Assignment 1 - COL870

Ankish Chandresh(2018EE10447) Shobhit Singhal(2017ME10960)

April 21, 2021

1 ResNet over Convolutional Networks and different Normalization schemes

1.1 Augmentations on Cifar10

We use transforms as done in the Resnet paper,mean subtracted, pad 4 pixels and randomly crop 32x32 pixels, apply random horizontal flip on train and Validation transform. We finally also apply toTensor transform for both training and testing. The transform seem to reduce training time, numerical instability and increased stable training(as observed in No normalisation training.). We use this composition for all experiments.

1.2 Architecture,Design Choices and Implementation Details

The total number of layers in the network is 6n+2. We take n for total number of layers and r denoting the total number of classes as input. We choose n = 2, with a batch size of 128 for 100 epochs. For CIFAR 10, r = 10. We also use early stopping with a patience of 30 epochs, kept high for longer training for meaningful experimentation, for validation loss to improve and early stop otherwise. We randomly sample 10,000 of 50,000 training samples for validation and early stopping.
Lastly we use kaiming/xavier uniform strategy for initialisation of weights of Convolutional and Linear layers and 1 and 0 for normalisation parameters as suggested in the paper, and as needed for cases like no normalisation layer to break the symmetry that occasionally is observed after certain number of epochs maybe due to numerical instability.
There is small issue with data loaders being larger than expected size due to data size not being multiple of 128. So calculation of metrics could be larger for test time. As we didn't handle it, or trim it.
We use cross entropy as criterion and use SGD/Adam/Adadelta optimizer, as when needed based on convergence of different versions of models, Adadelta, only changed for no normalisation case for faster training, with initial learning rate of 0.1 with weight decay 0.0001 as in the paper and momentum 0.9. We schedule the learning rate with StepLR which decays the learning rate by 0.1/0.5 as when needed after step size of 25/50 epochs, which

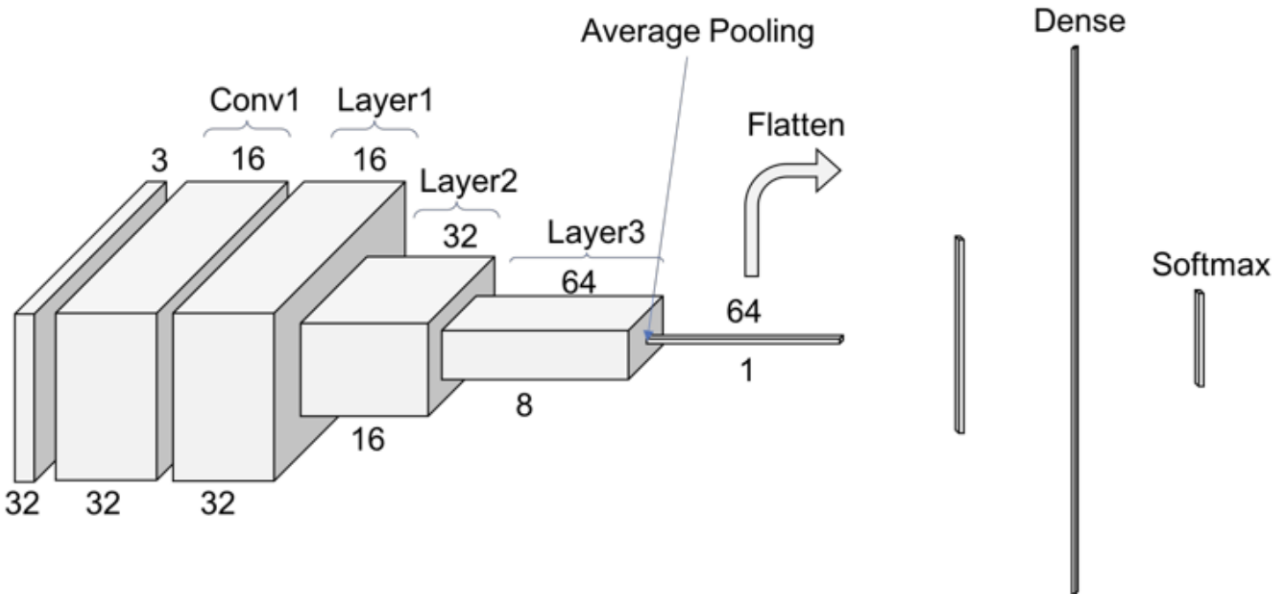

FIGURE 1: Architecture Illustration

	Case	Micro F1	Macro F1	Accuracy
1	Train	0.955	0.955	95.602
2	Validation	0.876	0.876	87.620
3	Test	0.849	0.850	84.995

TABLE 1: Pytorch Batch Normalisation Statistics

seems to give convergence.

1.3 Inbuilt Batch Normalisation

Following are Accuracy, Macro-f1 and Micro-f1 scores on Train, Validation and Test splits followed by error curves on Train and Val splits with torch batch normalisation:

1.3.1 Observations

The speed of computations and stability is improved with and without Batch normalisation. For training with the same optimizer and hyper parameters, the loss decreases slowly or blows up in first few epochs without batch normalisation. Output at each epoch is observed to be numerically unstable for few runs, causing the model to only learn one class. Batchnorm rectifies all these issues and gives convergence for most runs with all optimizer

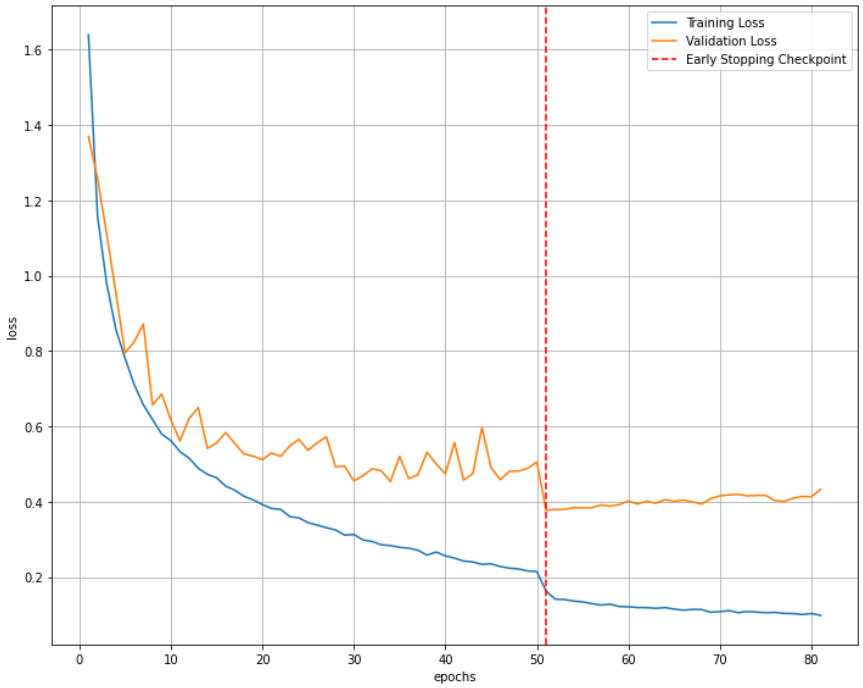

FIGURE 2: Pytorch Batchnorm Loss Plot

combinations.

1.4 Impact of Normalization

The different normalisation layers give varying performance statistics and training time. All with some improvement from No normalisation case, as is also seen in the loss plots and statistics/metrics.

1.4.1 No Normalization

Training is highly unstable, slow with small total Gradient norm. We use Adadelta here for training, with 0.1 learning rate and default parameters. And as described in previous sections.
Clearly, the performance is suboptimal even after training for complete 100 epochs, with larger per epoch time. Convergence is not attained always, and as described earlier it is numerically unstable for some runs, thus learning only one of the classes.
Following are Accuracy, Macro-f1 and Micro-f1 scores on Train, Validation and Test splits followed by error curves on Train and Val splits with no normalisation:

	Case	loss	Micro F1	Macro F1	Accuracy
1	Train	0.829	0.710	0.710	71.078
2	Validation	0.893	0.692	0.690	69.290
3	Test	0.928	0.684	0.678	68.429

TABLE 2: No Normalisation Statistics

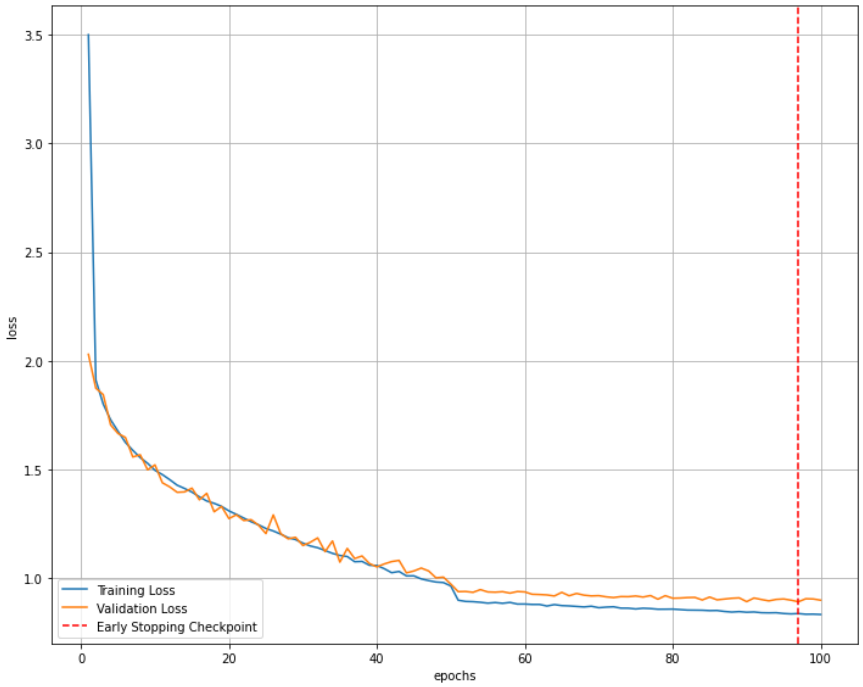

FIGURE 3: No normalisation Loss Plot

1.4.2 Batch Normalization (BN)

We initialise the learn able Batch normalisation parameters with 1 and 0. We don't implement inference time debaising as done in BatchNorm paper. We also see an improvement in training time and performance metrics. Performance is almost similar to the inbuilt batch normalisation, with convergence for most runs.

	Case	loss	Micro F1	Macro F1	Accuracy
1	Train	0.129	0.956	0.955	90.9
2	Validation	0.403	0.803	0.802	80.37
3	Test	0.496	0.849	0.850	84.99

TABLE 3: Batch Normalisation Statistics

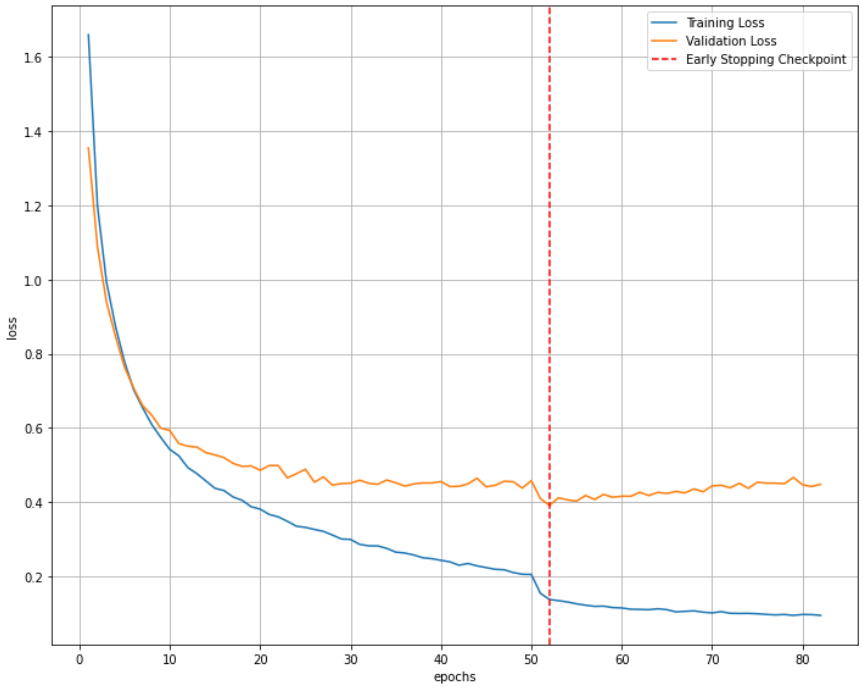

FIGURE 4: Batch normalisation Loss Plot

	Case	loss	Micro F1	Macro F1	Accuracy
1	Train	0.128	0.954	0.954	95.47
2	Validation	0.418	0.875	0.875	87.56
3	Test	0.553	0.845	0.845	84.57

TABLE 4: BIN Statistics

1.4.3 Batch Instance Normalization (BIN)

All configurations and experiment parameters are similar as above.
We set the adaptive parameter to be 0.5, rest all configurations and experiment paramters are similar as above.
Following are Accuracy, Macro-f1 and Micro-f1 scores on Train, Validation and Test splits followed by error curves on Train and Val splits with torch batch normalisation:

1.4.4 Layer Normalization (LN)

All configurations and experiment parameters are similar as above.
Following are Accuracy, Macro-f1 and Micro-f1 scores on Train, Validation and Test splits followed by error curves on Train and Val splits with torch batch normalisation:

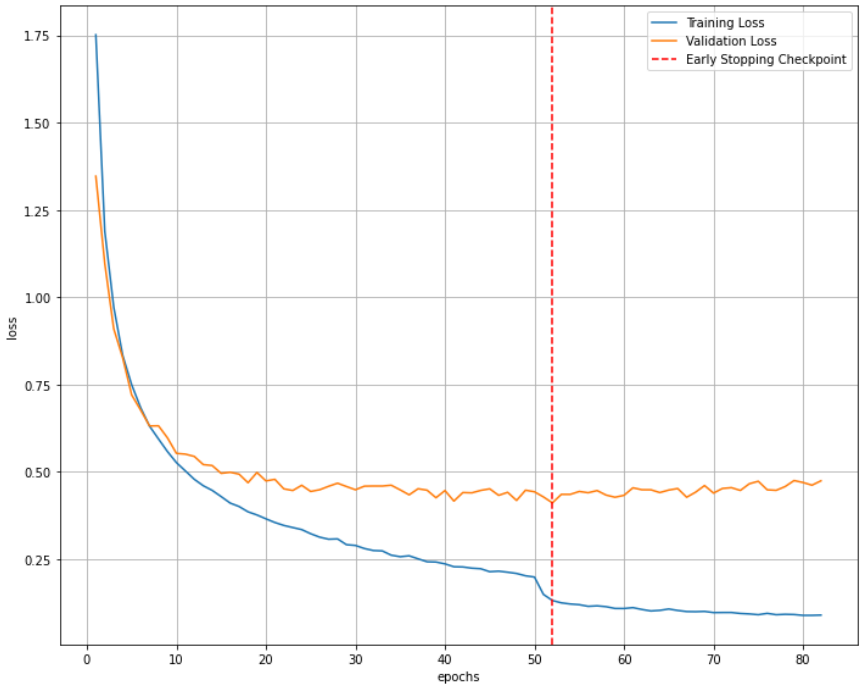

FIGURE 5: BIN loss plot

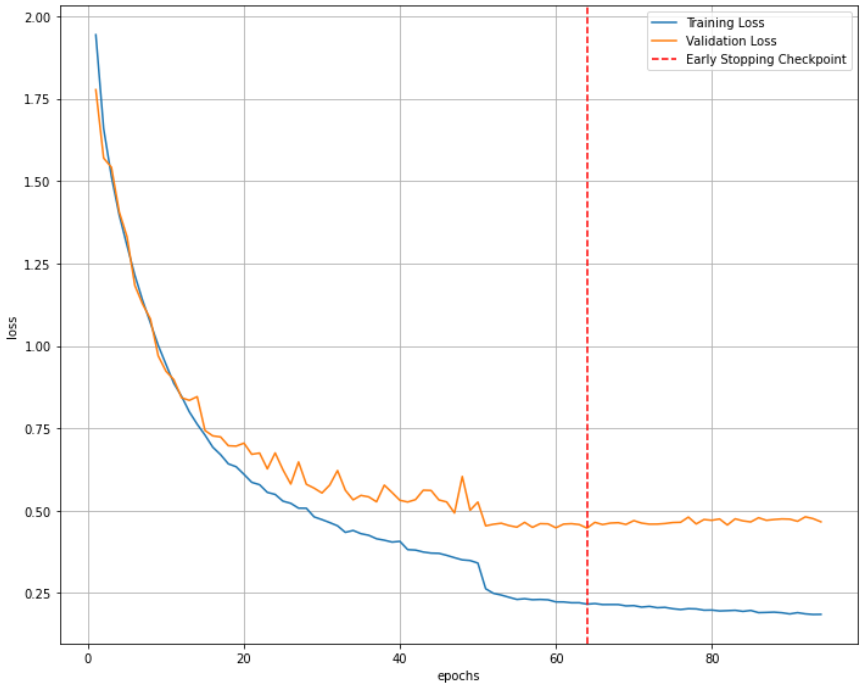

FIGURE 6: LN loss plot

1.4.5 Instance Normalization (IN)

It is a reasonable assumption that IN would be beneficial not only in generative tasks but also in discriminative tasks for addressing unnecessary style variations.

	Case	loss	Micro F1	Macro F1	Accuracy
1	Train	0.214	0.924	0.924	92.44
2	Validation	0.453	0.852	0.851	85.20
3	Test	0.485	0.845	0.845	84.59

TABLE 5: LN Statistics

	Case	loss	Micro F1	Macro F1	Accuracy
1	Train	0.171	0.942	0.942	94.24
2	Validation	0.416	0.865	0.864	86.51
3	Test	0.480	0.843	0.843	84.33

TABLE 6: IN Statistics

We set the adaptive parameter to be 0.5, rest all configurations and experiment paramters are similar as above.
Following are Accuracy, Macro-f1 and Micro-f1 scores on Train, Validation and Test splits followed by error curves on Train and Val splits with torch batch normalisation:

1.4.6 Group Normalization (GN)

Here we choose G = 2 and G = 4. The statistics and loss are shown below, we choose G = 4 to be default for comparisons in later sections.

1.4.7 Effect of group size

As G=1 is IN and G=C is LN, the performance is comparable, with most G values within this range, with no significant improvement.

	Case	loss	Micro F1	Macro F1	Accuracy
1	Train	0.871	0.688	0.687	68.82
2	Validation	0.901	0.681	0.679	68.15
3	Test	0.98 2	0.646	0.637	64.63

TABLE 7: GN Statistics with 4 groups

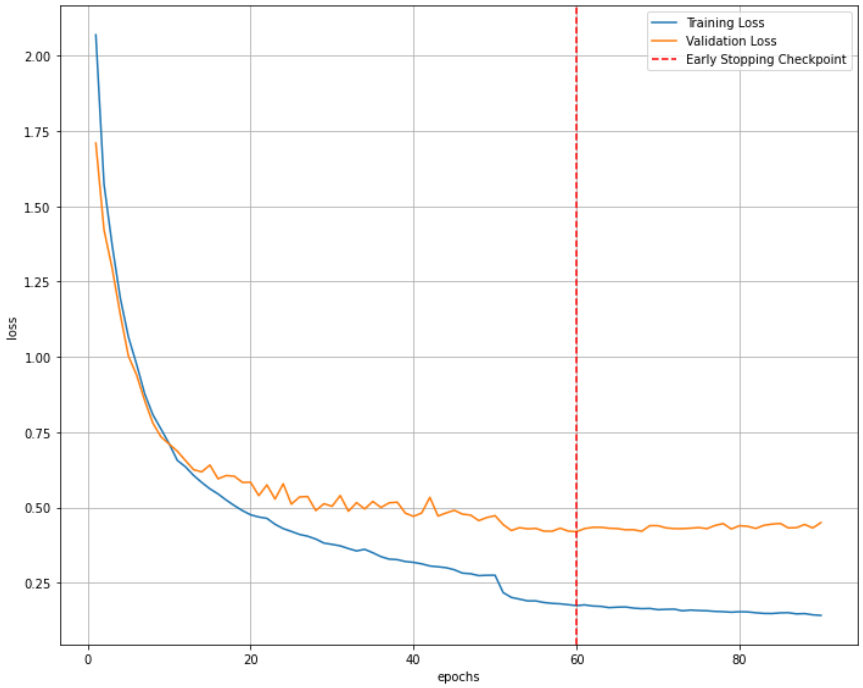

FIGURE 7: Instance Normalisation Loss plot

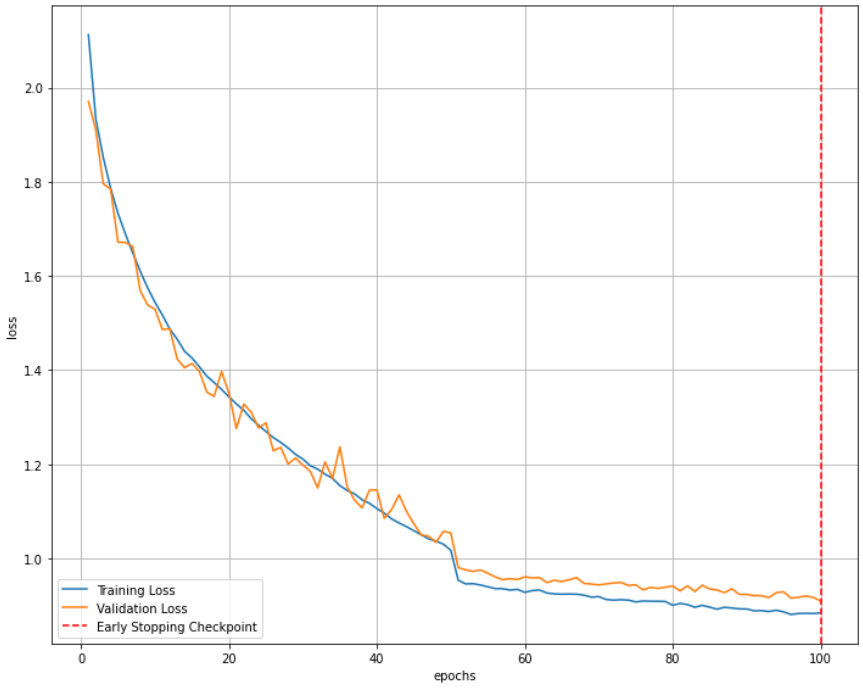

FIGURE 8: GN Loss plot with 4 groups

	Case	loss	Micro F1	Macro F1	Accuracy
1	Train	0.237	0.918	0.918	91.82
2	Validation	0.425	0.859	0.859	85.92
3	Test	0.466 2	0.840	0.839	84.02

TABLE 8: GN Statistics with 2 groups

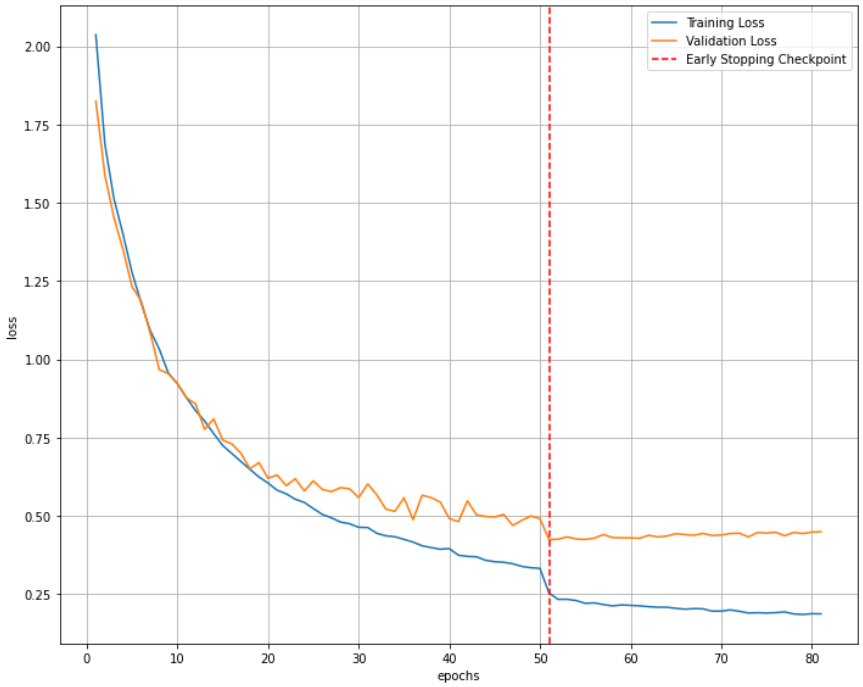

FIGURE 9: GN Loss plot with 2 groups

1.5 Comparison of torch Batch Normalisation with Batch Normalisation implementation

Clearly, the two versions perform similarly, with similar percentile plots, metrics and training dynamics.
Following are performance statistics and loss plot of the model trained with Inbuilt Batch Normalisation and with the Batch Normalisation implementation, which are almost similar:

Clearly, the training dynamics are almost identical as seen in the train loss curve.

	Case	Micro F1	Macro F1	Accuracy
1	Train	0.955	0.955	95.602
2	Validation	0.876	0.876	87.620
3	Test	0.849	0.850	84.995

TABLE 9: Pytorch Batch Normalisation Statistics

	Case	loss	Micro F1	Macro F1	Accuracy
1	Train	0.129	0.956	0.955	90.9
2	Validation	0.403	0.803	0.802	80.37
3	Test	0.496	0.849	0.850	84.99

TABLE 10: Batch Normalisation Statistics

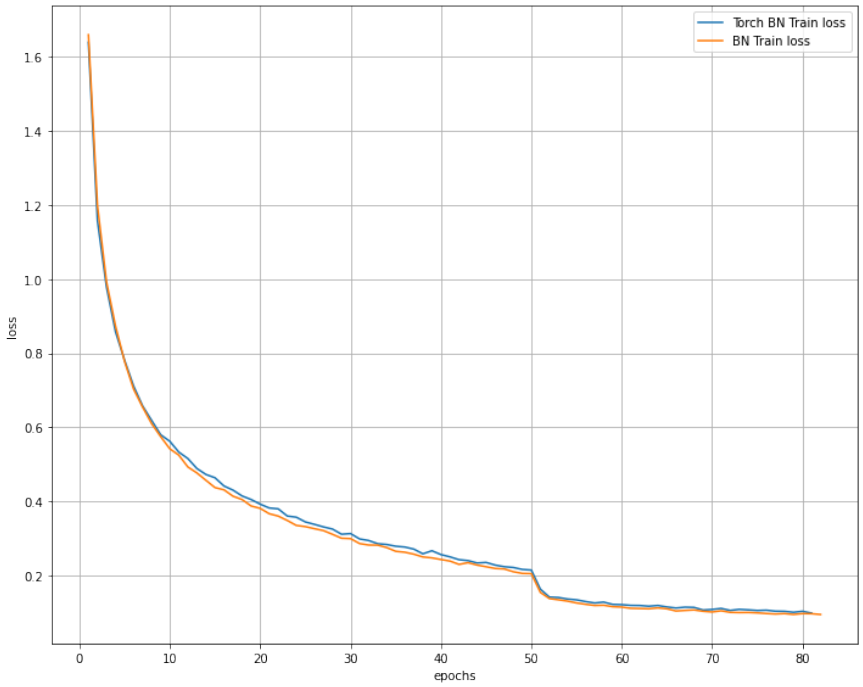

FIGURE 10: BN and inbilut BN training loss comparison

Also, qualitatively they seem to be very similar in controlling the feature distribution as seen in the percentile plots below:

1.6 Comparison of the error curves and performance statistics (accuracy, micro F1, macro F1 on train / val / test splits) of all the six models.

Clearly, NN performs worst as in metrics table and loss plots(lower and higher respectively). Versions of BN(BN,BIN and Torch BN) perform better as expected, training dynamics are also smooth, stable and converges as expected. Versions of GN(GN,LN,IN) also perform well as expcted. GN has substantially lower error than IN as claimed in the paper. Qualitively they are similar(expect NN) as in the percentile plots.
Comparison on Train/Validation and Test set are shown below:(in order)

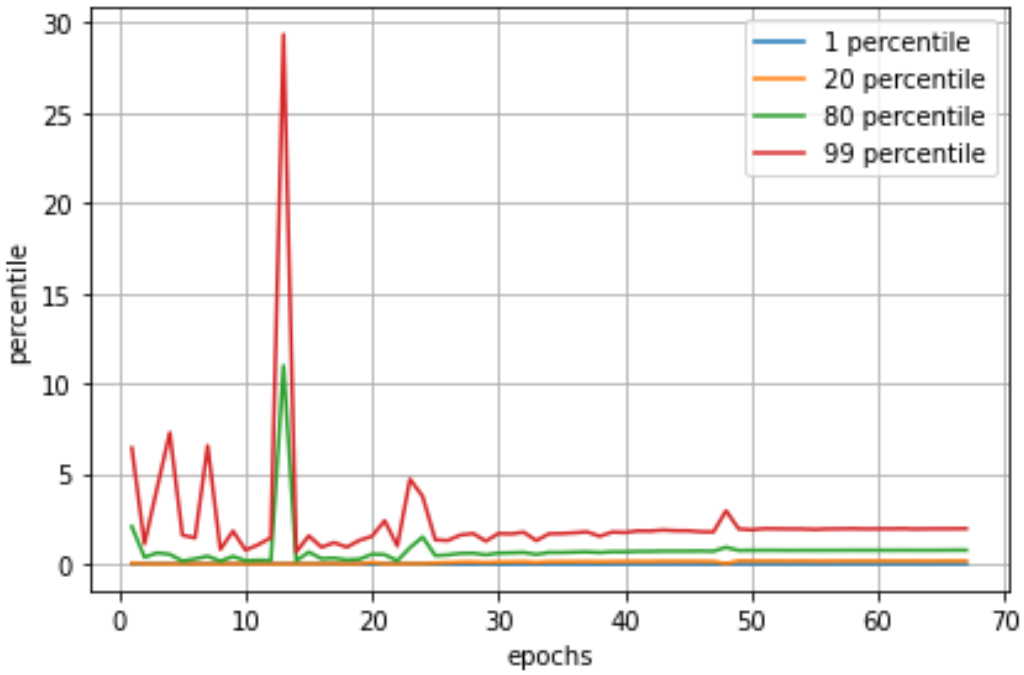

FIGURE 11: Percentile Plot for Torch Batch Normalisation

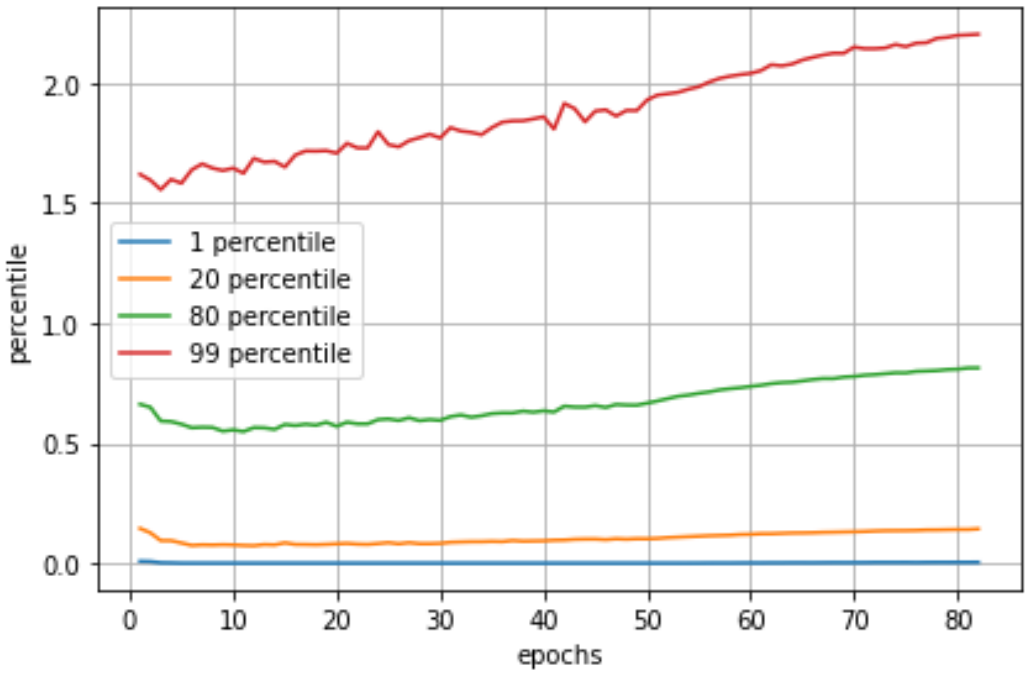

FIGURE 12: BN percentile plot

1.7 Impact of Batch Size

The advantages of GN over BN is its insensitivity to batch size. We retrain the BN and GN variants of the model with Batch Size 8 and 4 number of groups for 15-20 epochs as

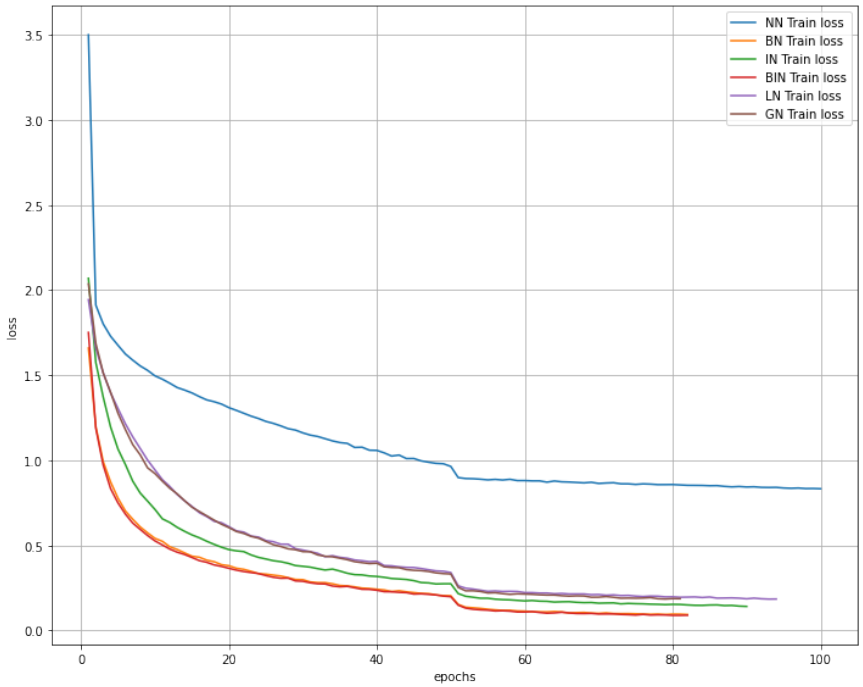

FIGURE 13: Comparison of train loss plots

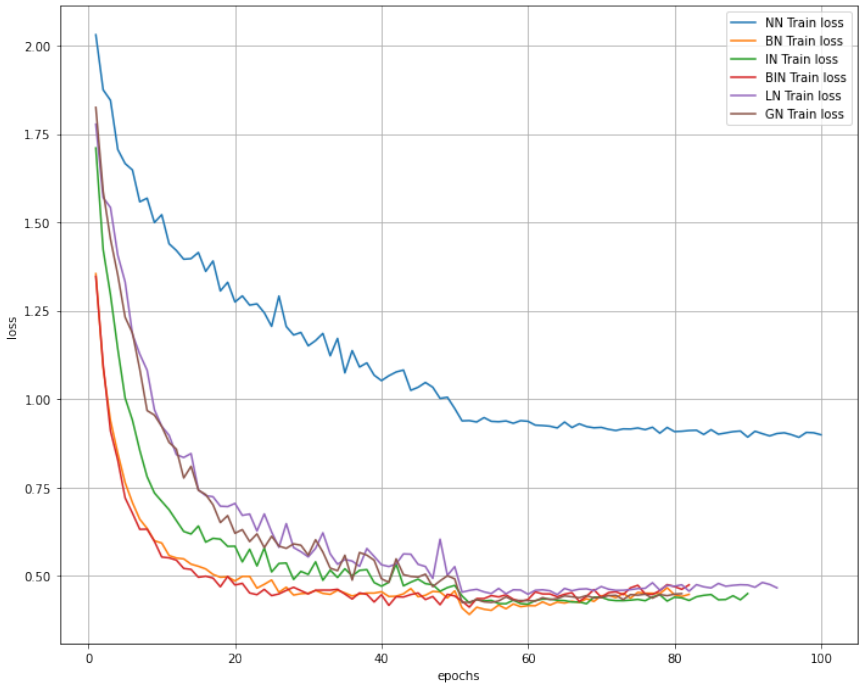

FIGURE 14: Comparison of validation loss plots

needed.

Clearly, the performance of GN is not affected(suboptimal due to convergence issues for longer trainning time on Colab) drastically, in our case improved. BN performance is clearly

	Case	Micro F1	Macro F1	Accuracy
1	Torch BN	0.955	0.955	95.602
2	BN	0.956	0.955	90.9
3	IN	0.942	0.942	94.24
4	BIN	0.954	0.954	95.47
5	LN	0.924	0.924	92.44
6	GN	0.688	0.687	68.82
7	NN	0.710	0.710	71.078

TABLE 11: Train

	Case	Micro F1	Macro F1	Accuracy
1	Torch BN	0.876	0.876	87.620
2	BN	0.803	0.802	80.37
3	IN	0.865	0.864	86.51
4	BIN	0.875	0.875	87.56
5	LN	0.852	0.851	85.20
6	GN	0.681	0.679	68.15
7	NN	0.692	0.690	69.290

TABLE 12: Validation

decreased as seen in the table.
The loss plots are as expected, with not much shift in the loss plots for initial epochs in GN and even BN for intial epochs.

1.8 Evolution of feature distributions

Clearly, as expected the implemented BN and torch BN are qualitatively similar. Similarly, GN(G=2/4) and BN seem to be qualitatively similar.
And as claimed : GN and BN substantially different with the variant that uses no normalization. This comparison suggests that performing normalization is essential for controlling the distribution of features, the plots and the metrics are indicative of it.

2 LSTM with Layer Normalization and CRF

2.1 Implementation details

We used Adam as optimizer with learning rate of 0.001. For early stopping, patience of 5 was used. Negative of log of probability was used as loss function. We exclude the

	Case	Micro F1	Macro F1	Accuracy
1	Torch BN	0.849	0.850	84.995
2	BN	0.849	0.850	84.99
3	IN	0.843	0.843	84.33
4	BIN	0.845	0.845	84.57
5	LN	0.845	0.845	84.59
6	GN	0.646	0.637	64.63
7	NN	0.684	0.678	68.429

TABLE 13: Test

	Case	Micro F1	Macro F1	Accuracy
1	GN + 128 batchsize Train	0.688	0.687	68.82
2	GN + 8 batchsize Train	0.757	0.756	75.77
3	BN + 128 batchsize Train	0.956	0.955	90.9
4	BN + 8 batchsize Train	0.762	0.762	76.2
5	GN + 128 batchsize Test	0.646	0.637	64.63
6	GN + 8 batchsize Test	0.7264	0.721	72.64
7	BN + 128 batchsize Test	0.849	0.850	84.99
8	BN + 8 batchsize Test	0.718	0.721	72.19

TABLE 14: Statistics with Varying Batchsize

padding words during loss computation. Accuracy score as well as F1 scores are calculated for all tags except 'O'. Vocabulary in the Pre-trained embeddings case was formed by the GloVe file. While, in the Random embeddings case, it was formed from train data file. The pretrained embeddings are fine-tuned during training.
All the comparisons are done for the metrics of test set. We also use jit module for speed concerns due to larger per epoch time in Layer Normalisation section.

2.2 Simple Bi-LSTM with and without pre-trained word embeddings:

It is observed that model with pre-trained word embeddings perform better than random embeddings. Note that pre-trained word embeddings are fine tuned during training.

2.2.1 Pre-trained word embeddings with dropout

Train Accuracy: 0.845, f1_micro: 0.841, f1_macro: 0.653
Val Accuracy: 0.794, f1_micro: 0.812, f1_macro: 0.618
Test Accuracy: 0.795, f1_micro: 0.815, f1_macro: 0.632

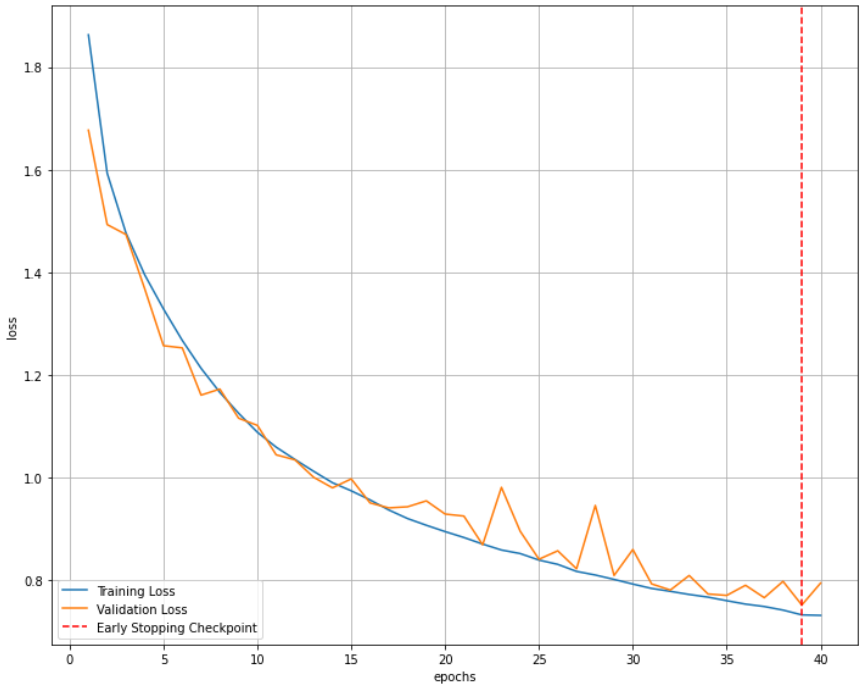

FIGURE 15: GN batchsize 8 loss plot

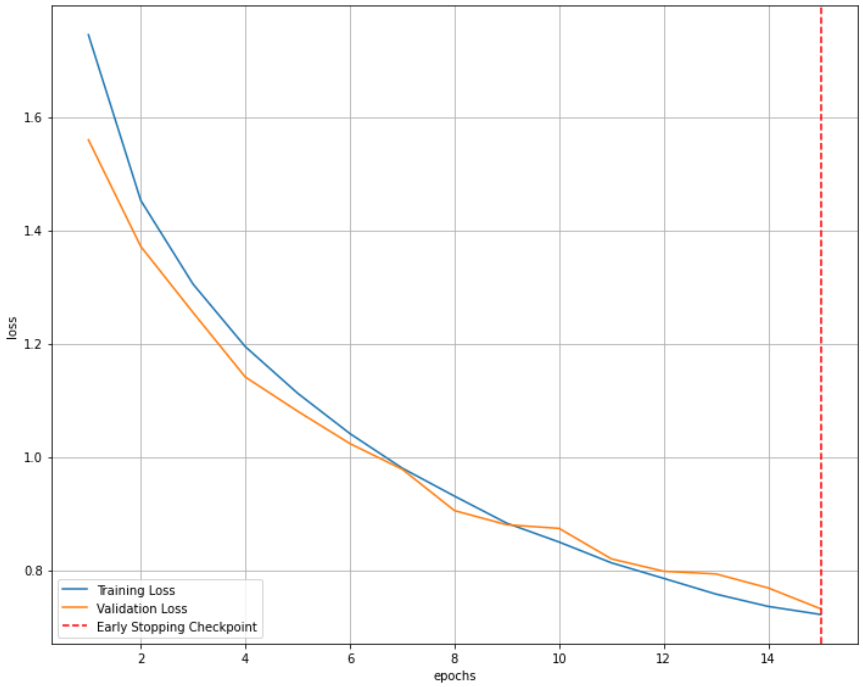

FIGURE 16: BN batchsize 8 loss plot

2.2.2 Random word embeddings with dropout

Train Accuracy: 0.862, f1_micro: 0.862, f1_macro: 0.655
Val Accuracy: 0.780, f1_micro: 0.808, f1_macro: 0.611

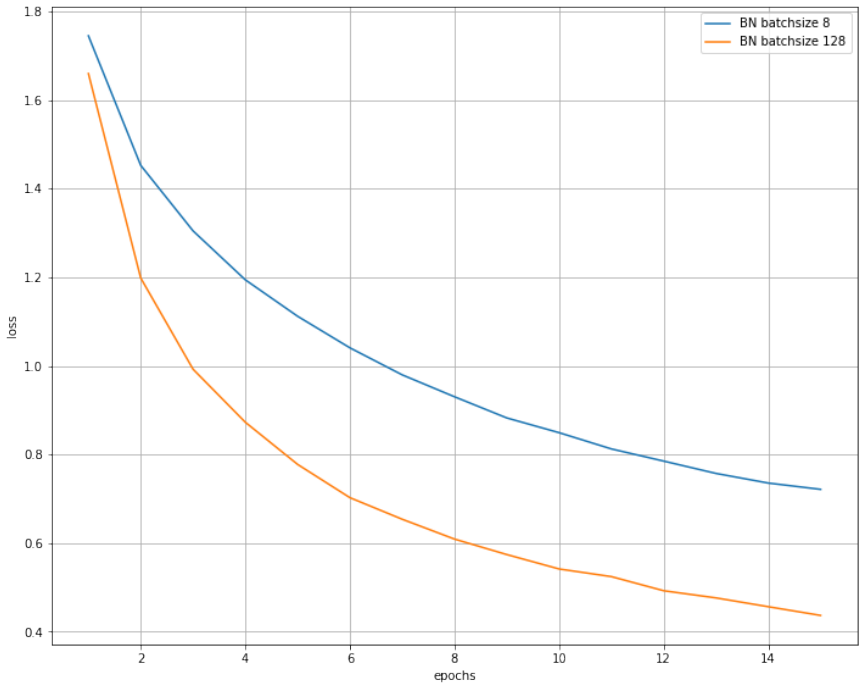

FIGURE 17: BN comparison

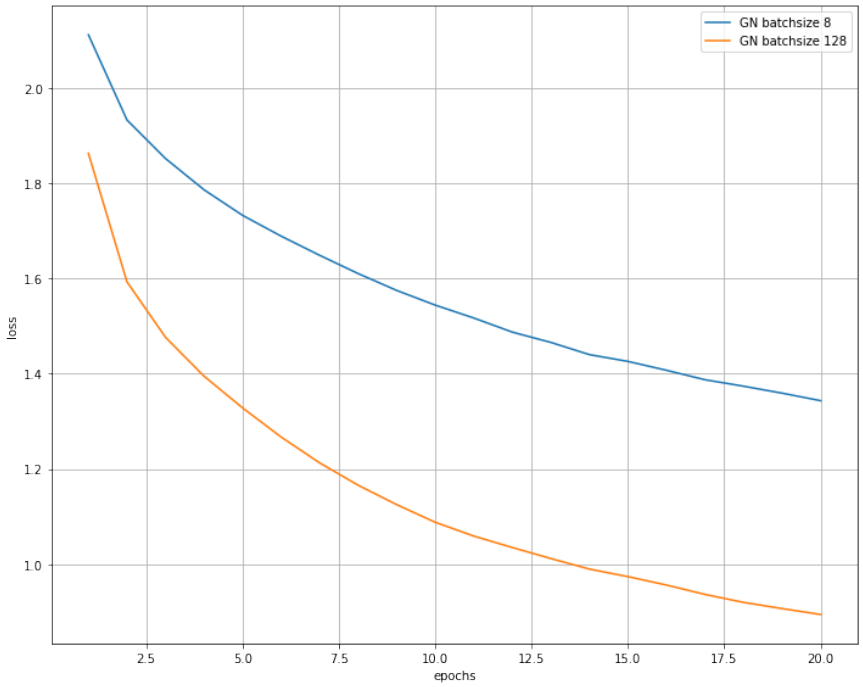

FIGURE 18: GN comparison

Test Accuracy: 0.783, f1_micro: 0.806, f1_macro: 0.612

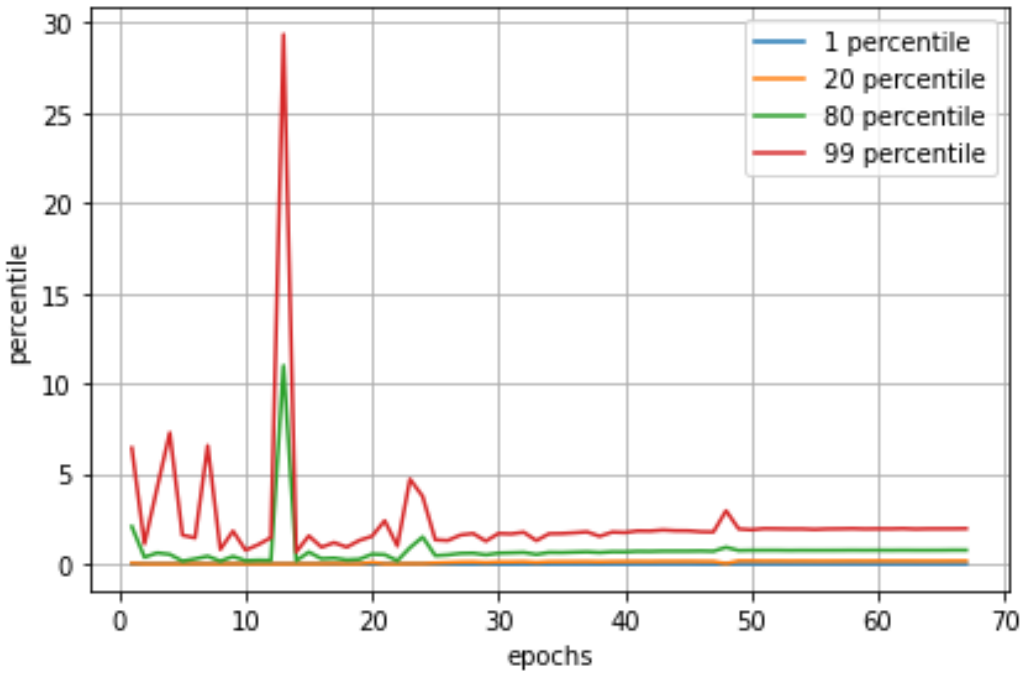

FIGURE 19: Torch BN percentile plot

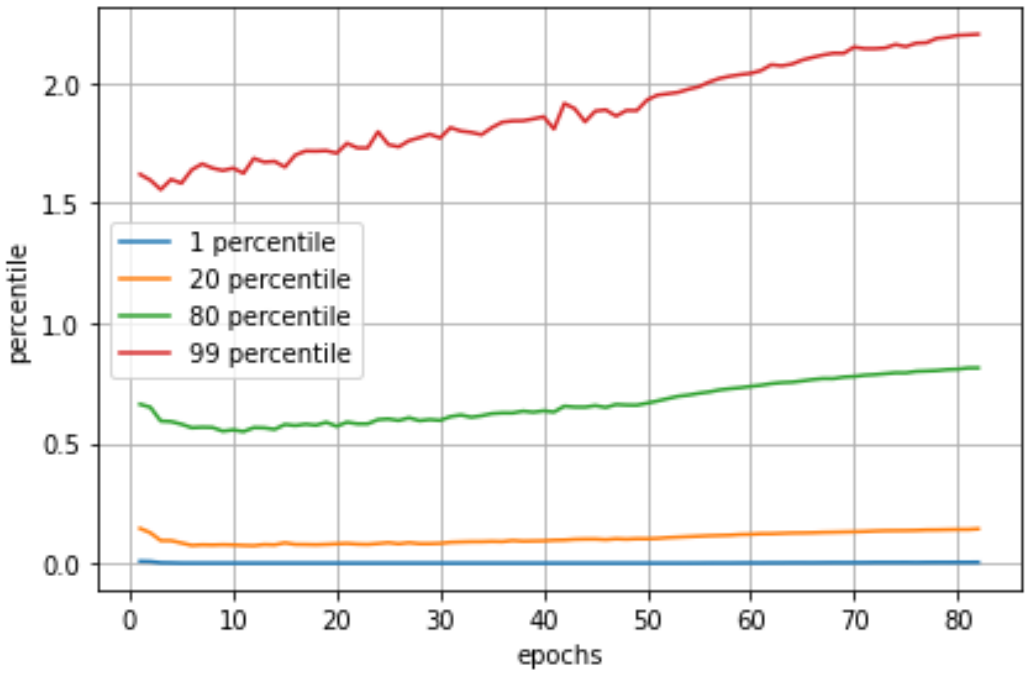

FIGURE 20: BN percentile plot

2.3 Effect of Dropout

Dropout is crucial for good generalisation performance as the features are learnt independent of each other. We can verify this by these empirical results.

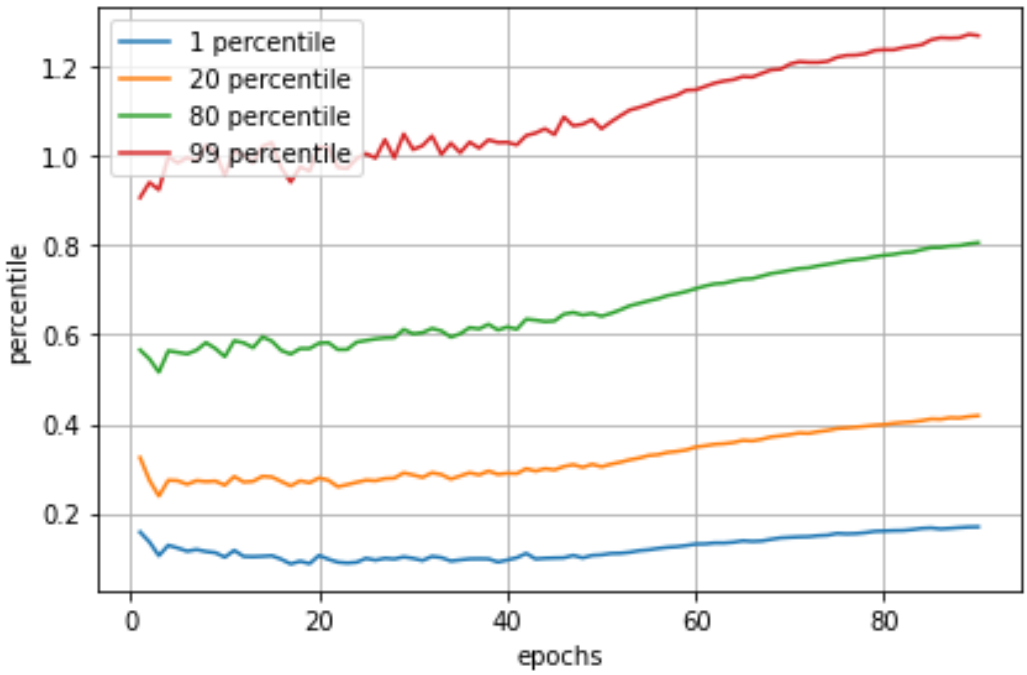

FIGURE 21: IN percentile plot

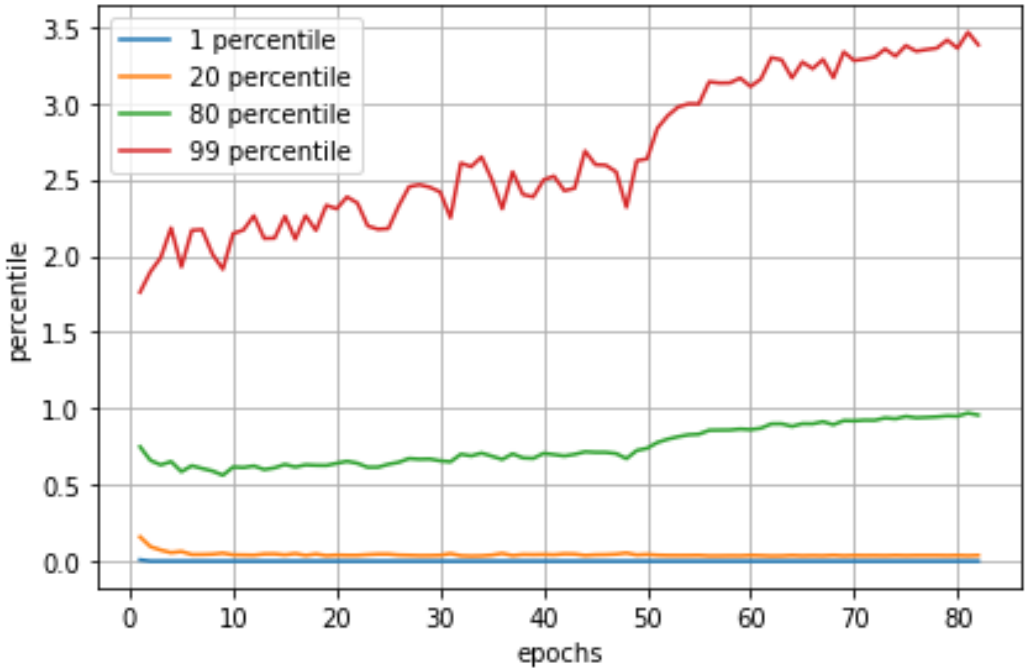

FIGURE 22: BIN percentile plot

2.3.1 Random word embeddings without dropout

Train Accuracy: 0.845, f1_micro: 0.841, f1_macro: 0.603
Val Accuracy: 0.751, f1_micro: 0.778, f1_macro: 0.560

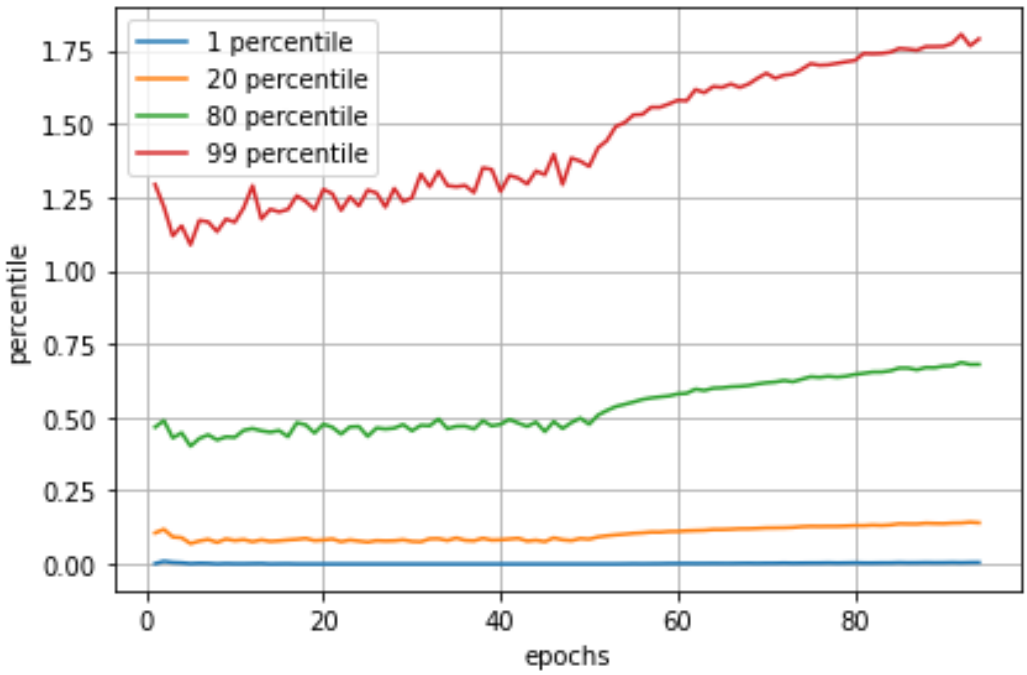

FIGURE 23: LN percentile plot

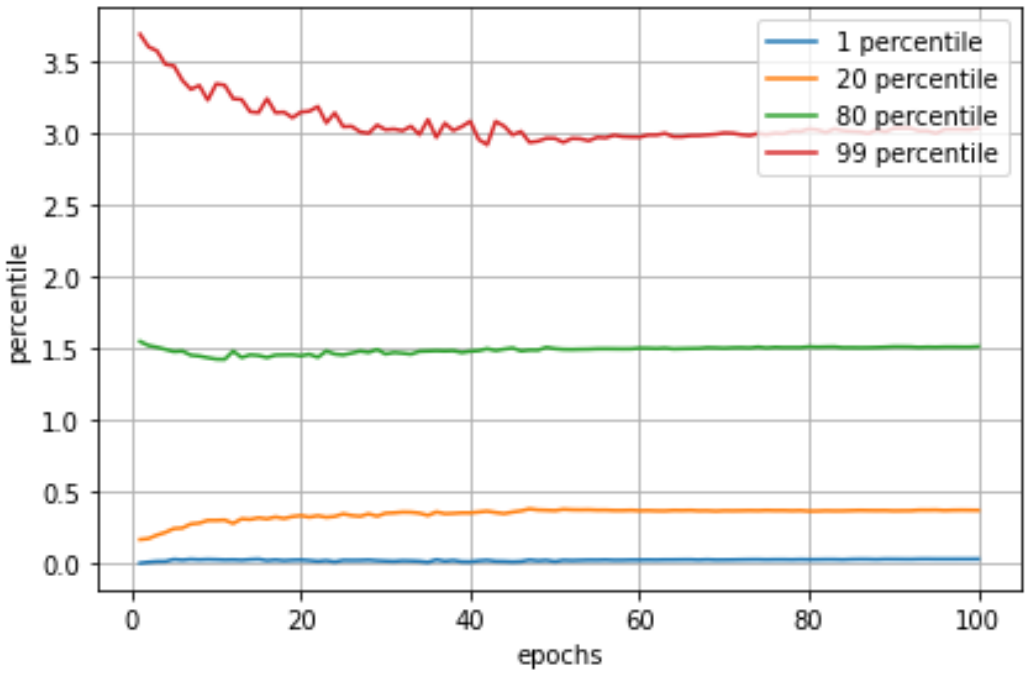

FIGURE 24: GN percentile plot

Test Accuracy: 0.752, f1_micro: 0.775, f1_macro: 0.544
Test Micro F1 and Macro F1 for Random embeddings with dropout are 0.806>0.775 is 0.612>0.544.

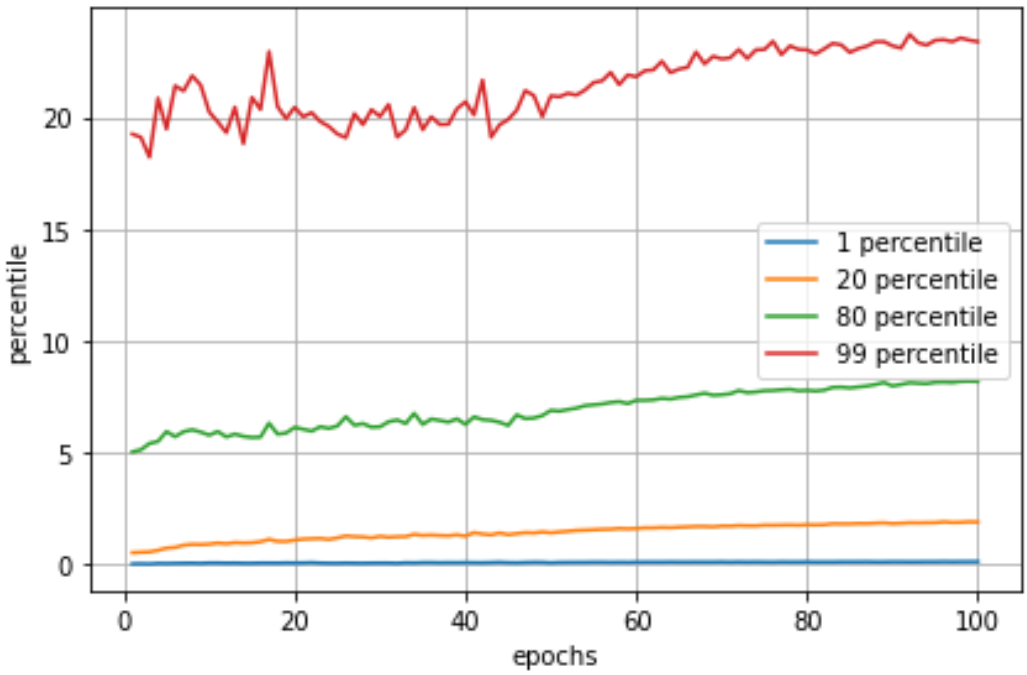

FIGURE 25: NN percentile plot

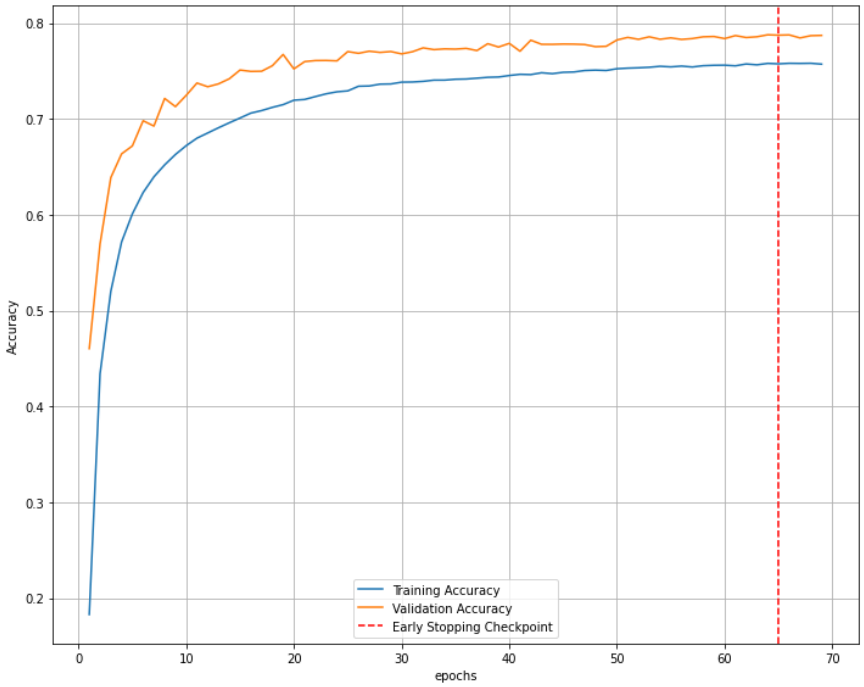

FIGURE 26: Accuracy with epochs for simple BiLSTM with Glove embeddings and 0.5 dropout

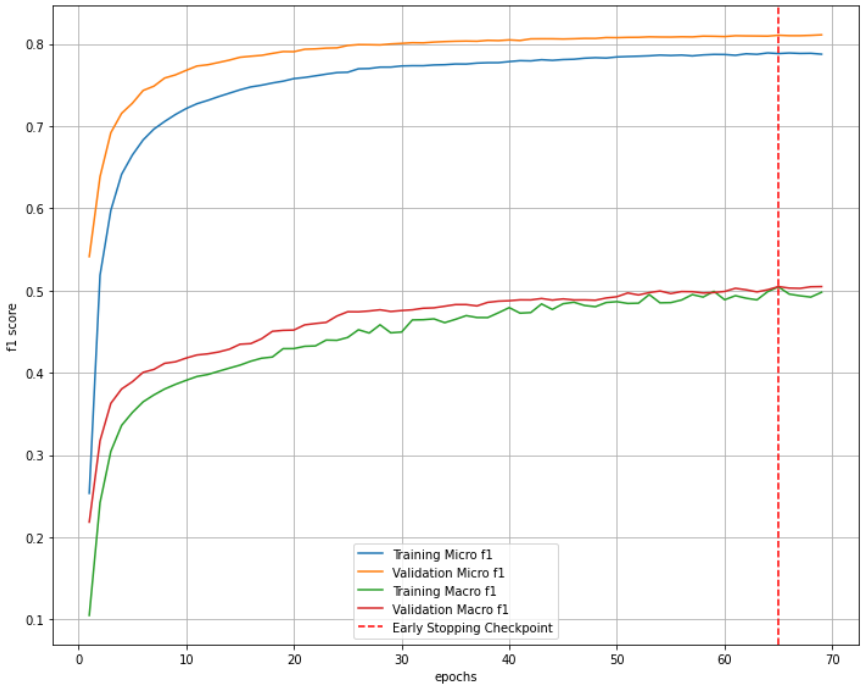

FIGURE 27: F1 scores with epochs for simple BiLSTM with Glove embeddings and 0.5 dropout

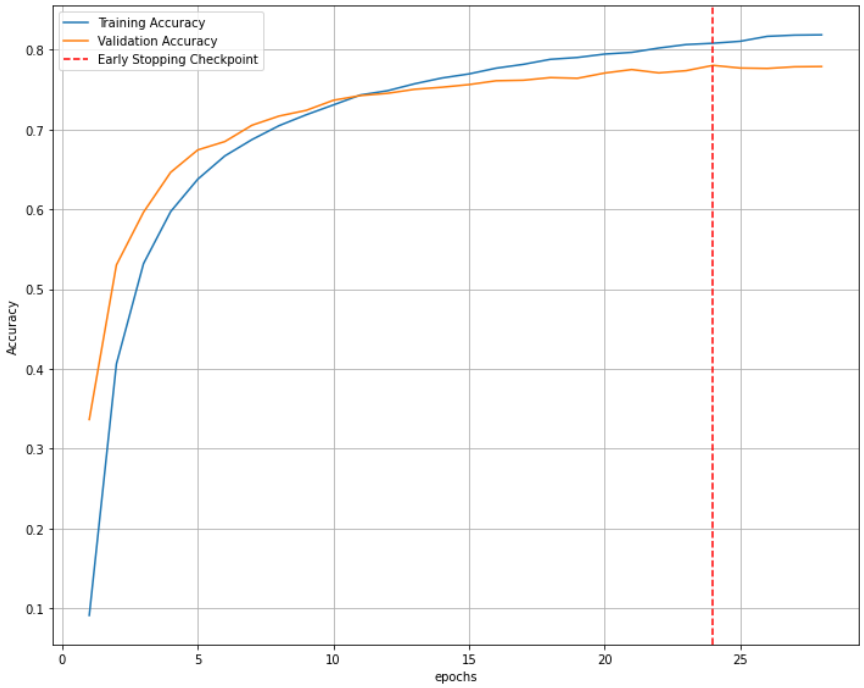

FIGURE 28: Accuracy with epochs for simple BiLSTM with Random embeddings and 0.5 dropout

2.3.2 Pre-trained word embeddings without dropout

Train Accuracy: 0.8676602852370551, f1_micro: 0.865, f1_macro: 0.640
Val Accuracy: 0.806, f1_micro: 0.817, f1_macro: 0.606

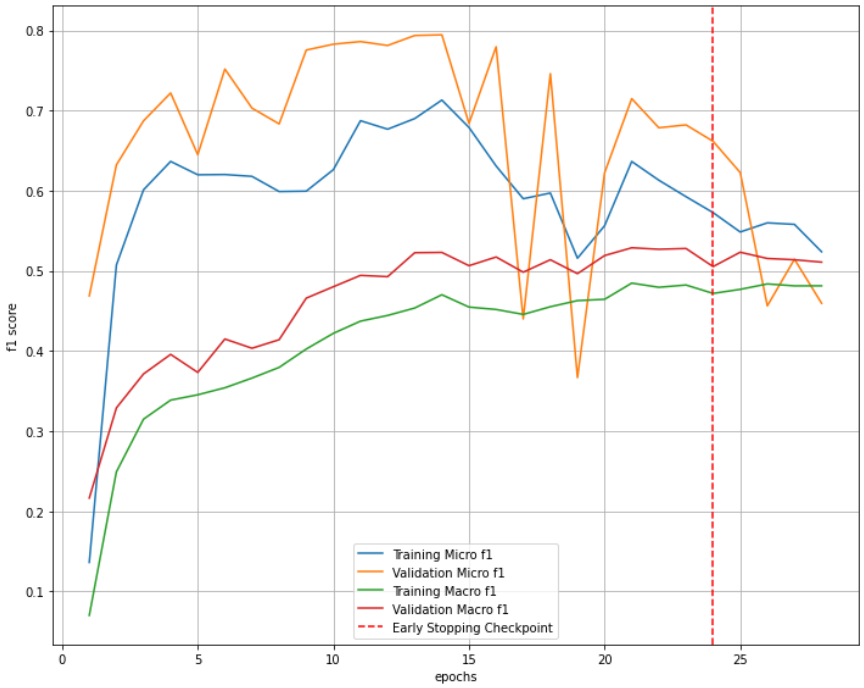

FIGURE 29: F1 scores with epochs for simple BiLSTM with Random embeddings and 0.5 dropout

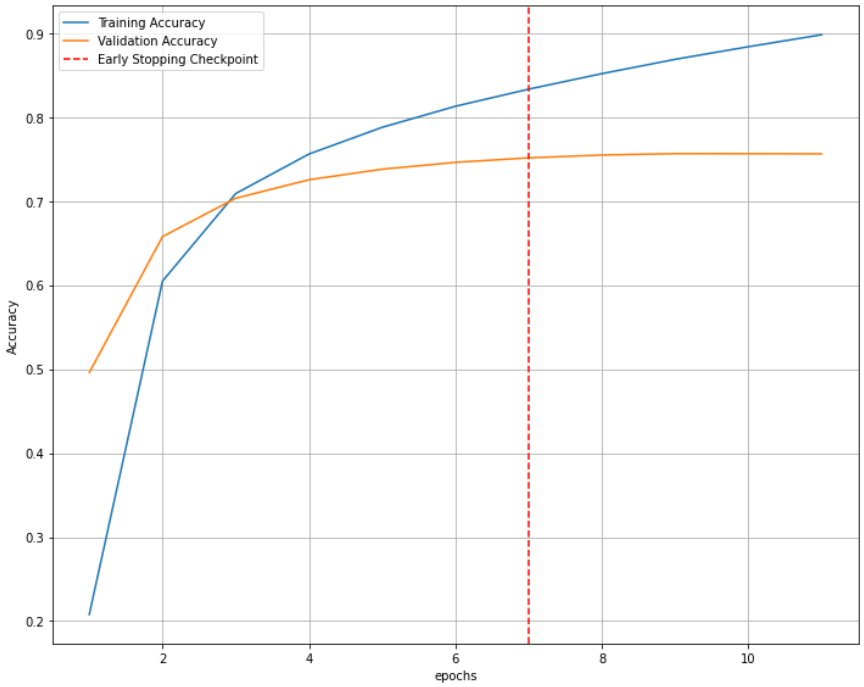

FIGURE 30: Accuracy with epochs for simple BiLSTM with Random embeddings and 0 dropout

Test Accuracy: 0.808, f1_micro: 0.817, f1_macro: 0.596

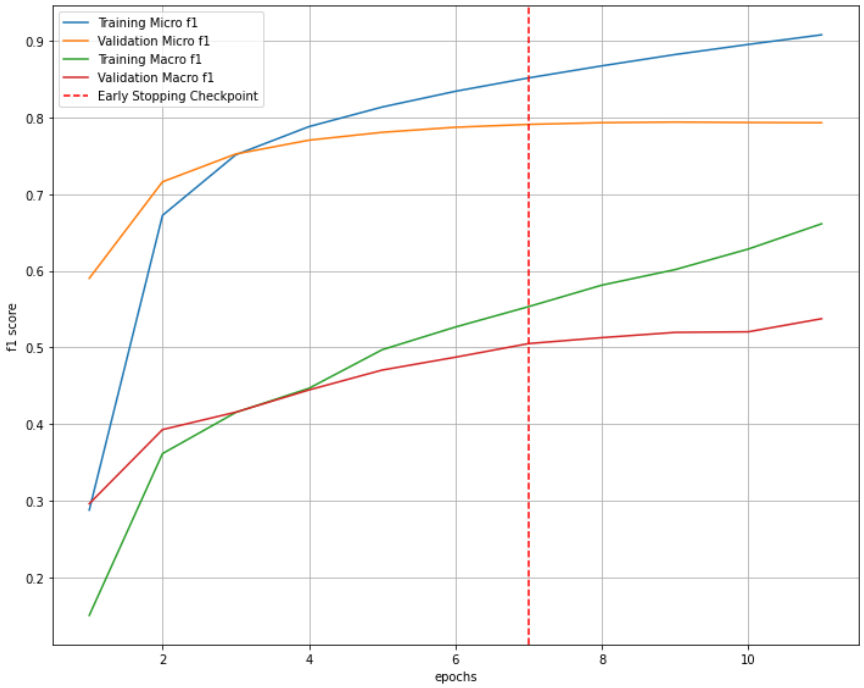

FIGURE 31: F1 scores with epochs for simple BiLSTM with Random embeddings and 0 dropout

An improvement of -0.002 in Micro F1 and + 0.036 in Macro F1 is observed for pre-trained embeddings with dropout.

2.4 Char level features

Character level features are formed using a BiLSTM. The forward and reverse output of the last cell in the direction is concatenated. Further, the Character level features are concatenated with pre-trained word level embeddings. Thus, effective input size here is increased which in our case is 100 + 25 + 25. Character level features extract the morphological information from the words. It is particularly useful in morphologically rich languages. A seperate character vocabulary is formed using all the character in the training file. We see that using dropout is essential here, as it promotes character level feature learning independent of pre-trained word level features. It is also observed in these results that there is no improvement by using char level without dropout. Micro F1 and Macro F1 for Pre-trained word embeddings without dropout are 0.817 and 0.596. This performance is identical to that of Character level features with pre-trained embeddings wthout dropout.

2.4.1 Char level embeddings with dropout

The performance for this case with dropout is significantly better than without dropout. Train Accuracy: 0.87, f1_micro: 0.87, f1_macro: 0.68

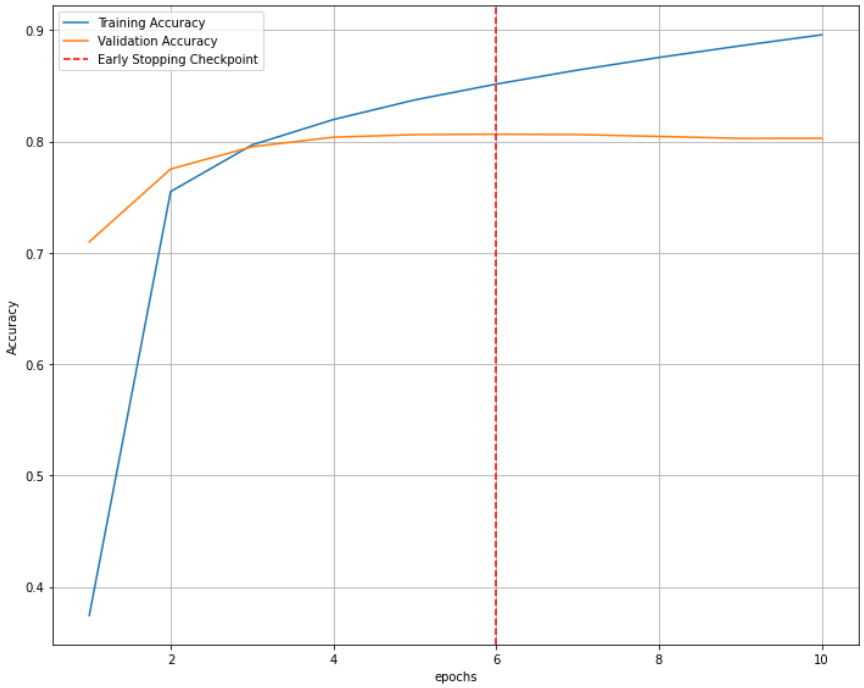

FIGURE 32: Accuracy with epochs for simple BiLSTM with Glove embeddings and 0 dropout

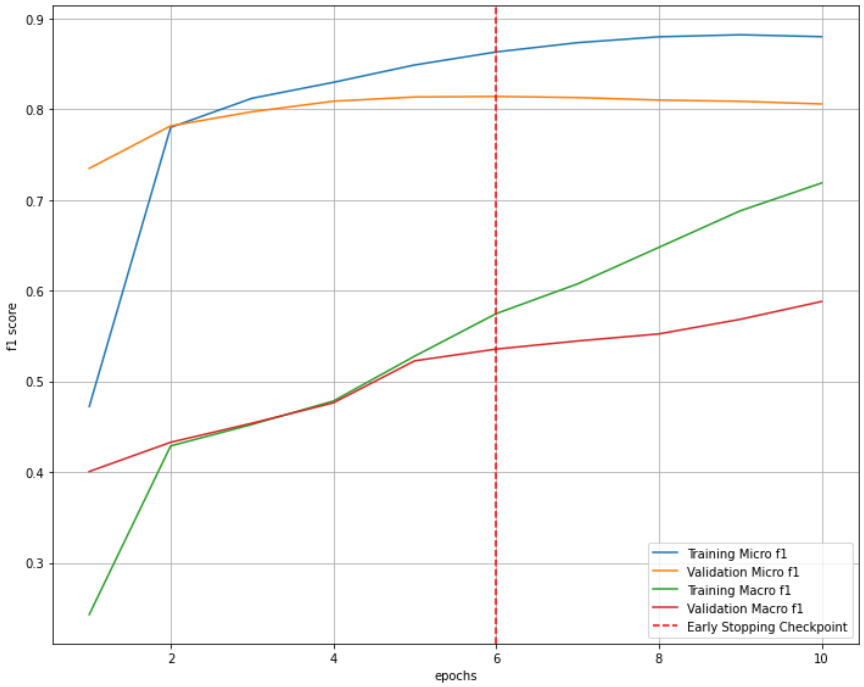

FIGURE 33: F1 scores with epochs for simple BiLSTM with Glove embeddings and 0 dropout

Val Accuracy: 0.820, f1_micro: 0.834, f1_macro: 0.635
Test Accuracy: 0.821, f1_micro: 0.831, f1_macro: 0.628

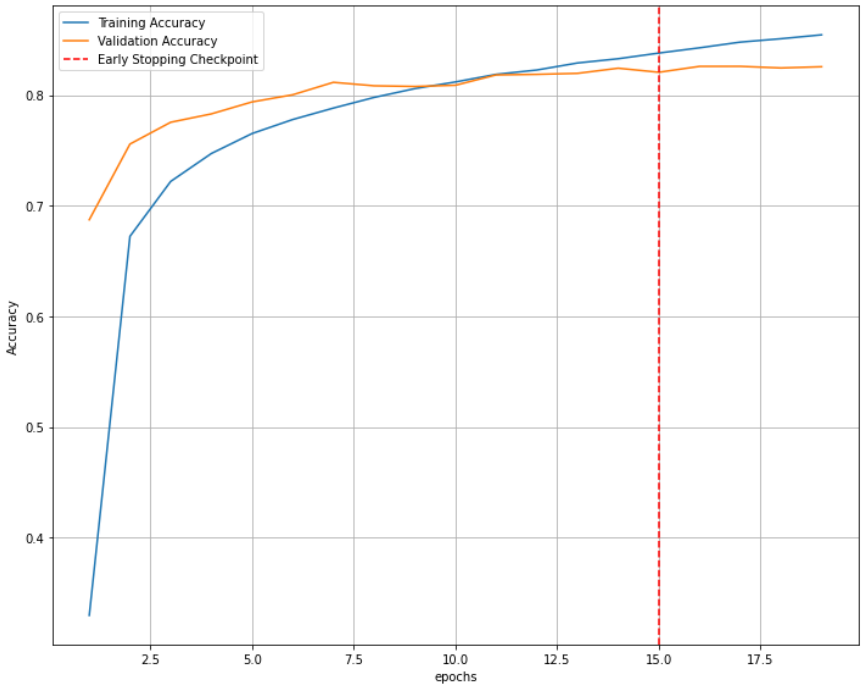

FIGURE 34: Accuracy with epochs for simple BiLSTM with character level features and 0.5 dropout

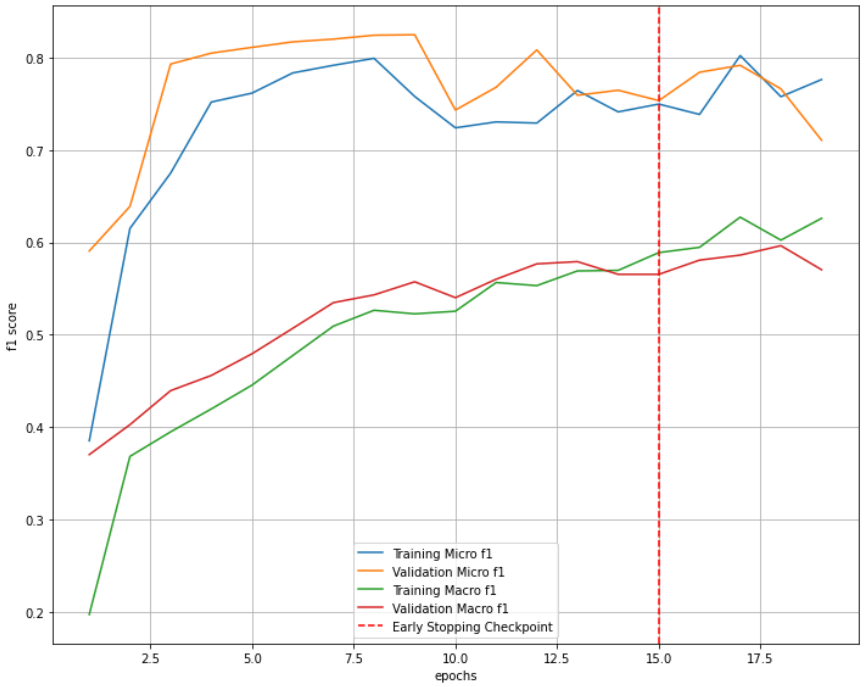

FIGURE 35: F1 scores with epochs for simple BiLSTM with character level features and 0.5 dropout

2.4.2 Char level embeddings without dropout

Train Accuracy: 0.858, f1_micro: 0.856, f1_macro: 0.625
Val Accuracy: 0.805, f1_micro: 0.816, f1_macro: 0.609
Test Accuracy: 0.807, f1_micro: 0.814, f1_macro: 0.586

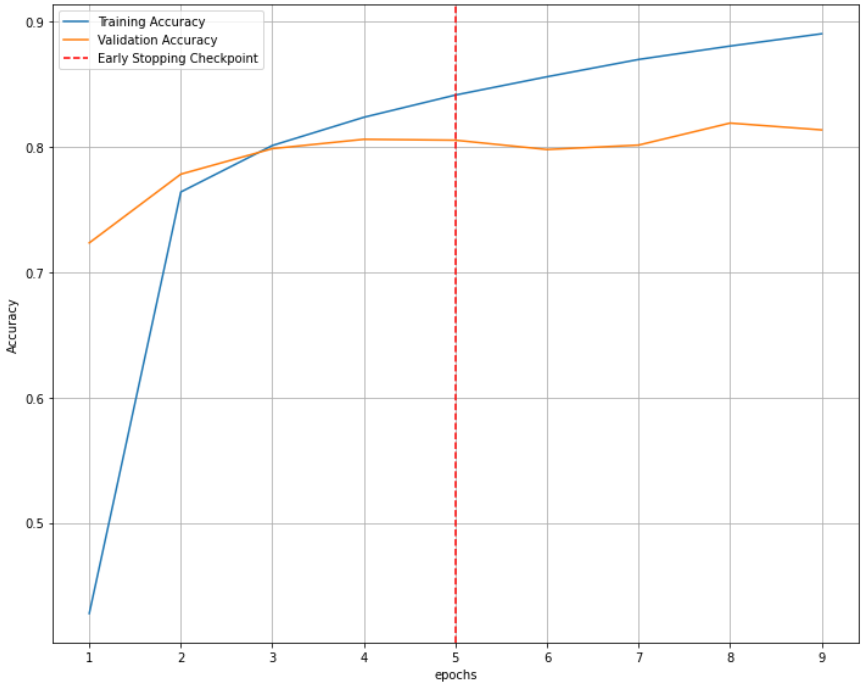

FIGURE 36: Accuracy with epochs for simple BiLSTM with character level features and 0 dropout

2.5 Layer Normalization in LSTM

Layer normalisation is performed on all the gates inside the LSTM Cell.

2.5.1 Layer normalised LSTM Cell with Pre-trained word embeddings along with character level features and dropout

Train Accuracy: 0.882, f1_micro: 0.878, f1_macro: 0.718
Val Accuracy: 0.821, f1_micro: 0.833, f1_macro: 0.670
Test Accuracy: 0.821, f1_micro: 0.830, f1_macro: 0.648
Using Layer normalised LSTM Cell improves Macro F1 by 0.02 when compared to the same case without normalization.

2.5.2 Layer normalised LSTM Cell with Pre-trained word embeddings and dropout

Train Accuracy: 0.880, f1_micro: 0.878, f1_macro: 0.714
Val Accuracy: 0.820, f1_micro: 0.833, f1_macro: 0.656

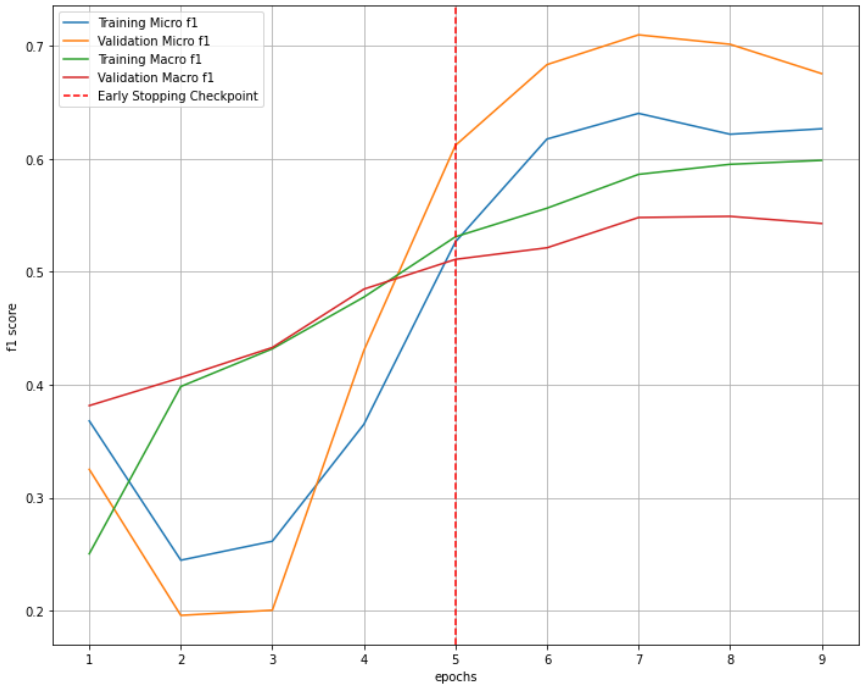

FIGURE 37: F1 scores with epochs for simple BiLSTM with character level features and 0 dropout

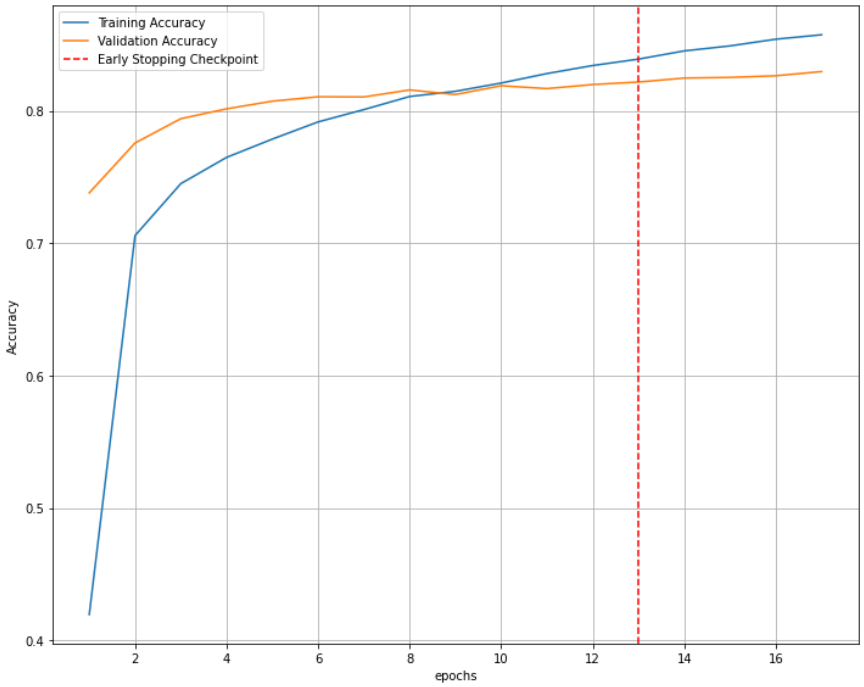

FIGURE 38: Accuracy with epochs for Layer normalised LSTM Cell with Character level features and dropout

Test Accuracy: 0.822, f1_micro: 0.831, f1_macro: 0.640

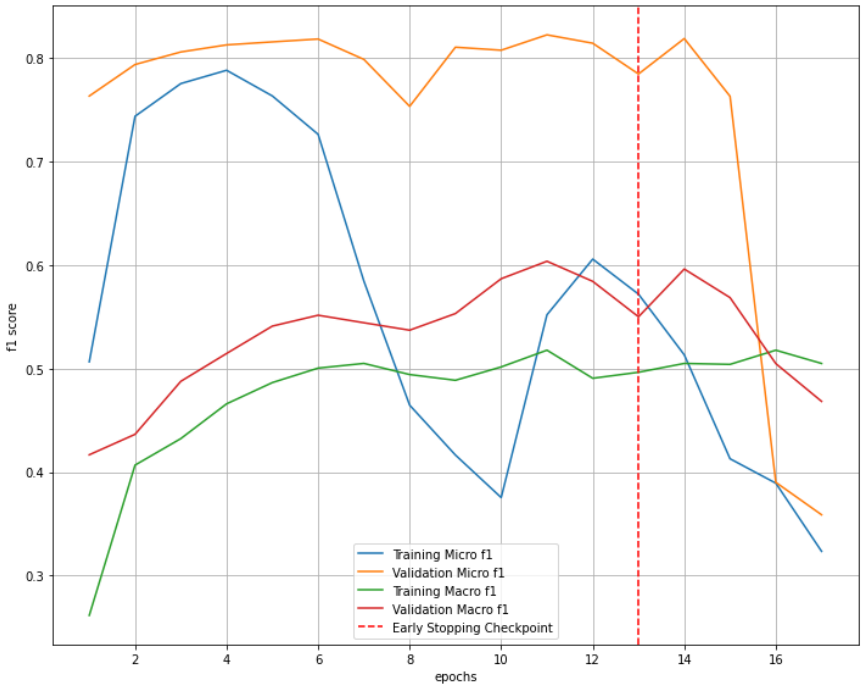

FIGURE 39: F1 scores with epochs for Layer normalised LSTM Cell with Character level features and dropout

Using Layer normalized LSTM Cell improves Micro F1 by +0.016 and Macro F1 by +0.008 when compared to same case without normalization.

2.6 Linear chain CRF

2.6.1 Implementtation Details

2.6.2 CRF with random word embeddings and dropout

Train Accuracy: 0.842, f1_micro: 0.863, f1_macro: 0.678
Val Accuracy: 0.772, f1_micro: 0.818, f1_macro: 0.627
Test Accuracy: 0.775, f1_micro: 0.817, f1_macro: 0.630
There is improvement of 0.018 in Macro F1 and 0.011 in Micro F1 by using CRF layer when compared to the same case without CRF.

2.6.3 CRF with Glove embeddings and dropout

Train Accuracy: 0.87, f1_micro: 0.88, f1_macro: 0.73
Test Accuracy: 0.81, f1_micro: 0.84, f1_macro: 0.66
Test Accuracy: 0.82, f1_micro: 0.84, f1_macro: 0.65
We observe an improvement of +0.025 in Micro F1 and +0.018 in Macro F1 when compared with the same case without CRF layer.

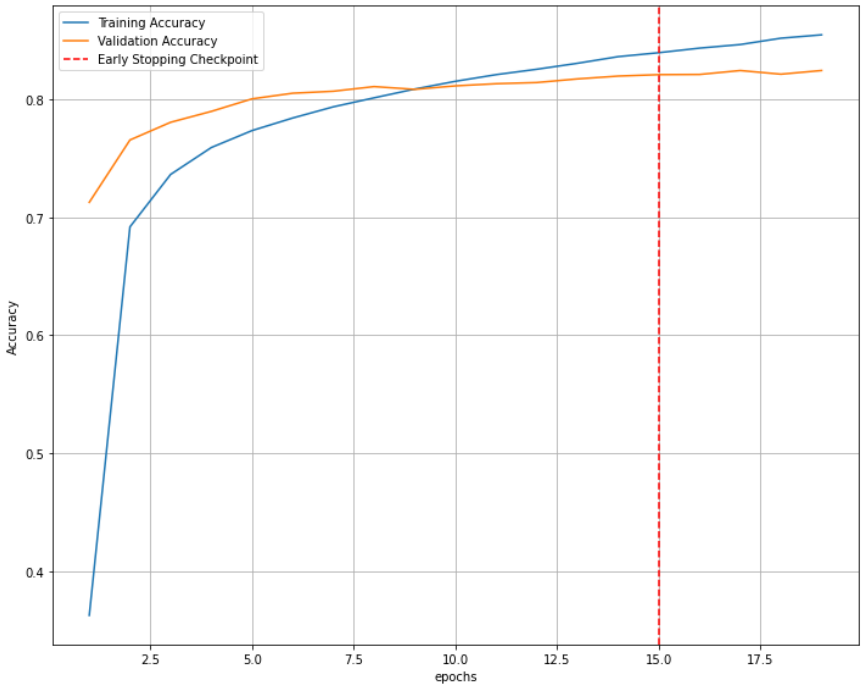

FIGURE 40: Accuracy with epochs for Layer normalised LSTM Cell with glove embeddings and dropout

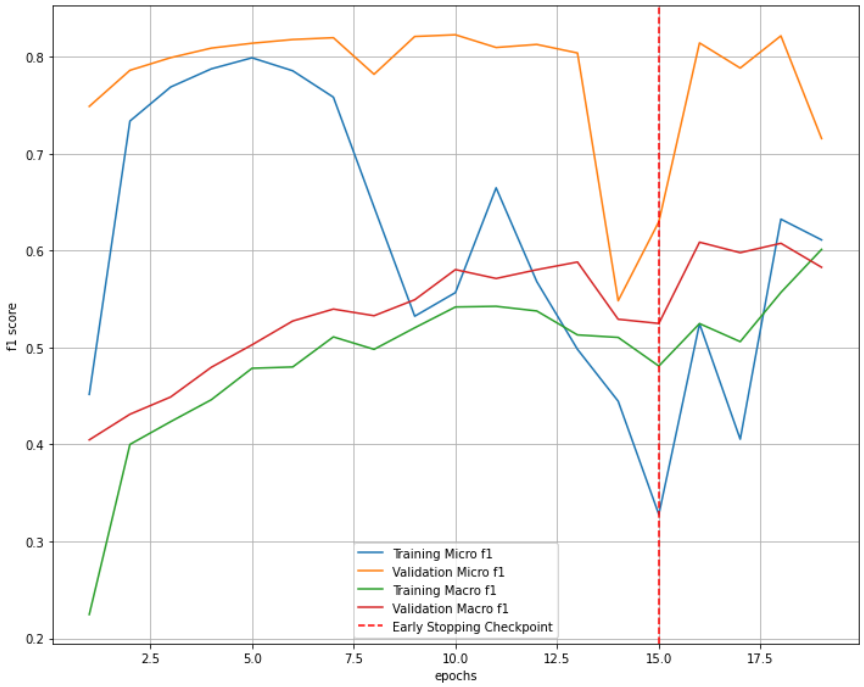

FIGURE 41: F1 scores with epochs for Layer normalised LSTM Cell with glove embeddings and dropout

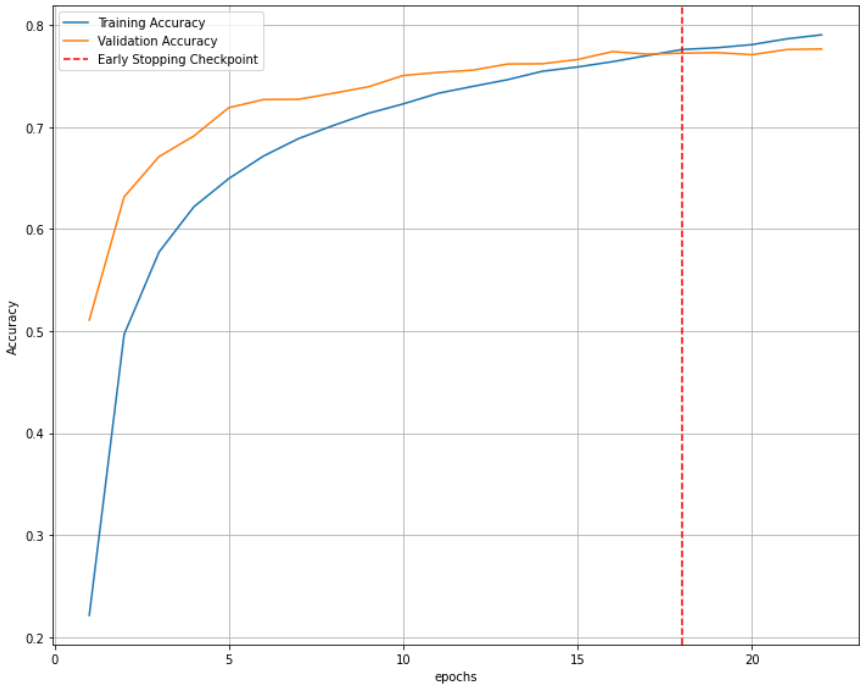

FIGURE 42: Accuracy with epochs for Linear Chain CRF with random word embeddings and dropout

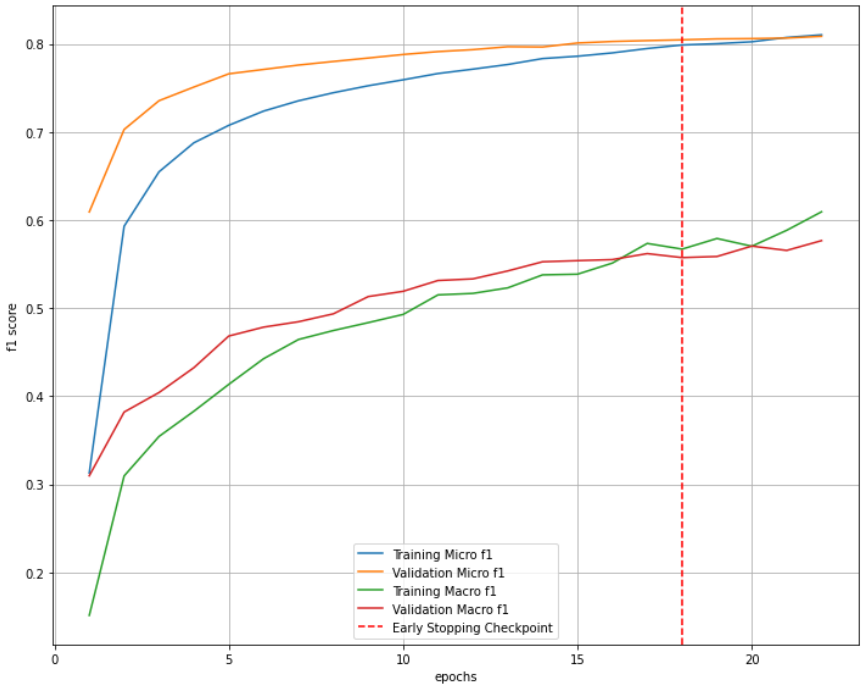

FIGURE 43: F1 scores with epochs for Linear Chain CRF with random word embeddings and dropout

2.6.4 CRF with Character level features and dropout

Test Accuracy: 0.939, f1_micro: 0.939, f1_macro: 0.902
Test Accuracy: 0.828, f1_micro: 0.851, f1_macro: 0.682

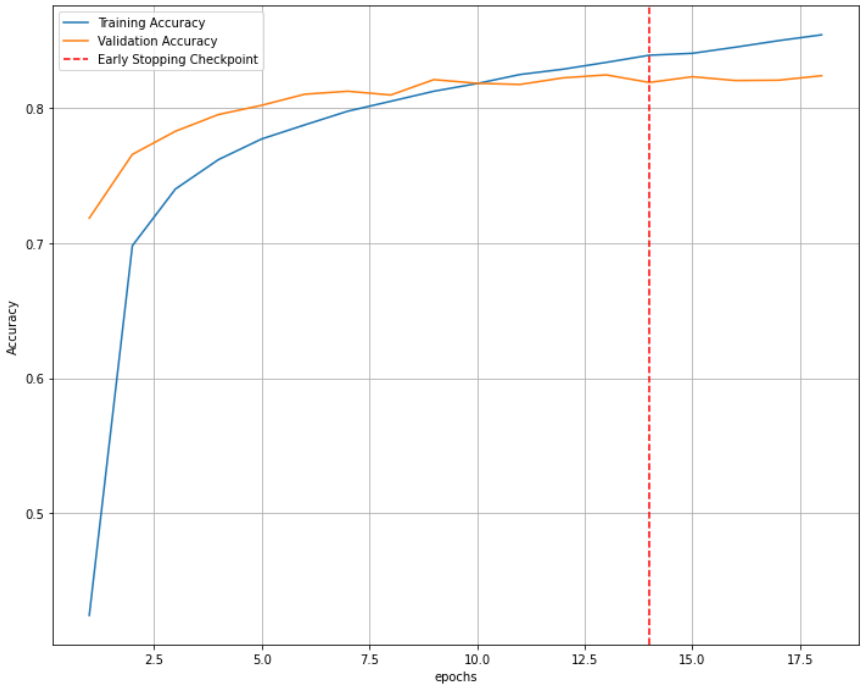

FIGURE 44: Accuracy with epochs for Linear Chain CRF with Glove embeddings and dropout

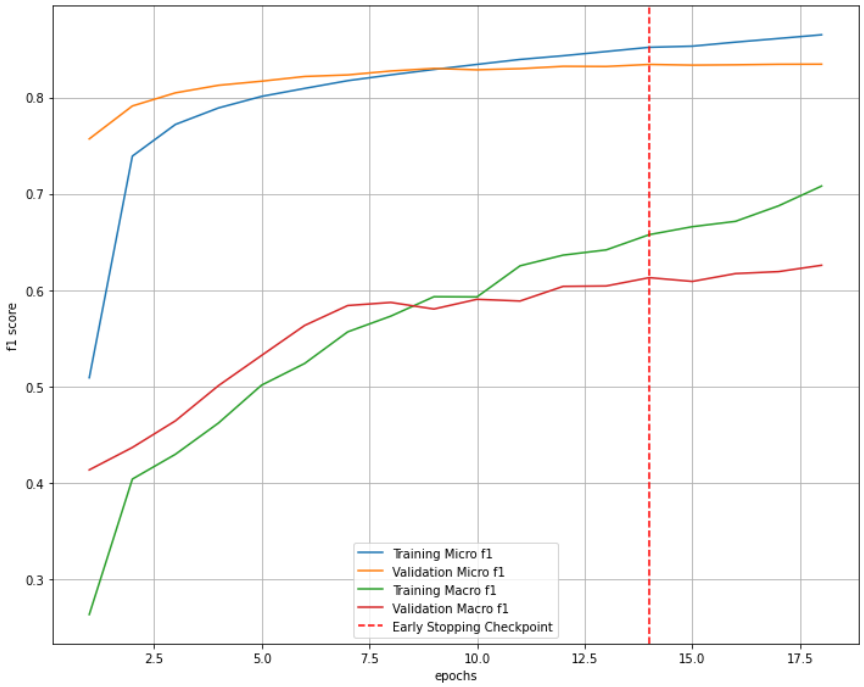

FIGURE 45: F1 scores with epochs for Linear Chain CRF with Glove embeddings and dropout

Test Accuracy: 0.829, f1_micro: 0.850, f1_macro: 0.685
There is an improvement of +0.016 in Micro F1 and +0.057 in Macro F1 when compared to the same case without CRF layer.

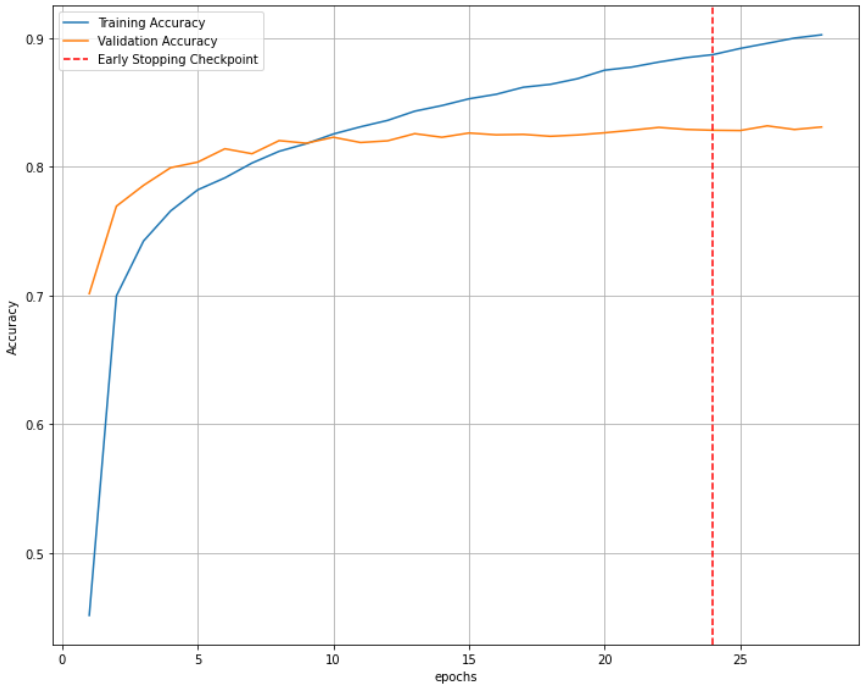

FIGURE 46: Accuracy with epochs for Linear Chain CRF with Character level features and dropout

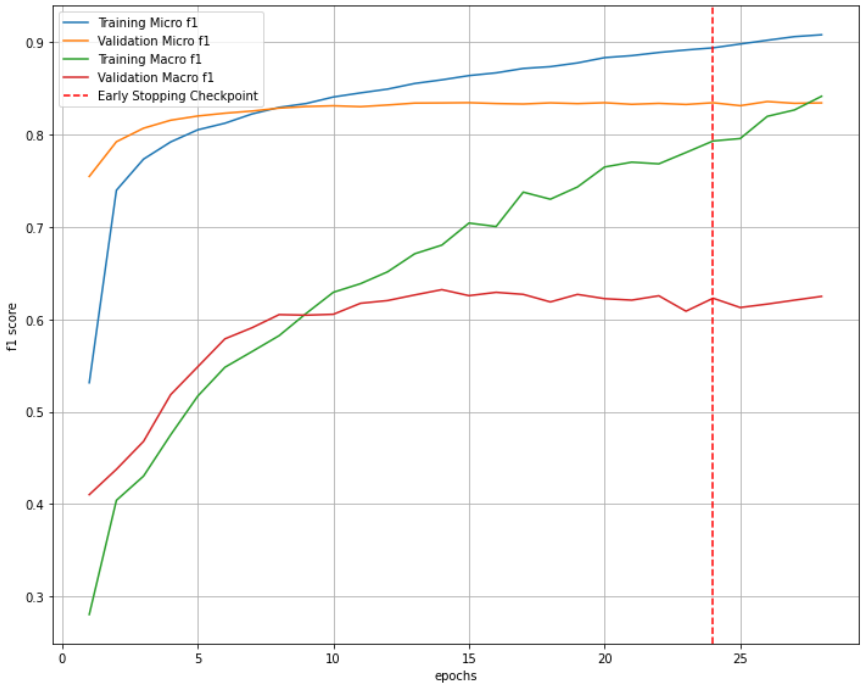

FIGURE 47: F1 scores with epochs for Linear Chain CRF with Character level features and dropout

2.6.5 CRF with Layer normalized LSTM Cell with Glove embeddings and dropout

Test Accuracy: 0.908, f1_micro: 0.913, f1_macro: 0.842
Test Accuracy: 0.823, f1_micro: 0.849, f1_macro: 0.696
Test Accuracy: 0.825, f1_micro: 0.849, f1_macro: 0.688
There is an improvement of +0.018 in Micro F1 and +0.048 in Macro F1 when compared to the same case without CRF layer.

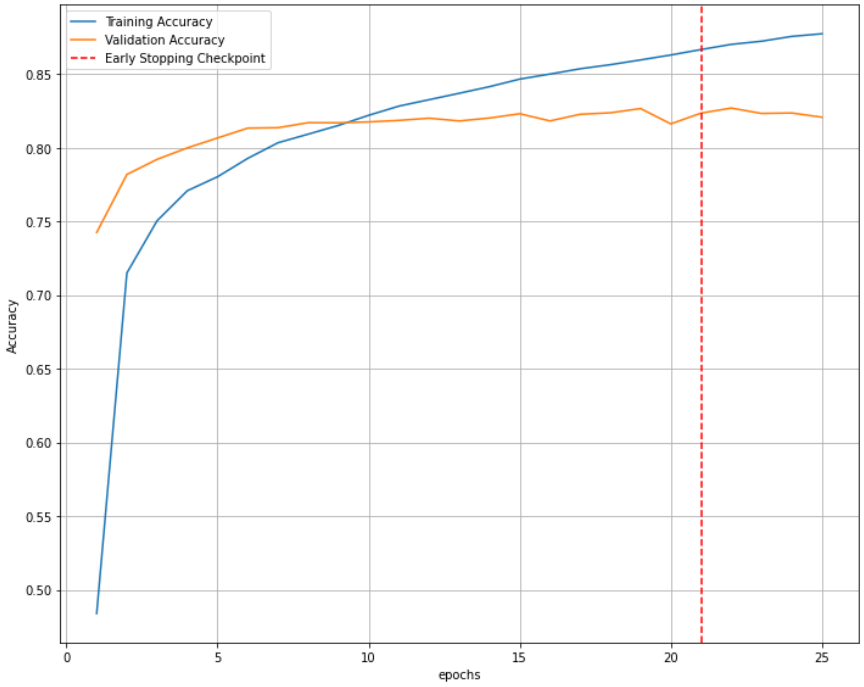

FIGURE 48: Accuracy with epochs for Linear Chain CRF with Layer normalised LSTM Cell, Glove embeddings and dropout

2.7 Discussion

All the scores are reported for the test set. Random word embeddings are to be assumed by default.

3 Path to the folder containing all of the models

https://drive.google.com/drive/folders/1Xx3mQO9j9el3E-Qa6Ki70LnveHCeWgpR?usp=sharing

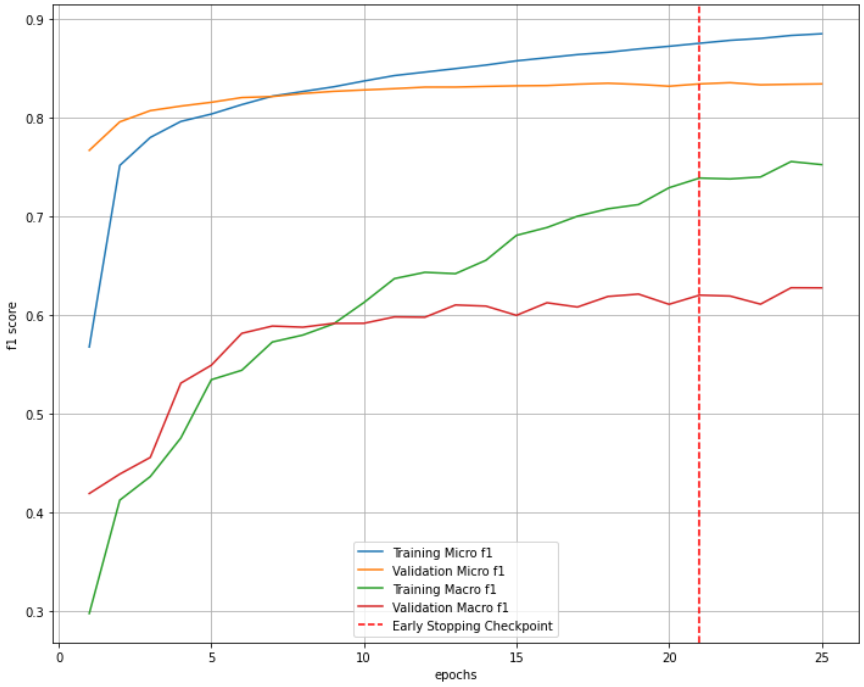

FIGURE 49: F1 scores with epochs for Linear Chain CRF with Layer normalised LSTM Cell, Glove embeddings and dropout

	Case	Micro F1	Macro F1	Accuracy
1	-	0.775	0.544	0.752
2	Glove	0.817	0.596	0.808
3	Dropout	0.806	0.612	0.783
4	Glove + Dropout	0.815	0.632	0.795
5	Glove + Char	0.814	0.586	0.807
6	Glove + Char + dropout	0.831	0.628	0.821
7	Glove + Layer norm + Dropout	0.831	0.640	0.822
8	Glove + Char + Layer norm + Dropout	0.830	0.648	0.821
9	CRF + Dropout	0.817	0.63	0.775
10	Glove + CRF + Dropout	0.840	0.651	0.821
11	Glove + Char + CRF + Dropout	**0.850**	0.685	0.829
12	Glove + Layer norm + CRF + Dropout	0.849	**0.688**	0.825
13	Glove + Char + Layer norm + CRF + Dropout	0.849	0.672	0.824