EE 782-AML
Assignment 1 Report

Rishabh Dahale 17D070008
Nitish Tongia 170010042
Shyam Thombre 170010023

November 2, 2020

1 Baseline Model

Our baseline model consists of Unet architecture. We have used a unet of depth 4. The Upconvolution is done using the ConvTranspose2d layers. The detailed architecture of the Unet is as shown

	Down Convolution 1	Down Convolution 2	Down Convolution 3	Down Convolution 4
# Filters	64	128	256	512
Activation	ReLU	ReLU	ReLU	ReLU
BatchNorm	After Activation	After Activation	After Activation	After Activation

	Up Convolution 4	Up Convolution 3	Up Convolution 2	Up Convolution 1
# Filters	512	256	128	3
Activation	ReLU	ReLU	ReLU	ReLU
BatchNorm	After Activation	After Activation	After Activation	After Activation

We have trained this baseline architecture on adam optimizer for 300 epochs. Loss chosen for the model was the average of dice loss and inverse dice loss. Weights were initialized by xavier initialization to keep the output variance low. We have used a batchsize of 32, with train-validation split of 80-20. Along with this we have used learning rate scheduler. We have used steplr as the scheduler. It reduced the learning rate by 90% after every 100 epochs. For training we used a simple average of dice loss and inverse dice loss as the net loss.

The results, comprising of the training loss, validation loss and our evaluation metric (Dice coefficient) are shown in the figure below. We can observe that both training and validation loss decrease as we increase the number of epochs and as expected the training loss decreases a bit more. Also the value of dice coefficient increases rapidly and then saturate for large number of epochs. This is also consistent with our resultant segmented images which keeps getting closer and closer to the actual ones.

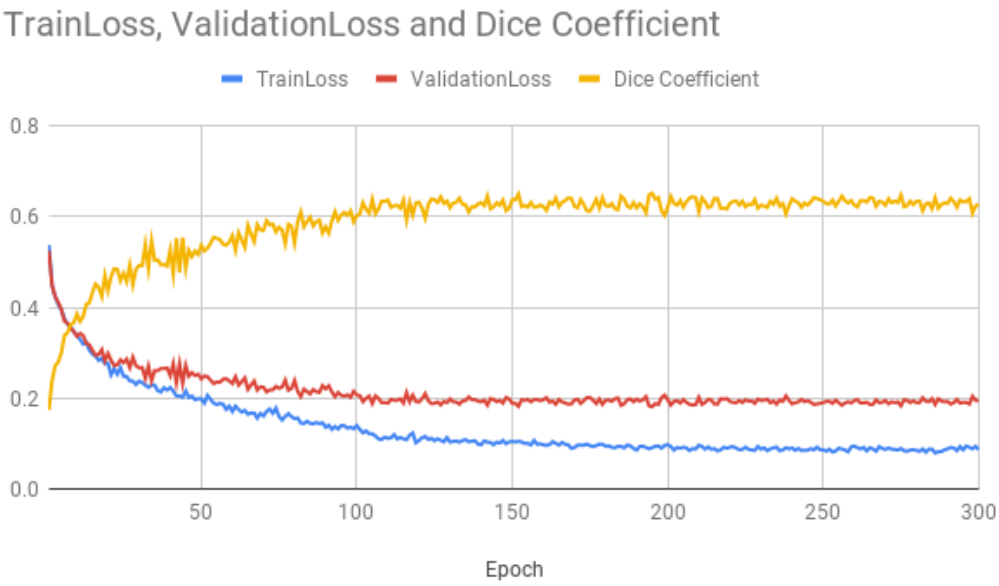
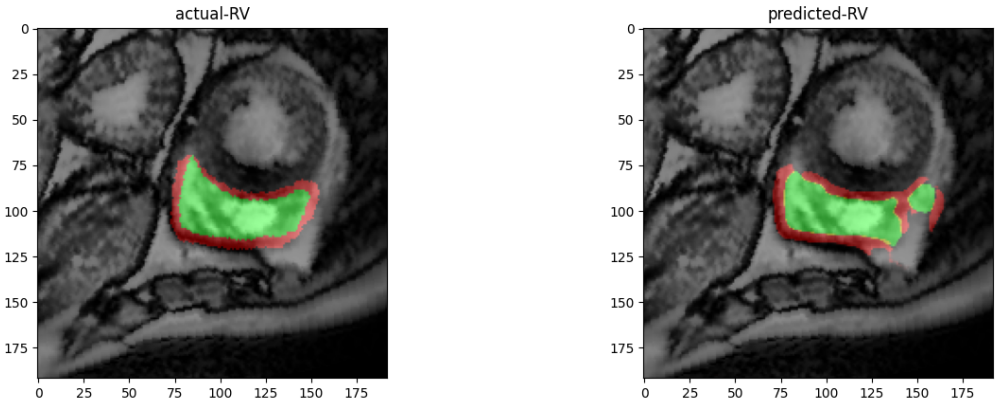


Figure 1: Training Loss, Validation Loss and Dice Coefficient



(a) RV Ground Truth (b) RV Predicted

Figure 2: Ground Truth and Predicted RV by baseline model

2 Data Prepossessing

As we have limited data, we have done some data transformations.

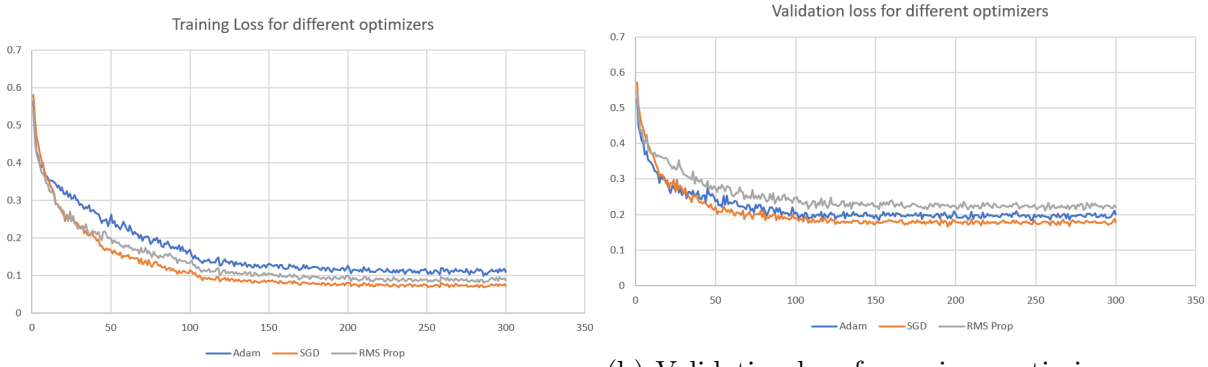
1. Random Rotate: The image is rotated with a probability of 0.5 by an random angle between -180 and 180. This helps the model to not to overfit on the training set and better generalize it

2. Random Crop: The image is randomly cropped to size of 192×192. This results in linear translation of the image helping model to better generalize

3. CLAHE: To improve the contrast of the images we have applied CLAHE

3 Optimizer

Every optimizer have a unique way to update the weights of the network. Some may use momentum, some may not. To get a better understanding of which optimizer is benefits us, we ran the experiment with 3 different algorithms: Adam, SGD and RMS Prop.



(a) Train loss for different optimizers

(b) Validation loss for various optimizers

Figure 3: Training and validation loss vs number of epochs for Adam, SGD and RMD Prop optimizers

The experiments were ran with same parameeters as that is baseline model except for the change of optimizer. SGD optimizer also allows us to use momentum while training. Momentum was set to 0.99 for SGD.
From the plots in figure 3 it can be seen that SGD gives a slightly faster and better convergence over both training and validation set. This is mainly because of the use of momentum. We had set the momentum value to 0.95. The initial learning rate was fixed at $2 \cdot 10^{-3}$ for all the experiments and step learning rate scheduler was used with step of 100 epochs (i.e. after every 100 epochs, the learning rate was reduced to 10% of it's value).

	Adam	SGD	RMS Prop
Dice Coefficient	0.617	0.657	0.579

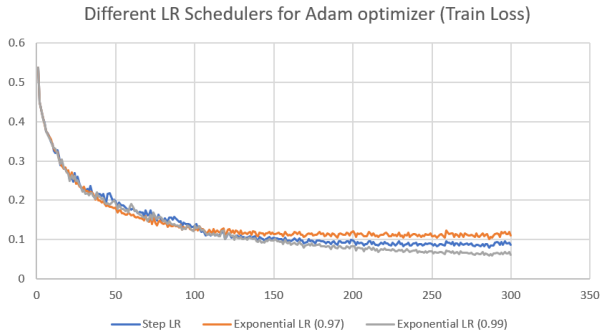
Table 1: Dice Coefficient for various optimizers

The final average dice coefficient over the validation set is shown in the table 1. It can be seen that SGD performs the best followed by Adam and then RMS prop. Hence we can say that for faster convergence, we can use SGD optimizer with a high value of momentum.

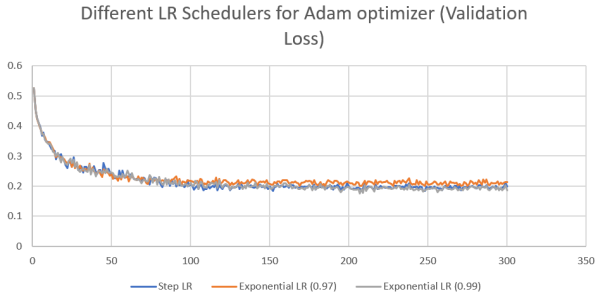
4 Learning Rate Scheduler

We ran experiments with 2 learning rate scheduler:

1. **Step LR**: This scheduler reduced the learning rate after a fixed number of epochs. This step size is fixed to 100 epochs
2. **Exponential LR** (0.97): Learning rate was reduced to 97% of the value in previous epoch for every epoch
3. **Exponential LR** (0.99): Learning rate was reduced to 99% of the value in previous epoch for every epoch



(a) Train loss for different LR Schedulers



(b) Validation loss for different LR Schedulers

Figure 4: Training and validation loss for different LR Schedulers

From 4 it can be seen that while training, exponential lr scheduler with decay of 0.97 works slightly better for initial epochs but soon starts to operate sub optimal. After first 100 epochs the learning rate which was $2 \cdot 10^{-3}$ initially would have become $2 \cdot 10^{-3} \cdot 0.97^{100} = 9.5 \cdot 10^{-6}$ for exponential lr with decay of 0.97, $2 \cdot 10^{-3} \cdot 0.99^{100} = 7.3 \cdot 10^{-4}$ for exponential lr with decay of 0.99 and $2 \cdot 10^{-4}$ for step lr. As it can be seen that exponential LR with decay of 0.99 provides a comparatively slower decay of the LR, the direct impact can be seen in the training loss. For validation loss, if we compare the step lr and exponential LR (0.99 decay) loss curves, step LR is slightly above the exponential indicating that the exponential LR helped in better training. This can also be seen from the average dice coefficient for the validation set shown in table 2

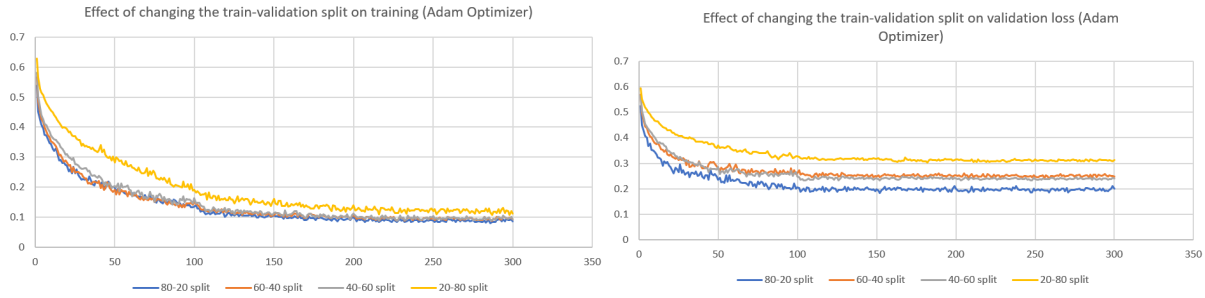
	Step LR	Exponential LR (0.97)	Exponential LR (0.99)
Dice Coefficient	0.617	0.591	0.643

Table 2: Dice Coefficient for different LR schedulers

5 Training Data

5.1 Optimizer

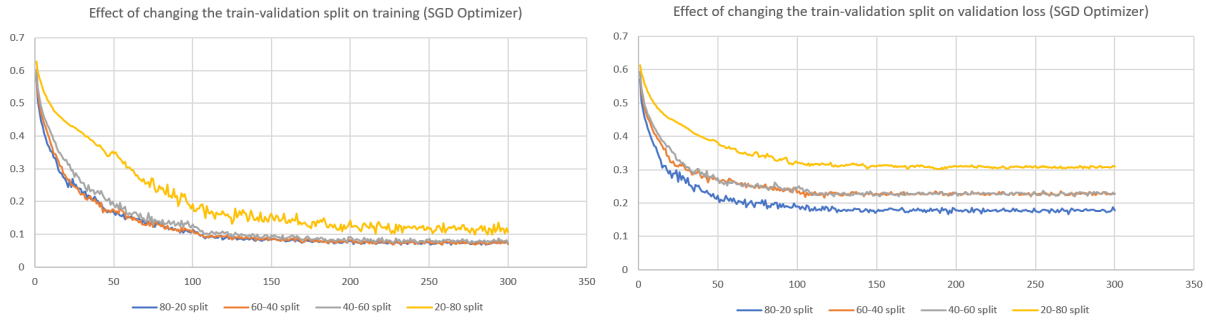
As we decrease the training data, the it is expected for validation loss to increase and for the model to overfit on the training set leading to poorer generalization of the model. This can be evidently be seen in the figure 5



(a) Train loss

(b) Validation loss

Figure 5: Variation of the training and validation loss as the training data varies for Adam optimizer. 80-20 split indicates 80% training data and 20% validation data.

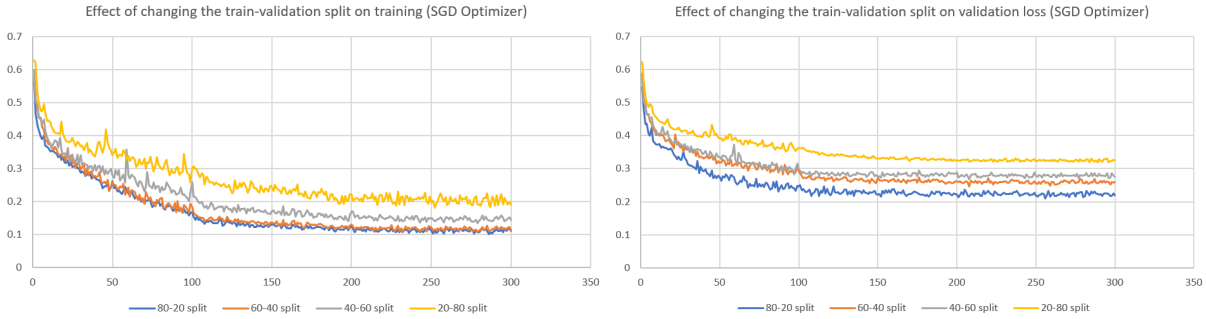


(a) Train loss

(b) Validation loss

Figure 6: Variation of the training and validation loss as the training data varies for SGD optimizer. 80-20 split indicates 80% training data and 20% validation data.

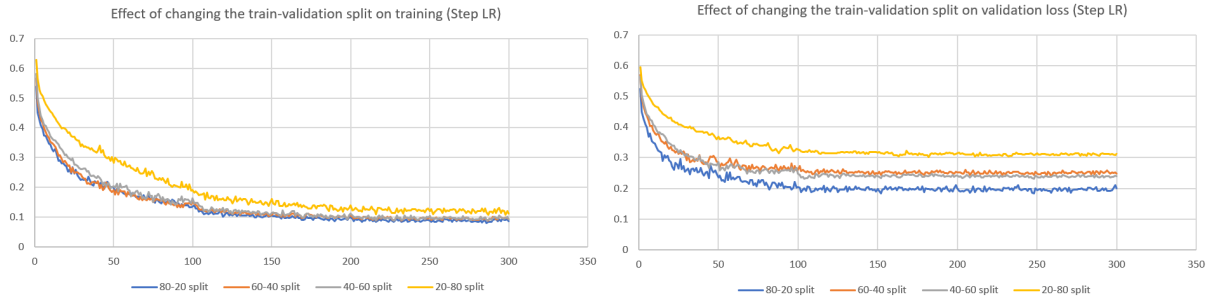
If we look at the losses in 5 6 and 7, the RMS Prop is the most affected by changing the number of training points. In SGD and Adam decreasing the training set from 60% to 40% incurs very small or almost no penalty in the validation whereas for RMS Prop a huge penalty can be seen. Moreover if we look at the train loss, it can be seen that for SGD and Adam the speed of learning the training set is not much altered except got 20-80 split, where as for RMS Prop we can see a gradual decrease in the learning speed as the training st decreases.



(a) Train loss (b) Validation loss

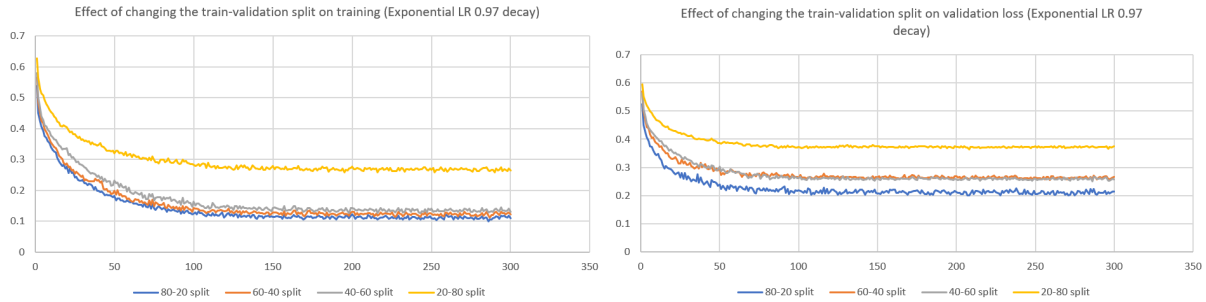
Figure 7: Variation of the training and validation loss as the training data varies for RMS Prop optimizer. 80-20 split indicates 80% training data and 20% validation data.

5.2 LR Scheduler



(a) Train loss (b) Validation loss

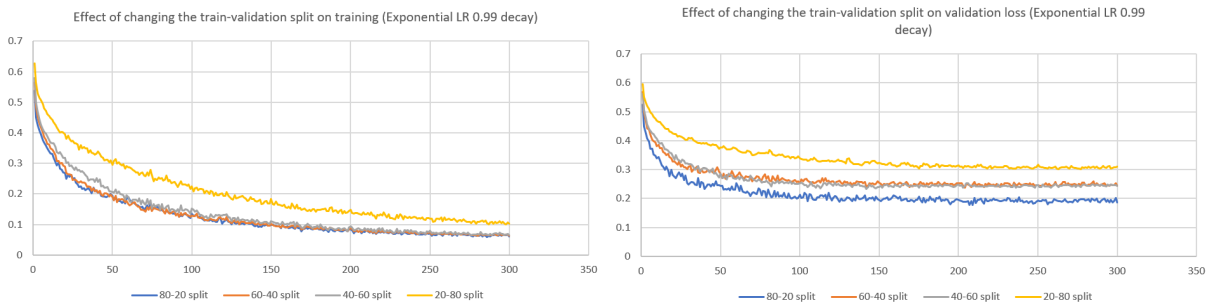
Figure 8: Variation of the training and validation loss as the training data for Step LR scheduler. 80-20 split indicates 80% training data and 20% validation data.



(a) Train loss (b) Validation loss

Figure 9: Variation of the training and validation loss as the training data for exponential LR with decay of 0.97. 80-20 split indicates 80% training data and 20% validation data.

From the plots in 8, 9 and 10, it can be seen that exponential lr with decay of 0.97 gives a degraded performance as the training set decreases. This can be seen directly in the



(a) Train loss

(b) Validation loss

Figure 10: Variation of the training and validation loss as the training data for exponential lr with decay of 0.99. 80-20 split indicates 80% training data and 20% validation data.

validation loss plot. For exponential lr with decay 0.99 it can be seen that for 20% training data the model was still able to learn and reduce training loss, implying that for exponential lr with decay 0.97, the model could not learn the training data properly.

6 Weight Decay and Dropout Probabilities

In this section, we wish to explore the effects of varying weight decay and dropout probabilities on the learning rate of our model and the change in dice coefficient values. We studied this variation for a fixed adam optimizer, steplr learning rate scheduler with initial learning rate $2\text{x}10^{-3}$ and a learning rate decay of 100, and trained the model on 300 epochs with a validation fraction of 0.2.

6.1 Variation in Weight Decay

We try to study the behaviour of our model with varying weight decays. For this we observed the variation in learning rate of our model. The figure below shows the variation in training loss for our model with varying weight decays where the curve Trainloss-w represents the training loss with weight decay = w . We can observe from the plot as we varied weight decay from as low as 0.0001 to 0.1, there isn't much difference in the learning rate as the training loss follows a similar path as the one with no weight decay.

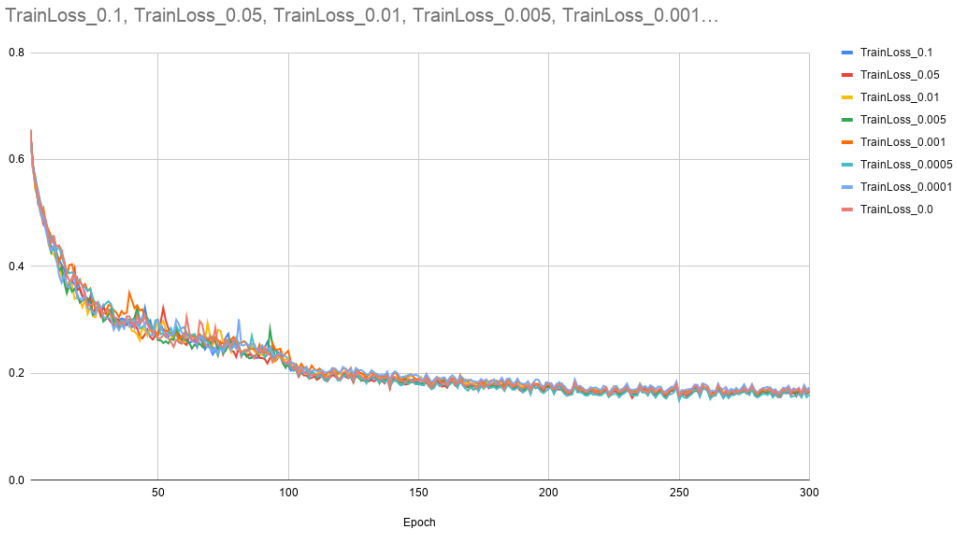


Figure 11: Training Loss for Different Weight decay

A similar trend is observed for the validation loss as shown in figure below where there is not much difference in the learning rate with varying weight decay values. Here again ValdationLoss-w represents the validationloss with weight decay = w varying between 0.0001 to 0.1.

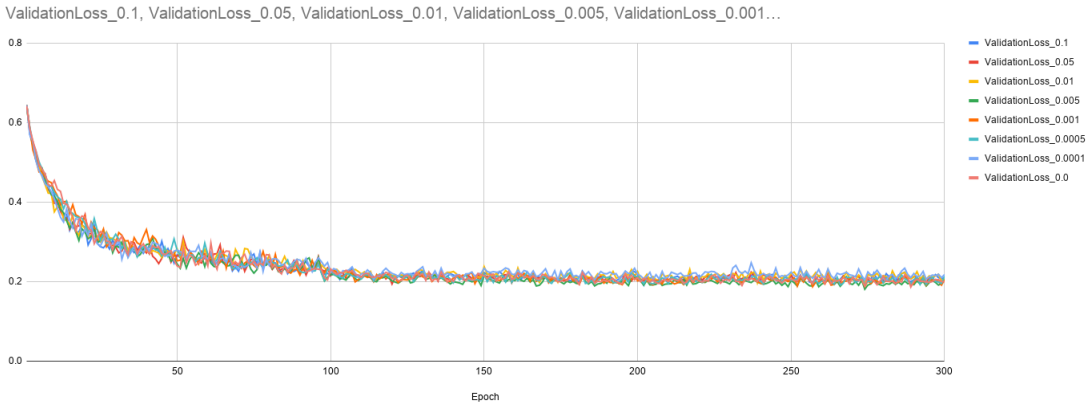


Figure 12: Validation Loss for Different Weight decay

Although the loss curves does not show that much variation, we can see from the resulted segmented masks that as the weight decay is increases, the masks keeps on getting more and more deformed. This can be explained on the basis that we use an average of dice and inverse dice loss as our loss function which does not give a very high penalty for false positives and hence our the shape of segmented images gets distorted for increasing weight decays. The results of our evaluation metric (Dice Coefficient) also shows similar results as shown in the table below where there is not any specific trend in the dice coefficient values.

Weight decay (w)	0.1	0.05	0.01	0.005	0.001	0.0005	0.0001	0
Dice Coefficient	0.7348	0.7238	0.7323	0.7338	0.7136	0.7160	0.7440	0.7282

6.2 Variation in Dropout Probabilities

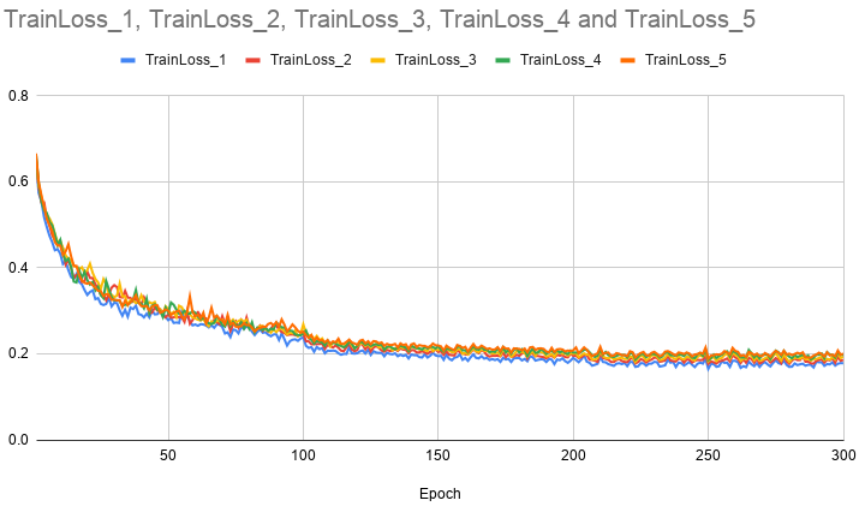


Figure 13: Training Loss for Different dropout probabilities

We added a dropout layer in the down-convolution part of our model and tried to study the variation of dropout probabilities of that layer and observe its effect on the learning rate. The figure below shows the variation in training loss for our model with varying dropout probabilities where the curve Trainloss-p represents the training loss with dropout probability = p . We can observe from the plot as we varied the probabilities from 0.1 to 0.5, there isn't much difference in the learning rate as the training loss follows a similar path for all dropout probabilities.

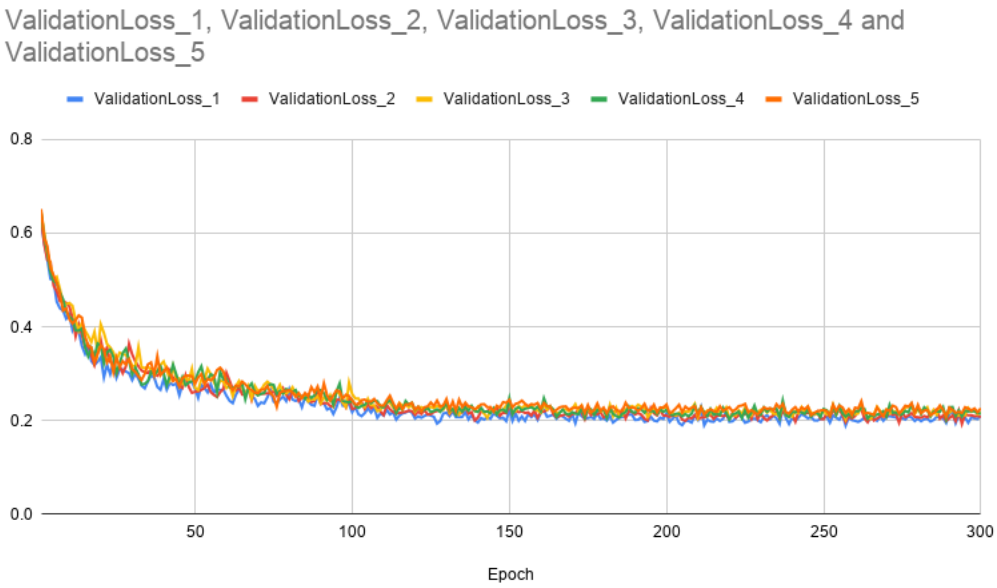


Figure 14: Validation Loss for Different dropout probabilities

A similar trend can be seen for the validation loss as shown in figure below where again there is not much variation in the learning rate as we vary the dropout probability values. Here again ValdationLoss-p represents the validationloss with dropout probability = p varying between 0.1 to 0.5.

Although the loss curves does not show that much variation, we can see from the resulted segmented masks that as the dropout probabilities increases, the masks keeps on getting more and more broken or incomplete. This is because of the fact that the learning is reduced as we increase the dropout. The results of our evaluation metric (Dice Coefficient) also shows similar results as shown in the table below where we can see that the dice coefficient value reduces as we increase the dropout probabilities.

Dropout Probabilities (p)	0.1	0.2	0.3	0.4	0.5
Dice Coefficient	0.7262	0.7226	0.7028	0.7042	0.6793

7 Loss Functions

We tried two different types of losses taking inspiration from the given paper. Ones is the weighted combination of binary cross-entropy (BCE) loss, dice loss, and inverse dice loss. However, even after trying and debugging for lot of time, the output was a complete black image. We concluded that the source for the problem was the range difference of dice, inverse dices losses and that of BCE loss summed up over the entire image.
So, going further from here we removed the BCE loss, and took a weighted combination of dice and inverse dice loss.
The exact loss we used was:

$$L_{combined} = \lambda L_D + (1 - \lambda) L_I$$

where, L_D and L_I denote the dice and inverse dice losses respectively.
Notice that, $\lambda = 1$ is pure dice loss and $\lambda = 0$ is pure inverse dice loss. We experimented with $\lambda = 0, 0.25, 0.4, 0.6, 0.75, 1$. The case of $\lambda = 0.5$ is the baseline case.

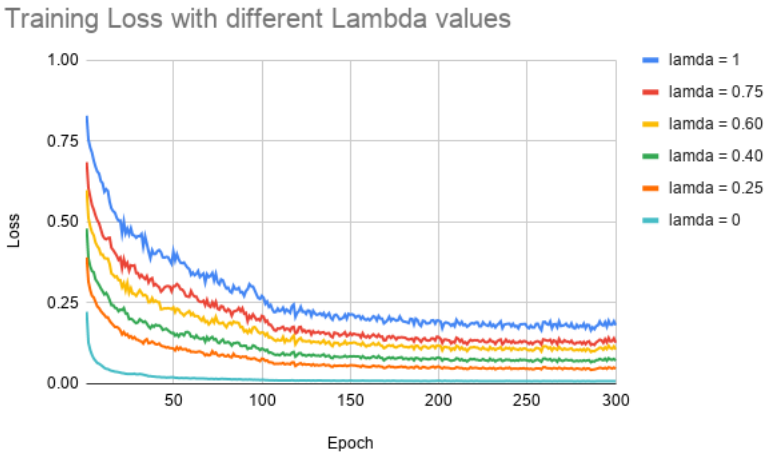


Figure 15: Training Loss using different values of λ

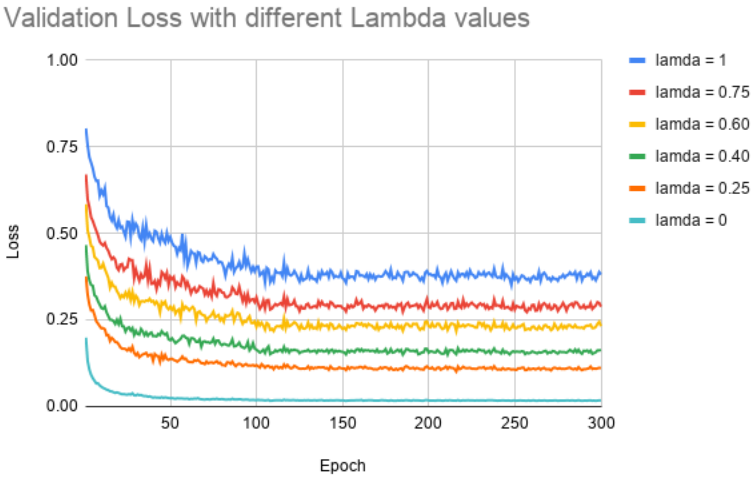


Figure 16: Validation Loss using different values of λ

As can be observed, the inverse dice loss has in general a lower value, which is as expected. This also can be seen by the trend that as λ decreases, the loss curves shift downwards. The images corresponding to these losses for different epochs can be seen in "Losses" folder in the drive link *here*. A λ value of 0.6 and 0.75 seemed to work good for the task. However, pure inverse dice works very poorly, as can be seen in Table 3. This is because it tries to match

the image output with the background black pixels, which very high in number compared to foreground pixels.

Lambda values	1	0.75	0.6	0.4	0.25	0
Dice Coefficient	0.619	0.622	0.625	0.618	0.61	0.53

Table 3: Dice Coefficient for different λ values for loss

8 Mode of Upsampling

We tested with two different types of upsampling techniques.

1. ConvTranspose2d
2. Upsampling (using 'bilinear' interpolation) + Conv 1x1

There is quite a bit of difference in the approaches of ConvTranspose2d, and Upsample layer followed by Conv 1x1 for the purpose of upsampling from the latent representation. However, as can be seen form the graphs below, and Table **??**, there isn't much difference between using the two.

Figure 17: Training Loss when changing the mode of upsampling

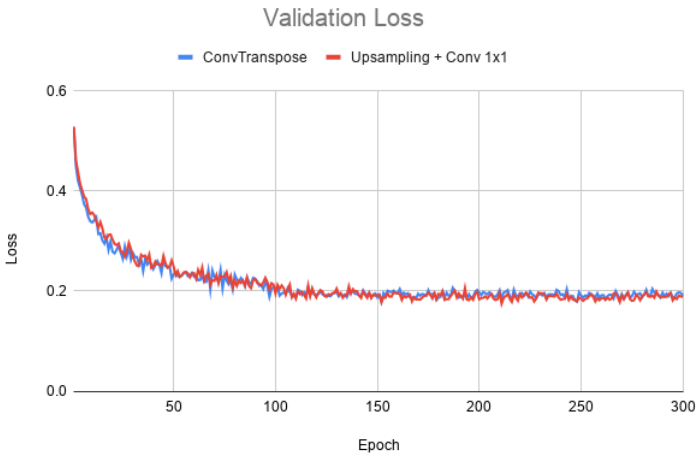


Figure 18: Validation Loss when changing the mode of upsampling

The dice coefficients obtained by training using these modes of upsampling are

Mode	ConvTranspose2d	Upsampling + Conv 1x1
Dice Coefficients	0.634	0.639

Table 4: Dice Coefficient for different modes of upsampling

9 L1 Regularization

We also tried to test what are the results obtained using a L1 regularization. The regularization was applied using all the weights of the model. While weight decay didn't change the performance much, L1 regularization actually degraded the performance of the baseline model. From this, we deduced this might be an indication of some high correlation among the weights of the model.

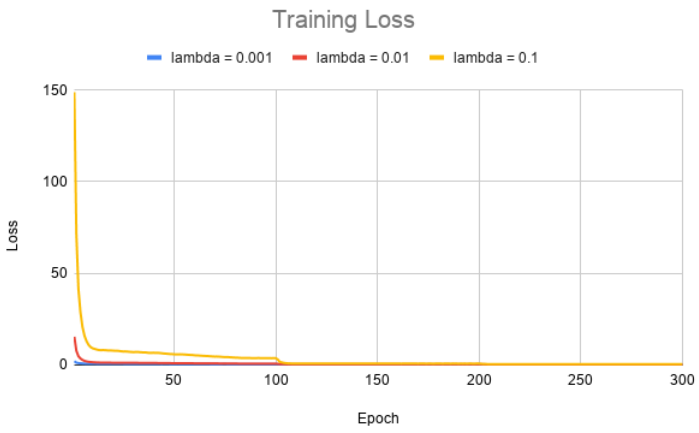


Figure 19: Training Loss for L1 regularization with different values of λ

As can be seen, depending on the initialization of weights, using high value of λ for L1 regularization, leads to high initial loss. However, it decays very fast. Let's concentrate on the steady state values.

Figure 20: Training Loss for L1 regularization with different values of λ after 106 epochs

Here, the degradation due to L1 regularization is very clear. This, can also be observed in the segmentation masks generated by the model for this regularization in the "L1 regularization" folder *here*. The validation loss for these models is shown below, which follow similar trend as that of the training loss.

Figure 21: Validation Loss for L1 regularization with different values of λ

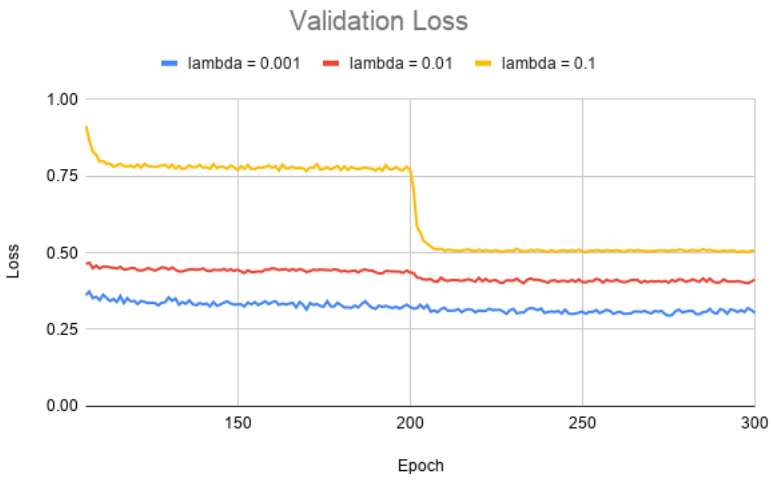


Figure 22: Validation Loss for L1 regularization with different values of λ after 106 epochs

10 L1L2 Regularization

After trying L1 and L2 regularization separately, we also tried the combination of the two

$$\text{total regularization} = \lambda_1(\text{L1-regularization}) + \lambda_2(\text{L2-regularization})$$

Using this format of regularization confirmed the hypothesis that the L1 regularization actually degraded the performance while L2 regularization didn't affect the performance much. We varied λ_1 and λ_2 as 0.001, 0.01, 0.1 and looked at the following combinations.

Figure 23: Training Loss for L1L2 regularization with different values of λ

Similar to previous case of L1 regularization, we see that the initial loss is very high, owing to the initialization of the weights. So looking at these curves after 106 epochs we get

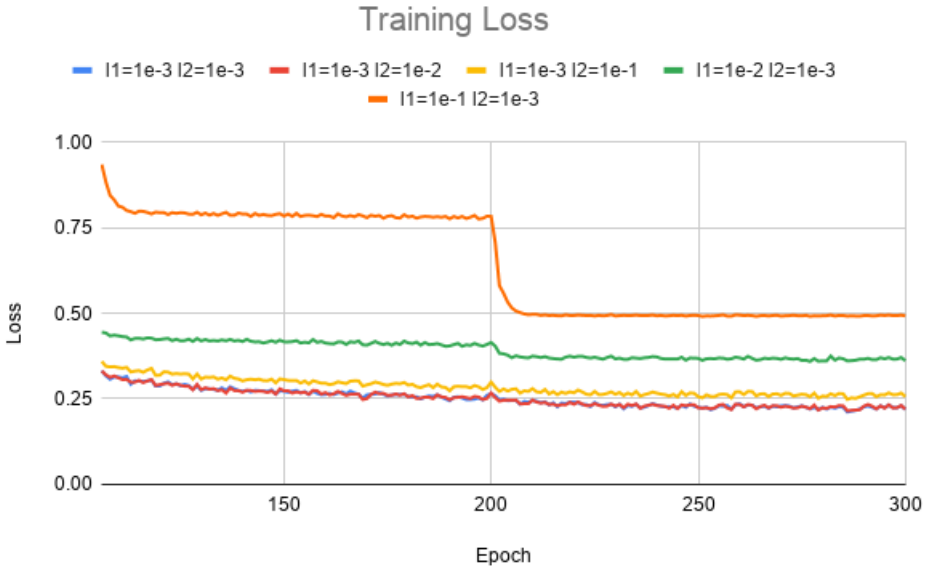


Figure 24: Training Loss for L1L2 regularization with different values of λ after 106 epochs

The validation loss curves for these models are shown below.

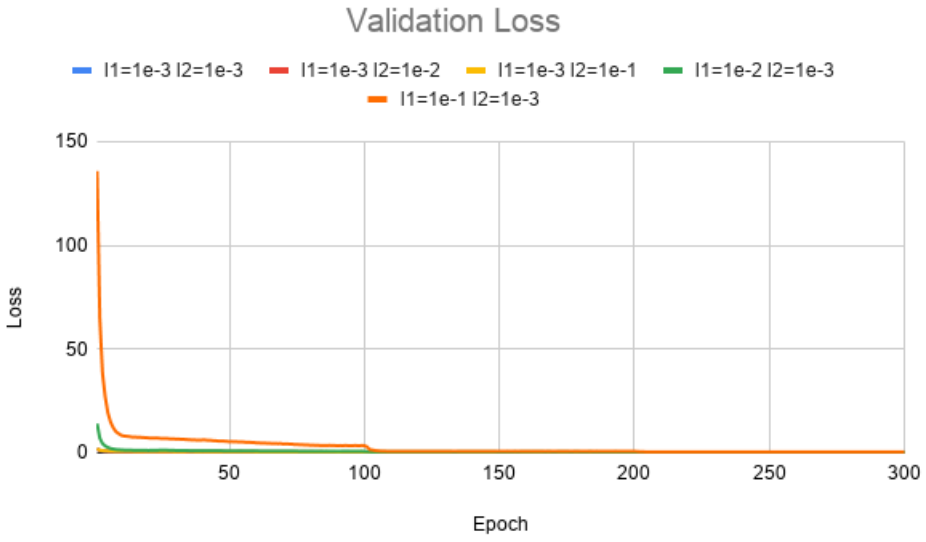


Figure 25: Validation Loss for L1L2 regularization with different values of λ

Figure 26: Validation Loss for L1L2 regularization with different values of λ after 106 epochs

As can be clearly seen, the L1-regularization degrades the performance much more than the L2-regularization. This comparison is important as it highlights the pros and cons of using L1 and L2 regularization, in a quantitative manner. For the qualitative comparison, the segmentation outputs can also be seen in the "L1L2 regularization" folder *here*.

11 Conclusion

In this assignment we have tested many different variations over a given baseline model to evaluated how the different hyper-parameters, losses, optimizers, and learning rate schedulers affect the performance of a model, by using the evaluation metric, dice coefficient. These experiments gave a great insight into the impact of various changes mentioned above, which helps better understand a problem statement and the corresponding model. It also highlights the challenges in medical image segmentation tasks, and gives the foundation and requirement for research in this domain.