Deep Learning for Visual Computing - Assignment 2 Report

Giovanni Beraldi Martin Miesbauer

May 23, 2024

Contents

1 Introduction

This report describes the implementation and experimentation of deep learning models for image semantic segmentation using OxfordPets and CityScapes datasets. The objective is to understand and compare different models, data processing techniques, and training strategies.

2 Data Processing and Dataset Structure

The Oxford-IIIT Pet Dataset is a comprehensive collection of images of pets, specifically dogs and cats, used for image classification and segmentation tasks. The labels consist of 3 different segmentation classes, i.e background, boundary and foreground. The images have no fixed size, with RGB color channels. As an example, this are 4 random training dataset images and labels:

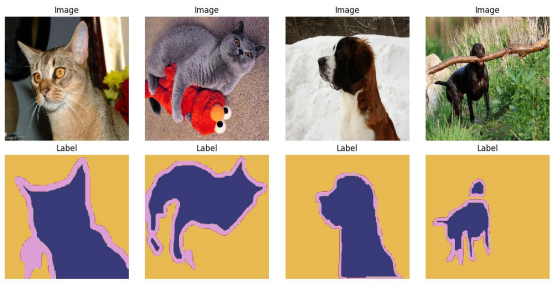

Figure 1: 4 random 0xford-IIT Pet images and labels

The Cityscapes Dataset is a large-scale dataset focused on the semantic understanding of urban street scenes. It has 30 classes, however we used only 19 of them for training and evaluation (the pixels that have the label-value 255 were excluded from the optimisation and metric calculation). As an example, this are 4 random training dataset images and labels:

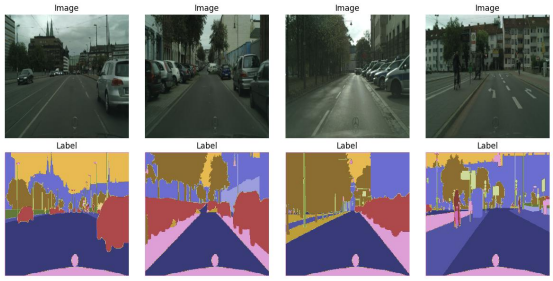

Figure 2: 4 random CityScapes images and labels

Before training, we preprocessed the data, resizing them all to size (64, 64, 3) and transforming them into PyTorch tensors, normalizing pixel values.

3 Loss Functions and Metrics

3.1 Loss Function

The chosen loss function is Cross-Entropy, which measures the divergence between predicted class scores and ground-truth labels. In this context, PyTorch's built-in cross-entropy function was applied to evaluate the model's predictions against the true class labels. In particular, for training on the CityScapes dataset, we ignored pixels with value 255 for the calculation of the loss function.

3.2 Metrics

The evaluation metric used is Mean Intersection over Union (mIoU). A custom `SegMetric` class was implemented to measure the accuracy of semantic segmentation models, using mIoU metric. The `update` method of the class compares predicted segmentation maps with ground-truth labels, updating the intersection and union counts for each class while ignoring specified invalid pixels (for CityScapes dataset). The `mIoU` method computes the IoU for each class and returns the mean IoU across all classes, providing a single performance measure. The class has also a method to reset its internal state.

4 Training Loop and Strategies

The main training loop consists of computing the forward pass for the whole pass at once, then perform the backward propagation. After every 2 epochs, the performance is tested on the validation set. If the performance increased, we save the model weights.

5 Models Implementations

5.1 Fully Convolutional Networks for Semantic Segmentation with a ResNet-50 backbone

Fully Convolutional Networks (FCNs) are designed for semantic segmentation, where the goal is to assign a class label to each pixel in an image. For this project, we employed FCNs with a ResNet-50 backbone from the `torchvision.models.segmentation` module to segment images from the Oxford-IIIT Pet Dataset. To accommodate this, we modified the final layer (prediction head) of the model to output a probability distribution over 3 classes instead of the original number.

5.2 SegFormer: Semantic Segmentation with Transformers

SegFormer is a recent model architecture designed for semantic segmentation tasks. Unlike traditional CNNs used for segmentation, SegFormer employs a transformer-based architecture, inspired by the success of transformers in natural language processing tasks. In SegFormer, the image is divided into patches, similar to how text is tokenized in natural language processing. These patches are then processed by transformer blocks, which enable capturing long-range dependencies and contextual information effectively. Additionally, SegFormer incorporates spatial positional encoding to preserve spatial information during processing. In our project, SegFormer is utilized for segmenting both the Oxford Pets and Cityscapes datasets.

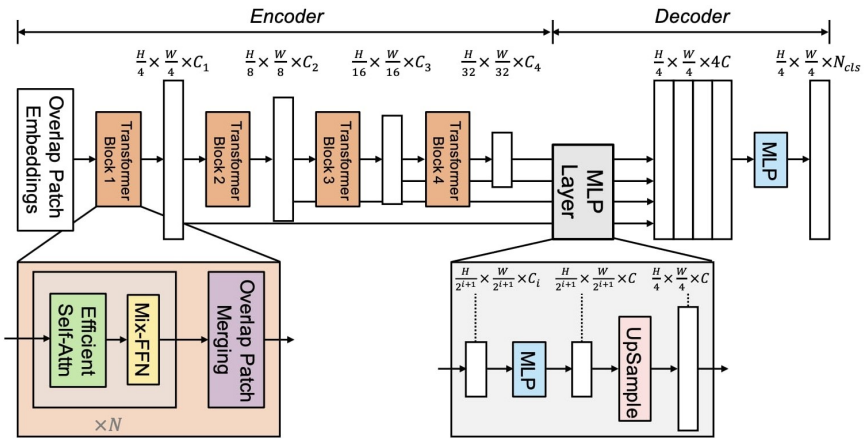

Figure 3: SegFormer model architecture

6 Experimentation and Results

6.1 Training and Validation

We used the following training configuration as a base for all training unless stated otherwise:

- **Epochs**: 30
- **Batch Size**: 64
- **Validation Frequency**: Every 2nd epoch
- **Optimizer**: AdamW
  - Learning Rate: 0.001
  - AMSGrad: True
- **Learning Rate Scheduler**: ExponentialLR
  - Gamma: 0.98

Throughout this process, all training metrics were logged on Weights and Biases for comprehensive tracking and analysis, ensuring a thorough evaluation of the model's performance. The SegMetric class was used for showcasing the evolution of mIoU over the training epochs.

6.1.1 FCN ResNet-50 training

Two training approaches were employed to train the `FCN ResNet50` model on the OxfordPets dataset: training from scratch and fine-tuning with pre-trained weights for the encoder part. The pre-trained weights were utilized solely for the encoder, as the output prediction head required adjustments due to the disparity in the number of output classes between the pre-trained model and the target dataset.

Training from Scratch : Initially, the model was trained from scratch on the OxfordPets dataset. The model was initialized with random weights, and all layers were trained simultaneously. This approach allowed the model to learn features directly from the dataset without any prior knowledge.

Fine-tuning with Pre-trained Weights : Subsequently, the model was fine-tuned using pre-trained weights for the encoder part. The pre-trained weights were obtained from a model trained on a large-scale dataset with a similar architecture. However, the pre-trained weights were used exclusively for the encoder layers, as the output prediction head required customization to accommodate the specific number of output classes in the OxfordPets dataset. In this approach, for an in-depth analysis, pre-trained weights were used both as weights-init and frozen to train only the prediction head part.

The following plots show the metrics recorded during the training.

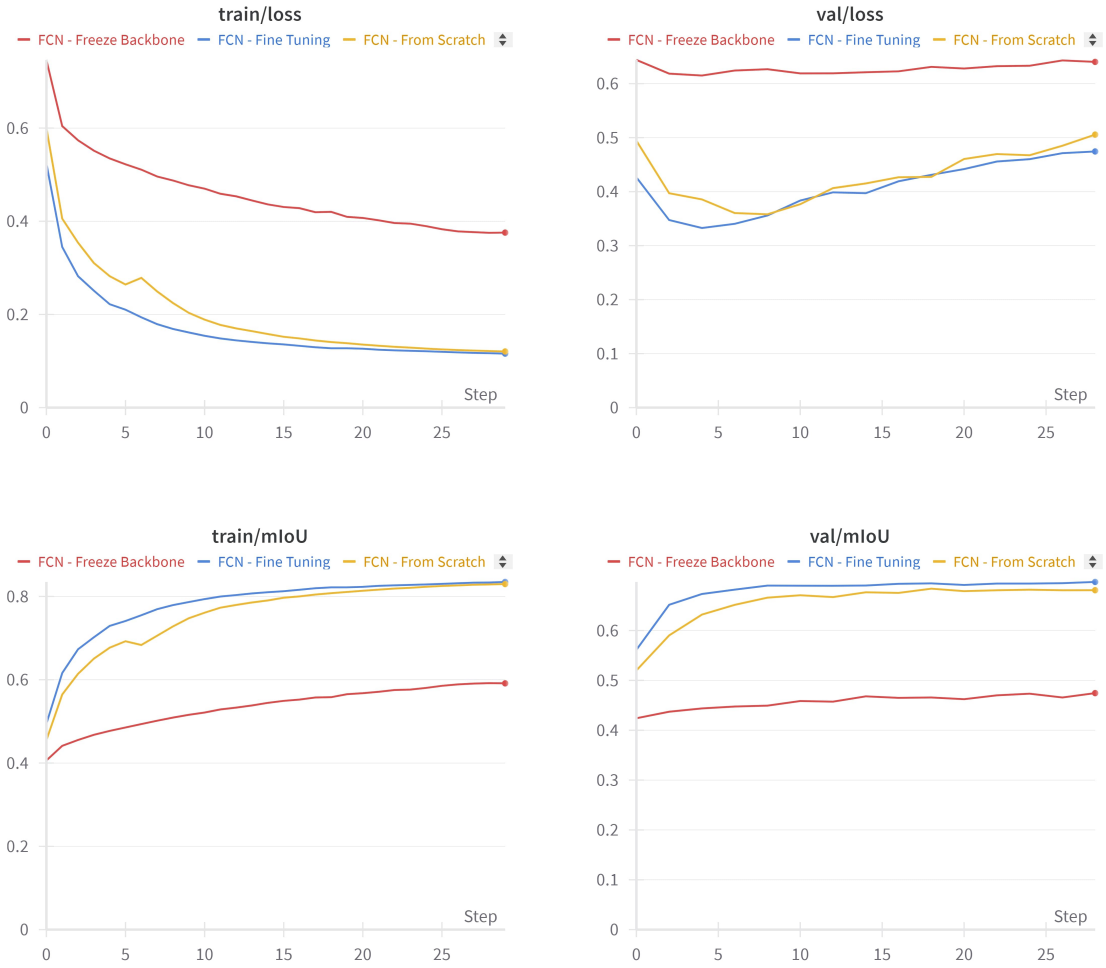

Figure 4: FCN ResNet-50 training metrics

The plots compare three different training strategies: training from scratch, fine-tuning and fine-tuning freezing the encoder. In summary:

- **From Scratch**: Shows steady learning with good performance, but slower initial improvement compared to fine-tuning. This suggests that while training from scratch can achieve good results, it requires more time and resources.
- **Fine Tuning**: Shows rapid learning and good performance, indicating that fine-tuning pre-trained weights is an effective strategy.
- **Freeze Encoder Weights**: Shows slower learning and poorer performance, both in training and validation, due to the limited capacity of the network to adapt.

These plots indicate that fine-tuning the pre-trained weights provides the best performance in terms of both loss reduction and mIoU improvement.

6.1.2 SegFormer Training

In this section, we outline the training process of the SegFormer model on the OxfordPets dataset. Two training strategies were employed: training from scratch and fine-tuning with pre-trained weights.

Training From Scatch : Initially, the SegFormer model was trained from scratch directly on the OxfordPets dataset. This approach allowed the model to learn features and representations specific to the target dataset without any prior knowledge.

Fine-tuning on OxfordPets : Subsequently, the SegFormer model was pre-trained using the Cityscapes dataset and then fine-tuned for the OxfordPets dataset. For pre-training on Cityscapes, pixels with a label of 255 were excluded from loss and metric calculations. During fine-tuning for OxfordPets, the pre-trained weights were used solely for the encoder, while the decoder was always trained from scratch due to the differing number of classes between the datasets. Two fine-tuning strategies were implemented:

- **Fine-tuning Encoder**: The pre-trained encoder weights were used as weight initialization, and the entire model was trained on the OxfordPets dataset.
- **Freezing Encoder**: The pre-trained encoder was kept frozen, and only the decoder was trained on the OxfordPets dataset.

Different learning rates were applied during fine-tuning, including the original learning rate (0.001) and half of it, to explore the impact on performance.

The following plots show the metrics recorded during the training, showcasing the performance of the SegFormer model trained from scratch and with the two fine-tuning options.

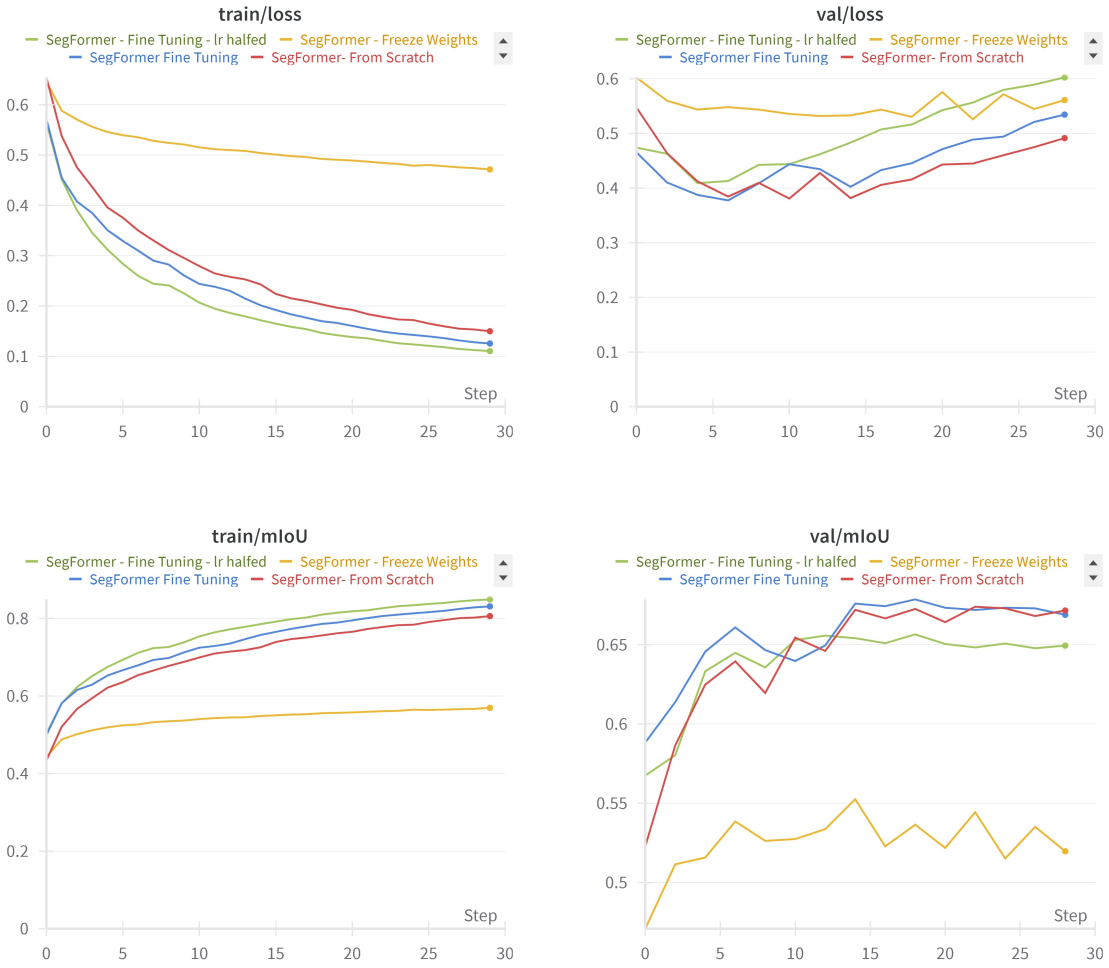

Figure 5: SegFormer training metrics

The pots compare four different training strategies: training from scratch, fine-tuning, fine-tuning with learning rate halved and freezing encoder weights. In summary:

- **From Scratch**: Shows steady learning with good performance, but slower initial improvement compared to fine-tuning. Indicates that while training from scratch can achieve good results, it requires more time and resources.
- **Fine Tuning**: Shows rapid learning and good performance but with some instability in the validation loss and mIoU.
- **Fine Tuning - lr halved**: Shows rapid learning and performance similar to the regular fine-tuning, suggesting that halving the learning rate does not significantly impact the training dynamics.
- **Freeze Encoder Weights**: Shows slower learning and poorer performance in both training and validation, similar to the freezing strategy for the previous model.

These plots indicate that fine-tuning the SegFormer model, both with and without halving the learning rate, provides the best performance in terms of both loss reduction and mIoU improvement, though some instability is observed.

6.2 Test Results

Following the training phase, we evaluated the best FCN ResNet-50 and SegFormer models on the OxfordPets test set to assess their performance. In particular, we tested the two models subjected to a simple fine-tuning (in blue in the respective plots). We generated confusion matrices and ROC curves for each model to visualize the accuracy of predictions and identify any misclassifications. Additionally, we also plotted the prediction masks of the two models for four images of the training set. These evaluations provided crucial insights into the models' effectiveness and helped compare the various models.

6.2.1 FCN ResNet-50 test results

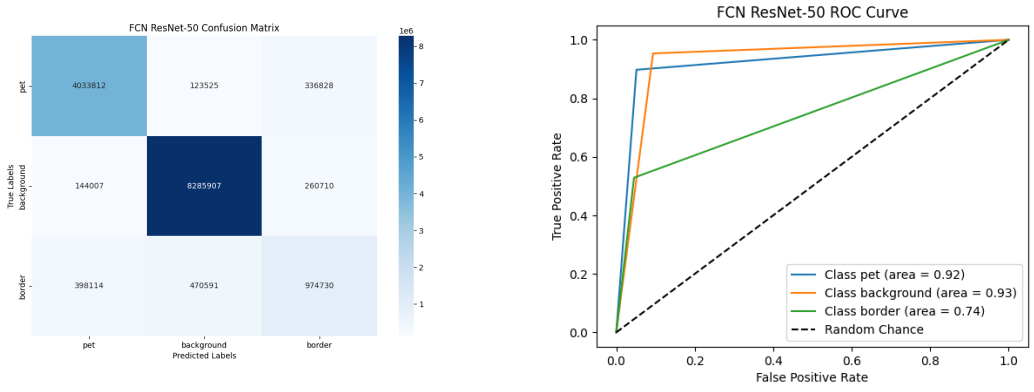

Figure 6: FCN ResNet-50 test results

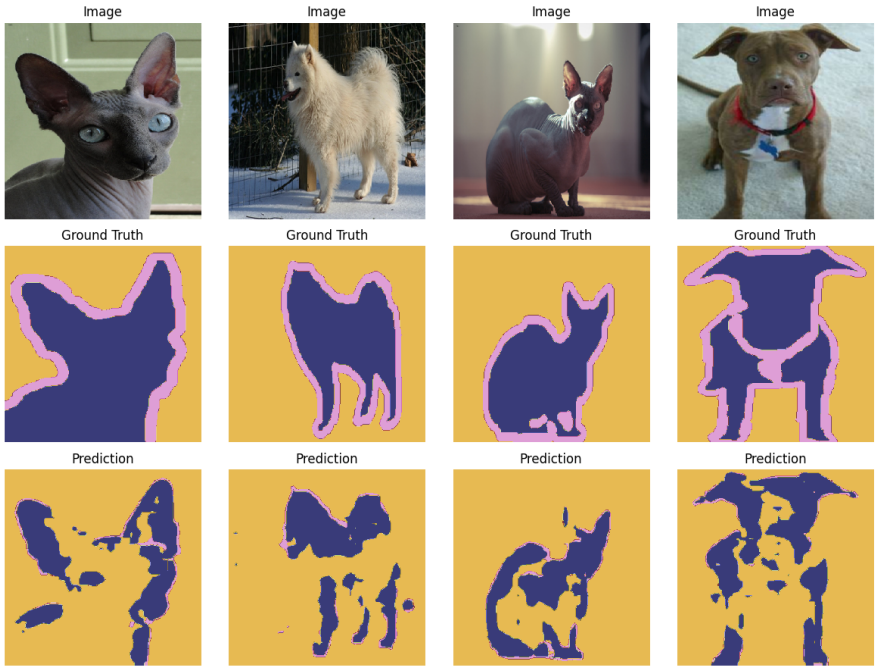

Figure 7: FCN ResNet-50 predicted mask

From the confusion matrix, ROC curve and prediction masks, it can be seen that the model has greater difficulty in correctly identifying edges, while it performs better in pet segmentation, especially background (probably because it is the most frequent class of all). In particular, it can be seen that the model confuses itself by often classifying the pet class as background and the border class as pet.

6.2.2 SegFormer test results

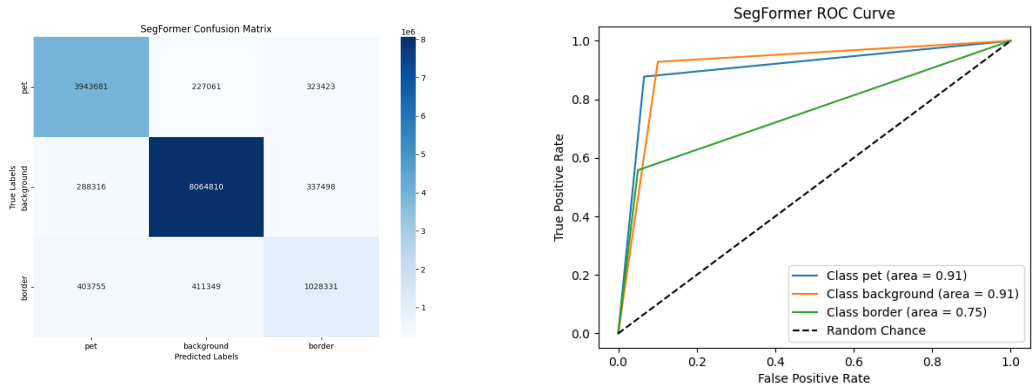

Figure 8: SegFormer test results

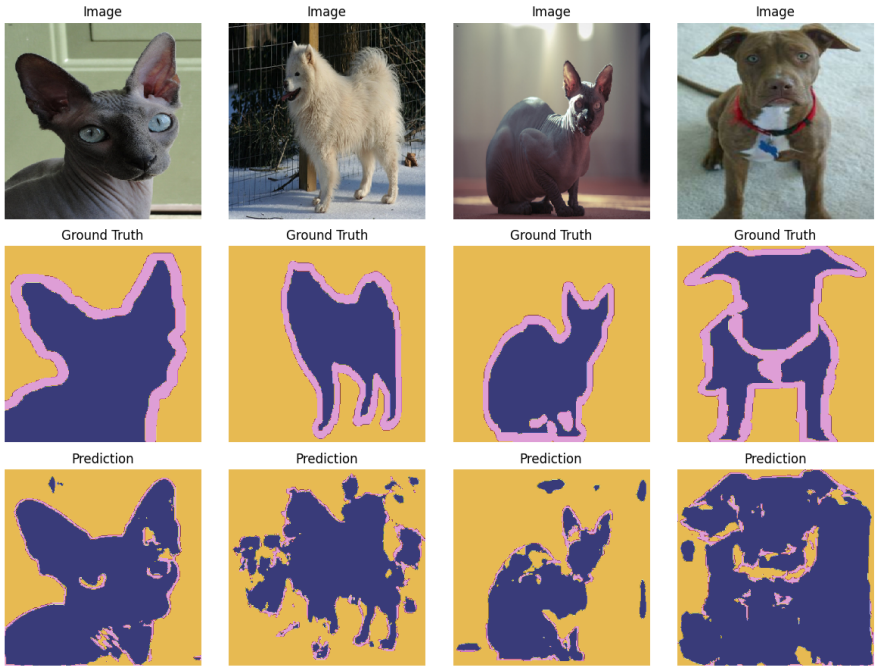

Figure 9: SegFormer predicted mask

Again, the performance and errors are similar to the previous case, but it can be seen that the SegFormer model incorrectly classifies the background class as pet more often than the previous model.

6.3 Comparison of Best Models

The followind plots compare the training metrics of the SegFormer and FCN ResNet-50 models, both using the fine-tuning strategy.

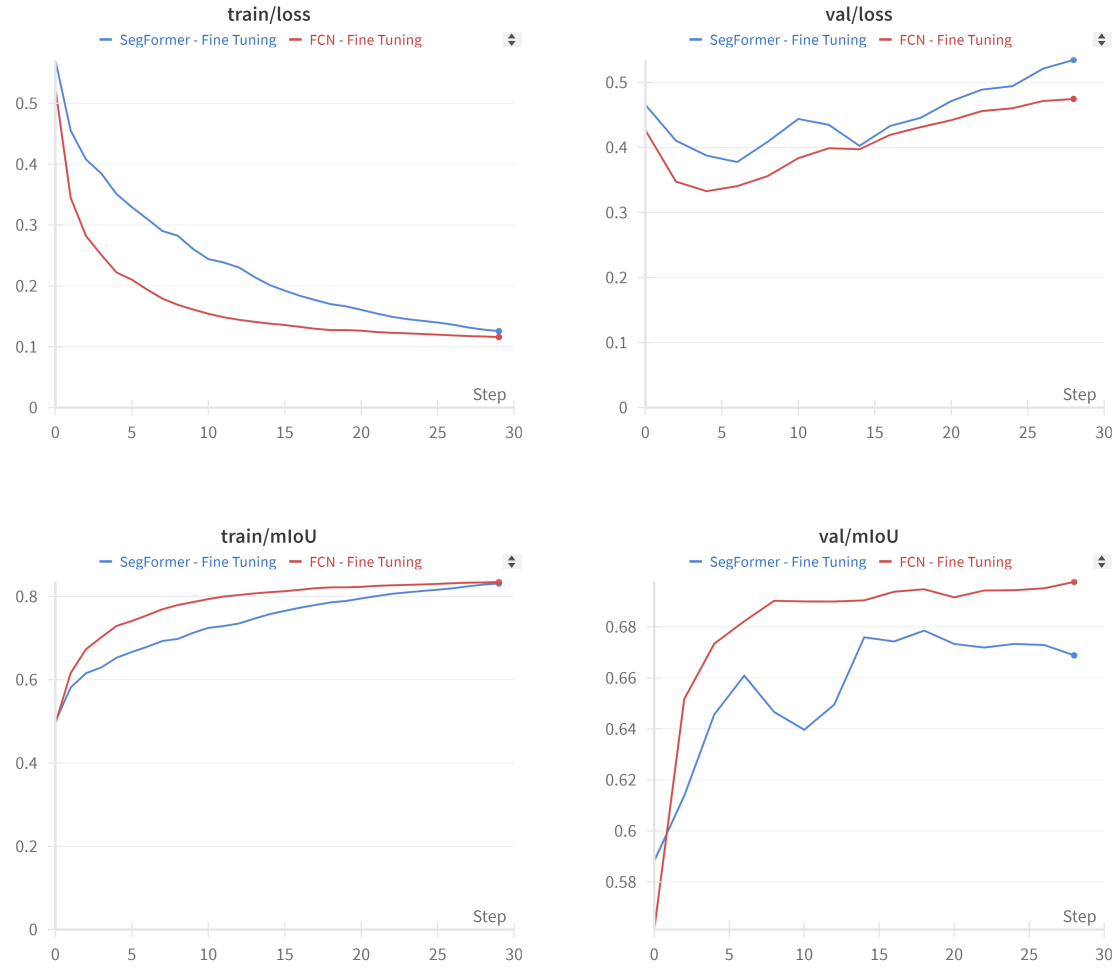

Figure 10: Best Models Comparison

In summary:

- **SegFormer - Fine Tuning**: Shows steady and consistent learning with effective training (as indicated by train/loss and train/mIoU). However, there is some instability in the validation metrics (val/loss and val/mIoU), suggesting potential overfitting and variability in generalization performance.
- **FCN ResNet-50 - Fine Tuning**: Demonstrates faster initial learning and convergence (as seen in train/loss and train/mIoU). The validation metrics (val/loss and val/mIoU) indicate better initial generalization with less fluctuation compared to SegFormer, though there is still a potential overfitting trend.

Overall, FCN ResNet-50 appears to provide faster initial learning and more stable validation performance, while SegFormer fine-tuning shows consistent training but with some instability in validation performance.

7 Discussion and Conclusions

We compared the performance of two different semantic segmentation models, FCN ResNet-50 and SegFormer, using various training strategies. The analysis was conducted using training and validation loss, as well as mean Intersection over Union (mIoU) metrics. In conclusion, the fine-tuning strategy is the most effective for both FCN ResNet-50 and SegFormer models, providing the

best balance between rapid learning and high performance. FCN ResNet-50 fine-tuning offers faster convergence and more stable validation performance, making it suitable for applications requiring quick training and robust generalization. On the other hand, SegFormer fine-tuning demonstrates consistent learning and high training performance, but with some instability in validation metrics, which may be addressed with further hyperparameter tuning or regularization techniques.

8 Addressing Key Questions

8.1 Architectures for segmentation and how they differ from the SegFormer

In the lecture four main architectures for segmentation were presented. The first approach is a sliding window across the image where for each pixel a neighborhood gets used as input in a CNN to classify the pixel's class. Here we do not use shared features between overlapping pixels and might lose some information due to the limited receptive field. Another architecture is the Fully Convolutional Network. Here, the network consists of a bunch of convolutional layers. At first the images get downsampled to a lower resolution and afterward get upsampled to the original resolution. There is a risk of information loss due to the encoding and decoding, but each pixel has a full receptive field. The U-Net utilizes skip connections between encoding and decoding stages to enable more information to pass through the whole network. The SegFormer uses transformers to update the pixel embeddings of each patch depending on the other pixel embeddings in the same patch. Afterward, a decoder gets applied to every pixel embedding to get the class predictions. This decoder is lightweight and can be trained very easily.

8.2 Purpose for down- and up-sampling

The previously mentioned architectures of Fully Convolutional Network and U-Net utilize down- and up-sampling to enable each pixel in the final classification layer to get a larger receptive field with fewer layers than without downsampling. In addition, downsampling drastically reduces the amount of computing power and storage required increasing prediction and training performance. To get an output of the same size as the original images, upsampling has to be applied afterward.

8.3 Purpose of pre-training and fine-tuning

For many tasks, there are not many training images available. This might be caused by a high costs or difficulty for gaining these images. Generating CT images has a high cost and health risks for the subject. To combat this, models or parts of models are pretrained on large available datasets which are more readily available. This pretraining can consist of feature detection, which is similar for most datasets. This pretrained layers can be used to transform the training images, and a simple MLP for classification can be trained on the output. This saves computing time and therefore costs and prevents overfitting on the training data.

8.4 How and why using pre-trained weights influenced the performance and differences between the fine-tuning option

We noticed that using pre-trained weights increased the performance in the first few epochs and therefore led to acceptable results sooner. Freezing the weights in the backbone turned out to be not a good idea, as this affected the metric negatively. More details can be found in the previous sections of the report.