Neural Computation Project Report - Group 7

Introduction

The aim of the project is to train a convolutional neural network for the task of semantic segmentation of Magnetic Resonance Images. Semantic segmentation is the process of assigning pixel wise labels to an image, such that each pixel represents a particular feature/class that we wish to segment. In our work, we train a model to segment the left ventricle, right ventricle and myocardium, with an additional class for background pixels.

The dataset used contains a total of 200 images, which is split into a training set (100 images), a validation set (20 images) and a test set (80 images). The input data is a grayscale image of dimension 96x96, and our label is an encoded mask where each pixel contains the value of the class to which it belongs.

Implementation

- **Model Architecture :**

For this task we implemented a Convolutional Neural Network based UNet architecture, which is lightweight and used commonly for medical image segmentation. The model consists of three main blocks; an encoder that progressively downsamples the data in stages, a bottleneck to decrease our data dimensionality, and a decoder block to upsample the encoded feature vector back to the original image dimension.

Each of the three main blocks can be summarised as follows:

Encoder: The encoder downsamples the data in 3 stages, where each stage consists of a set of repeated operations of Convolution + Batch Normalization + Activation, followed by a pooling layer. Each convolution operation uses a 3x3 filter with a stride of 1 and padding of 1. The activation function used is leaky relu since it allows a small negative gradient to flow through our weight update. The pooling layers are followed with a dropout layer (p=0.2) that helps us better generalize our trained weights by reducing overfitting.

Bottleneck: The bottleneck layer consists of a single 3x3 convolution followed by batch normalization and a leaky relu activation. The use of the bottleneck is to constrain the model to learn a more compressed version of our input data.

Decoder: The decoder upsamples the data in 3 stages similar to the encoder, with an added filter concatenation step. At each step, we concatenate the output from the corresponding block of our encoder to allow us to carry forward lower level features from our network. A transposed convolution (kernel size=2, stride=2) is used to perform upsampling since it allows us to perform learnable upsampling of the data. The final layer of the decoder is a convolutional layer with a 1x1 filter size to remap our data to match our desired channel output size of 4.

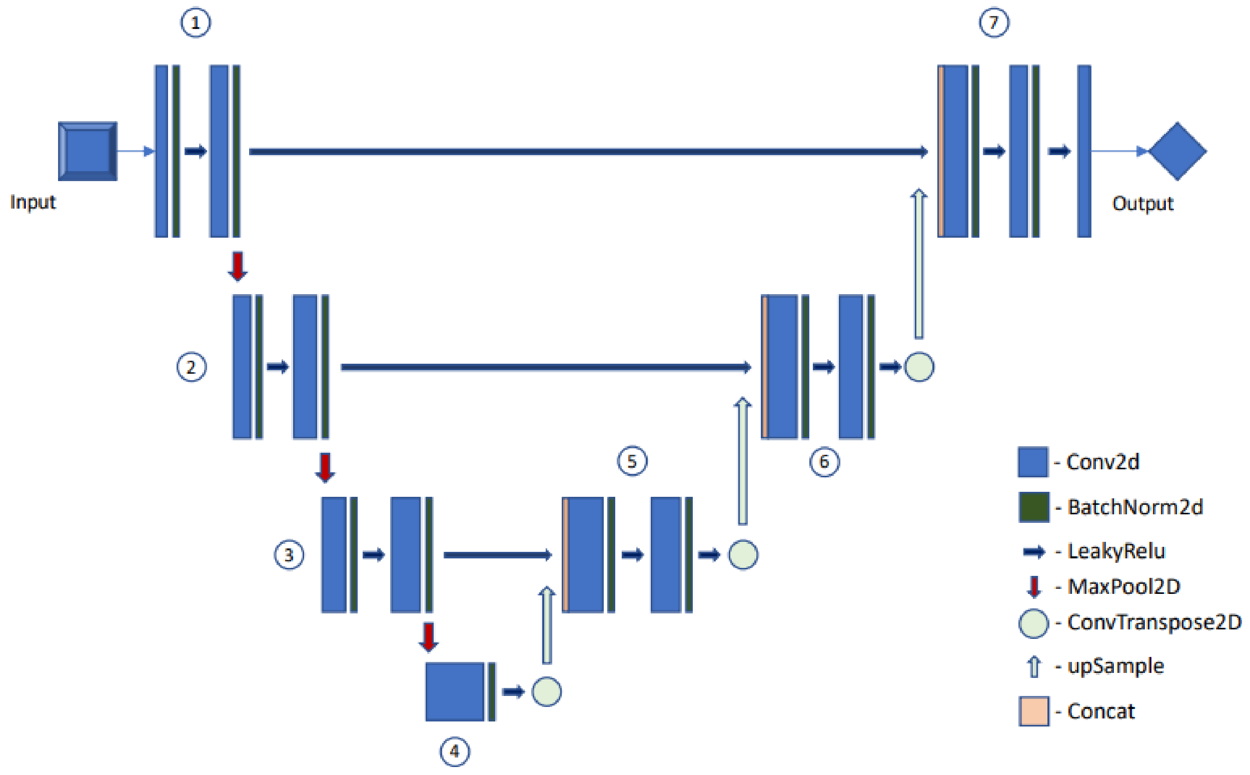


Figure 1 UNet architecture

- **Optimizer:**

We used the Adam optimizer to train our model since it offers an adaptive technique of finding the minima with a lower number of iterations as compared to other techniques like Stochastic Gradient Descent. The optimizer was initialized with a learning rate of 1e-3, and a L2 weight regularization of 1e-4. The weight decay term adds an additional penalty to allow us to minimize our weights and reduce overfitting.

- **Early Stopping:**

To prevent our model from overfitting to training data, we made use of an early stopping technique that keeps track of the validation loss of our model and ends the training when a stop condition is hit. For each epoch the best validation loss is saved, and compared with the loss of the next epoch. If the validation loss of the current epoch is lower than the best loss, a counter is updated. The training is ended when the counter reaches a predefined value(patience=**5**). This prevents us from training a model that overfits to the train set rather than learning a generalized model. This also removes the necessity of tuning an additional parameter for the number of epochs, since the training gets halted based on calculated metrics.

- **Loss Function:**

The loss function chosen to optimize the model plays an important role in determining the kind of features our model learns, and the overall quality of the predictions. The figure below contains a depiction of the percentage of pixels contained in each individual class over the overall dataset. Class 0, which is the background, contains 82.35% of the pixels, whereas the remaining 3 classes contain only a total of 16.65% of the pixels. The chosen loss function should therefore be able to deal with this large class imbalance.

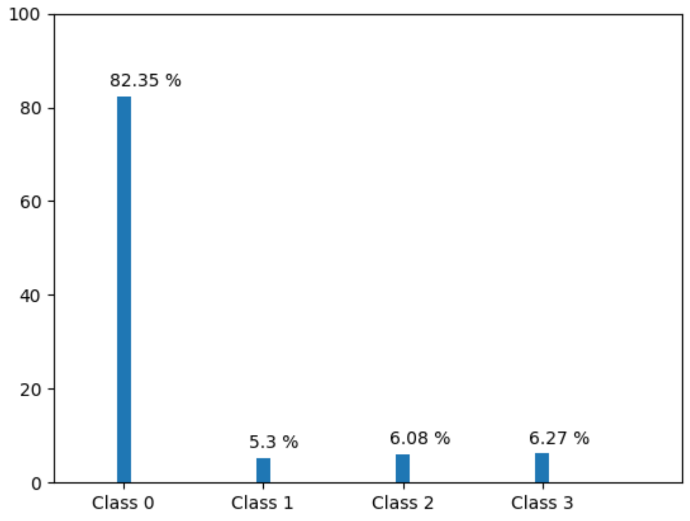


Figure 2. Percentage of each class in overall dataset

Dice-Focal loss: To deal with the issue of class imbalance, we make use of a combination of dice and focal loss during the training which allows the model to optimize to the task of pixel wise segmentation, and maximum overlap between train labels and predictions. The use of focal loss is to enforce an additional weight term on the calculated loss based on the number of size of each class in the dataset. The choice of loss function is explained in more detail in the section below.

Experimentation

- **Loss Function:** We have experimented with a combination of loss functions listed below, and evaluated their results using the validation metrics obtained.
    1. Cross Entropy: A simple class wise categorical cross entropy function which performs a pixelwise loss by maximizing the value of the true class. The torch cross entropy function used in the code performs softmax followed by an nll log loss to compute the cross entropy loss.

$$Loss = -\sum_{n=1}^{N} \sum_{m=1}^{num\ classes} ylog(\hat{y}) + (1-y)log(1-\hat{y})$$

2. Weighted Cross Entropy: The cross entropy loss calculated for each class is weighted by a factor called the effective number of samples. This factor is calculated using the normalized number of pixels in each class. We initialized the weights for each class using the calculated percentages calculated from Figure 2.

$$\mathbf{weight} = \frac{(\mathbf{1}-\beta)}{(\mathbf{1}-\beta^{\mathbf{n}})}$$

$$\mathbf{Loss} = -\sum_{\mathbf{n=1}}^{\mathbf{N}} \sum_{\mathbf{m=1}}^{\mathbf{num\ classes}} \mathbf{weight} * (\mathbf{ylog\hat{y}} + (\mathbf{1}-\mathbf{y})\mathbf{log}(\mathbf{1}-\hat{\mathbf{y}}))$$

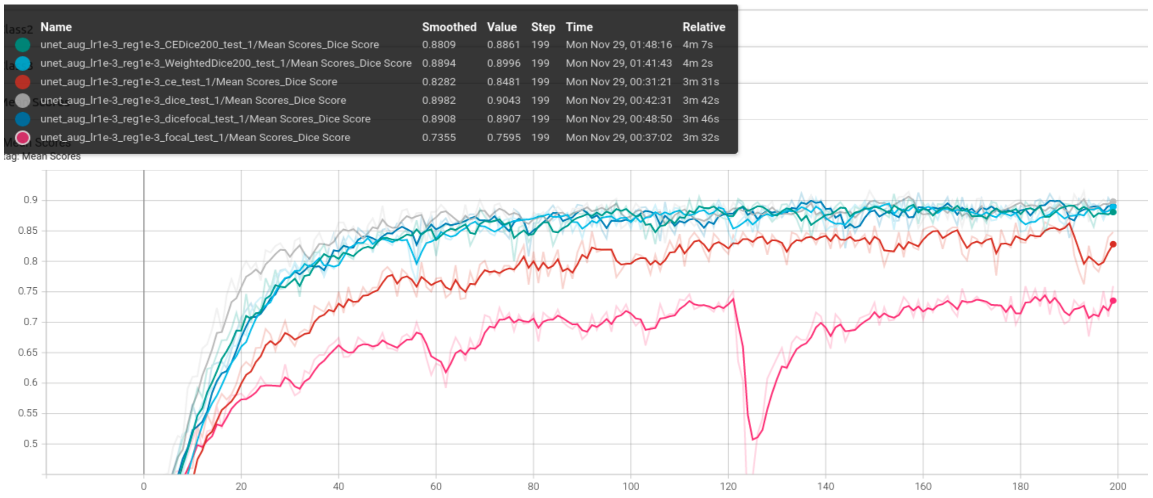
3. Focal loss: Focal loss is another technique of weighted cross entropy that tries to handle two issues pertaining to class imbalance and classifying difficult examples. The term $(1 - p_t)^{\gamma}$ penalizes the predictions based on difficulty(background pixels are considered to be easy), and the alpha term is used as a weight factor for cross entropy. The value gamma is a modulating factor that is chosen such that γ >= 0.

$$\mathbf{Loss} = -\alpha(\mathbf{1}-\mathbf{p})^{\gamma}\mathbf{log}(\mathbf{p})$$

4. Soft Dice loss : Soft Dice loss is based on the dice coefficient that is used to measure the amount of overlap between two masks. The dice score is calculated to be twice the intersection, divided by the total area covered by both masks. The score ranges from [0,1], where the dice score of 1 represents the perfect overlap. The loss function is therefore formulated to be 1 - dice score, which constrains the model to learn a mapping that maximizes the intersection of the predicted mask and ground truth.

$$\mathbf{Loss} = \mathbf{1} - \frac{\mathbf{2} * (\mathbf{pred}\ \cap\ \mathbf{ground\ truth})}{(\mathbf{pred}\ +\ \mathbf{ground\ truth})}$$

We implemented combinations of these loss functions such as Dice-Focal, CE-Dice and Weighted CE Dice loss, and made comparisons in tensorboard for accuracy, IOU and dice score. The comparisons were made using both a mean score over all classes, and individual class performance. The Dice-Focal loss function was chosen since it gave us the best results over both the mean class, and individual class scores.

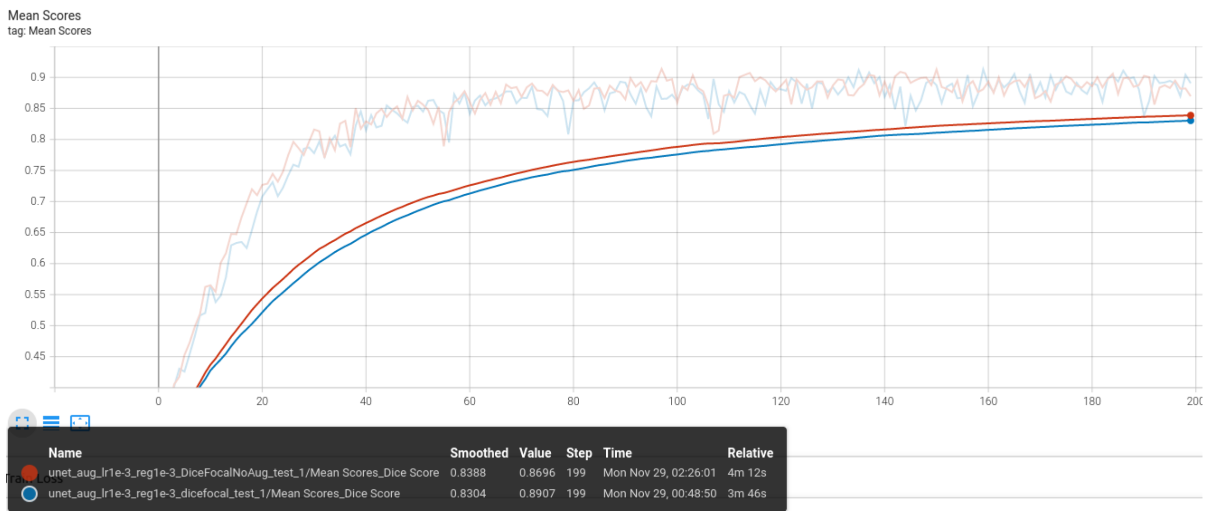


	Name	Smoothed	Value	Step	Time	Relative
●	unet_aug_lr1e-3_reg1e-3_CEDice200_test_1/Mean Scores_Dice Score	0.8809	0.8861	199	Mon Nov 29, 01:48:16	4m 7s
●	unet_aug_lr1e-3_reg1e-3_WeightedDice200_test_1/Mean Scores_Dice Score	0.8894	0.8996	199	Mon Nov 29, 01:41:43	4m 2s
●	unet_aug_lr1e-3_reg1e-3_ce_test_1/Mean Scores_Dice Score	0.8282	0.8481	199	Mon Nov 29, 00:31:21	3m 31s
●	unet_aug_lr1e-3_reg1e-3_dice_test_1/Mean Scores_Dice Score	0.8982	0.9043	199	Mon Nov 29, 00:42:31	3m 42s
●	unet_aug_lr1e-3_reg1e-3_dicefocal_test_1/Mean Scores_Dice Score	0.8908	0.8907	199	Mon Nov 29, 00:48:50	3m 46s
●	unet_aug_lr1e-3_reg1e-3_focal_test_1/Mean Scores_Dice Score	0.7355	0.7595	199	Mon Nov 29, 00:37:02	3m 32s

Figure 3 Comparison of validation metrics for loss functions

- **Data Augmentation:** Data Augmentation is used to account for variations that may be present in our real-world data that may not exist in our training set. Convolutional neural networks require huge amounts of data covering a wide variety of variations to be able to perform accurately at test time. Since our dataset contains only 100 training images, we make use of random data augmentation to help us synthetically increase the size of our training data. This allows the model to better generalize to the task, and helps us to obtain much better results on our validation set. Since heavy augmentation may misrepresent the anatomical structure of the heart[2], we instead use only light data augmentation techniques such as random rotation and flipping of the images. For each training image, 1 of 4 possible combinations is used :
    1. No Augmentation: No Augmentation is applied to the image.
    2. Horizontal Flip: The image is flipped horizontally along the y axis.
    3. Vertical Flip: The image is flipped horizontally along the x axis
    4. Random Rotation: The image is rotated randomly between an angle of -10 to +10 degrees.

  We compared the model's performance with and without augmentation using our validation metrics(for individual class, and mean score) below, and observed on average an increase in accuracy of around 2-3% depending on the chosen model parameters.



	Name	Smoothed	Value	Step	Time	Relative
●	unet_aug_lr1e-3_reg1e-3_DiceFocalNoAug_test_1/Mean Scores_Dice Score	0.8388	0.8696	199	Mon Nov 29, 02:26:01	4m 12s
○	unet_aug_lr1e-3_reg1e-3_dicefocal_test_1/Mean Scores_Dice Score	0.8304	0.8907	199	Mon Nov 29, 00:48:50	3m 46s

Figure 4. Comparison of validation metrics with and without augmentation

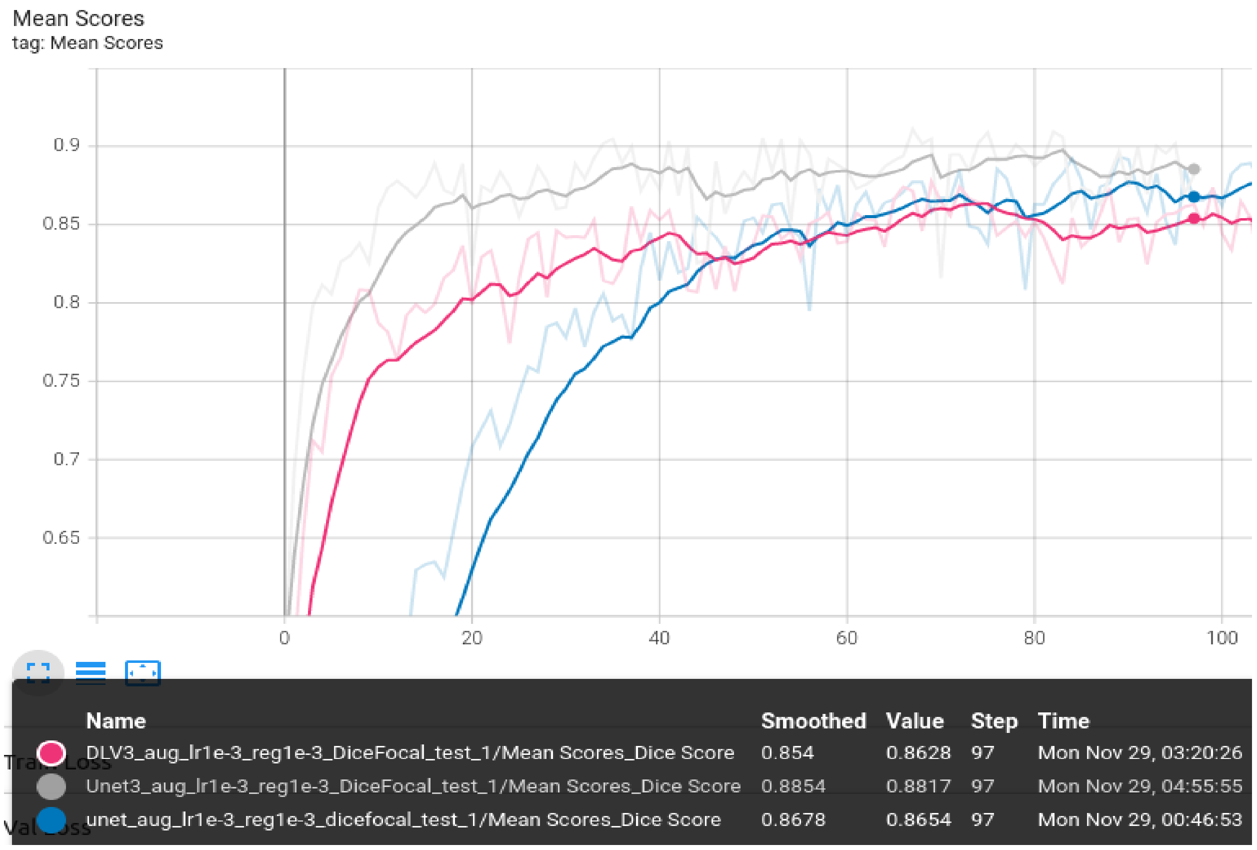
- **Different Model Architectures:**

We made comparisons between different model architectures based on our validation metrics, and have listed them below:

1. DeepLabV3 : It is based on a similar encoder-decoder architecture like UNet. The encoder contains a pre-trained CNN model that is used to obtain encoded feature maps for the input image, while the decoder reconstructs the output mask by upsampling the compressed feature map extracted by the encoder. The DeepLab model builds on a technique called Spatial Pyramid Pooling(SPP) that divides the feature maps from the last convolutional layer into a fixed number of spatial bins having size proportional to the image size. Since this pooling leads to an increased computational complexity of the model, DeepLabV3 makes use of dilated/atrous convolutions. A normal convolution makes use of every pixel in its defined kernel size, while a dilated convolution skips pixels in the input depending on a dilation rate r which defines the spacing between the values in the kernel. The benefit from this is that it enhances the receptive field of filters by allowing us to incorporate a larger overall image context without increasing the number of parameters in our model. In our implementation of this approach we used Resnet18 as backbone which is pre-trained on the ImageNet dataset.

2. UNet3+: The UNet model suffers from a few issues pertaining to calculating an optimal model depth which requires an extensive search over different architectures, and the use of skip connections which restrict the model to only fuse feature maps of the same scale. The UNet++ architecture deals with this issue by adding convolutional blocks on the skip pathways that bridge the gap between the encoder and decoder feature maps,

allowing for better feature concatenation and overall gradient flow. In our work we experimented with the UNet3+ architecture from [3], and compared its results on the validation data with the other models.



	Name	Smoothed	Value	Step	Time
●	DLV3_aug_lr1e-3_reg1e-3_DiceFocal_test_1/Mean Scores_Dice Score	0.854	0.8628	97	Mon Nov 29, 03:20:26
●	Unet3_aug_lr1e-3_reg1e-3_DiceFocal_test_1/Mean Scores_Dice Score	0.8854	0.8817	97	Mon Nov 29, 04:55:55
●	unet_aug_lr1e-3_reg1e-3_dicefocal_test_1/Mean Scores_Dice Score	0.8678	0.8654	97	Mon Nov 29, 00:46:53

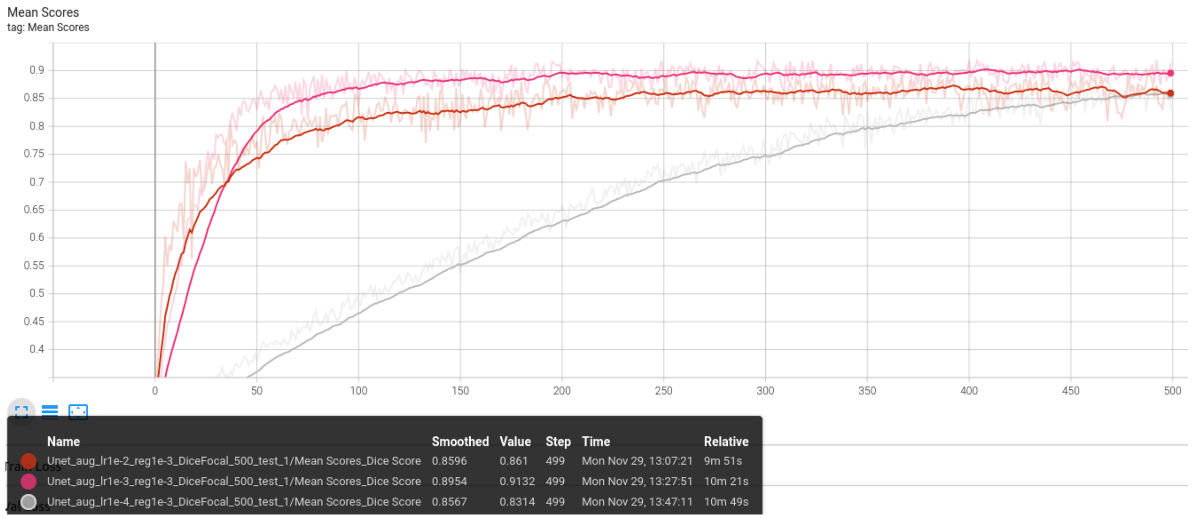
Figure 5. Comparison of Mean score Dice score for different architectures.

We trained the three architectures for 100 epochs each and compared their performance using the mean dice score on the validation data. We can see that the Unet3+ architecture converges much faster and provides a much better overall performance when compared to the DeepLabV3 and UNet architectures.

- **Optimizer parameter tuning (Learning Rates and Regularization)**

1. Learning Rate: The learning rate hyperparameter controls the rate or speed at which the model learns. Specifically, it controls the amount by which the weights of the model are updated each epoch. Choosing a learning rate that is too high can lead to our weights

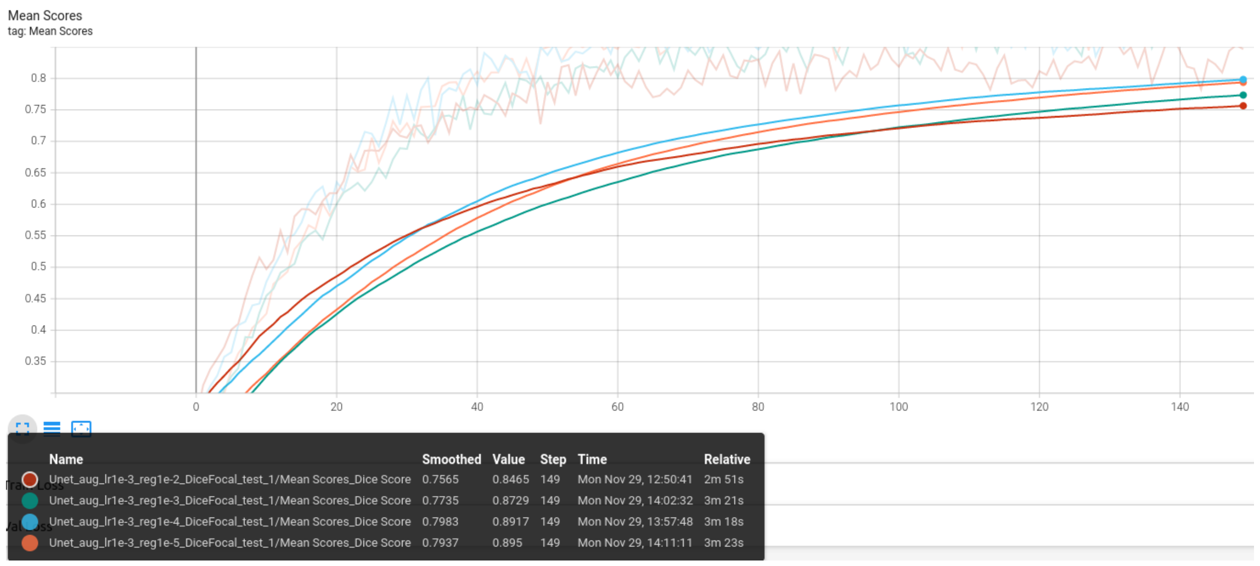
being updated too rapidly, causing us to overshoot the ideal minimum, whereas a learning too small will make our training process require a large number of epochs to converge. In our work, to choose the learning rate effectively we experimented with different values [1e-2, 1e-3, 1e-4] and compared the model performance after 500 epochs. The figure below shows the mean dice score calculated on the validation data for each learning rate. We can infer that the learning rate of 1e-2 shows the worst performance, since the model oscillates greatly around the minima, while the learning rate of 1e-4 takes a large number of iterations to start showing convergence. For our experiments we therefore make use of a learning rate of 1e-3, since it offers a good tradeoff between speed of training and model performance.



	Name	Smoothed	Value	Step	Time	Relative
●	Unet_aug_lr1e-2_reg1e-3_DiceFocal_500_test_1/Mean Scores_Dice Score	0.8596	0.861	499	Mon Nov 29, 13:07:21	9m 51s
●	Unet_aug_lr1e-3_reg1e-3_DiceFocal_500_test_1/Mean Scores_Dice Score	0.8954	0.9132	499	Mon Nov 29, 13:27:51	10m 21s
●	Unet_aug_lr1e-4_reg1e-3_DiceFocal_500_test_1/Mean Scores_Dice Score	0.8567	0.8314	499	Mon Nov 29, 13:47:11	10m 49s

Figure 6. Comparison of Mean Dice score for different learning rates.

2. Regularizer parameter: Regularization is a widely used strategy to prevent overfitting and help the model generalize well for unknown data. For our experiments we make use of the L2 regularizer that penalizes the square of the weights for each update. The parameters for the weight regularizer were tested as a part of the Adam optimizer. We compared the performance of the trained model for a learning rate of 1e-3 with regularization values of [1e-2, 1e-3, 1e-4, 1e-5], and inferred their performance using mean dice score on the validation data. We can observe from the graph below, that a value 1e-4 yields best validation performance.



	Name	Smoothed	Value	Step	Time	Relative
🟥	Unet_aug_lr1e-3_reg1e-2_DiceFocal_test_1/Mean Scores_Dice Score	0.7565	0.8465	149	Mon Nov 29, 12:50:41	2m 51s
🟩	Unet_aug_lr1e-3_reg1e-3_DiceFocal_test_1/Mean Scores_Dice Score	0.7735	0.8729	149	Mon Nov 29, 14:02:32	3m 21s
🟦	Unet_aug_lr1e-3_reg1e-4_DiceFocal_test_1/Mean Scores_Dice Score	0.7983	0.8917	149	Mon Nov 29, 13:57:48	3m 18s
🟧	Unet_aug_lr1e-3_reg1e-5_DiceFocal_test_1/Mean Scores_Dice Score	0.7937	0.895	149	Mon Nov 29, 14:11:11	3m 23s

Figure 7. Comparison of Mean Dice score for different regularizer parameters.

Conclusion

Summarising our learnings, for the task of semantic segmentation of Magnetic Resonance Images, our trained UNet performs well on the training data, and gives us moderate results on our test set (test dice score = **85%**). By carrying out exploratory analysis of the training dataset we observed that the huge class imbalance between the background, and the remaining three classes, lead to us obtaining varied levels of performance in each class. Also since the number of samples available for training is very small (**100**), it is difficult to obtain a model that generalizes to all cases. The use of L2 weight regularization, dropout layers in our model architecture and augmentation of our training data lead to a better performance on the test set. Our experiments with different architectures like DeepLabV3[4] and Unet3+[3] also showed a better performance compared to a standard UNet model. The use of early stopping aided in saving a model that provides the best tradeoff between training and validation performance. The graph below depicts the performance metrics of the final trained model that was trained on both training and validation datasets, once the ideal hyperparameters were selected.

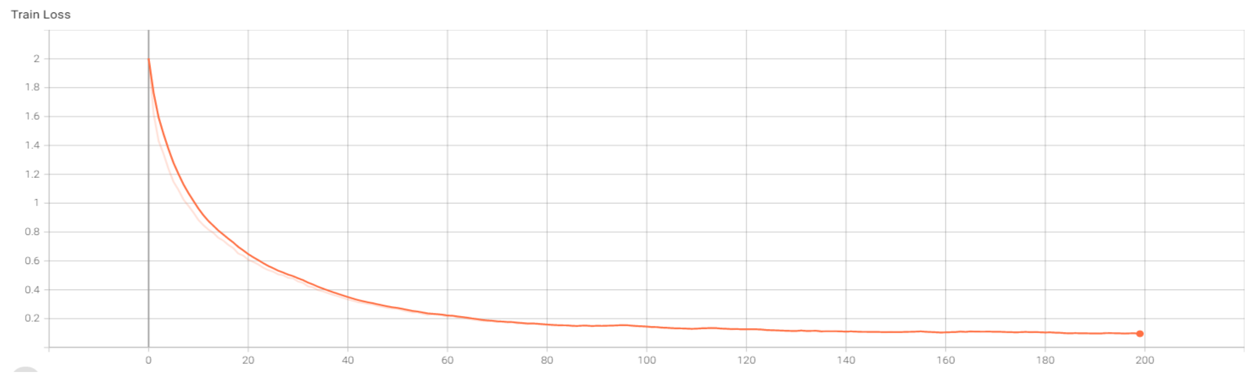


Figure 8. Training loss

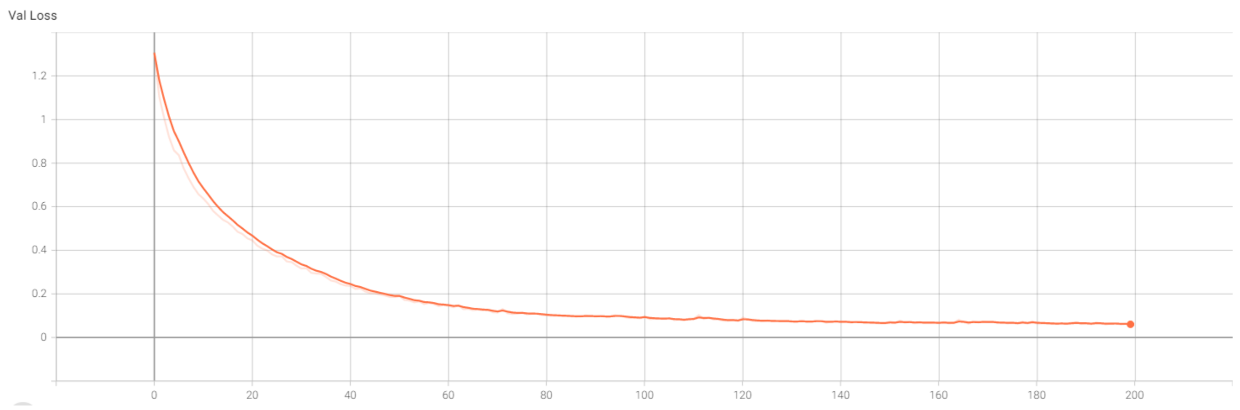


Figure 9. Validation loss

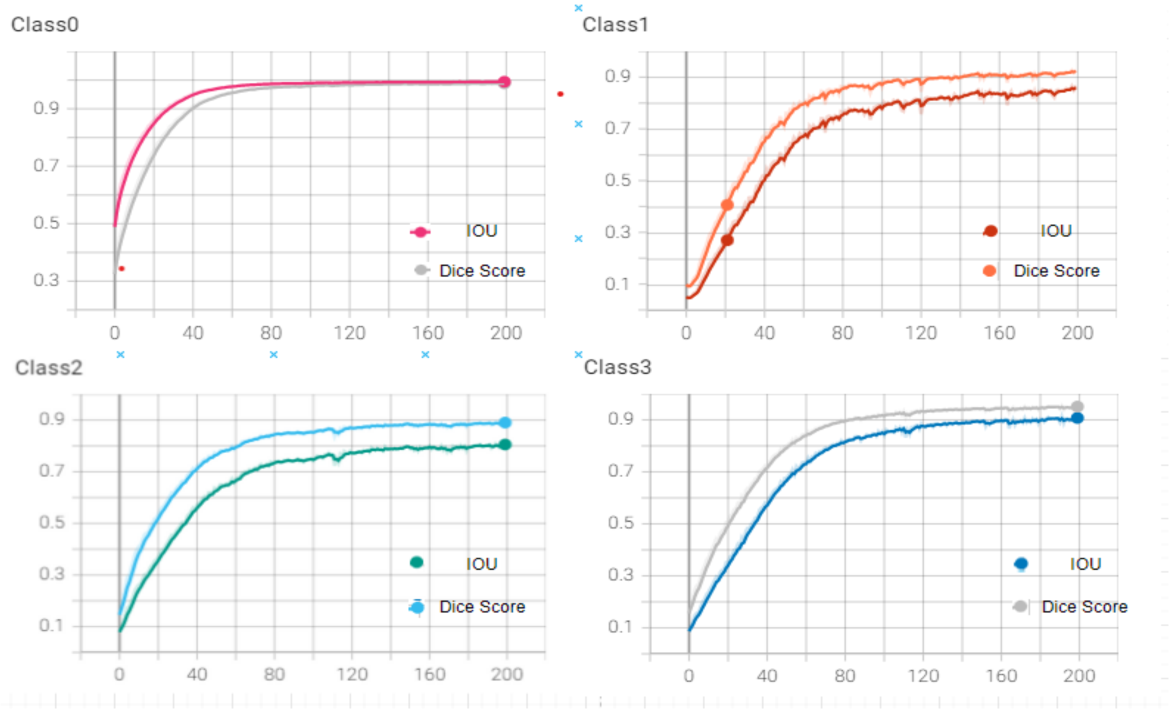


Figure 10. Class wise metric for dice score and IOU

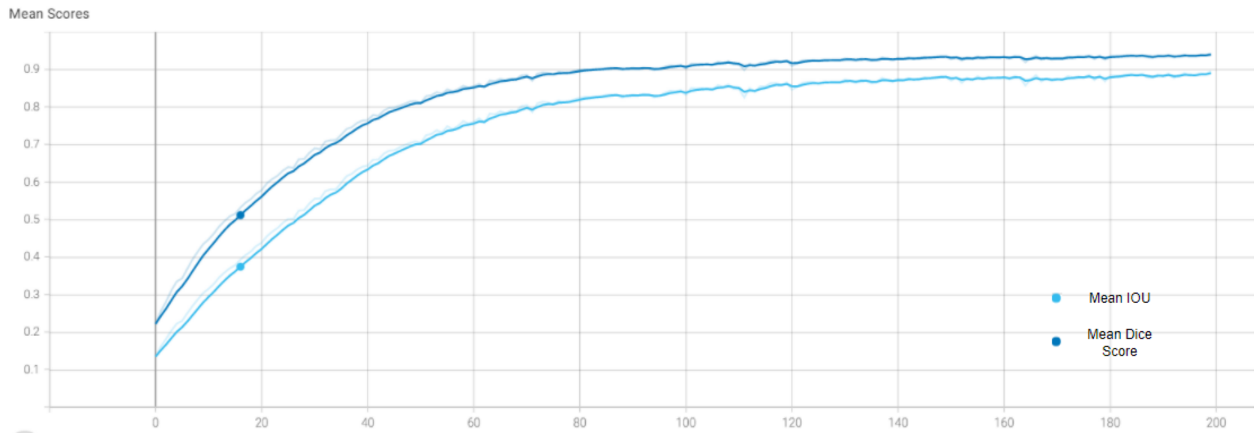


Figure 11. Mean dice score and for loss

References

[1] Class-Balanced Loss Based on Effective Number of Samples - https://arxiv.org/pdf/1901.05555.pdf
[2] Patravali, S2D-3D Fully Convolutional Neural Networks for Cardiac MR Segmentation. In Statistical Atlases and Computational Models of the Heart. ACDC and 'MMWHS' Challenges 2018 (pp. 130–139). Springer International Publishing.
[3] Code for ICASSP 2020 paper 'UNet 3+: A full-scale connected unet for medical image segmentation' - https://github.com/ZJUGiveLab/UNet-Version
[4] Liang-Chieh Chen and George Papandreou and Florian Schroff and Hartwig Adam (2017). Rethinking Atrous Convolution for Semantic Image Segmentation. CoRR, https://arxiv.org/abs/1706.05587. Implementation referenced from - http://www.fregu856.com/project/deeplabv3/