Image Segmentation Using UNET Strategic Hyper-parameter Tuning *Assignment-1, EE-782*

Rathod Harekrissna Upendra, 17D070001
Utkarsh Bhalode, 17D070006
Prajval Nakrani, 17D070014

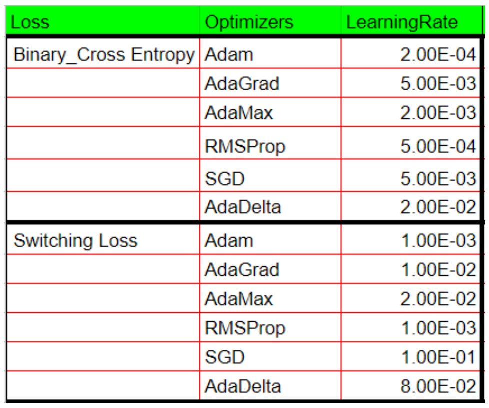
November 3, 2020

1 Introduction

Generally, training a neural network is a very tedious task as it has many hyper-parameters to tune which takes a lot of time. In this assignment we will try to find out strategies to tune one (or a few) hyper-parameters for image segmentation tasks. Our experiments include different optimizers, learning rate schedulers at different training-validation splits along with different regularizers, loss functions and different types of augmentation. Each of the segments focus on one particular hyper-parameter and it's major variations w.r.t other parameters possible.

2 Optimizers

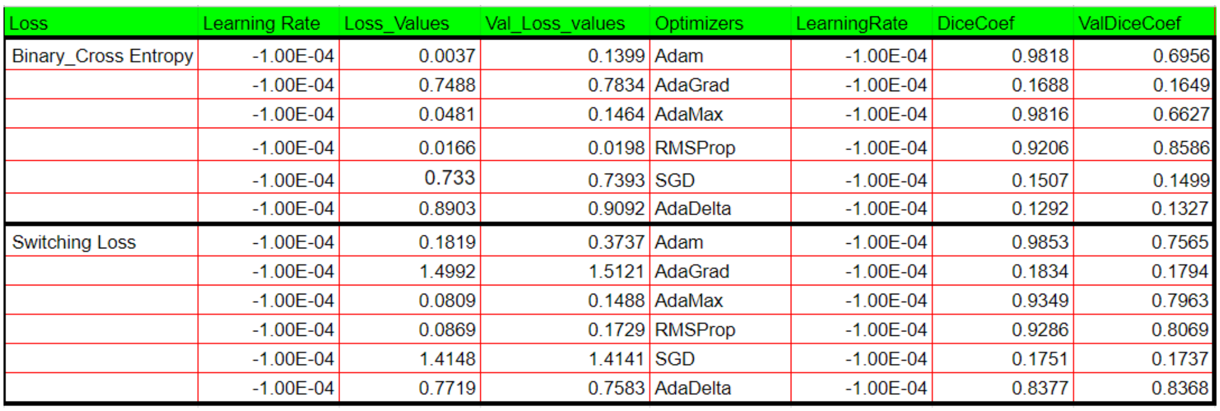
In this section, we try to search Learning Rates for fast convergence for different Optimizers. For the experiment we use two instances of losses: BCE Switching Loss. We ran the experiments multiple times to reach convergence within a current number of epochs. The strategy was to 1st get the rough order of magnitude for learning rate then get the best integer in that order, which we found using simple binary search. Hence each optimizer took around 4+5 = 9 experiments to get the optimum value for faster convergence. The results for optimal learning rate for both the losses per Optimizer are mentioned in the table below. Thus we get an estimate about the appropriate learning rate values.



Loss	Optimizers	LearningRate
Binary_Cross Entropy	Adam	2.00E-04
	AdaGrad	5.00E-03
	AdaMax	2.00E-03
	RMSProp	5.00E-04
	SGD	5.00E-03
	AdaDelta	2.00E-02
Switching Loss	Adam	1.00E-03
	AdaGrad	1.00E-02
	AdaMax	2.00E-02
	RMSProp	1.00E-03
	SGD	1.00E-01
	AdaDelta	8.00E-02

Figure 1: Learning rate for Fast convergence for different Optimizers

In the next part, we trained the model using different optimizers and a constant learning rate of 1E-4 and 150 epochs (for uniformity). We can see that for some optimizers with the give learning rate convergence is reached but for some its not reached. The values of losses and evaluation metric (dice coefficients) are mentioned in the table that follows.

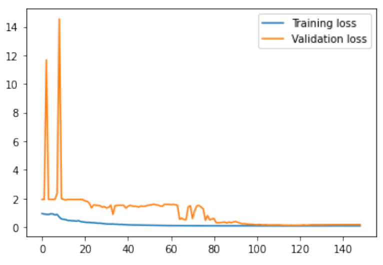


Loss	Learning Rate	Loss_Values	Val_Loss_values	Optimizers	LearningRate	DiceCoef	ValDiceCoef
Binary_Cross Entropy	-1.00E-04	0.0037	0.1399	Adam	-1.00E-04	0.9818	0.6956
	-1.00E-04	0.7488	0.7834	AdaGrad	-1.00E-04	0.1688	0.1649
	-1.00E-04	0.0481	0.1464	AdaMax	-1.00E-04	0.9816	0.6627
	-1.00E-04	0.0166	0.0198	RMSProp	-1.00E-04	0.9206	0.8586
	-1.00E-04	0.733	0.7393	SGD	-1.00E-04	0.1507	0.1499
	-1.00E-04	0.8903	0.9092	AdaDelta	-1.00E-04	0.1292	0.1327
Switching Loss	-1.00E-04	0.1819	0.3737	Adam	-1.00E-04	0.9853	0.7565
	-1.00E-04	1.4992	1.5121	AdaGrad	-1.00E-04	0.1834	0.1794
	-1.00E-04	0.0809	0.1488	AdaMax	-1.00E-04	0.9349	0.7963
	-1.00E-04	0.0869	0.1729	RMSProp	-1.00E-04	0.9286	0.8069
	-1.00E-04	1.4148	1.4141	SGD	-1.00E-04	0.1751	0.1737
	-1.00E-04	0.7719	0.7583	AdaDelta	-1.00E-04	0.8377	0.8368

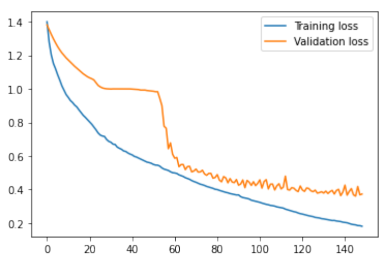
Figure 2: Constant Learning rate for different Optimizers

Thus we can see that using the same learning rate for different optimizers is not a good idea as some optimizers converge but some take more number of epochs to converge (more than 800 epochs in some cases.). Thus it is important to pre-hand decide the appropriate learning rate values. For getting a better feel of what is happening, we have also recorded the graph of loss vs epochs for each of this experiments on one of the validation data and given below.

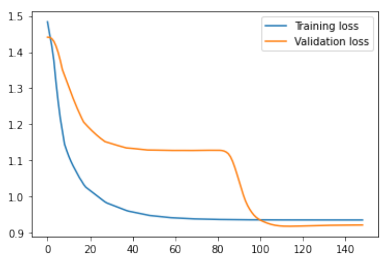
Note that this part has no learning rate scheduler implemented as we decide it in the next step of the experiments and hence all the graphs ae very wiggly and not properly converging which shows the need of learning rate scheduler.



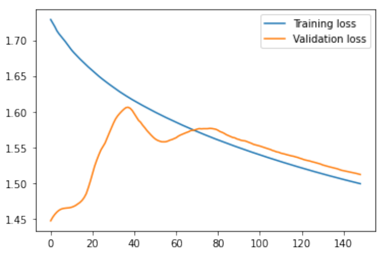
(a) RMSProp - Switching Loss



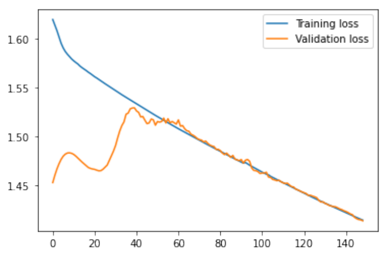
(b) Adam - Switching Loss



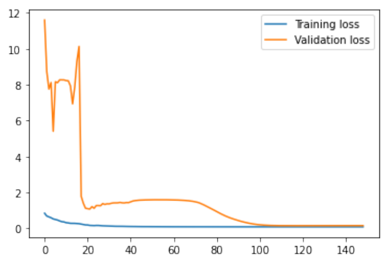
(c) AdaDelta - Switching Loss



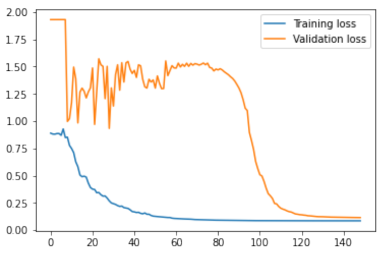
(d) AdaGrad - Switching Loss



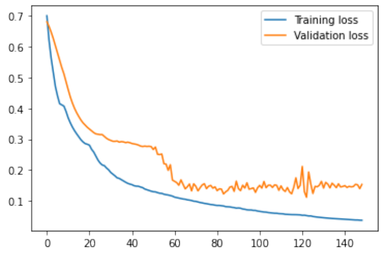
(e) SGD - Switching Loss



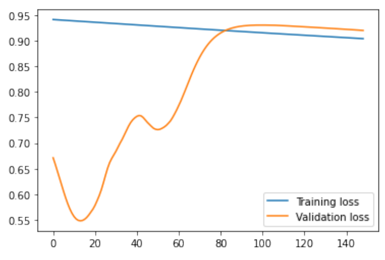
(f) AdaMax- Switching Loss



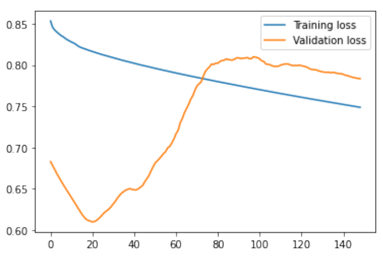
(g) RMSProp - BCE Loss



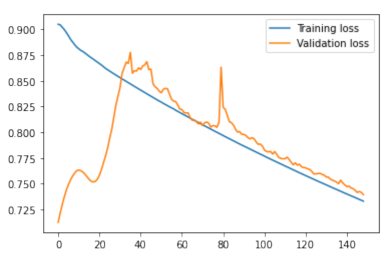
(h) Adam - BCE Loss



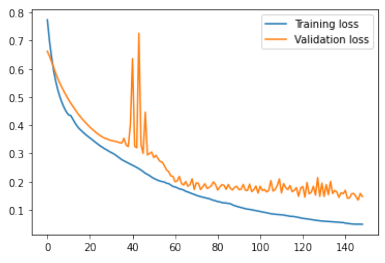
(i) AdaDelta - BCE Loss



(j) AdaGrad - BCE Loss



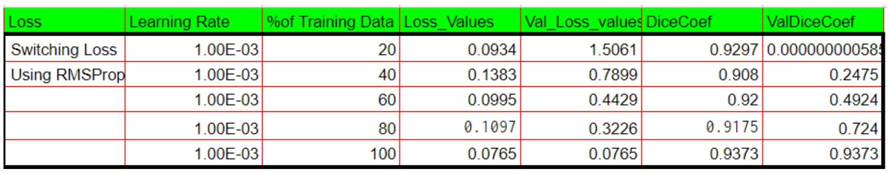
(k) SGD - BCE Loss



(l) AdaMax - BCE Loss

Figure 3: Training and Validation Loss Curves for varying Optimizers and Loss Functions

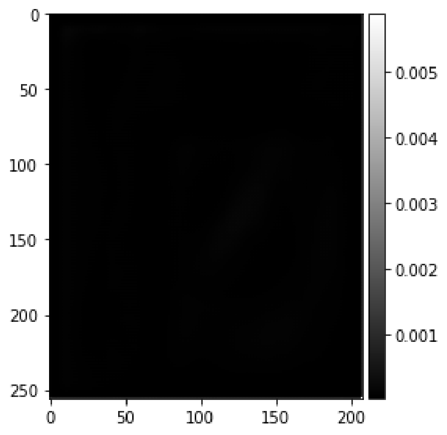
Next part is to train the model using 20 %,40%,60%,80%,100% of the training data use other for validation for 150 epochs. We trained the model using RMSProp as Optimizer (As we don't know the best optimizer yet) and used Switching Loss (again just a hunch that the switching loss is best according to the previous set of experiments, online resources and the paper we were provided). The values of losses and evaluation metric for both training and testing are noted below:

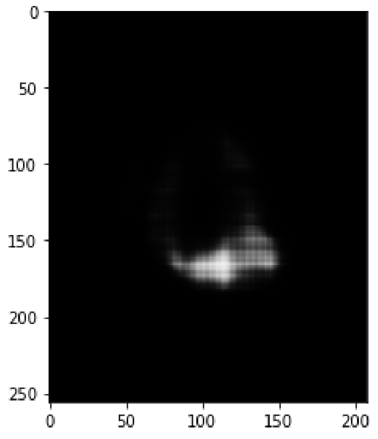


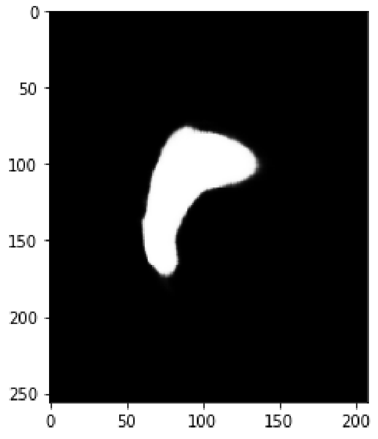
Loss	Learning Rate	%of Training Data	Loss_Values	Val_Loss_values	DiceCoef	ValDiceCoef
Switching Loss	1.00E-03	20	0.0934	1.5061	0.9297	0.00000000058
Using RMSProp	1.00E-03	40	0.1383	0.7899	0.908	0.2475
	1.00E-03	60	0.0995	0.4429	0.92	0.4924
	1.00E-03	80	0.1097	0.3226	0.9175	0.724
	1.00E-03	100	0.0765	0.0765	0.9373	0.9373

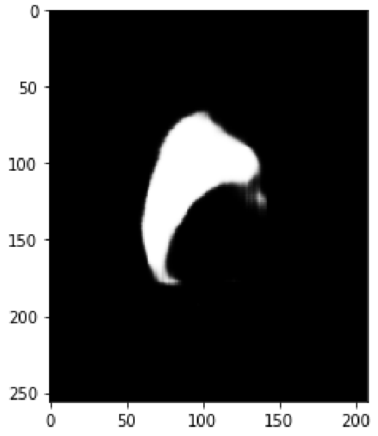
Figure 4: Constant Learning rate for different Optimizers

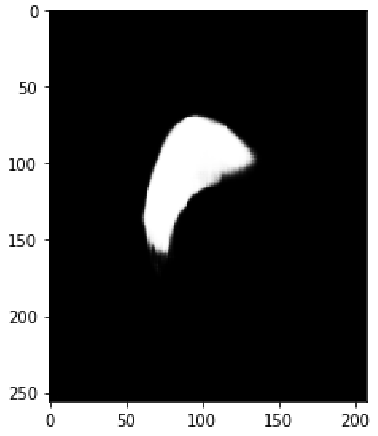
Attached Below are the outputs for better clarity:


(a) RMSProp - 20%


(b) RMSProp - 40%


(c) RMSProp - 60%


(d) RMSProp - 80%


(e) RMSProp - 100%

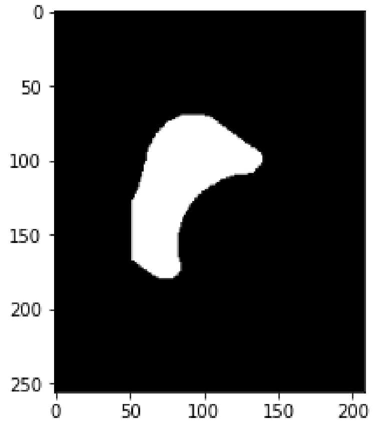

(f) Target Image

Figure 5: Training Images for RMSprop Optimizer for varying Train-Validation Split

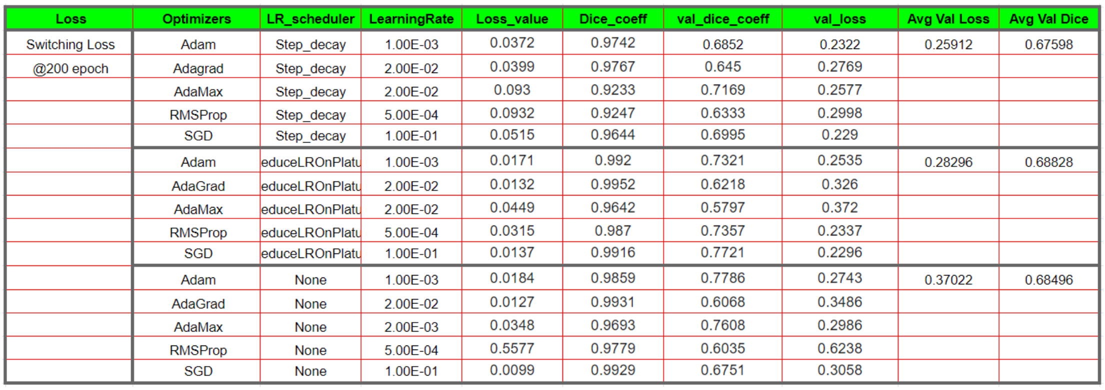
As we can see that RMSProp doesn't work well in small training data. In the next section we try the same experiment with Adam and SGD. Looking at the empirical results, we can conclude that Adam works quite well with less data. It was also seen that Adam was more sensitive to learning rate.

2.1 Conclusion

- We can see that for appropriate learning rate RMSProp, Adam and SGD gives the results. But if we use same learning rate then depending on that rate, some optimizers converge but some don't converge.
- Thus it is important to pre decide the appropriate learning rate values for the corresponding optimizer.
- Switching loss in general gave better results than using BCE loss across the experiments
- Different Optimizers have a range of appropriate learning rates for fast convergence
- For lower training data, RMSProp doesnt give good results.Adam and SGD on give comparatively better results. And Adam give better results than SGD.

3 Learning Rate Scheduler

In this section, we select 2 good learning rate schedulers based on current number of epochs namely Reduce Learning Rate on Plateau (ReduceLRonPLat), step decay and we also see the behaviour of each loss on both of these LR schedulers and also without any LR scheduler. The results of all the experiments is given below



Loss	Optimizers	LR_scheduler	LearningRate	Loss_value	Dice_coeff	val_dice_coeff	val_loss	Avg Val Loss	Avg Val Dice
Switching Loss	Adam	Step_decay	1.00E-03	0.0372	0.9742	0.6852	0.2322	0.25912	0.67598
@200 epoch	Adagrad	Step_decay	2.00E-02	0.0399	0.9767	0.645	0.2769		
	AdaMax	Step_decay	2.00E-02	0.093	0.9233	0.7169	0.2577		
	RMSProp	Step_decay	5.00E-04	0.0932	0.9247	0.6333	0.2998		
	SGD	Step_decay	1.00E-01	0.0515	0.9644	0.6995	0.229		
	Adam	educeLROnPlatu	1.00E-03	0.0171	0.992	0.7321	0.2535	0.28296	0.68828
	AdaGrad	educeLROnPlatu	2.00E-02	0.0132	0.9952	0.6218	0.326		
	AdaMax	educeLROnPlatu	2.00E-02	0.0449	0.9642	0.5797	0.372		
	RMSProp	educeLROnPlatu	5.00E-04	0.0315	0.987	0.7357	0.2337		
	SGD	educeLROnPlatu	1.00E-01	0.0137	0.9916	0.7721	0.2296		
	Adam	None	1.00E-03	0.0184	0.9859	0.7786	0.2743	0.37022	0.68496
	AdaGrad	None	2.00E-02	0.0127	0.9931	0.6068	0.3486		
	AdaMax	None	2.00E-03	0.0348	0.9693	0.7608	0.2986		
	RMSProp	None	5.00E-04	0.5577	0.9779	0.6035	0.6238		
	SGD	None	1.00E-01	0.0099	0.9929	0.6751	0.3058		

Figure 6: Different learning rate scheduler for different losses

We have recorded the training and validation plot for all these experiments which are given below. Note that in the below images, 1st row corresponds to Adam, 2nd Row corresponds to Adamax, 3rd row corresponds to Adagrad, 4th row corresponds to RMSprop, 5th row corresponds to sgd and 1st column corresponds to LRplat, 2nd column corresponds to Step decay and last corresponds to None.

For getting a better feel of what is happening, we have also recorded the output of each of this experiments on one of the validation data and given below. Note that in the below images, 1st row corresponds to Adam, 2nd Row corresponds to Adamax, 3rd row corresponds to Adagrad, 4th row corresponds to RMSprop, 5th row corresponds to sgd and 1st column corresponds to LRplat, 2nd column corresponds to Step decay and last corresponds to No LR scheduler

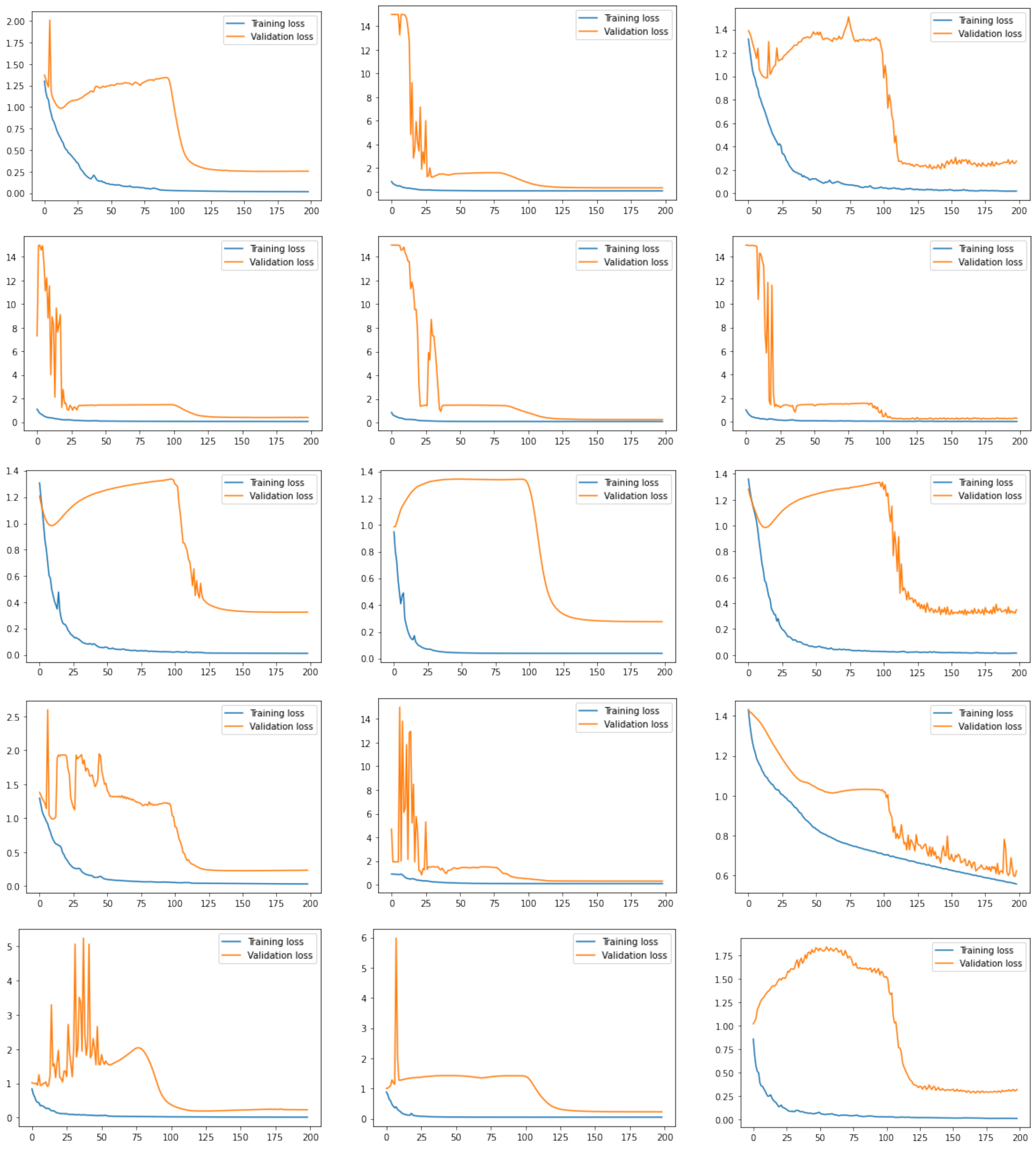


Figure 7: The training and validation plot for all the LR scheduler experiments

Note that the Target Value for the all output images produced henceforth is:

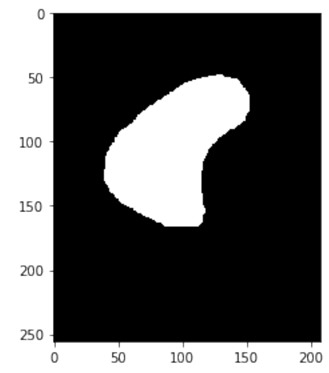


Figure 8: True Validation Output

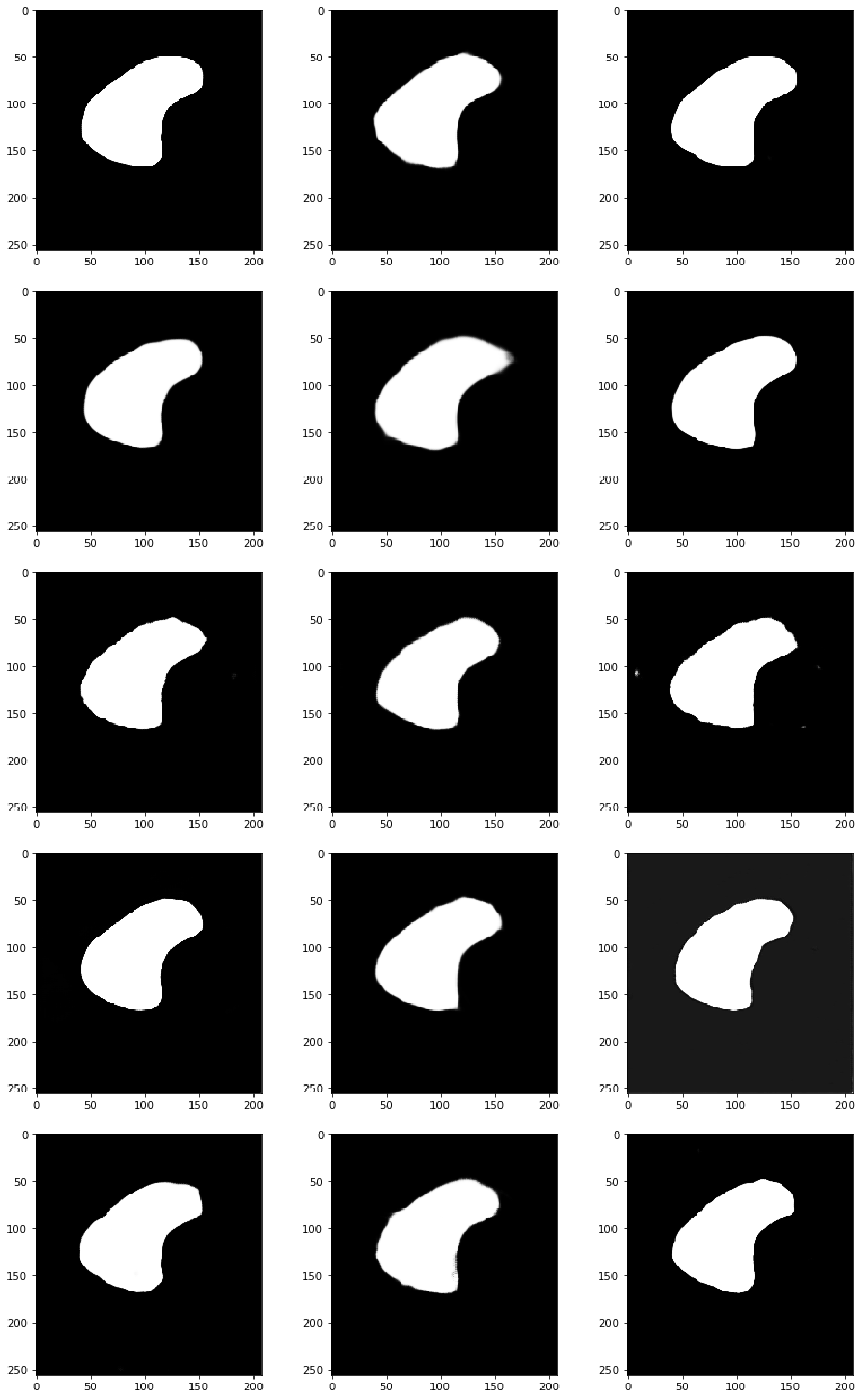


Figure 9: Sample validation outputs at different LR scheduler

We have noted various things from these set of experiments. One Is that not all the optimizers work well with learning rate scheduler, some work better without it. Some do need scheduler badly while others can make do even without any scheduler. We have made a summay chart for better visualiztion of data which is given below

Legend:(Blue is Step Decay, Red is ReduceLRplat and orange is None), y-axis: Val dice coefficient

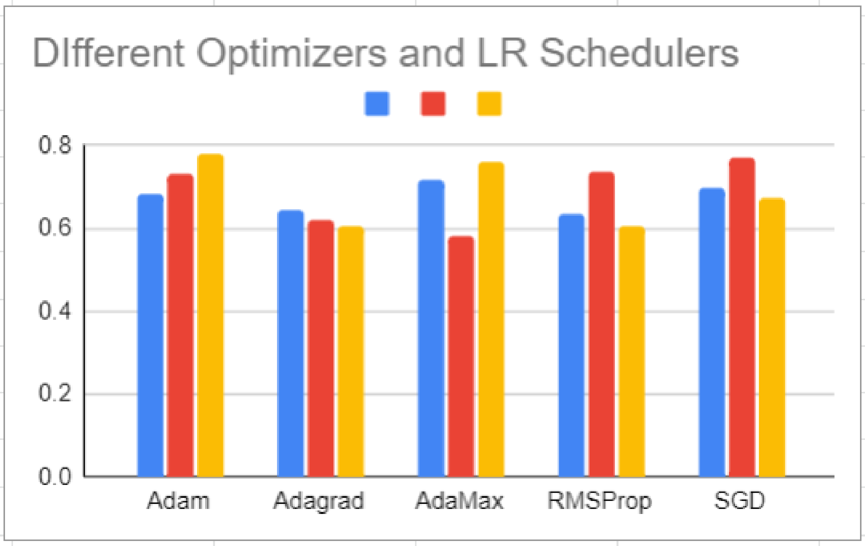
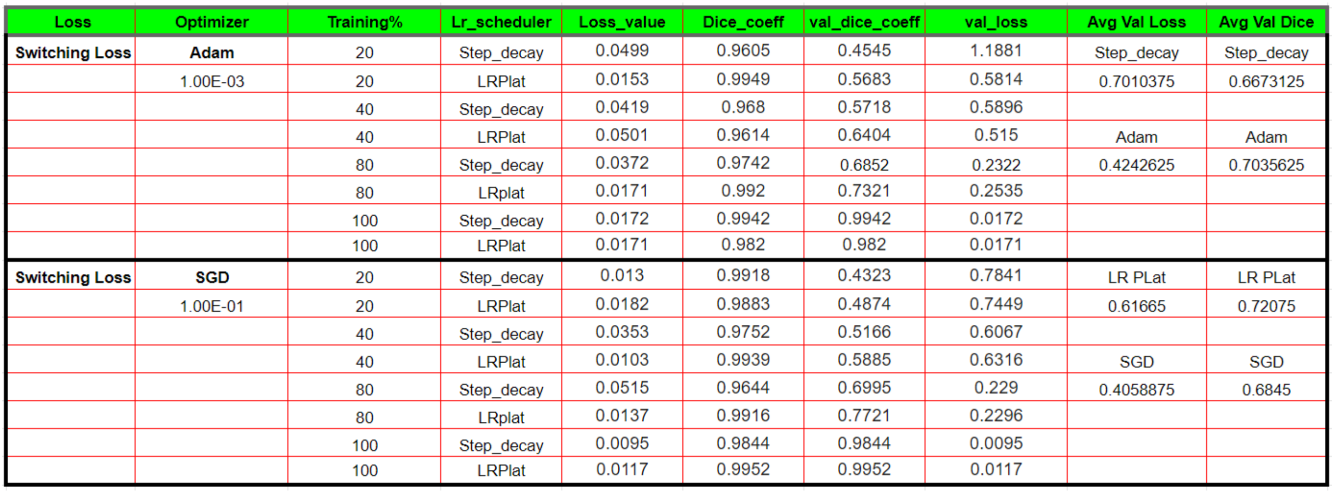


Figure 10: LR schedulers at different learning rates

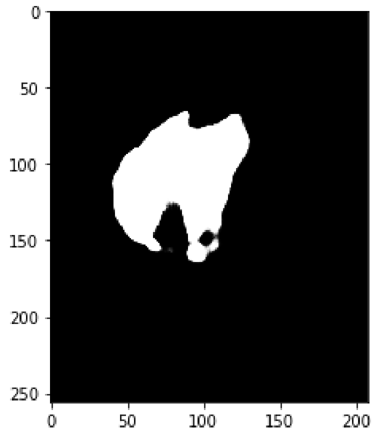
Now we train on very small to very large training dataset on different learning rate scheduler and 2 different optimizers which we found best from the above observations namely Adam and SGD. The results are as shown in the table below. We have also stored the training and validation plat for each of the experiments done above, which can be found in the zip file attached int he particular folders

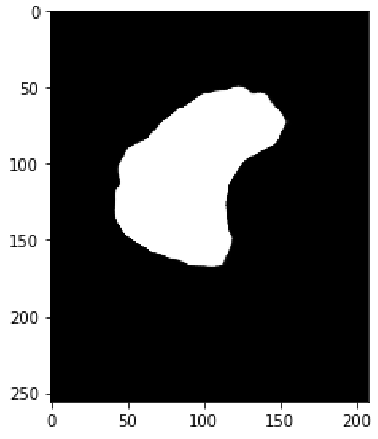


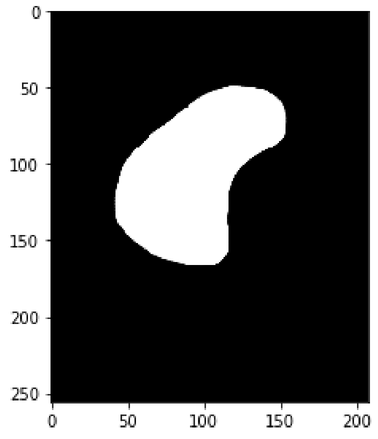
Loss	Optimizer	Training%	Lr_scheduler	Loss_value	Dice_coeff	val_dice_coeff	val_loss	Avg Val Loss	Avg Val Dice
Switching Loss	**Adam**	20	Step_decay	0.0499	0.9605	0.4545	1.1881	Step_decay	Step_decay
	1.00E-03	20	LRPlat	0.0153	0.9949	0.5683	0.5814	0.7010375	0.6673125
		40	Step_decay	0.0419	0.968	0.5718	0.5896		
		40	LRPlat	0.0501	0.9614	0.6404	0.515	Adam	Adam
		80	Step_decay	0.0372	0.9742	0.6852	0.2322	0.4242625	0.7035625
		80	LRplat	0.0171	0.992	0.7321	0.2535		
		100	Step_decay	0.0172	0.9942	0.9942	0.0172		
		100	LRPlat	0.0171	0.982	0.982	0.0171		
Switching Loss	**SGD**	20	Step_decay	0.013	0.9918	0.4323	0.7841	LR PLat	LR PLat
	1.00E-01	20	LRPlat	0.0182	0.9883	0.4874	0.7449	0.61665	0.72075
		40	Step_decay	0.0353	0.9752	0.5166	0.6067		
		40	LRPlat	0.0103	0.9939	0.5885	0.6316	SGD	SGD
		80	Step_decay	0.0515	0.9644	0.6995	0.229	0.4058875	0.6845
		80	LRplat	0.0137	0.9916	0.7721	0.2296		
		100	Step_decay	0.0095	0.9844	0.9844	0.0095		
		100	LRPlat	0.0117	0.9952	0.9952	0.0117		

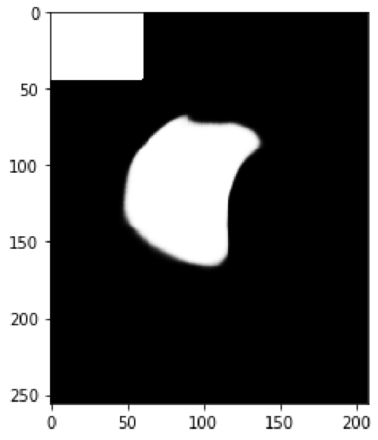
Figure 11: LR schedulers at different learning rates

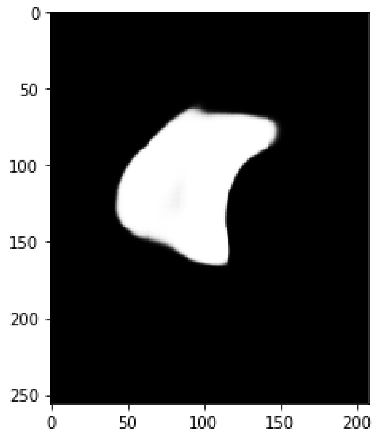
Note that the 20% dataset is run on 300 epochs and others are run on 200 epochs. The outputs on the same validation input clearly shows the improvements in output with increasing the training data. We can also make certain conclusions which are stated in the paragrph given after the photos. We also have the plots of training and val which are given in the zip file

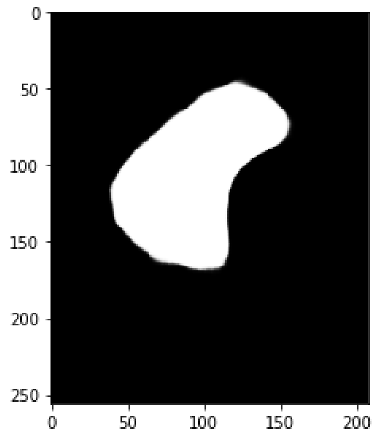

(a) 20% Adam LRplat

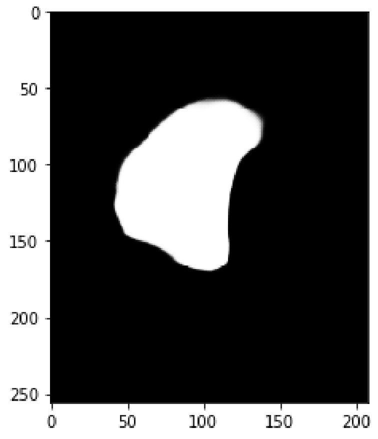

(b) 40% Adam LRplat

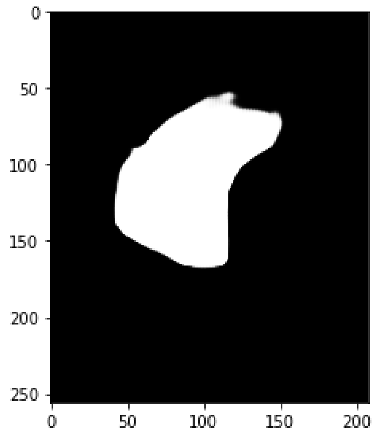

(c) 80% Adam LRplat

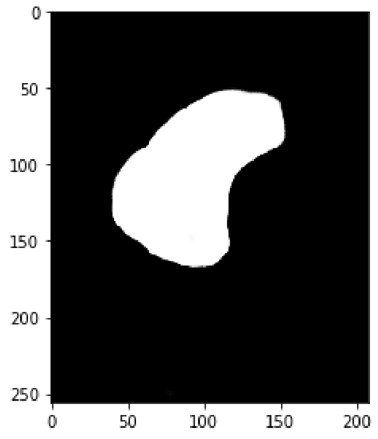

(d) 20% Adam step decay

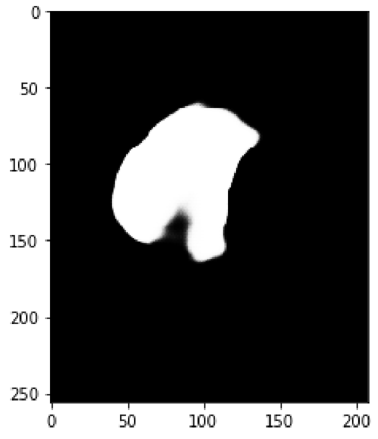

(e) 40% Adam step decay

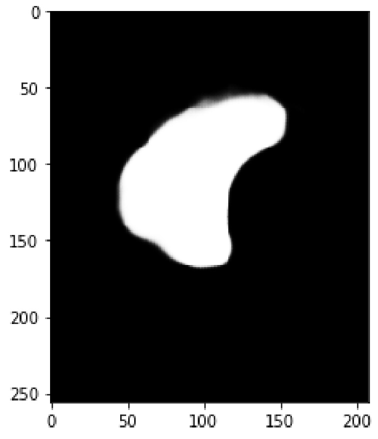

(f) 80% Adam step decay


(g) 20% SGD LRplat


(h) 40% SGD LRplat


(i) 80% SGD LRplat


(j) 20% SGD step decay


(k) 40% SGD step decay

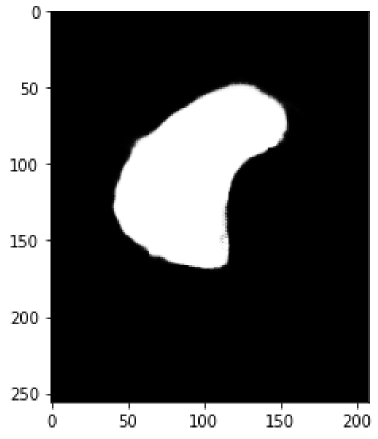

(l) 80% SGD step decay

Figure 12: Validation Images for varying Optimizers and Train-Validation Split

3.1 Conclusion

- We can see that Using Learning rate scheduler is not good for all the optimizers although it improves the performance in most cases, but the Adamax optimizer clearly works better without any scheduler which we have used.
- Adagrad has a very smooth learning plot for validation dataset with schedulers and also starts to validate better at lesser number of epochs compared to others but doesnt reach the better optimum while others take longer time but give better results. Thus Adagrad follows a more aggressive learning approach
- Adam and SGD doesnt have much problem with or without the schedulers although keeping scheduler certainly helps stabilizing the final convergent values in both the cases however these optimizers can perform well without schedulers also.
- RMSProp clearly needs a learning rate scheduler to work properly and the Reduce LR on plateau works even better.
- Overall, validation Dice coefficient wise, Reduce Learning rate on plateau works better
- Even at lower training data, LRplat is producing more sensible and complete results
- For lower training data, Adam is producing better results than SGD on both LR schedulers
- Because of limited time, we were only able to operate on 2 well known classes of LR scheduler there are others like CosineAnnealing, Cyclic LR scheduler and so on which can also be explored.

4 Weight Decay

In this section, we will try to tune to the optimum value of weight decay (lambda) for a U-NET architecture, for the task of image segmentation. We perform the experiment for various validation splits. Specifically, we try to find optimum weight decay value for 20%, 40%, 60% and 80% of the training data used for training the neural network.

4.1 Experiment Strategy

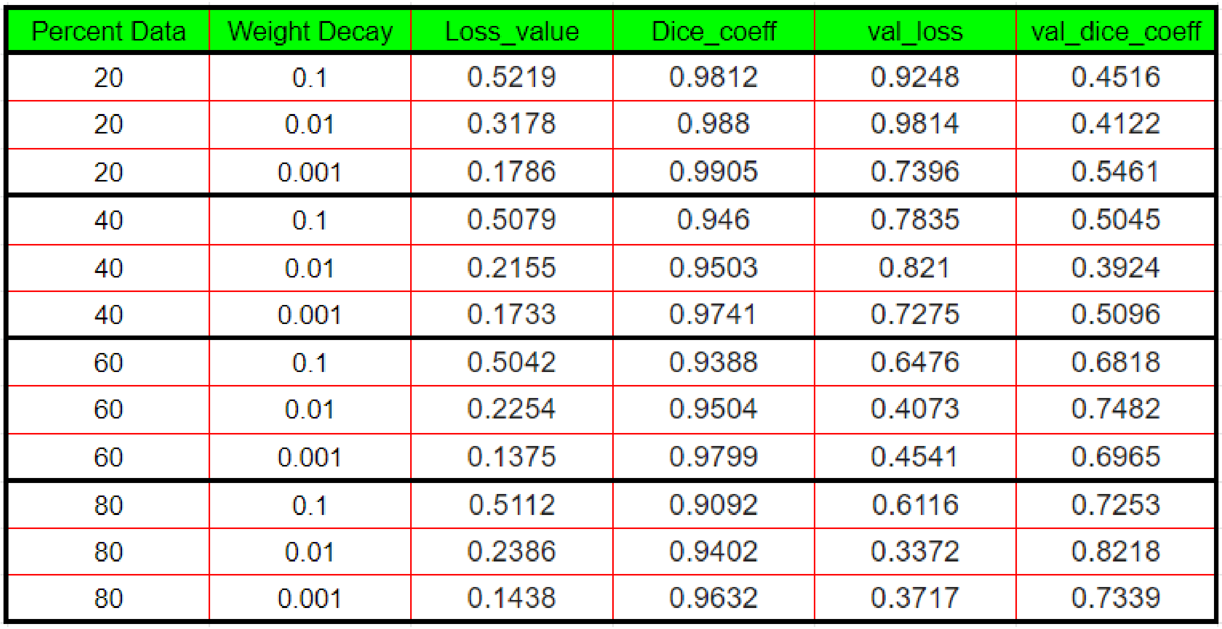
We had to tune weight decay values for each combination of train-validation split. Ideally, to tune the correct weight decay value for a single train-validation split would require one to try all the various values of weight decay. And to train the network to its optimum least point for a single value of weight decay for a fixed train-validation split would require immense amount of time. Hence, the strategy I adopted was as follows.

First I looked up on internet for typical values of weight decay. It turned out that typical values of weight decay lies between 0 and 0.1 typically on the logarithmic scale. Hence, I chose the weight decay values to be 0.1, 0.01 and 0.001.

Next, for training a single network, I decided I will first train the network with 80% data and observe how much time does it take for the network to converge. Whatever number of epochs it takes, I will fix that and train the other networks for the same number of epochs and compare the results.

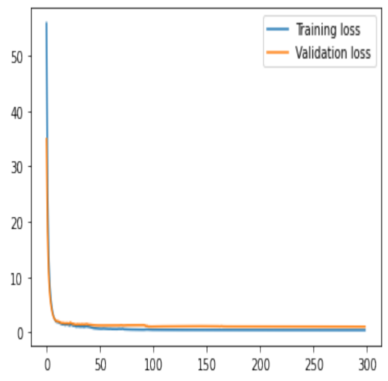
The loss is kept as switching loss and the optimizer is Adam (lr = 1e-3) with ReduceLRplat scheduler, which were found out to be the best combination from the above experiments

4.2 Results

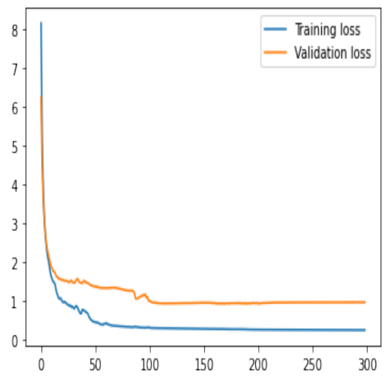


Percent Data	Weight Decay	Loss_value	Dice_coeff	val_loss	val_dice_coeff
20	0.1	0.5219	0.9812	0.9248	0.4516
20	0.01	0.3178	0.988	0.9814	0.4122
20	0.001	0.1786	0.9905	0.7396	0.5461
40	0.1	0.5079	0.946	0.7835	0.5045
40	0.01	0.2155	0.9503	0.821	0.3924
40	0.001	0.1733	0.9741	0.7275	0.5096
60	0.1	0.5042	0.9388	0.6476	0.6818
60	0.01	0.2254	0.9504	0.4073	0.7482
60	0.001	0.1375	0.9799	0.4541	0.6965
80	0.1	0.5112	0.9092	0.6116	0.7253
80	0.01	0.2386	0.9402	0.3372	0.8218
80	0.001	0.1438	0.9632	0.3717	0.7339

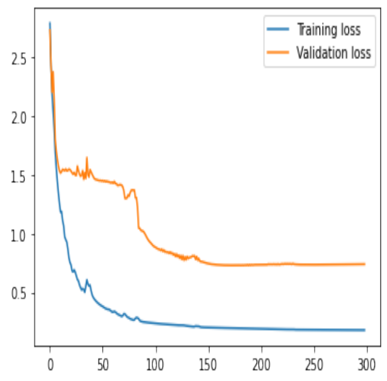
Figure 13: Weight Decay for various combinations of Train-Validation Split



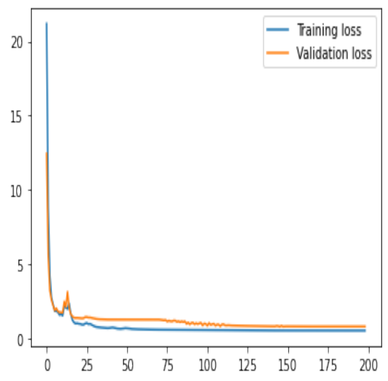
(a) 20% Data, Decay = 0.1



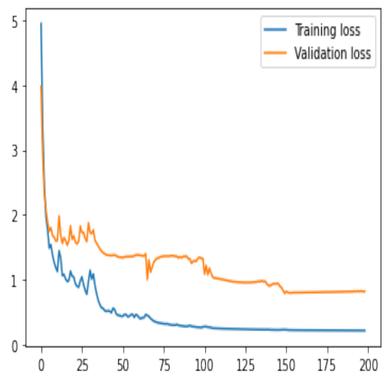
(b) 20% Data, Decay = 0.01



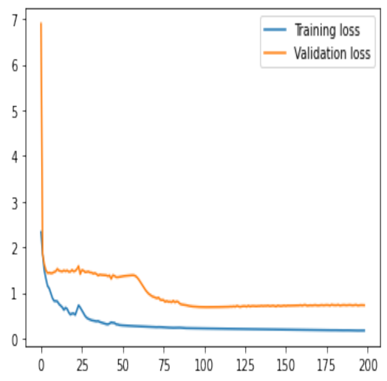
(c) 20% Data, Decay = 0.001



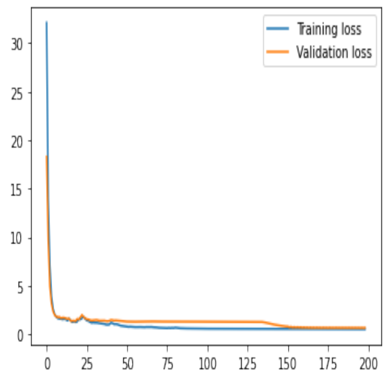
(d) 40% Data, Decay = 0.1



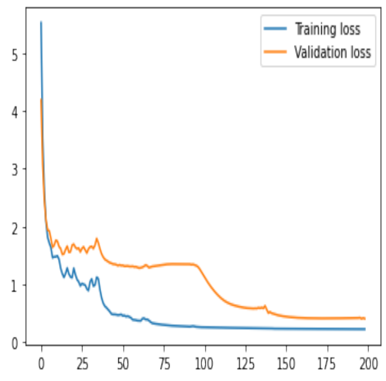
(e) 40% Data, Decay = 0.01



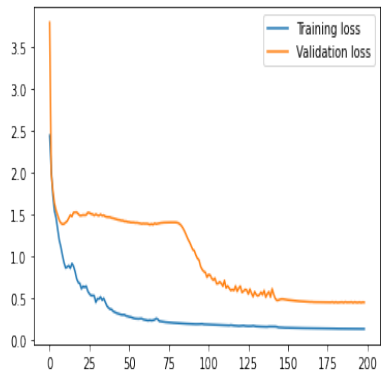
(f) 40% Data, Decay = 0.001



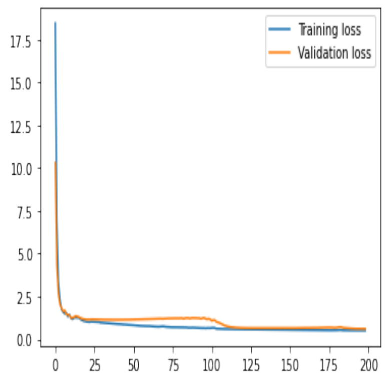
(g) 60% Data, Decay = 0.1



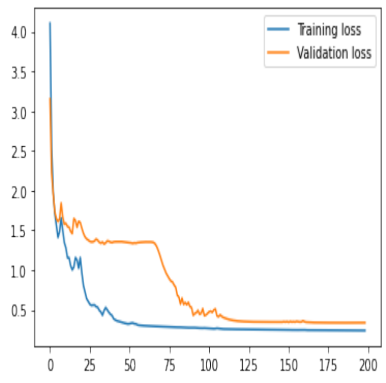
(h) 60% Data, Decay = 0.01



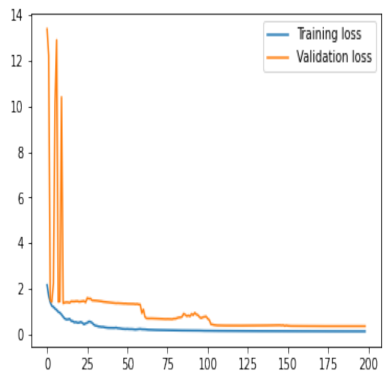
(i) 60% Data, Decay = 0.001



(j) 80% Data, Decay = 0.1



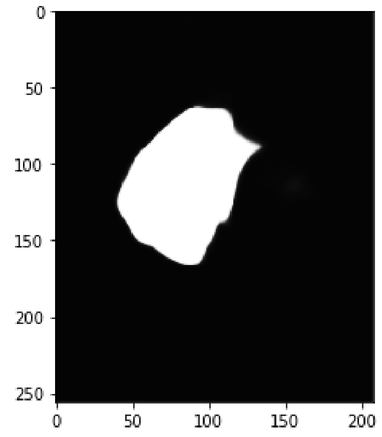
(k) 80% Data, Decay = 0.01

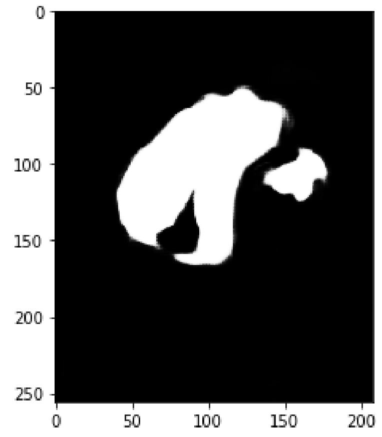


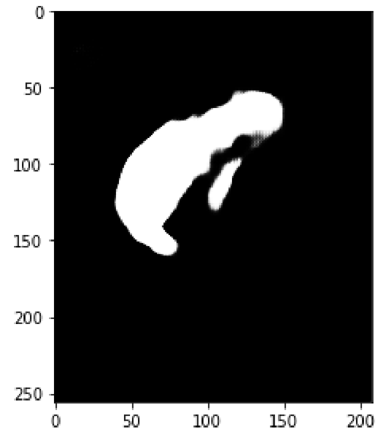
(l) 80% Data, Decay = 0.001

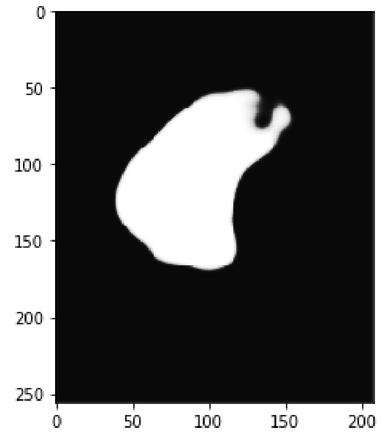
Figure 14: Training and Validation Loss Curves for varying Weight Decay and Validation Split

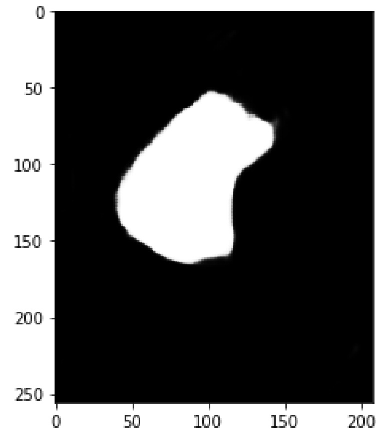
Now we are also keeping the various outputs of images for these experiments

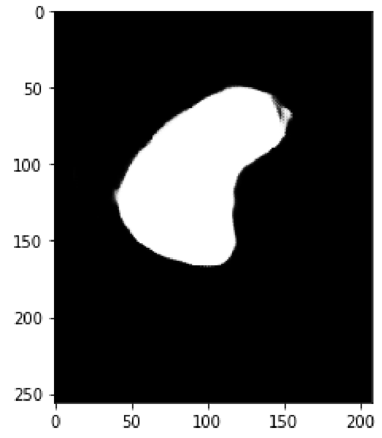

(a) 20% Data, Decay = 0.1

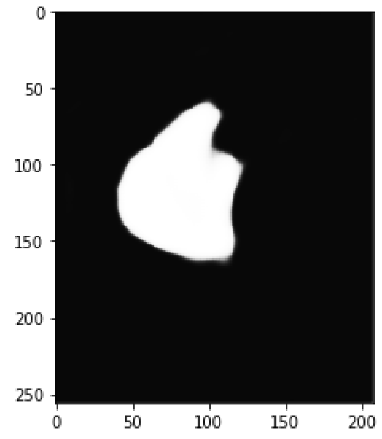

(b) 20% Data, Decay = 0.01

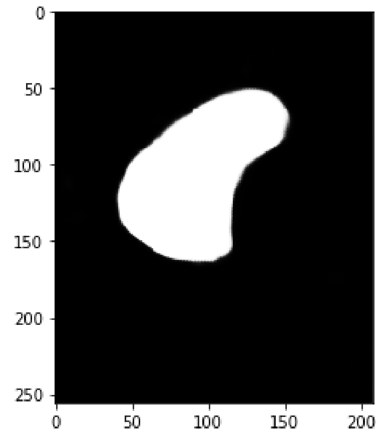

(c) 20% Data, Decay = 0.001

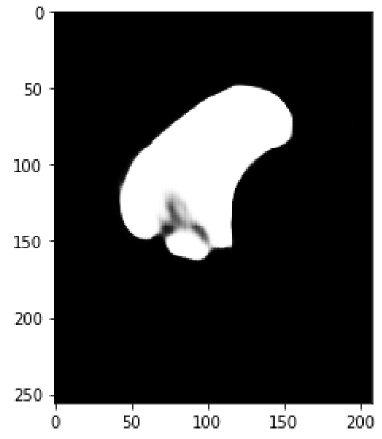

(d) 40% Data, Decay = 0.1

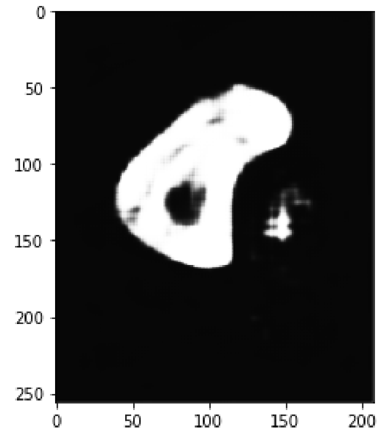

(e) 40% Data, Decay = 0.01

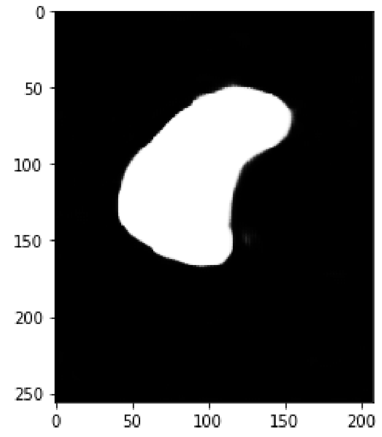

(f) 40% Data, Decay = 0.001


(g) 60% Data, Decay = 0.1


(h) 60% Data, Decay = 0.01


(i) 60% Data, Decay = 0.001


(j) 80% Data, Decay = 0.1


(k) 80% Data, Decay = 0.01

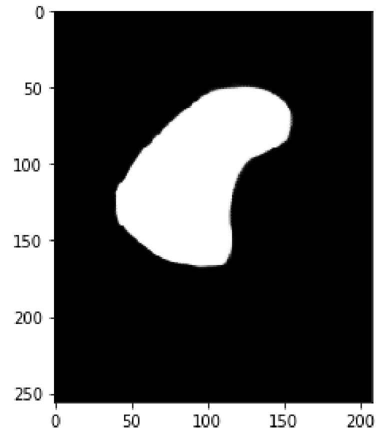

(l) 80% Data, Decay = 0.001

Figure 15: Validation Images for varying Weight Decay and Validation Split

4.3 Conclusion

- When the network is trained on higher amount of data, validation loss converges to a very low value with or without weight decay. That is, when the network is supplied with sufficient data, it generalizes well even without regularization. This can be observed from the training graphs of 80% train validation split.
- When the network is trained on lower amount of data, then the validation loss doesn't converge to low values for lower values of weight decay, however it converges to a lower value at higher weight decay rates. This implies that when network is supplied with low training data, it requires high regularization to generalize well. This is evident from the training loss curves for 20% validation split.
- However, when the model is supplied with higher amounts of training data, then due to excessive regularization model may not be able to even fit the given training data well. That is because the amount of data over which network has to generalize is already high and excessive regularization penalty hinders its capacity to learn complex patterns from the images. This is evident from the Validation images for 80% training data.
- The best is found out at 80% split with 0.01 weight decay which can be observed from table, giving 0.8218 val dice coefficient, which is quite good.

5 Dropout

In this section, we will try to tune optimum dropout rate for a U-NET architecture, for the task of image segmentation. We perform the experiment for various validation splits. Specifically, we try to find optimum dropout rate for 20%, 40%, 60% and 80% of the training data used for training the neural network. Note that the values in the table below are specifically dropout rates and not "keep" dropout rates.

5.1 Experiment Strategy

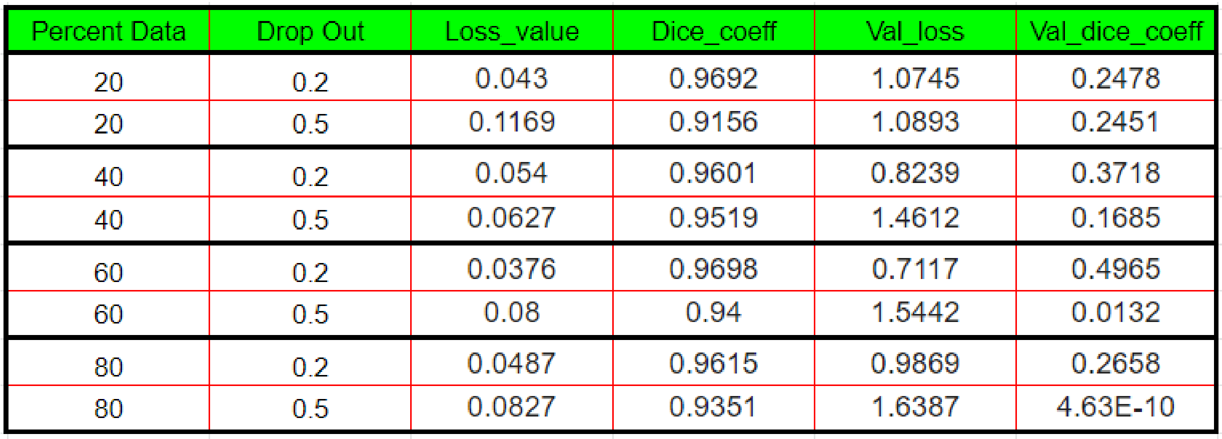
We had to tune dropout rates for each combination of train-validation split. Again here as well, ideally, to tune the correct dropout rate for a single train-validation split would require one to try all the various values of dropout rates. And to train the network to its optimum least point for a single value of dropout rate for a fixed train-validation split would require immense amount of time. Hence, the strategy I adopted was as follows.

I initially chose the dropout rates as 0.2, 0.5, 0.8. But, while training I realized that for 0.8 dropout rate, the network isn't converging at all even after training for a long time. The reason is that, we have not used the exact architecture as mentioned in the paper. We have used a simplified/smaller version of the baseline architecture given in the paper. As a result our model is already very simple and adding aggressive regularization like dropout values of 0.8 renders the network completely incapable of learning any patterns from the images. Hence, finally only two values of dropout rates were chose, i.e. 0.2 and 0.5

Next, for training a single network, I decided I will first train the network with 80% data with 0.5 dropout rate and observe how much time does it take for the network to converge. Whatever number of epochs it takes, I will fix that and train the other networks for the same number of epochs and compare the results.

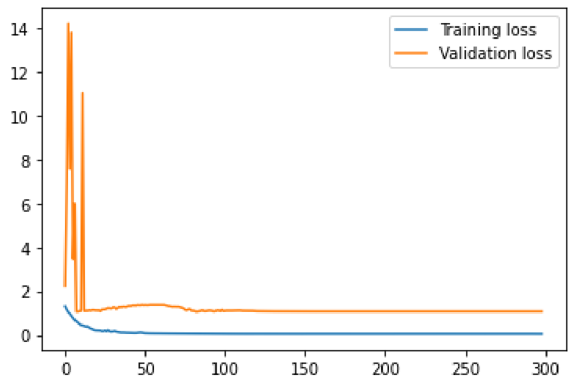
Again the other hyper parameters are fixed from the best results of previous experiments i.e. Adam(1e-3) with ReduceLRplat on switching loss and 0.01 weight decay.

5.2 Results:

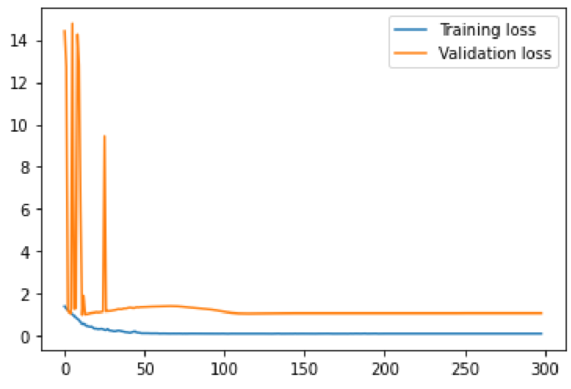


Percent Data	Drop Out	Loss_value	Dice_coeff	Val_loss	Val_dice_coeff
20	0.2	0.043	0.9692	1.0745	0.2478
20	0.5	0.1169	0.9156	1.0893	0.2451
40	0.2	0.054	0.9601	0.8239	0.3718
40	0.5	0.0627	0.9519	1.4612	0.1685
60	0.2	0.0376	0.9698	0.7117	0.4965
60	0.5	0.08	0.94	1.5442	0.0132
80	0.2	0.0487	0.9615	0.9869	0.2658
80	0.5	0.0827	0.9351	1.6387	4.63E-10

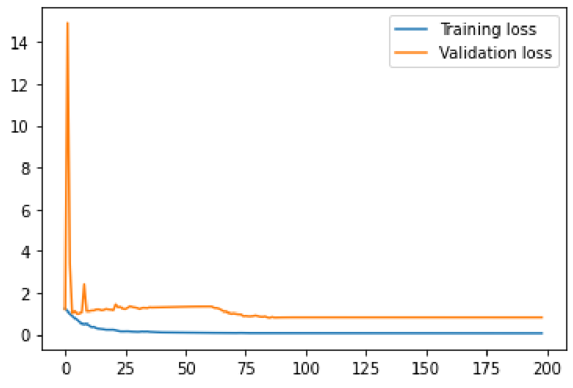
Figure 16: Dropout rates for various combinations of Train-Validation Split



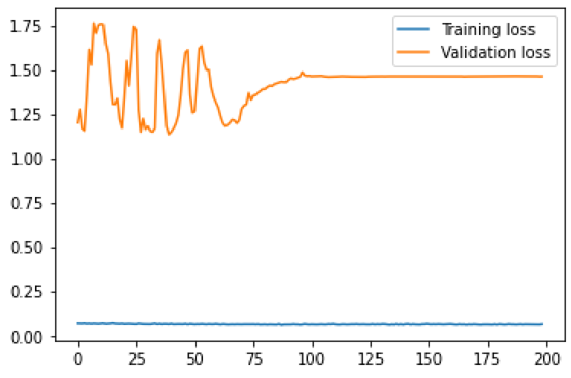
(a) 20% Data, Dropout = 0.2



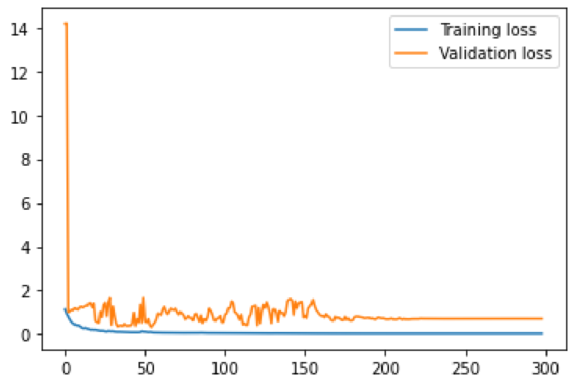
(b) 20% Data, Dropout = 0.5



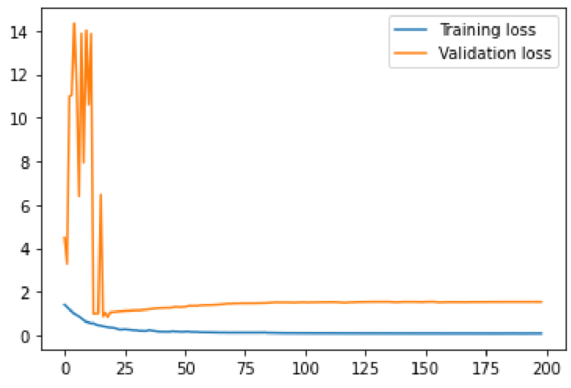
(c) 40% Data, Dropout = 0.2



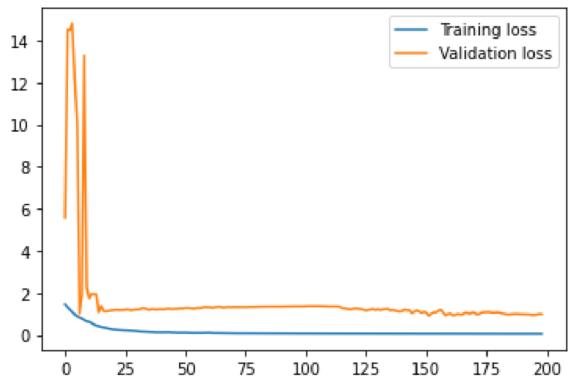
(d) 40% Data, Dropout = 0.5



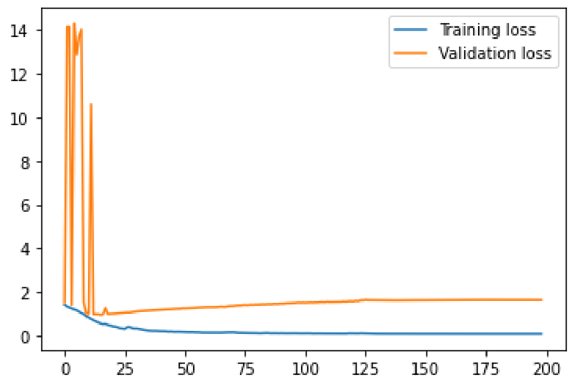
(e) 60% Data, Dropout = 0.2



(f) 60% Data, Dropout = 0.5

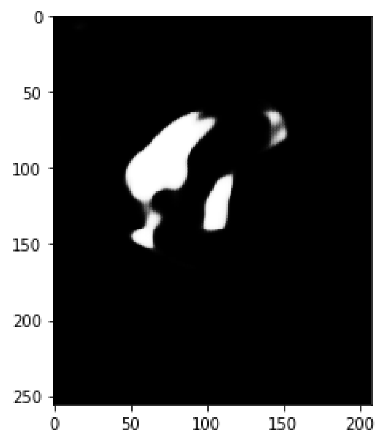


(g) 80% Data, Dropout = 0.2

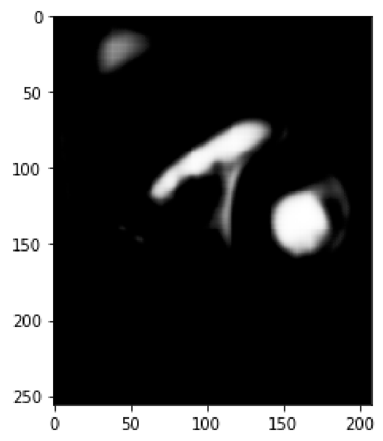


(h) 80% Data, Dropout = 0.5

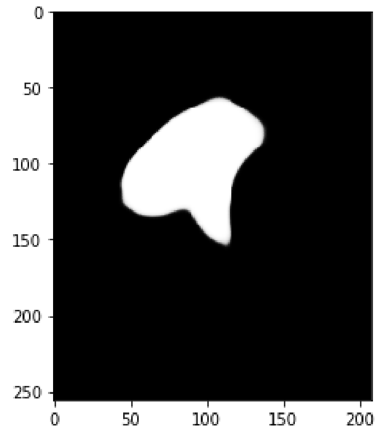
Figure 17: Training and Validation Loss Curves for varying Dropout Rates and Validation Split



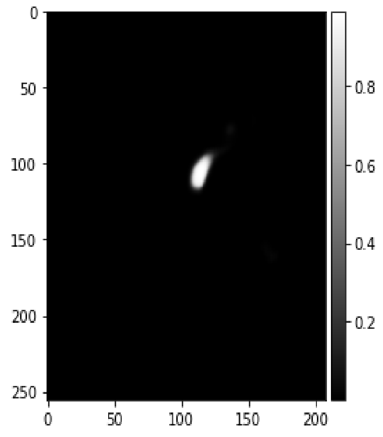
(a) 20% Data, Dropout = 0.2



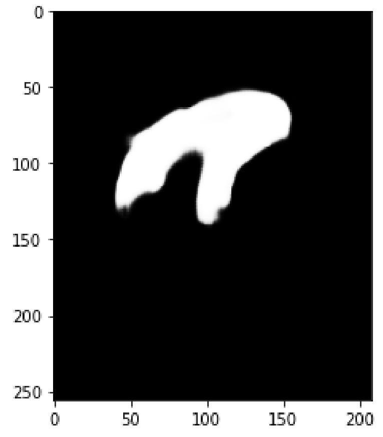
(b) 20% Data, Dropout = 0.5



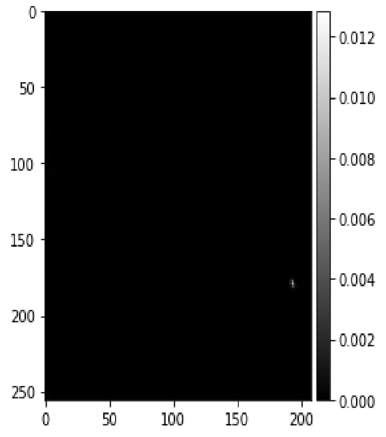
(c) 40% Data, Dropout = 0.2



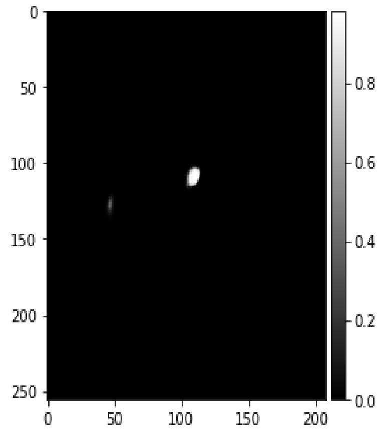
(d) 40% Data, Dropout = 0.5



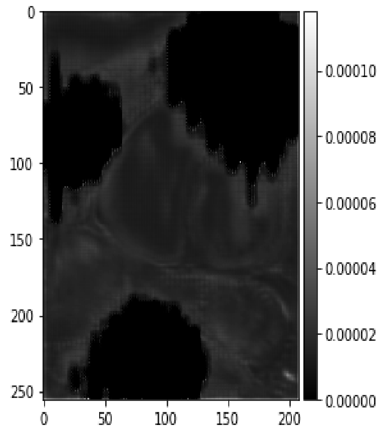
(e) 60% Data, Dropout = 0.2



(f) 60% Data, Dropout = 0.5



(g) 80% Data, Dropout = 0.2



(h) 80% Data, Dropout = 0.5

Figure 18: Validation Images for varying Dropout Rates and Validation Split

5.3 Conclusion

- An overall observation from the validation images would be that dropout rates of 0.2 and 0.5 both are pretty high for this network. The reason is that our network is already small/simple compared to the baseline UNET architecture mentioned in the paper. Hence, it is already simple enough to not over-fit and hence adding huge regularization penalty renders the network incapable to learn patterns effectively from images. It just so turns out that even dropout rate of 0.2 is high for our network.

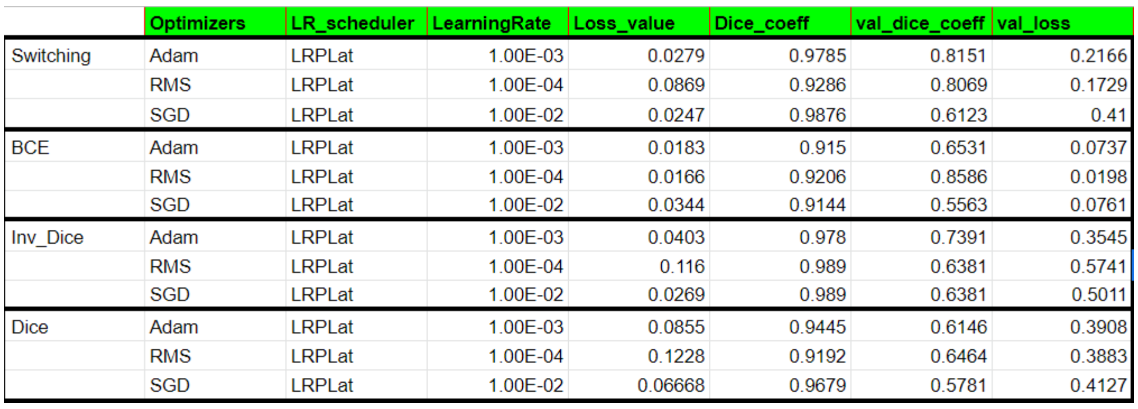
- From the validation images of 80% training split, we observe that even with a dropout rate of 0.2 the network is not able to learn the patterns as there are more images in dataset which increases the generalization of the dataset. So, for higher amounts of training data, we should not keep high dropout rate or in general high regularization penalty because the data itself is vast enough to ensure that the model generalizes well. Regularization in this case will in fact hinder the learning of the network.

6 Different Losses

We have finally selected the best hyper para maters considering switching loss everytime. But now given a set of good tools, we need to find a proper loss surface to minimize. Hence, In this section we ran Adam, SGD, RMSProp optimizers (3 best optimizers found yet) for 4 different losses: Switching loss, BCE loss, Dice BCE Loss, Inverse Dice BCE loss. Other parameters are fixed ie ReduceLRplat, weight decay = 0.01, no dropout for any of the layers.

6.1 Results

The following are the results of losses values dice coefficients for each run:



	Optimizers	LR_scheduler	LearningRate	Loss_value	Dice_coeff	val_dice_coeff	val_loss
Switching	Adam	LRPLat	1.00E-03	0.0279	0.9785	0.8151	0.2166
	RMS	LRPLat	1.00E-04	0.0869	0.9286	0.8069	0.1729
	SGD	LRPLat	1.00E-02	0.0247	0.9876	0.6123	0.41
BCE	Adam	LRPLat	1.00E-03	0.0183	0.915	0.6531	0.0737
	RMS	LRPLat	1.00E-04	0.0166	0.9206	0.8586	0.0198
	SGD	LRPLat	1.00E-02	0.0344	0.9144	0.5563	0.0761
Inv_Dice	Adam	LRPLat	1.00E-03	0.0403	0.978	0.7391	0.3545
	RMS	LRPLat	1.00E-04	0.116	0.989	0.6381	0.5741
	SGD	LRPLat	1.00E-02	0.0269	0.989	0.6381	0.5011
Dice	Adam	LRPLat	1.00E-03	0.0855	0.9445	0.6146	0.3908
	RMS	LRPLat	1.00E-04	0.1228	0.9192	0.6464	0.3883
	SGD	LRPLat	1.00E-02	0.06668	0.9679	0.5781	0.4127

Figure 19: Various Losses for RMSProp, Adam, SGD Optimizers

The following is the plot of results on losses values dice coefficients for each run:

Legend: (Blue is Adam, Red is RMSprop, yellow is SGD)

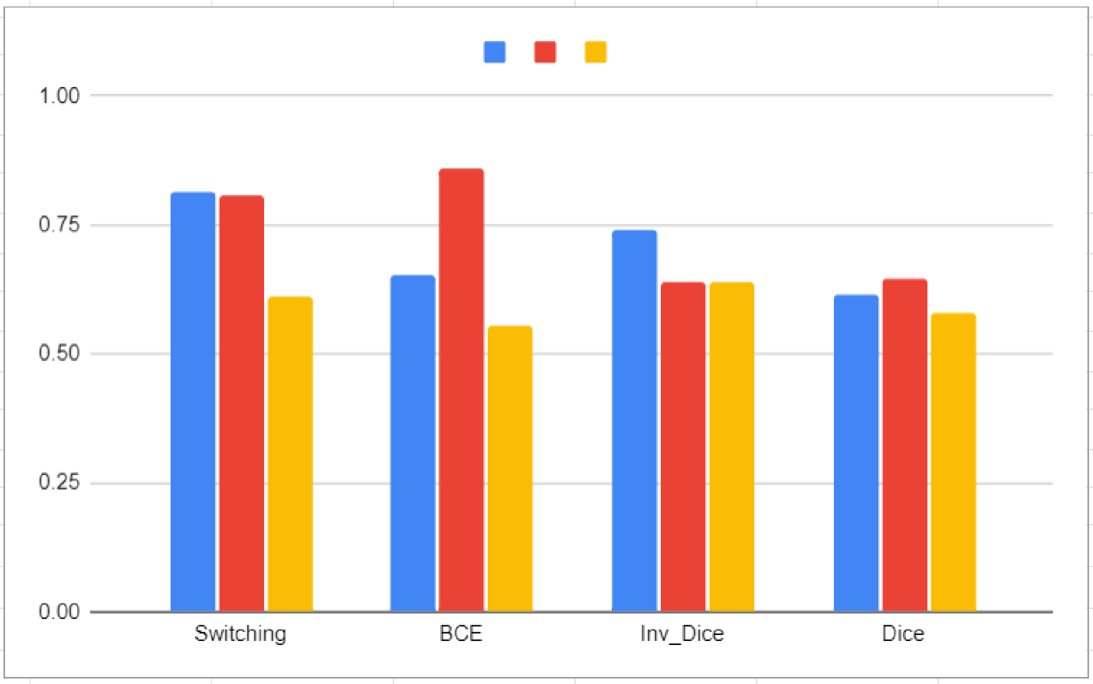
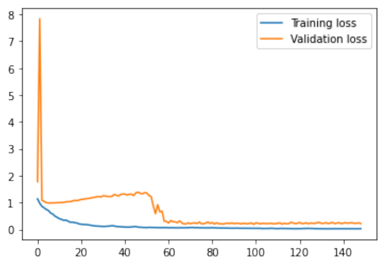
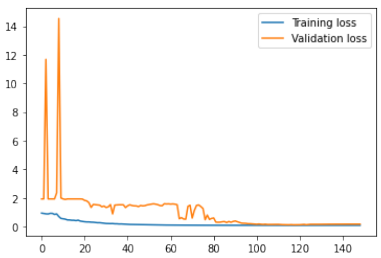


Figure 20: Various Losses for RMSProp, Adam, SGD Optimizers

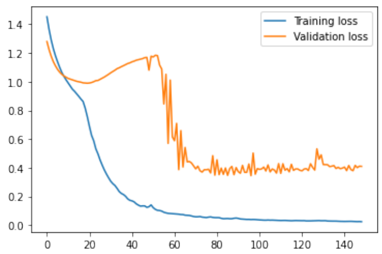
6.2 Results



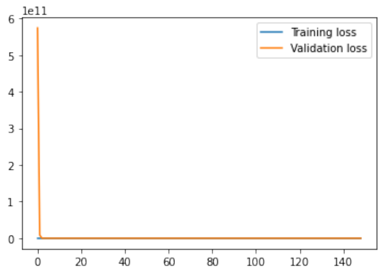
(a) Switching Loss Adam



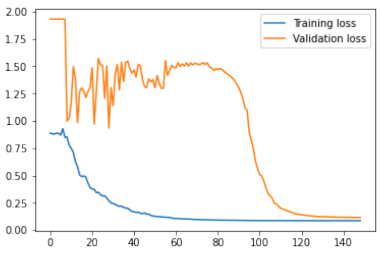
(b) Switching Loss RMS



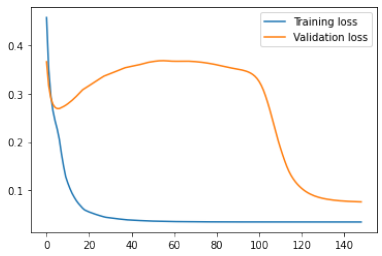
(c) Switching Loss SGD



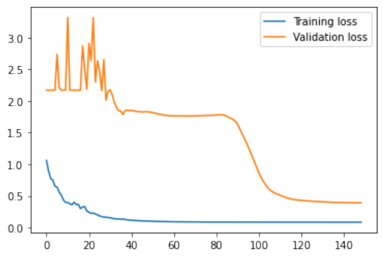
(d) BCE Loss Adam



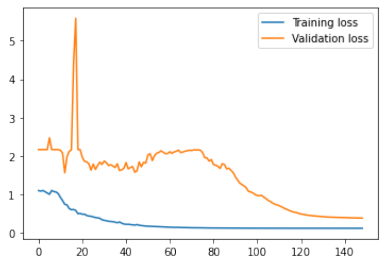
(e) BCE Loss RMS



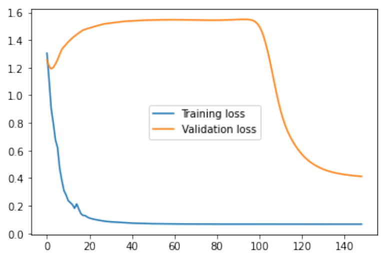
(f) BCE Loss SGD



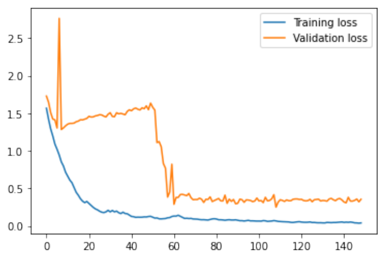
(g) Dice Loss Adam



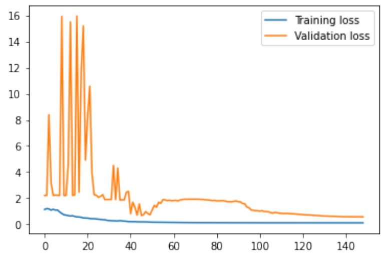
(h) Dice Loss RMS



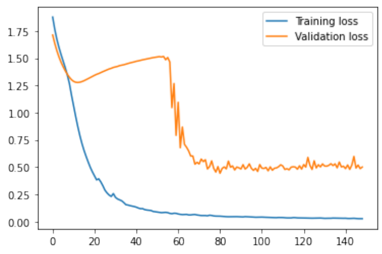
(i) Dice Loss SGD



(j) InverseDice Adam



(k) InverseDice RMS



(l) InverseDice SGD

Figure 21: Validation and Loss Curves for varying Optimizers and Loss Functions

6.3 Conclusion

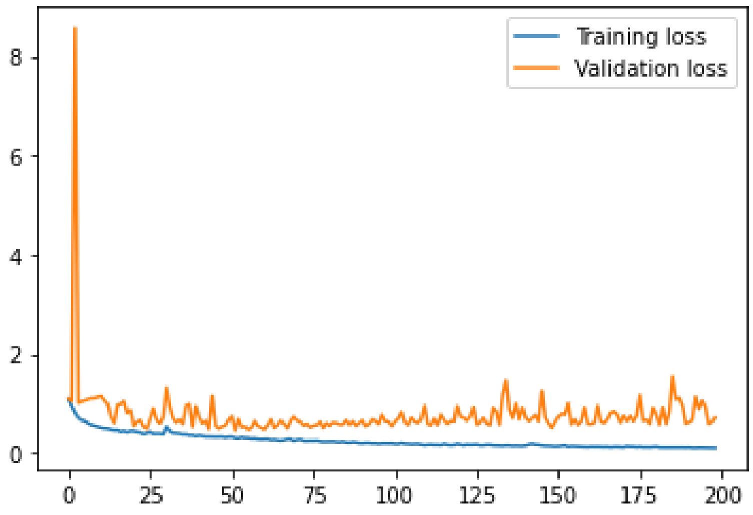
From the table and the graphs we can see that Switching loss works best overall. Next Inverse Dice loss and BCE loss works good in sequence. Dice loss works poorer than the above mentioned losses for the experiment.This can be concluded by looking at the Dice Coefficient values

However, RMSprop with ReduceLRplat and weight decay is the best on BCE (Binary cross entropy) loss producing a result of 0.8586 (which is very surprising), however following closely is Switching loss with Adam optmizer with 0.82, which is not surprising.

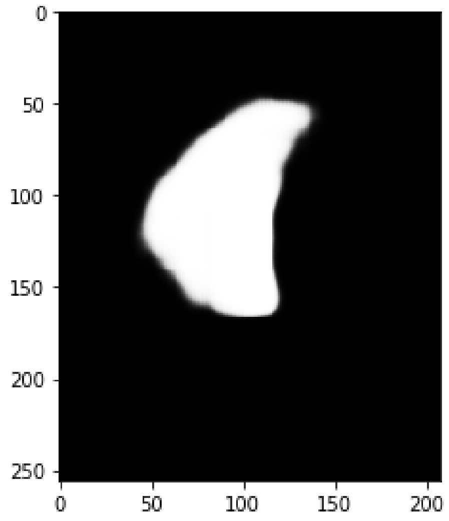
We note that introducing weight decay has significantly reduced performance of SGD while boosting performance of RMSprop on all the losses.

7 Data pre-processing

In the last section, we try to do some data pre-processing like image transformations and see if we can get better results. We have done random rotation with a range of 20 degrees, random horizontal flips and random vertical flips with a probability of 0.5 in the input training data and checking the output on transformed data aswell (we should also check it on non-transformed data). The results are attached below:



(a) Augmentation loss function

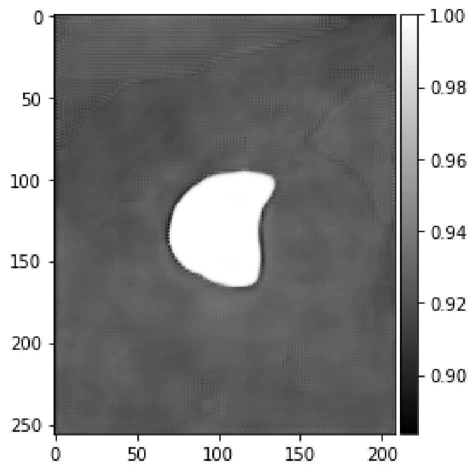


(b) Val output after augmentation

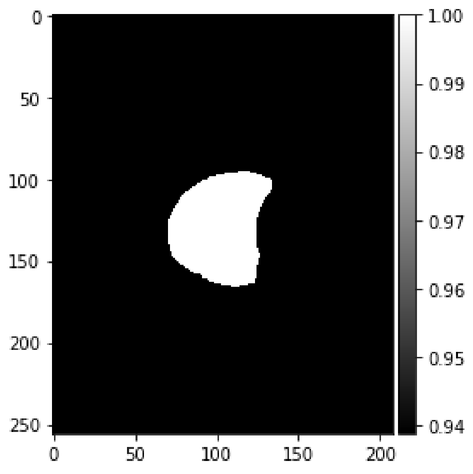
Figure 22: Plots and val output for augmented data

Note that in the above experiment, we have kept Adam(lr = 1e-3) on switching loss and weight decay = 0.01 (Although RMSProp performed better in one instance, Adam has been giving consistent results and hence checking on that). We can see that the model is not able to fit well on the validation data maybe it is because we have also applied transformation on the validation data (which ideally we shouldn't). It also can be because our model is already small and the parameters are not sufficient enough to fit the transformed data as it will be more complex but more general. We can train it for more number of epochs if given time or also increase the model size.

We had also applied histogram equalization on the target masks however, it produced very bad results which can be seen in the image given below (maybe because of lower range of targets):



(a) Val output after Hist-eq



(b) True val output

Figure 23: Val output after doing hist eq on target masks

8 Final Summary and Conclusive hyper parameters

We have conducted many experiments in the sections above to find out the optimum hyper-parameters for this problem. The overall strategy was to first fix the model then start from the best initial learning rates for each optimizer then to get best pair of optimizer and LR scheduler for a fixed loss. Then we tuned weight decay of each layer based on best optimizer, LR scheduler pair at best init learning rate. Next we found out if drop outs help or model. Now we have the best set of tools (on one loss curve) and we tested this tools on all the losses and found out the best loss function for our task.

We always went from a simpler and more general hyper-parameter to more complex yet more important hyper parameter. Generally, loss functions play a vital role in training while learning rates wont significantly affect if loss function and other parameters are good. Hence we kept optimizing which loss function to use at last. Even later we did augmentation as it can boost the output in some cases on any set of hyper parameter (however in our case it didnt as the model was small).

In conclusion we found out the best possible output val dice score was on **(BCE loss with RMSprop (lr = 5e-4), ReduceLRplat, weight decay = 0.01 and no dropout)** which gave a val dice score of 0.85. However, **(Switching loss with Adam(lr = 1e-3), ReduceLRplat, weight decay = 0.01 and no dropout)** gave a close score of 0.82 but each of it's component gave a more consistent result even with other hyper parameters and hence may perform better on different validation set. Hence conclude the latter set as the best hyper-parameters for this problem.

9 Refrences

1) For the loss functions and dice coefficients, we have taken inspiration from various internet sources like some kaggle sites on segmentation using Keras.

2) We have used Baseline UNET model (all the channels are divided by a factor of 4 as compared to the BASELINE model because our training task was simpler)