.

5.7 Video Segmentation

The models reported in Section 5.6 have shown to detect blood vessels on static images reliably. However, to be useful for clinical use, they have to perform well

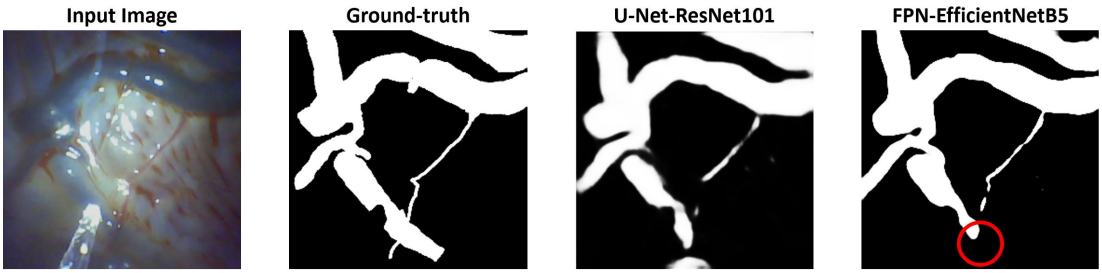

Figure 27: Comparison of FPN-EfficientNetB5 with the U-Net-ResNet101 as a baseline

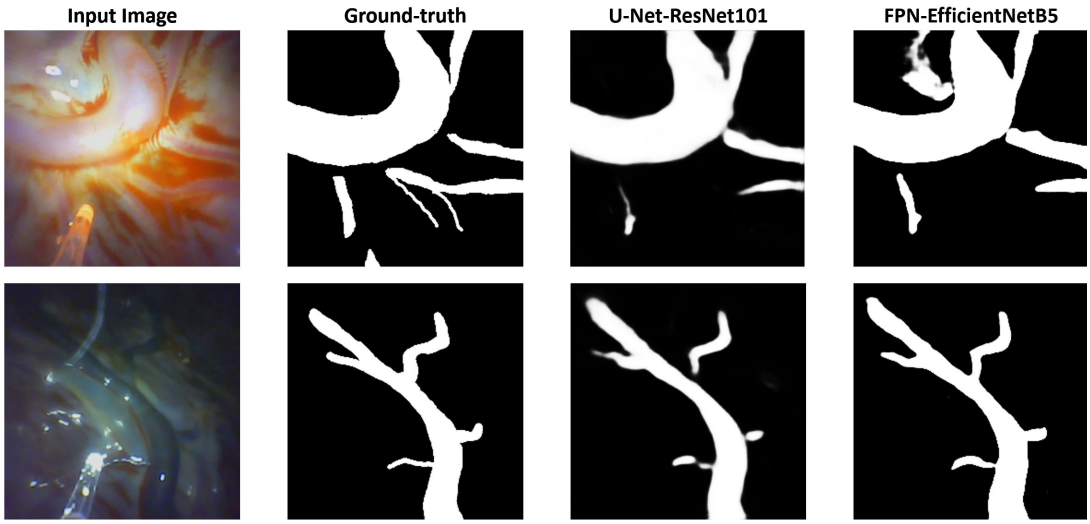

Figure 28: Segmentation results of FPN-EfficientNetB5 on test images compared to a baseline architecture

on video sequences. The FPN with EfficientNetB5 [36] backbone achieved the best overall performance on static images outperforming 25 other segmentation models. Therefore, this network is selected as the segmentation architecture for detecting placental blood vessels on videos obtained through the MSRL fetoscopic system. Since no other video material of the placental region was available, the videos are reused that resulted in the MSRL dataset described in Section 5.1.2. In a first step binary video segmentations were made, where the segmentation network created a prediction on every frame of the video. The prediction image was post-processed using a threshold of 0.5 that resulted in a binary segmentation image illustrated in Figure 29.

The binary segmentation videos showed that the FPN-EfficientNetB5 could also detect blood vessels on recorded video sequences. Although creating only

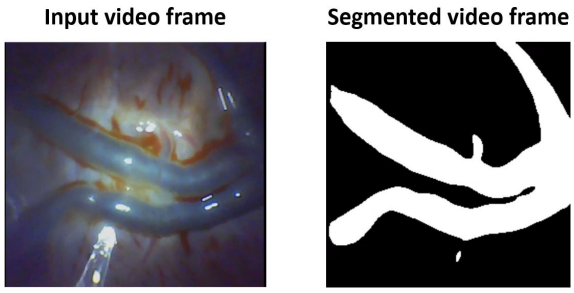

Figure 29: Segmentation with FPN-EfficientNetB5 on a video frame

a segmentation video that runs parallel to the fetoscopic video can be useful to detect blood vessels, it is not very practical during surgery to have two videos running. The surgeon would benefit the most if the predicted segmentation can directly be seen on the original input fetoscopic video frames. Two different ways of enhancing the view are presented in Figure 30, which both assist the surgeon in various situations.

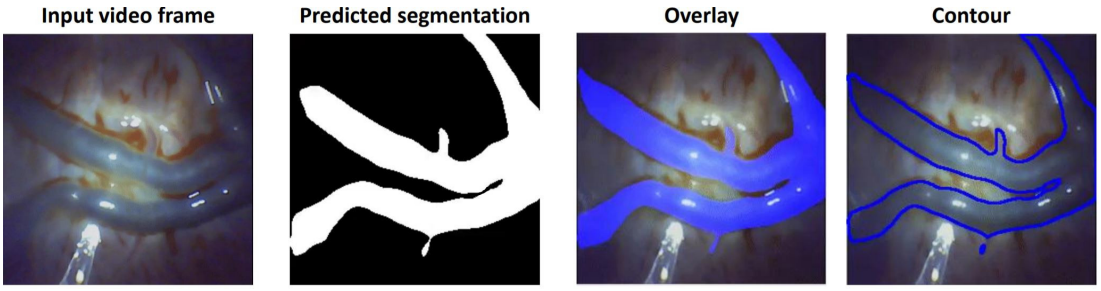

Figure 30: Visualisation methods of predictions on fetoscopic video

The first possibility overlays the predicted segmentation on the fetoscopic video, which highlights the whole blood vessel. This modality can be useful to detect the blood vessels in the first place when the visibility is terrible due to problematic light conditions or turbidity in the amniotic fluid.
The second way of enhancing the view on the blood vessel is highlighting their edges, which is useful if the surgeon needs a clear view of the anastomoses while lasering them. Since both modalities are based on the binary segmentation, there exists no difference in performance in detecting blood vessels. The vision is that the surgeon can switch between the different modalities depending on which best suits at the given moment or with which they are most comfortable working.

6 Discussion

In this chapter the results of the experiments presented in Chapter 5 are discussed. The Discussion focuses on three of the most important parts for detecting placental blood vessels in an intraoperative environment using deep learning techniques. As for every deep learning model, a dataset is needed, which is discussed in Section 6.1. To make accurate predictions on the fetoscopic video frames, a neural network is necessary, where the results are discussed in Section 6.2 followed by the discussion on the performance in video sequences.

6.1 Dataset

The foundation of a well-performing deep learning model lies in a useful dataset. The used dataset was created with the new MSRL fetoscopic system as part of this thesis and is presented in Section 5.1.2. The acquired data represents the variety in the placental region with blood vessels in all sizes and colours. From massive red blood vessels to thin blood vessels with very light colouring, all can be observed. Overall the dataset acts as a good baseline for demonstrating the automatic identification of placental blood vessels.
Although the dataset is a solid baseline for testing networks in the placental region, there is room for improvement in quality and quantity. Since a non-expert human rater created the dataset annotations, they are not optimal with some erroneous manual annotations as seen in Section 5.6. Additionally, minimal variety is present in the dataset because only a single placenta could be used for its creation. In contrast, other placental datasets [6, 5] used data from multiple surgeries, resulting in more diverse data for training the deep-learning models. While other datasets were able to train on images where lighting conditions are poor, and many turbidities are present, the used dataset could not recreate those situations. Even if the models could detect the blood vessels, the dataset can be improved with more data covering all the difficult conditions. A more realistic experimental setup with fluid and turbidity would help the dataset improve quality, leading to more robust network performance.

6.2 Networks

The networks used have shown to segment placental blood vessels reliably on both datasets tested achieving state-of-the-art performance. With 26 different models, a wide variety of different segmentation networks with well-known feature extractors has been tested and compared, resulting in multiple similar performing segmentation networks that could potentially be used for blood vessel detection. The goal in choosing a network was on achieving the best possible performance on the provided datasets. While training and evaluating the models, it could be observed that the best performing models require a lot of processing power, which led to errors when trying to change the batch size.
For the networks to be broadly applicable without massive processor units needed, improvements could be made in the size of the deep-learning models. Nevertheless, this decision is not as easy as it seems, since reducing the size leads to a decrease in performance, where a trade-off has to be made between size and performance. A possible solution would be to use a smaller segmentation network with more fine-tuning, leading to similar results as the models presented.

6.3 Video Segmentation

The proposed networks transformed well onto the video sequences as they were able to identify the blood vessels on the fetoscopic video frames quickly. However, there are some limitations to this, as the models operate on still video frames and therefore ignore the temporal information of the video. This information could be useful to minimise false predictions which are present in one video frame and disappear in the next that lead to a flickering in the segmentation. Additionally generating a prediction on every single video frame uses very much computational power, which is unnecessary to detect blood vessels. Since the surgeon moves rather slowly in the surgery, it would suffice to predict frames at a lower rate, decreasing the computational power needed. As the models could only be tested on pre-recorded video sequences, no exact information is available on how the models perform on real-time fetoscopic videos.

7 Conclusion

7.1 Summary of the Results

This thesis is a part of a project at the MSRL that develops a new fetoscopic system for Twin-to-Twin Transfusion Syndrome treatment. The thesis focuses on implementing automated blood vessel detection using deep learning models. This feature can be important in the surgery, as the view of the placental blood vessels can be restricted due to environmental factors such as the difficult lighting conditions or turbidity in the amniotic fluid.

In first tests, deep-learning models were trained on the only publicly available dataset of the placental region published by Bano et al. [6]. Near state-of-the-art performance was achieved quickly on this dataset. The trained models were further used to test on the data acquired with the new MSRL fetoscope, resulting in very poor segmentation. The experiment showed that the models could not generalize to a new fetoscope, which means that the deep learning models need to be trained on a new dataset with images obtained through the new MSRL fetoscope.

A new dataset of the placental region was created as part of the thesis consisting of 270 manually annotated images, which can be used to further research placental vessel segmentation. The dataset was created from a single placenta in an ex vivo setup with the MSRL fetoscopic system used to obtain images.

The newly created dataset was then used to train and compare 26 different deep learning models through six-fold cross-validation over the dataset. The best performing models resulting from the cross-validation were then further optimised through fine-tuning parameters. Tuning parameters such as the loss functions, the batch size and the initial learning rate resulted in the overall best performing model, the FPN architecture using EfficentNetB5 [36] as an encoder network. The proposed architecture was evaluated through quantitative and qualitative comparison with the model used in [6] and other optimised networks where it outperformed the others. The network achieved an overall IoU score of 0.68 on the test set, which resulted in accurate blood vessel segmentations.

7.2 Future Work

Due to time and resource limitations, the deep-learning model was only tested on prerecorded video sequences of a single placenta. In the future, the model should be tested in realtime on a new placenta within the existing fetoscopic system developed at the MSRL. The application in realtime can lead to delays in the predictions, so it is of interest to see how the model performs. Additionally, it is of interest if the model can generalise in detecting vessels on a new, never before seen placenta or if any changes in size or architecture need to be made.

A necessary improvement is increasing the dataset with annotated images of more placentas representing multiple different conditions. As of now, only a single ex vivo placenta is used. In reality, the surgery is performed in vivo, where amniotic fluid is present during the whole procedure. Increasing the quality and quantity of the dataset leads to better generalisation performance of the segmentation networks. This can be done by changing the data acquisition setup and letting an expert clinician annotate the training images.

The models created for automatically segmenting blood vessels is not limited to only highlighting blood vessel on the fetoscopic video. The segmentation can further be used to create vascular maps of the chorionic plate using mosaicking algorithms that extend the field of view of the surgeon and make fast localization of abnormal blood vessels easier.

References

[1] A. Baschat, R. H. Chmait, J. Deprest, E. Gratacós, K. Hecher, E. Kontopoulos, R. Quintero, D. W. Skupski, D. V. Valsky, and Y. Ville, "Twin-to-twin transfusion syndrome (ttts)*," *Journal of Perinatal Medicine*, vol. 39, no. 2, pp. 107 – 112, 01 Mar. 2011.

[2] J. M. M. van Klink, H. M. Koopman, M. Rijken, J. M. Middeldorp, D. Oepkes, and E. Lopriore, "Long-term neurodevelopmental outcome in survivors of twin-to-twin transfusion syndrome," *Twin Research and Human Genetics*, vol. 19, no. 3, p. 255–261, 2016.

[3] C. Bamberg and K. Hecher, "Update on twin-to-twin transfusion syndrome," *Best Practice and Research Clinical Obstetrics and Gynaecology*, vol. 58, 07 2019.

[4] F. I. Luks, "Schematic illustration of endoscopic fetal surgery for twin-to-twin transfusion syndrome." `https://neoreviews.aappublications.org/content/neoreviews/18/1/e22/F1.large.jpg`, 2009. [Online; accessed 12-January-2021].

[5] P. Sadda, M. Imamoglu, M. Dombrowski, X. Papademetris, M. Bahtiyar, and J. Onofrey, "Deep-learned placental vessel segmentation for intraoperative video enhancement in fetoscopic surgery," *International Journal of Computer Assisted Radiology and Surgery*, vol. 14, 11 2018.

[6] S. Bano, F. Vasconcelos, L. M. Shepherd, E. Vander Poorten, T. Vercauteren, S. Ourselin, A. L. David, J. Deprest, and D. Stoyanov, "Deep placental vessel segmentation for fetoscopic mosaicking," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, 2020.

[7] L. Lewi, J. Jani, I. Blickstein, A. Huber, L. Gucciardo, T. Van Mieghem, E. Doné, A.-S. Boes, K. Hecher, E. Gratacós, P. Lewi, and J. Deprest, "The outcome of monochorionic diamniotic twin gestations in the era of invasive fetal therapy: a prospective cohort study," *American Journal of Obstetrics and Gynecology*, vol. 199, no. 5, pp. 514.e1 – 514.e8, 2008.

[8] SSMHealth, "Twin to twin transfusion syndrome." https://www.ssmhealth.com/cardinal-glennon/fetal-care-institute/twin-abnormalities/twin-twin-transfusion-syndrome. [Online; accessed 12-January-2021].

[9] W. Diehl, K. Hecher, L. Zikulnig, M. Vetter, and B.-J. Hackelöer, "Placental vascular anastomoses visualized during fetoscopic laser surgery in severe mid-trimester twin–twin transfusion syndrome," *Placenta*, vol. 22, no. 10, pp. 876 – 881, 2001.

[10] R. Quintero, W. Morales, M. Allen, P. Bornick, P. Johnson, and M. Kruger, "Staging of twin-twin transfusion syndrome," *Journal of perinatology : official journal of the California Perinatal Association*, vol. 19, pp. 550–5, 1999.

[11] F. Slaghekke, E. Lopriore, L. Lewi, J. M. Middeldorp, E. W. van Zwet, A.-S. Weingertner, F. J. Klumper, P. DeKoninck, R. Devlieger, M. D. Kilby, M. A. Rustico, J. Deprest, R. Favre, and D. Oepkes, "Fetoscopic laser coagulation of the vascular equator versus selective coagulation for twin-to-twin transfusion syndrome: an open-label randomised controlled trial," *The Lancet*, vol. 383, no. 9935, pp. 2144 – 2151, 2014.

[12] H. Sago, K. Ishii, R. Sugibayashi, K. Ozawa, M. Sumie, and S. Wada, "Fetoscopic laser photocoagulation for twin-twin transfusion syndrome: Fetoscopic laser photocoagulation," *Journal of Obstetrics and Gynaecology Research*, vol. 44, 02 2018.

[13] J. Akkermans, S. Peeters, F. Klumper, E. Lopriore, J. Middeldorp, and D. Oepkes, "Twenty-five years of fetoscopic laser coagulation in twin-twin transfusion syndrome: A systematic review," *Fetal Diagn Ther*, vol. 38, 08 2015.

[14] P. A. Campochiaro, "Molecular pathogenesis of retinal and choroidal vascular diseases," *Progress in Retinal and Eye Research*, vol. 49, pp. 67 – 81, 2015.

[15] S. Moccia, E. De Momi, S. El Hadji, and L. S. Mattos, "Blood vessel segmentation algorithms — review of methods, datasets and evaluation metrics," *Computer Methods and Programs in Biomedicine*, vol. 158, pp. 71 – 91, 2018.

[16] A. Hoover, V. Kouznetsova, and M. Goldbaum, "Locating blood vessels in retinal images by piecewise threshold probing of a matched filter response," *IEEE transactions on medical imaging*, vol. 19, pp. 203–10, 04 2000.

[17] J. Staal, M. D. Abramoff, M. Niemeijer, M. A. Viergever, and B. van Ginneken, "Ridge-based vessel segmentation in color images of the retina," *IEEE Transactions on Medical Imaging*, vol. 23, no. 4, pp. 501–509, 2004.

[18] F. Gaisser, S. Peeters, B. Lenseigne, P. Jonker, and D. Oepkes, "Stable image registration for in-vivo fetoscopic panorama reconstruction," *Journal of Imaging*, vol. 4, p. 24, 01 2018.

[19] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *Lecture Notes in Computer Science*, p. 21–37, 2016.

[20] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[21] A. F. Frangi, W. J. Niessen, K. L. Vincken, and M. A. Viergever, "Multiscale vessel enhancement filtering," in *Medical Image Computing and Computer-Assisted Intervention — MICCAI'98* (W. M. Wells, A. Colchester, and S. Delp, eds.), (Berlin, Heidelberg), pp. 130–137, Springer Berlin Heidelberg, 1998.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[23] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez, "A review on deep learning techniques applied to semantic segmentation," 2017.

[24] F.-F. Li, J. Johnson, and S. Young, "Lecture notes in convolutional neural networks for visual recognition," May 2018.

[25] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," 2015.

[26] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[27] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[28] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv 1409.1556*, 09 2014.

[30] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," 2016.

[31] Z. Guyu, G. Huang, H. He, H. He, and J. Ren, "Regional spatiotemporal collaborative prediction model for air quality," *IEEE Access*, vol. PP, pp. 1–1, 09 2019.

[32] S. Saha, "A comprehensive guide to convolutional neural networks — the eli5 way." `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a` 2018. [Online; accessed 17-January-2021].

[33] D. Mishra, "Transposed convolution demystified." `https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba`, 2020. [Online; accessed 18-January-2021].

[34] A. Chaurasia and E. Culurciello, "Linknet: Exploiting encoder representations for efficient semantic segmentation," in *2017 IEEE Visual Communications and Image Processing (VCIP)*, pp. 1–4, 2017.

[35] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 936–944, 2017.

[36] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[37] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015.

[38] A. Kirillov, K. He, R. Girshick, and P. Dollár, "A unified architecture for instance and semantic segmentation," 2017.

[39] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6230–6239, 2017.

[40] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," 2016.

[41] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," 2016.

[42] P. Yakubovskiy, "Segmentation models." `https://github.com/qubvel/segmentation_models`, 2019.

[43] S. Bano, F. Vasconcelos, E. Vander Poorten, T. Vercauteren, S. Ourselin, J. Deprest, and D. Stoyanov, "Fetnet: A recurrent convolutional network for occlusion identification in fetoscopic videos," *International Journal of Computer Assisted Radiology and Surgery*, vol. 15, 04 2020.

[44] A. Bréhéret, "Pixel Annotation Tool." `https://github.com/abreheret/PixelAnnotationTool`, 2017.

[45] S. Tomar, "Converting video formats with ffmpeg," *Linux Journal*, vol. 2006, no. 146, p. 10, 2006.

[46] J. Jordan, "Evaluating image segmentation models.." https://www.jeremyjordan.me/evaluating-image-segmentation-models/, 2018. [Online; accessed 4-February-2021].

[47] T. Eelbode, J. Bertels, M. Berman, D. Vandermeulen, F. Maes, R. Bisschops, and M. B. Blaschko, "Optimization for medical image segmentation: Theory and practice when evaluating with dice score or jaccard index," *IEEE Transactions on Medical Imaging*, vol. 39, no. 11, pp. 3679–3690, 2020.

[48] R. Shanmugamani, "Deep learning for computer vision," 2018.

[49] willem (https://stats.stackexchange.com/users/159052/willem), "F1/dice-score vs iou." https://stats.stackexchange.com/q/276144, 2017. [Online; accessed 4-February-2021].

[50] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2018.

[51] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," 2015.

[52] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," 2016.

[53] F. Chollet, *Deep Learning with Python*. Manning, 2018.

[54] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[55] I. Kandel and M. Castelli, "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset," *ICT Express*, vol. 6, no. 4, pp. 312–315, 2020.

[56] M. D. Zeiler, "Adadelta: An adaptive learning rate method," 2012.

[57] M. Jadhav, "Gradient descent: Why and how?." https://medium.com/analytics-vidhya/gradient-descent-why-and-how-e36950ae7d3, 2019. [Online; accessed 14-February-2021].

[58] A. Dertat, "Applied deep learning - part 4: Convolutional neural networks." https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2, 2017. [Online; accessed 16-February-2021].

[59] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[60] F. Chollet *et al.*, "Keras." https://keras.io, 2015.

A How-To Guide

This chapter should help understand the workflow of creating a deep-learning model that can segment images into binary classes. This thesis focused on segmenting fetoscopic images of the chorionic plate into regions with a blood vessel and no blood vessel, which is used as an example for this workflow.

1. The first step to solve any problem using deep-learning techniques is to get high quality and quantity data, described in Section 5.1.2.

2. Create binary segmentation masks using an annotation tool of your choice, where this thesis used the PixelAnnotationTool [44] available for free on GitHub. While creating the dataset, keep in mind that your model tries to learn to perform as good as your created masks, try being as accurate to achieve better results in the segmentation. Figure 31 illustrates how the annotations are made in this thesis using a tablet with a corresponding stylus.

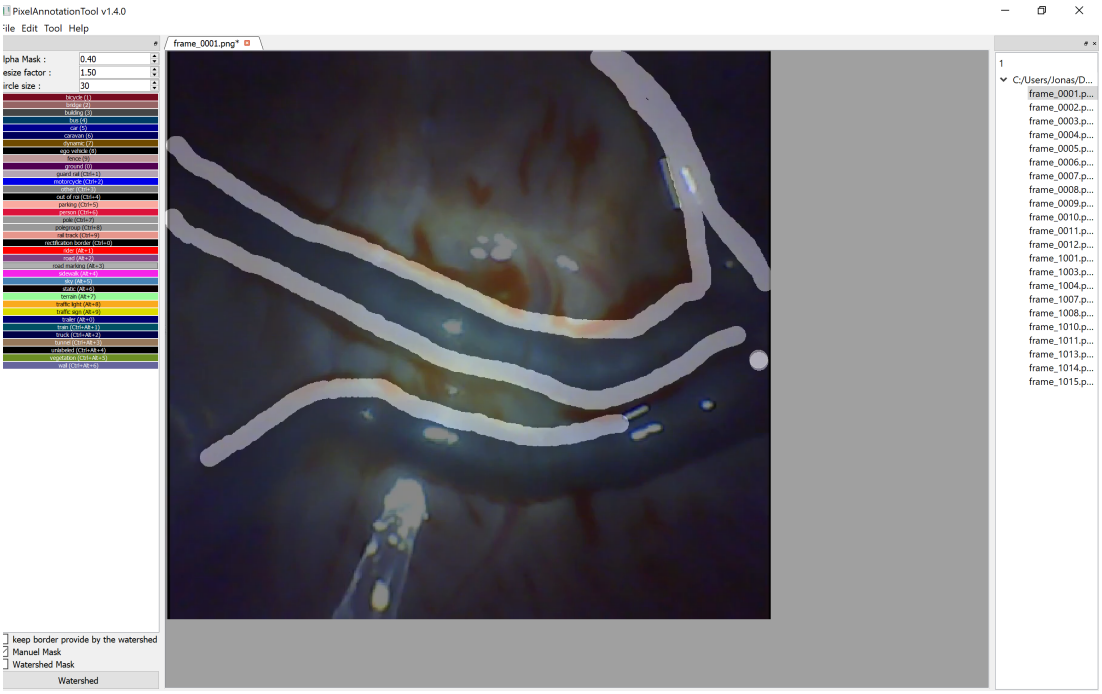

Figure 31: Pixel annotation interface

3. After the dataset is created, randomly split the data into a training, validation

and test set. This can quickly be done using functions from sklearn [59] or Keras [60]. Then save the data using the following folder structure, which is vital for a successful proceeding.

```
Dataset
├── train
│   ├── images
│   │   └── images
│   └── masks
│       └── images
├── validation
│   ├── images
│   │   └── images
│   └── masks
│       └── images
└── test
    ├── images
    │   └── images
    └── masks
        └── images
```

4. Define all paths where the images are saved in your code, where an example of the training images is shown in the following code section.

```
import os

train_path = "Dataset/train"
train_images_path = os.path.join(train_path, 'images')
train_masks_path = os.path.join(train_path, 'masks')
```

5. Define all your hyperparameters and variables as the batch size, initial learning rate, image size, epochs, etc.

```
IMG_WIDTH = 256
IMG_HEIGHT = 256
IMG_CHANNELS = 3 #indicating a colored image
batch = 8
learning_rate = 3e-4
epochs = 1000
```

6. Prepare your data with a generator, where the self-created generator is shown in the code section below. The generator can also augment your data directly during training, which saves much memory.

```
from tensorflow.keras.preprocessing.image import
                                ImageDataGenerator

def generator(img_path, mask_path, batch_size=8,
                IMG_SIZE=(256, 256), seed=69, train=True):
    if train == True:
        data_gen_args = dict(rescale=1./255
                             horizontal_flip=True,
                             vertical_flip=True,
                             rotation_range=160,
                             brightness_range=(0.7, 1),
                             zoom_range=[0.8, 1]
                             )
    else:
        data_gen_args = dict(rescale=1. / 255)

    datagen = ImageDataGenerator(**data_gen_args)

    image_generator = datagen.flow_from_directory(
                                        img_path,
                                        target_size=IMG_SIZE,
                                        class_mode=None,
                                        color_mode='rgb',
                                        batch_size=batch_size,
                                        seed=seed)
    mask_generator = datagen.flow_from_directory(mask_path,
                                        target_size=IMG_SIZE,
                                        class_mode=None,
                                        color_mode='grayscale',
```

```
                                        batch_size=batch_size,
                                        seed=seed)
    return (pair for pair in zip(image_generator,mask_generator))
```

Only the training data must be augmented while the validation and test set are only normalised using the rescaling of the colour values. In this step, all the data is prepared and resized to be fed into the deep-learning model, where each set needs its generator depicted in the next code section.

```
train_generator = generator(train_images_path, train_masks_path,
                                batch, (IMG_WIDTH, IMG_HEIGHT),
                                seed, train=True)
val_generator = generator(val_images_path, val_masks_path,batch,
                                (IMG_WIDTH, IMG_HEIGHT), seed,
                                train=False)
test_generator = generator(test_images_path, test_masks_path,
                                batch, (IMG_WIDTH, IMG_HEIGHT),
                                seed, train=False)
```

7. Define your deep-learning model. Much work is already done in segmenting images, where Yakubovskiy [42] provides a very good starting point for any segmentation problem. Although it is possible to solve most segmentation tasks very good using this library, it is recommended to create the models by yourself. Besides the learnings in how a good segmentation architecture is built, much more control over finer parameters is given, which makes tuning the models a lot easier, leading to better performance. The code section below shows how a simple U-Net architecture with a ResNet101 encoder (backbone) and pre-trained Imagenet weights is defined using the models from Yakubovskiy [42].

```
import segmentation_models as sm

backbone = 'resnet101'

model = sm.Unet(backbone, encoder_weights='imagenet')
```

8. Before the model can be trained, it has to be compiled. You have to define the loss function, the optimiser and the useful evaluation metrics, to be able to build the model. In segmentation tasks, the metrics mostly consist of IoU and Dice scores described in Section 5.3. While some metrics can be used directly from Keras [60], some have to be calculated with a few lines of code.

```
loss = sm.losses.DiceLoss()
opt = keras.optimizers.Adam(learning_rate=lr)
metrics = ["acc", keras.metrics.Recall(), keras.metrics.Precision
                                  (), iou, dice]

model.compile(loss=loss, optimizer=opt, metrics=metrics)
```

9. After compiling the model, callbacks are defined. Callbacks such as Tensorboard are useful to precisely evaluate the performance of your networks after or during training. There are many different callbacks, wherein this thesis mainly the early stopping and learning rate reduction are used that are described in Section 5.4.1.

```
from keras.callbacks import EarlyStopping, ReduceLROnPlateau,
                            TensorBoard

callbacks = [
    TensorBoard(log_dir='logs'),
    ReduceLROnPlateau(monitor='val_loss',
                      factor=0.1, patience=4),
    EarlyStopping(monitor='val_loss', patience=10,
                      restore_best_weights=True)
    ]
```

10. Perform training on the defined model using the built generator for training and validation. This leads to your final trained segmentation model, which can be evaluated and optimised to later test in the left-out test images.

```
model.fit_generator(generator=train_generator,
                    steps_per_epoch=train_steps,
                    validation_data=val_generator,
                    validation_steps=val_steps,
                    epochs=epochs,
                    callbacks=callbacks)
```

11. The optimised model can then be used to segment your images into different classes when done correctly leads to something similar to Figure 32.

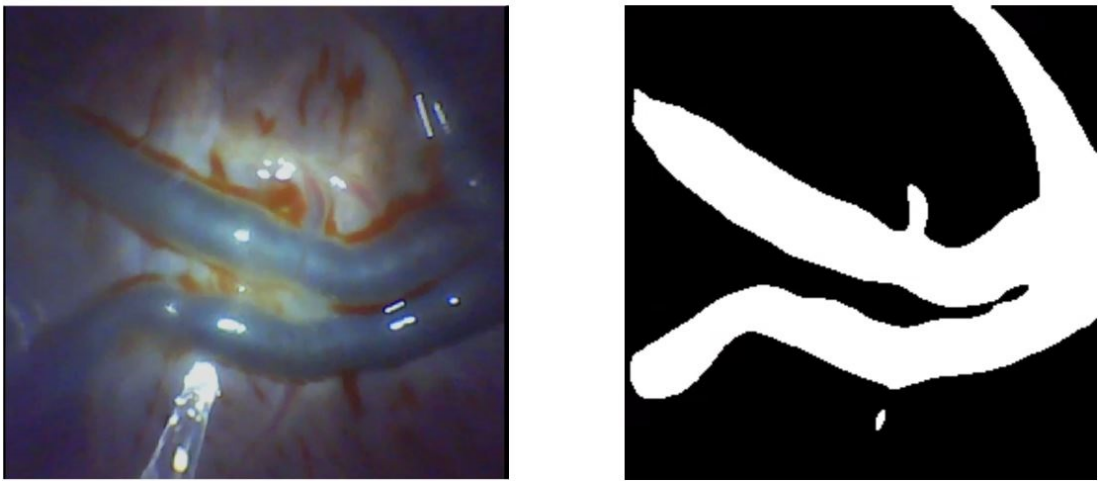

Figure 32: Binary segmentation

The full code used can be found on the authors GitHub (`https://github.com/bohnjonas/Fetopscopy_Python`).

B Cross-Validation Tables

U-Net (UCL)		Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		Fold 6		Overall	
No. training images		1810		1905		2215		1970		2225		1925			
No. validation images		120		101		39		88		37		97		482	
Vanilla	Dice	0.805	0.80 (+/- 0.11)	0.692	0.68 (+/- 0.17)	0.8008	0.78 (+/- 0.10)	0.5851	0.62 (+/- 0.22)	0.671	0.68 (+/- 0.13)	0.698	0.65 (+/- 0.16)	0.7087	0.7 (+/- 0.15)
	IoU	0.6745	0.67 (+/- 0.13)	0.5367	0.54 (+/- 0.17)	0.6696	0.65 (+/- 0.13)	0.4195	0.48 (+/- 0.23)	0.511	0.53 (+/- 0.14)	0.5392	0.50 (+/- 0.17)	0.5589	0.56 (+/- 0.17)
VGG16	Dice	0.8231	0.81 (+/- 0.09)	0.7257	0.69 (+/- 0.17)	0.8109	0.79 (+/- 0.10)	0.5558	0.57 (+/- 0.24)	0.6575	0.68 (+/- 0.14)	0.554	0.53 (+/- 0.17)	0.686	0.67 (+/- 0.15)
	IoU	0.7001	0.69 (+/- 0.12)	0.5738	0.55 (+/- 0.18)	0.6835	0.66 (+/- 0.13)	0.3916	0.44 (+/- 0.24)	0.5036	0.53 (+/- 0.15)	0.3863	0.38 (+/- 0.16)	0.5377	0.54 (+/- 0.17)
DenseNet201	Dice	0.8345	0.81 (+/- 0.08)	**0.7528**	**0.73 (+/- 0.16)**	0.8356	0.80 (+/- 0.11)	0.7328	0.73 (+/- 0.18)	**0.7186**	**0.73 (+/- 0.14)**	0.6308	0.57 (+/- 0.16)	0.749	0.72 (+/- 0.14)
	IoU	0.7165	0.68 (+/- 0.10)	**0.6082**	**0.59 (+/- 0.17)**	0.7187	0.68 (+/- 0.14)	0.5825	0.61 (+/- 0.19)	**0.5691**	**0.59 (+/- 0.16)**	0.4623	0.42 (+/- 0.15)	0.607	0.59 (+/- 0.15)
EfficientNetB5	Dice	0.8399	0.79 (+/- 0.10)	0.7155	0.70 (+/- 0.16)	0.822	0.79 (+/- 0.13)	**0.7426**	**0.75 (+/- 0.15)**	0.6296	0.70 (+/- 0.16)	**0.7671**	**0.72 (+/- 0.13)**	**0.7638**	**0.74 (+/- 0.13)**
	IoU	0.7248	0.67 (+/- 0.11)	0.5618	0.55 (+/- 0.17)	0.6996	0.66 (+/- 0.15)	**0.5955**	**0.62 (+/- 0.18)**	0.478	0.56 (+/- 0.17)	**0.6241**	**0.57 (+/- 0.15)**	**0.6258**	**0.61 (+/- 0.15)**
Inception ResNetV2	Dice	**0.8417**	**0.82 (+/- 0.08)**	0.6899	0.65 (+/- 0.20)	**0.84**	**0.82 (+/- 0.09)**	0.695	0.71 (+/- 0.19)	0.6853	0.69 (+/- 0.16)	0.7595	0.71 (+/- 0.17)	0.7544	0.73 (+/- 0.15)
	IoU	**0.7275**	**0.70 (+/- 0.10)**	0.5319	0.51 (+/- 0.20)	**0.7254**	**0.70 (+/- 0.12)**	0.5388	0.58 (+/- 0.20)	0.5289	0.55 (+/- 0.17)	0.6136	0.57 (+/- 0.18)	0.6137	0.6 (+/- 0.16)
InceptionV3	Dice	0.8342	0.80 (+/- 0.11)	0.6872	0.64 (+/- 0.18)	0.8389	0.81 (+/- 0.10)	0.7107	0.73 (+/- 0.14)	0.6534	0.72 (+/- 0.14)	0.7137	0.65 (+/- 0.18)	0.7431	0.72 (+/- 0.15)
	IoU	0.7168	0.68 (+/- 0.13)	0.5274	0.49 (+/- 0.17)	0.7239	0.69 (+/- 0.13)	0.5557	0.60 (+/- 0.17)	0.5044	0.58 (+/- 0.15)	0.5591	0.51 (+/- 0.18)	0.6002	0.58 (+/- 0.16)

LinkNet (UCL)		Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		Fold 6		Overall	
No. training images		1810		1905		2215		1970		2225		1925			
No. validation images		120		101		39		88		37		97		482	
Vanilla	Dice	0.8093	0.80 (+/- 0.08)	0.7112	0.69 (+/- 0.15)	0.801	0.78 (+/- 0.10)	0.5965	0.61 (+/- 0.23)	0.6806	0.69 (+/- 0.13)	0.5959	0.52 (+/- 0.19)	0.6964	0.67 (+/- 0.15)
	IoU	0.6802	0.67 (+/- 0.10)	0.5562	0.54 (+/- 0.16)	0.6696	0.65 (+/- 0.13)	0.433	0.48 (+/- 0.23)	0.5201	0.54 (+/- 0.14)	0.4284	0.37 (+/- 0.17)	0.5453	0.54 (+/- 0.16)
VGG16	Dice	0.8363	0.82 (+/- 0.09)	0.738	0.69 (+/- 0.13)	0.8073	0.78 (+/- 0.12)	0.6259	0.66 (+/- 0.21)	0.6614	0.68 (+/- 0.14)	0.5379	0.48 (+/- 0.18)	0.7015	0.68 (+/- 0.14)
	IoU	0.7192	0.71 (+/- 0.11)	0.5871	0.54 (+/- 0.14)	0.6782	0.65 (+/- 0.15)	0.4608	0.53 (+/- 0.22)	0.5014	0.53 (+/- 0.15)	0.3719	0.33 (+/- 0.16)	0.5544	0.55 (+/- 0.15)
DenseNet201	Dice	**0.849**	**0.84 (+/- 0.07)**	**0.7578**	0.71 (+/- 0.20)	0.837	**0.81 (+/- 0.10)**	0.7216	**0.74 (+/- 0.16)**	0.6297	0.68 (+/- 0.14)	0.6709	0.61 (+/- 0.18)	0.753	0.73 (+/- 0.14)
	IoU	**0.7382**	**0.73 (+/- 0.10)**	**0.6148**	0.57 (+/- 0.19)	0.7205	**0.70 (+/- 0.12)**	0.5702	**0.60 (+/- 0.18)**	0.4722	0.53 (+/- 0.14)	0.5066	0.47 (+/- 0.17)	0.6132	0.6 (+/- 0.15)
EfficientNetB5	Dice	0.8388	0.80 (+/- 0.08)	0.729	0.70 (+/- 0.18)	0.8436	0.78 (+/- 0.15)	**0.7404**	0.73 (+/- 0.18)	0.6285	0.71 (+/- 0.15)	**0.7408**	**0.70 (+/- 0.13)**	**0.7624**	**0.74 (+/- 0.14)**
	IoU	0.7231	0.68 (+/- 0.10)	0.5776	0.56 (+/- 0.18)	0.731	0.66 (+/- 0.17)	**0.5951**	0.60 (+/- 0.19)	0.4751	0.57 (+/- 0.16)	**0.591**	**0.56 (+/- 0.15)**	**0.6243**	**0.61 (+/- 0.15)**
Inception ResNetV2	Dice	0.8293	0.80 (+/- 0.11)	0.6959	0.67 (+/- 0.18)	**0.8441**	0.81 (+/- 0.10)	0.7175	0.73 (+/- 0.18)	0.6015	0.66 (+/- 0.16)	0.7141	0.67 (+/- 0.17)	0.7415	0.72 (+/- 0.15)
	IoU	0.7096	0.68 (+/- 0.13)	0.5403	0.53 (+/- 0.18)	**0.7316**	0.69 (+/- 0.13)	0.565	0.60 (+/- 0.19)	0.4458	0.51 (+/- 0.16)	0.558	0.52 (+/- 0.17)	0.5987	0.59 (+/- 0.16)
InceptionV3	Dice	0.8313	0.81 (+/- 0.10)	0.7382	**0.71 (+/- 0.17)**	0.8297	0.80 (+/- 0.09)	0.6812	0.71 (+/- 0.17)	**0.693**	**0.72 (+/- 0.12)**	0.7119	0.63 (+/- 0.20)	0.7496	0.73 (+/- 0.15)
	IoU	0.7123	0.69 (+/- 0.12)	0.5889	**0.57 (+/- 0.18)**	0.7105	0.68 (+/- 0.12)	0.5206	0.57 (+/- 0.19)	**0.5364**	**0.58 (+/- 0.14)**	0.5549	0.48 (+/- 0.20)	0.6061	0.59 (+/- 0.16)

FPN (UCL)		Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		Fold 6		Overall	
No. training images		1810		1905		2215		1970		2225		1925			
No. validation images		120		101		39		88		37		97		482	
VGG16	Dice	0.7805	0.78 (+/- 0.08)	0.7331	0.67 (+/- 0.18)	0.8087	0.78 (+/- 0.11)	0.6549	0.67 (+/- 0.21)	0.6665	0.69 (+/- 0.14)	0.6627	0.59 (+/- 0.17)	0.7175	0.69 (+/- 0.15)
	IoU	0.6407	0.65 (+/- 0.10)	0.583	0.53 (+/- 0.18)	0.6804	0.66 (+/- 0.14)	0.4942	0.54 (+/- 0.22)	0.509	0.55 (+/- 0.15)	0.4978	0.44 (+/- 0.17)	0.5662	0.56 (+/- 0.16)
DenseNet201	Dice	0.8484	**0.84 (+/- 0.09)**	0.7545	0.73 (+/- 0.15)	0.838	0.79 (+/- 0.13)	0.7122	0.72 (+/- 0.18)	**0.7295**	**0.73 (+/- 0.14)**	0.7352	0.63 (+/- 0.20)	0.7711	0.74 (+/- 0.15)
	IoU	0.7374	**0.73 (+/- 0.11)**	0.6121	0.59 (+/- 0.16)	0.7226	0.68 (+/- 0.16)	0.5598	0.59 (+/- 0.20)	**0.5823**	**0.59 (+/- 0.16)**	0.5832	0.49 (+/- 0.19)	0.6346	0.61 (+/- 0.16)
EfficientNetB5	Dice	**0.8488**	0.81 (+/- 0.07)	0.7267	0.69 (+/- 0.18)	0.8366	0.80 (+/- 0.13)	**0.7193**	**0.74 (+/- 0.18)**	0.562	0.48 (+/- 0.17)	**0.7686**	**0.70 (+/- 0.14)**	0.7604	0.72 (+/- 0.14)
	IoU	**0.7378**	0.69 (+/- 0.09)	0.5746	0.55 (+/- 0.18)	0.7209	0.68 (+/- 0.15)	**0.5718**	**0.61 (+/- 0.19)**	0.4037	0.33 (+/- 0.15)	**0.6254**	**0.56 (+/- 0.16)**	0.6237	0.59 (+/- 0.15)
Inception ResNetV2	Dice	0.8475	0.83 (+/- 0.09)	0.7587	**0.74 (+/- 0.14)**	**0.8472**	0.80 (+/- 0.11)	0.7044	0.73 (+/- 0.18)	0.719	0.72 (+/- 0.13)	0.7597	0.70 (+/- 0.17)	**0.7752**	**0.75 (+/- 0.14)**
	IoU	0.7372	0.72 (+/- 0.12)	0.6135	**0.60 (+/- 0.16)**	**0.7363**	0.68 (+/- 0.14)	0.5491	0.60 (+/- 0.20)	0.5675	0.58 (+/- 0.15)	0.6147	0.56 (+/- 0.18)	**0.6392**	**0.63 (+/- 0.16)**
InceptionV3	Dice	0.8471	0.83 (+/- 0.09)	**0.7707**	0.73 (+/- 0.13)	0.8445	**0.81 (+/- 0.10)**	0.6921	0.72 (+/- 0.17)	0.6239	0.70 (+/- 0.18)	0.7603	0.68 (+/- 0.19)	0.768	0.74 (+/- 0.14)
	IoU	0.7354	0.71 (+/- 0.11)	**0.6289**	0.59 (+/- 0.15)	0.7325	**0.69 (+/- 0.13)**	0.5336	0.59 (+/- 0.18)	0.4792	0.57 (+/- 0.18)	0.6147	0.54 (+/- 0.19)	0.632	0.62 (+/- 0.15)

PSPNet (UCL)		Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		Fold 6		Overall	
No. training images		1810		1905		2215		1970		2225		1925			
No. validation images		120		101		39		88		37		97		482	
VGG16	Dice	0.8069	0.79 (+/- 0.09)	0.6937	0.66 (+/- 0.17)	0.7722	0.73 (+/- 0.16)	0.5638	0.59 (+/- 0.22)	0.6751	0.67 (+/- 0.12)	0.6082	0.56 (+/- 0.16)	0.6859	0.67 (+/- 0.15)
	IoU	0.677	0.67 (+/- 0.11)	0.5351	0.51 (+/- 0.17)	0.632	0.60 (+/- 0.18)	0.3965	0.45 (+/- 0.22)	0.5133	0.51 (+/- 0.13)	0.4403	0.41 (+/- 0.16)	0.5322	0.53 (+/- 0.16)
DenseNet201	Dice	0.8219	0.80 (+/- 0.10)	0.6916	0.65 (+/- 0.18)	0.8028	0.76 (+/- 0.14)	0.6212	0.65 (+/- 0.19)	**0.6794**	**0.69 (+/- 0.13)**	0.6162	0.55 (+/- 0.20)	0.7041	0.68 (+/- 0.16)
	IoU	0.6985	0.67 (+/- 0.12)	0.532	0.51 (+/- 0.18)	0.6719	0.62 (+/- 0.16)	0.4554	0.51 (+/- 0.20)	**0.5207**	**0.54 (+/- 0.15)**	0.4485	0.40 (+/- 0.18)	0.5531	0.54 (+/- 0.16)
EfficientNetB5	Dice	0.8116	0.79 (+/- 0.10)	0.6754	0.65 (+/- 0.19)	0.7669	0.71 (+/- 0.20)	**0.6522**	**0.67 (+/- 0.18)**	0.6314	0.66 (+/- 0.13)	0.6919	**0.68 (+/- 0.15)**	**0.7124**	**0.7 (+/- 0.15)**
	IoU	0.6845	0.67 (+/- 0.11)	0.5144	0.51 (+/- 0.18)	0.624	0.58 (+/- 0.21)	**0.4872**	**0.54 (+/- 0.19)**	0.4645	0.51 (+/- 0.14)	0.5312	**0.54 (+/- 0.16)**	**0.5602**	**0.57 (+/- 0.16)**
Inception ResNetV2	Dice	**0.8239**	**0.81 (+/- 0.09)**	**0.7148**	**0.70 (+/- 0.17)**	**0.804**	**0.77 (+/- 0.13)**	0.6418	0.67 (+/- 0.20)	0.5811	0.66 (+/- 0.14)	0.6375	0.54 (+/- 0.20)	0.71	0.69 (+/- 0.16)
	IoU	**0.7011**	**0.69 (+/- 0.11)**	**0.5594**	**0.56 (+/- 0.17)**	**0.6737**	**0.64 (+/- 0.16)**	0.4773	0.53 (+/- 0.21)	0.4133	0.51 (+/- 0.15)	0.4704	0.40 (+/- 0.19)	0.5598	0.56 (+/- 0.16)
InceptionV3	Dice	0.7973	0.78 (+/- 0.10)	0.6764	0.64 (+/- 0.20)	0.7773	0.73 (+/- 0.15)	0.6143	0.66 (+/- 0.19)	0.5452	0.61 (+/- 0.16)	**0.6925**	0.66 (+/- 0.16)	0.6965	0.69 (+/- 0.16)
	IoU	0.6643	0.65 (+/- 0.12)	0.5159	0.50 (+/- 0.19)	0.6368	0.59 (+/- 0.17)	0.4475	0.52 (+/- 0.20)	0.3829	0.46 (+/- 0.16)	**0.5314**	0.51 (+/- 0.17)	0.5431	0.55 (+/- 0.17)

Top networks (UCL)		Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		Fold 6		Overall	
No. training images		1810		1905		2215		1970		2225		1925			
No. validation images		120		101		39		88		37		97		482	
U-Net	Dice	0.8399	0.79 (+/- 0.10)	0.7155	0.70 (+/- 0.16)	0.822	0.79 (+/- 0.13)	**0.7426**	**0.75 (+/- 0.15)**	0.6296	0.70 (+/- 0.16)	**0.7671**	**0.72 (+/- 0.13)**	0.7638	0.74 (+/- 0.13)
EfficientNetB5	IoU	0.7248	0.67 (+/- 0.11)	0.5618	0.55 (+/- 0.17)	0.6996	0.66 (+/- 0.15)	**0.5955**	**0.62 (+/- 0.18)**	0.478	0.56 (+/- 0.17)	**0.6241**	**0.57 (+/- 0.15)**	0.6258	0.61 (+/- 0.15)
LinkNet	Dice	0.8388	0.80 (+/- 0.08)	0.729	0.70 (+/- 0.18)	0.8436	0.78 (+/- 0.15)	0.7404	0.73 (+/- 0.18)	0.6285	0.71 (+/- 0.15)	0.7408	0.70 (+/- 0.13)	0.7624	0.74 (+/- 0.14)
EfficientNetB5	IoU	0.7231	0.68 (+/- 0.10)	0.5776	0.56 (+/- 0.18)	0.731	0.66 (+/- 0.17)	0.5951	0.60 (+/- 0.19)	0.4751	0.57 (+/- 0.16)	0.591	0.56 (+/- 0.15)	0.6243	0.61 (+/- 0.15)
FPN	Dice	0.8475	0.83 (+/- 0.09)	**0.7587**	**0.74 (+/- 0.14)**	**0.8472**	**0.80 (+/- 0.11)**	0.7044	0.73 (+/- 0.18)	0.719	0.72 (+/- 0.13)	0.7597	0.70 (+/- 0.17)	**0.7752**	**0.75 (+/- 0.14)**
InceptionResNetV2	IoU	0.7372	0.72 (+/- 0.12)	**0.6135**	**0.60 (+/- 0.16)**	**0.7363**	**0.68 (+/- 0.14)**	0.5491	0.60 (+/- 0.20)	0.5675	0.58 (+/- 0.15)	0.6147	0.56 (+/- 0.18)	**0.6392**	**0.63 (+/- 0.16)**
FPN	Dice	**0.8484**	**0.84 (+/- 0.09)**	0.7545	0.73 (+/- 0.15)	0.838	0.79 (+/- 0.13)	0.7122	0.72 (+/- 0.18)	**0.7295**	**0.73 (+/- 0.14)**	0.7352	0.63 (+/- 0.20)	0.7711	0.74 (+/- 0.15)
DenseNet201	IoU	**0.7374**	**0.73 (+/- 0.11)**	0.6121	0.59 (+/- 0.16)	0.7226	0.68 (+/- 0.16)	0.5598	0.59 (+/- 0.20)	**0.5823**	**0.59 (+/- 0.16)**	0.5832	0.49 (+/- 0.19)	0.6346	0.61 (+/- 0.16)
PSPNet	Dice	0.8116	0.79 (+/- 0.10)	0.6754	0.65 (+/- 0.19)	0.7669	0.71 (+/- 0.20)	0.6522	0.67 (+/- 0.18)	0.6314	0.66 (+/- 0.13)	0.6919	0.68 (+/- 0.15)	0.7124	0.7 (+/- 0.15)
EfficientNetB5	IoU	0.6845	0.67 (+/- 0.11)	0.5144	0.51 (+/- 0.18)	0.624	0.58 (+/- 0.21)	0.4872	0.54 (+/- 0.19)	0.4645	0.51 (+/- 0.14)	0.5312	0.54 (+/- 0.16)	0.5602	0.57 (+/- 0.16)

U-Net (MSRL)		Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		Fold 6		Overall	
No. training images		1526		1232		1722		1729		1631		1575			
No. validation images		52		93		23		22		36		44		270	
Vanilla	Dice	0.758	0.69 (+/- 0.13)	0.8006	0.68 (+/- 0.24)	0.8951	0.88 (+/- 0.04)	0.9173	0.91 (+/- 0.04)	0.6647	0.58 (+/- 0.20)	0.6931	0.67 (+/- 0.16)	0.7743	0.7 (+/- 0.17)
	IoU	0.6128	0.55 (+/- 0.15)	0.6727	0.56 (+/- 0.25)	0.8102	0.79 (+/- 0.07)	0.8473	0.83 (+/- 0.06)	0.4999	0.44 (+/- 0.19)	0.5389	0.53 (+/- 0.18)	0.6423	0.58 (+/- 0.18)
VGG16	Dice	0.7634	0.70 (+/- 0.12)	0.8058	0.68 (+/- 0.25)	0.8949	0.88 (+/- 0.06)	0.921	0.91 (+/- 0.04)	0.6555	0.57 (+/- 0.20)	0.6843	0.67 (+/- 0.18)	0.7748	0.7 (+/- 0.17)
	IoU	0.6201	0.56 (+/- 0.14)	0.6791	0.56 (+/- 0.26)	0.8102	0.79 (+/- 0.09)	0.8536	0.84 (+/- 0.06)	0.4886	0.42 (+/- 0.19)	0.532	0.53 (+/- 0.18)	0.6438	0.58 (+/- 0.18)
DenseNet201	Dice	0.7183	0.58 (+/- 0.22)	**0.8198**	**0.72 (+/- 0.22)**	0.8857	0.88 (+/- 0.05)	0.9185	0.91 (+/- 0.04)	0.7117	0.58 (+/- 0.26)	0.7728	**0.73 (+/- 0.12)**	0.7918	0.71 (+/- 0.18)
	IoU	0.5652	0.44 (+/- 0.21)	**0.6987**	**0.60 (+/- 0.23)**	0.7953	0.79 (+/- 0.08)	0.8494	0.84 (+/- 0.07)	0.5544	0.46 (+/- 0.25)	0.6326	**0.59 (+/- 0.14)**	0.6635	0.58 (+/- 0.19)
EfficientNetB5	Dice	0.7778	0.67 (+/- 0.19)	0.8081	0.72 (+/- 0.19)	0.8978	**0.90 (+/- 0.05)**	**0.9222**	**0.91 (+/- 0.03)**	**0.7688**	**0.66 (+/- 0.16)**	**0.7738**	0.73 (+/- 0.14)	**0.8084**	**0.73 (+/- 0.15)**
	IoU	0.6389	0.53 (+/- 0.20)	0.6831	0.59 (+/- 0.21)	0.8148	**0.82 (+/- 0.07)**	**0.8556**	**0.84 (+/- 0.05)**	**0.6257**	**0.51 (+/- 0.17)**	**0.6341**	0.59 (+/- 0.16)	**0.6842**	**0.61 (+/- 0.17)**
Inception ResNetV2	Dice	0.7464	0.68 (+/- 0.17)	0.7998	0.68 (+/- 0.24)	0.8943	0.89 (+/- 0.04)	0.9213	0.91 (+/- 0.03)	0.6917	0.56 (+/- 0.23)	0.7344	0.70 (+/- 0.13)	0.7824	0.7 (+/- 0.17)
	IoU	0.5987	0.53 (+/- 0.18)	0.6707	0.57 (+/- 0.25)	0.8093	0.80 (+/- 0.06)	0.8543	0.84 (+/- 0.06)	0.5316	0.42 (+/- 0.21)	0.5831	0.55 (+/- 0.15)	0.6508	0.58 (+/- 0.18)
InceptionV3	Dice	**0.7871**	**0.73 (+/- 0.13)**	0.8175	0.71 (+/- 0.22)	**0.9011**	0.89 (+/- 0.05)	0.9196	0.91 (+/- 0.04)	0.7086	0.59 (+/- 0.22)	0.7627	0.72 (+/- 0.13)	0.8036	0.73 (+/- 0.16)
	IoU	**0.6514**	**0.59 (+/- 0.16)**	0.6958	0.59 (+/- 0.23)	**0.8203**	0.81 (+/- 0.07)	0.8513	0.84 (+/- 0.06)	0.5491	0.45 (+/- 0.20)	0.6186	0.58 (+/- 0.15)	0.6784	0.61 (+/- 0.17)
ResNet101	Dice	0.7207	0.58 (+/- 0.23)	0.8071	0.69 (+/- 0.24)	0.8911	0.88 (+/- 0.05)	0.9206	0.91 (+/- 0.04)	0.6347	0.55 (+/- 0.23)	0.7354	0.70 (+/- 0.16)	0.7722	0.69 (+/- 0.19)
	IoU	0.5718	0.45 (+/- 0.23)	0.6818	0.57 (+/- 0.25)	0.804	0.80 (+/- 0.08)	0.853	0.84 (+/- 0.06)	0.4671	0.42 (+/- 0.20)	0.5855	0.55 (+/- 0.18)	0.6407	0.57 (+/- 0.2)

LinkNet (MSRL)		Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		Fold 6		Overall	
No. training images		1526		1232		1722		1729		1631		1575			
No. validation images		52		93		23		22		36		44		270	
Vanilla	Dice	0.7755	**0.73 (+/- 0.10)**	0.803	0.68 (+/- 0.24)	0.8882	0.87 (+/- 0.05)	0.9146	0.90 (+/- 0.04)	0.721	0.63 (+/- 0.19)	0.6733	0.65 (+/- 0.17)	0.782	0.71 (+/- 0.16)
	IoU	0.6354	**0.59 (+/- 0.13)**	0.6756	0.56 (+/- 0.25)	0.7994	0.78 (+/- 0.08)	0.8428	0.83 (+/- 0.07)	0.5645	0.48 (+/- 0.19)	0.513	0.50 (+/- 0.17)	0.6507	0.59 (+/- 0.18)
VGG16	Dice	0.7599	0.71 (+/- 0.11)	0.7978	0.69 (+/- 0.23)	0.8935	0.88 (+/- 0.04)	0.9177	0.90 (+/- 0.05)	0.6321	0.53 (+/- 0.22)	0.7247	0.71 (+/- 0.12)	0.7744	0.71 (+/- 0.16)
	IoU	0.6152	0.57 (+/- 0.13)	0.6682	0.57 (+/- 0.24)	0.8078	0.79 (+/- 0.06)	0.8481	0.83 (+/- 0.08)	0.4632	0.39 (+/- 0.20)	0.5735	0.56 (+/- 0.14)	0.6418	0.58 (+/- 0.17)
DenseNet201	Dice	0.7753	0.69 (+/- 0.15)	**0.8187**	0.71 (+/- 0.23)	0.8962	**0.89 (+/- 0.03)**	0.9192	**0.91 (+/- 0.04)**	0.7382	0.65 (+/- 0.18)	0.7536	0.70 (+/- 0.16)	0.8038	0.73 (+/- 0.16)
	IoU	0.6348	0.55 (+/- 0.17)	**0.6971**	0.59 (+/- 0.24)	0.8121	**0.81 (+/- 0.05)**	0.8505	**0.84 (+/- 0.06)**	0.5864	0.50 (+/- 0.19)	0.608	0.56 (+/- 0.18)	0.6781	0.6 (+/- 0.18)
EfficientNetB5	Dice	**0.7881**	0.71 (+/- 0.15)	0.8135	**0.71 (+/- 0.21)**	0.8954	0.89 (+/- 0.04)	0.9186	0.91 (+/- 0.04)	0.7267	0.58 (+/- 0.22)	**0.7858**	**0.74 (+/- 0.13)**	**0.8081**	0.73 (+/- 0.16)
	IoU	**0.6522**	0.56 (+/- 0.17)	0.6901	**0.59 (+/- 0.22)**	0.811	0.81 (+/- 0.07)	0.8496	0.83 (+/- 0.06)	0.5725	0.44 (+/- 0.21)	**0.6498**	**0.60 (+/- 0.15)**	**0.6838**	0.6 (+/- 0.17)
Inception ResNetV2	Dice	0.78	0.72 (+/- 0.13)	0.8056	0.70 (+/- 0.22)	0.8874	0.88 (+/- 0.05)	**0.9204**	**0.91 (+/- 0.04)**	**0.7544**	**0.67 (+/- 0.18)**	0.7487	0.71 (+/- 0.15)	0.8009	**0.73 (+/- 0.16)**
	IoU	0.6415	0.58 (+/- 0.14)	0.6783	0.58 (+/- 0.23)	0.798	0.78 (+/- 0.08)	**0.8526**	**0.84 (+/- 0.06)**	**0.6058**	**0.53 (+/- 0.19)**	0.6068	0.57 (+/- 0.17)	0.6743	**0.61 (+/- 0.17)**
InceptionV3	Dice	0.7822	0.68 (+/- 0.17)	0.8041	0.70 (+/- 0.23)	**0.8974**	0.89 (+/- 0.04)	0.9194	0.91 (+/- 0.04)	0.7289	0.64 (+/- 0.18)	0.7455	0.70 (+/- 0.14)	0.7976	0.72 (+/- 0.17)
	IoU	0.6463	0.54 (+/- 0.19)	0.6771	0.57 (+/- 0.24)	**0.8141**	0.80 (+/- 0.07)	0.851	0.84 (+/- 0.06)	0.5742	0.49 (+/- 0.18)	0.5981	0.56 (+/- 0.15)	0.6704	0.59 (+/- 0.18)
ResNet101	Dice	0.745	0.62 (+/- 0.21)	0.7998	0.68 (+/- 0.25)	0.8895	0.88 (+/- 0.06)	0.9185	0.91 (+/- 0.04)	0.6557	0.50 (+/- 0.27)	0.7385	0.71 (+/- 0.17)	0.7774	0.69 (+/- 0.2)
	IoU	0.5962	0.48 (+/- 0.21)	0.671	0.56 (+/- 0.25)	0.8012	0.79 (+/- 0.09)	0.8494	0.84 (+/- 0.07)	0.4878	0.37 (+/- 0.24)	0.5921	0.57 (+/- 0.18)	0.6449	0.56 (+/- 0.2)

FPN (MSRL)		Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		Fold 6		Overall	
No. training images		1526		1232		1722		1729		1631		1575			
No. validation images		52		93		23		22		36		44		270	
VGG16	Dice	0.7686	**0.72 (+/- 0.12)**	0.8048	0.67 (+/- 0.25)	0.8946	0.89 (+/- 0.04)	0.9181	0.91 (+/- 0.04)	0.7296	0.62 (+/- 0.20)	0.7521	0.71 (+/- 0.13)	0.7961	0.72 (+/- 0.16)
	IoU	0.6262	**0.58 (+/- 0.14)**	0.6784	0.55 (+/- 0.26)	0.8096	0.80 (+/- 0.06)	0.8487	0.83 (+/- 0.07)	0.5754	0.48 (+/- 0.20)	0.6062	0.57 (+/- 0.15)	0.6679	0.59 (+/- 0.18)
DenseNet201	Dice	0.7855	0.72 (+/- 0.13)	0.8095	0.71 (+/- 0.20)	0.8958	0.89 (+/- 0.04)	0.9207	0.91 (+/- 0.04)	0.7226	0.62 (+/- 0.19)	0.7574	**0.72 (+/- 0.14)**	0.8012	0.73 (+/- 0.15)
	IoU	0.6487	0.58 (+/- 0.15)	0.6853	0.59 (+/- 0.21)	0.8115	0.80 (+/- 0.06)	0.8531	0.84 (+/- 0.06)	0.5668	0.47 (+/- 0.20)	0.6141	**0.57 (+/- 0.16)**	0.6753	0.61 (+/- 0.16)
EfficientNetB5	Dice	**0.8039**	0.72 (+/- 0.13)	**0.8227**	**0.73 (+/- 0.21)**	**0.9067**	**0.91 (+/- 0.04)**	0.9212	0.91 (+/- 0.03)	**0.7702**	**0.69 (+/- 0.15)**	0.7168	0.65 (+/- 0.17)	**0.81**	**0.74 (+/- 0.15)**
	IoU	**0.6741**	0.58 (+/- 0.16)	**0.7026**	**0.61 (+/- 0.22)**	**0.8296**	**0.83 (+/- 0.06)**	0.8539	0.84 (+/- 0.06)	**0.6266**	**0.54 (+/- 0.16)**	0.5673	0.51 (+/- 0.18)	**0.6881**	**0.62 (+/- 0.17)**
Inception ResNetV2	Dice	0.7726	0.67 (+/- 0.18)	0.8064	0.70 (+/- 0.23)	0.8921	0.88 (+/- 0.06)	**0.9237**	0.91 (+/- 0.03)	0.7106	0.59 (+/- 0.23)	0.7145	0.70 (+/- 0.16)	0.789	0.71 (+/- 0.18)
	IoU	0.632	0.53 (+/- 0.19)	0.6808	0.58 (+/- 0.24)	0.8055	0.79 (+/- 0.08)	**0.8582**	0.84 (+/- 0.06)	0.5518	0.45 (+/- 0.21)	0.5689	0.56 (+/- 0.17)	0.661	0.59 (+/- 0.19)
InceptionV3	Dice	0.7699	0.65 (+/- 0.19)	0.817	0.72 (+/- 0.20)	0.8988	0.90 (+/- 0.05)	0.922	**0.91 (+/- 0.03)**	0.6866	0.58 (+/- 0.23)	**0.7619**	0.71 (+/- 0.14)	0.7971	0.72 (+/- 0.17)
	IoU	0.6301	0.51 (+/- 0.20)	0.6945	0.59 (+/- 0.22)	0.8165	0.81 (+/- 0.08)	0.8553	**0.84 (+/- 0.05)**	0.5244	0.45 (+/- 0.22)	**0.6168**	0.57 (+/- 0.16)	0.6702	0.59 (+/- 0.18)
ResNet101	Dice	0.7046	0.60 (+/- 0.21)	0.7969	0.65 (+/- 0.28)	0.8946	0.89 (+/- 0.04)	0.9172	0.91 (+/- 0.04)	0.6231	0.50 (+/- 0.23)	0.6741	0.66 (+/- 0.18)	0.7541	0.66 (+/- 0.2)
	IoU	0.5523	0.46 (+/- 0.21)	0.6677	0.54 (+/- 0.27)	0.8098	0.81 (+/- 0.07)	0.8473	0.84 (+/- 0.06)	0.4557	0.37 (+/- 0.21)	0.5204	0.52 (+/- 0.19)	0.6199	0.55 (+/- 0.2)

PSP (MSRL)		Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		Fold 6		Overall	
No. training images		1526		1232		1722		1729		1631		1575			
No. validation images		52		93		23		22		36		44		270	
VGG16	Dice	0.758	**0.71 (+/- 0.12)**	**0.7964**	0.69 (+/- 0.23)	0.8876	0.87 (+/- 0.06)	**0.9127**	0.91 (+/- 0.04)	0.7012	0.61 (+/- 0.19)	0.6771	0.67 (+/- 0.14)	0.7741	**0.71 (+/- 0.16)**
	IoU	0.613	**0.56 (+/- 0.14)**	**0.6668**	0.57 (+/- 0.23)	0.7982	0.78 (+/- 0.09)	**0.8395**	0.83 (+/- 0.06)	0.5404	0.47 (+/- 0.19)	0.5162	0.52 (+/- 0.16)	0.6403	**0.59 (+/- 0.17)**
DenseNet201	Dice	**0.7647**	0.70 (+/- 0.13)	0.795	0.70 (+/- 0.21)	0.8843	0.88 (+/- 0.05)	0.9111	**0.91 (+/- 0.03)**	0.6669	0.57 (+/- 0.19)	**0.7124**	**0.68 (+/- 0.12)**	0.7757	0.71 (+/- 0.15)
	IoU	**0.6222**	0.55 (+/- 0.15)	0.6647	0.58 (+/- 0.22)	0.7927	0.78 (+/- 0.07)	0.8367	**0.83 (+/- 0.05)**	0.5006	0.42 (+/- 0.17)	**0.5562**	**0.53 (+/- 0.14)**	0.6419	0.58 (+/- 0.16)
EfficientNetB5	Dice	0.7631	0.67 (+/- 0.15)	0.7747	0.69 (+/- 0.22)	0.8845	0.88 (+/- 0.06)	0.91	0.90 (+/- 0.05)	0.6589	0.53 (+/- 0.21)	0.6933	0.65 (+/- 0.16)	0.7641	0.69 (+/- 0.17)
	IoU	0.6226	0.52 (+/- 0.17)	0.6366	0.56 (+/- 0.22)	0.7931	0.79 (+/- 0.08)	0.8349	0.82 (+/- 0.08)	0.4924	0.39 (+/- 0.19)	0.5325	0.49 (+/- 0.16)	0.6272	0.56 (+/- 0.17)
Inception ResNetV2	Dice	0.7398	0.62 (+/- 0.21)	0.7795	0.69 (+/- 0.22)	0.8837	0.88 (+/- 0.04)	0.914	0.91 (+/- 0.05)	**0.7467**	**0.64 (+/- 0.19)**	0.6984	0.67 (+/- 0.14)	0.7741	0.7 (+/- 0.17)
	IoU	0.5911	0.49 (+/- 0.22)	0.6434	0.56 (+/- 0.22)	0.7917	0.79 (+/- 0.06)	0.8416	0.83 (+/- 0.08)	**0.5963**	**0.49 (+/- 0.19)**	0.5417	0.52 (+/- 0.15)	0.6393	0.57 (+/- 0.18)
InceptionV3	Dice	0.7447	0.60 (+/- 0.22)	0.795	**0.70 (+/- 0.20)**	0.8819	0.87 (+/- 0.06)	0.9107	0.91 (+/- 0.04)	0.7063	0.62 (+/- 0.18)	0.7074	0.68 (+/- 0.13)	**0.776**	0.7 (+/- 0.16)
	IoU	0.5985	0.47 (+/- 0.22)	0.664	**0.58 (+/- 0.22)**	0.7891	0.78 (+/- 0.08)	0.8363	0.83 (+/- 0.06)	0.5462	0.47 (+/- 0.18)	0.5501	0.53 (+/- 0.15)	**0.6418**	0.57 (+/- 0.18)
ResNet101	Dice	0.7073	0.56 (+/- 0.23)	0.7818	0.66 (+/- 0.25)	**0.8895**	**0.89 (+/- 0.04)**	0.8969	0.89 (+/- 0.06)	0.5696	0.44 (+/- 0.25)	0.6339	0.63 (+/- 0.17)	0.7336	0.64 (+/- 0.2)
	IoU	0.5513	0.43 (+/- 0.22)	0.6471	0.54 (+/- 0.25)	**0.8012**	**0.80 (+/- 0.06)**	0.8133	0.80 (+/- 0.09)	0.3997	0.32 (+/- 0.21)	0.4694	0.48 (+/- 0.17)	0.5934	0.52 (+/- 0.2)

Top networks (MSRL)		Fold 1		Fold 2		Fold 3		Fold 4		Fold 5		Fold 6		Overall	
No. training images		1526		1232		1722		1729		1631		1575			
No. validation images		52		93		23		22		36		44		270	
U-Net	Dice	0.7778	0.67 (+/- 0.19)	0.8081	0.72 (+/- 0.19)	0.8978	0.90 (+/- 0.05)	0.9222	0.91 (+/- 0.03)	0.7688	0.66 (+/- 0.16)	0.7738	0.73 (+/- 0.14)	0.8084	0.73 (+/- 0.15)
EfficientNetB5	IoU	0.6389	0.53 (+/- 0.20)	0.6831	0.59 (+/- 0.21)	0.8148	0.82 (+/- 0.07)	0.8556	0.84 (+/- 0.05)	0.6257	0.51 (+/- 0.17)	0.6341	0.59 (+/- 0.16)	0.6842	0.61 (+/- 0.17)
U-Net	Dice	0.7871	**0.73 (+/- 0.13)**	0.8175	0.71 (+/- 0.22)	0.9011	0.89 (+/- 0.05)	0.9196	0.91 (+/- 0.04)	0.7086	0.59 (+/- 0.22)	0.7627	0.72 (+/- 0.13)	0.8036	0.73 (+/- 0.16)
IncpetionV3	IoU	0.6514	**0.59 (+/- 0.16)**	0.6958	0.59 (+/- 0.23)	0.8203	0.81 (+/- 0.07)	0.8513	0.84 (+/- 0.06)	0.5491	0.45 (+/- 0.20)	0.6186	0.58 (+/- 0.15)	0.6784	0.61 (+/- 0.17)
LinkNet	Dice	0.7881	0.71 (+/- 0.15)	0.8135	0.71 (+/- 0.21)	0.8954	0.89 (+/- 0.04)	0.9186	0.91 (+/- 0.04)	0.7267	0.58 (+/- 0.22)	**0.7858**	**0.74 (+/- 0.13)**	0.8081	0.73 (+/- 0.16)
EfficientNetB5	IoU	0.6522	0.56 (+/- 0.17)	0.6901	0.59 (+/- 0.22)	0.811	0.81 (+/- 0.07)	0.8496	0.83 (+/- 0.06)	0.5725	0.44 (+/- 0.21)	**0.6498**	**0.60 (+/- 0.15)**	0.6838	0.6 (+/- 0.17)
LinkNet	Dice	0.78	0.72 (+/- 0.13)	0.8056	0.70 (+/- 0.22)	0.8874	0.88 (+/- 0.05)	0.9204	0.91 (+/- 0.04)	0.7544	0.67 (+/- 0.18)	0.7487	0.71 (+/- 0.15)	0.8009	0.73 (+/- 0.16)
InceptionResNetV2	IoU	0.6415	0.58 (+/- 0.14)	0.6783	0.58 (+/- 0.23)	0.798	0.78 (+/- 0.08)	0.8526	0.84 (+/- 0.06)	0.6058	0.53 (+/- 0.19)	0.6068	0.57 (+/- 0.17)	0.6743	0.61 (+/- 0.17)
FPN	Dice	0.7855	0.72 (+/- 0.13)	0.8095	0.71 (+/- 0.20)	0.8958	0.89 (+/- 0.04)	0.9207	0.91 (+/- 0.04)	0.7226	0.62 (+/- 0.19)	0.7574	0.72 (+/- 0.14)	0.8012	0.73 (+/- 0.15)
DenseNet201	IoU	0.6487	0.58 (+/- 0.15)	0.6853	0.59 (+/- 0.21)	0.8115	0.80 (+/- 0.06)	0.8531	0.84 (+/- 0.06)	0.5668	0.47 (+/- 0.20)	0.6141	0.57 (+/- 0.16)	0.6753	0.61 (+/- 0.16)
FPN	Dice	**0.8039**	0.72 (+/- 0.13)	**0.8227**	**0.73 (+/- 0.21)**	**0.9067**	**0.91 (+/- 0.04)**	**0.9212**	**0.91 (+/- 0.03)**	**0.7702**	**0.69 (+/- 0.15)**	0.7168	0.65 (+/- 0.17)	**0.81**	**0.74 (+/- 0.15)**
EfficientNetB5	IoU	**0.6741**	0.58 (+/- 0.16)	**0.7026**	**0.61 (+/- 0.22)**	**0.8296**	**0.83 (+/- 0.06)**	**0.8539**	**0.84 (+/- 0.06)**	**0.6266**	**0.54 (+/- 0.16)**	0.5673	0.51 (+/- 0.18)	**0.6881**	**0.62 (+/- 0.17)**
PSPNet	Dice	0.758	0.71 (+/- 0.12)	0.7964	0.69 (+/- 0.23)	0.8876	0.87 (+/- 0.06)	0.9127	0.91 (+/- 0.04)	0.7012	0.61 (+/- 0.19)	0.6771	0.67 (+/- 0.14)	0.7741	0.71 (+/- 0.16)
VGG16	IoU	0.613	0.56 (+/- 0.14)	0.6668	0.57 (+/- 0.23)	0.7982	0.78 (+/- 0.09)	0.8395	0.83 (+/- 0.06)	0.5404	0.47 (+/- 0.19)	0.5162	0.52 (+/- 0.16)	0.6403	0.59 (+/- 0.17)
PSPNet	Dice	0.7647	0.70 (+/- 0.13)	0.795	0.70 (+/- 0.21)	0.8843	0.88 (+/- 0.05)	0.9111	0.91 (+/- 0.03)	0.6669	0.57 (+/- 0.19)	0.7124	0.68 (+/- 0.12)	0.7757	0.71 (+/- 0.15)
DenseNet201	IoU	0.6222	0.55 (+/- 0.15)	0.6647	0.58 (+/- 0.22)	0.7927	0.78 (+/- 0.07)	0.8367	0.83 (+/- 0.05)	0.5006	0.42 (+/- 0.17)	0.5562	0.53 (+/- 0.14)	0.6419	0.58 (+/- 0.16)

C Optimisation Tables

Loss functions		**Dice**		**Jaccard**		**BCE**		**Jaccard + BCE**		**Dice + BCE**	
No. training images		1141		1141		1141		1141		1141	
No. validation images		53		53		53		53		53	
FPN	Dice	0.8324	0.7825 (+/- 0.1627)	**0.835**	**0.7922 (+/- 0.1590)**	0.8072	0.7589 (+/- 0.1560)	0.8214	0.7736 (+/- 0.1622)	0.8259	0.7790 (+/- 0.1563)
EfficientNetB5	IoU	0.7143	0.6662 (+/- 0.1800)	**0.7179**	**0.6784 (+/- 0.1740)**	0.6779	0.6327 (+/- 0.1720)	0.6978	0.6542 (+/- 0.1807)	0.7044	0.6595 (+/- 0.1722)
U-Net	Dice	0.8178	0.7625 (+/- 0.1794)	0.8275	0.7724 (+/- 0.1717)	0.7933	0.7436 (+/- 0.1579)	**0.8289**	**0.7855 (+/- 0.1584)**	0.8138	0.7642 (+/- 0.1701)
EfficientNetB5	IoU	0.6933	0.6438 (+/- 0.1944)	0.7066	0.6552 (+/- 0.1901)	0.6582	0.6129 (+/- 0.1708)	**0.7086**	**0.6690 (+/- 0.1742)**	0.6874	0.6434 (+/- 0.1855)
LinkNet	Dice	0.8209	0.7624 (+/- 0.1669)	**0.8277**	0.7840 (+/- 0.1640)	0.8099	0.7684 (+/- 0.1551)	0.8244	**0.7877 (+/- 0.1545)**	0.8245	0.7764 (+/- 0.1606)
EfficientNetB5	IoU	0.6977	0.6406 (+/- 0.1860)	**0.7078**	0.6686 (+/- 0.1805)	0.681	0.6447 (+/- 0.1682)	0.7022	**0.6709 (+/- 0.1701)**	0.7022	0.6572 (+/- 0.1765)
U-Net	Dice	0.8233	0.7688 (+/- 0.1719)	0.8135	0.7569 (+/- 0.1954)	0.8093	0.7508 (+/- 0.1879)	**0.8279**	**0.7722 (+/- 0.1761)**	0.812	0.7563 (+/- 0.1794)
InceptionV3	IoU	0.701	0.6506 (+/- 0.1904)	0.6877	0.6413 (+/- 0.2094)	0.6823	0.6310 (+/- 0.2028)	**0.7083**	**0.6562 (+/- 0.1933)**	0.6853	0.6354 (+/- 0.1926)
LinkNet	Dice	0.8243	0.7597 (+/- 0.1948)	**0.8346**	**0.7783 (+/- 0.1814)**	0.8262	0.7688 (+/- 0.1756)	0.8249	0.7717 (+/- 0.1707)	0.8239	0.7669 (+/- 0.1831)
InceptionResNetV2	IoU	0.7024	0.6455 (+/- 0.2130)	**0.7176**	**0.6662 (+/- 0.1997)**	0.7055	0.6516 (+/- 0.1942)	0.7036	0.6543 (+/- 0.1903)	0.7026	0.6511 (+/- 0.1995)
FPN	Dice	0.8241	0.7669 (+/- 0.1854)	0.8186	**0.7768 (+/- 0.1623)**	**0.828**	0.7732 (+/- 0.1714)	0.802	0.7256 (+/- 0.2138)	0.797	0.7379 (+/- 0.1869)
DenseNet201	IoU	0.702	0.6516 (+/- 0.2003)	0.6937	**0.6589 (+/- 0.1834)**	**0.7077**	0.6563 (+/- 0.1901)	0.6708	0.6059 (+/- 0.2211)	0.6639	0.6145 (+/- 0.2051)

Batch sizes		**8**		**16**		**32**		**64**
No. training images		1141		1141		1141		1141
No. validation images		53		53		53		53
FPN	Dice	**0.835**	**0.7922 (+/- 0.1590)**	overflow		overflow		overflow
EfficientNetB5	IoU	**0.7179**	**0.6784 (+/- 0.1740)**	overflow		overflow		overflow
U-Net	Dice	**0.8289**	**0.7855 (+/- 0.1584)**	overflow		overflow		overflow
EfficientNetB5	IoU	**0.7086**	**0.6690 (+/- 0.1742)**	overflow		overflow		overflow
LinkNet	Dice	**0.8244**	**0.7877 (+/- 0.1545)**	overflow		overflow		overflow
EfficientNetB5	IoU	**0.7022**	**0.6709 (+/- 0.1701)**	overflow		overflow		overflow
U-Net	Dice	0.8279	**0.7722 (+/- 0.1761)**	**0.8308**	0.7655 (+/- 0.1779)	0.8209	0.7525 (+/- 0.1762)	overflow
InceptionV3	IoU	0.7083	**0.6562 (+/- 0.1933)**	**0.7116**	0.6471 (+/- 0.1907	0.6969	0.6303 (+/- 0.1958)	overflow
LinkNet	Dice	**0.8346**	**0.7783 (+/- 0.1814)**	0.8205	0.7377 (+/- 0.2023)	overflow		overflow
InceptionResNetV2	IoU	**0.7176**	**0.6662 (+/- 0.1997)**	0.6966	0.6180 (+/- 0.2130)	overflow		overflow
FPN	Dice	**0.8186**	**0.7768 (+/- 0.1623)**	0.817	0.7384 (+/- 0.1947)	overflow		overflow
DenseNet201	IoU	**0.6937**	**0.6589 (+/- 0.1834)**	0.6919	0.6167 (+/- 0.2064)	overflow		overflow

Learning rates		**3.00E-01**		**3.00E-02**		**3.00E-03**		**3.00E-04**		**3.00E-05**	
No. training images		1141		1141		1141		1141		1141	
No. validation images		53		53		53		53		53	
FPN	Dice	0.2796	0.2861 (+/- 0.1523)	0.8004	0.7272 (+/- 0.2044)	0.8346	0.7864 (+/- 0.1619)	**0.835**	**0.7922 (+/- 0.1590)**	0.7805	0.6943 (+/- 0.2126)
EfficientNetB5	IoU	0.1643	0.1766 (+/- 0.1095)	0.6684	0.6045 (+/- 0.2106)	**0.7179**	0.6715 (+/- 0.1794)	**0.7179**	**0.6784 (+/- 0.1740)**	0.6422	0.5671 (+/- 0.2201)
U-Net	Dice	0.174	0.1670 (+/- 0.0598)	0.7883	0.7188 (+/- 0.1906)	0.8281	0.7779 (+/- 0.1634)	**0.8289**	**0.7855 (+/- 0.1584)**	0.7359	0.6731 (+/- 0.1765)
EfficientNetB5	IoU	0.0955	0.0923 (+/- 0.0356)	0.6521	0.5901 (+/- 0.1988)	0.7081	0.6595 (+/- 0.1759)	**0.7086**	**0.6690 (+/- 0.1742)**	0.5834	0.5316 (+/- 0.1847)
LinkNet	Dice	0.1742	0.1672 (+/- 0.0599)	0.796	0.7271 (+/- 0.1903)	0.8055	0.7564 (+/- 0.1605)	**0.8244**	**0.7877 (+/- 0.1545)**	0.723	0.6524 (+/- 0.1893)
EfficientNetB5	IoU	0.0956	0.0924 (+/- 0.0357)	0.6627	0.6002 (+/- 0.1975)	0.6759	0.6307 (+/- 0.1773)	**0.7022**	**0.6709 (+/- 0.1701)**	0.5676	0.5108 (+/- 0.1916)
U-Net	Dice	0.1741	0.1671 (+/- 0.0599)	0.7595	0.6835 (+/- 0.1997)	0.7896	0.7226 (+/- 0.1872)	**0.8279**	**0.7722 (+/- 0.1761)**	0.7225	0.6485 (+/- 0.1997)
InceptionV3	IoU	0.0956	0.0923 (+/- 0.0356)	0.6142	0.5501 (+/- 0.2066)	0.6536	0.5942 (+/- 0.1981)	**0.7083**	**0.6562 (+/- 0.1933)**	0.5677	0.5094 (+/- 0.2024)
LinkNet	Dice	0.2796	0.2861 (+/- 0.1523)	0.7498	0.6437 (+/- 0.2526)	0.8092	0.7485 (+/- 0.1729)	**0.8346**	**0.7783 (+/- 0.1814)**	0.6805	0.5978 (+/- 0.2252)
InceptionResNetV2	IoU	0.1643	0.1766 (+/- 0.1095)	0.6018	0.5205 (+/- 0.2504)	0.6815	0.6236 (+/- 0.1889)	**0.7176**	**0.6662 (+/- 0.1997)**	0.518	0.4604 (+/- 0.2150)
FPN	Dice	0.2796	0.2861 (+/- 0.1523)	0.2796	0.2861 (+/- 0.1523)	0.7688	0.6712 (+/- 0.2396)	**0.8186**	**0.7768 (+/- 0.1623)**	0.7806	0.6968 (+/- 0.2139)
DenseNet201	IoU	0.1643	0.1766 (+/- 0.1095)	0.1643	0.1766 (+/- 0.1095)	0.6262	0.5479 (+/- 0.2405)	**0.6937**	**0.6589 (+/- 0.1834)**	0.642	0.5702 (+/- 0.2200)

Final networks		Loss function	Batch size	Initial learning rate	Validation set		Test set	
No. training images					1141		1141	
No. validation images					53		53	
FPN EfficientNetB5	Dice	Jaccard	8	3.00E-04	**0.835**	**0.7922 (+/- 0.1590)**	**0.8628**	**0.7954 (+/- 0.1423)**
	IoU				**0.7179**	**0.6784 (+/- 0.1740)**	**0.7592**	**0.6801 (+/- 0.1698)**
U-Net EfficientNetB5	Dice	Jaccard + BCE	8	3.00E-04	0.8289	0.7855 (+/- 0.1584)	0.8399	0.7762 (+/- 0.1685)
	IoU				0.7086	0.6690 (+/- 0.1742)	0.7249	0.6591 (+/- 0.1839)
LinkNet EfficientNetB5	Dice	Jaccard + BCE	8	3.00E-04	0.8244	0.7877 (+/- 0.1545)	0.8435	0.7736 (+/- 0.1709)
	IoU				0.7022	0.6709 (+/- 0.1701)	0.7299	0.6559 (+/- 0.1836)
U-Net InceptionV3	Dice	Jaccard + BCE	8	3.00E-04	0.8279	0.7722 (+/- 0.1761)	0.8568	0.7843 (+/- 0.1498)
	IoU				0.7083	0.6562 (+/- 0.1933)	0.7501	0.6665 (+/- 0.1754)
LinkNet InceptionResNetV2	Dice	Jaccard	8	3.00E-04	0.8346	0.7783 (+/- 0.1814)	0.8492	0.7834 (+/- 0.1467)
	IoU				0.7176	0.6662 (+/- 0.1997)	0.7387	0.665 (+/- 0.176)
FPN DenseNet201	Dice	Jaccard	8	3.00E-04	0.8186	0.7768 (+/- 0.1623)	0.8584	0.7889 (+/- 0.1494)
	IoU				0.6937	0.6589 (+/- 0.1834)	0.7527	0.6729 (+/- 0.1768)
U-Net ResNet101	Dice	Jaccard + BCE	8	3.00E-04	0.8019	0.74 (+/- 0.1965)	0.8429	0.7646 (+/- 0.1581)
	IoU				0.6717	0.6196 (+/- 0.2104)	0.7295	0.6424 (+/- 0.1857)