HuBMAP - Hacking the Human Vasculature

Christian Bernhard   Jonas Blenninger   Batuhan Gül   Thomas Ziereis

Abstract

Accurate segmentation of microvascular structures within 2D histology images holds crucial importance for understanding and locating positions in a human body. Inspired by a recently held Kaggle Challenge, we address the challenge of predicting polygonal segmentation masks for microvascular components, including capillaries, arterioles, and venules, within healthy human kidney tissue slides. We employ U-Net as a baseline segmentation model and subsequently evaluate distinct segmentation models by testing different combinations of hyperparameters and loss functions. We use the U-Net-inspired encoder-decoder architectures of DUCK-Net and TransResU-Net, with the latter also introducing the very popular transformer block. We conclude the experiments with SegFormer, a state-of-the-art transformer architecture and conclude that architectures that use transformers perform best in this segmentation task. Although reasonable results can be achieved with all mentioned models, the TransResU-Net and the SegFormer stand out with better predictions. Our code is available on GitHub[1] *or on Google Drive*[2].

1. Introduction

The advent of computer vision has revolutionized medical research by enabling accurate analysis of intricate tissue structures within histology images. One significant facet of this progress is the segmentation of microvascular structures, a task with profound implications for understanding tissue organization and function. The objective of this project is to devise a model capable of predicting polygonal segmentation masks for microvascular components—namely, capillaries, arterioles, and venules—within 2D histology images of healthy human kidney tissue slides stained with Periodic Acid-Schiff (PAS). Achieving automated segmentation for these structures is of paramount importance as it aligns with the overarching goals of the Human BioMolecular Atlas Program (HuBMAP) to establish a comprehensive platform for cellular mapping within the human body [2]. Central to this endeavor is the Vasculature Common Coordinate Framework (VCCF), which leverages capillary structures to ascertain cellular locations accurately. However, existing gaps in microvascular knowledge hinder the process made in creating the VCCF. Automated and precise segmentation of microvascular components presents a key solution to bridging these gaps, thereby contributing to the success of the HuBMAP initiative.

This project aims to evaluate and compare the efficacy of selected state-of-the-art computer vision models for medical segmentation, with the overarching goal to identify at least one that offers satisfying performance. This segmentation task also assumes the form of a Kaggle code competition, titled *HuBMAP - Hacking the Human Vasculature* [2]. Training and Test data used to train the model and evaluate the performance are provided in the context of the competition. We give a detailed analysis of the data in Section 2.

In Section 3, we provide an overview over the selected models. The popular U-Net [14] architecture serves as the baseline for the segmentation. Afterwards we dive deeper into the more complex architectures of DUCK-Net [8] and TransResU-Net [17], which extend the U-Net architecture in multiple ways. We also evaluate the SegFormer model [1] that makes use of the state-of-the-art Transformer architecture. We motivate the selection of the models and explain the the modifications made to the original implementations to adapt to our segmentation problem. Several metrics are used to evaluate and compare the performance of the models. Section 4 provides an overview over these metrics and afterwards in section 5, the experiment pipeline is described for each tested model. We discuss our results in section 6 and conclude on the strengths and weaknesses of the tested models in regard to the segmentation task at hand.

[1] https://github.com/JFLXB/CVDL_23_project_HuBMAP/releases/tag/submission-release

[2] https://drive.google.com/drive/folders/1wJuz8waqhhLFDo4hqDXrZZ1ZTIJXsof_?usp=sharing

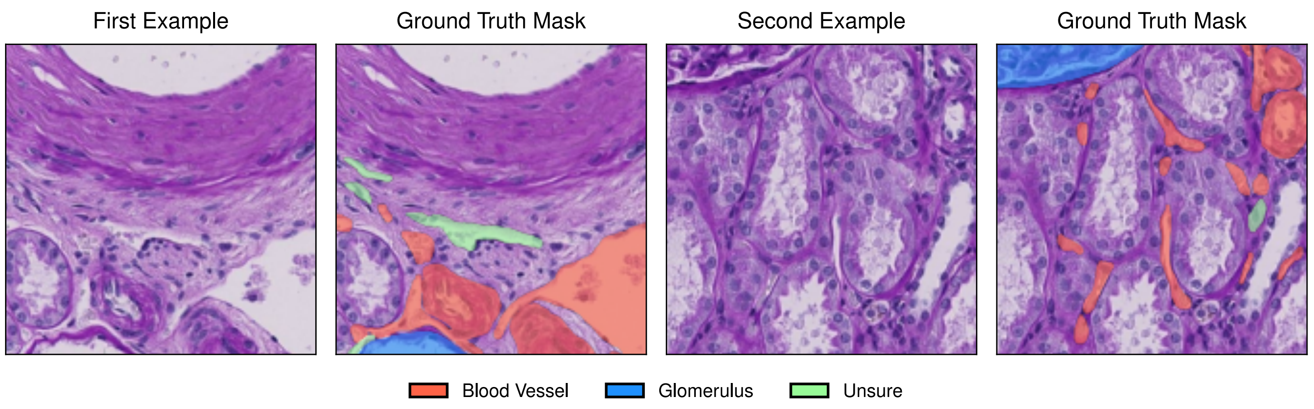


Figure 1. Example tiles of the training dataset with their corresponding ground truth segmentation mask as an overlay on the original image.

2. The Dataset

The data used for our competition are 2D periodic acid-Schiff (PAS)-stained histology images of healthy human kidney tissue sections. The tissue samples are from the renal cortex, renal medulla, and renal papilla. These images were extracted from 14 Whole Slide Images (WSIs) of human kidney tissue. The WSIs are high-resolution scanned slides of tissue samples that allow comprehensive examination at various magnification levels, providing a contextual view of tissue structure. From the large WSIs, small slices are taken to create multiple tiles [2].

The entire dataset contains a total of 7033 tiles of size 512x512 1, which are divided into three different categories. The first dataset contains tiles with annotations that have been thoroughly reviewed by experts, while the second dataset contains tiles with few annotations that have not been reviewed. The third dataset consists of unannotated tiles from 9 WSIs intended for self-learning methods to improve the segmentation process. Since we only evaluated supervised approaches, we are left with a total of 1633 annotated tiles, of which 422 were expert reviewed 2.

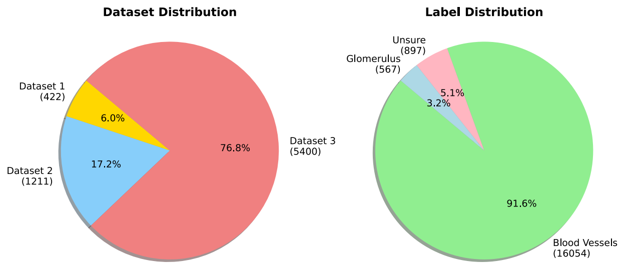


Figure 2. Dataset and label distribution

Each tile is labeled with three different labels: Blood vessels, glomerulus, and unsure, while blood vessels are the primary target of the study 2, 1. Unsure labels indicate those structures that could not be confidently classified as blood vessels by the annotators. This highlights the difficulty of recognizing blood vessels even for humans and shows how difficult the task of identification is. In addition, some images contain glomeruli representing capillary sphere structures in the kidney. The Kaggle competition, from which we received the task, states that: Glomeruli also contain blood vessels, but these are of no interest to the task and would be considered false positives during recognition. Therefore, we also segmented glomeruli, which converts this task into a multiclass instance segmentation.

3. The Models

In search of the most effective solution for our specific task, we have selected different models, all of which have a proven track record in the field of medical domain and instance segmentation. These models, which differ in architecture and complexity, were selected to provide a balanced range of approaches, from lightweight to robust and heavyweight methods.In the following, models that did not perform as expected are shortly described while more detailed explanations are provided for the models that delivered promising results. We chose the U-Net model, first introduced in 2015, as the baseline for comparison.

3.1. U-Net

The U-Net architecture is a seminal development in the field of biomedical image segmentation. Introduced in 2015 by researchers at the University of Freiburg [14], U-Net has become an essential tool for medical contexts, particularly when dealing with small datasets. The choice to utilize U-Net in our project was driven by several considerations. One reason is that U-Net promises solid results even with small data sets. This aligns with the general challenges that arise

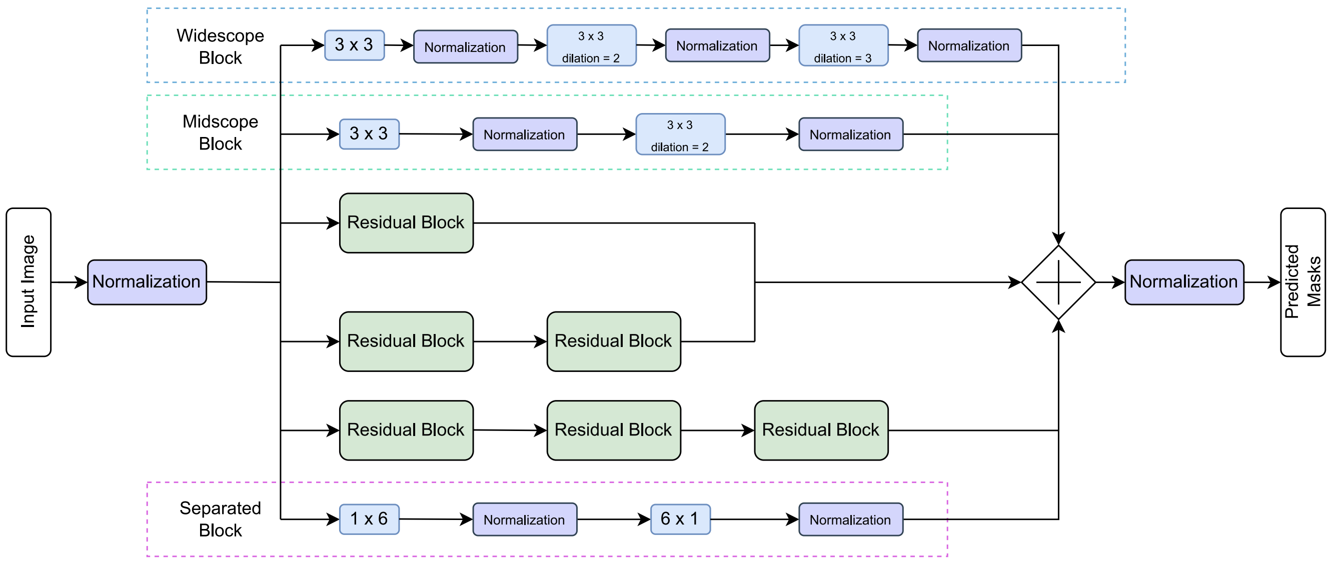


Figure 3. Block diagram of a DUCK-Block introduced in [8].

in medical image analysis due to the limited data available.

The architecture of U-Net exhibits a symmetrical design with an encoder-decoder structure, interconnected by skip connections. The name of the model is also derived from the resulting shape. The encoder gradually reduces the spatial dimension of the input through a series of convolutional layers followed by max-pooling layers, while increasing the number of feature channels. This process allows the network to compress the input into a more abstract, high-dimensional feature space, capturing complex patterns and structures.

Skip connections link each encoding layer with a corresponding decoding layer, ensuring that both low-level details and high-level features are combined. The decoder, in contrast to the encoder, gradually recovers the spatial dimensions through a series of transposed convolutions and concatenation with the corresponding encoder feature maps. This process allows the network to reconstruct the original spatial information, enriching it with semantic information learned in the encoding phase [14].

3.2. LinkNet

The LinkNet architecture was proposed by the authors of [4] in the year 2017. Until to this point, most architectures that provided sufficient performance for real-time applications such as YOLO [13] were designed to perform object detection. LinkNet aims to bring real-time capable performance to semantic segmentation tasks while staying accurate. The model uses the encoder-decoder strategy similar to U-Net [14] but instead of convolutional blocks, residual blocks [7] are used in the encoder-decoder-structure. Resnet18 is used as the encoder of the network [4], which results in a lighter network. By passing the input of each encoder layer to the output of its respective decoder, spatial information is recovered and delivers similar performance with a lower number of model parameters [4].

While the segmentation task of detecting vasculature in kidney tissue slides is not a real-time application, it is still intriguing to compare the LinkNet architecture to the U-Net baseline in terms of performance and efficiency. Higher efficiency might allow more intensive training loops with higher-resolution data.

3.3. DUCK-Net

One novel supervised convolutional architecture that is based on U-Net is the DUCK-Net model introduced in [8]. Its state-of-the-art performance in polyp segmentation in colonoscopy images makes it a compelling choice for our medical segmentation task.

While the DUCK-Net (Deep Understanding Convolutional Kernel Network) architecture builds up the encoder-decoder design of the U-Net architecture, it diverges in three pivotal ways [8]. Firstly, the model replaces the standard 3x3 convolutional blocks from the U-Net architecture [14] with a DUCK block at every step except the last one. A DUCK-Block, detailed in Fig. 3, contains different convolutional blocks operating in parallel and allowing the network to autonomously select the most optimal block for training.
One component inside the DUCK-Block is the residual block that was first shown in [12]. In addition to that, the authors of [8] introduce a Midscope and a Widescope block

that both dilate convolutions. This reduces the parameters that would otherwise be needed to improve understanding of high-level-features. As dilation also causes information loss, these blocks are designed to learn prominent features without the attention to detail. The separated block is the last newly introduced block and. This block also simulates larger kernels by combining an N x 1 kernel with a 1 x N kernel [8]. The result is a behaviour similar to an N x N kernel. The drawback of losing information on diagonality is compensated by the other blocks.As seen in Fig. 3, we used fixed 6 x 1 and 1 x 6 convolutions for the separated block.
Since the novel DUCK-Block does have negative side effects like losing fine details, the authors also used a secondary downscaling layer without any convolutional processing [8]. Two-dimensional 2 x 2 convolutions with strides of 2 are used for downscaling. Lastly, to save computational resources without sacrificing results, DUCK-Net combines outputs by adding them together instead of concatenating like in the U-Net architecture [8].

The code implementation of [8], provided by the authors on GitHub[3], makes use of the TensorFlow framework and Keras layers. In order to keep the code consistent within this project, the model architecture was rewritten as a PyTorch model. The implementation is flexible in regards to input channels and the number of prediction channels. Like in the models mentioned before, 3 input channels for the RGB images and 4 output channels are used. We need 4 output channels to obtain predictions for each of the 4 classes, namely blood vessels, glomeruli, unsure and background. The background class denotes all background pixels as a separate mask. Additionally, the TensorFlow as well as the PyTorch implementation allow adjusting the sizes of the convolutional layers by passing a *starting_filter* argument. This parameter controls the overall size of the model. More details on which filter sizes were used are provided in Sec. 5.3.

In the original introduction of the DUCK-Net in [8], the authors do not use any kind of pretraining for their model. This provides the opportunity to implement modified versions of the original DUCK-Net, allowing the use of pretrained weights. The pretrained DUCK-Net models use a ResNet18 [10] backbone and a ResNet34 [10] backbone with their respective default weights. The backbones replaces the pyramid feature extractor. To match outputs of the backbone layers, the filter sizes are fixed and can not be adjusted by a parameter anymore. For both ResNet18 and ResNet34 backbones, we only use the first 2 layers to initiate the feature extraction. The number of parameters using the ResNet18 backbone approximately matches the non-pretrained version when using 32 starting filters while using the ResNet34 backbone results in a larger number. In the following, we refer to the DUCK-Net model with pretrained ResNet18 weights as DUCK-Net18 and the model with pretrained ResNet34 weights as DUCK-Net34 respectively.

[3]https://github.com/RazvanDu/DUCK-Net

3.4. TransResU-Net

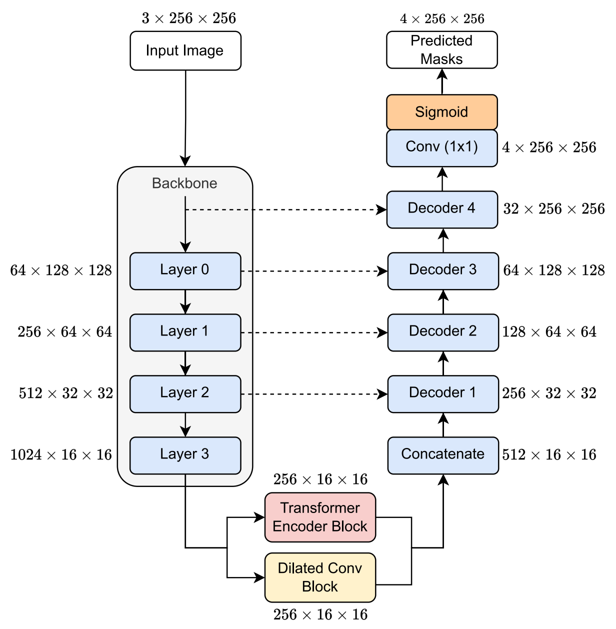
Tomar et al. [17] propose the TransResU-Net for colonoscopy polyp segmentation to identify cancerous regions in colonoscopy images. The model achieves state-of-the-art results for two different medical image segmentation data sets. TransResU-Net is an extension to the ResUNet [6] that itself builds on the U-Net [14] architecture by introducing residual units. In addition, the TransResU-Net uses dilation convolution blocks [23] and - following the generally increasing uptake of transformers in computer vision tasks - Transformer [5] blocks to develop the novel architecture. The TransResU-Net architecture is implemented as an encoder-decoder network. The encoder is defined by a backbone network, e.g., ResNet50 [10], followed by a bottleneck consisting of a transformer encoder block and a dilated convolution block. The output from the two bottleneck blocks is concatenated and passed to the decoder. The decoder is built from four decoder blocks. Each comprises an upsampling layer, a concatenation with the skip connection from the encoder block and two consecutive residual blocks. The output of the last decoder block is passed to a 1×1 convolution layer two to reduce the channel depth to the desired number of masks predicted by the model.

In the use case of [17], the model has to predict a single binary segmentation mask. However, in our use case, we have to predict masks for three (or four) different types of pixels: blood vessel, glomeruli and unsure pixels, and optionally background pixels, which are neither of the three preceding pixel classes. [2]. We, therefore, make a slight change to the final 1×1 convolution layer of the TransResU-Net by changing the number of output channels from 1 to 3 (4).

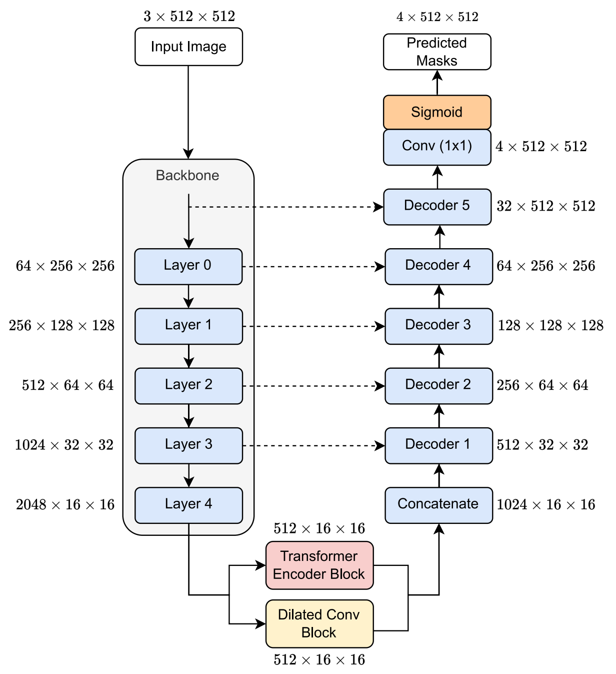
Another limitation of the TransResU-Net [17] is its restriction to images with dimension 256×256. To work with the images provided by the competition [2], we adapt the architecture by an additional encoder and decoder layer. These changes increased the number of channels after the last encoder layer of the original implementation from 1024 channels to 2048 channels. So the code provided by [17] on GitHub[4] had to undergo further changes to work with the increased image size. Our implementations of the TransResU-Net are based on the code provided by the authors of [17] available in their GitHub[4] repository.

Both the architecture of the TransResU-Net for 256×256 images and 4 output channels and the architecture for

[4]https://github.com/nikhilroxtomar/TransResUNet



(a) Block diagram of the TransResU-Net-4x256 architecture based on [17]. The output sizes of each block are written next to it, to make the difference between the TransResU-Net-4x256 and the TransResU-Net-4x512 clearer. The block diagram of the decoder block and the Transformer Encoder block is given in Fig. 5a and Fig. 5b, respectively.



(b) Block diagram of the TransResU-Net-4x512 architecture based on [17]. The output sizes of each block are written next to it, to make the difference between the TransResU-Net-4x512 and the TransResU-Net-4x256 clearer. The block diagram of the decoder block and the Transformer Encoder block is given in Fig. 5a and Fig. 5b, respectively.

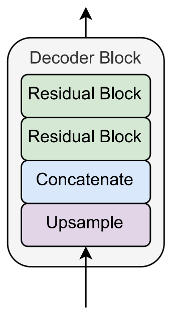
Figure 4. Block diagram of the TransResU-Net-4x256 in Fig. 4a and the TransResU-Net-4x512 in Fig. 4b.

images with dimensions 512 × 512 and 4 output channels are visualized in Fig. 4a and Fig. 4b, respectively.

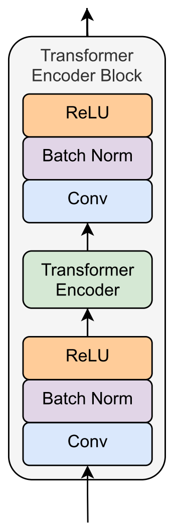
In the following sections, we refer to the TransResU-Net for 256 × 256 images and 4 output channels as TransResU-Net-4x256, and the TransResU-Net for 512 × 512 images and 4 output channels as TransResU-Net-4x512.

3.5. Fully Convolutional Transformer

The Fully Convolutional Transformer [18] is a model for medical image segmentation based on Transformers. According to the authors, the Fully Convolutional Transformer solves several drawbacks of traditional Convolutional Neural Networks (CNNs) and hybrid approaches combining CNN and transformers. According to the authors, one of the drawbacks is that architectures based on a CNN backbone have limited semantic context due to their disability to look beyond their receptive fields. Another issue is the computational complexity of Transformer-CNN hybrids. The proposed model's architecture is purely Transformer-based, consisting of novel, Fully Convolutional Transformer layers. Each layer has a Convolutional Attention module to learn long-range semantic context and a Wide-Focus module to create hierarchical local-to-global context.



(a) Block diagram of the decoder block of the TransResU-Net-4x256 and the TransResU-Net-4x512 architecture based on [17].



(b) Block diagram of the Transformer Encoder block of the TransResU-Net-4x256 and the TransResU-Net-4x512 architecture based on [17].

Figure 5. Block diagram of the decoder block (Fig. 5a) and Transformer Encoder Block (Fig. 5b) of the TransResU-Net-4x256 and TransResU-Net-4x512.

3.6. SegFormer

The SegFormer model, introduced in October 2021 by a research group at Nvidia, has emerged as a prominent method for image segmentation, achieving state-of-the-art results across various datasets, including medical data domains. The model consists of a novel positional-encoding-free hierarchical Transformer encoder, designed to generate both high-resolution coarse features and low-resolution fine features. These multi-level features are then fused through a lightweight MLP decoder to create the final segmentation mask. [20] A visual example of the architecture can be seen in Fig. 6.

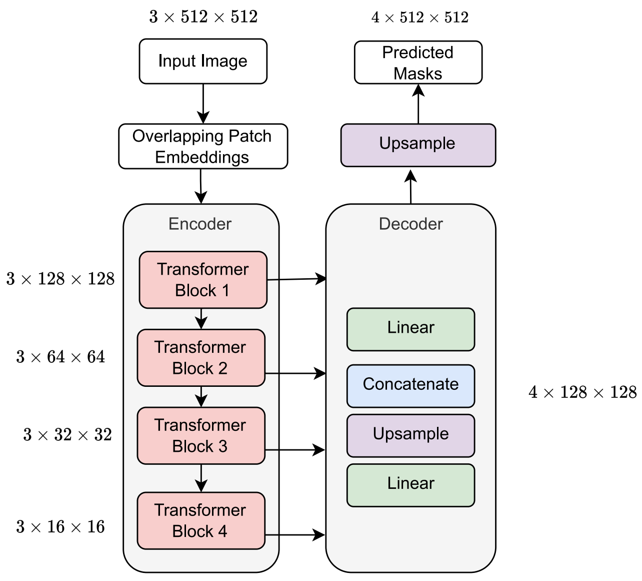


Figure 6. Simplified version of the Segformer architecture in [20] for 512x512 images and 4 classes to predict

The Segformer approach divides the input images into 4×4 patches. These patches are used as input to the encoder to obtain multilevel features at resolutions of $1/4$, $1/8$, $1/16$, and $1/32$ of the original image resolution. This is achieved through a method called Overlapped Patch Merging (OPM). [20]

The OPM process works by unifying a $2 \times 2 \times C_i$ feature path into a $1 \times 1 \times C_{i+1}$ feature map. The process can be described by the following transformation:

$$2 \times 2 \times C_i \rightarrow 1 \times 1 \times C_{i+1}$$

This transformation essentially reduces the spatial dimensions of the feature map by half, making it easy to iterate through this process to generate multilevel features. By sequentially applying this transformation, the different resolutions are obtained, corresponding to feature maps with increasingly lower spatial dimensions. [20]

Name	Params (M)	ImageNet-1k Top 1
MiT-b0	3.7	70.5
MiT-b1	14.0	78.7
MiT-b2	25.4	81.6
MiT-b3	45.2	83.1
MiT-b4	62.6	83.6
MiT-b5	82.0	83.8

Table 1. Different Backbones of the Segformer as taken from [1] and [20].

In typical Transformer encoders, the attention layer often becomes the computational bottleneck due to its quadratic time complexity $O(N^2)$, where N is the sequence length. The Segformer addresses this issue by employing a sequence reduction process that modifies the time complexity.

The sequence reduction in Segformer leads to a time complexity of $O\left(\frac{N^2}{R}\right)$, where R is the reduction ratio. In the Segformer architecture, this reduction ratio is set differently for the stages 1 to 4, with values of $R = 64, 16, 4, 1$ respectively. [20]

Another thing that makes the Segformer fast is the Decoder. Instead of using hand-crafted and computationally demanding components typically used by other alternatives. The segformer just uses a simple MLP decoder.

In the original paper, six different encoder backbones are introduced, as detailed in Tab. 1. These range from lightweight to high-performance models, tailored to various application needs. The MiT-b0 version, for instance, is designed as a lightweight model to enable fast inference. In contrast, the MiT-B5 variant aims to provide the best possible performance, suitable for more demanding tasks. [20]

A comparison presented in the paper ”Can segformer be a true competitor tou-net for medical image segmentation?” [15] further demonstrates the effectiveness of the Segformer approach. In a medical image segmentation task, even the smallest Segformer model, MiT-b0, outperforms a traditional U-Net. This results suggests, that similar or even better performance could be expected for out Task, especially when utilizing larger encoders.

4. Metrics

In the following section we introduce the different metrics used to evaluate the models' performance on the competition's dataset.

4.1. Precision

The precision is helpful to evaluate a model's ability to make correct positive classifications. It is calculated by dividing the number of correctly classified positive samples by the total number of samples that are classified as positive. [16, 19]

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

4.2. Recall

In order to evaluate whether a model is good at detecting positive samples, the recall can be used. It is calculated as the ration between all correctly classified positive samples and the total number of positive samples. This includes samples that are falsely classified as negative. [16, 19]

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

4.3. F_β-Measure

The F_β-Measure [16, 19] provides a single metric that takes false positives as well as false negatives into account. It is useful if a trade-off between precision and recall is desired, without favoring one over the other. Mathematically it is calculated as the harmonic mean of precision and recall. The general equation of the F_β-Measure is given by

$$F_\beta = \frac{\left(\beta^2 + 1\right) \cdot \text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}. \tag{3}$$

The F_1- and F_2-Measure are special cases of the F_β-Measure. The F_1 measure is also often referred to as the Dice coefficient. [16] The F_1-Measure is given by

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{4}$$

Accordingly, the F_2-Measure is given by

$$F_2 = \frac{(2^2 + 1) \cdot \text{Precision} \cdot \text{Recall}}{2^2 \cdot \text{Precision} + \text{Recall}}. \tag{5}$$

4.4. Jaccard Index (IoU)

The Jaccard Index, also reffered to as Intersection over Union (IoU), is one of the most used evaluation metrics for image segmentation tasks. It is used to calculate how similar two sets are to each other. [24] We use the Jaccard Index as measure on how close the predicted segmentation mask is to the ground truth segmentation mask of blood vessels. The general formula of the Jaccard Index according to [24] is given in Eq. (6).

$$J(P, T) = \frac{P \cap T}{P \cup T}, \tag{6}$$

where P is the predicted segmentation mask and T is the ground truth segmentation mask.

4.5. Accuracy

The accuracy (Acc) is another common performance metric in image segmentation tasks. It is defined as the the number of correctly predicted pixels over the total number of pixels of a segmentation mask. [16] Following [16] we can calculate accuracy by

$$\text{Acc} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{7}$$

One should however be careful with the results of the accuracy metric, since according to [11] a high accuracy does not necessarily imply superior segmentation performance in cases where there is a high imbalance of the classes in the dataset.

5. Experiments

This chapter delineates the overall experimental setup and methodology that were employed. Furthermore we describe the individual training process and results of every model. We then compare the models performance against each other.

The dataset was carefully partitioned into three distinct sets: training, validation, and test. This division plays a vital role in assessing the model's ability to generalize, not just fit the available data. We roughly did a 75:15:10 split of the 1633 tiles resulting in the following sets. We ensured that the validation set contains only expert-labeled data with no unsure labeled blood vessels. This is essential to ensure a fair and reasonable comparison during the tuning phase.

- Training Set (1226 samples): Utilized to train the model, allowing it to learn and recognize underlying patterns and features.
- Validation Set (244 samples): This separate portion of the data was used to fine-tune hyperparameters and to give an unbiased evaluation of a model fit during training. It assists in preventing overfitting and gives insight into the model's performance on unseen data.
- Test Set (164 samples): Reserved for our final evaluation as an unbiased assessment of the finalized model fit, simulating its potential real-world performance.

Furthermore we used certain data augmentation techniques across all approaches. Data augmentation is a pivotal technique, particularly in the context of medical imaging, where acquiring large quantities of labeled data can be challenging. It not only enhances the diversity of the training set but also helps in preventing overfitting. For our particular use case, we made the following augmentations. Those augmentations made sense, since the tiles were crops from a full WSI. Other types of augmentations that could change the structure of the instances we are looking for (e.g., shear) would not be suitable for our task.

- Random Horizontal Flip: This introduces variations in orientation, assisting the model in recognizing features irrespective of their alignment.
- Random Vertical Flip: Similar to horizontal flipping, vertical flipping also extends the orientation variations, further enhancing the model's robustness.

In our initial experimentation, training the models to only segment for the three given pixel labels was unsuccessful. Therefore, we extended our code to include an additional segmentation mask containing all background pixels, i.e., all pixels that are neither associated with a blood vessel, a glomerulus, nor a pixel of the unsure mask. We evaluate the models' performance during experimentation using the performance metrics introduced in Sec. 4 on the validation set. However, we will not evaluate the model on the F_1-Measure (Dice coefficient) following the reasoning of [16], we do not gain additional information when selecting both measures as they both measure the same aspects and the same ranking. Therefore our evaluation metrics reduced to the following list: Precision, Recall, F_2-Measure, IoU (Jaccard Index) and Accuracy (Acc). The same metrics are used in the final comparison of the models evaluated on the test set.

5.1. U-Net

Despite its inherent ability to handle small datasets, training the U-Net model from scratch didn't result in promising results, as our dataset is exceptionally small and unique. To mitigate this, we incorporated pre-trained weights from the ResNet architecture (ResNet-18, ResNet-34) into the model's encoder [26]. The use of pre-trained weights leverages patterns and features learned from larger and more diverse datasets, and improves the model's ability to generalize and learn the unique features of our specific task. Without them, the model wasn't able to make any useful predictions, which shows again the hard task of detecting bloodvessels.

5.1.1 Example Experiment

We tested different combinations of hyperparameters to find the best setting for our task, while considering both time and energy consumption. Tab. 2 gives a little insight into how the training was approached. In this run, the images were downsampled to 256x256 and different learning rates, ResNet architectures, and loss functions were tested. The first value represents the result for the Cross Entropy Loss (CEL) the second one for Focal Loss which bouth will be introduced in the upcoming section. Experiments such as these helped to understand how these parameters affect our model's ability to detect blood vessels. The training was done by trying out different combinations of the hyperparameters over 50 epochs and log multiple metrics and the model checkpoints so we could compare them afterwards. The following table provides an overview of the different runs and their results. The metrics are calculated by taking the average from epoch 30 to 40 as all models have converged at that time.

5.1.2 Compare Losses

For the U-Net two prominent loss functions were employed: Cross Entropy Loss and Focal Loss [22]. The former, CEL, is a standard choice for classification problems and is often used for multi-class segmentation.

Focal Loss, on the other hand, was specifically chosen for its potential benefits in handling class imbalance, which is a common challenge in our task [22]. As the majority of the pixels in the images are background, and only a few represent blood vessels or other structures, a significant class imbalance is present. Focal Loss addresses this by introducing two hyperparameters: α and γ, where α controls the weight given to different classes, and γ modulates the contribution of hard-to-classify examples. The loss is computed as:

$$\text{Focal Loss} = \alpha_t \cdot (1 - p_t)^\gamma \cdot \text{CEL}$$

As the experiment tables shows, the Cross Entropy Loss consistently performed better than the Focal Loss for the configurations tested. However, this observation does not definitively favor CEL over Focal Loss. The effectiveness of Focal Loss depends on the precise tuning of the α and γ values. Given our limited opportunity to explore various combinations, it remains possible that a different set of hyperparameters could lead to Focal Loss outperforming CEL. The potential benefits of Focal Loss in handling class imbalance suggest that further exploration could be warranted, and its performance might be enhanced with more refined tuning.

5.1.3 Compare Learning Rates

Furthermore, our experiments revealed a significant impact of different learning rates on the model's performance. All lower values such as 0.0001, 0.0005, and 0.001 showed pretty solid and comparable Intersection over Union (IoU) metrics for the validation set. The fact these learning rates performed similar suggests that picking the highest among these leads to faster convergence during training without sacrificing to much accuracy.

Learning Rate	Backbone	Precision	Recall	F2	IoU	Acc
lr=0.0001	**ResNet-18**	0.6303 / 0.274	0.485 / 0.6411	0.4851 / 0.4531	0.365 / 0.2306	0.9678 / 0.9205
lr=0.0001	**ResNet-34**	0.6647 / 0.4212	0.5050 / 0.7129	0.5083 / 0.5947	0.3866 / 0.3505	**0.9714** / 0.9526
lr=0.0005	**ResNet-18**	**0.7179** / 0.3942	0.4608 / 0.6766	0.4781 / 0.5441	0.3788 / 0.312	0.9734 / 0.9427
lr=0.0005	**ResNet-34**	**0.6452** / 0.4391	**0.5433** / 0.5913	**0.5419** / 0.5091	**0.4110** / 0.3139	**0.9716** / 0.9444
lr=0.001	**ResNet-18**	0.7095 / 0.4609	0.4053 / 0.4775	0.4216 / 0.3686	0.3322 / 0.2139	0.9710 / 0.9232
lr=0.001	**ResNet-34**	**0.7043** / 0.8163	0.4518 / 0.006	0.4666 / 0.0061	0.366 / 0.0038	**0.9723** / 0.9577
lr=0.005	**ResNet-18**	1.0000 / 1.0000	0.0000 / 0.0000	0.0000 / 0.0000	0.0000 / 0.0000	0.9602 / 0.9602
lr=0.005	**ResNet-34**	0.5475 / 1.0000	0.2694 / 0.0000	0.2043 / 0.0000	0.1307 / 0.0000	0.9085 / 0.9602
lr=0.01	**ResNet-18**	1.0000 / 1.0000	0.0000 / 0.0000	0.0000 / 0.0000	0.0000 / 0.0000	0.9602 / 0.9602
lr=0.01	**ResNet-34**	1.0000 / 1.0000	0.0000 / 0.0000	0.0000 / 0.0000	0.0000 / 0.0000	0.9602 / 0.9602

Table 2. U-Net Hyperparameter finetuning example - averaged metrics from epoch 30 to 40 (1st value=Cross Entropy Loss, 2nd value=Focal Loss)

In contrast, larger values such as 0.005 and 0.01 showed a substantial drop in performance. The following plot Fig. 7 illustrates the evolution of loss and IoU over the epochs for learning rates 0.0005 and 0.005 (the plot was from an older run which used a slighty different IoU than the table before. Therefore the resulting IoUs are abit higher than the IoUs in the table before. However the relations and the interpreted outcome is the same). If the learning rate is to high, that leads to rapid convergence into a local optimum, thus preventing further improvements. This phenomenon results from taking overly large steps during the optimization process, causing the model to overshoot the optimal solution. As seen in Tab. 2, the metrics for the higher learning rates remain consistent across epochs 30 to 40, reflecting that the model stopped learning. The training of the lower learning rate approach looks a lot more reasonable. Additionally, it is noticeable from the graph that the superior model, characterized by the lower learning rate, shows convergence at about 10 epochs on the validation set. Beyond this point, although there is a continuous improvement in performance on the training set, it seems to be followed by a divergence between training and validation metrics, indicating the beginning of overfitting.

As already mentioned, in order to gain the best model, a lot more combinations of different hyperparameter configurations need to be tested and the previous table only showcased some training examples. Additionally techniques like learning rate scheduling could enable even more refined tuning. However for our testing, the "**U-Net with ResNet-34 architecture, lr=0.0005 and Cross Entropy Loss** " configuration on 256x256 downsampled images showed the best result. Therefore this model will build the baseline with which the more sophisticated models will be compared.

5.2. LinkNet

With the Implementation of the LinkNet after [4], the goal was to get at least similar performance as the U-Net baseline while taking up less computational resources. The code was implemented by the authors of [4] and provided on GitHub[5]. Out of the box, the model works with 256×256 images with varying input channels and output classifiers. Our semantic segmentation task needed 3 input channels for RGB images and 4 output channels for the 4 classes.
Unfortunately, the LinkNet implementation had issues predicting the segmentation masks of blood vessels and did not deliver satisfying performance. Considering the underwhelming results and the fact that LinkNet was chosen to be a more efficient alternative to the baseline, we do not go into more detail regarding this architecture and move on to more promising ones.

5.3. DUCK-Net

The encoder-decoder structure of the DUCK-Net was first put to test by using 128×128 input images. The goal of this first run of tests is to determine which of the available loss functions work best for DUCK-Net as well as the pretrained DUCK-Net with a ResNet18 backbone. The loss functions compared are the Dice Loss, the BCE Dice Loss, the Cross Entropy Loss and the Focal Loss [22]. For the Focal Loss, blood vessels were weighted with 0.4 and background pixels with 0.3. Both glomeruli and unsure masks were weighted with 0.15, as our primary goal is to predict the blood vessels correctly. More details on the FocalLoss are provided in Sec. 5.1. Due to time constraints the selection of the weights was not optimized, but empirically selected by comparing a few configurations. In addition to that we included another weighted

[5]https://github.com/e-lab/pytorch-linknet

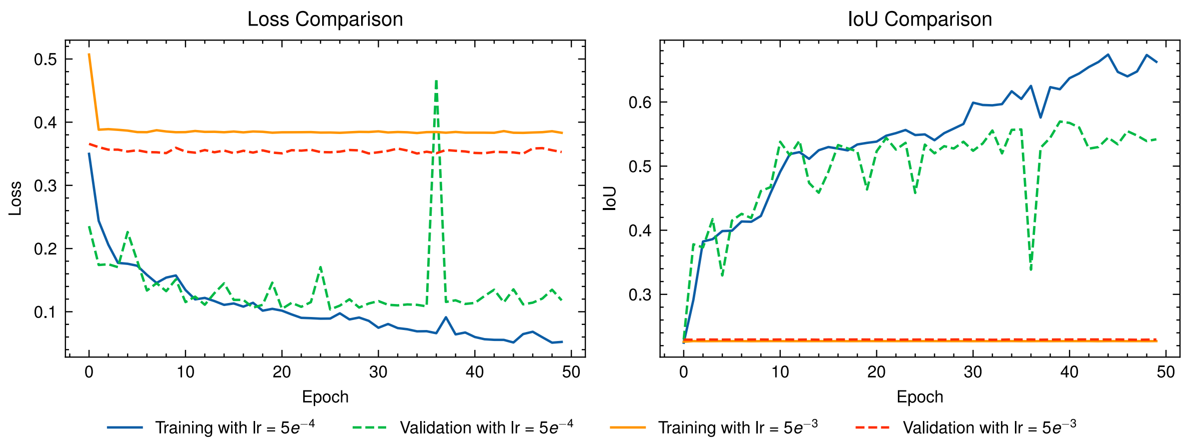


Figure 7. Comparison of the per epoch loss and IoU of the U-Net using a learning rate of $5e^{-4}$ and $5e^{-3}$.

loss, the Channel Weighted Dice BCE Loss. This loss was introduced for the experiment detailed in Sec. 5.4 and also compared for the DUCK-Net. In this case the weight values are derived from the occurence of class pixels in the training set. Subsection Sec. 5.4 describes the exact weights in more detail.

In the following, DUCK-Net refers to the original implementation without a backbone and without pretrained weights, while DUCK-Net18 and DUCK-Net34 refer to the pretrained models with ResNet18 [12] and ResNet34 [12] backbones respectively, as described in Sec. 3.3. All DUCK-Net experiments were run on an NVIDIA RTX 3080 GPU and used the Adam Optimizer. The experiments conducted with image sizes of 128×128 used a batch size of 8, while later experiments with 256×256 images used a batch size of 4 to accommodate the limited memory of the GPU. We compare results after training with 20 epochs.

In order to find a suitable learning rate for following test runs, DUCK-Net was tested with Dice-Loss for 20 epochs for learning rates of $1e-3$ and $1e-4$. It is clearly visible in Fig 8 that the lower learning rate does not converge fast enough for our training setup with 20 epochs. While the higher learning rate shows a few jumps in the validation loss, it is still much more suitable and delivers better results. For this reason, all following runs were conducted with a learning rate of 0.001 to compare other parameters and variables.

It is noted that for our specific task and dataset, the accuracy metric is not useful since there is a large bias in the class distribution. There is no need to discuss the accuracy in the following DUCK-Net experiments. Tab. 3 shows both DUCK-Net and the pretrained DUCK-Net18 with various losses. For the DUCK-Net, the *starting_filter* parameter was set to 32, as this parameter matched the size of the pretrained DUCK-Net well and resulted in

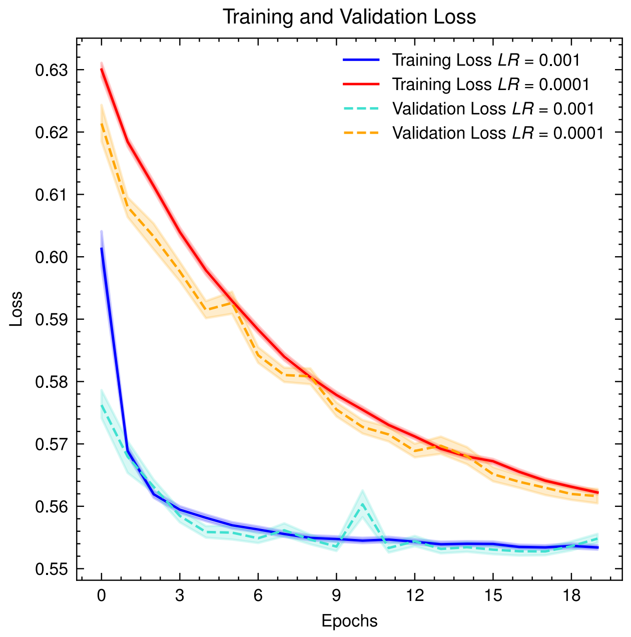


Figure 8. The training and validation losses during training of the DUCK-Net using learning rates of $LR = 0.001$ and $LR - 0.0001$

similar training times of around 1 minute per epoch. Looking at DUCK-Net scores, the scores achieved with the DiceBCELoss come to attention. From these scores it can be assumed that this model did not learn to predict blood vessels. The DiceLoss on the other hand was able to perform segmentation with a rather high precision of 0.7233. However, the low recall and F2 scores indicate that the model misses most of the blood vessel masks, therefore also achieving low IoU scores.

Using the DiceBCELoss on DUCK-Net18 showed the same outcome as plain DUCK-Net: The model was not

Model	Loss	Precision	Recall	F2	IoU	Acc
DUCK-Net	DiceLoss	**0.7233**	0.2561	0.2802	0.2250	0.9682
DUCK-Net	DiceBCELoss	1.0000	0.0000	0.0000	0.0000	0.9602
DUCK-Net	FocalLoss	0.5372	0.3964	0.3804	0.2717	0.9544
DUCK-Net18	CrossEntropyLoss	**0.6936**	0.3490	0.3713	0.2923	0.9694
DUCK-Net18	FocalLoss	0.5809	**0.5091**	**0.4999**	**0.3618**	0.9663
DUCK-Net18	DiceLoss	0.6043	0.4381	0.4438	0.3273	0.9682
DUCK-Net18	DiceBCELoss	1.0000	0.0000	0.0000	0.0000	0.9602
DUCK-Net18	ChannelWeightedDiceBCELoss	0.6915	0.4131	0.4298	0.3358	0.9706

Table 3. Results of the DUCK-Net and DUCK-Net18 models as described in Sec. 3.3 for different losses.

able to learn to predict blood vessel masks. Other losses showed more promising results, with the ChannelWeightDiceBCELoss and the CrossEntropyLoss having decent precision values of 0.692 and 0.694 respectively. Using the channel-weighted loss did outperform CrossEntropyLoss in other metrics though, such as recall and IoU. The best recall score was achieved by using the FocalLoss at 0.509, meaning that half of all blood vessel pixels in the validation set were found by the model. This loss also achieved the highest IoU score. An IoU of 0.362 is still rather low and leaves room for improvement.
The first results show more potential for DUCK-Net18 due to using pretrained weights and ResNet18 layers as a feature extractor. For this reason we now focus on the pretrained DUCK-Net versions. Even though the original plan was to also test smaller model sizes by adjusting the *starting_filter* parameter, we deviated from this idea since performance is not satisfying while the hardware does not bottleneck the progress at the moment.
In the next run of experiments, we wanted to investigate whether higher-resolution input images improve training results. 256×256 images were used and fed into the networks. Using a lower batch-size as explained before resulted in longer training times of approximately 2 minutes per epoch.

Tab. 4 shows the scores achieved with different input sizes. Using larger input images shows to improve overall results. Comparing both models using the ChannelWeightedDiceLoss, the higher-resolution model achieves similar precision, but improves recall and therefore also F2 scores. It also achieves a better IoU of 0.3916. The DiceLoss model shows very similar results to the CHannelWeightedDiceLoss. This implicates that for the used input image size, DiceLoss is already able to deal with the large discrepencies in the number of mask pixels per class and does not need any weighting. Comparing the FocalLoss models shows some interesting differences. The DUCK-Net18 that was trained with 128×128 images has a higher precision than the model trained with larger images. As a side effect, DUCK-Net18 with 256×256 images improves the recall score and is able to detect 65.3% of all blood vessel pixels in the validation set. As a consequence, it also improves the F2 and IoU scores. Although this configuration is not the best in terms of precision, the high recall value leads to this model being the preferred one so far.
Finally, we also conducted tests on the DUCK-Net34 architecture using ResNet34 as a backbone. In theory, using a larger pretrained model as a backbone leads to a larger overall network and might boost performance. The tests with DUCK-Net34 are conducted using 128×128 images as the larger backbone causes increased training time already.

In contrast to our expectations, the results for DUCK-Net34 in Tab. 5 do not show any advantages compared to the similarly efficient DUCK-Net18 with 256×256 images. While precision is quite high for both DiceLoss and ChannelWeightedDiceloss, the recall score is significantly lower than the score we achieved with DUCK-Net18. Since IoU scores also do not show any improvements no further experiments were made using DUCK-Net34.
To conclude the experiments for the DUCK-Net models, we have chosen DUCK-Net18 with the FocalLoss and input image sizes of 256×256 as the best performing model for the vascular segmentation task.

5.4. TransResU-Net

Using a backbone in the TransResU-Net [17] architecture allows us to compare the effect on the performance metrics of utilizing different (pretrained) models as the encoder in predicting blood vessels in microvascular structure. We compare the models' performance on the untrained and pretrained backbones. The GitHub[6] repository of [17] implements code to retrieve different (pretrained) backbones, namely the ResNet50 [10], the ResNet101 [10], the ResNet-152 [10], the ResNeXt-50 32x4d [21], the ResNeXt-101 32x8d, the Wide ResNet-50-2 [25] and the Wide ResNet-

[6]https://github.com/nikhilroxtomar/TransResUNet

Image Size	Loss	Precision	Recall	F2	IoU	Acc
128×128	FocalLoss	0.5809	**0.5091**	0.4999	0.3618	0.9663
128×128	ChannelWeightedDiceBCELoss	0.6915	0.4131	0.4298	0.3358	0.9706
256×256	DiceLoss	**0.6746**	0.5009	0.5092	0.3936	0.9716
256×256	ChannelWeightedDiceBCELoss	**0.6829**	0.4963	0.5053	0.3916	0.9712
256×256	CrossEntropy	0.6346	0.4251	0.4350	0.3307	0.9690
256×256	FocalLoss	0.5093	**0.6525**	**0.5844**	**0.3931**	0.9548

Table 4. Results of DUCK-Net18 from Sec. 3.3 for different image sizes and losses.

Model (Image Size)	Loss	Precision	Recall	F2	IoU	Acc
DUCK-Net18 (256×256)	ChannelWeightedDiceBCELoss	0.6915	0.4131	0.4298	0.3358	0.9706
DUCK-Net34 (128×128)	DiceLoss	0.7056	0.3470	0.3691	0.2904	0.9693
DUCK-Net34 (128×128)	ChannelWeightedDiceBCELoss	0.6990	0.3658	0.3866	0.3045	0.9701
DUCK-Net34 (128×128)	FocalLoss	0.6285	0.4689	0.4743	0.3569	0.9695

Table 5. Results of DUCK-Net34 detailed in Sec. 3.3 for different losses

101-2 [25].

We split the experimentation of the TransResU-Net-4x256 and the TransResU-Net-4x512 into a three-step process.

1. Train TransResU-Net-4x256 and TransResU-Net-4x512 using pretrained weights and from scratch for all available backbones for a few epochs.

2. Select the best combinations for both the TransResU-Net-4x256 and the TransResU-Net-4x512 and continue training for the selected combinations for a further 200 epochs.

3. Choose the single best combination for TransResU-Net-4x256 and TransResU-Net-4x512, respectively, and try to increase the performance.

All experimentation is conducted on an NVIDIA RTX 3090 GPU. First, we want to identify the most promising combination of a backbone with either pretrained weights or training the backbone from scratch. We run all possible combinations for 20 warmup epochs and evaluate the resulting models on the evaluation metrics. Following the experimentation of [17], we set the learning rate to $1e^{-4}$, the batch size of 16 for the TransResU-Net-4x256 and a batch size of 8 for the TransResU-Net-4x512İn both cases, we use the DiceBCELoss[7], and the Adam optimizer.

In our early stage experimentation, we also tried a larger learning rate of $1e^{-3}$ and a smaller learning rate of $1e^{-5}$. However, changing the learning rate to $1e^{-3}$, resulted in a high variance of the models' validation loss. Setting the learning rate to $1e^{-5}$, all models converged too early, giving lower evaluation metrics than using a learning rate of $1e^{-4}$. The batch sizes are chosen according to the memory usage for the different image sizes. The model needs more memory for 512×512 sized images than for 256×256 sized images, so we need a smaller batch size accordingly.

We diverge from their setup for the warmup training and do not use a learning rate scheduler or early stopping to give all models an equal chance in this short period. The results of the warmup training are given in Tab. 6 and Tab. 7 for the TransResU-Net-4x256 and the TransResU-Net-4x512 respectively.

Based on the results of the warmup training, no single backbone outperforms any of the other[8]. However, we can see a large difference in the metrics for pretrained backbones compared to training from scratch. The results are, in almost all cases, better for pretrained backbones. Comparing the pretrained networks, we identify four candidates for the TransResU-Net-4x256 and three candidates for the TransResU-Net-4x512 that achieve the best result in one of the evaluation metrics. For the TransResU-Net-4x256 the best backbones according to the metrics are the pretrained ResNet-152 [10], ResNeXt-50 32x4d [21], ResNeXt-101 32x8d [21] and Wide ResNet-50-2 [25]. For the TransResU-Net-4x512 the best backbones are the ResNeXt-50 32x4d [21], Wide ResNet-50-2 [25] and the Wide ResNet-101-2 [25].

For the next step of our evaluation, we set the most promising candidates as the intersection of the best backbones of the TransResU-Net-4x256 and TransResU-Net-

[7]An equal-weighted combination of the Dice loss and the Binary Cross Entropy (BCE) loss.

[8]A model outperforms another model if three or more metrics of the model are better than the metric values of the other model.

Backbone	**Pretrained**	**Precision**	**Recall**	**F2**	**IoU**	**Acc**
ResNet-50 [10]		**0.8047**	0.1617	0.1855	0.1528	0.9655
ResNet-101 [10]		0.6736	0.2202	0.2417	0.1914	0.9658
ResNet-152 [10]		0.7710	0.1623	0.1857	0.1517	0.9653
ResNeXt-50 32x4d [21]		0.7732	0.1706	0.1941	0.1595	0.9659
ResNeXt-101 32x8d [21]		0.7035	0.2651	0.2912	0.2333	0.9672
Wide resnet-50-2 [25]		0.7839	0.2291	0.2569	0.2107	0.9670
Wide ResNet-101-2 [25]		0.5841	0.3947	0.4023	0.2973	0.9666
ResNet-50 [10]	✓	0.6678	0.4917	0.5039	0.3906	0.9724
ResNet-101 [10]	✓	0.6411	0.4774	0.4891	0.3740	0.9716
ResNet-152 [10]	✓	0.7203	0.4398	0.4627	0.3721	**0.9739**
ResNeXt-50 32x4d [21]	✓	0.5473	**0.6122**	**0.5761**	0.4019	0.9681
ResNeXt-101 32x8d [21]	✓	0.7185	0.4593	0.4786	0.3823	**0.9739**
Wide ResNet-50-2 [25]	✓	0.6491	0.5249	0.5294	**0.4042**	0.9710
Wide ResNet-101-2 [25]	✓	0.7194	0.4524	0.4732	0.3800	0.9734

Table 6. Results of the TransResU-Net-4x256 as described in Sec. 3.4 after 20 epochs on the validation set for the different backbones available in the authors's GitHub repository.

Backbone	**Pretrained**	**Precision**	**Recall**	**F2**	**IoU**	**Acc**
ResNet-50 [10]		0.7519	0.3047	0.3318	0.2677	0.9687
ResNet-101 [10]		0.7646	0.3617	0.3894	0.3154	0.9698
ResNet-152 [10]		0.7438	0.3691	0.3955	0.3174	0.9691
ResNeXt-50 32x4d [21]		0.7110	0.4683	0.4851	0.3825	0.9715
ResNeXt-101 32x8d [21]		0.7267	0.3866	0.4104	0.3270	0.9694
Wide ResNet-50-2 [25]		0.7514	0.4267	0.4509	0.3632	0.9712
Wide ResNet-102-2 [25]		0.7803	0.3711	0.4001	0.3276	0.9708
ResNet-50 [10]	✓	0.7036	0.5768	0.5796	0.4553	0.9752
ResNet-101 [10]	✓	0.7257	0.5595	0.5717	0.4564	0.9764
ResNet-152 [10]	✓	0.6701	0.5986	0.5933	0.4542	0.9743
ResNeXt-50 32x4d [21]	✓	**0.7588**	0.5431	0.5593	0.4523	**0.9766**
ResNeXt-101 32x8d [21]	✓	0.7048	0.5719	0.5768	0.4526	0.9759
Wide ResNet-50-2 [25]	✓	0.6708	**0.6219**	**0.6109**	0.4676	0.9737
Wide ResNet-102-2 [25]	✓	0.6983	0.5836	0.5868	**0.4599**	0.9749

Table 7. Results of the TransResU-Net-4x512 as described in Sec. 3.4 after 20 epochs on the validation set for the different backbones available in the authors's GitHub repository.

4x512 from the warmup step. We train the most promising candidates for both the TransResU-Net-4x256 and the TransResU-Net-4x512 for further 200 epochs. The training is continued for all models after the last epoch of the warmup run. Following the experimentation of [17], we include a learning rate scheduler that reduces the learning rate by 0.1 if the validation loss does not decrease for five consecutive epochs. We also incorporate an early stopping mechanism that stops the training if the validation loss does not decrease for 50 consecutive epochs. The result of the experimentation step is summarized in Tab. 8.

We can see that TransResU-Net-4x256 with the pretrained Wide ResNet-50-2 [25] backbone performs the best out of all TransResU-Net-4x256 models in the final evaluation. This model has the highest metric score for three of the five metrics evaluated. For TransResU-Net-4x512, the best performance can be achieved using the pretrained ResNeXt-101 32x8d [21] as the backbone. This combination yields the best result for four out of the five performance metrics. Comparing the metric results of the best model for 512$\times$512 sized images to the best model for 256 $\times$ 256 sized images, we can see that TransResU-Net-4x512 significantly outperforms the TransResU-Net-4x256 model.

Interestingly, the two models achieving the highest met-

Model	Backbone	Precision	Recall	F2	IoU	Acc
TransResU-Net-4x256	ResNet-152 [10]	0.6763	0.5370	0.5436	0.4235	0.9740
TransResU-Net-4x256	ResNeXt-50 32x4d [21]	0.6630	0.5400	0.5466	0.4228	0.9742
TransResU-Net-4x256	ResNeXt-101 32x8d [21]	**0.6926**	0.5428	0.5524	0.4317	**0.9757**
TransResU-Net-4x256	**Wide ResNet-50-2** [25]	0.6798	**0.5459**	**0.5547**	**0.4338**	0.9752
TransResU-Net-4x256	Wide ResNet-102-2 [25]	0.6873	0.5299	0.5418	0.4230	0.9748
TransResU-Net-4x512	ResNet-152 [10]	0.7296	0.5930	0.6002	0.4790	0.9773
TransResU-Net-4x512	ResNeXt-50 32x4d [21]	0.7254	0.5885	0.5939	0.4745	0.9770
TransResU-Net-4x512	**ResNeXt-101 32x8d** [21]	**0.7381**	**0.6103**	**0.6146**	**0.4929**	0.9777
TransResU-Net-4x512	Wide ResNet-50-2 [25]	0.7147	0.5897	0.5962	0.4725	0.9772
TransResU-Net-4x512	Wide ResNet-101-2 [25]	0.7288	0.5916	0.6015	0.4829	**0.9778**

Table 8. Results of the TransResU-Net-4x256 and TransResU-Net-4x512 as described in Sec. 3.4 after training according to the setup described in Sec. 5.4.

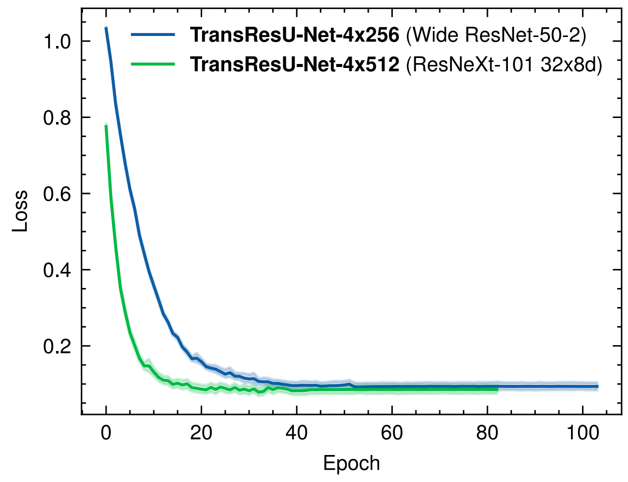


Figure 9. The validation loss during training of the TransResU-Net-4x256 with the pretrained Wide ResNet-50-2 [25] backbone and the TransResU-Net-4x512 using the pretrained ResNeXt-101 32x8d [21] backbone using the DiceBCELoss.

ric in two or more cases in the warmup results are not the best models after continuing the training until convergence.

In the final evaluation step, we use the two best model combinations for 256×256 and 512×512 sized images. The loss of the two best models is given in Fig. 9.

The third and last step of the experimentation with the TransResU-Net-4x256 and TransResU-Net-4x512 is to try further adaptions to the training process. We decided not to increase the number of epochs, the learning rate scheduler, and early stopping patience due to their apparent convergence after $\approx$ 60 epochs (see Fig. 9). We also decided not to adjust the learning rate based on our experience of the initial experimentation prior to this evaluation. Instead, we decided to train the best TransResU-Net-4x256 and TransResU-Net-4x512 models with an adaption to the loss function. Until now, we kept the DiceBCELoss used in [17]. Due to the changes made to the model's architecture to fit the task at hand (see Sec. 2 and Sec. 3.4), the four predicted masks are weighted equally in the loss. Our models' main application is to segment blood vessels in a given input image. To incorporate the differences in the importance of each segmentation mask, we introduce the *Channel Weighted DiceBCELoss*. This loss functions enables us to assign a weight to each channel. The final loss is the mean of the linear combination of the weighted per-channel dice loss and the weighted per-channel BCE loss.

We decided to weight each channel by their respective relative occurrence of class pixels in the training set. On average, $\approx 4.35\%$ of the pixels of an image belong to blood vessels, $\approx 4.47\%$ belong to glomeruli, $\approx 0.36\%$ are unsure pixels, and $\approx 90.83\%$ are background pixels[9]. To calculate the weights, we first subtract each relative occurrence from 1. Then, we normalize the values so that their sum equals 4. The sum of each weight has to add up to 4 since for all weights set to 1, the sum of the weights is 4, and the Channel Weighted DiceBCELoss is equal to the DiceBCELoss. This procedure results in the following weights for each segmentation mask: the blood vessel channel is weighted with ≈ 1.2754, the glomerulus channel is weighted with ≈ 1.2737, the unsure channel is weighted with ≈ 1.3286 and the background channel is weighted with ≈ 0.1223. Based on the first and second evaluation step results, we train the model for 20 warmup epochs without early stopping and learning rate scheduling. After this warmup phase, we train the models for further 60 epochs without early stopping. The remaining setup stays the same as in the previous evaluation steps.

The results for the Channel Weighted DiceBCELoss compared to the DiceBCELoss are summarized in Tab. 9. Still, TransResU-Net-4x512 with the pretained ResNeXt-

[9]This holds for both 256×256 and 512×512 sized images.

Model	**Backbone**	**Precision**	**Recall**	**F2**	**IoU**	**Acc**
DiceBCELoss						
TransResU-Net-4x256	Wide ResNet-50-2 [25]	0.6798	0.5459	0.5547	0.4338	0.9752
TransResU-Net-4x512	**ResNeXt-101 32x8d** [21]	**0.7381**	0.6103	**0.6146**	**0.4929**	**0.9777**
Channel Weighted DiceBCELoss						
TransResU-Net-4x256	Wide ResNet-50-2 [25]	0.6619	0.5592	0.5629	0.4352	0.9738
TransResU-Net-4x512	ResNeXt-101 32x8d [21]	0.6800	**0.6218**	0.6141	0.4708	0.9751

Table 9. Comparison of the results of the TransResU-Net-4x256 and TransResU-Net-4x512 as described in Sec. 3.4 after training using different loss functions.

101 32x8 [21] backbone and DiceBCELoss outperforms all the other models evaluated. It achieves the best result in four out of five metrics. We will use this model and backbone combination in our final evaluation on the test dataset (see Sec. 6).

5.5. Fully Convolutional Transformer

Despite numerous efforts, we could not successfully train the Fully Convolutional Transformer. The trained model either predicted all pixels to be blood vessels or none of them. Our first guess was that there was an implementation error in our adaption of the model to support RGB images. To check this hypothesis, we trained the Fully Convolutional Transformer on a single simple 16×16 sized RGB image, forcing the model to overfit this example heavily. After training, the model could successfully predict the different masks. We concluded that the implementation was correct. Next, we tried to apply the same approach to one of the images in our training set, resized to have dimensions 128×128.

The objective was to overfit the Fully Convolutional Transformer on this single example. However, the model could not predict any of the blood vessel pixels after training. Guessing the model's inability to learn was due to the small image size, we tried to increase the image size to 256×256 and even 512×512 without success. Our last effort was to transform the image of our dataset to grayscale as the figures in [18] and the code provided on GitHub[10] indicate that the Fully Convolutional Transformer was evaluated on grayscale images. This adaption to our code did not affect the training results. As the training took significantly longer than for any of the other models evaluated and due to the limited time frame of this project, we decided not to do more experimentation with the Fully Convolutional Transformer and try out something else. Nevertheless, given a much longer training time frame (based on our experimentation of multiple days to a week) and full-scale images of 512×512, we are convinced that the model could indeed learn to predict blood vessels.

[10]https://github.com/Thanos-DB/FullyConvolutionalTransformer

5.6. SegFormer

In the present work, the SegFormer implementation from Huggingface was utilized, primarily due to its extensive documentation and user-friendly interface. As part of our experiments, we aimed to evaluate the impact of varying data transformations, learning rates, and backbones on the results. It should be noted that all experiments conducted with the SegFormer employed pre-trained backbones. This decision was informed by both the practices observed in related work [15], where pre-trained SegFormers are commonly used, and by our early experiments, which demonstrated that training the model from scratch yielded substantially poorer performance compared to using the pre-trained backbones.

A typical preprocessing operation for image segmentation and machine learning tasks in general is the normalization of images. [3] For our experiments we used Z-Score normalization to adjust the mean standard deviation to approximately 0.0 and 1.0 respectively. This is carried out by calculating the mean and standard deviation on the train set, and then using these values to normalize both the train and validation set. This particular step had a noticeable positive impact on the model's performance, as reflected in the final results table. Subsequently, this form of normalization was applied uniformly across all other experiments.

The next part of the experiments focused on finding the best backbone by analyzing the performance across different model sizes. The original paper suggested five different Transformer encoder backbones of varying sizes, shown in Tab. 1. Because of hardware limits (we used an NVIDIA RTX 3070 GPU with 8 GB of RAM), we could only test models 0-3. The original paper suggested that larger models would yield superior performance and we found this to be true with our data, but the difference was smaller than we anticipated. Additionally, it is important to note, that the training time of 37 seconds per epoch for model MiT-b0 increased to 62 and 136 seconds per epoch

Encoder	Transforms	LR	Precision	Recall	F2	IoU	Acc
MiT-b0		6e-4	**0.7700**	0.4167	0.4431	0.3646	0.9722
MiT-b0	norm	6e-4	0.7307	0.4892	0.5048	0.4033	0.9744
MiT-b1	norm	6e-4	0.7492	0.4782	0.4990	0.4027	0.9748
MiT-b2	norm	6e-4	0.6792	0.5899	0.5850	0.4511	0.9736
MiT-b2(128)	norm	6e-4	0.4103	0.1668	0.1707	0.1239	0.9560
MiT-b2(256)	norm	6e-4	0.6155	0.4734	0.4733	0.3505	0.9688
MiT-b2	norm	1e-4	0.6686	**0.6366**	0.6203	0.4756	0.9733
MiT-b2	norm	1e-5	0.7276	0.5248	0.5371	0.4273	0.9740
MiT-b2	norm,flip	1e-4	0.6854	**0.6353**	**0.6272**	**0.4843**	0.9769
MiT-b2	norm,flip,HSV	1e-4	0.5552	0.5661	0.5296	0.3668	0.9670

Table 10. Segformer results

for models MiT-b1 and MiT-b2, respectively.

When comparing these models, the losses seemed to converge, but the IoU score on the validation set had a lot of variance between different epochs. So, we tried different learning rates. Lowering the learning rate from 6e-4 to 1e-4 helped, but going further down to 1e-5 didn't make things more stable but made the training significantly longer.

We also tried to make the results more consistent by using data augmentation. We started with random vertical and horizontal flips, which helped increasing the performance of various metrics and made the training more stable. Another common Transformation that is done to increase the Robustness of the Results is Random HSV transformations. [9] In our specific case, this did not enhance performance nor contribute to more stable training. This is likely because random HSV transformations are often employed in segmentation tasks, where variations in lighting conditions within the images are typical. Since the Images used in our data are laboratory Images with fixed lighting conditions, the random HSV transformations didn't improve the results.

Fig. 10 illustrates the validation losses associated with various augmentation strategies. A general observation from the figure is that the Segformer converges rapidly, there is barely any improvement for any experiment after around 12 epochs. What is also visible is that the variance between the epochs, is rather high, even for our best model using a learning rate of 1e-4 and random flips. his inconsistency is likely attributable to the fact that the model is trained on a batch size of only 2. Ideally, we would have experimented with larger batch sizes to investigate if this would lead to more stable training. However, due to GPU memory limitations this was not possible. We also tried out Resizing the original 512x512 image to 128x128 and 256x256. We didn't expect these transformation to improve the performance but we were curious if these transformations could increase the training time. As visible, resizing

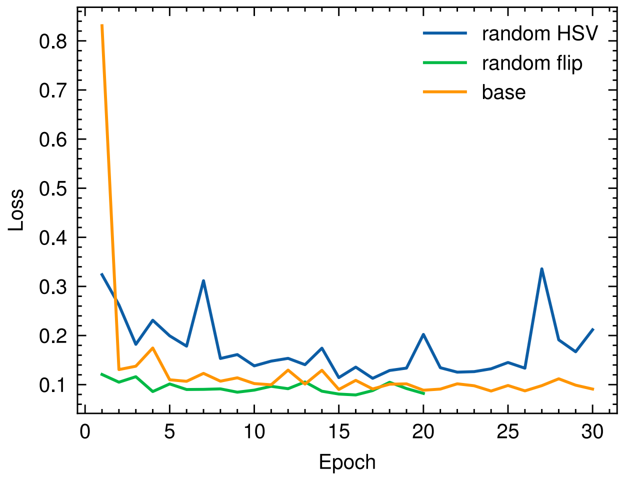


Figure 10. Segformer losses of different Augmentations Strategies

the image to 128x128 made a performance a lot worse while 256x256 images kept somewhat competitive Results even though there is a noticeable drop in performance in comparison to the native Images. Interestingly the training time for 128x128 images and 256x256 images was roughly the same at around 42 seconds per epoch. This is probably because generating the multilevel feature maps in the encoder is not that big of computational bottleneck anymore since the generated features are very small in this case (128: 32,16,8,4,2). So training the model on 256x256 images seems to be a good trade off between performance and training time.

Since we utilized pretrained backbones and there was no straightforward method to alter the loss function used in the Segformer within the Huggingface interface, we did not conduct any experiments on different loss functions for the Segformer model.

Model	Backbone	Precision	Recall	F2	IoU	Acc
U-Net [14]	ResNet-34	0.6452	0.5433	0.5419	0.4110	**0.9716**
DuckNet-18	ResNet18 [12]	0.5496	0.7185	0.6208	0.4312	0.9341
Segformer [20]	MiT-b2	0.7012	**0.7449**	**0.7034**	0.5452	0.9631
TransResU-Net-4x512	ResNeXt-101 32x8d [21]	**0.7908**	0.6562	0.6574	**0.5548**	0.9701

Table 11. Results for the best model configurations using the test dataset

In conclusion, our analysis determined that the optimal model was configured with the MiT-b2 backbone, normalization, random flips, and a learning rate of 1e-4. This model exhibited the highest performance across nearly all evaluated metrics, and the training results were the most stable compared to other configurations. Moreover, it was trained for only 16 epochs, resulting in an efficient training time of just 32 minutes, further highlighting its effectiveness.

6. Result and Discussion

In this section, we select the best model from each of our previous experiments. Namely, the U-Net, lr=0.0005, ResNet-34, Cross Entropy Loss, the DUCK-Net18 with FocalLoss, the TransResU-Net-4x512 with pretrained ResNeXt-101 32x8 [21] and DiceBCELoss, and the Segformer, lr=0.0001, MiT-B2, with z-score normalization and random flips. We will ignore the LinkNet as well as the Fully Convolutional Transformer in the final evaluation as we were not able to train these model successfully (see Sec. 5.2 and Sec. 5.5).

Finally, each of the selected models is evaluated using the same metrics as in Sec. 5 but on the test set. The results are summarized in Tab. 11. We observe that the two models based on Image Transformers, SegFormer and TransResNet, significantly outperform the other two models. Both these models achieved similar IoU scores, yet some distinctions are notable.

The TransResNet model has a Precision score that is approximately 0.09 higher than that of the SegFormer model. Conversely, the SegFormer model has a Recall score that's approximately 0.09 higher than that of TransResNet.

This observation indicates that TransResNet is more conservative in predicting blood vessels, leading to fewer false positives. In contrast, SegFormer is more aggressive in predicting blood vessels, resulting in more false positives.

Given that the IoU of the two models is nearly identical, it is challenging to declare one model as superior to the other. Instead, the choice between SegFormer and TransResNet should be based on specific needs and preferences, such as the relative importance of having fewer false positives versus more blood vessel predictions.

Finally we show five example images in Fig. 11, ground truth masks, and predictions, several key observations were noted. All models had no problem detecting the glomeruli, which were pretty easily characterizable, showcasing a consistent ability to recognize these structures across different architectures.

Both the TransResUNet and U-Net models displayed a tendency to predict the unsure mask more often, possibly revealing a certain behavior or attribute within these specific architectures. The study also showed that bigger blood vessels were easier to predict than smaller ones. Smaller vessels were sometimes undetected or only partially captured, exposing potential limitations in the sensitivity of the models to detect more delicate structures.

Additionally, the U-Net and DuckNet models exhibited a tendency to ”fantasize” or over-predict, in that they not only drew more blood vessels than there were but also randomly placed dots in the middle of the image.

Nevertheless, from a visual inspection of these images, it can be concluded that all models performed a decent job in detecting blood vessels. The results were generally satisfactory across the board, with notable better performance observed for the Segformer and TransResUnet models.

7. Conclusion

The exploration into accurate segmentation of microvascular structures within human kidney tissue slides, as detailed in this study, uncovers valuable insights and advancements in the field of medical image segmentation. In this study, the efficacy of various state-of-the-art computer vision models, including U-Net, DUCK-Net, TransResU-Net, and SegFormer, was assessed, revealing differing strengths and weaknesses among these architectures.

Through rigorous experimentation and evaluation, it was observed that transformer-based architectures such as TransResU-Net and SegFormer outperformed the other models in the task at hand. These models demonstrated a balanced trade-off between precision and recall, offering a flexible choice that can be tailored to specific needs and preferences, whether the goal is to minimize false positives or maximize the detection of blood vessels.

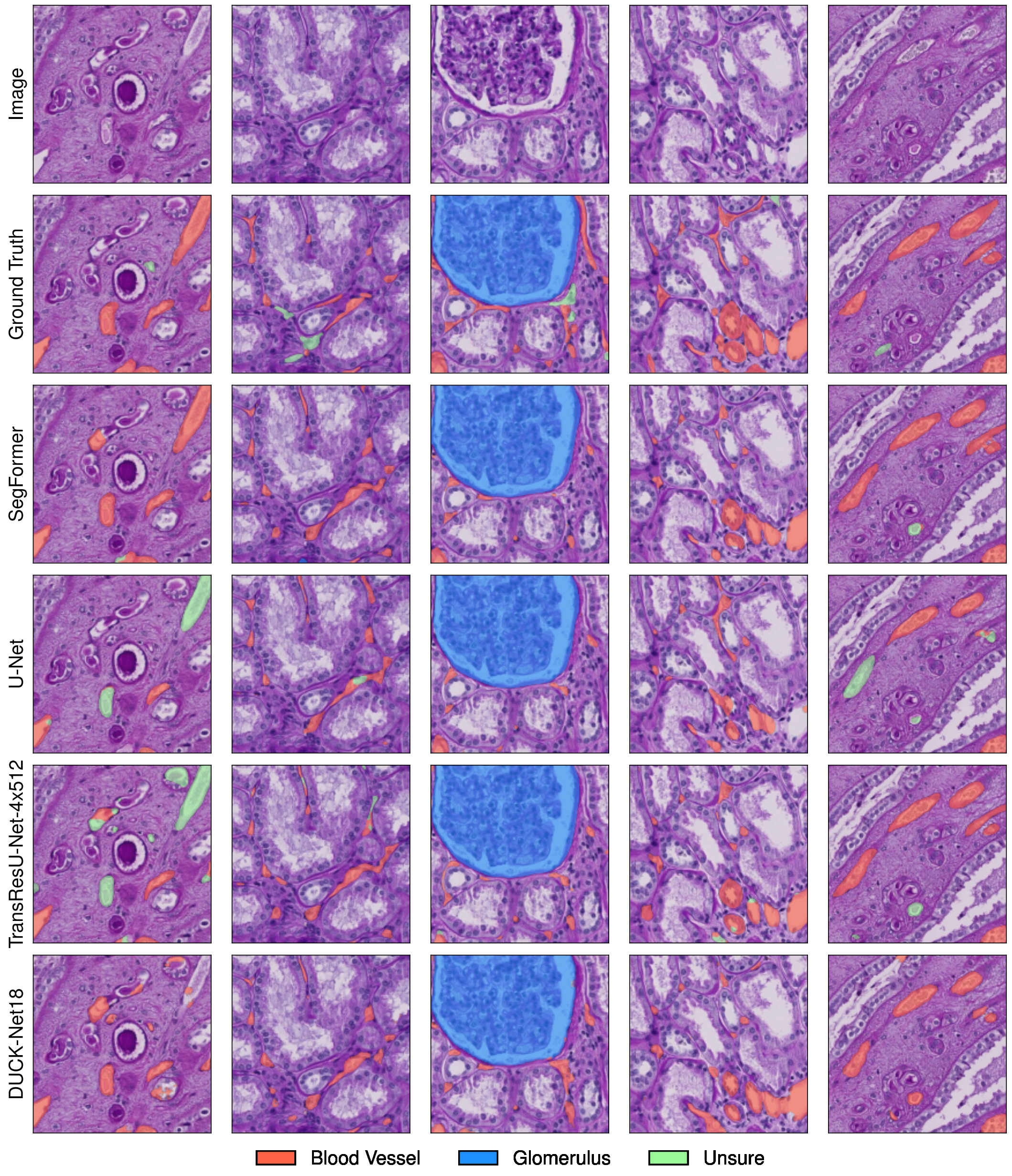


Figure 11. Final Results

However, the study was not without its challenges. While the more advanced models succeeded in capturing larger blood vessels, they occasionally struggled with the detection of smaller vessels. This limitation hints at potential areas for further research and optimization, particularly in enhancing the sensitivity of models to recognize more intricate structures within the tissue slides.

The evaluation also unveiled specific behavioral tendencies within the models. For instance, both TransResU-Net and U-Net showed a propensity to predict the ”unsure” mask more frequently, which could be explored further to understand the underlying causes. The tendency for certain models to ”fantasize” or over-predict, placing random structures within the images, also merits closer examination.

Visually, the models performed satisfactorily across the board, although a clear edge was observed in the predictions made by the SegFormer and TransResUnet models. Their success in the task reinforces the potential of transformer-based architectures in medical segmentation, aligning with the growing trend of applying transformers in various domains.

In conclusion, this study presents a comprehensive examination of selected models for microvascular segmentation, contributing to the broader goal of automated and precise microvascular knowledge discovery. It not only sheds light on the current capabilities and limitations of modern segmentation models but also sets the stage for future research directions. The insights gained can be instrumental in refining existing models and in the creation of new methodologies to further the HuBMAP initiative's objective of detailed cellular mapping within the human body.

References

[1] Segformer. `https://huggingface.co/docs/transformers/model_doc/segformer`. Accessed: 17-08-2023. 1, 6

[2] Katherine Gustilo Katy Borner Ryan Holbrook Yashvardhan Jain Addison Howard, HCL-Jevster. Hubmap - hacking the human vasculature. https://kaggle.com/competitions/hubmap-hacking-the-human-vasculature, 2023. 1, 2, 4

[3] Peshawa Jamal Muhammad Ali, Rezhna Hassan Faraj, Erbil Koya, Peshawa J Muhammad Ali, and Rezhna H Faraj. Data normalization and standardization: a technical report. *Mach Learn Tech Rep*, 1(1):1–6, 2014. 15

[4] Abhishek Chaurasia and Eugenio Culurciello. Linknet: Exploiting encoder representations for efficient semantic segmentation. pages 1–4, 12 2017. 3, 9

[5] Ugur Demir, Zheyuan Zhang, Bin Wang, Matthew Antalek, Elif Keles, Debesh Jha, Amir Borhani, Daniela Ladner, and Ulas Bagci. Transformer based generative adversarial network for liver segmentation, 2022. 4

[6] Foivos I. Diakogiannis, François Waldner, Peter Caccetta, and Chen Wu. ResUNet-a: A deep learning framework for semantic segmentation of remotely sensed data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 162:94–114, apr 2020. 4

[7] Michal Drozdzal, Eugene Vorontsov, Gabriel Chartrand, Samuel Kadoury, and Chris Pal. The importance of skip connections in biomedical image segmentation. 08 2016. 3

[8] Razvan-Gabriel Dumitru, Darius Peteleaza, and Catalin Craciun. Using duck-net for polyp image segmentation. *Scientific Reports*, 13, 06 2023. 1, 3, 4

[9] Philipp Gräbel, Martina Crysandt, Reinhild Herwartz, Melanie Baumann, Barbara M Klinkhammer, Peter Boor, Tim H Brümmendorf, and Dorit Merhof. Reduction of stain variability in bone marrow microscopy images: Influence of augmentation and normalization methods on detection and classification of hematopoietic cells. In *Bildverarbeitung für die Medizin 2021: Proceedings, German Workshop on Medical Image Computing, Regensburg, March 7-9, 2021*, pages 141–146. Springer, 2021. 16

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 4, 11, 12, 13, 14

[11] Juana Valeria Hurtado and Abhinav Valada. Chapter 12 - semantic scene segmentation for robotics. In Alexandros Iosifidis and Anastasios Tefas, editors, *Deep Learning for Robot Perception and Cognition*, pages 279–311. Academic Press, 2022. 7

[12] Debesh Jha, Pia H. Smedsrud, Michael A. Riegler, Dag Johansen, Thomas de Lange, Pal Halvorsen, and Havard D. Johansen. Resunet++: An advanced architecture for medical image segmentation, 2019. 3, 10, 17

[13] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. pages 779–788, 06 2016. 3

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing. 1, 2, 3, 4, 17

[15] Théo Sourget, Syed Nouman Hasany, Fabrice Mériaudeau, and Caroline Petitjean. Can segformer be a true competitor to u-net for medical image segmentation? In *27th Conference on Medical Image Understanding and Analysis 2023*, 2023. 6, 15

[16] Abdel Aziz Taha and Allan Hanbury. Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool, Aug 2015. 7, 8

[17] Nikhil Kumar Tomar, Annie Shergill, Brandon Rieders, Ulas Bagci, and Debesh Jha. Transresu-net: Transformer based resu-net for real-time colonoscopy polyp segmentation, 2022. 1, 4, 5, 11, 12, 13, 14

[18] Athanasios Tragakis, Chaitanya Kaul, Roderick Murray-Smith, and Dirk Husmeier. The fully convolutional transformer for medical image segmentation, 2023. 5, 15

[19] Meysam Vakili, Mohammad Ghamsari, and Masoumeh Rezaei. Performance analysis and comparison of machine and deep learning algorithms for iot data classification, 2020. 7

[20] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers, 2021. 6, 17

[21] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks, 2017. 11, 12, 13, 14, 15, 17

[22] Michael Yeung, Evis Sala, Carola-Bibiane Schönlieb, and Leonardo Rundo. Unified focal loss: Generalising dice and cross entropy-based losses to handle class imbalanced medical image segmentation, 2021. 8, 9

[23] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions, 2016. 4

[24] Jiaqian Yu, Jingtao Xu, Yiwei Chen, Weiming Li, Qiang Wang, Byungin Yoo, and Jae-Joon Han. Learning generalized intersection over union for dense pixelwise prediction. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12198–12207. PMLR, 18–24 Jul 2021. 7

[25] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2017. 11, 12, 13, 14, 15

[26] Zharfan Zahisham, Chin Poo Lee, and Kian Ming Lim. Food recognition with resnet-50. In *2020 IEEE 2nd International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*, pages 1–5, 2020. 8