EE782: Assignment 1 - HyperParameter tuning for Image Segmentation

Jayesh Choudhary (170070038)
Bhavesh Garg (17D070031)

November 2020

1 Motivation

Hyperparameter tuning is an important part of training a neural network. For the given task of 'Pixel-wise Segmentation of Right Ventricle of Heart', we have selected to tune the training w.r.t the **Loss function** and **Regularization** in the network.
Our motivation behind choosing these hyperparameters is that the loss function is one of the most important hyperparameter in a deep learning model and selecting an appropriate loss for a problem can go a long way in solving it.
Similarly, regularization is an important tool in deep learning to solve the problem of overfitting/underfitting in a model which is necessary for obtaining good results on unseen data points. When training set is limited in size, it is dangerous to train without any constraint since the network will soon try to model noise in the training set, and become hard to generalize beyond the training set, i.e. the issue of overfitting. One method to deal with overfitting is regularization - the idea of encouraging weights to have smaller values.

2 Experiment Design for Loss functions

2.1 Training Details

Our training dataset consists of 253 labelled samples, each image is of size 256x216. For each image there are 2 contours available as the ground truth value - one for endocardium and one for epicardium segmentation.
Since we are tuning the loss function and regularization, we have tried to keep all other hyperparameters of the model constant throughout our experiments.

- Architecture - U-Net (3 encoder blocks - bottleneck - 3 decoder blocks)
- Batch size - 16
- Optimizer - Adam
- Epochs - 100

- Learning rate - 10e-3 for first 50 epochs, then decreased by a factor of 0.8 after every 10 epochs
- ReLU activation for all layers except the last one for which sigmoid is best suited
- Batch normalization has been used in all layers, after the ReLU activation function
- We have used CLAHE as a pre-processing technique on the given grayscale images to improve their contrast and hence make them easier to segment. We further scaled the pixel values to be in the range [0, 1].

The above hyperparameters are true for both parts of our experiment - Loss functions and Regularization.

2.2 Loss Function

We have taken into consideration 5 different loss functions, all suitable for the type of problem at hand: **Focal** loss, Binary Cross Entropy**(BCE)** loss, **DICE** loss, **Inverse DICE** loss, **Tversky** loss.

2.2.1 Binary Cross Entropy loss

$$L_B(p, y) = \begin{cases} -log(p) & y = 1 \\ -log(1-p) & else \end{cases}$$

where y is the ground truth label, p is the estimated probability of the class y=1.

2.2.2 Focal loss

Focal loss is used to counter the class imbalance problem in segmentation/classification problems. Formally it introduces a modulating factor $(1 - p_t)^\gamma$ in the BCE loss.

$$L_F(p_t) = -(1 - p_t)^\gamma log(p_t)$$

$$p_t = \begin{cases} p & y = 1 \\ 1 - p & else \end{cases}$$

2.2.3 DICE loss and Inverse DICE loss

DICE loss, intuitively, tries to maximize the overlap of predicted segmentation region of foreground and ground truth foreground.
The computation of DICE loss on the background region is the inverse DICE loss which makes sure that the background is classified well.

$$L_D = 1 - \frac{2\sum_i \sum_j p(i,j)g(i,j) + \epsilon}{\sum_i \sum_j p(i,j) + \sum_i \sum_j g(i,j) + \epsilon}$$

$$L_{ID} = 1 - \frac{2\sum_i \sum_j (1 - p(i,j))(1 - g(i,j)) + \epsilon}{\sum_i \sum_j (1 - p(i,j)) + \sum_i \sum_j (1 - g(i,j)) + \epsilon}$$

where ϵ is a stabilizing constant.

2.2.4 Tversky loss

Tversky loss is a generalization of the DICE loss, useful in cases where we need a finer level of control.

$$L_T = \frac{\sum_i \sum_j p(i,j)g(i,j) + \epsilon}{\sum_i \sum_j p(i,j)g(i,j) + \alpha \sum_i \sum_j (1 - p(i,j))g(i,j) + \beta \sum_i \sum_j p(i,j)(1 - g(i,j)) + \epsilon}$$

2.2.5 Switching loss

Switching loss is a combination of the BCE, DICE and and inverse DICE loss.

$$L_S = L_B + \lambda L_D + (1 - \lambda) L_I$$

Here, λ is another hyperparameter which we have experimented with.

2.3 Strategy for tuning loss function

We first train our model on all individual loss functions described above to see how they are perform the given task. We also tune the hyperparameter associated with each loss, e.g., in focal loss (λ), Tversky loss (α, β) and switching loss. We then move on to try different combinations of the losses to get the best possible results. These combinations are based on what each loss contributes, for e.g., inverse DICE improves background segmentation. The weight of each loss is again a hyperparameter for which we have tried to find the best value. The results are summarised in the following sub-subsection.

2.4 Observations for loss function

Loss	**Dice score(endocardium)**	**Dice score(epicardium)**
$L_F, \gamma = 1$	0.689	0.723
$L_F, \gamma = 1.5$	0.707	0.735
L_D	0.716	0.746
$L_D + L_F(\gamma = 1.5)$	0.717	0.756
L_B	0.706	0.746
$L_B + 0.5L_D$	0.724	0.755
$0.5L_B + L_D$	0.715	0.75
$L_T, \alpha = 0.7, \beta = 0.3$	0.667	0.735
$0.5L_T + L_D + 0.5L_B$	**0.749**	**0.771**
L_{ID}	0.703	0.745
$L_{ID} + L_B + L_T(0.7, 0.3)$	0.724	0.758
$L_B + 0.75L_D + 0.25L_{ID}$	0.715	0.752
$L_B + 0.5L_D + 0.5L_{ID}$	0.731	0.759
$L_B + 0.25L_D + 0.75L_{ID}$	0.712	0.756
$L_B + 0.25L_D + 0.75L_{ID}$	0.72	0.749

Table 1: Experiments on loss function

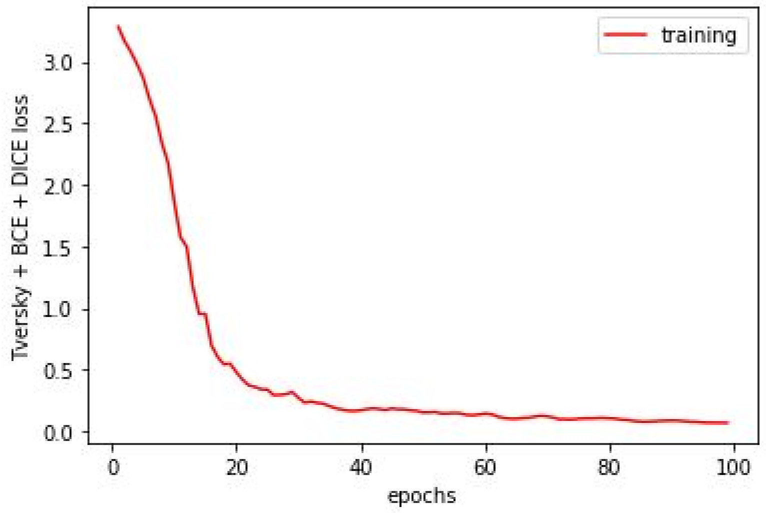

Figure 1: Sample training curve, the training converges after around 40 epochs

2.5 Key Insights

- For focal loss, the model performs better when the value of the hyperparameter λ is higher, i.e., DICE scores are better for (λ=1.5) as compared to (λ=1).

- For model trained on only 1 type of loss, i.e., either Focal or BCE or DICE or Inverse DICE loss, the best performance is obtained for DICE loss as expected, since it tries to maximise the 'differentiable' dice score itself.
- BCE loss and focal loss both suffer from the problem of class imbalance here (lot of '0' ground truth values).
- Adding DICE loss and Inverse DICE loss to BCE loss generally improves the performance of the model. We obtained best performance in this case when $\lambda = 0.5$ (the hyperparameter in switching loss)
- We obtained the best Dice scores for the case when the loss function is a combination of Tversky, BCE and DICE loss. This can be explained by the fact that Tversky loss is a more general case of inverse DICE loss, thus fine tuning it has helped improve the performance of the model here.

3 Experiments with Regularization

To understand the affect of various types of regularization, we use a base model and add one type of regularization and we keep on changing the parameters to get the best possible variation.

3.1 Weight Decay or L2 Regularization

Base model :

- Architecture - UNET
- Learning-rate = 1e-3
- Loss function = Dice + BCE Loss

Dataset	BatchSize	Weight Decay	Dice (endocardium)	Dice (epicardium)
20%	4	1e-4	0.258	0.299
20%	4	1e-3	0.360	0.402
20%	4	1e-2	0.322	321
20%	4	5e-4	0.367	0.388
20%	4	5e-3	0.354	0.360
60%	8	1e-4	0.646	0.667
60%	8	1e-3	0.494	0.548
60%	16	5e-4	0.608	0.617
60%	16	1e-5	0.636	0.640
100%	16	1e-5	0.708	0.721
100%	16	1e-4	0.710	0.741
100%	16	2e-4	0.678	0.708

Table 2: Dice score for Weight Decay for different sized dataset

Here we observe that when the dataset is small, relatively larger weight decay works better to avoid overfitting. It is evident from the fact that for 30% dataset, weight decay of 0.001 works better, for 60% dataset, weight decay of 0.0001, and for 100 dataset, weight decay of 0.0001 and 0.00001 works nearly the same.

3.2 Dropout

Base model :

- Architecture - UNET
- Learning-rate = 1e-3
- Loss function = Dice + BCE Loss

Dataset	Dropout p	Dice (endocardium)	Dice (epicardium)
60%	0.1	0.668	0.694
60%	0.2	0.676	0.697
60%	0.4	0.593	0.633
30%	0.2	0.452	0.470
100%	0.2	0.765	0.787
100%	0.4	0.713	0.735
100%	0.3	0.758	0.779
100%	0.1	0.760	0.781

Table 3: Dice score for Dropout for different sized dataset

For 30 % dataset, we see that dropout has significant effect. For larger dataset, the the model improves with dropout but the effect diminishes as the size increases. Although the difference between the training loss and validation loss goes down on using dropout which suggests that it is an effective way to deal with overfitting especially if dataset is smaller. Now we try out dropout such that the probability of a neuron to be dropped is different for different layers (Contractive, bottleneck and expansive layer)

Contractive	Bottleneck	Expansive	Dice (endocardium)	Dice (epicardium)
0.2	0.05	0.2	0.744	0.767
0.2	0.05	0.1	0.751	0.774
0.2	0.05	0.05	0.754	0.774

Table 4: Dice score for Dropout for different layers for 100% dataset

There was not much improvement on changing the dropout of different layers. The results suggest that the dropout in the bottleneck and decoder/expansive side does not have as much effect on the dice coefficient as the dropout in encoder or contractive side.

3.3 Dropout with weight decay

Base model :

- 100% dataset

- Architecture - UNET
- Learning-rate = 1e-3
- Loss function = Dice + BCE Loss

Weight Decay	Dropout p	Dice (endocardium)	Dice (epicardium)
1e-5	0.3-0.2-0.2	0.751	0.771
1e-4	0.1	0.759	0.781
5e-5	0.2	0.761	0.782

Table 5: Dice score for Dropout and weight decay combination

On using the combination of weight decay and dropout, we achieved dice coefficient in the range of 0.76-0.78. This seems to be the saturation limit of the model and hence no conclusion can be drawn on the behaviour of this combination. Although increasing dropout and keeping a weight decay of 0.00005 does increase the dice coefficient and the gap between the training loss and validation loss decreases, meaning this combination reduces overfitting.

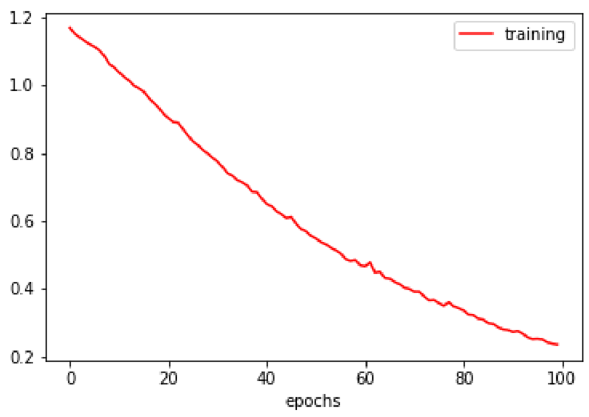

Figure 2: After 100 epochs, the training loss reaches very near to the validation loss mentioned above

3.4 L1 regularization

Base model :

- 100% dataset
- Architecture - UNET
- Learning-rate = 1e-3
- Loss function = Dice + BCE Loss

In the model L1 loss was implemented by explicitly adding the norm of the weights to the loss function and so the normalization term is so small.

lambda$_{L1}$	Dice (endocardium)	Dice (epicardium)
1e-7	0.596	0.597
1e-9	0.738	0.759
1e-8	0.643	0.672
1e-10	0.736	0.758

Table 6: Dice score for L1 regularization

Lambda here may look small but it is because of the normalization. We see that on increasing the $lamda_{L1}$, dice score comes down. L1 regularization is used to increase sparsity in weights but increasing it beyond a limit leads to poorer training.

3.5 Some more variations

Model: 100 % dataset, UNET architecture, Learning Rate: 1e-3 and after 50 epochs-decreased by 0.8 for every 10 epochs, Tversky Loss, L1-L2 regularization,Weight Decay = 1e-5, $lambda_{L1}$ = 1e-9
Dice(endocardium) = 0.684
Dice(epicardium) = 0.717

Model: 100 % dataset, UNET architecture, Learning Rate: 1e-3 and after 50 epochs-decreased by 0.8 for every 10 epochs, BCE + Dice Loss, L1-L2 regularization, Weight Decay = 1e-5, $lambda_{L1}$ = 1e-9
Dice(endocardium) = 0.669
Dice(epicardium) = 0.707

Model: 100 % dataset, UNET architecture, Learning Rate: 1e-3, BCE + Dice Loss, L1-L2 regularization, Weight Decay = 1e-5, $lambda_{L1}$ = 1e-9
Dice(endocardium) = 0.637
Dice(epicardium) = 0.656

Model: 100 % dataset, UNET architecture, Learning Rate: 1e-3, BCE + Dice Loss, L1-L2 regularization, Weight Decay = 1e-5, $lambda_{L1}$ = 1e-9
Dice(endocardium) = 0.738
Dice(epicardium) = 0.754

4 Conclusion

In the above experiments, we have tried to tune 2 important hyperparameters in a deep neural network - loss function and regularization. We have tried to gain some insight into how the model behaves on changing the loss function and regularization used in the model. A combination of BCE, DICE and Tversky (generalized inverse DICE) losses gives us the best performance of the model in terms of DICE score. We also investigated how L2 regularization (weight decay), dropout and L1 regularization affect the training and performance of a model.

5 Further Work

Due to the constraints of time and computational power, we could only perform a limited number of experiments while tuning the loss function and regularization. Given more time, we would have liked to investigate further - for e.g., training for much larger number of epochs, performing random grid search on the hyperparameters. We would have also liked to see if there is any relation between tuning loss functions and tuning regularization - whether some loss functions perform better with a particular method of regularization.