SEQUENTIAL NORMALIZATION: AN IMPROVEMENT OVER GHOST NORMALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Batch normalization (BatchNorm) is an effective yet poorly understood technique for neural network optimization. It is often assumed that the degradation in BatchNorm performance to smaller batch sizes stems from it having to estimate layer statistics using smaller sample sizes. However, recently, Ghost normalization (GhostNorm), a variant of BatchNorm that explicitly uses smaller sample sizes for normalization, has been shown to improve upon BatchNorm in some datasets. Our contributions are: (i) three types of GhostNorm implementations are described, two of which employ BatchNorm as the underlying normalization technique, (ii) we uncover a source of regularization that is unique to GhostNorm, and not simply an extension from BatchNorm, and visualise the difference in their loss landscapes, (iii) we extend GhostNorm and introduce a new type of normalization layer called Sequential Normalization (SeqNorm), (iv) we compare both GhostNorm and SeqNorm against BatchNorm alone as well as with other regularisation techniques, (v) for both GhostNorm and SeqNorm, we report superior performance over state-of-the-art methodologies on CIFAR–10, CIFAR–100, and ImageNet data sets.

1 INTRODUCTION

The effectiveness of Batch Normalization (BatchNorm), a technique first introduced by Ioffe & Szegedy (2015) on neural network optimization has been demonstrated over the years on a variety of tasks, including computer vision (Krizhevsky et al., 2017; Huang et al., 2016; He et al., 2015), speech recognition (Graves et al., 2013), and other (Sutskever et al., 2014; Silver et al., 2016; Mnih et al., 2015). BatchNorm is typically embedded at each neural network (NN) layer either before or after the activation function, normalizing and projecting the input features to match a Gaussian-like distribution. Consequently, the activation values of each layer maintain more stable distributions during NN training which in turn is thought to enable faster convergence and better generalization performances (Ioffe & Szegedy, 2015; Santurkar et al., 2018; Bjorck et al., 2018).

Despite the wide adoption and practical success of BatchNorm, its underlying mechanics within the context of NN optimization has yet to be fully understood. Initially, Ioeffe and Szegedy suggested that it came from it reducing the so-called internal covariate shift (Ioffe & Szegedy, 2015). At a high level, internal covariate shift refers to the change in the distribution of the inputs of each NN layer that is caused by updates to the previous layers. This continual change throughout training was conjectured to negatively affect optimization (Ioffe & Szegedy, 2015; Santurkar et al., 2018). However, recent research disputes that with compelling evidence that demonstrates how BatchNorm may in fact be increasing internal covariate shift (Santurkar et al., 2018). Instead, the effectiveness of BatchNorm is argued to be a consequence of a smoother loss landscape (Santurkar et al., 2018).

Following the effectiveness of BatchNorm on NN optimization, a number of different normalization techniques have been introduced (Ioffe, 2017; Lei Ba et al., 2016; Qiao et al., 2019; Wu & He, 2018; Ulyanov et al., 2016). Their main inspiration was to provide different ways of normalizing the activations without being inherently affected by the batch size. In particular, it is often observed that BatchNorm performs worse with smaller batch sizes (Ioffe, 2017; Wu & He, 2018; Yan et al., 2020). This degradation has been widely associated to BatchNorm computing poorer estimates of mean and variance due to having a smaller sample size. However, recent demonstration of the effectiveness of GhostNorm comes in antithesis with the above belief (Summers & Dinneen, 2020). GhostNorm explicitly divides the mini–batch into smaller batches and normalizes over them independently (Hoffer

et al., 2017). Nevertheless, when compared to other normalization techniques (Ioffe, 2017; Lei Ba et al., 2016; Qiao et al., 2019; Wu & He, 2018; Ulyanov et al., 2016), the adoption of GhostNorm has been rather scarce, and narrow to large batch size training regimes (Summers & Dinneen, 2020).

The contributions of this paper are as follows: (i) Outlining three different ways of using GhostNorm as a normalization layer, (ii) Identifying a source of regularization in GhostNorm that is fundamentally different from BatchNorm, (iii) Visualizing the loss landscape of GhostNorm under vastly different experimental setups, and observing that GhostNorm consistently decreases the smoothness of the loss landscape, especially on the later epochs of training, (iv) Introducing a new normalization layer called SeqNorm, (v) Comparing GhostNorm and SeqNorm with other regularization methods, to first investigate how both stand out against established regularization techniques but to also observe whether there are any synergies, (vi) Showcasing consistently better generalization performances on CIFAR–10, CIFAR–100, and ImageNet when BatchNorm is replaced with either GhostNorm or SeqNorm, with the latter even surpassing the current SOTA on CIFAR–100 and ImageNet.

1.1 Related Work

Ghost Normalization is a technique originally introduced by Hoffer et al. (2017). Over the years, the primary use of GhostNorm has been to optimize NNs with large batch sizes and multiple GPUs (Summers & Dinneen, 2020). However, when compared to other normalization techniques (Ioffe, 2017; Lei Ba et al., 2016; Qiao et al., 2019; Wu & He, 2018; Ulyanov et al., 2016), the adoption of GhostNorm has been rather scarce.

Closest in spirit to the present work is the recent research by Summers & Dinneen (2020) who have experimented with GhostNorm on both small and medium batch size training regimes.Summers & Dinneen (2020) tuned the number of groups within GhostNorm (see section 2.1) on CIFAR–100, Caltech–256, and SVHN, and reported positive results on the first two data sets. More results are reported on other data sets through transfer learning. However, the use of other new optimization methods confounds the attribution of the observed improvement.

The closest line of work to SeqNorm is, again, found in the work of Summers & Dinneen (2020). Therein they employ a normalization technique which although at first glance may appear similar to SeqNorm, at a fundamental level, is rather different. This stems from the vastly different goals between our works, i.e. Summers & Dinneen (2020) try to increase the available information when small batch sizes are used (Summers & Dinneen, 2020), whereas we strive to improve BatchNorm in the more general setting. At a high level, where SeqNorm performs GroupNorm and GhostNorm sequentially, their normalization method applies both simultaneously. At a fundamental level, the latter embeds the stochastic nature of GhostNorm (see section 2.2) into that of GroupNorm, thereby potentially disrupting the learning of channel grouping within NNs. Switchable normalization is also of some relevance to SeqNorm as it enables the NN to learn which normalization techniques to employ at different layers (Luo et al., 2019). However, similar to the previous work, simultaneously applying different normalization techniques has a fundamentally different effect than SeqNorm.

Related to our work is also research geared towards exploring the effects of BatchNorm on optimization (Bjorck et al., 2018; Santurkar et al., 2018; Kohler et al., 2019). Finally, of some relevance is also the large body of work that exists on improving BatchNorm at the small batch training regime (Ioffe, 2017; Wu & He, 2018; Luo et al., 2020; Yan et al., 2020).

2 Methodology

2.1 Formulation

Given a fully-connected or convolutional neural network, the parameters of a typical layer l with normalization, $Norm$, are the weights $\boldsymbol{W}^l$ as well as the scale and shift parameters $\boldsymbol{\gamma^l}$ and $\boldsymbol{\beta^l}$. For brevity, we omit the l superscript. Given an input tensor $\mathbf{X}$, the activation values $\boldsymbol{A}$ of layer l are computed as:

$$\boldsymbol{A} = g(Norm(\mathbf{X} \odot \boldsymbol{W}) \otimes \boldsymbol{\gamma} + \boldsymbol{\beta}) \tag{1}$$

where $g(\cdot)$ is the activation function, $\odot$ corresponds to either matrix multiplication or convolution for fully-connected and convolutional layers respectively, and $\otimes$ describes an element-wise multiplication.

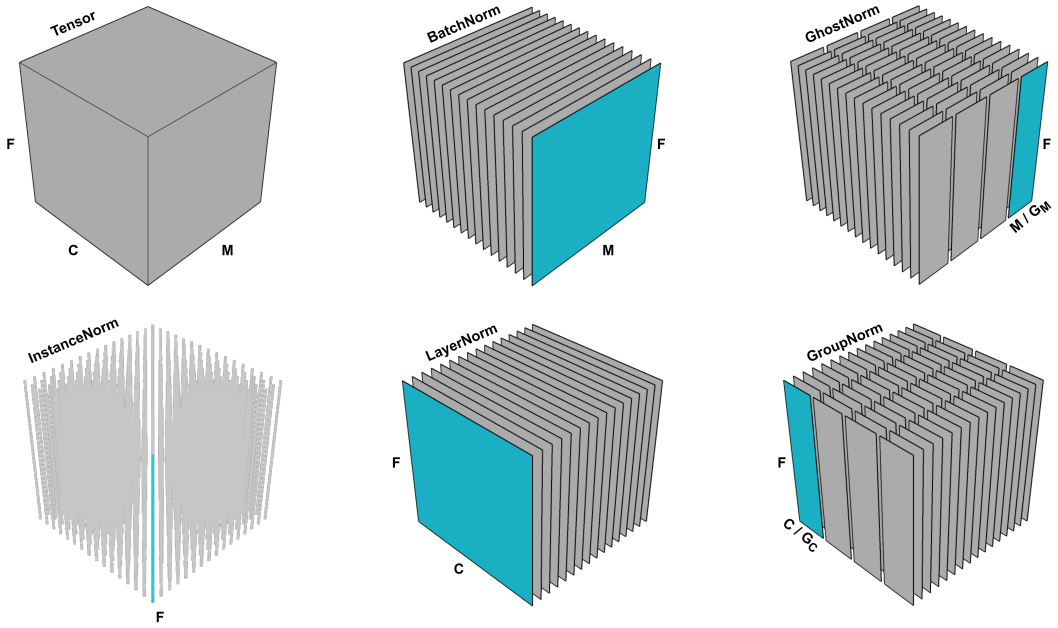

Figure 1: The input tensor is divided into a number of line (1D) or plane (2D) slices. Each normalization technique slices the input tensor differently and each slice is normalized independently of the other slices. SeqNorm sequentially slices and normalizes the tensor as GroupNorm and then as GhostNorm.

Most normalization techniques differ in how they transform the product $\mathbf{X} \odot \boldsymbol{W}$. Let the product be a tensor with (M, C, F) dimensions where M is the so-called mini–batch size, or just batch size, C is the channels dimension, and F is the spatial dimension.

In *BatchNorm*, the given tensor is normalized across the channels dimension. In particular, the mean and variance are computed across C number of slices of (M, F) dimensions (see Figure 1) which are subsequently used to normalize each channel $c \in C$ independently. In *LayerNorm*, statistics are computed over M slices, each having the dimension (C, F), normalizing the values of each data sample $m \in M$ independently. *InstanceNorm* normalizes the values of the tensor over both M and C, i.e. computes statistics across $M \times C$ slices of F dimension.

GroupNorm can be thought as an extension to LayerNorm wherein the C dimension is divided into G_C number of groups, i.e. $(M, G_C, {}^{C}/_{G_C}, F)$. Statistics are calculated over $M \times G_C$ slices of $({}^{C}/_{G_C}, F)$ dimensions. Similarly, *GhostNorm* can be thought as an extension to BatchNorm, wherein the M dimension is divided into G_M groups, normalizing over $C \times G_M$ slices of $({}^{M}/_{G_M}, F)$ dimensions. Both G_C and G_M are hyperparameters that can be tuned based on a validation set. All of the aforementioned normalization techniques are illustrated in Figure 1.

SeqNorm employs both GroupNorm and GhostNorm in a sequential manner. Initially, the input tensor is divided into $(M, G_C, {}^{C}/_{G_C}, F)$ dimensions, normalizing across $M \times G_C$ number of slices, i.e. same as GroupNorm. Then, once the G_C and ${}^{C}/_{G_C}$ dimensions are collapsed back together, the input tensor is divided into $(G_M, {}^{M}/_{G_M}, C, F)$ dimensions for normalizing over $C \times G_M$ slices of $({}^{M}/_{G_M}, F)$ dimensions, i.e. same as GhostNorm.

Any of the slices described above is treated as a set of values $\mathbb{S}$ with one dimension. The mean and variance of $\mathbb{S}$ are computed in the traditional way (see Equation 2). The values of $\mathbb{S}$ are then normalized as shown below.

$$\mu = \frac{1}{M} \sum_{x \in \mathbb{S}} x \text{ and } \sigma^2 = \frac{1}{M} \sum_{x \in \mathbb{S}} (x - \mu)^2 \tag{2}$$

$$\forall x \in \mathbb{S}, x = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{3}$$

Once all slices are normalized, the output of the $Norm$ layer is simply the concatenation of all slices back into the initial tensor shape.

2.2 THE EFFECTS OF GHOST NORMALIZATION

There is only one other published work which has investigated the effectiveness of Ghost Normalization for small and medium mini-batch sizes (Summers & Dinneen, 2020). Therein, the authors hypothesise that GhostNorm offers stronger regularization than BatchNorm as it computes the normalization statistics on smaller sample sizes (Summers & Dinneen, 2020). In this section, we support that hypothesis by providing insights into a particular source of regularization, unique to GhostNorm, that stems from normalizing groups of activations during a forward pass.

Consider as an example the tuple X with $(35, 39, 30, 4, 38, 26, 27, 19)$ values which can be thought as an input tensor with $(8, 1, 1)$ dimensions. Given to a BatchNorm layer, the output is the normalized version $\bar{X}$ with values $(0.7, 1.1, 0.3, -2.2, 1.0, -0.1, -0.02, -0.8)$. Note how although the values have changed, the ranking order of the activation values has remained the same, e.g. the 2nd value is larger than the 5th value in both X ($39 > 38$) and $\bar{X}$ ($1.1 > 1.0$). More formally, the following holds true:

$$\begin{aligned} \text{Given n-tuples } X = (x_0, x_1, ..x_n) \text{ and } \bar{X} = (\bar{x}_0, \bar{x}_1, ..., \bar{x}_n), \\ \forall i, j \in I, \bar{x}_i > \bar{x}_j \iff x_i > x_j \\ \bar{x}_i < \bar{x}_j \iff x_i < x_j \\ \bar{x}_i = \bar{x}_j \iff x_i = x_j \end{aligned} \tag{4}$$

On the other hand, given X to a GhostNorm layer with $G_M = 2$, the output $\bar{X}$ is $(0.6, 0.9, 0.2, -1.7, 1.5, -0.2, -0.07, -1.2)$. Now, we observe that after normalization, the 2nd value has become much smaller than the 5th value in $\bar{X}$ ($0.9 < 1.5$). Where BatchNorm preserves the ranking order, GhostNorm can modify the importance of each sample, and hence alter the course of optimization. Our experimental results demonstrate how GhostNorm improves upon BatchNorm, supporting the hypothesis that the above type of regularization can be beneficial to optimization. Note that for BatchNorm the condition in Equation 4 only holds true across the $M \times F$ dimension of the input tensor whereas for GhostNorm it cannot be guaranteed for any dimension.

GhostNorm to BatchNorm One can argue that the same type of regularization can be found in BatchNorm over different mini–batches, e.g. given $[35, 39, 30, 4]$ and $[38, 26, 27, 19]$ as two different mini–batches. However, GhostNorm introduces the above during each forward pass rather than between forward passes. Hence, it is a regularization that is embedded during learning (GhostNorm), rather than across learning (BatchNorm).

GhostNorm to GroupNorm Despite the visual symmetry between GhostNorm and GroupNorm, there is one major difference. Grouping has been employed extensively in classical feature engineering, such as SIFT, HOG, and GIST, wherein independent normalization is often performed over these groups (Wu & He, 2018). At a high level, GroupNorm can be thought as motivating the network to group similar features together (Wu & He, 2018). However, for GhostNorm, this would not be possible due to random sampling, and random arrangement of the data within each mini–batch. Therefore, we hypothesise that the effects of these two normalization techniques could be combined for their benefits to be accumulated. Specifically, we propose SeqNorm, a normalization technique that employs both GroupNorm and GhostNorm in a sequential manner.

2.3 IMPLEMENTATION[1]

Ghost Normalization The direct approach of implementing GhostNorm is shown in Appendix Figure 4. Although the exponential moving averages are omitted for brevity, it is worth mentioning

[1]The implementations of GhostNorm and SeqNorm layers will be provided in a public repository.

that they were accumulated in the same way as BatchNorm. In addition to the above direct implementation, GhostNorm can be effectively employed while using BatchNorm as the underlying normalization technique.

When the desired batch size exceeds the memory capacity of the available GPUs, practitioners often resort to the use of accumulating gradients. That is, instead of having a single forward pass with M examples through the network, n_{fp} number of forward passes are made with ${}^{M}/_{n_{fp}}$ examples each. Most of the time, gradients computed using a smaller number of training examples, i.e. ${}^{M}/_{n_{fp}}$, and accumulated over a number of forward passes n_{fp} are identical to those computed using a single forward pass of M training examples. However, it turns out that when BatchNorm is employed in the neural network, the gradients can be substantially different for the above two cases. This is a consequence of the mean and variance calculation (see equation 2) since each forwarded smaller batch of ${}^{M}/_{n_{fp}}$ data will have a different mean and variance than if all M examples were present. Accumulating gradients with BatchNorm can thus be thought as an alternative way of using GhostNorm with the number of forward passes n_{fp} corresponding to the number of groups G_M. A PyTorch implementation of accumulating gradients is shown in Appendix Figure 5.

Finally, the most popular implementation of GhostNorm via BatchNorm, albeit typically unintentional, comes as a consequence of using multiple GPUs. Given n_g GPUs and M training examples, M/n_g examples are forwarded to each GPU. If the BatchNorm statistics are not synchronized across the GPUs, often the case for image classification, then n_g corresponds to the number of groups G_M.

A practitioner who would like to use GhostNorm should employ the implementation shown in Figure 4. Nevertheless, under the discussed circumstances, one could explore GhostNorm through the use of the other implementations.

Sequential Normalization The implementation of SeqNorm is straightforward since it applies GroupNorm, a widely implemented normalization technique, and GhostNorm, for which we have discussed three possible implementations, in a sequential manner.

3 EXPERIMENTS

In this section, we first strive to take a closer look into GhostNorm by visualizing the smoothness of the loss landscape during training; a component which has been described as the primary reason behind the effectiveness of BatchNorm. Then, we conduct a number of ablation experiments comparing both GhostNorm and SeqNorm against with other regularization approaches. Finally, we evaluate the effectiveness of both GhostNorm and SeqNorm on the standard image classification data sets of CIFAR–10, CIFAR–100, and ImageNet. Note that in all of our experiments, the smallest ${}^{M}/_{G_M}$ we employ for both SeqNorm and GhostNorm is 4. A ratio of 1 would be undefined for normalization, whereas a ratio of 2 results in large information corruption, i.e. all activations are reduced to either 1 or -1 values.

3.1 LOSS LANDSCAPE VISUALISATION

Loss landscape We visualize the loss landscape during optimization on MNIST and CIFAR–10 using an approach that was described by Santurkar et al. (Santurkar et al., 2018). Each time the network parameters are to be updated, we walk towards the gradient direction and compute the loss at multiple points. This enable us to visualise the smoothness of the loss landscape by observing how predictive the computed gradients are. In particular, at each step of updating the network parameters, we compute the loss at a range of learning rates, and store both the minimum and maximum loss. Implementation details are provided in the Appendix.

For both data sets and networks, we observe that the smoothness of the loss landscape deteriorates when GhostNorm is employed. In fact for MNIST, as seen in Figure 2, the loss landscape of GhostNorm bears closer resemblance to our baseline which did not use any normalization technique. For CIFAR–10, this is only observable towards the last epochs of training. In spite of the above observation, we have consistently witness better generalization performances with GhostNorm in almost all of our experiments, even at the extremes wherein G_M is set to 128, i.e. only 4 samples per group.

Our experimental results challenge the often established correlation between a smoother loss landscape and a better generalization performance (Santurkar et al., 2018; Qiao et al., 2019). Although

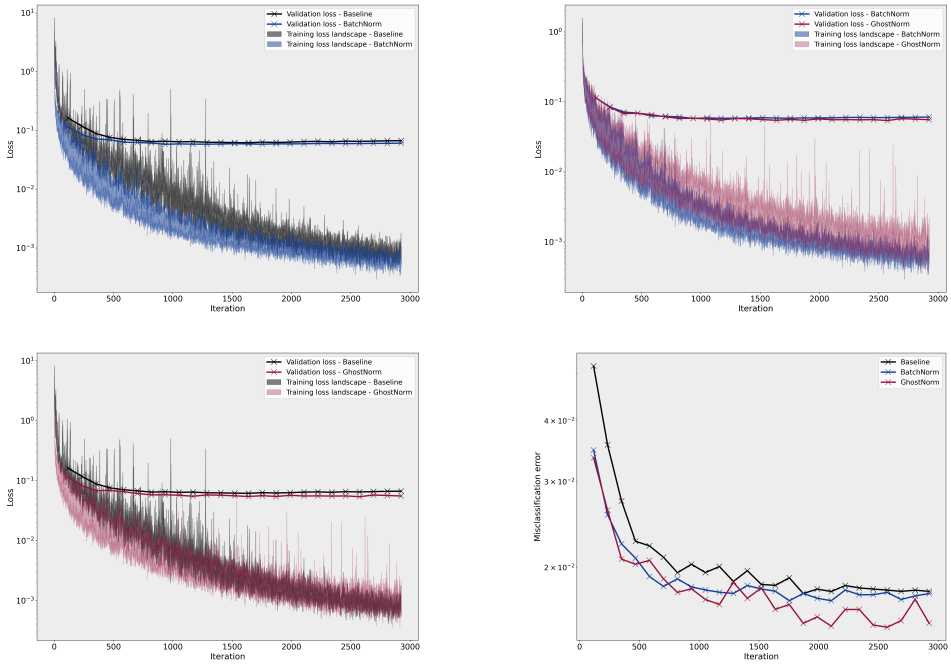

Figure 2: Comparison of the loss landscape on MNIST between BatchNorm, GhostNorm, and the baseline.

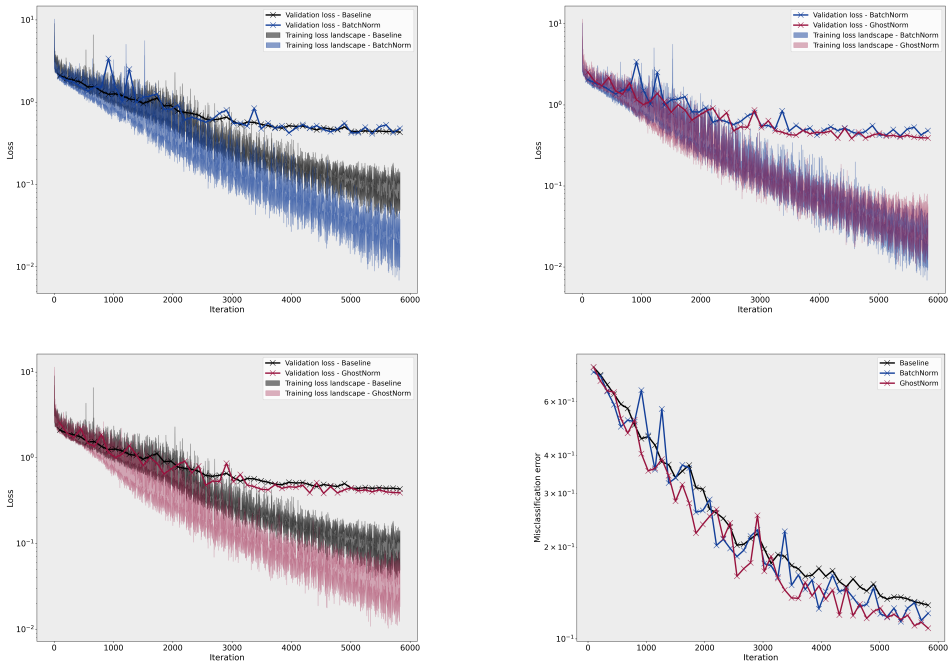

Figure 3: Comparison of the loss landscape on CIFAR–10 between BatchNorm, GhostNorm, and the baseline.

Table 1: Results on CIFAR–100. For SeqNorm, we only show the best results for each G_C. Both validation and testing performances are averaged over two different runs.

	Validation accuracy	Testing accuracy
BatchNorm	$80.6 \pm 0.2\%$	$82.1 \pm 0.2\%$
Noisy BatchNorm	$80.8 \pm 0.1\%$	$82.0 \pm 0.3\%$
GhostNorm ($G_M = 2$)	$80.9 \pm 0.1\%$	-
GhostNorm ($G_M = 4$)	$81.2 \pm 0.1\%$	-
GhostNorm ($G_M = 8$)	$81.4 \pm 0.5\%$	$82.8 \pm 0.6\%$
GhostNorm ($G_M = 16$)	$80.3 \pm 0.5\%$	-
SeqNorm ($G_C = 1, G_M = 4$)	$82.3 \pm 0.1\%$	-
SeqNorm ($G_C = 2, G_M = 4$)	$82.4 \pm 0.2\%$	-
SeqNorm ($G_C = 4, G_M = 8$)	$\mathbf{82.5 \pm 0.1\%}$	$\mathbf{83.8 \pm 0.04\%}$
SeqNorm ($G_C = 8, G_M = 8$)	$82.4 \pm 0.2\%$	-
SeqNorm ($G_C = 16, G_M = 8$)	$82.3 \pm 0.3\%$	-
BatchNorm w/ RandAugment	$81.4 \pm 0.0\%$	$82.9 \pm 0.2\%$
GhostNorm w/ RandAugment	$82.3 \pm 0.1\%$	$83.5 \pm 0.1\%$
SeqNorm w/ RandAugment	$82.4 \pm 0.2\%$	$\mathbf{83.8 \pm 0.3\%}$

beyond the scope of our work, a theoretical analysis of the implications of GhostNorm when compared to BatchNorm could potentially uncover further insights into the optimization mechanisms of both normalization techniques.

3.2 IMAGE CLASSIFICATION

CIFAR–100 Initially, we turn to CIFAR–100, and tune the hyperparameters of both GhostNorm and SeqNorm in a grid-search fashion. The results are shown in Table 1. We also examine a noisy version of BatchNorm wherein we inject Gaussian noise on the activations just before normalization. Finally, for all normalization layers, we also train models that employ dropout as well as RandAugment (Cubuk et al., 2019). All of the aforementioned regularization techniques were tuned as described in Appendix.

Both GhostNorm and SeqNorm improve upon the BatchNorm baseline by a large margin ($+0.7\%$ and $+1.7\%$ respectively). On the other hand, noisy BatchNorm does not improve the generalization performance. Models with dropout are omitted since they fail to provide any improvement on the validation set over the baselines. RandAugment substantially improves the BatchNorm ($+0.9\%$) and GhostNorm models ($+0.7\%$), but fails to benefit models with SeqNorm. Despite the lack of synergy with RandAugment, it is important to note that SeqNorm still manages to surpass the current SOTA performance on CIFAR–100 by 0.5% (Cubuk et al., 2019). These results support our hypothesis that sequentially applying GhostNorm and GroupNorm layers can have an additive effect on improving NN optimization.

However, the grid–search approach to tuning G_C and G_M of SeqNorm can be rather time consuming (time complexity: $\Theta(G_C \times G_M)$). Hence, we attempt to identify a less demanding hyperparameter tuning approach. The most obvious, and the one we actually adopt for the next experiments, is to sequentially tune G_M and G_C. In particular, we find that first tuning G_M, then selecting the largest $g_M \in G_M$ for which the network performs well (amongst similarly performing models, select the one with the lowest variance), and finally tuning G_C with g_M to be an effective approach (time complexity: $\Theta(G_C + G_M)$). In other words, for tuning the hyperparameters of SeqNorm, one first tunes the hyperparameter of GhostNorm G_M, and then the hyperparameter of GroupNorm G_C while keeping G_M constant. Note that by following this approach on CIFAR-100, we still end up with the same best SeqNorm, i.e. $G_C = 4$ and $G_M = 8$.

CIFAR–10 As the first step, we tune G_M for GhostNorm. We observe that for $G_M \in (2, 4, 8)$, the network performs similarly on the validation set at $\approx 96.6\%$ accuracy. We choose $G_M = 4$ for GhostNorm since it exhibits slightly higher accuracy.

Based on the tuning strategy described in the previous section, for SeqNorm, we adopt $G_M = 8$ (lowest variance) and tune G_C for values between 1 and 16, inclusively. Although the network performs similarly at $\approx 96.8\%$ accuracy for $G_C \in (1, 8, 16)$, we choose $G_C = 16$ as it achieves

Table 2: Results on CIFAR–10 and ImageNet data sets. Both validation and testing performances of CIFAR–10 are averaged over two different runs. For ImageNet, each model is evaluated on the conventional validation set, as well as on three newly released test sets Recht et al. (2019).

	Validation accuracy	Testing accuracy
CIFAR–10		
BatchNorm	$96.6 \pm 0.1\%$	$97.1 \pm 0.05\%$
GhostNorm ($G_M = 4$)	$96.7 \pm 0.2\%$	$97.3 \pm 0.1\%$
SeqNorm ($G_C = 16$, $G_M = 8$)	$\mathbf{96.8 \pm 0.1\%}$	$\mathbf{97.4 \pm 0.2\%}$
ImageNet		
BatchNorm	$71.2 \pm 0.01\%$	$67.0 \pm 6.9\%$
GhostNorm ($G_M = 4$)	$71.6 \pm 0.1\%$	$67.4 \pm 6.7\%$
SeqNorm ($G_C = 4$, $G_M = 4$)	$\mathbf{72.3 \pm 0.2\%}$	$\mathbf{68.1 \pm 6.8\%}$

slightly higher accuracy than the rest. Using the above configuration, SeqNorm is able to match the current SOTA on the testing set (Cubuk et al., 2019), yet as with CIFAR–100 without the employment RandAugment.

ImageNet We first train GhostNorm models with $G_M = (2, 4, 8, 16, 32)$, and find that $G_M = 4$ achieves the best validation accuracy (69.2%). SeqNorm with $G_C = 4$ (and $G_M = 4$) achieves the best performance (69.8%) out of $(2, 4, 8, 16, 32, 64)$ G_C values. BatchNorm models only achieve 68.3% top1 accuracy, 0.9% lower than GhostNorm, and 1.5% lower than SeqNorm. Following hyperparameter tuning, the models are trained for more epochs (250 vs 50) and re-evaluated on the validation set. The difference in performance between the normalization layers is consistent with all the previous results, i.e. the highest is SeqNorm (72.3%), then GhostNorm (71.6%), and finally BatchNorm (71.2%).

In addition to the original ImageNet validation set, we also evaluate our models on three recently released test sets for ImageNet (Recht et al., 2019). Without any further retraining (i.e. on the validation set), SeqNorm is able to substantially surpass the reproduced top1 accuracy of BatchNorm by 1.5% while GhostNorm also improves the accuracy by 0.8%.

4 Conclusion

It is generally believed that the cause of performance deterioration of BatchNorm with smaller batch sizes stems from it having to estimate layer statistics using smaller sample sizes (Ioffe, 2017; Wu & He, 2018; Yan et al., 2020). In this work we challenged this belief by demonstrating the effectiveness of GhostNorm on a number of different networks, learning policies, and data sets. For instance, when using super–convergence on CIFAR–10, GhostNorm performs better than BatchNorm, even though the former normalizes the input activations using 4 samples whereas the latter uses all 512 samples. By providing novel insight on the source of regularization in GhostNorm, and by introducing a number of possible implementations, we hope to inspire further research into GhostNorm.

Moreover, based on the understanding developed while investigating GhostNorm, we introduce SeqNorm and follow up with empirical analysis. Surprisingly, SeqNorm not only surpasses the performances of BatchNorm and GhostNorm, but even meets or surpasses current SOTA methodologies on CIFAR–10, CIFAR–100, and ImageNet (Cubuk et al., 2019; Cubuk et al., 2019; Recht et al., 2019). Finally, we also describe and validate a hyperparameter tuning strategy for SeqNorm that provides a faster alternative to the traditional grid–search approach.

References

Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. Understanding batch normalization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 7694–7705. Curran Associates, Inc., 2018.

Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. RandAugment: Practical automated data augmentation with a reduced search space. *arXiv e-prints*, art. arXiv:1909.13719, 2019.

Terrance DeVries and Graham W. Taylor. Improved Regularization of Convolutional Neural Networks with Cutout. *arXiv e-prints*, art. arXiv:1708.04552, 2017.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv e-prints*, art. arXiv:1706.02677, 2017.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 1729—1739. Curran Associates Inc., 2017.

G. Huang, Z. Liu, and Q. K. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach and D. Blei (eds.), *International Conference on Machine Learning*, volume 37, pp. 448–456, July 2015.

Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1942—-1950. Curran Associates Inc., 2017.

Jonas Kohler, Hadi Daneshmand, Aurelien Lucchi, Thomas Hofmann, Ming Zhou, and Klaus Neymeyr. Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization. In Kamalika Chaudhuri and Masashi Sugiyama (eds.), *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pp. 806–815. PMLR, 16–18 Apr 2019.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv e-prints*, art. arXiv:1607.06450, 2016.

Chunjie Luo, Jianfeng Zhan, Lei Wang, and Wanling Gao. Extended Batch Normalization. *arXiv e-prints*, art. arXiv:2003.05569, March 2020.

Ping Luo, Jiamin Ren, Zhanglin Peng, Ruimao Zhang, and Jingyu Li. Differentiable learning-to-normalize via switchable normalization. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ryggIs0cYQ`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Weight Standardization. *arXiv e-prints*, art. arXiv:1903.10520, 2019.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do ImageNet classifiers generalize to ImageNet? volume 97 of *Proceedings of Machine Learning Research*, pp. 5389–5400. PMLR, Jun 2019.

S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How does batch normalization help optimization? In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, pp. 2483–2493, 2018.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Leslie N. Smith and Nicholay Topin. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates. *arXiv e-prints*, art. arXiv:1708.07120, August 2017.

Cecilia Summers and Michael J. Dinneen. Four things everyone should know to improve batch normalization. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJx8HANFDH.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27*, pp. 3104–3112. Curran Associates, Inc., 2014.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance Normalization: The Missing Ingredient for Fast Stylization. *arXiv e-prints*, art. arXiv:1607.08022, July 2016.

Y. Wu and K. He. Group normalization. In *European Conference on Computer Vision (ECCV)*, 2018.

Junjie Yan, Ruosi Wan, Xiangyu Zhang, Wei Zhang, Yichen Wei, and Jian Sun. Towards stabilizing batch statistics in backward propagation of batch normalization. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SkgGjRVKDS.

S. Zagoruyko and N. Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016.

A Appendix

A.1 GhostNorm Implementations

Herein, we provide both the direct implementation of GhostNorm (Figure 4) as well as through the use of accumulating gradients (Figure 5), as described in section 2.3.

A.2 Loss landscape visualization

Implementation details On MNIST, we train a fully-connected neural network (SimpleNet) with two fully-connected layers of 512 and 300 neurons. The input images are transformed to one-dimensional vectors of 784 channels, and are normalized based on the mean and variance of the training set. The learning rate is set to 0.4 for a batch size of 512 on a single GPU. In addition to training SimpleNet with BatchNorm and GhostNorm, we also train a SimpleNet baseline without any normalization technique.

A residual convolutional network with 56 layers (ResNet–56) (He et al., 2015) is employed for CIFAR–10. We achieve super–convergence by using the one cycle learning policy described in the work of Smith and Topin (Smith & Topin, 2017). Horizontal flipping, and pad-and-crop transformations are used for data augmentation. Most of the hyperparameter values were adopted from the work of Smith & Topin (2017). In particular, we employ stochastic gradient descent with a weight decay of 0.0001, and a one-cycle learning policy linearly increasing from 0.1 to 3.0 in 15 epochs, linearly decreasing to 0.1 in the next 15 epochs, and decreasing to 0.003 linearly over the last 10

```
def GhostNorm(X, groupsM, eps=1e-05):
    """
    X: Input Tensor with (M, C, F) dimensions
    groupsM: Number of groups for the mini-batch dimension
    eps: A small value to prevent division by zero
    """
    # Split the mini-batch dimension into groups of smaller batches
    M, C, F = X.shape
    X = X.reshape(groupsM, -1, C, F)

    # Calculate statistics over dim(0) x dim(2) number
    # of slices of dim(1) x dim(3) dimension each
    mean = X.mean([1, 3], keepdim=True)
    var = X.var([1, 3], unbiased=False, keepdim=True)

    # Normalize X
    X = (X - mean) / (torch.sqrt(var + self.eps))

    # Reshape into the initial tensor shape
    X = X.reshape(M, C, F)

    return X
```

Figure 4: Python code for GhostNorm in PyTorch.

epochs. The optimizer does not employ any momentum. In order to train ResNet–56 without a normalization technique (baseline), we had to adjust the cyclical learning rate schedule to $(0.1, 1)$.

We train the networks on $50,000$ and $60,000$ training images (CIFAR–10 and MNIST respectively), and evaluate on $10,000$ testing images.

Loss landscape For MNIST, we compute the loss at 8 learning rates $\in [0.1, 0.2, 0.3, ..., 0.8]$, whereas for CIFAR–10, we do so for 4 cyclical learning rates $\in [(0.05, 1.5), (0.1, 3), (0.15, 4.5), (0.2, 6)]$, and analogously for the baseline.

Results On MNIST, the smoothness of the loss landscape decreases with a larger G_M (see Figure 6). The best model used GhostNorm with G_M set to 64, normalizing over 8 samples. On CIFAR–10, we only observe an effect on the training loss landscape with large values of G_M, $\forall g_m \in G_M = \{16, 32, 64, 128\}$. Nevertheless, all G_M configurations with $G_M > 1$, i.e. GhostNorm, improved optimization with models performing better than the BatchNorm baseline on the testing set.

```
def train__for_an_epoch():
    model.train()
    model.zero_grad()
    for i, (X, y) in enumerate(train_loader):
        outputs = model(X)
        loss = loss_function(outputs, y)
        loss = loss / acc_steps
        loss.backward()
        if (i + 1) % acc_steps == 0:
            optimizer.step()
            model.zero_grad()
```

Figure 5: Python code for accumulating gradients in PyTorch.

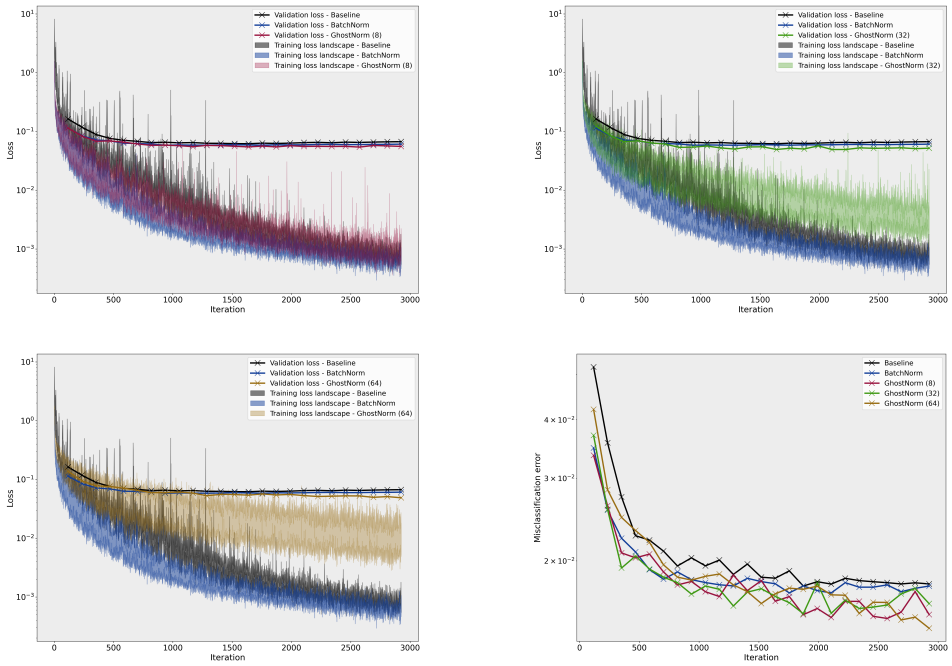

Figure 6: Comparison of the loss landscape on MNIST between the baseline, BatchNorm, and GhostNorm with different G_M values. The last figure (bottom right) depicts the misclassification error on the testing set during training.

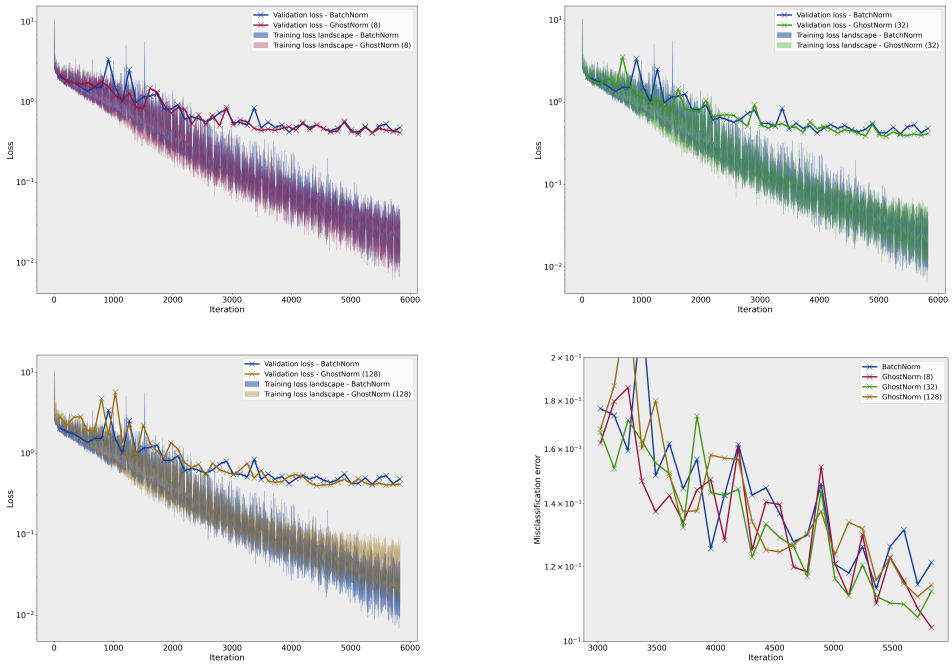

Figure 7: Comparison of the loss landscape on CIFAR–10 between BatchNorm and GhostNorm with different G_M values. The last figure (bottom right) depicts the misclassification error on the testing set during training.

A.3 IMAGE CLASSIFICATION

Implementation details For both CIFAR–10 and CIFAR–100, we employ a training set of $45,000$ images, a validation set of $5,000$ images (randomly stratified from the training set), and a testing set of $10,000$. The input data were stochastically augmented with horizontal flips, pad-and-crop as well as Cutout (DeVries & Taylor, 2017). We use the same hyperparameter configurations as Cubuk et al. (Cubuk et al., 2019). However, in order to speed up optimization, we increase the batch size from 128 to 512, and apply a warmup scheme (Goyal et al., 2017) that increases the initial learning rate by four times in 5 epochs; thereafter we use the cosine learning schedule. Based on the above experimental settings, we train Wide-ResNet models of 28 depth and 10 width (Zagoruyko & Komodakis, 2016) for 200 epochs. Note that since 8 GPUs are employed, our BatchNorm baselines are equivalent to using GhostNorm with $G_M = 8$. Nevertheless, to avoid any confusion, we refer to it as BatchNorm. It's worth mentioning that setting G_M to 8 on 8 GPUs is equivalent to using 64 on 1 GPU.

For ImageNet, we train on 1.28 million training images and evaluate on 50,000 validation images, as well as on three testing sets with 10,000 images each (Luo et al., 2020). We adopt the methodology described in the NVIDIA's public repository for training on 8 GPUs (20.08 docker container) using a ResNet-18 v1.5 architecture. See the repository for the full implementation details[2]. We tune our hyperparameters using the 50 epoch script, and then retrain using the 250 epochs script. Both mixed precision (AMP) and DALI are employed. Other than the addition of GhostNorm and SeqNorm, the only other change we implement is to clip the gradients (threshold=2) as it allows for a more stable training.

For the ablation studies on CIFAR–100, we tune noisy Batch Norm with Gaussian noise of zero mean and standard deviations of 0.00003, 0.0001, 0.0003, $\ldots$, 0.1. The best validation accuracy is achieved with a standard deviation of 0.0003. For models with dropout, we test values of 0.03, 0.1, 0.2, 0.3, and 0.4 for all BatchNorm, GhostNorm, and SeqNorm. Dropout consistently worsens the validation accuracy and is thus omitted from the results. Finally, for RandAugment, we try N values of $[1, 2]$ and M values of $[2, 6, 10, 14]$ as also reported by Cubuk et al. (2019). The best configurations are as follows: $(1, 6)$ for BatchNorm, $(1, 14)$ for GhostNorm, and $(1, 4)$ for SeqNorm.

A.4 NEGATIVE RESULTS

A number of other approaches were adopted in conjunction with GhostNorm and SeqNorm. These preliminary experiments on CIFAR–100 did not surpass the BatchNorm baseline performances on the validation sets (most often than not by a large margin), and are therefore not included in detail. Note that given a more elaborate hyperparameter tuning phase, i.e. that would include learning rate, weight decay, etc., these approaches may had otherwise succeeded.

In particular, we have also experimented with placing GhostNorm and GroupNorm in reverse order for SeqNorm (in retrospect, this could have been expected given what we describe in Section 2.2), and have also experimented with augmenting SeqNorm and GhostNorm with weight standardisation (Qiao et al., 2019) as well as by computing the variance of batch statistics on the whole input tensor (Luo et al., 2020). Finally, on all datasets, we have attempted to tune networks with only GroupNorm (Wu & He, 2018) but the networks were either unable to converge or they achieved worse performances than the BatchNorm baselines.

[2]Repository available at https://github.com/NVIDIA/DeepLearningExamples/