Cryo-EM Image Segmentation Report

Phase 1: Manual Implementation

Approach

As a disclaimer, I was not very consistent in keeping track of all of my experiments which is why my table might be more concise than the description for my approach. This is also because I had lots of failed attempts which straight out pointed to bad performance and at the time of experimenting i did not consider worthwhile recording. Nevertheless I've learnt that it's better to maintain an organised approach and record what works and what doesn't. I still think it's worthwhile going through some of the experiments I tried and remember did not work.

My first step in this assignment was understanding the data. During my initial data exploration I visualised some of the images with their corresponding masks, which revealed a significant class imbalance given that most of the image pixels belonged to the clean category (0). To quantify this I computed mask coverage statistics which revealed the Mean contamination coverage of each image was 3.3% contained contaminants (according to my computations)

In terms of data handling i seeked out chatGPT and claude for possible data augmentations. They provided me with solutions like Clahe, an Albumentations library and Elastic transformations, brightness and blur transformations among others which were claimed to be efficient for micrograph. However in actual practice, and with the help of managing to visualise images from my data loader i soon realised that due to my images being very simple, just greyscale, some aggressive transformations where just distorting the entire image and feeding the model with nothing but noise. I decided then to limit my data handling to less aggressive ones like random crops, flips and rotations. A next step for the future would be to increase this data augmentation but after failed attempts i decided to accommodate with that. I did manage to compute a global std and mean which I noticed really helped rather than image wise normalisation.

The first thing I attempted for this first phase of the competition was keeping the code provided by the teacher and implementing a CNN but changing the learning rate to ReduceOnPlateau, increasing the number of epochs and investing lots of time in researching possible transformations for the data. I was not very successful in this. I soon dropped the CNN approach and went for a U NET which as I researched and we learned in class is one of the best general-purpose segmentation architectures. This improved my performance slightly but still required work. It was at this moment also that I implemented the improved data handling I talked about above. Moreover I also decided to add a more convolutional layer to my model as well as combining BCE Loss with Dice loss which provided a better approach as the data showed an imbalance according to my findings. Within this step I actually played with many more things such as different optimisers, window sizes for the crops of the training and dropouts or batch normalisation layers for regularisation. Something else that really helped me was implementing attention gates and in my last model even multi-scale attention gates.

After various trials my most successful architecture was a U- Net base in which I incorporated multi-scale attention in order to refine skip connections before concatenation in

the decoder. I used a double convolution with BatchNorm (for regularisation) and ReLU, max pooling in the encoder, and transposed convolutions for upsampling. The final layer had a 1x1 convolution with sigmoid for binary segmentation.

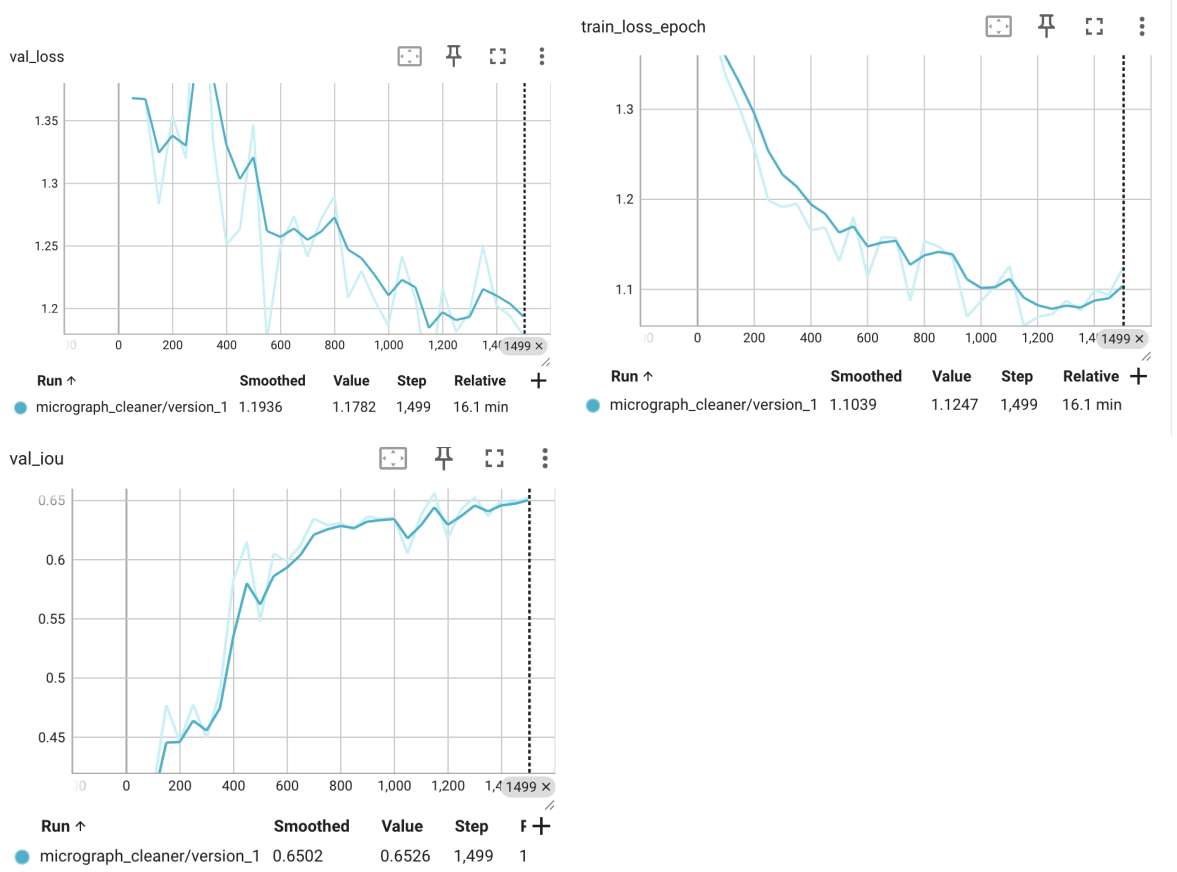
During my experiments, given the imbalance nature of my data I actually tried using various loss functions. Combined BCE and Dice loss was my standard for my initial trials. I maintained this because I wanted to be able to compare changes in val loss and when changing architectures and different training parameters. I soon realised I could also log val_iou as a basis for comparisson. Due to the class imbalance I found in my data exploration I tried implementing different loose to my best architecture. Tversky loss which was supposed to work very well with imbalance data according to my research yielded very inconsistent research. I could've researched further these inconsistencies but instead I decided to create a custom loss function which accounted for the necessities of Cyro micrograph images. For this i combined binary cross-entropy, Dice loss, and a focal loss component to specifically handle class imbalance. Within this function I also penalised confident false predictions using a tunable contamination ratio which was very useful given the sparsity of the positive (contaminated) class. In my loss function I also clamped logits and probabilities which improved numerical stability and prevented log(0) errors. The implementation of this loss required research and lots of trials for adjustment but overall resulted to be very successful.

Experiments

Given that producing predictions for submission was really time consuming I did not submit and have scores for all my experiments.

Experiment	Description	Training Loss	Validation Loss	Public Score	Private Score
Baseline	**Simple CNN**	**BCE Loss**		**NA**	**NA**
Exp 1	**U Net**	**BCE LOSS**	**Val Loss: 0.6799**	**0.0654**	**0.0617**
Exp 3	**U Net + Double Conv**	**Dice + BCE**	**Did not record	**0.116**	**0.21**
Exp 4	**U Net with Attention Gate**	**Dice + BCE**	**val_loss=0.228 val_iou=0.589,**	**0.26**	**0.359**
Exp 5	**U Net Attention Gate**	**Tversky Loss**	Val_loss = 0.8	***Did not** upload because performance was really bad	
Exp 6	**U Net with Multi scale Attention Gate**	**BCE + Dice Loss**	**val_loss=1.19 val_iou=0.696**	**0.3430**	**0.32649**

Training Curves For best submission:
Loss is custom using Focal Loss.



Analysis
Analysing my experiments and looking back, in terms of data handling what definitely worked was visualising images and checking for results. This helped me not implement excessively aggressive augmentations. Moreover, enhancing the U Net architecture with self attention was really useful and the customised Cryo loss adapted for the class imbalance and contamination rate I think was one of my best implementations.

Phase 2: Open Resources

Approach

When given the chance to implement anything I researched what were the best models in segmenting micrograph images. I found this pretrained model https://github.com/jianlin-cheng/CryoSegNet which gave me high hopes as it allowed me to download the checkpoints and just fine tune it (i had to remove the SAM layer from it because it was very memory intensive for my available resources. Despite being able to implement it the epochs started at a very low BCE + Dice loss - around 0.01 which was a strong indicator that my training was not fine tuning but rather starting at random. I attempted to solve that in various ways but was not able to. The val loss my model reached was still very high and a very low val_iou.

The next path I decided to explore was experimenting with SAM from hugging face but I never managed to actually run this, even after trying to optimise the training. I kept having the error saying my storage was not enough to host the model. I tried to do this with colab and also through kaggle notebooks but was still unsuccessful.

Still persistent in being able to fine-tune a model, I decided to look into the models available from the library segmentation_models.pytorch. In my experiments i included DeepLabV3 with a ResNet34 backbone, and Attention-Enhanced U-Net++ using EfficientNet-B5 with SCSE blocks. I experimented with adjusting training strategies (such as learning rate optimisers and mixed precision) to improve stability and performance during the finetuning. Such included defreezing layers gradually rather than all at once after some certain epochs. After several trials with different of them (i did not log all of trials) the attention enhanced model provided the best performance.

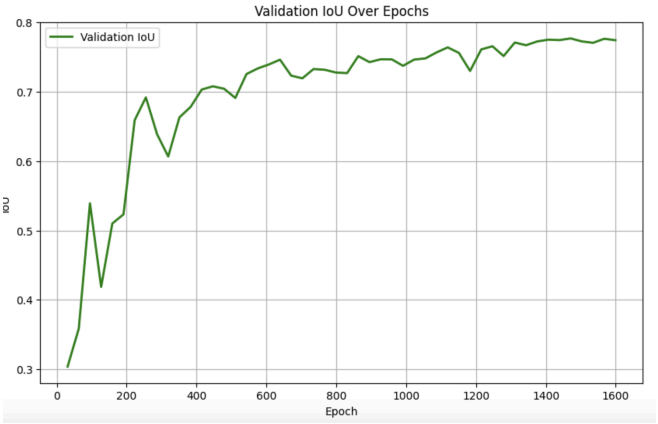
Despite testing performance of different architectures with a combined BCE & Dice loss I ended up implementing my CryroLoss for the last experiment. This yielded an even better improvement: my loss actually dropped to 0.8 (in terms of CryoLoss) which is only comparable to the previous 1.19. In terms of val_iou which I also tracked it yielded as a very high 0.775 (78% overlap) which in the private score was equivalent to a 0.48.

Results

Experiment	Description	Training Loss	Validation Loss	Private Score
Exp 1	**CyroSegNet**	**BCE + Dice Loss**	Fine Tuning started too low and did not improve, did not record final val_loss	Did not upload because it wa very bad
Exp 3	**Deeplabv3 with resnet34**	**BCE + Dice Loss**	**0.88**	Did not upload because it wa very bad
Exp 4	**Attention-Enhanced U-Net++ with EfficientNet-B5 and SCSE Blocks**	**BCE + Dice Loss**	**0.26**	**0.309**
Exp 5	**Attention-Enhanced U-Net++ with EfficientNet-B5 and SCSE Blocks -** tried adjusting training for more stable learning	**BCE + Dice Loss**	**0.28**	**0.3032**

Exp 6	**Attention-Enhanced U-Net++ with EfficientNet-B5 and SCSE Blocks -**	**Cyro Loss**	**0.8 Val_iou = 0.704**	**0.48**

Public Score was not available and so it is not included in this table.



Val_iou tracking for my last fine tuned model.

Comparison

Just by looking at the improvement in val_iou which managed to reach 0.78 in my best attempt I think I improved a lot from one phase to another. Moreover, although not the best metric for comparison as it only includes 30% of the data, my public score increased from 0.34 to 0.48 in this phase. I think the possibility of implementing pretrained models and predefined architectures and fine tuning them appropriately (which involved lots of trials and errors) was really helpful in the competition as manual implementation had a lot of limitations.

Conclusions

I learnt a lot during this project. I have to admit for me it was incredibly challenging but it was a great opportunity to learn everything learnt in class and I do believe I got much more comfortable with the coding of the models. I learnt the importance of transformations and implementing them properly as well as how to improve architectures and how well self attention can work. Moreover I learnt the importance of an appropriate loss function for your specific use case as I do believe the custom cryo loss leveraged my standing in this competition. In the second phase I learnt stabilising learning and freezing gradually worked really effectively. As future steps I would explore further some of the paths I was unsuccessful with as I do feel like a SAM and the Cryosegnet implementation could've worked much better than my resulting model but I feel I did not have the resources or time to do so.