Four Things Everyone Should Know to Improve Batch Normalization

Cecilia Summers
Department of Computer Science
University of Auckland
cecilia.summers.07@gmail.com

Michael J. Dinneen
Department of Computer Science
University of Auckland
mjd@cs.auckland.ac.nz

Abstract

A key component of most neural network architectures is the use of normalization layers, such as Batch Normalization. Despite its common use and large utility in optimizing deep architectures that are otherwise intractable, it has been challenging both to generically improve upon Batch Normalization and to understand specific circumstances that lend themselves to other enhancements. In this paper, we identify four improvements to the generic form of Batch Normalization and the circumstances under which they work, yielding performance gains across all batch sizes while requiring no additional computation during training. These contributions include proposing a method for reasoning about the current example in inference normalization statistics which fixes a training vs. inference discrepancy; recognizing and validating the powerful regularization effect of Ghost Batch Normalization for small and medium batch sizes; examining the effect of weight decay regularization on the scaling and shifting parameters γ and β; and identifying a new normalization algorithm for very small batch sizes by combining the strengths of Batch and Group Normalization. We validate our results empirically on four datasets: CIFAR-100, SVHN, Caltech-256, and ImageNet.

1 Introduction

Neural networks have transformed machine learning, forming the backbone of models for tasks in computer vision, natural language processing, and robotics, among many other domains [15, 5, 17, 28, 4]. A key component of many neural networks is the use of normalization layers such as Batch Normalization [13], Group Normalization [34], or Layer Normalization [1], with Batch Normalization the most commonly used for vision-based tasks. While the true reason why these methods work is still an active area of research [24], normalization techniques typically serve the purpose of making neural networks more amenable to optimization, allowing the training of very deep networks without the use of careful initialization schemes [27, 36], custom nonlinearities [14], or other more complicated techniques [35]. Even in situations where training without normalization layers is possible, their usage can still aid generalization [36]. In short, normalization layers make neural networks train faster and generalize better.

Despite this, it has been challenging to improve normalization layers. In the general case, a new approach would need to be uniformly better than existing normalization methods, which has proven difficult. It has even been difficult to tackle a simpler task: characterizing when specific changes to common normalization approaches might yield benefits. In all, this has created an environment where approaches such as Batch Normalization are still used as-is, unchanged since their creation.

In this work we identify four techniques that everyone should know to improve their usage of Batch Normalization, arguably the most common method for normalization in neural networks. Taken together, these techniques apply in all circumstances in which Batch Normalization is currently used, ranging from large to very small batch sizes, including one method which is even useful when the batch size $B = 1$, and for each technique we identify the circumstances under which it is expected to be of use. In summary, our contributions are:

1. A way to more effectively use the current example during inference, fixing a discrepancy between training and inference that had been previously overlooked,
2. Identifying Ghost Batch Normalization, originally designed for very large-batch multi-GPU training [9], as surprisingly effective even in the medium-batch, single-GPU regime,
3. Recognizing weight decay of the scaling and centering variables γ and β as a valuable source of regularization, an unstudied detail typically neglected, and
4. Proposing a generalization of Batch and Group Normalization in the small but greater-than-one batch size setting, effectively making use of cross-example information present in the minibatch even when such information is not enough for effective normalization on its own.

Experimentally, we study the most common use-case of Batch Normalization, image classification, which is fundamental to most visual problems in machine learning. In total, these four techniques can have a surprisingly large effect, improving accuracy by over 6% on one of our benchmark datasets while only changing the usage of Batch Normalization layers.

The remainder of this paper is organized as follows: in Sec. 2 we provide an overview of work related to Batch Normalization, and in Sec. 3 we give details of each of the techniques. For clarity, we demonstrate experiments throughout the exposition, examining each technique in detail, and then provide experiments combining the techniques in Sec. 4, concluding with Sec. 5.

2 Related Work/Background on normalization methods

Most normalization approaches in neural networks, including Batch Normalization, have the general form of normalizing their inputs x_i to have a learnable mean and standard deviation:

$$\hat{x}_i = \gamma \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta \tag{1}$$

where γ and β are the learnable parameters, typically initialized to 1 and 0, respectively. Where approaches typically differ is in how the mean μ_i and variance σ_i^2 are calculated.

Batch Normalization [13], the pioneering work in normalization layers, defined μ_i and σ_i^2 as calculated for each channel or feature map separately across a minibatch of data. For example, in a convolutional layer, the mean and variance are computed across all spatial locations and training examples in a minibatch. During inference, these statistics are replaced with an exponential moving average of the mean and variance during training, which allows for the predictions for each image to be independent from all other images. The effectiveness of Batch Normalization is undeniable, playing a key role in nearly all state-of-the-art convolutional neural networks since its discovery [30, 29, 6, 7, 37, 38, 11, 10, 23]. Despite this, there is still a fairly limited understanding of Batch Normalization's efficacy — while Batch Normalization's original motivation was to reduce internal covariate shift during training [13], recent work has instead proposed that its true effectiveness stems from making the optimization landscape smoother [24].

One weakness of Batch Normalization is its critical dependence on having a reasonably large batch size, due to the inherent approximation of estimating the mean and variance with a single batch of data. Several works propose methods without this limitation: Layer Normalization [1], which has found use in many natural language processing tasks [33], tackles this by calculating μ_i and σ_i^2 over all channels, rather than normalizing each channel independently, but does not calculate statistics across examples in each batch. Instance Normalization [31], in contrast, only calculates μ_i and σ_i^2 using the information present in each channel, relying on the content of each channel at different spatial locations to provide effective normalization statistics. Group Normalization [34] generalizes Layer and Instance Normalization, calculating statistics in "groups" of channels, allowing for stronger normalization power than Instance Normalization, but still allowing for each channel to contribute significantly to the statistics used for its own normalization. The number of normalization groups per normalization layer is typically set to a global constant in group normalization, though alternatives such as specifying the number of channels per group have also been tried [34].

Besides these most common approaches, many other forms of normalization also exist: For example, Weight Normalization [22] normalizes the weights of each layer instead of the inputs, parameterizing them in terms of a vector giving the direction of the weights and an explicit scale, which must be initialized very carefully. Batch Renormalization [12] explicitly tries to address the small batch problem by using the moving average of batch statistics to normalize during training, parameterized in such a way that gradients still propagate through the minibatch mean and standard deviation, but introduces two new hyperparameters and still suffers somewhat diminished performance in the small-batch setting, performing worse than Group Normalization [34]. More recently, Switchable Normalization [19] aims to learn a more effective normalizer by calculating μ_i and σ_i^2 as learned weighted combinations of the statistics computed from other normalization methods. While flexible, care must be taken for two reasons: First, as the parameters are learned differentiably, they are fundamentally aimed at minimizing the training loss, rather than improved generalization, which typical hyperparameters are optimized for on validation sets. Second, the choice of which normalizers to include in the weighted combination remains important, manifesting in Switchable Normalization's somewhat worse performance than Group Normalization for small batch sizes. Beyond these, there are many approaches we omit for lack of space [18, 2, 8, 14, 35, 36].

3 Improving Normalization: What Everyone Should Know

In this section we detail four methods for improving Batch Normalization. We also refer readers to the Appendix for a discussion of methods which do *not* improve normalization layers (sometimes surprisingly so). For clarity, we choose to interleave descriptions of the methods with experimental results, which aids in understanding each of the approaches as they are presented. We experiment with four standard image-centric datasets: CIFAR-100, SVHN, Caltech-256, and ImageNet, reporting results on validation datasets in this section, and give additional experiments and details in Sec. 4.1.

3.1 Inference Example Weighing

Batch Normalization has a disparity in function between training inference: As previously noted, Batch Normalization calculates its normalization statistics over each minibatch of data separately while training, but during inference a moving average of training statistics is used, simulating the expected value of the normalization statistics. Resolving this disparity is a common theme among methods that have sought to replace Batch Normalization [1, 31, 22, 34, 12]. Here we identify a key component of this training versus inference disparity which can be fixed within the context of Batch Normalization itself, improving it in the general case: when using a moving average during inference, each example does not contribute to its own normalization statistics.

To give an example of the effect this has, we consider the output range of Batch Normalization. During training, due to the inclusion of each example in its own normalization statistics, it can be shown[1] that the minimum possible output of a Batch Normalization layer is:

$$\min_{x_1,\ldots,x_{B-1}} \gamma \frac{x_0 - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta = -\gamma\sqrt{B-1} + \beta \tag{2}$$

with a corresponding maximum value of $\gamma\sqrt{B-1} + \beta$, where B is the batch size, and we assume for simplicity that Batch Norm is being applied non-convolutionally. In contrast, during inference the output range of Batch Normalization is *unbounded*, creating a discrepancy.

Fortunately, once this problem has been realized, it is possible to fix — we need only figure out how to incorporate example statistics during inference. Denoting m_x as the moving average over x and m_{x^2} the corresponding moving average over x^2, we apply the following normalization:

$$\begin{aligned} \mu_i &= \alpha E[x_i] + (1-\alpha) m_x \\ \sigma_i^2 &= (\alpha E[x_i^2] + (1-\alpha) m_{x^2}) - \mu_i^2 \\ \hat{x}_i &= \gamma \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta \end{aligned} \tag{3}$$

where α is the contribution of x_i to the normalization statistics, and we have reparameterized the variance as $\sigma_i^2 = E[x_i^2] - E[x_i]^2$.

[1]Proof in Appendix.

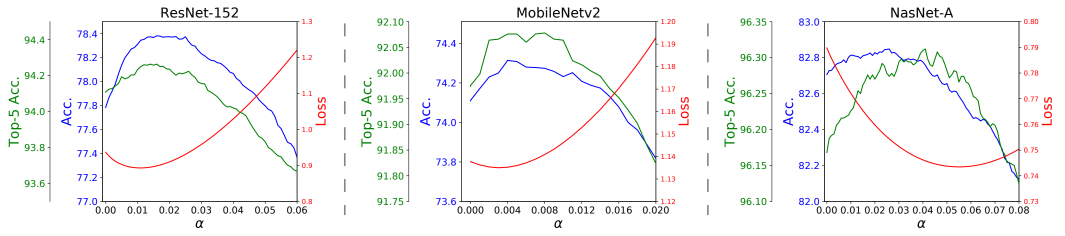

Figure 1: Effect of the example-weighing hyperparameter α on ImageNet for ResNet-152, MobileNetV2, and NASNet-A, measuring top-1 and top-5 accuracies and the cross-entropy loss.

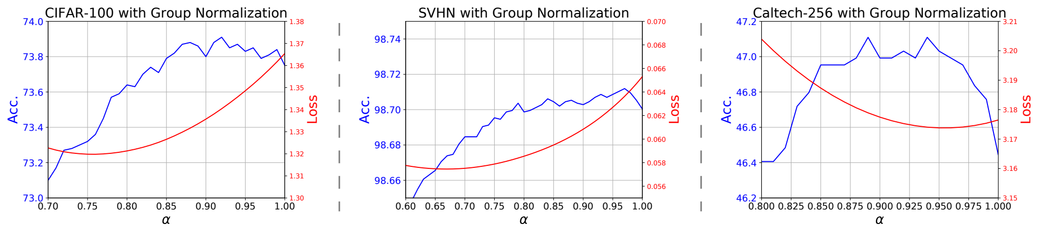

Figure 2: Effect of the example-weighing hyperparameter α for models trained with Group Normalization on CIFAR-100, SVHN, and Caltech-256.

Given this formulation, a natural question is the choice of the parameter α, where $\alpha = 0$ corresponds to the classical inference setting of Batch Normalization and $\alpha = 1$ replicates the setting of techniques which do not use cross-image information in calculating normalization statistics. Intuitively, it would make sense for the optimal value to be $\alpha = \frac{1}{B}$. However, this turns out to not be the case — instead, α is a hyperparameter best optimized on a validation set, whose optimal value may depend the model, dataset, and metric being optimized. While counterintuitive, this can be explained by the remaining set of differences between training and inference: for a basic yet fundamental example, the fact that the model has been fit on the training set (also typically with data augmentation) may produce systematically different normalization statistics between training and inference.

An advantage of this technique is that we can apply it retroactively to any model trained with Batch Normalization, allowing us to verify its efficacy on a wide variety of models. In Fig. 1 we show the effect of α on the ImageNet ILSVRC 2012 validation set [21] for three diverse models: ResNet-152 [7], MobileNetV2 [23], and NASNet-A Large [38][2]. On ResNet-152, for example, proper setting of α can increase accuracy by up to 0.6%, top-5 accuracy by 0.16%, and loss by a relative 4.7%, which are all quite significant given the simplicity of the approach, the competitiveness of ImageNet as a benchmark, and the fact that the improvement is essentially "free" — it involves only modifying the inference behavior of Batch Normalization layers, and does not require any re-training. Across models, the optimal value for α was largest for NASNet-A, the most memory-intensive (and therefore smallest batch size) model of the three.

Surprisingly, it turns out that this approach can have positive effects on models trained without any cross-image normalization at all, such as models trained with Group Normalization [34]. We demonstrate this in Fig. 2, where we find that adding a tiny amount of information from the moving average statistics can actually result in small improvements, with relatively larger improvements in accuracy on Caltech-256 and cross entropy loss on CIFAR-100 and SVHN. This finding is extremely surprising, since adding in any information from the moving averages at all represents a clear difference from the training setting of Group Normalization. Similar to the unintuitive optimal value for α, we hypothesize that this effect is due to other differences in the settings of training and inference: for example, models are generally trained on images that have had data augmentation techniques applied to them, such as random cropping. During inference, though, images appear unperturbed, and it might be the case that incorporating information from the moving averages is a way of influencing the model's intermediate activations to be more similar to those of data augmented

[2]Models obtained from [26].

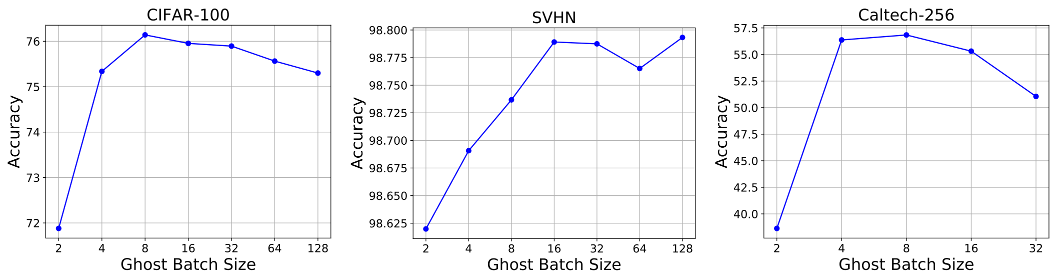

Figure 3: Accuracy vs. Ghost Batch Normalization size for CIFAR-100, SVHN, and Caltech-256.

images, which it has been trained on. This mysterious behavior may also point to more general approaches for resolving training-inference discrepancies, and is worthy of further study.

Summary: Inference example weighing resolves one disparity between training and inference for Batch Normalization, is uniformly beneficial across all models and very easy to tune to metrics of interest, and can be used with any model trained with Batch Normalization, even retroactively.

3.2 Ghost Batch Normalization for Medium Batch Sizes

Ghost Batch Normalization, a technique originally developed for training with very large batch sizes across many accelerators [9], consists of calculating normalization statistics separately for disjoint subsets of each training batch. Concretely, with an overall batch size of B and a "ghost" batch size of B' such that B' evenly divides B, the normalization statistics for example i are calculated as

$$\begin{aligned} \mu_i &= \frac{1}{B'} \sum_{j=1}^{B} x_j \left[\left\lfloor \frac{jB'}{B} \right\rfloor = \left\lfloor \frac{iB'}{B} \right\rfloor \right] \\ \sigma_i^2 &= \frac{1}{B'} \sum_{j=1}^{B} x_j^2 \left[\left\lfloor \frac{jB'}{B} \right\rfloor = \left\lfloor \frac{iB'}{B} \right\rfloor \right] - \mu_i^2 \end{aligned} \tag{4}$$

where $[\cdot]$ is the Iverson bracket, with value 1 if its argument is true and 0 otherwise. Ghost Batch Normalization was previously found to be an important factor in reducing the generalization gap between large-batch and small-batch models [9], and has since been used by subsequent research rigorously studying the large-batch regime [25]. Here, we show that it is also important in the medium-batch setting[3].

In Fig. 3 we show the effects of training with Ghost Batch Normalization. Surprisingly, just using this one simple technique was capable of improving performance by up to 5.8% on Caltech-256, additionally improving CIFAR-100 by 0.84%, which is remarkable given that there is no additional cost during training time. On SVHN, though, where baseline performance is already a very high 98.79% and models do not overfit much, usage of Ghost Batch Normalization did not result in an improvement, giving evidence that at least part of its effect is regularization in nature. In practice, B' may be treated as an additional hyperparameter to optimize over.

Ghost Batch Normalization also has a synergistic effect with inference example weighing — it has the effect of making each example more important in calculating its own normalization statistics μ_i and σ_i^2, with greater effect the smaller B' is, precisely the setting that inference example weighing corrects for. We show these results in Fig. 4, where we find increasing gain from inference example weighing as B' is made smaller, a gain that compounds from the benefits of Ghost Batch Normalization itself. Interestingly, these examples also demonstrate the fact that accuracy and cross-entropy, a standard loss commonly used to optimize for accuracy, are only partially correlated, with the optimal values for the inference example weight α sometimes differing wildly between the two (*e.g.* for SVHN).

[3]We experiment with batch sizes up to 128 in this work.

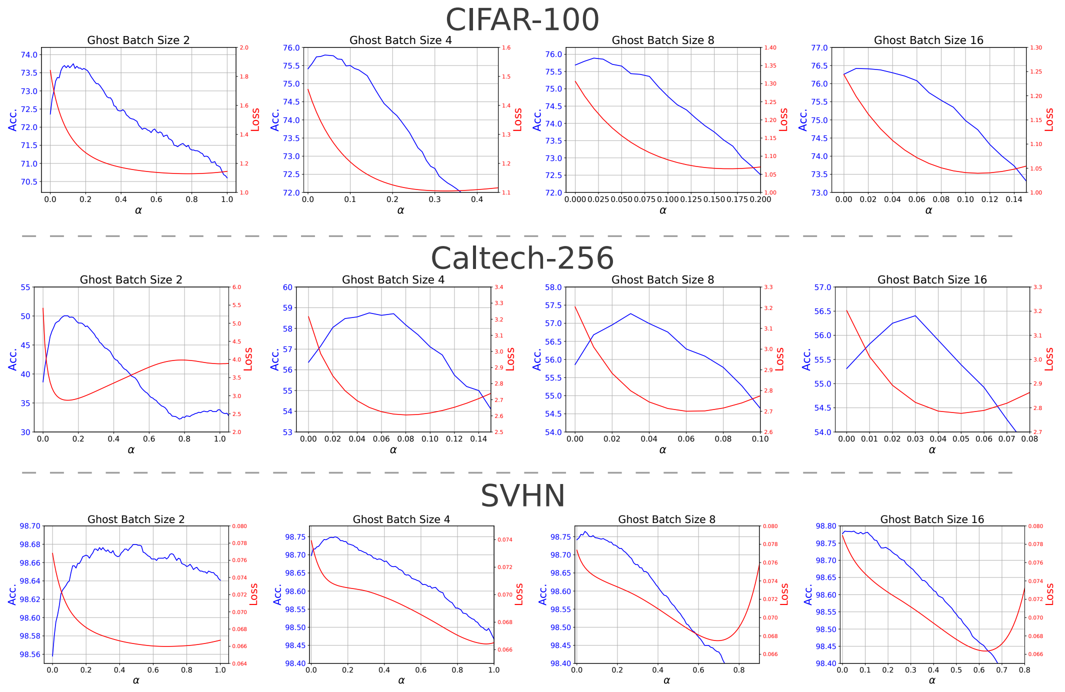

Figure 4: The complementary effects of Inference Example Weighing (Sec. 3.1) and Ghost Batch Normalization (Sec. 3.2) on CIFAR-100, SVHN, and Caltech-256.

Summary: Ghost Batch Normalization is beneficial for all but the smallest of batch sizes, has no computational overhead, is straightforward to tune, and can be used in combination with inference example weighing to great effect.

3.3 Batch Normalization and Weight Decay

Weight decay [16] is a regularization technique that scales the weight of a neural network after each update step by a factor of $1 - \delta$, and has a complex interaction with Batch Normalization. In fact, it may at first seem paradoxical that weight decay has any effect in a network trained with Batch Normalization, as scaling the weights immediately before a normalization layer by any non-zero constant has mathematically almost no effect on the output of the normalization layer (and no effect at all when $\epsilon = 0$). However, related work [8, 32] has shown that weight decay actually has a subtle effect on the effective learning rate of networks trained with Batch Normalization — without weight decay, the weights in a batch-normalized network grow to have large magnitudes, which has an inverse effect on the effective learning rate, hampering training.

Here we turn our attention to the less studied scale and bias parameters common in most normalization methods, γ and β. As far as we are aware, the effect of regularization on γ and β has not been studied to any great extent — the authors of [34] briefly mention weight decay with these parameters, where weight decay was used for γ and β when training from scratch, but not fine-tuning, and two other papers [3, 6] have this form of weight decay explicitly turned off.

Unlike weight decay on weights in *e.g.* convolutional layers, which typically directly precede normalization layers, weight decay on γ and β can have a regularization effect so long as there is a path in the network between the layer in question and the ultimate output of the network, as if such paths do not pass through another normalization layer, then the weight decay is never "undone" by normalization. This structure is only common in certain types of architectures; for example, Residual Networks [6, 7] have such paths for many of their normalization layers due to the chaining of skip-connections. However, Inception-style networks [30, 29] have no residual connections, and despite the fact that each "Inception block" branches into multiple paths, every Batch Normalization layer other than those in the very last block do not have a direct path to the network's output.

We evaluated the effects of weight decay on γ and β on CIFAR-100 across 10 runs, where we found that incorporating it improved accuracy by a small but significant 0.3% ($P = 0.002$). Interestingly, even though γ has a multiplicative effect, we did not find it mattered whether γ was regularized to 0 or 1 ($P = 0.46$) — what was important was whether it had weight decay applied at all.

We did the same comparison on Caltech-256 with Inception-v3 and ResNet-50 networks, where we found evidence that the network architecture plays a crucial effect: for Inception-v3, incorporating weight decay on γ and β actually *hurt* performance by 0.13% (mean across 3 trials), while it improved performance for the ResNet-50 network by 0.91%, supporting the theory that the structure of paths between layers and the network's output are what matter in determining its utility.

On SVHN, where the baseline ResNet-18 already had a performance of 98.79%, we found a similar pattern as with Ghost Batch Normalization — introducing this regularization produced no change.

Summary: Regularization in the form of weight decay on the normalization parameters γ and β can be applied to any normalization layer, but is only effective in architectures with particular connectivity properties like ResNets and in tasks for which models are already overfitting.

3.4 Generalizing Batch and Group Normalization for the Small-Batch Setting

While Batch Normalization is very effective in the medium to large-batch setting, it still suffers in performance when not enough examples are available to calculate reliable normalization statistics. Although we have shown that techniques such as Inference Example Weighing (Sec. 3.1) can help significantly with this, it is still only a partial solution. At the same time, Group Normalization [34] was designed for a batch size of $B = 1$ or greater, but ignores all cross-image information.

In order to generalize Batch and Group Normalization in the batch size $B > 1$ case, we propose to expand the grouping mechanism of Group Normalization from being over only channels to being over both channels and examples — that is, normalization statistics are calculated both *within* groups of channels of each example and *across* examples in groups within each batch.

In principle, this would appear to introduce an additional hyperparameter on top of the number of channel groups used by Group Normalization, both of which would need to be optimized by expensive end-to-end runs of model training. However, in this case we can actually take advantage of the fact that the target batch size is small: if the batch size B is ever large enough that having multiple groups in the example dimension is useful, then it is *also* large enough to eschew usage of the channel groups from Group Normalization, in a regime where either vanilla Batch Normalization or Ghost Batch Normalization is more effective. Thus, when dealing with a small batch size, in practice we only need to optimize over the same set of hyperparameters as Group Normalization.

To demonstrate, we target the extreme setting of $B = 2$, and incorporate Inference Example Weighing to all approaches. For CIFAR-100, this approach improves validation set performance over a tuned Group Normalization by 0.69% in top-1 accuracy (from 73.91% to 74.60%, average over three runs), and on Caltech-256, performance dramatically improved by 5.0% (from 48.2% to 53.2%, average over two runs). However, this approach has one downside: due to differences in feature statistics across examples, when using only two examples the variability in the normalization statistics can still be quite high, even when using multiple channels within each normalization group. As a result, a regularization effect can occur, which may be undesirable for tasks which models are not overfitting much. As in Sec. 3.2 and Sec. 3.3, we see this effect in SVHN, where this approach is actually ever so slightly worse than Group Normalization on the validation set (from 98.75% to 98.73%, average over two runs). On such datasets and tasks, it may be more fruitful to invest in higher-capacity models.

Summary: Combining Group and Batch Normalization leads to more accurate models in the setting of batch sizes $B > 1$, and can have a regularization effect due to Batch Normalization's variability in statistics when calculated over small batch sizes.

4 Additional Experiments

4.1 Experimental Details

All results in Sec. 3 were performed on the validation datasets of each respective dataset. Of the four datasets experimented with, only ImageNet [21] has its own pre-defined validation dataset, so we

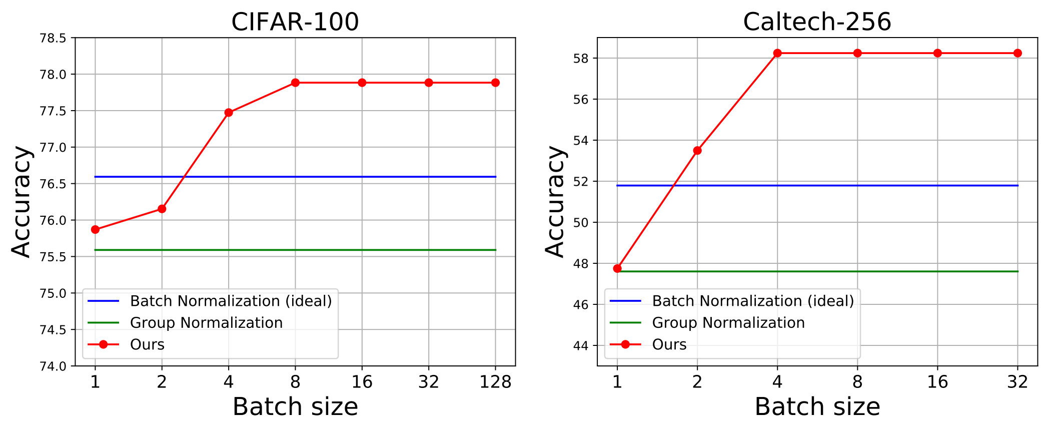

Figure 5: Total performance changes across batch sizes for CIFAR-100 and Caltech-256, incorporating all proposed improvements to Batch Normalization. Also shown is the performance of Group Normalization and an idealized Batch Normalization that scales perfectly across batch sizes.

constructed validation splits for each other datasets as follows: for CIFAR-100 [15], we randomly took 40,000 of the 50,000 training images for the training split, and the remaining 10,000 as a validation split. For SVHN [20], we similarly split the 604,388 non-test images in a 80-20% split for training and validation. For Caltech-256, no canonical splits of any form are defined, so we used 40 images of each of the 256 categories for training, 10 images for validation, and 30 images for testing.

The model used for CIFAR-100 and SVHN was an 18-layer ResNet [7, 6] with 64, 128, 256, and 512 filters across blocks. For Caltech-256, a much larger Inception-v3 [30] model was used. All experiments were done on two Nvidia Geforce GTX 1080 Ti GPUs, and code will be published.

4.2 Improvements Across Batch Sizes

Here we show the end-to-end effect of these four improvements on the test sets of each dataset, comparing against both Batch and Group Normalization. We plot results for CIFAR-100 and Caltech-256 in Fig. 5, comparing against Group Normalization and an idealized Batch Normalization with constant performance across batch sizes. On CIFAR-100, we see improvements against the best available baseline across all batch sizes, noting that Batch Normalization degrades in performance significantly for $B \leq 4$. For larger batch sizes ($B \geq 8$), improvements are driven by the combination of Ghost Batch Normalization (Sec. 3.2), Inference Example Weighing (Sec. 3.1), and weight decay introduced on γ and β (Sec. 3.3), for $B = 1$ improvements are due to the introduced weight decay, and for $B = 2$ the generalization of Batch and Group Normalization leads to the improvement (Sec. 3.4), with some additional effect from weight decay. Improvements on Caltech-256 follow the same trends, but to greater magnitude, with a total increase in performance of 6.5% over Batch Normalization and an increase of 5.9% over Group Normalization for $B = 2$.

5 Conclusion

In this work, we have demonstrated four improvements to Batch Normalization that should be known by all who use it. These include: a method for leveraging the statistics of inference examples more effectively in normalization statistics, fixing a discrepancy between training and inference with Batch Normalization; demonstrating the surprisingly powerful effect of Ghost Batch Normalization for improving generalization of models without requiring very large batch sizes; investigating the previously unstudied effect of weight decay on the scaling and shifting parameters γ and β; and introducing a new approach for normalization in the small batch setting, generalizing and leveraging the strengths of both Batch and Group Normalization. In each case, we have done our best to not only demonstrate the effect of the method, but also provide guidance and evidence for precisely which cases in which it may be effective, which we hope will aid in their applicability.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Lucas Deecke, Iain Murray, and Hakan Bilen. Mode normalization. In *International Conference on Learning Representations*, 2019.

[3] 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 preprint arXiv:1706.02677*, 2017.

[4] 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*, pages 6645–6649. IEEE, 2013.

[5] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[8] Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. In *Advances in Neural Information Processing Systems*, pages 2164–2174, 2018.

[9] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pages 1731–1741, 2017.

[10] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[11] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[12] Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Advances in Neural Information Processing Systems*, pages 1945–1953, 2017.

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *In Proceedings of The 32nd International Conference on Machine Learning*, pages 448–456, 2015.

[14] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980, 2017.

[15] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[16] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[17] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[18] Etai Littwin and Lior Wolf. Regularizing by the variance of the activations' sample-variances. In *Advances in Neural Information Processing Systems*, pages 2119–2129, 2018.

[19] Ping Luo, Jiamin Ren, and Zhanglin Peng. Differentiable learning-to-normalize via switchable normalization. In *International Conference on Learning Representations*, 2019.

[20] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[21] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[22] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016.

[23] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[24] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2488–2498, 2018.

[25] Christopher J Shallue, Jaehoon Lee, Joe Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600*, 2018.

[26] N. Silberman and S. Guadarrama. Tensorflow-slim image classification model library. `https://github.com/tensorflow/models/tree/master/research/slim`.

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[28] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[29] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[30] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[31] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[32] Twan van Laarhoven. L2 regularization versus batch and weight normalization. *arXiv preprint arXiv:1706.05350*, 2017.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[34] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

[35] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. In *International Conference on Machine Learning*, 2018.

[36] Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. In *International Conference on Learning Representations*, 2019.

[37] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

[38] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

Appendix

A Proof of Batch Normalization Output Bounds

Here we present a proof of Eq. 2. We first prove the bound as an inequality and then show that it is tight. Without loss of generality, we assume that x_0 is the minimum of $\{x_i\}_{i=0}^{B-1}$ and that $\gamma \geq 0$. Then we want to show that

$$\min_{x_1,\ldots,x_{B-1}} \gamma \frac{x_0 - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta = -\gamma\sqrt{B-1} + \beta \tag{5}$$

Expanding μ_i and σ_i^2 (using the maximum likelihood estimator for σ_i^2), and canceling the scaling and offset terms γ and β, we want to show

$$\min_{x_1,\ldots,x_{B-1}} \frac{x_0 - \frac{1}{B}\sum_{i=0}^{B-1} x_i}{\sqrt{\frac{1}{B}\sum_{i=0}^{B-1}(x_i - \frac{1}{B}\sum_{j=0}^{B-1} x_j)^2 + \epsilon}} = -\sqrt{B-1} \tag{6}$$

From here we assume without loss of generality that $x_0 = 0$ and that all $x_i \geq 0$, and frame the minimum as a bound

$$\frac{-\frac{1}{B}\sum_{i=0}^{B-1} x_i}{\sqrt{\frac{1}{B}\sum_{i=0}^{B-1}(x_i - \frac{1}{B}\sum_{j=0}^{B-1} x_j)^2 + \epsilon}} \geq -\sqrt{B-1} \tag{7}$$

$$-\frac{1}{B}\sum_{i=0}^{B-1} x_i \geq -\sqrt{\frac{B-1}{B}\sum_{i=0}^{B-1}\left(x_i - \frac{1}{B}\sum_{j=0}^{B-1} x_j\right)^2 + \epsilon} \tag{8}$$

$$-\frac{1}{B}\sum_{i=0}^{B-1} x_i \geq -\sqrt{\frac{B-1}{B}\sum_{i=0}^{B-1}\left(x_i^2 - \frac{2x_i}{B}\sum_{j=0}^{B-1} x_j + \frac{1}{B^2}\left(\sum_{j=0}^{B-1} x_j\right)^2\right) + \epsilon} \tag{9}$$

$$\sum_{i=0}^{B-1} x_i \leq \sqrt{\sum_{i=0}^{B-1}\left((B-1)Bx_i^2 - 2(B-1)x_i\sum_{j=0}^{B-1} x_j + \frac{B-1}{B}\left(\sum_{j=0}^{B-1} x_j\right)^2\right) + \epsilon} \tag{10}$$

$$\sum_{i=0}^{B-1} x_i \leq \sqrt{(B-1)B\sum_{i=0}^{B-1} x_i^2 - 2(B-1)\left(\sum_{i=0}^{B-1} x_i\right)^2 + (B-1)\left(\sum_{i=0}^{B-1} x_i\right)^2 + \epsilon} \tag{11}$$

$$\sum_{i=0}^{B-1} x_i \leq \sqrt{(B-1)B\sum_{i=0}^{B-1} x_i^2 - (B-1)\left(\sum_{i=0}^{B-1} x_i\right)^2 + \epsilon} \tag{12}$$

$$\left(\sum_{i=0}^{B-1} x_i\right)^2 \leq (B-1)B\sum_{i=0}^{B-1} x_i^2 - (B-1)\left(\sum_{i=0}^{B-1} x_i\right)^2 + \epsilon \tag{13}$$

$$B\left(\sum_{i=0}^{B-1} x_i\right)^2 \leq (B-1)B\sum_{i=0}^{B-1} x_i^2 + \epsilon \tag{14}$$

$$\left(\sum_{i=0}^{B-1} x_i\right)^2 \leq (B-1)\sum_{i=0}^{B-1} x_i^2 + \epsilon \tag{15}$$

Using the fact that $x_0 = 0$ and $\epsilon > 0$, it suffices to show

$$\left(\sum_{i=1}^{B-1} x_i\right)^2 \leq (B-1)\sum_{i=1}^{B-1} x_i^2 \tag{16}$$

With a change of variables, we have the more general

$$\left(\sum_{i=0}^{N-1} x_i\right)^2 \leq N\sum_{i=0}^{N-1} x_i^2 \tag{17}$$

$$\frac{1}{N^2}\left(\sum_{i=0}^{N-1} x_i\right)^2 \leq \frac{1}{N}\sum_{i=0}^{N-1} x_i^2 \tag{18}$$

$$\mathbb{E}[x]^2 \leq \mathbb{E}[x^2] \tag{19}$$

$$\mathbb{E}[x^2] - \mathbb{E}[x]^2 \geq 0 \tag{20}$$

which is simply an alternate form for the variance of x, which is always non-negative, completing the bound.

To show that the bound is tight, we can set $x_0 = 0$ and $x_i = a$ for all $i > 0$, where a is a non-negative constant:

$$\frac{-\frac{1}{B}\sum_{i=0}^{B-1} x_i}{\sqrt{\frac{1}{B}\sum_{i=0}^{B-1}(x_i - \frac{1}{B}\sum_{j=0}^{B-1} x_j)^2 + \epsilon}} \tag{21}$$

$$\frac{-\frac{1}{B}\sum_{i=1}^{B-1} a}{\sqrt{\frac{1}{B}\left(\frac{(B-1)^2}{B^2}a^2 + \sum_{i=1}^{B-1}(a - \frac{B-1}{B}a)^2\right) + \epsilon}} \tag{22}$$

$$\frac{-\frac{B-1}{B}a}{\sqrt{\frac{1}{B}\left(\frac{(B-1)^2}{B^2}a^2 + (B-1)a^2\left(1 - \frac{2(B-1)}{B} + \frac{(B-1)^2}{B^2}\right)\right) + \epsilon}} \tag{23}$$

$$\frac{-(B-1)a}{B\sqrt{\frac{a^2(B-1)}{B}\left(\frac{B-1}{B^2} + 1 - 2\frac{B-1}{B} + \frac{(B-1)^2}{B^2}\right) + \epsilon}} \tag{24}$$

$$\frac{-(B-1)a}{\sqrt{a^2(B-1)\left(\frac{B-1}{B} + B - 2B + 2 + \frac{(B-1)^2}{B}\right) + \epsilon}} \tag{25}$$

$$\frac{-(B-1)a}{\sqrt{a^2(B-1)\left(\frac{B^2-2B+1+B-1-B^2+2B}{B}\right) + \epsilon}} \tag{26}$$

$$\frac{-(B-1)a}{\sqrt{a^2(B-1) + \epsilon}} \tag{27}$$

As $a \to \infty$ (or if $\epsilon = 0$), then this approaches

$$\frac{-(B-1)a}{a\sqrt{(B-1)}} \tag{28}$$

which is simply

$$-\sqrt{(B-1)} \tag{29}$$

completing the proof.

B Negative Results: Approaches That Didn't Work.

Here we detail a handful of approaches which seemed intuitively promising but ultimately failed to produce positive results.

Batch Normalization Moving Averages. In an attempt to resolve the other disparities Batch Normalization has between its training and inference behaviors, we experimented with a handful of different approaches for modifying the moving averages used during inference. First, since examples at inference time do not have data augmentation applied to them, we tried computing the moving averages over examples without data augmentation (implemented by training the model for a few extra epochs over non-augmented examples with a learning rate of 0, but while still updating the moving average variables). This decreased accuracy on CIFAR-100 by roughly half a percent, though it did yield mild improvements to the test set cross-entropy loss.

Next, we experimented with calculating the moving averages over the *test set*, not making use of any of the test labels. Perhaps surprisingly, this behaved very similar to when moving averages were calculated over the training examples (within 0.1% in accuracy and within 1% in cross-entropy), with trends holding regardless of whether data augmentation was applied or not.

Adding Batch Normalization-like Stochasticity to Group Normalization. One of the hypotheses for why Group Normalization generally performs slightly worse than Batch Normalization is the regularization effect of Batch Normalization due to random minibatches producing variability in the normalization statistics. Therefore, we tried introducing stochasticity to Group Normalization in a variety of ways, none of which we could get to work well: 1) Adding gaussian noise to the normalization statistics, where the noise is based on a moving average of the normalization statistics, 2) Using random groupings of channels for calculating normalization statistics (optionally only doing randomization a fraction of the time), and 3) changing the number of groups throughout the training procedure, either as increasing or decreasing functions of training steps.

More Principled Group Size Computation. As part of generalizing Batch and Group Normalization, we examined whether it was possible to determine the number of groups in each normalization layer in a more principled way that simply specifying it as a constant throughout the network. For example, one approach we had mild success with was setting the number of elements per group (height $\times$ width $\times$ group size) to a constant, making the number of elements contributing to the normalization statistics uniform across layers. However, we were unable to get any of these ideas to work in a way that generalized properly across datasets. We also tried learning group sizes in a differentiable way with Switchable Normalization, but found that this made models overfit too much.