A Comparison of Batch Normalization and Group Normalization

CS 726 : Advanced Machine Learning Project

Team Name : abnormal

Team Members :
Srivatsan Sridhar - 150070005
Saqib Azim - 150070031
Chinmay Talegaonkar - 15D070046

I. Introduction

Proposed in 2013, batch normalization (BN) is a very commonly used technique to speedup the optimization of deep neural networks. Batch normalization works by normalizing the outputs of a layer of the network, using its mean and variance computed across the batch of training samples used in each iteration. While BN has been shown to give large improvements in the accuracy over many different tasks, it is not well understood as to why it works so well.

Group normalization (GN), proposed recently in 2018, is an alternative method of normalization, mainly designed for convolutional neural networks. Here again, each layer's output is normalized, but the mean and variance are computed over a group of channels in each sample. It has been claimed that group normalization gives better accuracy than batch normalization for several tasks, particularly when low batch sizes are used.

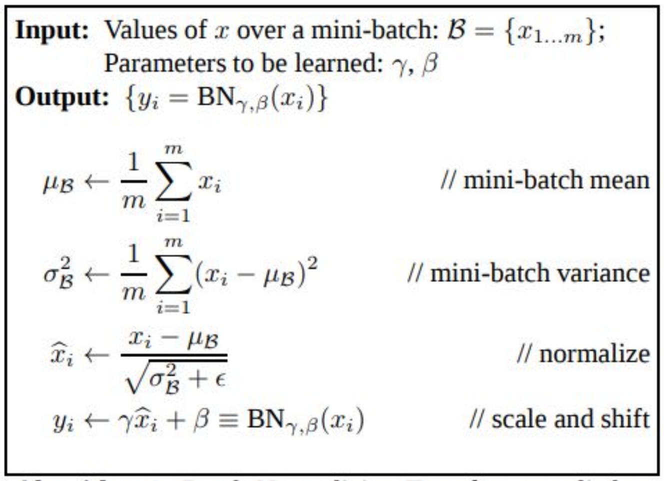
In this project, we study the details about both batch normalization and group normalization. We analyze the reasons why batch normalization and group normalization improve training, empirically and theoretically. We present a set of experiments to observe the effect of both BN and GN on the training accuracy, for different batch sizes, group sizes and learning rates.

II. Study of Literature

Batch Normalization:

Deep neural networks are trained using mini-batch gradient descent. In each iteration, a mini-batch of *m* samples is taken, and the gradient of the loss with respect to the parameters is computed using these *m* samples. When the parameters of one layer are thus updated in one iteration, the distribution of the inputs of the next layer would change in the next iteration. This change in the distribution of the inputs of each layer, is termed as **internal covariate shift**, and this makes the optimization difficult by allowing the inputs to drift towards extreme values where the gradients may vanish or explode.

With batch normalization, the mean and variance of these inputs are set to 0 and 1 respectively at each iteration, followed by a linear scaling and centering transform. The mathematical description of BN is shown below.



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

It was originally proposed that BN reduces internal covariate shift because the distributions of the inputs of the next layer are smoothened. Thus it allows using larger training rates, and hence makes the training faster. The batch size *m* plays an important role in deciding the performance of BN.

<u>Group Normalization:</u>
Group normalization (GN) works is a similar method of normalization. It is used in the context of convolutional networks. The difference from BN is that the mean and variance are computed for a group of channels across the dimensions of the image. GN is closely related to layer normalization and instance normalization, which are special cases of GN.

A very good visualization of all these normalization methods is shown below:

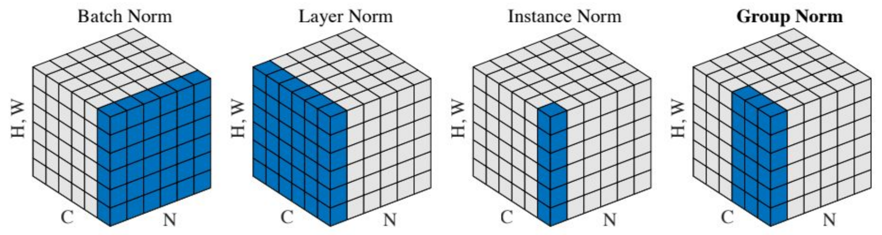


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

The intuition behind GN is that different channels of a convolutional layer are correlated with each other, and hence normalization across a group of channels can stabilize their distributions.

GN has a significant advantage over BN, particularly when the batch size is small. When the memory is constrained, we may be required to choose smaller batch size, then the estimated mean and variance from the mini-batch of samples may not be consistent with the true distribution. Indeed, it has been shown experimentally that GN works better than BN for small batch sizes, while the performance of GN does not depend on the batch size.

How does Batch Normalization help Optimization?

The original authors of BN postulated that BN makes training faster by reducing the internal covariate shift. However, it has been studied through experiments that BN does not always reduce internal covariate shift, but still improves the achieved accuracy. This was shown by demonstrating that there is no definite link between reduced internal covariate shift and better accuracy.

On the other hand, BN does help optimization in the following other ways :

1. Batch normalization makes the loss landscape smoother by decreasing the magnitude of the resulting gradients. Therefore, in any given direction, the loss does not change by a very large amount.

$$\left|\left|\nabla_{\boldsymbol{y}_j}\widehat{\mathcal{L}}\right|\right|^2 \leq \frac{\gamma^2}{\sigma_j^2}\left(\left|\left|\nabla_{\boldsymbol{y}_j}\mathcal{L}\right|\right|^2 - \frac{1}{m}\left\langle \mathbf{1}, \nabla_{\boldsymbol{y}_j}\mathcal{L}\right\rangle^2 - \frac{1}{\sqrt{m}}\left\langle \nabla_{\boldsymbol{y}_j}\mathcal{L}, \hat{\boldsymbol{y}}_j\right\rangle^2\right)$$

2. BN makes the gradients more Lipschitz, so that the gradients are more smooth. This means that the gradient at the next iteration is predictable from the gradient in the current iteration.

$$\left(\nabla_{\boldsymbol{y}_j}\widehat{\mathcal{L}}\right)^\top \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j \partial \boldsymbol{y}_j}\left(\nabla_{\boldsymbol{y}_j}\widehat{\mathcal{L}}\right) \leq \frac{\gamma^2}{\sigma^2}\left(\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j}\right)^\top \boldsymbol{H}_{jj}\left(\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j}\right) - \frac{\gamma}{m\sigma^2}\left\langle \hat{\boldsymbol{g}}_j, \hat{\boldsymbol{y}}_j\right\rangle \left|\left|\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j}\right|\right|^2$$

3. BN prevents exploding and vanishing gradients by keeping the activations away from the saturation regions of the non-linearity.
4. BN allows us to increase the learning rate because the losses change gradually, and the activations do not reach the saturation regimes.
5. BN makes the model more robust towards hyperparameters such as the initialization of weights.
6. BN also regularizes the network, and reduces the need for other methods such as dropout and weight penalty.

III. Experiments and Results

Experiment Setup

We conduct experiments on the CIFAR10 (image classification) dataset. Each image is of size 32x32 and can belong to one of 10 classes. We have implemented a small convolutional neural network of the following architecture:

1. Convolutional layer with 16 filters of size 3x3
2. 2x2 Max-pooling layer
3. Convolutional layer with 32 filters of size 3x3
4. 2x2 Max-pooling layer
5. Fully connected layer with 256 hidden units
6. Fully connected layer with softmax to generate 10 probabilities

Each convolutional layer has batch/group normalization at its output before the ReLU activation. The loss used is cross-entropy loss. A small network is used primarily because of computational constraints. Further this allows the gradients and distributions to be easily visualised.
We conduct experiments only on CIFAR10 because it was observed that MNIST trains to high accuracy very quickly irrespective of changes in the model, so CIFAR10 is better to observe the differences between the methods.

The code was written using Tensorflow for Python, and was executed using the GPU accelerator from Google Colaboratory. While the standard library function was used for batch normalization, the function for group normalization was written independently, because the library function does not support varying batch sizes. The code spans about 300 lines, and can be found here:
https://github.com/ssrivatsan97/batch_group_norm

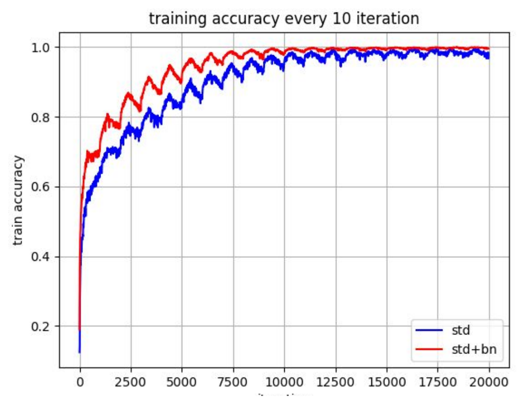
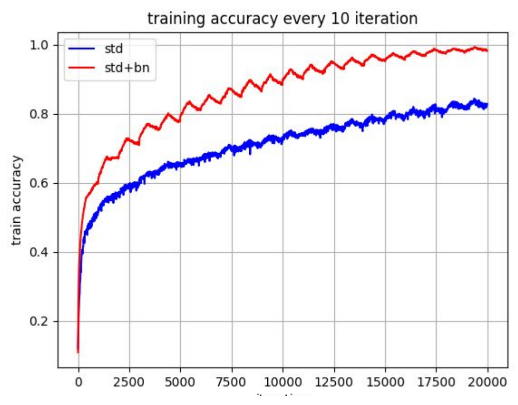
Batch Normalization Experiments

(Abbreviations - **BS** : batch size, **ES** : epoch size, **LR** : learning rate, **GS** : number of groups)

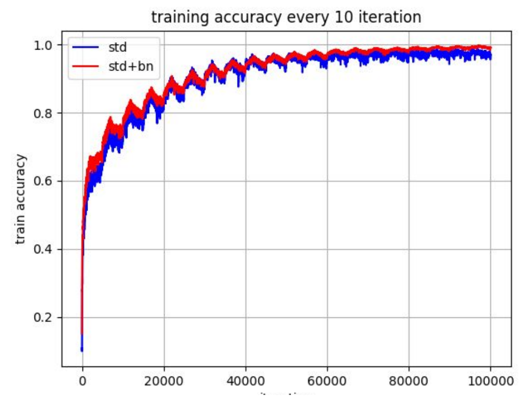
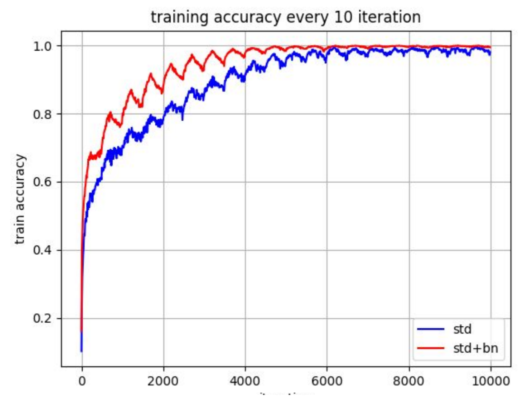
Few Empirical Observations :

1. In these set of experiments, as expected, one can observe from the *training_accuracy* as well as *training_loss* plots that the training is faster for the case when the convolutional layers are batch normalized compared to the case when these are not.
2. Decreasing the learning rate from 1e-3 to 1e-4 slows down the convergence. We empirically found that approx. LR = 1e-3 turns out to be a good learning rate for most cases. With batch normalization, we were able to increase the learning rate and achieve similar performance results with reduced iterations.
3. Using batch norm with lower batch size (< 20) fails to provide any advantage in terms of faster convergence as it might fail to capture the statistical properties of the data.

Training Accuracy Plot

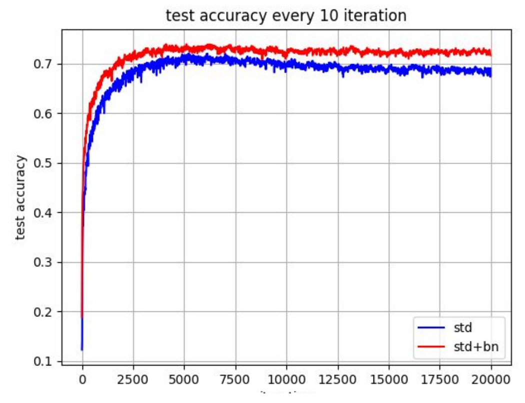
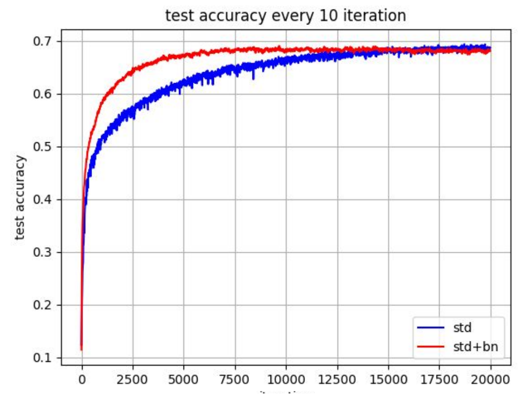
	

BS:50, ES:20, LR:1e-3	BS:50, ES:20, LR:1e-4

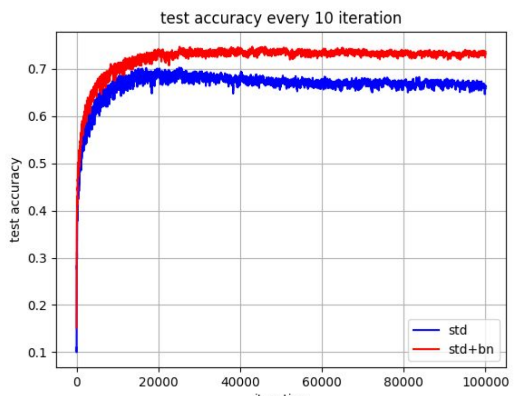
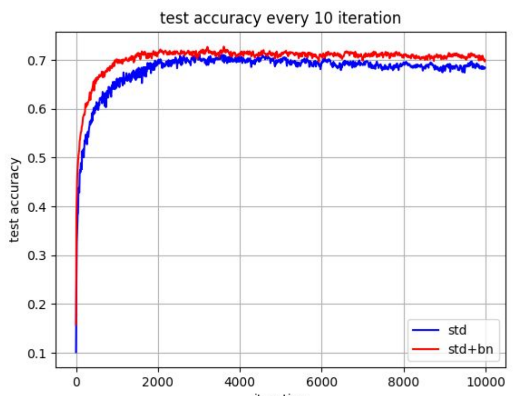
	

BS:10, ES:20, LR:1e-3	BS:100, ES:20, LR:1e-3

Testing Accuracy Plot

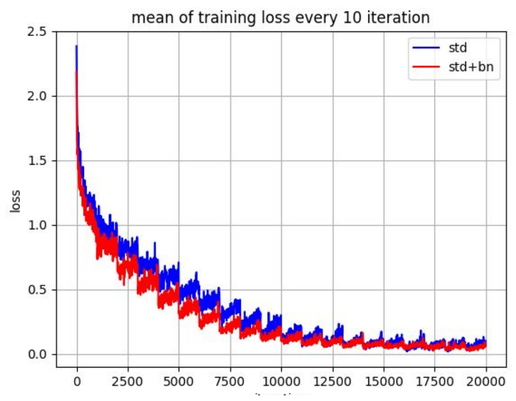
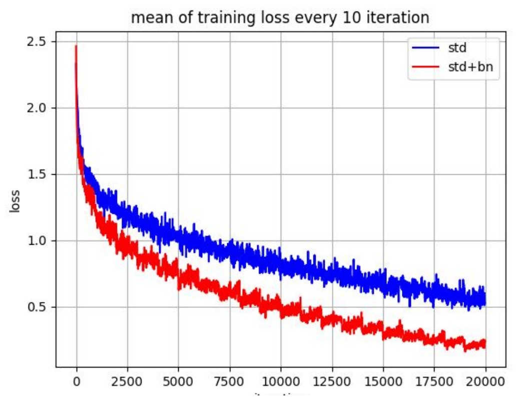
	

BS:50, ES:20, LR:1e-3	BS:50, ES:20, LR:1e-4

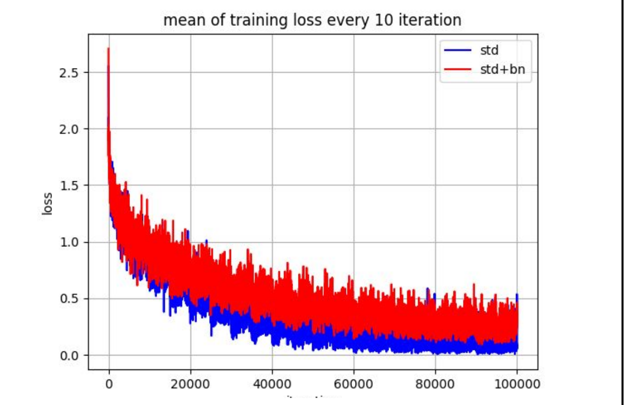
	

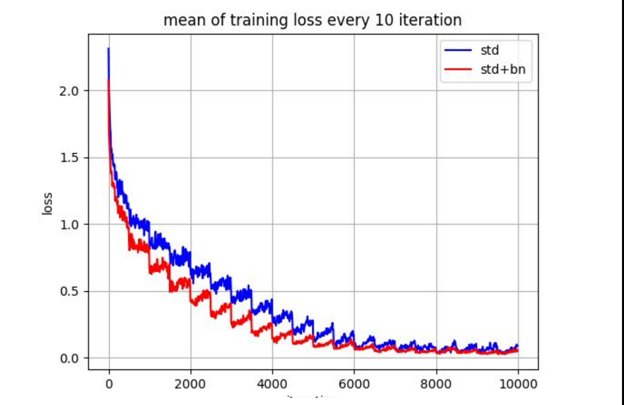
BS:10, ES:20, LR:1e-3	BS:100, ES:20, LR:1e-3

Training Loss Plot

BS:50, ES:20, LR:1e-3	BS:50, ES:20, LR:1e-4





BS:10, ES:20, LR:1e-3	BS:100, ES:20, LR:1e-3

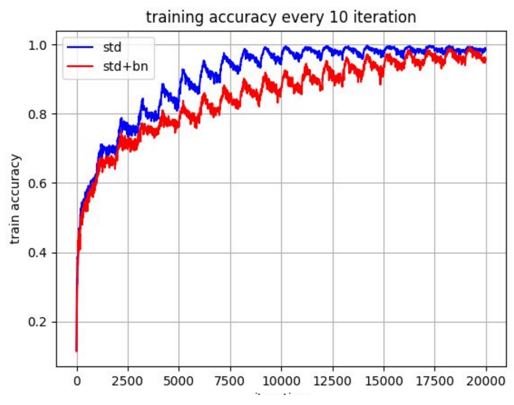
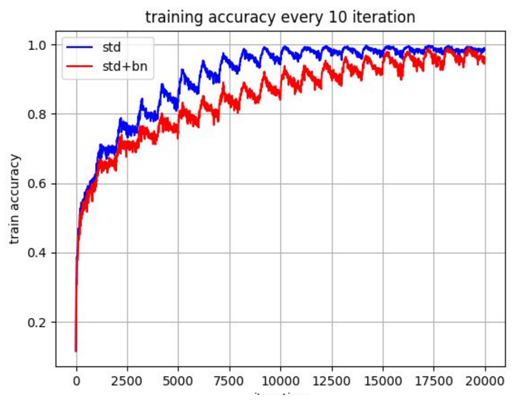
Group Normalization Experiments

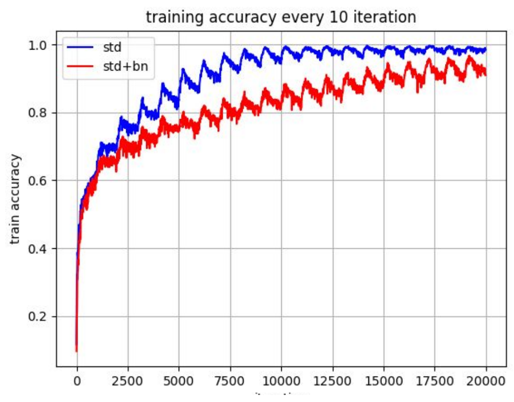
(Abbreviations - **BS** : batch size, **ES** : epoch size, **LR** : learning rate, **GS** : number of groups)

Few Empirical Observations :

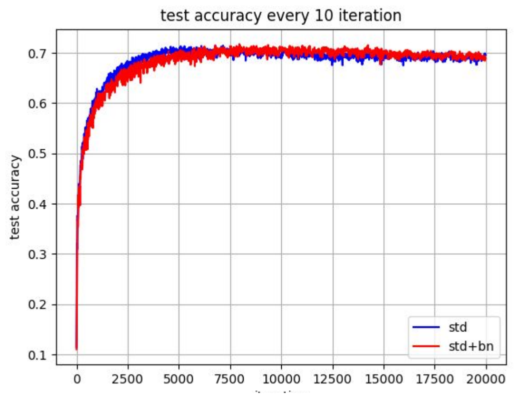
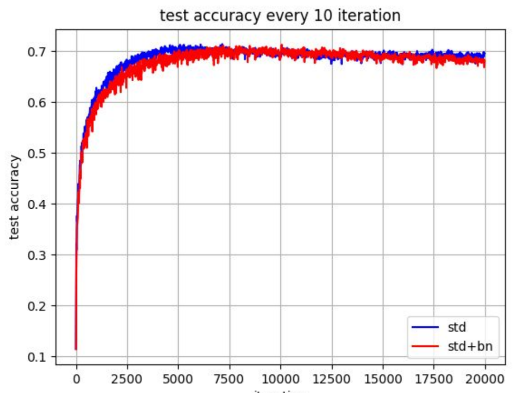
1. In this test, the training accuracy is less when GN is used as against no normalization, or using BN
2. There is hardly any change in training accuracy when the number of groups is varied from 1 to 16.
3. The test accuracy is almost the same whether GN is used or not.
4. The poor performance of GN in this case, may be due to the small group sizes.

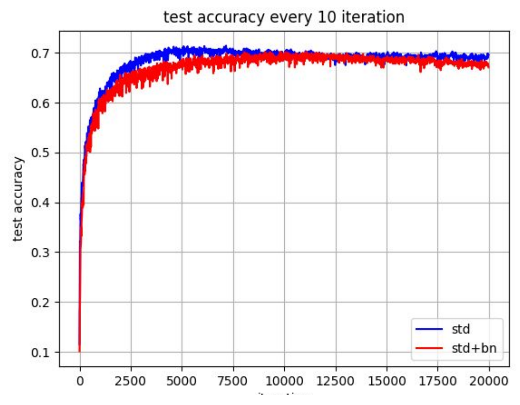
Training Accuracy Plot

	
BS:50, ES:20, GS:1, LR:1e-3	BS:50, ES:20, GS:4, LR:1e-3

	
BS:50, ES:20, GS:16, LR:1e-3	

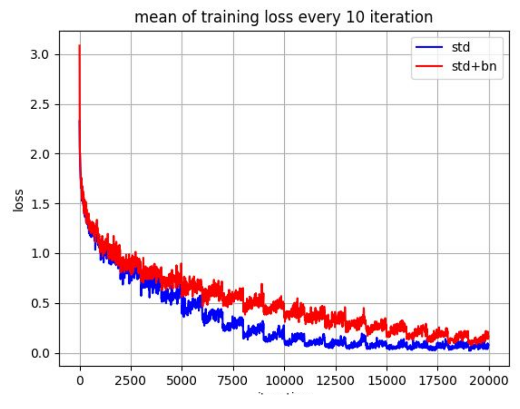
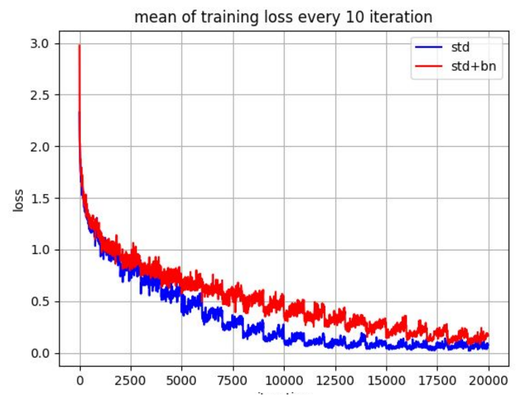
Testing Accuracy Plot

	
BS:50, ES:20, GS:1 LR:1e-3	BS:50, ES:20, GS:4, LR:1e-3

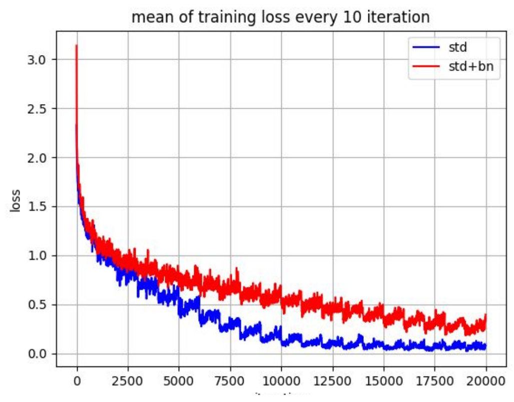
	

BS:50, ES:20, GS:16, LR:1e-3	

Training Loss Plot

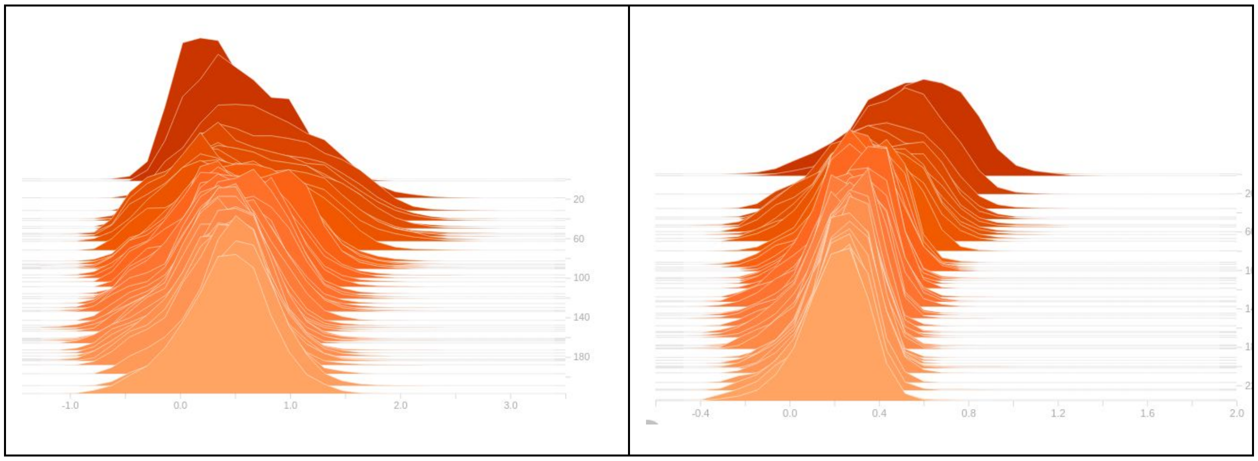
BS:50, ES:20, GS:1, LR:1e-3	BS:50, ES:20, GS:4, LR:1e-3

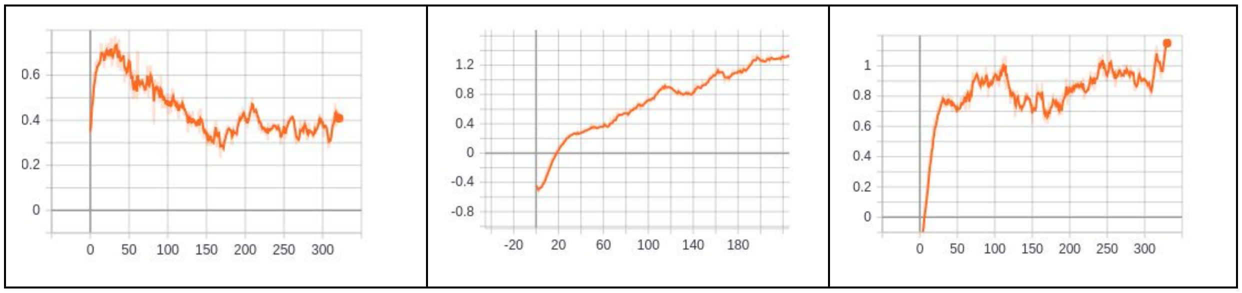
BS:50, ES:20, GS:16, LR:1e-3	

Visualizations

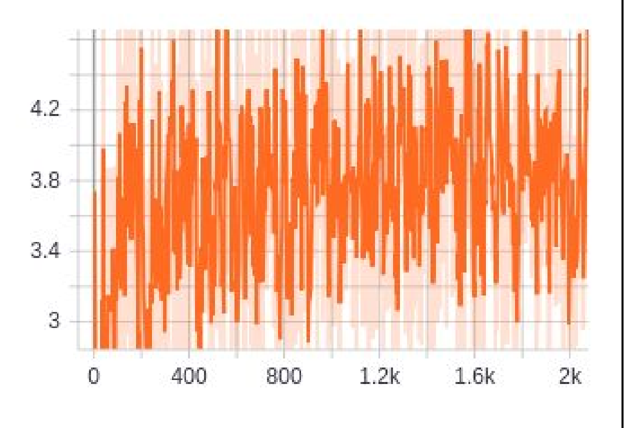
All the visualizations shown below were done using TensorBoard. The histogram of output values at a selected channel were observed, as they evolve with time. One example is shown below. The left and right images shows the outputs before and after group normalization. The z-axis shows time progressing from the back to the front. Normalization makes the histogram regular and makes the variance close to 1. This might be especially useful at the starting timesteps where the “before” gradient histograms are highly skewed.

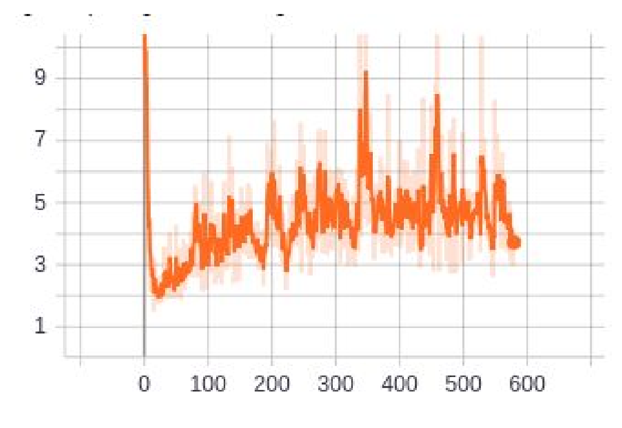


Shown below are the plots of the means of three of the channel outputs of the second layer (before normalization), as they evolve with time. It is our belief that when several channels have similar means and variances, group normalization using them as a group would be advantageous. In this case, it appears that group normalization using these groups would not be very helpful since their distributions are different.



Below are the plots of the L-2 norm of the gradient at each time step. Left is without normalization and right is with GN. In this case, the gradients are more constrained without the normalization (note the difference in the axis scaling in both plots).





These visualizations are helpful to get an intuition about how these normalizations work. In the latter two visualizations it was seen that with GN, the circumstances for training were worse. This could be a reason for lower accuracy with GN in this model.

IV. Effort

Approximate fraction of time spent on various parts of the project:

1. Theoretical analysis - 30%
2. Basic implementation - 10%
3. Experiments - 60%

The limitations in the execution of this project were that we were unable to examine the theoretical bounds that exist for BN, in the case of GN. Further, due to computational constraints, we were only able to test using a small model. These were certain challenges that we faced in this project.

Contributions to the project:

1. Srivatsan - theory, experiments
2. Saqib - implementation, experiments
3. Chinmay - theory, implementation

V. Conclusion and Future Work

This project has done a basic comparative study of batch normalization and group normalization. Both methods were understood in great depth while doing the project. Existing theoretical bounds were studied, and intuitions were developed about both the methods. Due to computational constraints, we were only able to test using a very small network. It appeared in this test that group normalization seemed to lower accuracy. However, it may be conjectured that this is because the group sizes were very small here (typically there are 32 groups in large convolutional networks). Thus, GN faced the same problem here that BN faces at low batch sizes.

As mentioned earlier, the reasons why normalization helps to train neural networks is not very clear, and there are many hypotheses. While batch normalization has been studied theoretically, this has not been done for other techniques, and it is largely unclear about which method is beat and when. A thorough analysis of GN for different group sizes was not possible in this project, and must be done by using larger models. Further, it may be interesting to consider if we can achieve a good normalization by dynamically choosing the groups based on correlation between channels, rather than sequentially.

VI. References

1. S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv 1502.03167, 2018.* https://arxiv.org/pdf/1502.03167.pdf
2. S. Santurkar, D. Tsipras, A. Ilyas, A. Madry. How Does Batch Normalization Help Optimization? *NeurIPS 2018.* https://arxiv.org/pdf/1805.11604.pdf
3. Y. Wu, K. He. Group Normalization. *arXiv 1803.08494, 2018.* https://arxiv.org/pdf/1803.08494.pdf