ASSIGNMENT – 1

Part – 1 (ResNet over Convolutional Networks and different Normalization Schemes)

1.1 Image Classification using Residual Network

ResNet learns the residual mapping by providing the input of k-th layer to (k+2)th layer via residual connections. The skip connections help reducing the problem of vanishing gradients that can rise in deep neural networks. The skip connections allow information to travel directly between non-adjacent layers allowing for more efficient training of deep neural networks.

First, I experimented with the architecture described as follows using Data augmentation.

Hyperparameters : Batch size – 32, n = 2 (No of layers in total = 6n+2) , r=25, epochs = 50.

Data Augmentation : I experimented with random rotation of images upto 20 degrees, Horizontal flip and vertical flip of images.

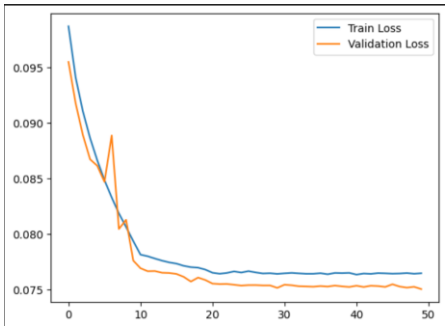
Criteria : The loss function used here is Cross Entropy Loss.

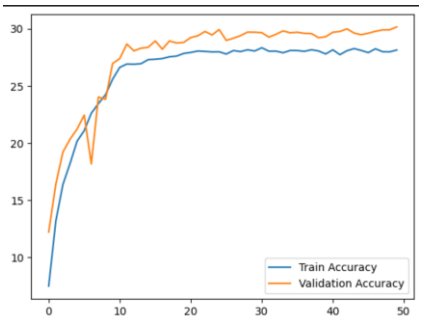
Optimizer : The optimizer experimented here is SGD with learning rate = 0.0001

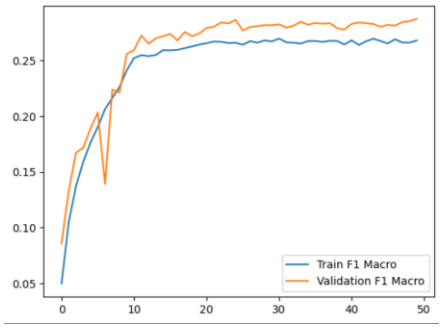
Scheduler : I experimented with StepLR scheduler with step size 10.

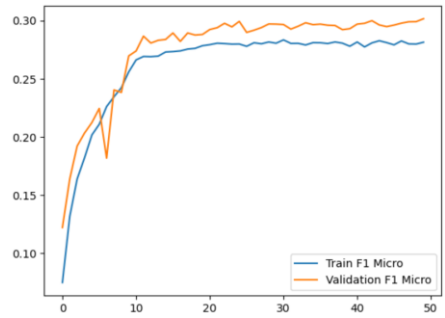
The results I got with the above setup are as follows on the training and validation sets are as follows.

	Loss	Accuracy	Macro F1	Micro F1
Training set	0.076	28.123	26.793	28.123
Validation set	0.075	30.143	28.741	30.143









Since the accuracies are low, next I have experimented with **Adam's optimizer** having **CrossAnnealingLR** as the scheduler in all my experiments with the same learning rate **10^-4**.

Using PyTorch's Batch Normalization, with no data augmentation the results are as follows.

Also, the final architecture for all the experiments is the same as this.

Hyperparameters : Batch size – 32, n = 2 (No of layers in total = 6n+2) , r=25, epochs = 50.

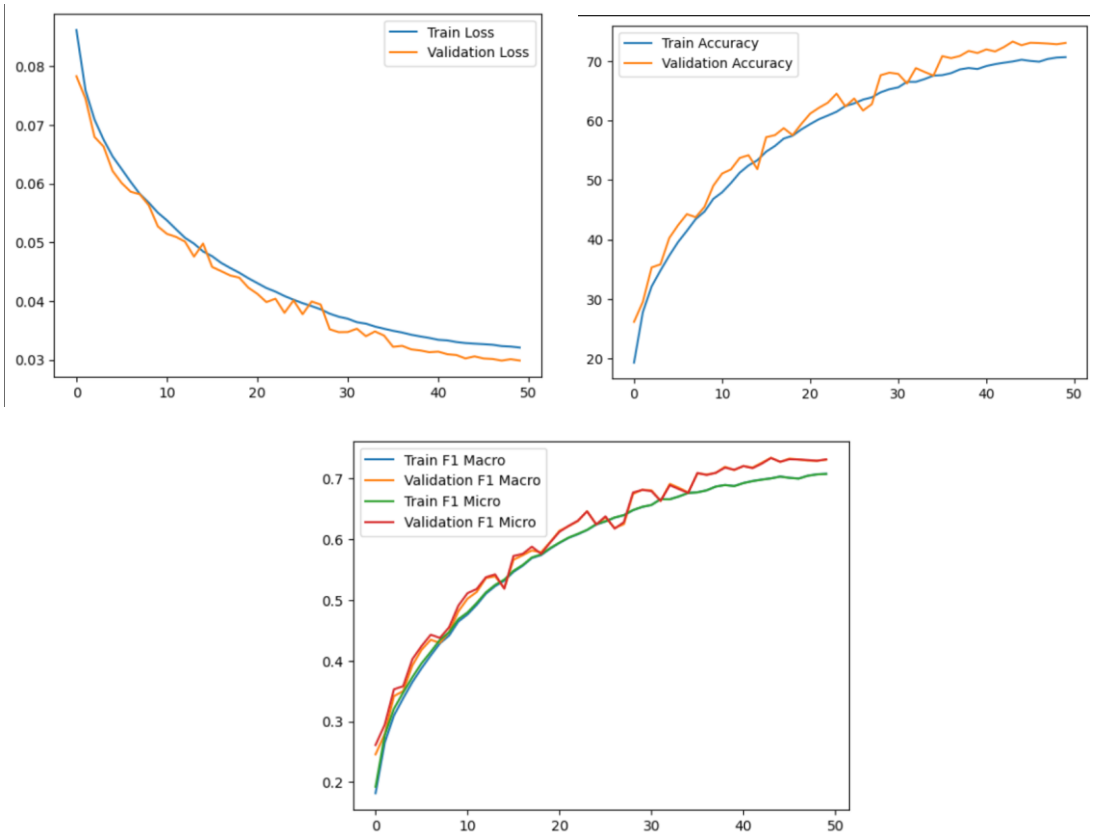
Criteria : The loss function used here is Cross Entropy Loss.

Optimizer : The optimizer experimented here is **Adam** with learning rate = 0.0001

Scheduler : I experimented with **CrossAnnealingLR** scheduler.

	Loss	Accuracy	Macro F1	Micro F1
Training set	0.032	70.704	70.735	70.304
Validation set	0.029	73.101	73.053	73.301

The graphs using PyTorch's BN are as follows :



Observations :

1. The training loss curve decreases smoothly whereas validation loss curve decreases a bit irregularly.

2. The training F1 Micro and Macro are almost the same throughout all the epochs. Similarly with the validation F1 Micro and Macro too.

3. Validation accuracy greater than that of training accuracy tells that the training done is good.

4. We can't comment much about overfitting with these 50 epochs but since the validation accuracy is more than that of training ones, we can say that there is no overfitting.

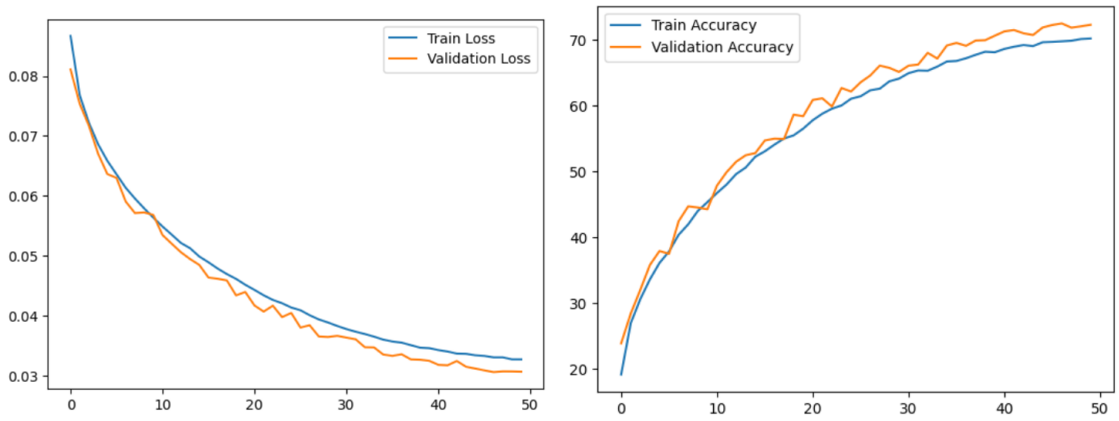
2 Impact of Normalization

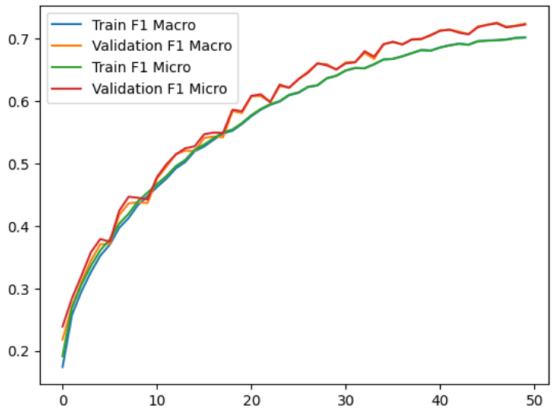
All the six variants NN, BN, IN, BIN, GN, LN are implemented from scratch.

a) Custom Batch Normalisation (BN)

	Loss	Accuracy	Macro F1	Micro F1
Training set	0.032	70.204	70.247	70.704
Validation set	0.030	72.292	72.443	72.101

The graphs are as follows for the Custom Batch Normalisation :





Sanity check of custom batch normalisation in comparison to PyTorch's batch normalisation

When you compare the accuracies, loss values and the F1 micro and the macro values of custom implementation, you can find that the values are ***almost the same*** *as that of PyTorch's implementation.*

This implies that the custom implementation is correct.

Comparison of curves

1. The values of validation accuracies and training accuracies are slightly less in custom's BN.

2. The training loss and validation loss in PyTorch's BN is very slightly lower (very negligible).

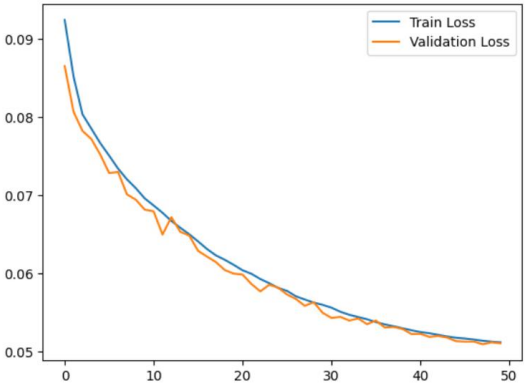
3. Training and validation curves in both the implementations are similar.

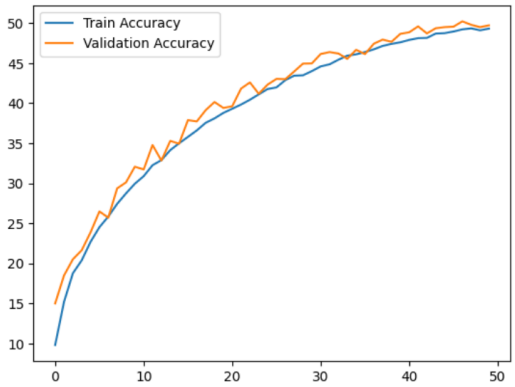
Hence, we can say that the implementation of Batch Normalisation in torch is correct and is performing well.

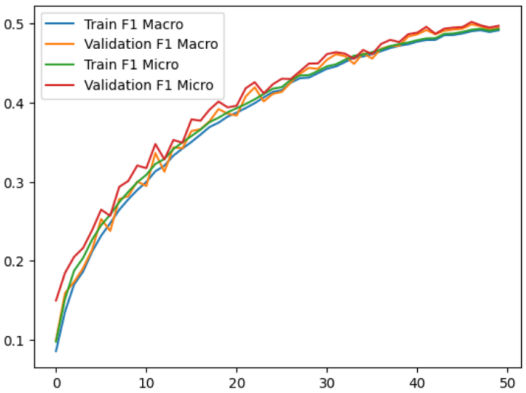
b) No Normalisation (NN)

When there is No Normalisation, the values and the findings are as follows.

	Loss	Accuracy	Macro F1	Micro F1
Training set	0.0511	49.325	49.160	49.326
Validation set	0.0510	49.732	49.460	49.723

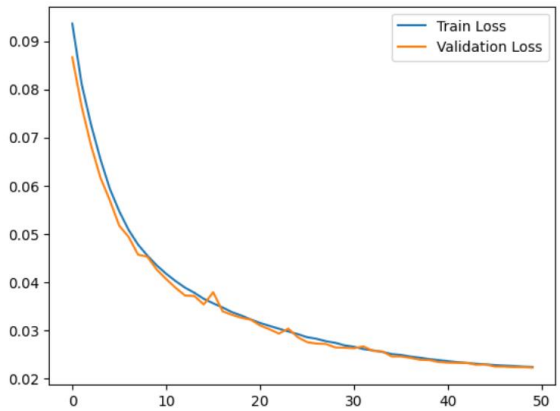


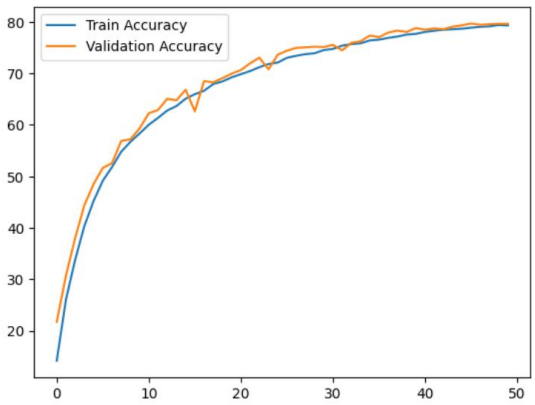


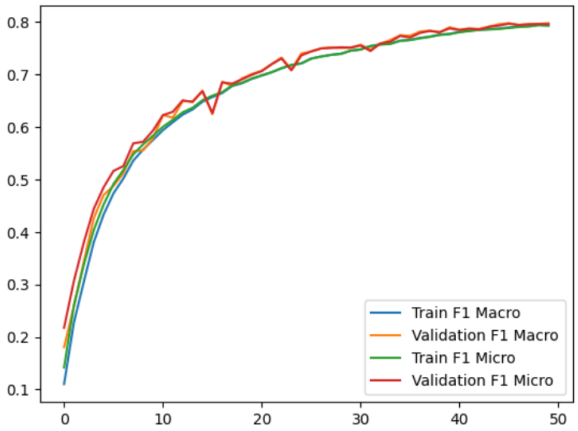


c) Instance Normalisation (IN)

	Loss	Accuracy	Macro F1	Micro F1
Training set	0.0223	79.307	79.374	79.308
Validation set	0.0222	79.614	79.752	79.615

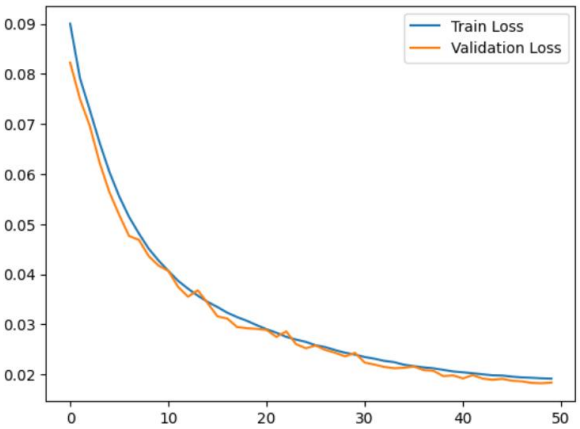


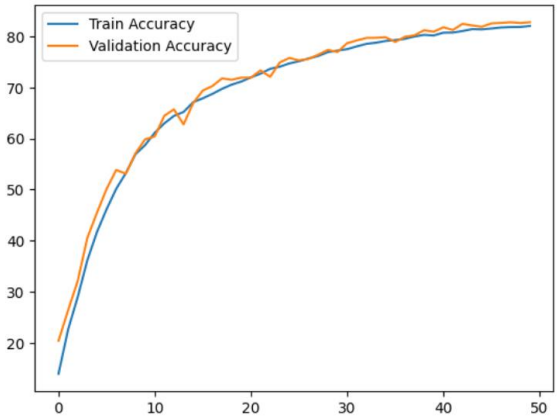


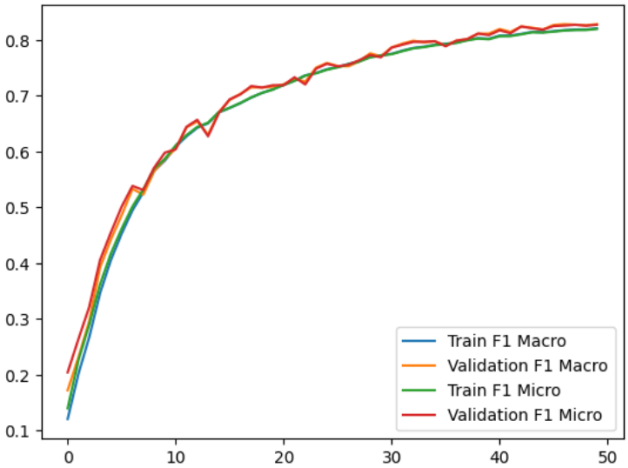


d) Batch Instance Normalisation (BIN)

	Loss	Accuracy	Macro F1	Micro F1
Training set	0.0193	82.010	82.092	82.011
Validation set	0.0183	82.740	82.850	82.741

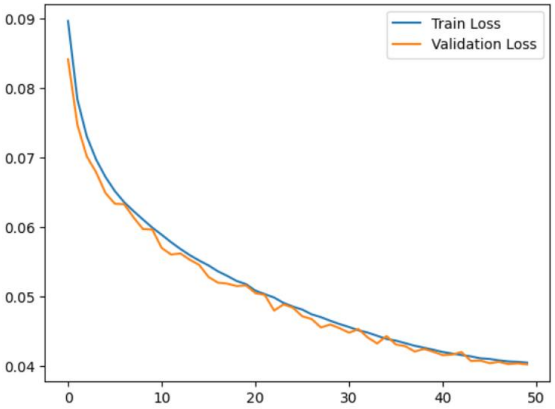


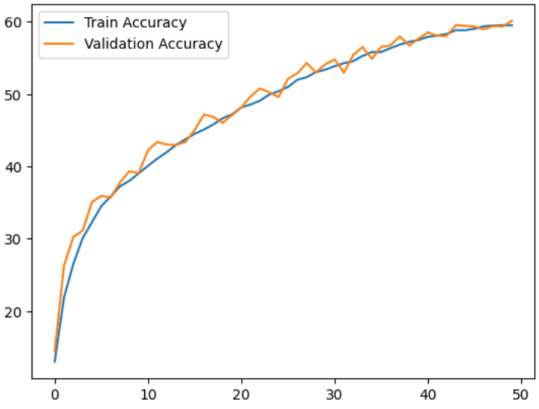


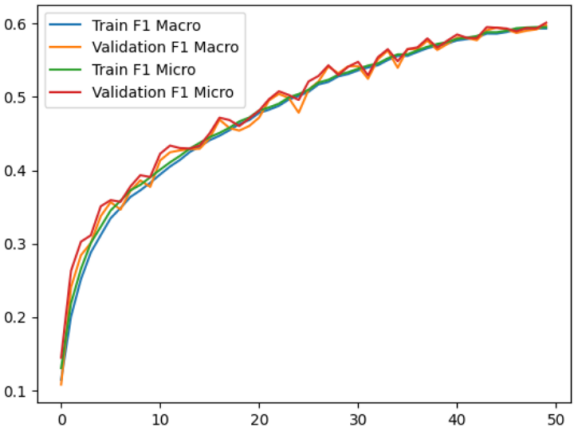


e) Group Normalisation (GN)

	Loss	Accuracy	Macro F1	Micro F1
Training set	0.0404	59.520	59.345	59.521
Validation set	0.0402	60.133	59.785	60.132

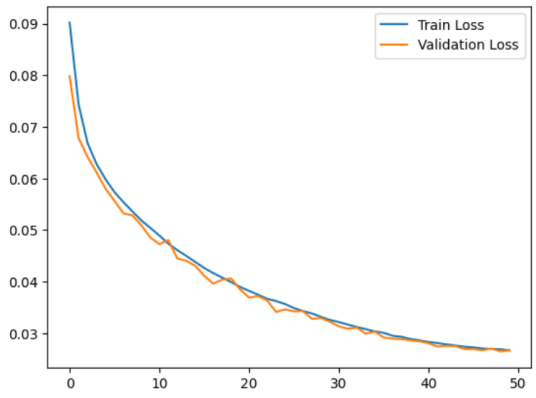


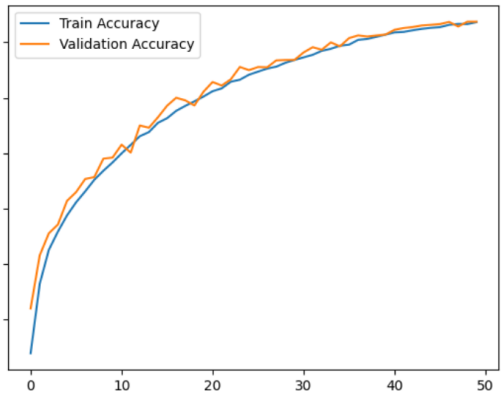


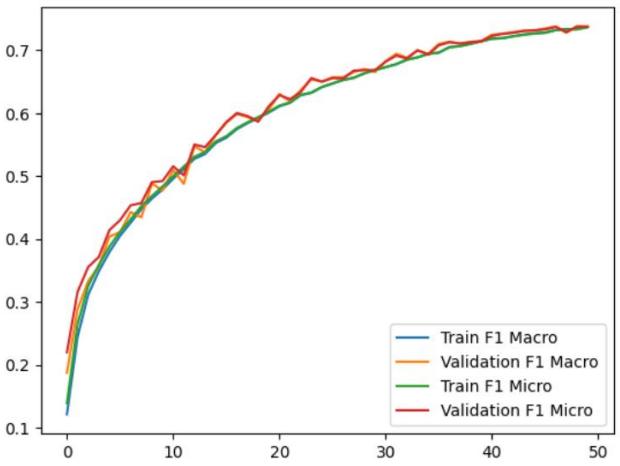


f) Layer Normalisation (LN)

	Loss	Accuracy	Macro F1	Micro F1
Training set	0.0266	73.637	73.683	73.638
Validation set	0.0266	73.719	73.765	73.720







Comparison of Loss Curves for all the variations

1. Initial rate of decay is the highest for BIN, then IN, then followed by LN, BN, NN, GN.

2. The final loss values are in the decreasing order of NN, GN, BN, LN, IN, BIN. This implies that NN decays initially faster but has higher training loss and lowest accuracy than all the other schemes.

3. Final loss values are the highest for NN scheme which tells us that various normalisation schemes are better and offers better training as compared to no normalisation.

4. From all the graphs obtained, we can see that BIN and IN have offered the best training and the performance is very good. The NN performs the worst as expected.

Accuracies of all the models when put together

Model	**Training**		**Validation**	
	Accuracy	**Macro F1**	**Accuracy**	**Macro F1**
BN	70.204	70.247	72.292	72.443
NN	49.325	49.160	49.732	49.460
IN	79.307	79.374	79.614	79.752
BIN	82.010	82.092	82.740	82.850
GN	59.520	59.345	60.133	59.785
LN	73.637	73.683	73.719	73.765

Observations :

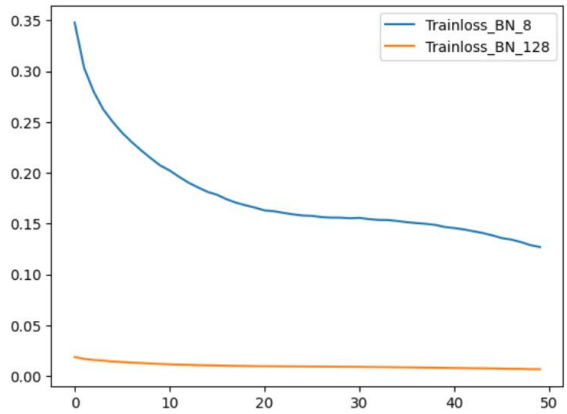
1. As seen here, BIN performs the best followed by IN, LN, BN, GN, NN.

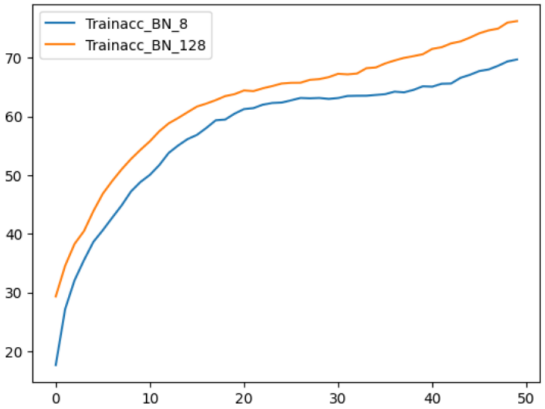
2. The accuracy values follow the same order as that of with the validation loss order.

Impact of Batch size in BN

After training the BN with batch sizes 8 and 128, the training loss and the accuracy curves are as follows

```
Train Loss_8:  0.12700071408683627 Train Accuracy_8:  69.684196213
Train Loss_128:0.00674350703318187 Train Accuracy_128:76.214433830
```





As seen from the graphs, BN is sensitive to batch size. As the batch size increases, the loss values drop a lot.

Impact of Batch size in GN

After training the BN with batch sizes 8 and 128, the training losses are as follows

```
Train Loss_8:  0.12700071408683627 Train Accuracy_8:  69.684196213
Train Loss_128:0.1174350703318187 Train Accuracy_128: 70.214433830
```

As seen from the values, GN is insensitive to batch size. As the batch size increases, the loss values doesn't change much.
Hence, this is the inference which we can draw from the GN. And GN is better than BN for this reason. As the batch size increases, GN's loss doesn't differ much. Hence, this is the benefit as compared to BN.

Gdrive link :

https://drive.google.com/drive/folders/1ZQGmtFOK6qgjLO3mSNvtIkG-0ADmq0fy?usp=sharing