Right Ventricle Segmentation Challenge

Rohan Gupta, Shubham Seth, Yash Sanjeev

November 2020

Contents

1 Introduction

The following report contains our detailed observations after running various Experiments a U-Net architecture as reported in this paper.
The report goes through our attempts at trying various learning rate optimizers and schedulers, regularization techniques and different pre post-processing techniques.
The baseline model we use to compare these various processes is same as the U-Net architecture given in the paper, and uses an Adam optimizer with Reduce Learning Rate on Plateau scheduling scheme and an initial learning rate of 3e-3, trained on 80% of the training data for 30 epochs.
The metric we are going to use to compare the different models is their Average Dice Score on the test set.

2 Learning Rate Schedulers

2.1 LR Schedulers with different Optimizers

Optimizer	test score
Adam	**82.38**
Adagrad	66.18
RMSProp	73.95
SGD	36.59

Table 1: Different Optimizers with ReduceLROnPlateau Scheduler

In the following figures, pink signifies Adam, Gray signifies Adagrad, Green Signifies RMSProp and Orange signifies Stochastic Gradient Descent.

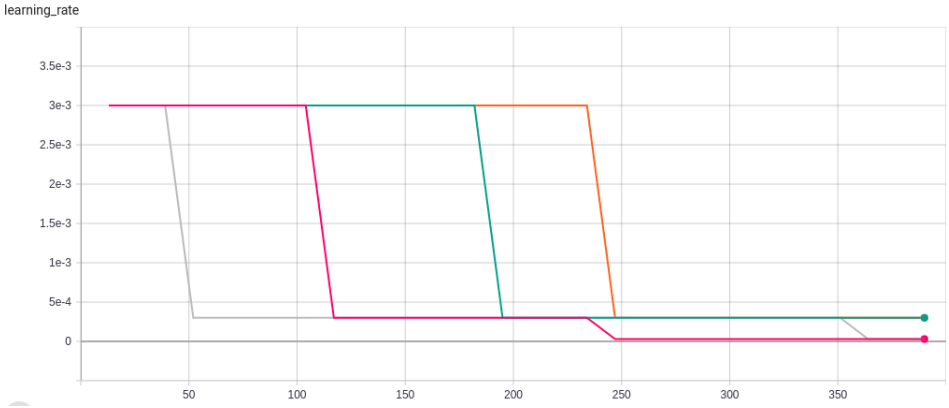

Figure 1: Learning Rate with Different Optimizers

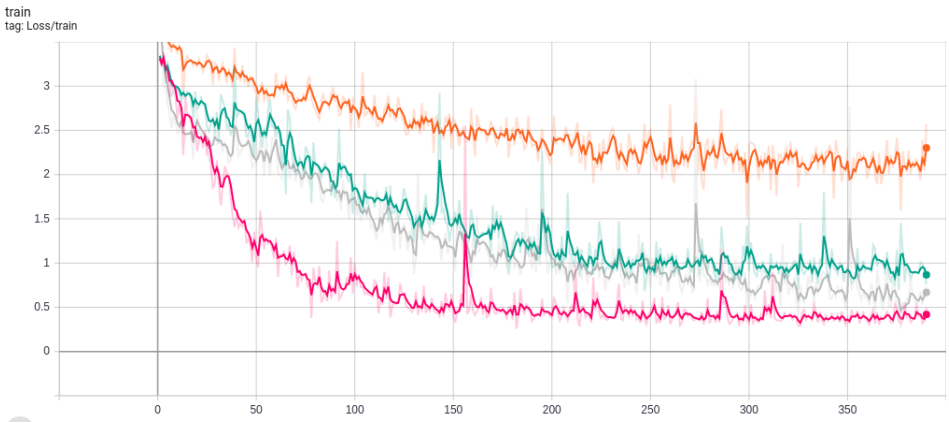

Figure 2: The Training Error with Different Optimizers

2.2 LR Schedulers with different Training-Validation Split

Train-Val Split	test score
0.2	24.12
0.4	23.19
0.6	81.11
0.8	**82.38**
1	56.88

Table 2: Different Split Values with ReduceLROnPlateau Scheduler

2.3 Different Learning Rate Schedulers

Here, pink is ReduceLROnPlateau, Green is StepLR, Gray is ExponentialLR and Orange is CosineLR

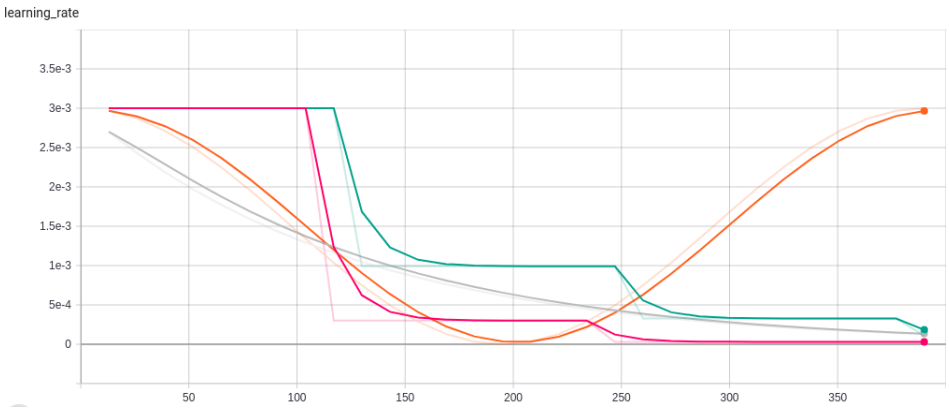

Figure 3: Comparison of different Learning Rates

2.3.1 ReduceLROnPlateau

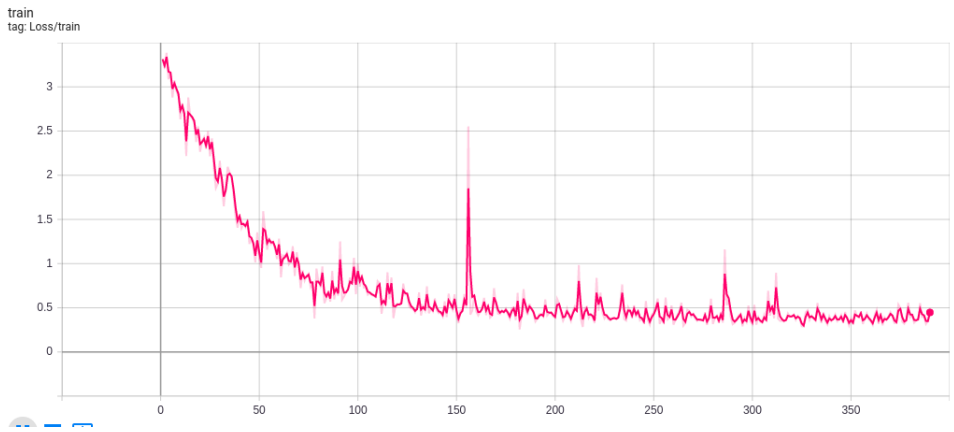

Figure 4: Training with ReduceLR

Average Dice Score = 82.38

2.3.2 StepLR

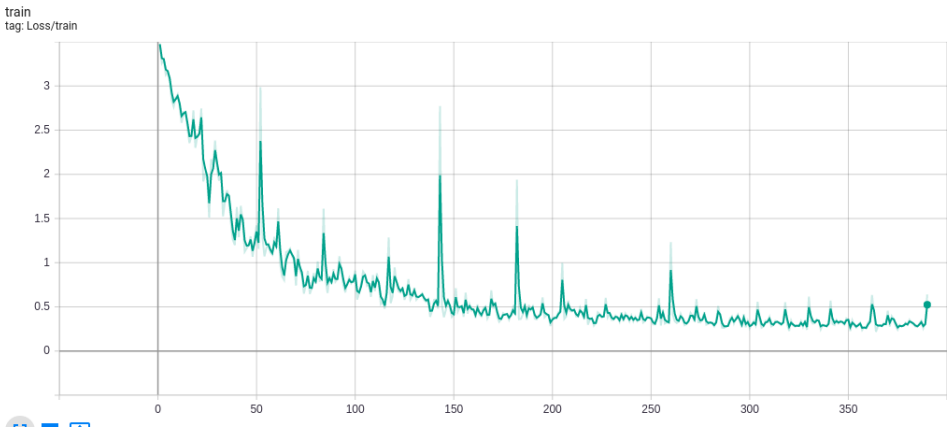

Figure 5: Training with StepLR

The step Learning Rate Function decayed the learning rate to a third of its value every 10 epochs.
Average Dice Score = 82.93

2.3.3 ExponentialLR

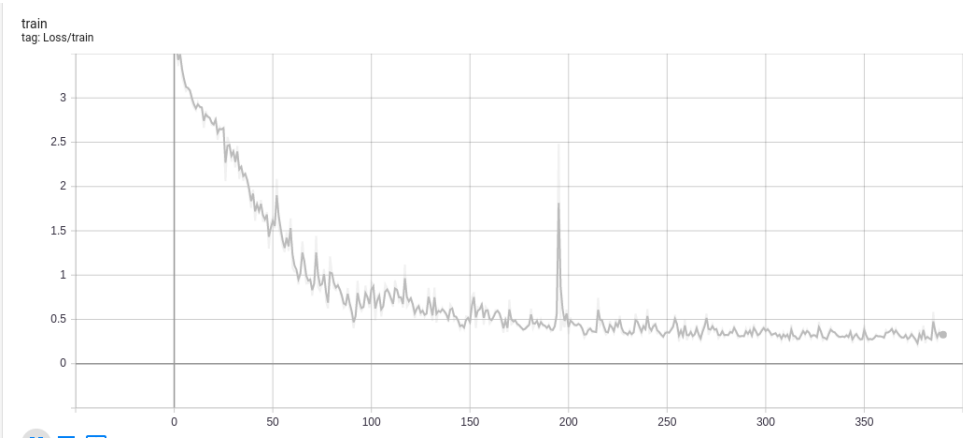

Figure 6: Training with ExponentialLR

The step Learning Rate Function decayed the learning rate to 0.9 times its previous value after every epoch.
Average Dice Score = 86.37

2.3.4 CosineAnnealingLR

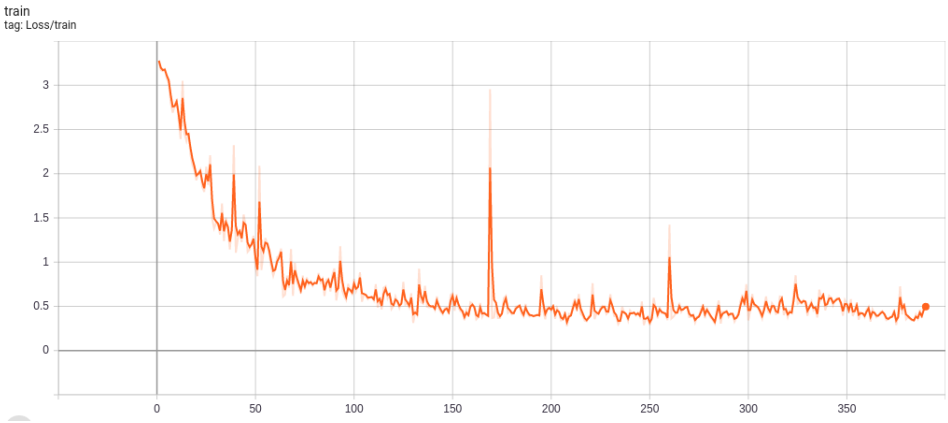

Figure 7: Training with CosineLR

Average Dice Score = 82.99

3 Optimizers

In these experiments, we remove the learning rate scheduling to see how different optimizers have different optimal Learning Rates, and how they behave at the same learning rate.

3.1 Different Optimizers with different LR

Optimizer	Tuned Learning Rate	test score
Adam	**0.001**	**80.41**
Adagrad	0.003	74.34
RMSProp	0.001	78.43
SGD	0.01	60.31

Table 3: Fine-Tuned Learning Rates for Different Optimizers

For the following figures, Pink: Adam, Green: Adagrad, Gray: RMSProp, Blue: SGD

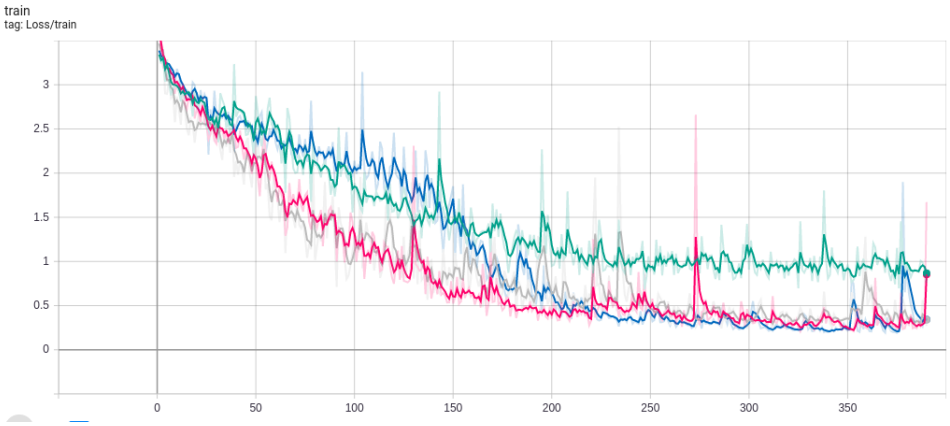

Figure 8: Training Loss Convergence

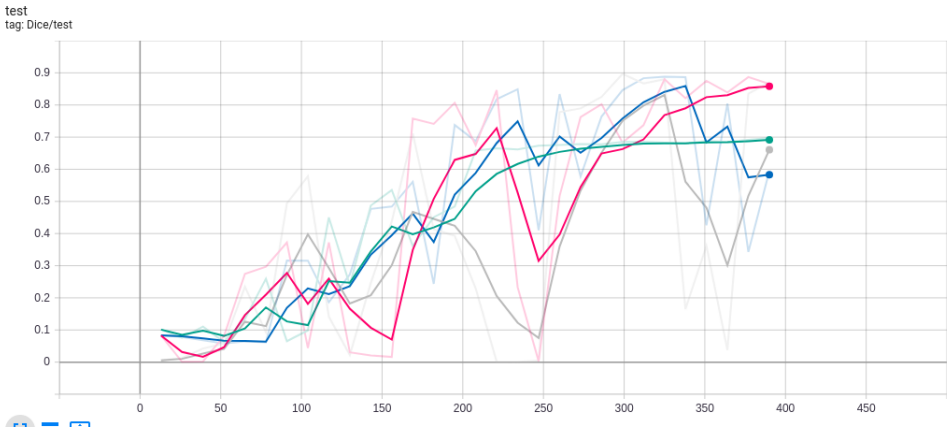

Figure 9: Validation Dice Score

3.2 Different Optimizers with same LR

Optimizer	Learning Rate	test score
Adam	**0.001**	**80.41**
Adagrad	0.001	42.69
RMSProp	0.001	78.43
SGD	0.001	25.64

Table 4: Same Learning Rate for Different Optimizers

Adam: Blue, Adagrad: Green, RMSProp: Gray, SGD: Orange

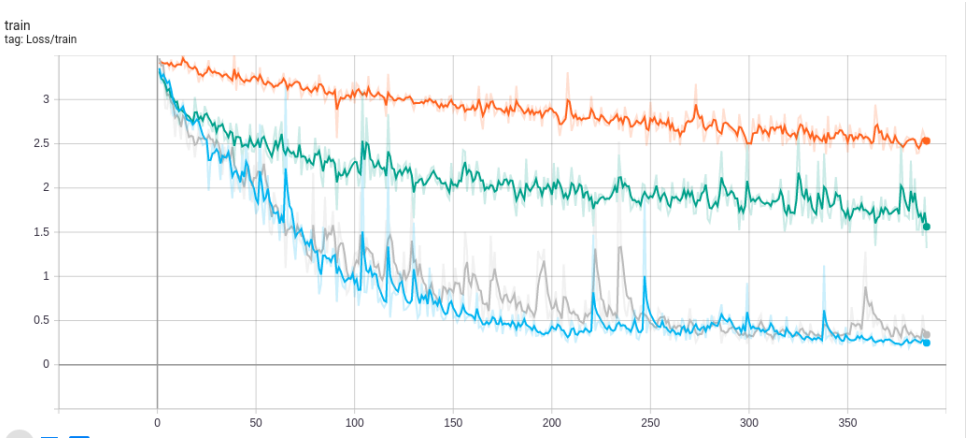

Figure 10: Convergence of Loss for same LR

3.3 Adam Optimizer with different Train-Val split

Train-Val Split	test score
0.2	11.07
0.4	57.49
0.6	63.29
0.8	**80.41**
1	78.70

Table 5: Adam Optimizer at Different Train-Val split

4 Regularization Techniques

This section will summarise the regularization techniques that were analysed and tuned by the group. The models were trained for 30 epochs, with Adam optimiserand an initial learning rate of 0.001. The learning rate was decreased 0.1 times if the loss function started saturating. The loss function used was (1 - Dice coefficient) + (Binary cross entropy loss).

The following techniques were analysed:

4.1 L2 Regularization

The weight decay was tuned for 0.2, 0.4, 0,6, 0.8 of the training dataset and the results are shown in Table 6.

train-set fraction	lambda	val score	test score
0.2	1e-03	36.76	36.75
0.4	1e-05	74.6	70.723
0.6	1e-05	82.8	80.9727
0.8	**1e-05**	**86.09**	**80.27**

Table 6: Tuned hyperparameter results for L2 regularization

4.2 L1 Regularization

The results obtained on tuning the regularization hyperparameter for various fractions of the training dataset is shown in Table 7.

train-set fraction	lambda	val score	test score
0.2	1e-05	16.9	16.9
0.4	1e-03	71.2	67.49
0.6	0.1	83.7	81.99
0.8	**1e-02**	**84.4**	**83.18**

Table 7: Tuned hyperparameter results for L1 regularization

4.3 Dropout

The same keep probability is applied to all the convolutional layers. The results for various train-val splits are shown in Table 8. We can clearly see the improvement for less training data by comparing with Table 5. As the dataset size increases, however, the dropout does not seem to give much difference. In fact, higher dropout rates inhibits the model to perform well. So *80% of the data is enough if we do not want to use dropout.*

train-set fraction	keep probability	val score	test score
0.2	0.1	23.927	23.79
0.4	0.2	72.6	72.36
0.6	0.1	84.27	81.68
0.8	**0.1**	**85.14**	**82.93**

Table 8: Tuned global keep probabilities for fractions of dataset

Different keep probabilities- where does dropout matter the most?

Two runs were made. In the first run, the keep probability was increased for deeper layers and were kept less for shallower layers(symmetrically from both sides). So, the first and last layers would have lower probability compared to the middle layers.

In the next run, keep probability was decreased for deeper layers and were kept more for shallower layers(symmetrically from both sides). So, the first and last layers would have higher keep probabilities compared to the mid-layers.

Run	val score	test score
1	**86.36**	**84.59**
2	84.81	77.60

Table 9: Summary of different regularizers

The results obtained are shown in 9. Run 1 clearly performs better than Run 2. Hence, we can say that dropout in deeper layers(with more specialised features) are more meaningful than in shallower layers(with low level features).

4.4 A comparison of different regularizers

A summary of various regularizers analysed is provided in Table 10. The dataset used was 80% of the RSVC training dataset.

A combination of L1 and L2 regularization seems to perform the best on the validation set. The coefficients used were 1e-5 for L2 and 1e-2 for L1 losses respectively. Dropout outperforms it on the test dataset. The keep probabilites used were 0.0 for all layers except the middle most layers in the U-net architecture(0.2 was used for the last downsampling and the first upsampling conv. layers). Note: The dropout result was averaged over two runs.

Regularizer	val score	test score
Dropout	87.25	**84.10**
L1	84.4	83.1
None	83.97	83.62
L1 + L2	**89.69**	83.69

Table 10: Summary of different regularizers

5 Pre and Post-Processing Approaches

5.1 Pre-Processing Techniques

Here are a few approaches we came up with:

- **CLAHE**: Contrast Limited Adaptive Histogram Equalization is a very successful contrast enhancement technique that can be used to improve the contrast quality of the input images. As seen from the sample result below, CLAHE can be used to enhance details of the image.

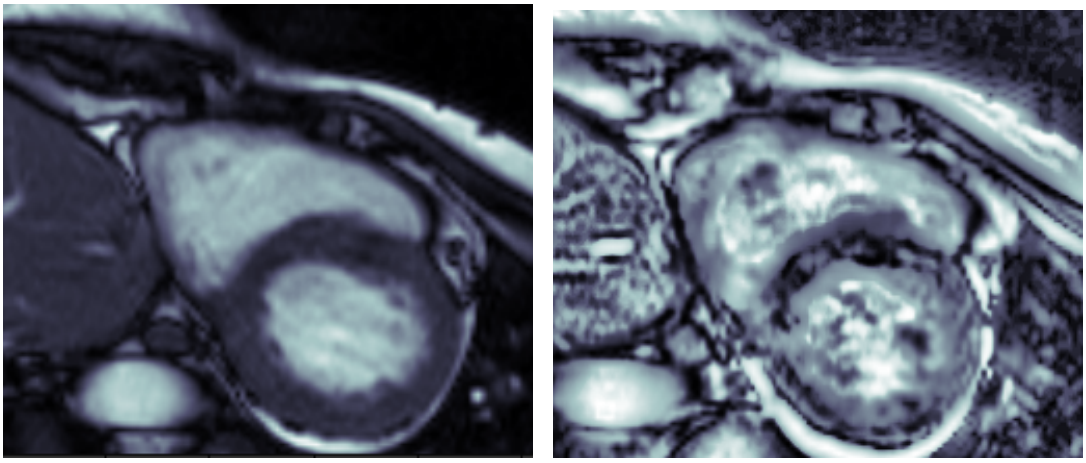

Figure 11: Effect of CLAHE on sample input image

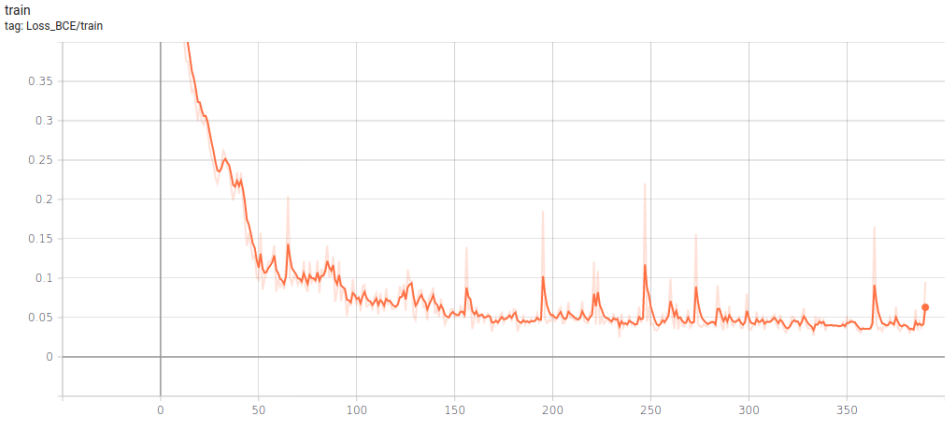

Figure 12: Training with CLAHE and Adam Optimizer

CLAHE sometimes also enhances the useless regions in the process. Some images received great enhancement and others suffered accuracy loss. The result was too varied to conclude anything, although we believe that the results could have been improved with appropriate hyperparameter tuning for CLAHE, which we could not do due to lack of time. Still, most of the optimizers did benefit from CLAHE preprocessing.

Optimizer	CLAHE score	Without CLAHE
Adam	81.09	**82.38**
Adagrad	**68.23**	66.18
RMSProp	73.12	**73.95**
SGD	**41.34**	36.59

Table 11: Different Optimizers with CLAHE pre-processing

- **Mask Creation**: To begin the training a very accurate mask was needed to be made from the coordinates of the contours given. Since the coordinates were provided as discrete points, it was necessary to choose appropriate interpolation technique. For this dataset, the contour provided was detailed enough for even the simple polygonal line to suffice.

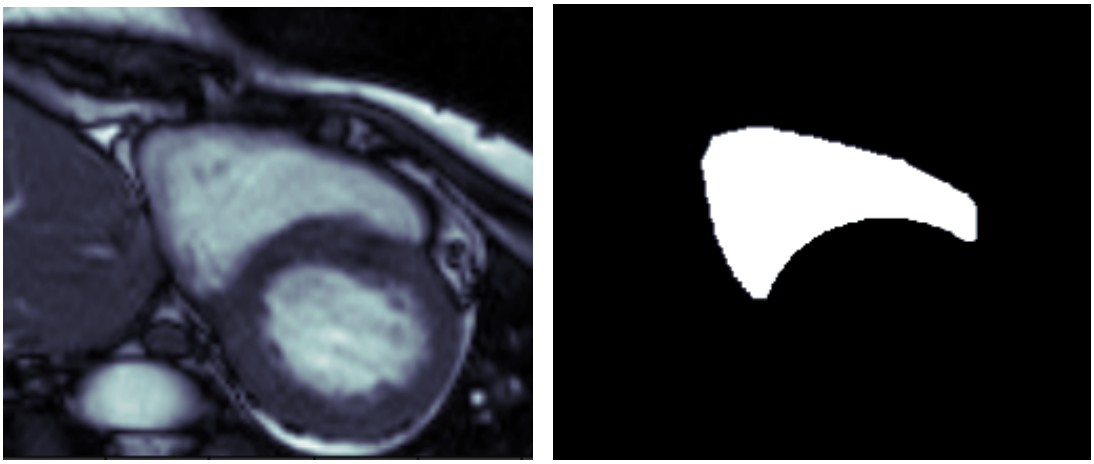

Figure 13: Sample mask creation

As to fill this contour we used BFS filling techniques where a seed was provided within the contour and then color spread to neighboring pixels until the boundary was encountered.

- **Normalisation**: The dataset due to the small number of training samples was skewed and the training was constantly falling victim to

vanishing gradients. With proper scaling and translation, the skewness of the data was reduced and the training completed smoothly.
In order to centralize the right skewed data, we first took the square root of the pixel values before normalizing. This yielded data which was much more central.

- **Weiner Filter**: The Weiner Filter is an advanced filter for removing noise as well as deblurring damaged images. But the original dataset had pretty clear images, on application of the filter most of the images suffered blurring and reduction of clarity thereby negatively affecting the result.

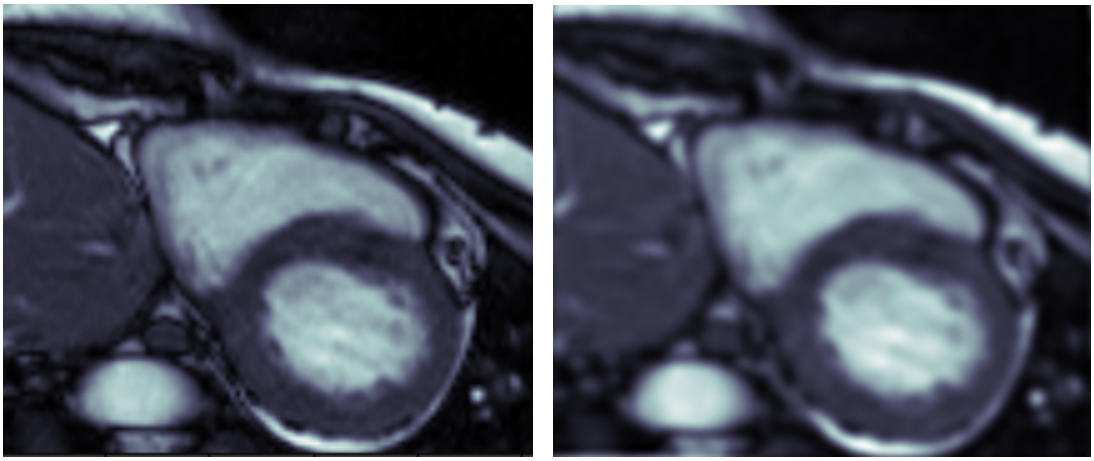

Figure 14: Weiner Filter Application

Thus, the Weiner Filters did not improve the performance of the model as the input dataset was already denoised and clear.

5.2 Post-Processing Techniques

- **Ensemble**: We can use multiple models with different initializations and thresholds. for every pixel in final output, we binary mask the multiple outputs and do a majority vote count. This technique is sure to improve upon the final performance of the model but could not be implemented due to the huge training time and computational cost.

- **Region Growing Methods**: For filling blobs and holes in the final output, graph based methods based on interior search and BFS filling was applied to the output of the models. With proper tuning, it was able to boost the average Dice Score of some models by 0.1-0.15. The low improvement depicts the robustness and completeness of the predicted contour by the original model.

6 Loss Function Weights

Throughout the experiment, we used two kinds of loss functions Binary Cross Entropy and Dice Loss. In our Baseline model, the weights ratio for the BCE Loss and Dice Loss is 1:3 respectively. Here, we try several other weight ratios:-

BCE Loss Weight	Dice Loss Weight	Average Dice Score	Colour
3	1	18.58	Green
1	**1**	**83.84**	**red**
1	3	82.38	pink
1	10	82.05	blue

Table 12: Average Dice Score with Different Loss Weights

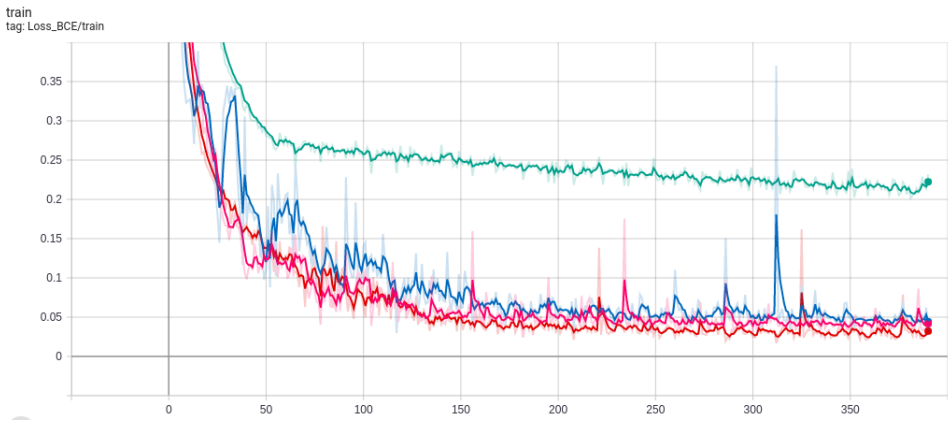

Figure 15: BCE Loss for Different Weights

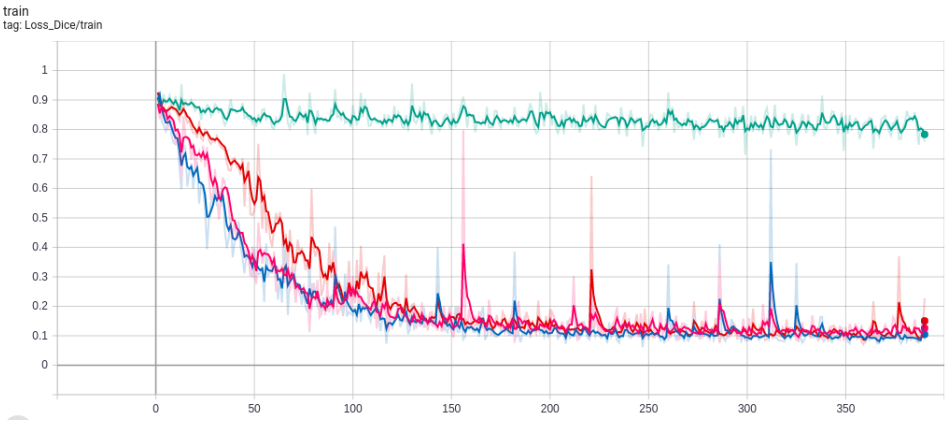

Figure 16: Dice Loss for Different Weights