SenNetworks Final Project
CS284A: AI in Biology Fall 2023

Andrew Kettle [*1] **Hitarth Gandhi** [*1]

Abstract

Semantic segmentation is a computer vision task that has been generating significant interest in the field of machine learning. Although many use-cases exist in the space of natural images, such as self-driving cars and robotics, medical image analysis could also benefit greatly from well tuned segmentation networks. In this work, we aim to explore existing architectures on the SenNet + HOA Kaggle Vascular dataset. We will show a baseline U-Net model and then various transformer models.

1. Introduction

Innovations in medical imaging technology have ushered in a new era, providing scientists with unprecedented opportunities to unravel the intricacies of the human body. The Kaggle challenge, "SenNet + HOA - Hacking the Human Vasculature System," emerges as a pivotal opportunity to address a critical bottleneck in the Vasculature Common Coordinate Framework (VCCF) annotation process. This bottleneck is characterized by the slow and labor-intensive nature of expert annotators manually marking each image in the dataset, which can take more than six months to complete. This inefficiency poses a substantial obstacle to creating a comprehensive VCCF, impeding our understanding of how blood vessels intricately influence various cells within the human body. Our project's primary objective is to harness the power of machine learning to develop an efficient system for annotating blood vessels. Our system takes as input 2D slices derived from the 3D imaging of organs using Hierarchical Phase-Contrast Tomography (HiP-CT). The desired output is a segmentation mask of the blood vessels within the images.

In this project, we propose applying modern machine learning techniques, particularly comparing the performance of a Convolutional Neural Network (CNN) with modern Segmentation Transformers. Our comparative analysis will encompass established models, such as U-Net, and more recent approaches, including "Segformer" and "Segment Anything." This exploration evaluates these models' effectiveness regarding segmentation quality. By leveraging the strengths of machine learning, we aspire to streamline the laborious annotation process, thus contributing to the accelerated development of a comprehensive VCCF. Ultimately, our efforts aim to facilitate a deeper understanding of the intricate relationship between blood vessels and various cellular components in the human body, propelling advancements in human vasculature systems.

2. Related Works

Various approaches have been explored in the domain of blood vessel segmentation, ranging from classical algorithms to modern machine-learning techniques. A study, "Blood Vessel Enhancement and Segmentation Using Wavelet Transform," (Akram et al., 2009) focuses on classical methods employing wavelet transform for blood vessel visibility. Many similar classical approaches have used different kinds of approximations and patterns for segmentation.

One noteworthy work leveraging machine learning was the novel MEA-net model (Zhu et al., 2023), which employed CNNs, specifically adopting a U-Net-based architecture, for blood vessel segmentation within retina images. However, most machine learning applications in blood vessel segmentation remain focused on the retina, neglecting the exploration of other organs.

On a broader scale, many comprehensive reviews ((Moccia et al., 2018), (Fraz et al., 2012)) have been done that shed light on various segmentation methods but emphasize a predominant focus on machine learning applications within retinal images. One key finding from these reviews is that the absence of a single unified system for blood vessel segmentation is due to vast differences in the anatomy across organs and disparities in imaging technology used across different datasets. Thus, developing a system that can generalize over multiple organs is critical in aiding future blood vessel research.

3. Dataset

3.1. Dataset Details

We used the provided 3D Kidney scan dataset in the SenNet + HOA Kaggle challenge (SenNet & HOA, 2023). This dataset contains scans of five different people's kidneys. Kidneys 1-3 are used for training and validation and kidneys 5 and 6 are provided as a testset. Each input image is a 2D slice along the z-axis over the 3D volume of a particular kidney. These images appear as RGB tensors on disk because the grayscale value from the 2D map is copied depth wise. Each label is a 2D segmentation mask with 255 representing the target and 0 representing the background.

3.2. Importing Items to the Dataset

Importing the data was done through PyTorch's Dataset and Dataloader classes. Storing all of the images and masks in RAM would have been intractable because the dataset is around 40 GB. Initially, we considered only storing the image and label names and then appending the path in PyTorch's __get_item__() item function. But, the multi-level structure of the dataset (kidney_1, kidney_2, etc) presented an issue for this method. We then considered using only the image and label names, but that was not a concrete solution because of the overlapping image names. Multiple 00000.tif files exist in the dataset, one under each kidney. We elected to append the directory's name (kidney_1, kidney_2, etc) to the name of each tiff file. Our final implementation stores the full path to each image and mask directly, allowing us to use PyTorch's lazy loading via __get_item__() to minimize RAM usage. The downside of doing this is that reading from disk (as opposed to RAM) is very slow, so the training process was extended.

3.3. Data Transformations and Lessons

After importing the raw data, we were left with an RGB tensor (where R=G=B) for the input image and a grayscale binary mask containing 255 for the target and 0 for the background. The width and height of each image varied depending on which kidney was being used, so we elected to downsample both the input image and mask to 512x512. This was also the natural image size for U-Net and Segformer, so it made sense to perform this step in preprocessing. One of our most significant early mistakes came during this resizing. Using bilinear interpolation, neighboring pixels are used to calculate the downsampled pixel's size. This means that a collection of 255's and 0's can combine to create a new number, such as 167. The issue is that segmentation masks interpret different numbers as different classes, which is not the case for us. We remedied this error by shifting our image and mask preprocessing to Hugging Face's SegformerImageProcessing class, which allowed us to normalize the input image and fix the masks at the same time.

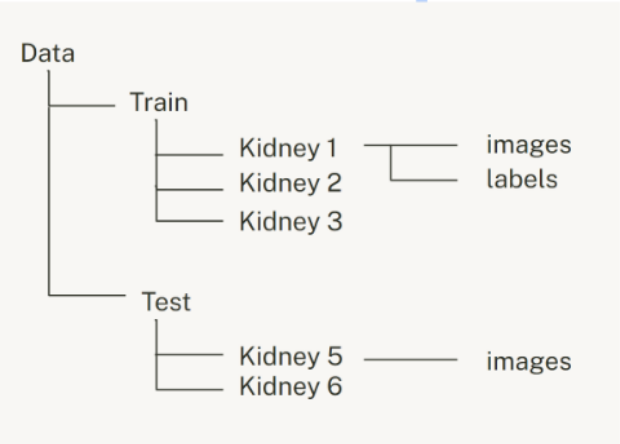

Figure 1. Full dataset appearance on disk. For this submission, only kidney 1 was provided.

4. Technical Approach

4.1. U-Net

For testing CNN, we used an architecture based on the U-Net model (Ronneberger et al., 2015). U-Net is renowned for its success in semantic segmentation tasks and features a downscaling path, a bottleneck layer, and an upscaling path. This architecture is particularly well-suited for segmenting blood vessels within 2D slices obtained from HiP-CT imaging of organs. The downscaling path of U-Net captures intricate high-level features, gradually reducing the spatial dimensions of the input. This reduction allows the model to learn contextual information efficiently. The bottleneck layer bridges the downscaling and upscaling paths, providing a compressed representation of features. In our implementation, we enhance the U-Net architecture with Batch Normalization. This technique normalizes the input of each layer, leading to faster convergence during training and improved generalization to new datasets. Batch Normalization introduces normalization layers after each convolutional layer, ensuring stable input distributions throughout the network. It also stabilizes and accelerates the training process, allowing the model to adapt more effectively to the complex features in the HiP-CT images.

4.2. Segformer

The first transformer we used for semantic segmentation was Segformer (Xie & et al., 2021). Segformer is one of the first transformers trained end-to-end (encoder and decoder) to do segmentation. Its encoder features a derivative of ViT (Dosovitskiy & et al., 2020) that adds hierarchi-

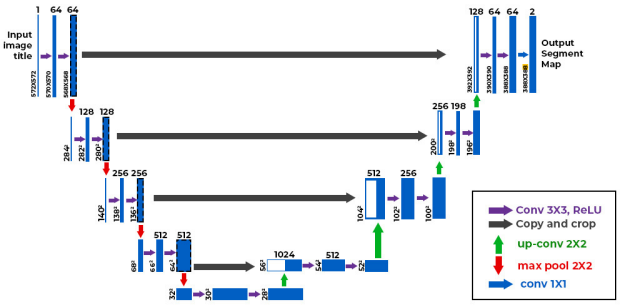

Figure 2. U-net architecture

cal feature representations, increased efficiency multi-head self-attention (MHSA) layers, and a feed-forward network replacement for positional encoding. One of the key insights in the paper was the analysis of Segformer's receptive field. Traditional convolutional approaches increase their receptive field as the network gets deeper and downsampling is used. Transformers are able to utilize MHSA to make connections between every pixel in the image in each encoder step. This leads to a much wider receptive field earlier in the network and thus more context of the entire segmentation scene throughout the training process. The figure used by the segformer team comparing a convolutional network and Segformer is featured below as Figure 4. The decoder features a a lightweight MLP. The combination of a lightweight encoder and decoder lead to a relatively lightweight architecture as far as transformers are concerned. NVIDIA released various models on HuggingFace, called MiT, where B0 is the most lightweight model and B5 is the heaviest. We elected to work using B0 because of compute and schedule requirements.

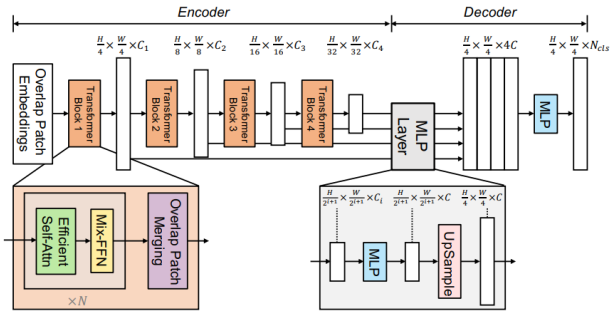

Figure 3. Segformer Architecture

4.3. Segment Anything Model (SAM)

After finishing up with U-Net and Segformer, we thought it would be interesting to experiment with SAM's (Kirillov & et al., 2023) zero-shot capabilities. SAM is a promptable

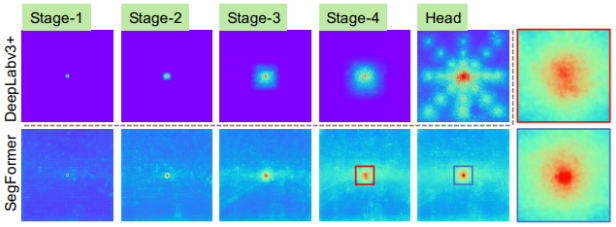

Figure 4. Segformer Receptive Field Analysis

segmentation transformer. This means that it accepts two inputs: the image to segment, and some kind of prompt, such as a bounding box or text describing what to segment. Zero-shot means that SAM did not see our images at any point during training. We used SAM's auto-prompting function to generate prompt embeddings. In our case, the image was sampled over different areas.

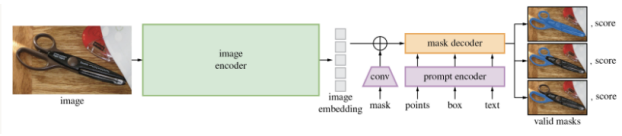

Figure 5. SAM Architecture

4.4. Software Libraries

4.5. A: Code that we wrote

We wrote both of the notebooks in the git repository[1], unet.ipynb and transformers.ipynb. Within those notebooks, we wrote the dataset, dataloader, data cleaning, and main training loop code. Both of our model definitions were derived from external sources, either tutorials or APIs. They are discussed in subsection B. The code for decoding and displaying the algorithms was written by us as well. In short, everything but the model definitions and select items mentioned in section B was written by us.

4.6. B: Code that we used

In general, our code was built on top of PyTorch and Python. We used various packages related to both of those packages. For the U-Net implementation, we followed the tutorial listed here: UNET tutorial. For the Segformer implementation, we used HuggingFace's API to interact with their Segformer class (HuggingFace, 2021). This code included some of the preprocessing steps and the model definition for Segformer. We used certain custom loss functions that combined the dice score and cross entropy from the Kaggle page here (RNA, 2021). They were used directly for the

[1] https://github.com/Akettle44/SenNetworks

U-Net implementation but modified for Segformer's use-case. We followed Facebook AI's Research Lab tutorial as a baseline for the SAM implementation, which can be found here (FAIR, 2023).

5. Experiments and Evaluation

5.1. U-Net

5.1.1. U-Net with Binary Crossentropy Loss

We employed a U-Net architecture for our initial experiment to tackle the blood vessel segmentation task, utilizing the Binary Cross entropy loss function. We used Adam Optimizer, a popular choice for its adaptive learning rates.

Table 1. Hyperparameter Values for Experiment 1

Hyperparameter	**Value**
Learning Rate (*lr_init*)	1×10^{-4}
Batch Size	16
Epochs	12
Loss Function	BCE loss
Optimizer	Adam

Binary Crossentropy suits our pixel-wise segmentation task. The Adam optimizer, known for adaptive learning rates, facilitated model optimization.

A learning rate (*lr_init*) of 1×10^{-4} controlled convergence steps and a batch size of 16 balanced efficiency and stability.

The experiment ran for 12 epochs, chosen for computational efficiency and model convergence. These hyperparameters form a starting point for further refinement.

Due to RAM and GPU limitations, the model was trained only the kidney_1 dataset and tested on kidney_2 dataset.

Results:

Below are the results from the experiment. As we can see the loss curve (Fig. 6) is not very smooth and the convergence is not ideal. From visual inspection, the results (Fig. 7) are quite good. We got a dice score of 0.45 on our test dataset.

5.1.2. U-Net with Combined DICE + BCE Loss

For our second experiment, we aimed to improve blood vessel segmentation by combining the DICE and Binary Crossentropy (BCE) loss functions. Additionally, we introduced an exponential decaying learning rate scheduler (StepLR) to enhance stable learning. The model was trained exclusively on the kidney_1 dataset due to RAM and GPU limitations, with testing performed on the kidney_2 dataset.

Results:

The loss curve for Experiment 2 (Fig. 8) exhibits improved

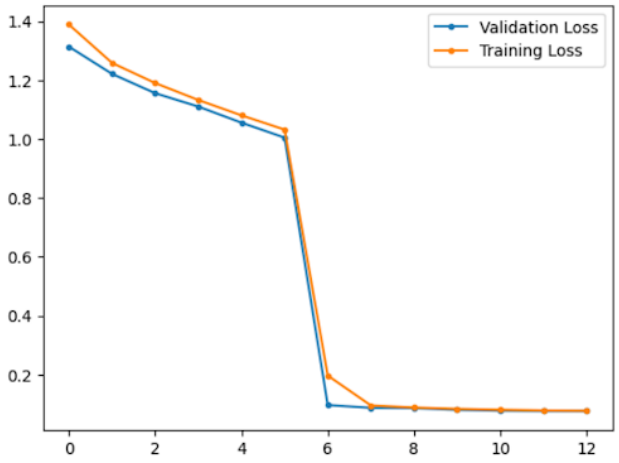

Figure 6. Training and Validation Loss Curve for Experiment 1

Table 2. Hyperparameter Values for Experiment 2

Hyperparameter	**Value**
Learning Rate (*lr_init*)	1×10^{-4}
Batch Size	16
Epochs	8
Loss Function	DICE + BCE Loss
Optimizer	Adam
Learning Rate Scheduler	StepLR(step=1, gam=0.95)

convergence compared to the first experiment. Visual inspection of the segmentation results (Fig. 9) indicates favorable outcomes, with predictions closely aligning with the ground truth labels. The dice score for test dataset is 0.68 which is higher than that of experiment 1 suggesting that this model is better the previous one.

The combination of DICE, BCE loss, and the learning rate scheduler demonstrates promising convergence and improvements in segmentation quality. When the model is tested on Kaggle testing data, it performs poorly. This suggests the model still does not generalize well to a wide range of data. Thus, future work might include developing a custom encoder module to help the model learn a more general trend.

5.2. Segformer

5.2.1. Initial Experiments

We used HuggingFace's Segformer API to interact with Segformer in the most approachable manner. Before getting into the model code, we had to preprocess our images to the format Segformer expects. After importing the tiff images using OpenCV, we used Segformer's *SegformerImageProcessor* class to preprocess inputs. Since we performed normalization, there was an option to use the default ima-

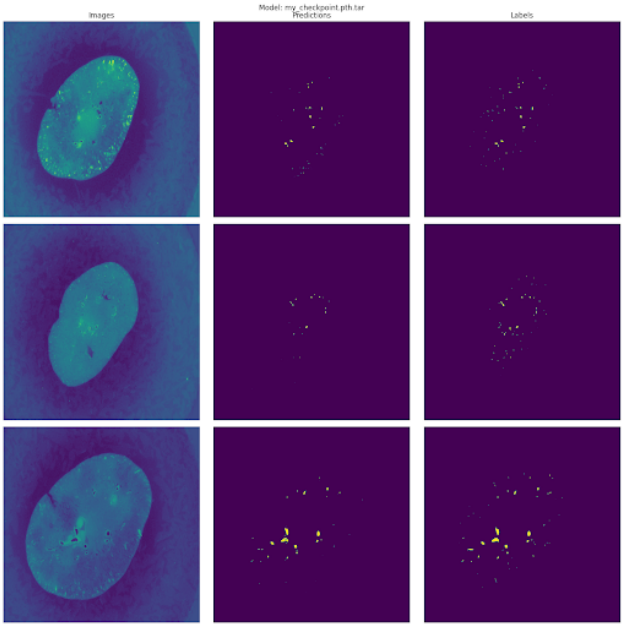

Figure 7. Results for Experiment 1

genet mean and standard deviation or the dataset's mean and standard deviation. We wrote a cell to calculate the datasets mean and stddev, but found that the imagenet mean and stddev performed better in the experiments. We used HuggingFace's *SegformerForSemanticSegmentation* class with pretrained imagenet weights on mit-b0 for our model. Our training process featured a classic PyTorch training and validation loop and loaded data using PyTorch's dataloader. Every single transformer experiment used fine-tuning instead of training from scratch due to the computational requirements. We experimented with numerous different loss functions with Segformer, detailed in table 3 below.

Table 3. Different Loss Functions for Segformer Experiments

Loss Function
Binary Cross Entropy
Weighted Cross Entropy
Cross Entropy and Dice

Using the entire dataset (kidney1, kidney2, kidney3), only Binary Cross Entropy converged. Even after that convergence, the outputs would be uniformly zero, suggesting that the model was underfitting by only predicting the negative class due to the frequency of its appearance relative to the positive class. The natural conclusion was to attempt to weight the loss function. The tricky part about this was that the loss function was hidden behind HuggingFace's API. We circumnavigated this issue by calculating loss ourselves using the unnormalized logits, which is now commented out in the *SegFormerCustom* class cell. After adding weight

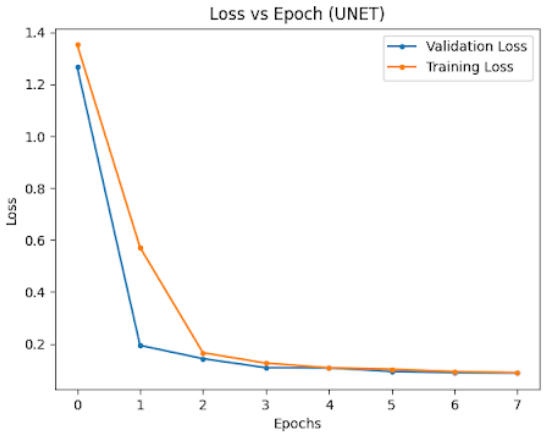

Figure 8. Training and Validation Loss Curve for U-Net Experiment 2

to the positive class relative to the frequency difference (we estimated it at about 50 to 1), the loss value increased, which was expected, but the model still did not converge. We also tried using Cross Entropy with the Dice loss, but that also didn't converge for our hyperparameter settings. We postulate that it is possible our hyperparameter settings were wrong or there was a bug in our code during these experiments. We found numerous bugs over the course of development so it is not outside the realm of possibility to think they could have affected training. Nevertheless, these results caused us to take a step back to analyze the dataset and our training process better. When looking at the dataset, it was clear that there was large variance in the different kidneys. For example, kidney3 often had extremely sparse segmentation maps, where one could easily see how a classifier could achieve low loss by uniformly predicting background. At the same time, we learned about a technique in segmentation where the background class is ignored during training. Using this technique, loss is only calculated over the target pixels, so one does not have to worry about weighting the loss function. We decided to simplify Segformer's problem by drawing training and validation samples only from the kidney1 folder and using the background ignoring technique.

5.2.2. Segformer Experiment 2

As discussed above, the modifications we made for experiment two were as follows:

- Draw training and validation samples only from kidney1
- Use background ignoring mask in loss

The rest of the hyperparameters are collected in the table

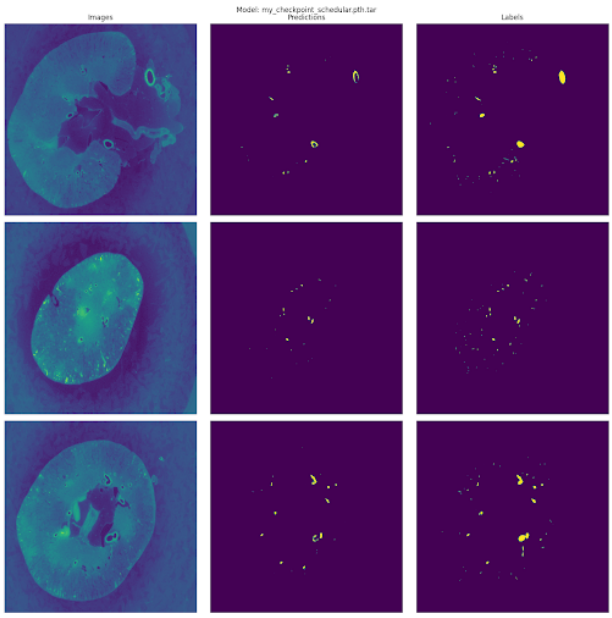

Figure 9. Results for U-Net Experiment 2

below. We chose AdamW with a learning rate of 0.000006 because that is an order of magnitude lower than the default Segformer learning rate. We chose this value because we are fine-tuning, so we only expect to make small updates to the weights. We trained for 60 epochs using a multi-step learning rate scheduler based on fixed milestones in training, at epochs 20 and 40. The decay rate was an order or magnitude, or 0.1.The batch size was set to 8 because that is the maximum size our NVIDIA RTX3080 with 12GB of VRAM could support during the training process. The loss function was Binary Cross Entropy with the background ignoring mask. This configuration is set by passing num_labels=1 to the Segformer initialization. Training took about 40 minutes for 60 epochs.

Table 4. Segformer Hyp. Values Experiment 2

Hyperparameter	**Value**
Learning Rate (*lr_init*)	$6 \times 10^{-}6$
Scheduler	Multi-Step
Batch Size	8
Epochs	60
Loss Function	BCE loss
Optimizer	AdamW

The results for Segformer experiment 2 are showcased below. Convergence happens very quickly, with many of the later epochs seemingly unnecessary. For the images, one can see that they pass the eye-test, the prediction and labeled segmentation maps look similar to one another. Using a more concrete test, we evaluated the prediction and label overlap using a dice score, which is the equivalent of an F1 score for segmentation maps. Dice scores are bounded between [0, 1], where 0 is no overlap and 1 is a perfect match. The dice scores are plotted over the predictions in a white box, but they can be hard to read in the conference format. For images [0, 1, 2], the dice scores are as follows: [0.643, 0.578, 0.438]. These dice scores are suggest that the model is definitely learning the underlying representation, but it is missing some details. Although it may seem counter-intuitive, this is not necessarily a bad thing. If the dice score was 1, it would likely represent that the model is overfitting, where as our results indicate it is learning a more general representation.

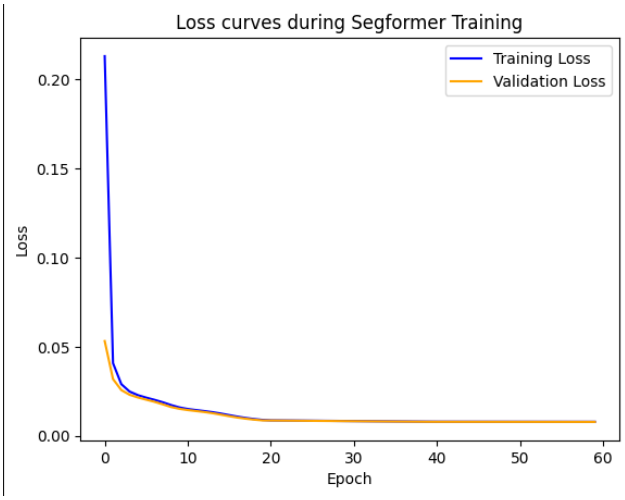

Figure 10. Loss Curves for Segformer Exp. 2 Training

5.3. Zero-Shot SAM

The zero-shot results for SAM shown below in Figure 12 used the checkpoint *sam_vit_h_4b8939.pth*. SAM performs poorly on our vascular dataset challenge. Looking at the bottom row, it is evident that some segmentation is taking place, but it isn't relative to our problem. The issues are summarized below. Problem one is the biggest issue for the given dataset. Generating bounding boxes seems difficult given the structure of the segmentation maps. Problems 2-4 could be solved by fine tuning SAM.

1. SAM needs to be prompted (bounding box, text, etc) which isn't included in our dataset. We could potentially augment bounding boxes, but the dataset doesn't seem that well suited to rectangular boxes.

2. The structure of biological images differs heavily from "natural images". Examples: Different collection modaltiies (X-Ray, ultrasound, etc), No color features (images are grayscale upscaled to RGB)

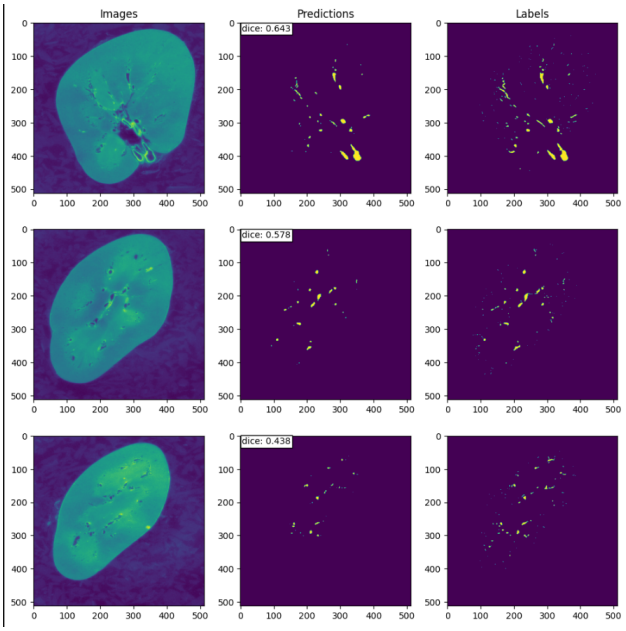

Figure 11. Results for Segformer Exp. 2 Model

3. SAM is performing instance segmentation, but we have a binary segmentation problem

4. SAM does not have knowledge of the vascular problem we are trying to solve.

6. Discussion and Conclusion

6.1. What We Learned

We learned a lot from this final project. For Andrew, this was his first time using HuggingFace for working with Transformers. Working with the different loss functions also helped him solidify the differences between cross-entropy and binary-cross-entropy, as well as converting unnormalized logits into predictions. For Hitarth, this was his first time working with large image datasets and CNNs. Dealing with such data helped him learn about preprocessing the images and creating data loaders. Working with such large data and models also helped him understand RAM and GPU memory limitations and how to effectively clear old memory from them.

6.2. Things We Could Have Done Differently

In general, we were surprised that Segformer did not generalize as well to the entire Kidney dataset. We expected that the increased receptive field would improve our ability to model varying scenes. Most of our initial attempts were run on the full dataset, but we should have started with a smaller subset of the dataset earlier. This would have use helped

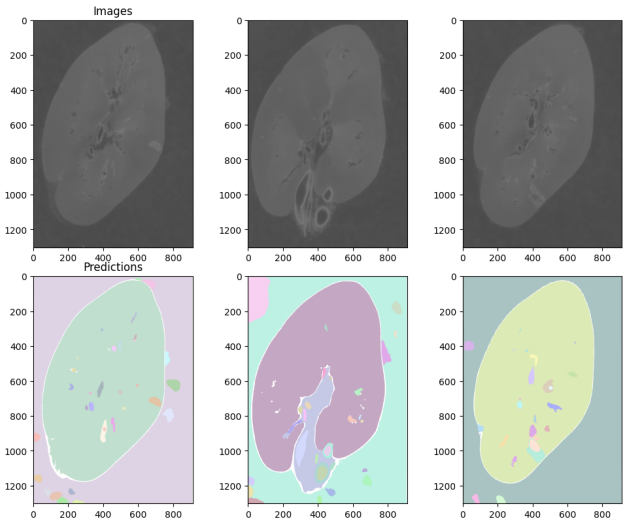

Figure 12. Zero-Shot SAM Results

iron out the many bugs we found out along the process of development. Next time we work on a Machine Learning problem with a large dataset, the goal will be to overfit on a small portion first and then progressively get more general.

6.3. Future Ideas

In the future, we think there is value for multiple industries in exploring the trade-offs between CNNs and Transformers. It would be beneficial to understand the trade-off between accuracy and inference speed for various models such as: Vanilla CNNs, Vanilla Transformers, and Hybrid architectures. Transformers are dominating right now, but it is possible that CNNs inductive biases, such as translation invariance, could be extremely useful within transformer architectures. As a part of this process, the first item on our research lab's agenda would be to increase our compute! Assuming we had no budget, having a cluster of NVIDIA H200s would dramatically decrease the time it takes to train models and implement new ideas.

References

Akram, M. U., Atzaz, A., Aneeque, S. F., and Khan, S. A. Blood vessel enhancement and segmentation using wavelet transform. In *2009 International Conference on Digital Image Processing*, pp. 34–38, 2009. doi: 10.1109/ICDIP.2009.70.

Dosovitskiy and et al. An image is worth 16x16 words: Transformers for image recognition at scale, 2020.

FAIR. Fair sam tutorial, 2023. URL https://github.com/facebookresearch/segment-anything/blob/main/notebooks/

automatic_mask_generator_example.ipynb.

Fraz, M., Remagnino, P., Hoppe, A., Uyyanonvara, B., Rudnicka, A., Owen, C., and Barman, S. Blood vessel segmentation methodologies in retinal images – a survey. *Computer Methods and Programs in Biomedicine*, 108(1):407–433, 2012. ISSN 0169-2607. doi: https://doi.org/10.1016/j.cmpb.2012.03.009. URL https://www.sciencedirect.com/science/article/pii/S0169260712000843.

HuggingFace. Huggingface segformer implementation, 2021. URL https://huggingface.co/docs/transformers/model_doc/segformer.

Kirillov, A. and et al. Segment anything, 2023.

Moccia, S., De Momi, E., El Hadji, S., and Mattos, L. S. Blood vessel segmentation algorithms — review of methods, datasets and evaluation metrics. *Computer Methods and Programs in Biomedicine*, 158:71–91, 2018. ISSN 0169-2607. doi: https://doi.org/10.1016/j.cmpb.2018.02.001. URL https://www.sciencedirect.com/science/article/pii/S0169260717313421.

RNA. Kaggle loss function library, 2021. URL https://www.kaggle.com/code/bigironsphere/loss-function-library-keras-pytorch.

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

SenNet and HOA. Kaggle hacking the human vasculature in 3d, 2023. URL https://www.kaggle.com/competitions/blood-vessel-segmentation/data.

Xie and et al. Segformer: Simple and efficient design for semantic segmentation with transformers, 2021.

Zhu, Y., Qiao, Y., and Yang, X. The optimal connection model for blood vessels segmentation and the mea-net, 2023.

Individual Contribution

Hitarth Gandhi

In this project, my primary contribution revolves around the implementation and execution of UNET training and testing code. This involved the development of a model from the U-Net architecture, the configuration of hyperparameters, and the seamless integration of the model into the training and evaluation pipeline. Additionally, I researched existing methodologies and work done in the area. I wrote the report's Introduction, Related Works, and UNET section.

Andrew Kettle

My focus on this project was on the dataset, Segformer, and SAM. I wrote the entire transformers.ipynb notebook, including the dataset generation and ingestion. I also wrote the custom training loop for training Segformer and the code to decode the outputs and display the results. I tested across different loss functions and differnt learning rates and schedulers. In terms of the project report, I wrote the abstract, Dataset, Technical Approach: Segformer/SAM/Software Libraries, Experiments and Evaluation: Segformer/Zero-Shot SAM, and the Discussion and Conclusion section.