Understanding the Role of Noisy Statistics in the Regularization Effect of Batch Normalization

Atli Kosson **Dongyang Fan** **Martin Jaggi**
EPFL, Switzerland
firstname.lastname@epfl.ch

Abstract

Normalization layers have been shown to benefit the training stability and generalization of deep neural networks in various ways. For Batch Normalization (BN), the noisy statistics have been observed to have a regularization effect that depends on the batch size. Following this observation, Hoffer et. al. proposed Ghost Batch Normalization (GBN), where BN is explicitly performed independently on smaller sub-batches, resulting in improved generalization in many settings. In this study we analyze and isolate the effect of the noisy statistics by comparing BN and GBN, introducing a noise injection method. We then quantitatively assess the effects of the noise, juxtaposing it with other regularizers like dropout and examining its potential role in the generalization disparities between batch normalization and its alternatives, including layer normalization and normalization-free methods.

1 Introduction

The use of normalization methods has become widespread in deep learning since the introduction of Batch Normalization [8] (BN) in 2015. For convolutional network training, the batch normalization of a tensor $\boldsymbol{X} \in \mathbb{R}^{N\times C\times H\times W}$ with batch size N, C channels, height H, and width W, is given by:

$$\hat{\boldsymbol{X}} = \frac{\boldsymbol{X} - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2 + \varepsilon}}, \qquad \boldsymbol{\mu} = \frac{1}{NHW}\sum_{n,h,w} \boldsymbol{X}_{n,:,h,w}, \quad \boldsymbol{\sigma}^2 = \frac{1}{NHW}\sum_{n,h,w} (\boldsymbol{X} - \boldsymbol{\mu})^2_{n,:,h,w} \tag{1}$$

where $\boldsymbol{\mu} \in \mathbb{R}^{1\times C\times 1\times 1}$ is the mean, $\boldsymbol{\sigma} \in \mathbb{R}^{1\times C\times 1\times 1}$ the variance, and operations are broadcasted and performed elementwise. The mean and variance are computed over the batch dimension causing the final prediction of a given sample to depend on others in the batch, *a cross-sample dependency*. With randomly sampled batches, the cross-sample dependency causes $\boldsymbol{\mu}, \boldsymbol{\sigma}$ to be noisy for a given sample. This is undesirable for inference, as data can arrive in small correlated batches or even one sample at a time. In practice, $\boldsymbol{\mu}, \boldsymbol{\sigma}$ from Equation 1 are replaced with exponential moving averages of the values used during training. Note that during training each sample contributes to its own statistics, forming a *self-dependency*, but the modified inference mode eliminates this resulting in a train-test discrepancy that can degrade performance [12, 13].Train-test discrepancy (we mainly refer to self-dependency here) and noise magnitude both increase with smaller batch sizes. While the first one is harmful, the second can be beneficial, introducing a potential trade-off for generalization performance.

Ghost Batch Normalization [7] is a modification of Batch Normalization where we split $\boldsymbol{X}$ into G sub-batches of size $N = B/G$ each, $\boldsymbol{X} = [\boldsymbol{X}_1, \ldots, \boldsymbol{X}_G]$, and then apply Equation 1 to each one independently. This results in different $\boldsymbol{\mu}_g, \boldsymbol{\sigma}_g$ for each group $g \in \{1, \ldots, G\}$ with increased noise due to the reduced batch size. We will refer to the noise in the batch norm statistics as *Ghost Noise*.

In this work we study the role of ghost noise in the improved generalization of batch normalization. We find that it is a potent source of regularization, outperforming explicit regularization methods like Dropout and simpler noise forms in our experiments. Our empirical results suggest that reducing the self-dependency in training while increasing the noise can both contribute to increased performance.

Mathematics of Modern Machine Learning Workshop at NeurIPS 2023.

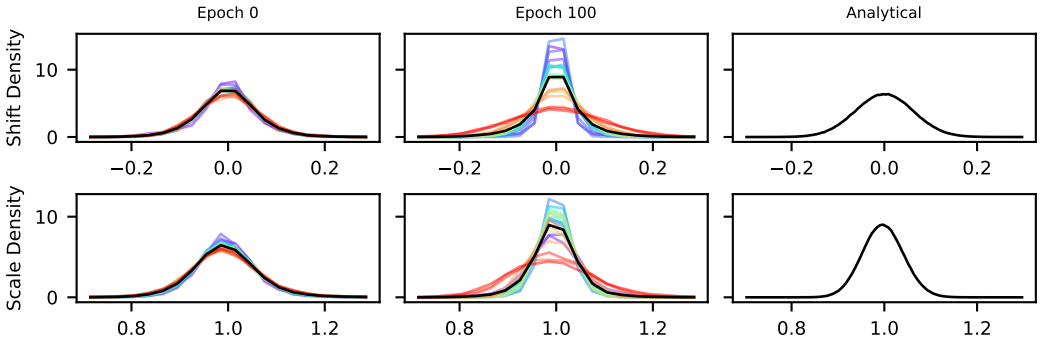

Figure 1: Measured noise distributions for ResNet18 on CIFAR100 with a Ghost Batch Size of 32 along with analytical fully connected distributions for batch size 256. Each GNI line is an average over a layer, the distributions also vary between channels inside each layer. The lines are plotted with a standard color spectrum (rainbow) from violet (first layer) to red (last layer).

2 Analysis

2.1 Modelling GBN as Double Normalization

We present a novel perspective on GBN aimed at separating noise effects from normalization effects. The key insight lies in the fact that applying standard BN before GBN preserves the output of GBN. For a batch $\boldsymbol{X}$, this relationship can be expressed as:

$$\mathrm{GBN}(\boldsymbol{X}) = \mathrm{GBN}(\ \mathrm{BN}(\boldsymbol{X})\ ) \tag{2}$$

This follows from the fact that normalization is invariant to affine transformations of the inputs. For a sub-batch $\boldsymbol{X}_g$ with (mean,std) of $(\boldsymbol{\mu}_g, \boldsymbol{\sigma}_g)$ on the original or $(\hat{\boldsymbol{\mu}}_g, \hat{\boldsymbol{\sigma}}_g)$ after BN we obtain:

$$(\boldsymbol{X}_g - \boldsymbol{\mu}_g)/\boldsymbol{\sigma}_g = 1/\hat{\boldsymbol{\sigma}}_g \left((\boldsymbol{X}_g - \boldsymbol{\mu})/\boldsymbol{\sigma} - \hat{\boldsymbol{\mu}}_g \right) \quad \Longrightarrow \quad \boldsymbol{\mu}_g = \boldsymbol{\mu} + \boldsymbol{\sigma}\hat{\boldsymbol{\mu}}_g, \quad \boldsymbol{\sigma}_g = \boldsymbol{\sigma}\hat{\boldsymbol{\sigma}}_g \tag{3}$$

where we have ignored ε. GBN would be equivalent to BN if $\hat{\boldsymbol{\mu}}_g = 0$ and $\hat{\boldsymbol{\sigma}}_g = 1$. This happens when the ghost batches are identical, but generally, they contain different samples resulting in random variations in $\hat{\boldsymbol{\mu}}_g$ and $\hat{\boldsymbol{\sigma}}_g$. These fluctuations induce noise in two components, additive (or shifting) noise from $\hat{\boldsymbol{\mu}}_g$ and multiplicative (or scaling) noise from $\hat{\boldsymbol{\sigma}}_g$. The increased train-test discrepancy of GBN arises from the dependency of $\hat{\boldsymbol{\mu}}_g$ and $\hat{\boldsymbol{\sigma}}_g$ on a specific sample (self-dependency).

2.2 Noise Distributions

The estimated mean $\hat{\boldsymbol{\mu}}_g$ and standard deviation $\hat{\boldsymbol{\sigma}}_g$ from the g-th ghost batch, can be interpreted as bootstrapped statistics derived from the empirical distribution $P_{\hat{\boldsymbol{X}}}$. Following the normalization in

```python
import torch
def ghost_noise_injection(X: torch.Tensor, N: int, eps: float=1e-3):
    B, C, _, _ = X.shape  # Shape: Batch (B), Channels (C), Height (H), Width (W)
    with torch.no_grad():
        batch_var, batch_mean = torch.var_mean(X, dim=(0,2,3), correction=0)
        ghost_var = torch.zeros(size=(B,C), device=X.device)
        ghost_mean = torch.zeros(size=(B,C), device=X.device)
        for idx in range(B):
            ghost_sample = torch.randint(high=B, size=(N,))
            ghost_var[idx], ghost_mean[idx] = torch.var_mean(
                X[ghost_sample], dim=[0,2,3], correction=0)
        shift = (ghost_mean - batch_mean)
        scale = torch.sqrt((ghost_var + eps)/(batch_var + eps))
    return (X - shift.view(B, C, 1, 1)) / scale.view(B, C, 1, 1)
```

Figure 2: A minimal PyTorch implementation of Ghost Noise Injection for CNNs without any performance optimizations. Epsilon helps with numerical stability (eps=1e-3 in all experiments).

Equation 1, the variable $\hat{\boldsymbol{X}}$ exhibits a zero mean and unit variance. If we assume that the individual elements of $\hat{\boldsymbol{X}}$ are normally and independently distributed, the distributions of $\hat{\boldsymbol{\mu}}_g = [\hat{\mu}_{g,1}, \ldots, \hat{\mu}_{g,C}]$ and $\hat{\boldsymbol{\sigma}}_g = [\hat{\sigma}_{g,1}, \ldots, \hat{\sigma}_{g,C}]$ for a **fully connected layer** are:

$$\hat{\mu}_{g,c} = \frac{1}{N}\sum_{n=1}^{N} \hat{X}_{n,c} \sim \mathcal{N}\left(0, \frac{1}{N}\right); \qquad \hat{\sigma}^2_{g,c} = \frac{1}{N}\sum_{n=1}^{N} (\hat{X}_{n,c} - 0)^2 \sim \frac{1}{N}\chi^2(N) \tag{4}$$

In **convolutional layers** the variance of each channel can both come from the spatial dimensions and the batch dimension. This results in different distributions for $\hat{\boldsymbol{\mu}}_g$ and $\hat{\boldsymbol{\sigma}}_g$ depending on how much of the variance comes from each dimension (Appendix E). The distributions can therefore vary between channels and over time, as we show empirically in Figure 1 and explore further in our experiments.

2.3 Injecting Ghost Noise without GBN

Equation 3 results in a noisy shift of $\boldsymbol{\sigma}\hat{\boldsymbol{\mu}}_g$ and additional noisy scaling of $\hat{\boldsymbol{\sigma}}_g$. We can decrease the self-dependency and decouple this noise from the normalization, applying it directly to the inputs:

$$\frac{\boldsymbol{X} - (\mathbf{m} - \boldsymbol{\mu})}{\mathbf{s}/\boldsymbol{\sigma}} \tag{5}$$

which we call Ghost Noise Injection, with $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ computed like in batch normalization, and $\mathbf{m}$ and $\mathbf{s}$ as the mean and standard deviation from a randomly sampled ghost batch of the *unnormalized* input $\boldsymbol{X}$. We interpret $\mathbf{m} - \boldsymbol{\mu}$ as *shift noise* and $\mathbf{s}/\boldsymbol{\sigma}$ as *scale noise*. In Appendix D we give a derivation of Equation 5 by "undoing" the effect of the first normalization in double normalization. A minimal implementation of GNI is shown in Figure 2. GNI offers some key advantages over GBN: (1) Each sample is unlikely to contribute strongly to its own normalization statistics, reducing the train-test discrepancy for a given level of noise; (2) The choice of normalization batch size N is not limited to being a divisor of the accelerator batch size B; (3) GNI can also be applied in BN-free settings.

3 Experiments

In the top-left panel of Figure 3 we observe that smaller ghost batch sizes result in a performance boost, consistent with other studies [7, 13]. Note that the validation accuracy is a direct measure of

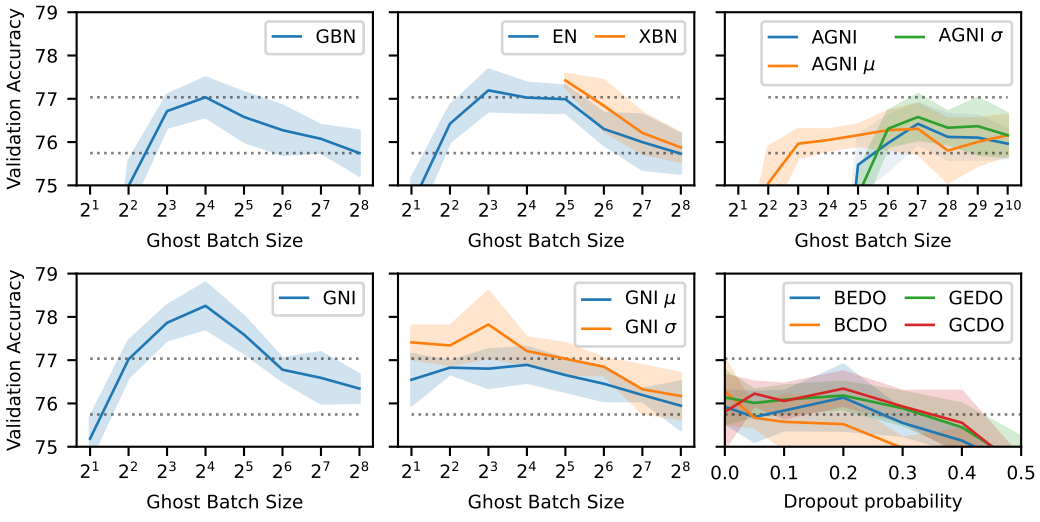

Figure 3: CIFAR-100 ResNet-18 validation accuracy varies with ghost batch size and dropout probability across methods (averaged over 5 runs ± std. deviation). The dotted lines show standard BN and maximum of GBN. In the dropout panel (lower right), options include B=Bernoulli, G=Gaussian, E=elementwise, and C=channelwise.

the generalization as ResNet-18 can easily overfit this dataset in practice, achieving roughly 100% train accuracy.

Reduced train-test discrepancy improves generalization performance: In the top-center panel, we see that methods that mitigate the train-test discrepancy, EvalNorm (EN [12]) and Exclusive Batch Normalization (XBN, see Appendix B) marginally improve this, but overall the train-test discrepancy seems to be less significant than the amount of noise present. The lower-left panel of Figure 3 demonstrates that GNI can significantly outperform GBN. This is likely due to additional ghost noise from independently sampling the ghost batch for every layer, without additional self-dependency. The lower-center panel explores GNI with only the scale (GNI σ) or shift (GNI μ) component of the noise. Both components regularize, but the scaling one has a slightly stronger effect.

Ghost Noise outperforms simpler additive/multiplicative noise methods: The upper right panel shows that Analytical GNI (AGNI) based on Equation 4 is unable to match the sampled version, suggesting that the differences over time and layers/channels (e.g. Figure 1) may matter for efficient regularization. This is supported by the lower-right panel where we place dropout layers where we compare four variants of dropout (Bernoulli/Gaussian and Element-wise/Channel-wise) to the other methods. We find that dropout performs similarly to AGNI but is unable to match GBN or sample-based GNI. We apply the dropout after the second normalization layer on each branch, as proposed in Wide Residual Networks [17].

Regularization effect of Ghost Noise in BN-free settings: GNI allows us to explore the effect of ghost noise in settings that do not include batch normalization. Figure 4 shows that this type of noise can close the gap between a network using Weight Standardization [11] with LayerNorm[1] and the ghost batch normalized version (Figure 3 top-left). Table 1 shows it can also give a slight to boost Normalization-Free networks as well as ViT [5, 4] and ConvMixer [14], both of which rely on Layer Normalization.

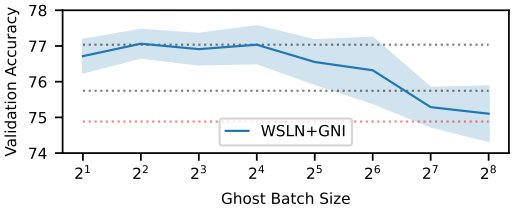

Figure 4: Accuracy impact of GNI on WSLN-ResNet18 trained on CIFAR100. The dotted lines correspond to a BN baseline, best GBN performance, WSLN without GNI (top to bottom).

Table 1: Test Accuracy (mean±std for 3 runs) on CIFAR-10 with/without data augmentation (Aug) and with/without GNI (using ghost batch size N). See Appendix A for details.

Network	Aug	Baseline	GNI (N)
NF ResNet	✗	84.31±0.86	86.18±0.58 (32)
NF ResNet	✓	93.60±0.21	94.19±0.24 (32)
Simple ViT	✗	71.45±0.12	74.36±0.27 (4)
Simple ViT	✓	88.19±0.08	89.55±0.05 (12)
ConvMixer	✗	91.18±0.24	91.57±0.08 (192)
ConvMixer	✓	93.29±0.17	93.66±0.12 (128)

4 Conclusion

In this study, we explored the generalization benefits of using smaller batch sizes in batch normalization. By splitting Ghost Batch Normalization into two normalization steps, we analyzed their impact on noise and train-test disparities. Our analysis revealed channel-dependent noise characteristics, involving shifting and scaling, which are crucial for overall effectiveness. Additionally, we showed that mitigating train-test discrepancies can be achieved by preventing over-contribution of samples to their own normalization statistics during training. Building on these findings, we introduced Ghost Noise Injection (GNI) as a novel technique to enhance ghost noise levels without requiring smaller batch sizes, reducing train-test mismatches. Empirical investigations highlighted the positive influence of ghost noise on generalization, even in BN-free scenarios. The use of Batch Normalization with moderate and small batch sizes provides potent regularization resembling the use of explicit regularization methods like Dropout in other settings.

References

[1] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] L. Beyer, X. Zhai, and A. Kolesnikov. Better plain vit baselines for imagenet-1k. *arXiv preprint arXiv:2205.01580*, 2022.

[3] A. Brock, S. De, S. L. Smith, and K. Simonyan. High-performance large-scale image recognition without normalization. In *International Conference on Machine Learning*, pages 1059–1071. PMLR, 2021.

[4] J.-B. Cordonnier, A. Loukas, and M. Jaggi. On the relationship between self-attention and convolutional layers. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=HJlnC1rKPB`.

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=YicbFdNTTy`.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. arXiv:1512.03385.

[7] E. Hoffer, I. Hubara, and D. Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *Advances in neural information processing systems*, 30, 2017.

[8] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015. arXiv:1502.03167.

[9] A. Krizhevsky and G. Hinton. Cifar-100 (canadian institute for advanced research). *Dataset available from https://www.cs.toronto.edu/ kriz/cifar.html*, 2009.

[10] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[11] S. Qiao, H. Wang, C. Liu, W. Shen, and A. L. Yuille. Weight standardization. *CoRR*, abs/1903.10520, 2019. URL `http://arxiv.org/abs/1903.10520`.

[12] S. Singh and A. Shrivastava. Evalnorm: Estimating batch normalization statistics for evaluation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3633–3641, 2019.

[13] C. Summers and M. J. Dinneen. Four things everyone should know to improve batch normalization. In *International Conference on Learning Representations*, 2019.

[14] A. Trockman and J. Z. Kolter. Patches are all you need? *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL `https://openreview.net/forum?id=rAnB7JSMXL`. Featured Certification.

[15] P. Wang. vit-pytorch. `https://github.com/lucidrains/vit-pytorch`, 2023.

[16] R. Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

[17] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

A Experimental Details

The code for our experiments is written in PyTorch [10] and based on the TIMM [16] library. It is available at `https://github.com/epfml/ghost-noise`.

Hyperparameter Tuning: The optimization hyperparameters employed in each experiment are grounded in pre-existing hyperparameters from the original works. We generally perform a light tuning of the baseline learning rate (all CIFAR-based experiments) and the warmup period (for extension to other architectures). This tuning is performed on a validation set created by randomly sampling 10% of the original training set. For Ghost Noise Injection we sweep over batch sizes in the set $\{2^1, 2^2, \ldots, 2^8\}$, except for extension to other architectures where we also consider the halfway points between these. This tuning is also performed on a validation set and typically averaged over three seeds. We do not re-tune other hyperparameters (e.g. the learning rate) for use with GNI.

A.1 ResNet Experiments

Base Setup: We employ ResNet-18 [6] for training on the CIFAR-100 dataset [9] in our primary experimental configuration. The dataset consists of 32x32 images split into 50'000 training and 10'000 test samples, evenly split between 100 classes. For hyperparameter tuning, we use a random subset of 10% of the training dataset as validation set. Our reported outcomes stem from test set results associated with the most effective hyperparameters, established through retraining on the complete training set. The training duration for all models spans 200 epochs, featuring a cosine decay learning rate schedule alongside a 5-epoch warmup, with a batch size of 256. Given the dataset's relatively modest size and the limited number of images per class, the significance of regularization is amplified. ResNet-18 is a moderately sized network that helps keep the computational cost of training reasonable, thereby permitting multiple runs under various settings. For all experiments, a learning rate of $\eta = 0.3$ is employed. This parameter was tuned on the validation set.

ResNet18 + CIFAR-100: We use the data augmentation from the original ResNet [6] paper, i.e. random flips and a zero padding of 2 with a random crop of the original size. Inputs are normalized with mean `(0.5071, 0.4867, 0.4408)` and std `(0.2675, 0.2565, 0.2761)`. Optimization is performed using SGD with momentum $\alpha = 0.9$ using a learning rate $\eta = 0.3$ and weight decay $\lambda = 5 \cdot 10^{-4}$ (gains and biases excluded) at a batch size (accelerator, optimizer) of 256. The learning rate is tuned on the validation set, and for other hyperparameters, we use recommended default values. A cosine decay schedule is used for the learning rate, with a 5-epoch warmup (stepwise). We train for a total of 200 epochs (which are shorter when using a portion of the train set for validation). One run takes around 45 minutes on a V100 GPU (baseline with batch normalization) and around 60 minutes with GNI (without any kernels or other optimizations). We use the same setup for the LayerNorm + Weight Standardization experiments. The Weight Standardization includes a learnable gain like in Normalization Free Networks [3] and scales the output to match the initialization magnitude.

ResNet20 + CIFAR-10: The setup for CIFAR-10 [9] is the same as for CIFAR-100 except otherwise stated. The inputs are normalized with mean `(0.4914, 0.4822, 0.4465)` and std `(0.2023, 0.1994, 0.2010)`. The base learning rate is $\eta = 0.2$ and the weight decay $\lambda = 2 \cdot 10^{-4}$. One run takes around 14 minutes on a V100 (batch norm baseline) and 25 minutes with GNI (without any optimizations for speed).

ResNet50 + ImageNet-1k: The data augmentation is based on the original ResNet [6] paper. The training transforms consists of a torchvision RandomResizedCrop to 224x224 using the default scaling range of $[0.08, 1.0]$ and ratio range of $[0.75, 1.33]$, a random horizontal flip and a normalization for mean `(0.485, 0.456, 0.406)` and std `(0.229, 0.224, 0.225)`. For inference, we resize input images to 256 and center crop 224. The optimizer is SGD with momentum $\alpha = 0.9$ using a learning rate $\eta = 0.1$ and weight decay $\lambda = 10^{-4}$. A batch size of 256 (optimizer and accelerator batch size) is applied. We use a step-wise cosine decay learning rate schedule, with a 1 epoch warmup, and train for a total of 90 epochs. Standard training takes around 31 hours on a single V100 GPU using float16 mixed precision.

A.2 Extension to Other Architectures

NF-ResNet + CIFAR-10: The network used is a Normalization Free ResNet proposed by Brock et al. [3]. We modified the existing implementation in the TIMM library [16] to use an input block

similar to the CIFAR style ResNets, i.e. a single 3x3 convolution without any stride or pooling. The network used in the experiments has a depth 26, uses bottleneck blocks, and has 4 stages (operating on different spatial sizes) each with 2 blocks. The number of channels in the stages is `(256, 512, 1024, 2048)` between the stages and one-fourth of that in the 3x3 convolutions. We enable the use of SkipInit in our experiments (disabled by default in TIMM) and use the default alpha value of 0.2. When using GNI we place it right before each activation function (ReLU). The inputs are normalized with mean `(0.4914, 0.4822, 0.4465)` and std `(0.2023, 0.1994, 0.2010)` like in our other CIFAR-10 experiments. The data augmentation (when used) is a random horizontal flip with probability 0.5, padding of 4 zero values on each side followed by selecting a random 32x32 patch of the resulting image. We train for 200 epochs using a cosine decay schedule with a 20-epoch learning rate warmup, updating the learning rate after every step. Optimization is performed using SGD with momentum 0.9, learning rate $2 \cdot 10^{-1}$, and weight decay $5 \cdot 10^{-5}$ at a batch size of 256 in float16-float32 mixed precision. One run takes around 50 minutes on a V100 (baseline) and 75 minutes with GNI (without speed optimizations).

Simple ViT + CIFAR-10: The Simple Visual Transformer is a modification of the original ViT [5] proposed by some of the original authors in a technical report [2]. Our implementation is based on the ViT-PyTorch repository [15]. The network we train has 6 blocks (depth), 16 heads, embedding dimension of 512, MLP dimension of 1024, patch size of 4x4 (on the original input size of 32x32) and uses GELU activations. When using GNI we insert it after every Layer Normalization (including the affine transformation). The inputs are normalized with mean `(0.4914, 0.4822, 0.4465)` and standard deviation `(0.2023, 0.1994, 0.2010)` like in our other CIFAR-10 experiments. The data augmentation (when used) is a random horizontal flip with probability 0.5, padding of 4 zero values on each side followed by selecting a random 32x32 patch of the resulting image. We train for 200 epochs using a cosine decay schedule with a 40-epoch learning rate warmup, updating the learning rate after every step. Optimization is performed using AdamW with learning rate 10^{-3}, weight decay 10^{-4}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$ at a batch size of 256 in float16-float32 mixed precision. One run takes around 45 minutes on a V100 (baseline) and 50 minutes with GNI (without speed optimizations).

ConvMixer + CIFAR-10: Trockman and Kolter [14] proposed the ConvMixer architecture. We use the implementation from the TIMM library [16]. The network used in the experiments has a depth of 8, dimension of 256, kernel size 5 and a patch size of 2x2 (on the original input size of 32x32). The GNI is applied inside of the batch normalization layers (between normalization and the affine transformation). The inputs are normalized with mean `(0.4914, 0.4822, 0.4465)` and std `(0.2023, 0.1994, 0.2010)` like in our other CIFAR-10 experiments. The data augmentation (when used) is a random horizontal flip with probability 0.5, padding of 4 zero values on each side followed by selecting a random 32x32 patch of the resulting image. We train for 200 epochs using a cosine decay schedule with a 5 epoch learning rate warmup, updating the learning rate after every step. Optimization is performed using AdamW with learning rate $2 \cdot 10^{-2}$, weight decay $5 \cdot 10^{-2}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-3}$ at a batch size of 256 in float16-float32 mixed precision. One run takes around 60 minutes on a V100 (baseline) and 80 minutes with GNI (without speed optimizations).

B Exclusive Batch Normalization (XBN)

We propose a straightforward technique in which all elements within a ghost batch are utilized except for the element under consideration, for the calculation of normalization statistics. This aims to mitigate the excessive reliance on a single sample during training and serves as a baseline or a tool to estimate the impact of self-dependency. We term this approach **Exclusive Batch Normalization (XBN)**. Note that XBN applies a separate set of normalization statistics for each sample. Consequently, the training-time normalization statistics become less dependent on the specific sample itself, aligning more closely with the conditions at test time. The mean and variance of the k-th sample in the g-th ghost batch of XBN is computed as:

$$\boldsymbol{\mu}_{g,k} = \frac{1}{(N-1)HW} \sum_{n \neq k,h,w} \boldsymbol{X}_{n,:,h,w} \tag{6}$$

$$\boldsymbol{\sigma}^2_{g,k} = \frac{1}{(N-1)HW} \sum_{n \neq k,h,w} (\boldsymbol{X}_{n,:,h,w} - \boldsymbol{\mu}_{g,k})^2 \tag{7}$$

Importantly, XBN maintains a similar level of noise as GBN while reducing the discrepancy between the train and test time. However, eliminating the self-dependency of normalization comes with a significant drawback: it does not guarantee a bounded output range. We present a detailed analysis in Appendix C. The absence of an output range constraint can lead to training instability. We empirically observe this behavior for smaller batch sizes. Yet, when the batch size is sufficiently large to support stable training, we do observe an increase in test accuracy, as demonstrated in Figure 3. This increase is likely attributed to the reduced train-test discrepancy.

XBN offers a way to address train-test discrepancies, though it requires a reasonably sized batch for optimal performance. Despite this, it remains a valuable baseline for performance comparison.

C Unbounded output range of XBN

Below we analyze the possible output range for Ghost Batch Normalization (GBN) and Exclusive Batch Normalization (XBN), as indicated in Section B. We show that in GBN the range of the output is bounded but XBN offers no such guarantees. This could be the source of the training instability observed with XBN at smaller batch sizes. Removing the self-dependency in the manner performed in XBN therefore results in a trade-off between a lower train-test discrepancy vs training stability.

Bounded output range of GBN Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ be a batch of inputs for a fully connected network (i.e. no spatial dimensions for simplicity). Then the output of Ghost Batch Normalization corresponding to an input $\boldsymbol{x}_i$ is given by:

$$\frac{\boldsymbol{x}_i - \frac{1}{N}\sum_j \boldsymbol{x}_j}{\sqrt{\frac{1}{N}\sum_j (\boldsymbol{x}_j - \frac{1}{N}\sum_t \boldsymbol{x}_t)^2 + \varepsilon}} = \frac{\boldsymbol{x}_i - \frac{1}{N}\sum_j \boldsymbol{x}_j}{\sqrt{\frac{1}{N}\left((\boldsymbol{x}_i - \frac{1}{N}\sum_t \boldsymbol{x}_t)^2 + \sum_{j\neq i}(\boldsymbol{x}_j - \frac{1}{N}\sum_t \boldsymbol{x}_t)^2\right) + \varepsilon}} \tag{8}$$

For analysis purposes, we treat ε as 0 for now. The reciprocal of equation 8 is:

$$\frac{\sqrt{\frac{1}{N}\sum_j (\boldsymbol{x}_j - \frac{1}{N}\sum_t \boldsymbol{x}_t)^2}}{\boldsymbol{x}_i - \frac{1}{N}\sum_j \boldsymbol{x}_j} = \sqrt{\left(\frac{1}{N} + \sum_{j\neq i} \frac{(\boldsymbol{x}_j - \frac{1}{N}\sum_t \boldsymbol{x}_t)^2}{(\boldsymbol{x}_i - \frac{1}{N}\sum_t \boldsymbol{x}_t)^2}\right)} \geq \frac{1}{\sqrt{N}} \tag{9}$$

which implies that the magnitude of the output is bounded by the square root of the ghost batch size.

Unbounded output range of XBN With XBN the output for a single element $\boldsymbol{x}_i$ will be:

$$\frac{\boldsymbol{x}_i - \frac{1}{N-1}\sum_{j\neq i} \boldsymbol{x}_j}{\sqrt{\frac{1}{N-1}\sum_{j\neq i}(\boldsymbol{x}_j - \frac{1}{N-1}\sum_{t\neq i} \boldsymbol{x}_t)^2 + \varepsilon}} \tag{10}$$

The numerator here has no dependency on $\boldsymbol{x}_i$ and if we ignore ε, it can therefore be arbitrarily small in comparison, making the output unbounded. For example when all $\boldsymbol{x}_j, j \neq i$ are identical, the numerator of equation 10 will be $\sqrt{\varepsilon}$, which is close to 0 in practice (the default value in PyTorch is $\varepsilon = 10^{-5}$). Large values like these can destabilize training. We observe this in practice when using smaller batch sizes, due to $\boldsymbol{\sigma}_i^2$ randomly being close to zero for some ghost batches. It may be possible to mitigate this issue, for example by clamping the computed sigma or the resulting output, but we do not pursue this further here.

D Derivation of Equation 5

Consider the double normalization setup described in the analysis section, repeated below:

$$\hat{\boldsymbol{X}} := \text{BN}\,(\boldsymbol{X}) = \frac{\boldsymbol{X} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} = [\hat{\boldsymbol{X}}_1, .., \hat{\boldsymbol{X}}_G] \tag{11}$$

$$\widetilde{\boldsymbol{X}}_g := \text{GBN}\,(\hat{\boldsymbol{X}})_g = \frac{\hat{\boldsymbol{X}}_g - \hat{\boldsymbol{\mu}}_g}{\hat{\boldsymbol{\sigma}}_g} \tag{12}$$

We identified the second transformation (Equation 12) as the source of the increased noise and train-test discrepancy in Ghost Batch Normalization. Our goal in Ghost Noise Injection is twofold, decoupling the noise from the normalization and eliminating the increased train-test discrepancy.

Eliminating the Normalization We first show how we can decouple Equation 12 from Equation 11. The output $\widetilde{\boldsymbol{X}}_g$ in Equation 12 can be written as:

$$\widetilde{\boldsymbol{X}}_g = \frac{1}{\hat{\boldsymbol{\sigma}}_g}\left(\frac{\boldsymbol{X}_g - \boldsymbol{\mu}}{\boldsymbol{\sigma}} - \hat{\boldsymbol{\mu}}_g\right) \tag{13}$$

$$= \frac{\boldsymbol{X}_g}{\hat{\boldsymbol{\sigma}}_g\boldsymbol{\sigma}} - \frac{\boldsymbol{\mu}}{\hat{\boldsymbol{\sigma}}_g\boldsymbol{\sigma}} - \frac{\hat{\boldsymbol{\mu}}_g}{\hat{\boldsymbol{\sigma}}_g} \tag{14}$$

We can undo the effects of the original normalization by multiplying this output by $\boldsymbol{\sigma}$ and then reverse the relative shift by adding $\boldsymbol{\mu}/\hat{\boldsymbol{\sigma}}_g$, accounting by the fact that in $\widetilde{\boldsymbol{X}}_g$ the original shift has been scaled by $\hat{\boldsymbol{\sigma}}_g^{-1}$. We will refer to this "denormalized" output of Equation 12 as GBN_Δ which we can write as:

$$\text{GBN}_\Delta(\boldsymbol{X}) := \boldsymbol{\sigma}\widetilde{\boldsymbol{X}}_g + \frac{\boldsymbol{\mu}}{\hat{\boldsymbol{\sigma}}_g} \tag{15}$$

$$= \frac{\boldsymbol{X} - \boldsymbol{\sigma}\hat{\boldsymbol{\mu}}_g}{\hat{\boldsymbol{\sigma}}_g} \tag{16}$$

The GBN statistics $\hat{\boldsymbol{\mu}}_g$ and $\hat{\boldsymbol{\sigma}}_g$ can be expressed in terms of the original inputs $\boldsymbol{X}$:

$$\hat{\boldsymbol{\mu}}_g = \frac{1}{NHW}\sum_{n,h,w}\hat{\boldsymbol{X}}_g \tag{17}$$

$$= \frac{1}{NHW}\sum_{n,h,w}\left(\frac{\boldsymbol{X}_g - \boldsymbol{\mu}}{\boldsymbol{\sigma}}\right) \tag{18}$$

$$= \frac{1}{\boldsymbol{\sigma}}\Big(\frac{1}{NHW}\sum_{n,h,w}(\boldsymbol{X}_g) - \boldsymbol{\mu}\Big) \tag{19}$$

$$=: \frac{\boldsymbol{\mu}_g - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \tag{20}$$

and we can similarly show that:

$$\hat{\boldsymbol{\sigma}}_g^2 = \frac{1}{\boldsymbol{\sigma}^2}\frac{1}{NHW}\sum_{n,h,w}(\boldsymbol{X}_g - \boldsymbol{\mu}_g)^2 \tag{21}$$

$$=: \boldsymbol{\sigma}_g^2/\boldsymbol{\sigma}^2 \tag{22}$$

Plugging this into Equation 16 gives:

$$\text{GBN}_\Delta(\boldsymbol{X}) = \frac{\boldsymbol{X} - (\boldsymbol{\mu}_g - \boldsymbol{\mu})}{\boldsymbol{\sigma}_g/\boldsymbol{\sigma}} \tag{23}$$

Reducing the Train-Test Discrepancy To mitigate the increased train-test discrepancy with smaller batch sizes in GBN, we propose to re-sample ghost batches for the calculation of normalization statistics. To do so, we can replace Equation 12 with the following:

$$\widetilde{\boldsymbol{X}} = \frac{\hat{\boldsymbol{X}} - \hat{\mathbf{m}}}{\hat{\mathbf{s}}} \tag{24}$$

Where $\hat{\mathbf{m}}$ and $\hat{\mathbf{s}}$ are analogous to $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\sigma}}$, except they are computed over a *random* subset of $\hat{\boldsymbol{X}}$ for each element in the batch. By doing this, each sample only contributes weakly to its own normalization statistics in expectation, similar to batch normalization over the full accelerator batch instead of ghost batch normalization over a smaller ghost batch. We can extend this trick to $\text{GBN}_\Delta(\boldsymbol{X})$ from Equation 23, replacing $\boldsymbol{\mu}_g$ and $\boldsymbol{\sigma}_g$ with $\mathbf{m}$ and $\mathbf{s}$, computed like $\hat{\mathbf{m}}$ and $\hat{\mathbf{s}}$ except *on the original inputs* $\boldsymbol{X}$ *instead of* $\hat{\boldsymbol{X}}$. This gives the expression for GNI in Equation 5:

$$\text{GNI}(\boldsymbol{X}) = \frac{\boldsymbol{X} - (\mathbf{m} - \boldsymbol{\mu})}{\mathbf{s}/\boldsymbol{\sigma}} \tag{25}$$

E Distribution of Ghost Noise for Convolutional Layers

In CNNs the batch statistics are computed across both the batch and spatial dimensions (e.g. the height and width of an image). As a result, the data can exhibit variability in two distinct manners: variation among samples along the batch dimension, termed *inter-sample variance*, and variation within a single sample across the spatial dimension, referred to as *intra-sample variance*. This stands in contrast to the fully connected scenario, where all variance emerges along the batch dimension, and the concept of intra-sample variance is absent.

To gain insight into how the inter- and intra-sample variances could affect the noise distributions we will present and analyze a simplified model. We will assume that $\boldsymbol{X} \in \mathbb{R}^{B\times C\times I}$, where I is the spatial dimension e.g. $I = H \times W$ where H is the height and W the width for image data. We further assume the true mean of b-th batch in a specific channel c is sampled i.i.d. from a normal distribution, i.e. $\mu_{bc} \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2_{Bc})$, where σ^2_{Bc} denotes the inter-sample variance and only varies across c, i.e. it is constant across B. We model the i-th spatial location in the b-th batch as being sampled from $x_{bci} \overset{i.i.d.}{\sim} \mathcal{N}(\mu_{bc}, \sigma^2_{Ic})$, with an intra-sample variance σ^2_{Ic} that does not vary across I. After batch normalization, we always have unit variance, which means $\sigma^2_{Bc} + \sigma^2_{Ic} = 1$, for any c. Now assume we sample a random ghost batch of size N. As the sample mean is an average of all samples, we still have it following a normal distribution.

The distribution of shift noise in channel c we have:

$$\mathbb{E}[\hat{\mu}_{gc}] = \mathbb{E}\left[\frac{1}{NI}\sum_n\sum_i x_{nci}\right] = \frac{1}{NI}\sum_n\sum_i \mathbb{E}[x_{nci}] = 0 \tag{26}$$

with variance:

$$\begin{aligned}
\mathrm{Var}[\hat{\mu}_{gc}] &= \mathbb{E}\left[\frac{1}{NI}\sum_n\sum_i x_{nci}\right]^2 && (27)\\
&= \mathbb{E}\left[\frac{1}{NI}\sum_n\sum_i x_{nci} - \mu_{nc} + \mu_{nc}\right]^2 && (28)\\
&= \frac{1}{N^2I^2}\sum_n\sum_i \sigma^2_{Ic} + \frac{1}{N^2I^2}NI^2\sigma^2_{Bc} && (29)\\
&= \frac{1}{NI}\sigma^2_{Ic} + \frac{1}{N}\sigma^2_{Bc} && (30)
\end{aligned}$$

The final $\hat{\sigma}^2_{gc}$ approximately follows a mixture of two Chi-square distributions:

$$\begin{aligned}
\hat{\sigma}^2_{gc} &= \frac{1}{NI}\sum_n\sum_i (x_{nci} - \mu_{nc} + \mu_{nc})^2 && (31)\\
&= \frac{1}{NI}\sum_n\sum_i \left((x_{nci} - \mu_{nc})^2 + (\mu_{nc} - \mu)^2 + 2(x_{nci} - \mu_{nc})(\mu_{nc} - \mu)\right) && (32)\\
&\overset{\overline{x}_{nc}\approx\mathbb{E}\,x_{nc}}{\approx} \frac{1}{NI}\sum_n\sum_i \sigma^2_{Ic}\left(\frac{x_{nci} - \mu_{nc}}{\sigma_{Ic}}\right)^2 + \frac{1}{N}\sigma^2_{Bc}\sum_n\left(\frac{\mu_{nc} - \mu}{\sigma_{Bc}}\right)^2 && (33)\\
&\quad + \frac{1}{N}(\mu_{nc} - \mu_{nc})(\mu_{nc} - \mu) && (34)\\
&= \frac{1}{NI}\sum_n\sum_i \sigma^2_{Ic}\left(\frac{x_{nci} - \mu_{nc}}{\sigma_{Ic}}\right)^2 + \frac{1}{N}\sigma^2_{Bc}\sum_n\left(\frac{\mu_{nc} - \mu}{\sigma_{Bc}}\right)^2 && (35)\\
&\sim \frac{\sigma^2_{Ic}}{NI}\chi^2(NI) + \frac{\sigma^2_{Bc}}{N}\chi^2(N) && (36)
\end{aligned}$$

F Additional Experimental Results

Generalization to other Datasets Table 2 shows that GNI can also regularize the training of ResNet-50 on ImageNet-1k and ResNet-20 on CIFAR-10. In both cases, GNI provides a decent boost in accuracy and outperforms GBN.

Table 2: Test accuracy for ResNet-20 CIFAR-10 (mean±std for 3 runs) and ResNet-50 ImageNet-1k for different normalization and noise setups. GNI is performed on top of the standard BN-256 for additional regularization.

Setup	CIFAR-10 RN-20	ImageNet-1k RN-50
BN-256	92.22±0.05	77.43
GBN-16	92.65±0.07	76.90
GBN-32	92.53±0.33	77.14
GBN-64	92.50±0.13	77.48
GNI-16	**93.01**±0.28	77.62
GNI-32	92.76±0.11	77.70
GNI-64	92.56±0.12	**77.98**

G Limitations

Studying stochastic regularization methods can be challenging. It requires many repeated experiments to average out the random effects and reveal the true benefits of the methods. This limits the size and number of settings we can explore. We have focused our efforts on ResNet-18 training on CIFAR-100, which is computationally tractable for us. Our findings can be generalized to the other settings of ResNet-50 on ImageNet, Layer-Normalized variant of ResNet-18 on CIFAR-100 and ResNet-20 on CIFAR but we are not able to perform ablations and detailed comparisons across a wider variety of settings.

The main limitations of Ghost Noise Injection are the higher computational cost and the requirement to sample from a batch. The computational cost is similar to normalization, which is higher than that of e.g. dropout. When used in conjunction with batch normalization, a lot of the computation can be shared. Without efficient kernels that can merge some of the operations, the slowdown increases relative to existing, highly optimized, PyTorch operations. We note that the computational cost is only present during training, not inference. The batch dependency could potentially be mitigated by keeping track of a history of mean and variances for multiple batches, at a somewhat increased memory cost during training.