

THE UNIVERSITY OF AUCKLAND

DOCTORAL THESIS

Understanding and Improving Deep Learning

Author:

Cecilia Summers

*A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy*

in the

School of Computer Science, the University of Auckland, 2021.

October, 2021

Abstract

Deep learning is a modern subfield of machine learning, itself a subfield of artificial intelligence. The key distinguishing feature of modern deep learning is the use of many layers of computation, which can be trained with relatively simple and scalable optimization methods such as stochastic gradient descent. By stacking many computation layers together, models in deep learning can represent complex functions that were previously difficult to approximate. This capability has led to significant advances in a variety of fields, including computer vision, natural language processing, game playing, and protein folding, among others.

Despite this success, key pieces of deep learning remain not well-understood, hampering the field's ability to advance. This thesis aims to shed light on a variety of fundamental components in deep learning, using the additional understanding to subsequently improve them. These components are varied: We develop an understanding of mixed-example data augmentation, disproving the hypothesis that linearity is responsible for its efficacy, while introducing new methods that are superior to linear approaches. We show that logit pairing methods aimed at enhanced adversarial robustness derive much of their benefit from logit regularization, then show how they can be extended by other logit regularization techniques. We analyze multiple facets of batch normalization, a critical component of many neural networks, and demonstrate four unique changes that improve it across a variety of settings. We study the effects of nondeterminism on model training, show that they are largely due to instability in model training, and propose two methods for reducing the effects of instability.

Acknowledgements

First and foremost, I would like to thank my advisor Michael J. Dinneen, who has supported me throughout my entire Ph.D. Michael has been an ally from the day I first met him, encouraging me to pursue the research topics that I've been interested in, even when they went in a different direction than originally anticipated. He has routinely been a source of good research advice throughout my Ph.D.

I would also like to thank my family, whose love and support has kept me going, especially when things were difficult. In particular, I thank my parents for raising me and guiding me on the path of life, and I feel truly fortunate to have such a wonderful family.

During my path through higher education, I also encountered many nice and interesting friends who I am thankful for — the number is large enough that I'll forgo listing them all out, but I want them to know that I cherish them all. I also want to explicitly thank other professors and researchers who I met during my Ph.D. for their support and advice.

Last, I'd like to thank the Department of Computer Science of the University of Auckland for supporting my Ph.D. financially in the form of a scholarship. I feel lucky to have done a Ph.D. at the excellent and beautiful University of Auckland.

Table of Contents

List of Tables	vii
List of Figures	ix
1 Introduction	1
2 Mixed-Example Data Augmentation	5
2.1 Introduction	5
2.2 Related Work	7
2.3 Methods	8
2.3.1 General Formulation	8
2.3.2 Linearity-Based Methods	9
2.3.3 Non-Linear Methods	10
2.4 Experiments	15
2.4.1 Implementation Details	15
2.4.2 Results	17
2.5 Conclusion	21

3	Logit Regularization for Adversarial Robustness	22
3.1	Introduction	22
3.2	Overview of Adversarial Training	24
3.3	Adversarial Logit Pairing and Logit Regularization	26
3.3.1	Experimental Evidence	28
3.4	Other forms of logit regularization	29
3.4.1	Decoupling Adversarial Logit Pairing	32
3.5	Additional experiments	33
3.5.1	Implementation Details	34
3.5.2	Towards a more robust model	35
3.5.3	Evaluating Stronger Attacks	37
3.6	Conclusion	38
4	Improving Batch Normalization	40
4.1	Introduction	40
4.2	Related Work/Background on normalization methods	42
4.3	Improving Normalization	43
4.3.1	Inference Example Weighing	44
4.3.2	Ghost Batch Normalization for Medium Batch Sizes	47
4.3.3	Batch Normalization and Weight Decay	50
4.3.4	Generalizing Batch and Group Normalization for Small Batches	51
4.4	Additional Experiments	52
4.4.1	Experimental Details	52
4.4.2	Combining All Four: Improvements Across Batch Sizes	53
4.4.3	Transfer Learning	53
4.4.4	Non-i.i.d. minibatches	55
4.5	Conclusion	56

5	Nondeterminism and Instability	57
5.1	Introduction	57
5.2	Related Work	59
5.3	Nondeterminism	59
5.3.1	Protocol for Testing Effects of Nondeterminism	60
5.3.2	Experiments in Image Classification	62
5.3.3	Experiments in Language Modeling	63
5.3.4	Nondeterminism Throughout Training	65
5.4	Instability	66
5.4.1	Instability and Nondeterminism	66
5.4.2	Instability and Depth	67
5.5	Reducing Variability	68
5.6	Generalization Experiments	69
5.7	Conclusion	72
6	Conclusion	73
A	Appendix: Improving Batch Normalization	74
A.1	Proof of Batch Normalization Output Bounds	74
A.2	Empirical Evidence of Batch Normalization Output Bounds	77
A.3	Negative Results: Approaches That Didn't Work.	77
A.4	Supplemental Inference Example Weighing Plots	79
B	Appendix: Nondeterminism and Instability	82
B.1	Linear, 2-Layer, and ResNet-10 Results	82
B.2	Impact of Random Bit Changes Over Time	82

B.3	Test-Time Augmentation Details	84
B.4	Approaches that don't reduce instability	84
B.5	Accelerated Ensembling in Language Modeling	88
B.6	Subtleties in Evaluation	88
	References	90

List of Tables

2.1	Experimental results on CIFAR-10.	17
2.2	Experimental results on CIFAR-100.	19
2.3	Experimental results on Caltech-256.	20
3.1	White-box accuracy of models on CIFAR-10.	34
3.2	Black-box accuracy of models on CIFAR-10.	36
3.3	White-box accuracy of models on CIFAR-100.	36
3.4	Black-box accuracy of models on CIFAR-100.	38
3.5	White-box accuracy of models on SVHN.	38
3.6	Model evaluation against the strongest white-box attacks on CIFAR-10.	39
4.1	Accuracy on CIFAR-100 with non-i.i.d. minibatches.	55
5.1	Effect of each source of nondeterminism on CIFAR-10.	63
5.2	Effect of each source of nondeterminism for a QRNN on Penn Treebank.	63
5.3	Effect of instability on CIFAR-10.	66
5.4	Effect of instability on Penn Treebank.	67
5.5	Comparison of ensemble model variability against the proposed methods for reducing the effects of nondeterminism on CIFAR-10.	69

5.6	Generalization experiments of nondeterminism and instability with other architectures on CIFAR-10, ImageNet, and MNIST.	70
B.1	Linear, 2-layer, and ResNet-10 experiments on CIFAR-10.	83
B.2	Experiments varying the learning rate and number of epochs for ResNet-14 on CIFAR-10.	87
B.3	Effect of accelerated model ensembling on Penn Treebank.	88

List of Figures

2.1	Illustrative example of mixed-example data augmentation.	6
2.2	Example outputs for each mixed-example data augmentation method.	11
3.1	Effect of logit pairing on the test set logit distribution and effect of combining logit pairing with logit regularization.	29
3.2	Effect of label smoothing on adversarial example accuracy and logit distributions. . . .	30
4.1	Effect of the example-weighting hyperparameter α on ImageNet.	45
4.2	Effect of the example-weighting hyperparameter α for models trained with Group Normalization.	46
4.3	Accuracy vs. Ghost Batch Normalization size for CIFAR-100, SVHN, and Caltech-256.	48
4.4	Interaction between Inference Example Weighing and Ghost Batch Normalization. . . .	49
4.5	Total performance changes across batch sizes and datasets when incorporating all improvements to Batch Normalization.	54
5.1	Average CKA representation similarity for pairs of ResNet-14 models on CIFAR-10. . . .	64
5.2	Effect of the onset of nondeterminism on the variability of accuracy in converged models. .	65
A.1	Comparison of observed output value range with theoretical bound on CIFAR-10. . . .	78
A.2	Effect of the example-weighting hyperparameter α on ImageNet; full range of α	80

A.3	Effect of the example-weighting hyperparameter α for models trained with Group Normalization; full range of α	80
A.4	The interaction between Inference Example Weighing and Ghost Batch Normalization; full range of α	81
B.1	Effect of random bit changes for linear vs 2-layer models.	85

Co-Authorship Form

This form is to accompany the submission of any PhD that contains published or unpublished co-authored work. **Please include one copy of this form for each co-authored work.** Completed forms should be included in all copies of your thesis submitted for examination and library deposit (including digital deposit), following your thesis Acknowledgements. Co-authored works may be included in a thesis if the candidate has written all or the majority of the text and had their contribution confirmed by all co-authors as not less than 65%.

Please indicate the chapter/section/pages of this thesis that are extracted from a co-authored work and give the title and publication details or details of submission of the co-authored work.

Chapter 2
 Improved mixed-example data augmentation. In IEEE Winter Conference on Applications of Computer Vision, 2019.

Nature of contribution by PhD candidate	Ideation, implementation, experimentation, and writing
---	--

Extent of contribution by PhD candidate (%)	95
---	----

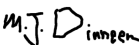
CO-AUTHORS

Name	Nature of Contribution
Michael J. Dinneen	Supervision and writing

Certification by Co-Authors

The undersigned hereby certify that:

- ❖ the above statement correctly reflects the nature and extent of the PhD candidate's contribution to this work, and the nature of the contribution of each of the co-authors; and
- ❖ that the candidate wrote all or the majority of the text.

Name	Signature	Date
Michael J. Dinneen		6 May 2021

Co-Authorship Form

This form is to accompany the submission of any PhD that contains published or unpublished co-authored work. **Please include one copy of this form for each co-authored work.** Completed forms should be included in all copies of your thesis submitted for examination and library deposit (including digital deposit), following your thesis Acknowledgements. Co-authored works may be included in a thesis if the candidate has written all or the majority of the text and had their contribution confirmed by all co-authors as not less than 65%.

Please indicate the chapter/section/pages of this thesis that are extracted from a co-authored work and give the title and publication details or details of submission of the co-authored work.	
Chapter 3	
Improved adversarial robustness via logit regularization methods. arXiv, 2019	

Nature of contribution by PhD candidate	Ideation, implementation, experimentation, and writing
---	--

Extent of contribution by PhD candidate (%)	95
---	----

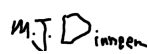
CO-AUTHORS

Name	Nature of Contribution
Michael J. Dinneen	Supervision and writing

Certification by Co-Authors

The undersigned hereby certify that:

- ❖ the above statement correctly reflects the nature and extent of the PhD candidate's contribution to this work, and the nature of the contribution of each of the co-authors; and
- ❖ that the candidate wrote all or the majority of the text.

Name	Signature	Date
Michael J. Dinneen		6 May 2021

Co-Authorship Form

This form is to accompany the submission of any PhD that contains published or unpublished co-authored work. **Please include one copy of this form for each co-authored work.** Completed forms should be included in all copies of your thesis submitted for examination and library deposit (including digital deposit), following your thesis Acknowledgements. Co-authored works may be included in a thesis if the candidate has written all or the majority of the text and had their contribution confirmed by all co-authors as not less than 65%.

Please indicate the chapter/section/pages of this thesis that are extracted from a co-authored work and give the title and publication details or details of submission of the co-authored work.

Chapter 4; Appendix A
 Four things everyone should know to improve batch normalization. In International Conference on Learning Representations, 2020.

Nature of contribution by PhD candidate	Ideation, implementation, experimentation, and writing
---	--

Extent of contribution by PhD candidate (%)	95
---	----

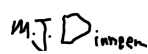
CO-AUTHORS

Name	Nature of Contribution
Michael J. Dinneen	Supervision and writing

Certification by Co-Authors

The undersigned hereby certify that:

- ❖ the above statement correctly reflects the nature and extent of the PhD candidate's contribution to this work, and the nature of the contribution of each of the co-authors; and
- ❖ that the candidate wrote all or the majority of the text.

Name	Signature	Date
Michael J. Dinneen		6 May 2021

Co-Authorship Form

This form is to accompany the submission of any PhD that contains published or unpublished co-authored work. **Please include one copy of this form for each co-authored work.** Completed forms should be included in all copies of your thesis submitted for examination and library deposit (including digital deposit), following your thesis Acknowledgements. Co-authored works may be included in a thesis if the candidate has written all or the majority of the text and had their contribution confirmed by all co-authors as not less than 65%.

Please indicate the chapter/section/pages of this thesis that are extracted from a co-authored work and give the title and publication details or details of submission of the co-authored work.

Chapter 5; Appendix B
 Nondeterminism and instability in neural network optimization. In International Conference on Machine Learning, 2021.

Nature of contribution by PhD candidate	Ideation, implementation, experimentation, and writing
---	--

Extent of contribution by PhD candidate (%)	95
---	----

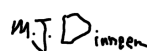
CO-AUTHORS

Name	Nature of Contribution
Michael J. Dinneen	Supervision and writing

Certification by Co-Authors

The undersigned hereby certify that:

- ❖ the above statement correctly reflects the nature and extent of the PhD candidate's contribution to this work, and the nature of the contribution of each of the co-authors; and
- ❖ that the candidate wrote all or the majority of the text.

Name	Signature	Date
Michael J. Dinneen		6 May 2021

1. Introduction

As a field, deep learning is focused on learning algorithms that would otherwise be difficult to create. It is a subfield of machine learning, which has largely similar goals, but what distinguishes deep learning methods is the use of models composed of many layers of computation, which allows for the flexible expression of complex functions. Each layer of computation is typically parameterized by a set of numbers, or *weights*, which are used to compute a simple function (*e.g.* a convolution [63]), and by stacking many of these simpler functions together and optimizing a task-specific objective, much more sophisticated functions can be learned. Examples of successes of deep learning in recent years include computer vision [59], natural language processing [116], game playing [97], and protein folding [92], among others.

Many components of deep learning are necessary in order to be successful. First, a representative dataset for the task in question is required. This dataset must be large enough that it is possible to effectively learn the appropriate task-specific patterns in the data: for example, early breakthroughs of deep learning in image classification [59] used a dataset with more than 1 million images, each annotated with one of 1,000 categories. When it is prohibitive to collect a large enough dataset, and often even when a fairly large dataset is available, it is common to apply a technique called *data augmentation* [59, 98, 14], which consists of transforming each example in a myriad of different ways to produce multiple synthetic examples, which can artificially increase the dataset size by several orders of magnitude. For example, in the context of images, horizontally flipping produces another image which is likely equally valid for a given training task.

Next, a suitable model architecture is required, which defines the structure of the model that will learn from the data. Although in theory very simple model architectures can learn arbitrarily complex patterns in data [11], in practice it is effective and worthwhile to use a somewhat task-specific architecture. For example, for many tasks in computer vision, the relevant patterns or features in the data are

similar at different locations in an image (*e.g.* a cat or bicycle looks the same regardless of where in an image it is), so it is common to use architectures with convolutional layers [63], in which the weights to be learned are shared across all locations in the image. Over time, newer architectures that are more effective at common tasks have been developed [59, 35, 128].

Last, a task-specific target for optimization, or *loss function*, and a method for optimizing it are required. For classification tasks, the most common loss function is the cross-entropy between the correct label given in the dataset and the probability distribution over labels predicted by the model. In terms of the optimization method itself, by far the most common approach is stochastic gradient descent [5] (SGD), or variants of it. Gradient descent minimizes the objective function by updating weights in the opposite direction of the gradient, calculated across the entire dataset, and stochastic gradient descent approximates it by considering the gradient on a small subset of the dataset, which is typically much more efficient in practice.

Despite the success of deep learning, many of these components remain poorly understood. The work in this thesis examines a range of these such topics, seeking to increase the field’s understanding and use the newfound understanding to develop better deep learning algorithms.

In Chapter 2 (published as [103] in WACV 2019, an A-ranked conference), we consider the topic of data augmentation. Specifically, we consider a new and interesting variant of data augmentation, which we term *mixed-example data augmentation*, consisting of mixing two different examples together. Unlike traditional data augmentation, which uses label-preserving transformations (*e.g.* horizontally flipping an image of a cat produces an image still containing a cat), mixed-example data augmentation is non-label preserving, since mixing an image of a cat together with an image of a dog produces an image that is neither entirely of a cat nor entirely of a dog. The prevailing hypothesis prior to our work for this approach’s efficacy was that mixed-example data augmentation, previously done only with linear combinations of pairs of examples, imposed a linear inductive bias on the final trained model, and that this linear inductive bias was the main benefit of mixed-example data augmentation. In our work, we tested this hypothesis by exploring a new, more generalized form of mixed-example data augmentation. By considering this broader scope, we found a much larger space of practical augmentation techniques, including methods that improve upon previous state-of-the-art. This generalization had benefits beyond improved performance, though, revealing a number of types of mixed-example data augmentation that are radically different from those considered in prior work. This provides evidence that existing theory for the effectiveness of such methods is incomplete and suggests that any such theory must explain a much broader phenomenon.

In Chapter 3 ([102]), we examine the end-to-end behavior of trained models. Specifically, while great progress has been made at making neural networks effective across a wide range of visual tasks, most models are surprisingly vulnerable. This frailness takes the form of small, carefully chosen perturbations of their input, known as adversarial examples [110], which represent a security threat for learned vision models in the wild. Our work examined a recent method for effective defense against adversarial examples, Adversarial Logit Pairing [54], and demonstrated that much of its effectiveness could in fact be attributed to a simpler phenomenon of logit regularization. Based on this, we showed how to extend the particular logit regularization technique used, thus improving its defense against two different types of adversarial example attacks, known as white-box and black-box attacks.

In Chapter 4 (published as [104] in ICLR 2020, an A-ranked conference), we investigated a particular type of layer used in nearly all neural network architectures, known as a *normalization layer*. Specifically, the normalization layer that we examine is Batch Normalization [51], used extremely commonly in computer vision. However, Despite its common use and large utility in optimizing deep architectures, it has been challenging both to generically improve upon Batch Normalization and to understand the circumstances that lend themselves to other enhancements. In our work, we identified 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 included proposing a method for reasoning about the current example in inference normalization statistics, fixing a training vs. inference discrepancy; recognizing and validating the powerful regularization effect of Ghost Batch Normalization [42] 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 [121].

Finally, in Chapter 5 (published as [105] in ICML 2021, an A-ranked conference), we considered the role of nondeterminism in the optimization process of neural networks. A topic that affects nearly all experiments in deep learning, nondeterminism in neural network optimization is due to various stochastic components of optimization that are typically not controlled (*e.g.* random weight initializations or random shuffling of training data), which produces uncertainty in performance and makes small improvements difficult to discern from run-to-run variability. While uncertainty can be reduced by training multiple model copies, doing so is time-consuming, costly, and harms reproducibility. In our work, we established an experimental protocol for understanding the effect of optimization nondeterminism on model diversity, allowing us to isolate the effects of a variety of sources of nondeterminism.

Surprisingly, we found that all sources of nondeterminism have similar effects on measures of model diversity. To explain this intriguing fact, we identified the instability of model training, taken as an end-to-end procedure, as the key determinant, showing that even one-bit changes in initial parameters result in models converging to vastly different values. We also proposed two approaches for reducing the effects of instability on run-to-run variability.

Put together, the chapters contained in this thesis span the full range of topics common across deep learning, ranging from data augmentation (Chapter 2) to model architecture (Chapter 4), optimization (Chapter 5), and the robustness of final trained models (Chapter 3). We hope that the research contributions contained herein continue to inspire further research and spark interesting ideas in the field of deep learning.

2. Mixed-Example Data Augmentation

We begin by considering a key component in most deep learning applications, data augmentation. More specifically, we develop an understanding of a new and interesting variant of data augmentation in which different examples are mixed together, termed *mixed-example data augmentation*. The prevailing hypothesis for the approach’s efficacy is that it biases neural networks towards linearity in predictions. Our work is focused on examining this hypothesis, showing linearity is unnecessary for effective mixed-example data augmentation. In the process, we introduce a number of non-linear methods which improve upon the previous state-of-the-art mixed-example methods.

2.1 Introduction

Deep neural networks have demonstrated remarkable performance on many tasks previously considered intractable, but they come with a cost: they require a large amount of data. While great progress has been made at making neural nets more data-efficient through the use of improved network architectures and training methods, the limitation remains with the greatest effect in data-starved domains such as robotics [65, 77] and medical applications [17, 31, 30].

To get around the requirement for large amounts of data, machine learning practitioners typically either transfer knowledge from other tasks, or employ large amounts of “data augmentation”, the practice of generating synthetic data based on real examples. This synthetic data artificially expands the size of datasets used for training models by many orders of magnitude, and typically takes the form of transformations that maintain the realistic appearance of the input, *e.g.* for image classification, flipping an image horizontally or making minor adjustments to its brightness.

A handful of recent work, however, has pointed out a new direction: data augmentation can take

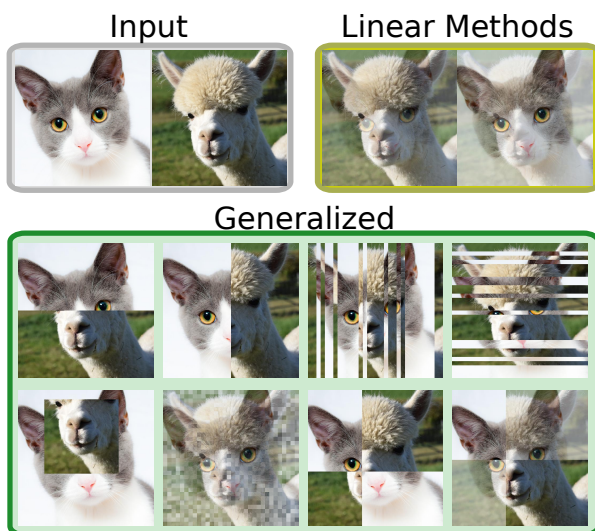


Figure 2.1: Given two input examples, mixed-example data augmentation consists of the set of functions that combine the inputs into a novel image, perhaps with an appearance unrealistic to humans. Previous work has considered the case of linear transformations, *i.e.* element-wise weighted averaging. In this chapter, we explore a more generalized space of functions in order to determine whether non-linear functions are similarly effective. Shown above are examples of 8 of the methods considered, all of which result in improved performance over non-mixed example augmentation, illustrating that the space of viable functions is much broader than previously realized.

the form of mixing training examples together, specifically via element-wise averages of pairs of inputs [124, 111, 112, 49]. Though this type of data augmentation does not produce data realistic to humans, it is surprisingly effective at training models, significantly improving performance across a variety of tasks and domains even *after* other forms of data augmentation are considered. Due to the nascency of such techniques, we currently do not have a good understanding of why they are effective. One hypothesis comes from Zhang et al. [124], who suggest that the added linearity this form of data augmentation encourages is a useful inductive bias, *i.e.* a bias that helps models in generalizing to unseen data. Another is from Tokozume et al. [111], who put forth the idea that CNNs treat imagery as waveform data and that such data augmentation puts constraints on internal feature distributions. While the true modus operandi by which these methods are successful remains an area of active research, these works nevertheless point to new directions for reducing the dependence of modern deep neural networks on large quantities of labeled data.

In this chapter, we ask the question: is linearity critical to the success of these data augmentation

methods that combine multiple input examples? To answer this question, we explore this new area of data augmentation as generalized functions of multiple inputs, which we denote “mixed-example data augmentation”. We propose a variety of alternative methods for example generation, and surprisingly find that almost all methods result in improvements over models trained without any form of mixed-example data augmentation (see Fig. 2.1 for examples). As a byproduct of increasing this search space, we find multiple methods that improve upon existing work [124, 111]. Though we do not propose new theoretical reasons for the utility of mixed-example data augmentation, our experiments shed empirical light on existing hypotheses, providing evidence that they are incomplete.

The remainder of this chapter is organized as follows: In Sec. 2.2 we review related work in data augmentation and regularization more generally, and in Sec. 2.3 we present our exploration of mixed-example data augmentation, overviewing 14 alternatives. We present experiments on CIFAR-10, CIFAR-100, and Caltech-256 in Sec. 2.4, then conclude with a discussion in Sec. 2.5.

2.2 Related Work

Data Augmentation. Data augmentation is key to the success of most modern neural networks across nearly every domain. Here we limit our discussion of data augmentation to applications with images, the focus of this chapter.

Common forms of data augmentation include random crops, horizontal flipping, and color augmentation [59], which improve robustness to translation, reflection, and illumination, respectively. Occasionally, random scalings are also done [98] as well as random rotations and affine transformations, though these tend to get somewhat less use. One more recent form of data augmentation consists of zeroing out random parts of the image [14, 126], a structured form of input-layer dropout, which works surprisingly well on the CIFAR-10/100 datasets. Though they differ greatly in technique, all of these methods are a form of label-preserving data augmentation, *i.e.* they are designed to maintain the label of the transformed image, which requires a small amount of task-specific knowledge. For example, in the context of image classification, an image of a cat, even after horizontally flipping it and altering its brightness, is still recognizable as an image of a cat.

More directly relevant to our work is recent progress on generating images for training as linear combinations of other training images [124, 111, 49]. While we save a detailed overview for Sec. 2.3.2, these methods take the general form of randomly generating a mixing coefficient and producing a new

image and label as a convex combination of two images/labels. Tokozume et al. [111] further considered an improved form based on the interpretation of images as waveform data, and Inoue restricted the sampling coefficient to 0.5. Our research, inspired by these, considers more generalized functions of a different form, showing that the space of effective mixed-example data augmentation is much more broad than linearity-based methods.

Regularization. Closely related to the topic of data augmentation is regularization. One of the most common approaches to regularization is weight decay [60], which is equivalent to L_2 -regularization when using a vanilla stochastic gradient descent learning rule. Other common types of regularization include Dropout [101], which can also be considered a form of feature-space data augmentation injected at intermediate layers in a neural network, and Batch Normalization [51], which has a regularization effect due to randomness in minibatch statistics. Other more exotic forms of regularization include randomly dropping out layers [46] and introducing disparities between forward- and backward-propagation [23]. Data augmentation and regularization can both be viewed as ways to incorporate prior knowledge into models, either via invariances in data (label-preserving data augmentation) or through priors on how weights and activations in neural networks should behave (regularization), and both have the goal of reducing the train-test generalization gap. As such, it is folk wisdom that there is a tradeoff between the optimal amount of data augmentation and regularization to use — for example, Zhang et al [124] found that using *mixup* well required a 5x lower amount of weight decay than without *mixup* on CIFAR-10.

2.3 Methods

2.3.1 General Formulation

Most uses of label-preserving data augmentation can be represented by stochastic functions of the form

$$(\tilde{x}, \tilde{y}) = \tilde{f}(x, y) = (f(x), y). \tag{2.1}$$

For example, f may be a function randomly altering the brightness of its input, horizontally flipping it, or applying a projective transformation. Of key note, however, is that \tilde{f} is an identity function with respect to its second input.

In this chapter, we consider generalized methods for data augmentation of the form:

$$(\tilde{x}, \tilde{y}) = \tilde{f}(\{(x_i, y_i)\}_{i=1}^2) \quad (2.2)$$

That is, we consider arbitrary functions mapping two examples into a single new training example. This is a strict generalization of the form of augmentation in Eq. 2.1, which can be obtained by ignoring either one of the two inputs. In theory one could consider functions with $N > 2$ examples as input, but in initial experiments (also agreeing with [124]) we did not see improvement beyond $N = 2$, so we restrict our methods to this setting.

2.3.2 Linearity-Based Methods

In this section we present previous work [124, 111, 112, 49], which can be represented as special cases of Eq. 2.2 in which \tilde{f} is a linear combination of $(x_1, y_1), (x_2, y_2)$:

Mixup. In *mixup*[124], the augmentation function \tilde{f} is represented by:

$$\begin{aligned} \tilde{x} &= \lambda x_1 + (1 - \lambda)x_2 \\ \tilde{y} &= \lambda y_1 + (1 - \lambda)y_2 \end{aligned} \quad (2.3)$$

where $\lambda \sim \text{Beta}(\alpha, \alpha)$ for each pair of examples, with α a hyperparameter. For experiments on CIFAR-10 and CIFAR-100, Zhang et al. [124] used the value $\alpha = 1$, which results in a uniform distribution between 0 and 1, and found that on larger datasets such as ImageNet [86] a smaller value of α was required due to underfitting. *Mixup* was motivated as encouraging linearity between training examples, with the hypothesis that linearity is an effective inductive bias (an assumption built into a model) for most models. Indeed, *mixup* was shown to be useful across a wide variety of tasks and models. As we shall show in our work, this picture is incomplete — linearity, while useful, is not required for mixed-example data augmentation to be effective, and extremely non-linear methods can perform nearly as well.

Between Class (BC+). Tokozume *et al.* [111] developed two methods for mixed-example data augmentation. The first, “BC” (Between-Class), is equivalent to *mixup* and was developed in parallel

with Zhang *et al.* [124]. Improving upon “BC” and building upon their previous work with audio [112], Tokozume *et al.* developed “BC+”, which is based on the intuition that neural networks (specifically CNNs) can treat imagery as waveform data. Using this intuition, two improvements were made:

First, waveforms are naturally zero-mean signals, while images are not. Therefore, Tokozume *et al.* first subtract each image’s mean (computed across all channels, *i.e.* a single number per image) from itself before further processing. This stands in contrast to most recent work with CNNs [123], which uses a mean across an entire dataset for normalization.

The second improvement comes from noting that combining examples in a strictly linear fashion does not produce a *perceptually* linear combination of images, a fact which was an acute concern in their prior work on audio [112]. To solve this, Tokozume *et al.* use the standard deviation of each image σ_1, σ_2 to measure “energy”, with the intuition that more variable images are more perceptually salient, and ultimately derive the mixing equation

$$\tilde{x} = \frac{p(x_1 - \mu_1) + (1 - p)(x_2 - \mu_2)}{\sqrt{p^2 + (1 - p)^2}} \tag{2.4}$$

$$\text{where } p = \frac{1}{1 + \frac{\sigma_1}{\sigma_2} \cdot \frac{1-\lambda}{\lambda}}$$

with $\lambda \sim U[0, 1]$ and the label being determined via the mixing coefficient λ as in *mixup*. While BC+ is technically a non-linear method, we group it together with linearity-based methods such as *mixup* since the non-linearity only occurs in the normalization term and the method is still fundamentally based on element-wise averaging.

2.3.3 Non-Linear Methods

We now illustrate the generality of our formulation (Eq. 2.2) by presenting many different non-linear methods for mixed-example data augmentation. Although we present them as alternatives to linearity-based approaches, we note that most of these methods are largely orthogonal both to such approaches as well as to more traditional forms of data augmentation and can potentially be employed in combination with them. Illustrative examples are shown in Fig. 2.2.

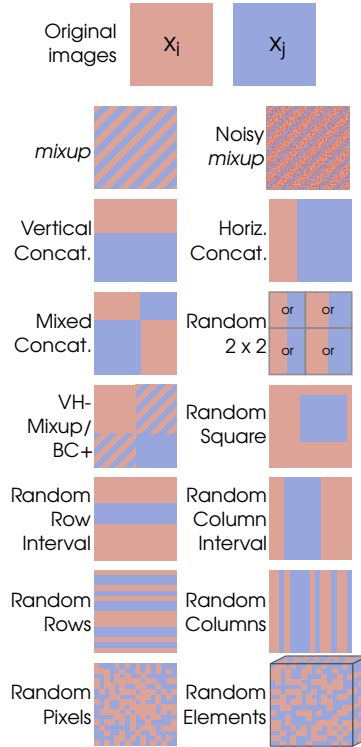


Figure 2.2: Example outputs for each mixed-example data augmentation method. See text (Sec. 2.3) for details of each method. Diagonal stripes are used to indicate element-wise weighted averaging. Note that for “VH-Mixup/VH-BC+”, the bottom-left and upper-right regions use different weights for the weighted average depending on λ_3 (see text).

Vertical Concat. As in *mixup* and similar to BC+, “Vertical Concat” begins by sampling a random mixing coefficient $\lambda \sim \text{Beta}(\alpha, \alpha)$. However, instead of element-wise averaging, in this method the top λ fraction of image x_1 is vertically concatenated with the bottom $(1 - \lambda)$ fraction of image x_2 . Formally, we have

$$\tilde{x}(r, c) = \begin{cases} x_1(r, c), & \text{if } r \leq \lfloor \lambda H \rfloor \\ x_2(r, c) & \text{otherwise} \end{cases} \quad (2.5)$$

where H is the height of the image and $x(r, c)$ denotes the 3-dimensional pixel at row r and column c of an image x . Though simple, this is an extremely non-linear transformation with respect to the input. The label \tilde{y} remains equal to the original labels weighted by the mixing coefficient: $\lambda y_1 + (1 - \lambda) y_2$.

Thus, a network trained with “Vertical Concat” must not only correctly classify the top and bottom portions of the image, but also correctly identify what fraction of the image they occupy.

Horizontal Concat. This method is similar to “Vertical Concat.”, but instead horizontally concatenates the left λ fraction of x_1 with the right $(1 - \lambda)$ fraction of x_2 :

$$\tilde{x}(r, c) = \begin{cases} x_1(r, c), & \text{if } c \leq \lfloor \lambda W \rfloor \\ x_2(r, c) & \text{otherwise} \end{cases} \quad (2.6)$$

where W is the width of the image. As before, we have $\tilde{y} = \lambda y_1 + (1 - \lambda)y_2$.

Mixed Concat. This is a combination of vertical and horizontal concatenation: first we sample $\lambda_1, \lambda_2 \sim \text{Beta}(\alpha, \alpha)$. Then we divide the output image in a 2×2 grid as shown in Fig. 2.2, where the horizontal boundary between grid members is determined by λ_1 and the vertical boundary is determined by λ_2 . The top-left and bottom-right portions of the output image are set to the corresponding pixel values in x_1 , and the top-right and bottom-left are set to x_2 , with $\tilde{y} = (\lambda_1\lambda_2 + (1 - \lambda_1)(1 - \lambda_2))y_1 + (\lambda_1(1 - \lambda_2) + (1 - \lambda_1)\lambda_2)y_2$, *i.e.* \tilde{y} is determined by the relative area of x_1 vs x_2 . Another interpretation of “Mixed Concat.” is an application of “Vertical Concat.” to two images produced by “Horizontal Concat.” with the same mixing coefficient but opposite argument order.

Random 2×2 . This method is a more randomized version of “Mixed Concat.”. First, this method divides the image into a 2×2 grid with random sizes as before, but instead of using a fixed assignment of grid cells to input images, “Random 2×2 ” randomly decides for each square in the grid whether it should take content from x_1 or x_2 . This prevents the network from relying on the fixed assignment in “Mixed Concat.”, forcing it to adapt to potentially changing positions of the input content. The target label \tilde{y} is measured as a function of the relative area of x_1 vs x_2 in the generated image.

One implementation detail that we have found to modestly help “Random 2×2 ” is a constraint on the 2×2 grid produced. Specifically, this constraint forces the intersection lines of the grid to occur in the middle p fraction of the image, preventing image content from becoming too long, narrow, or not even present. Though this constraint is not critical to the success of the method, we find that it tends to improve performance by a small but significant amount. In our experiments we set p to 0.5.

VH-Mixup. In order to explore whether it is possible to combine the strengths of non-linear methods with methods based on linearity, we introduce “VH-Mixup”, the goal of which is to leverage the advantages of “Vertical Concat.”, “Horizontal Concat.”, and *mixup* [124] (or equivalently BC [111]). First, two intermediate images are made as the result of “Vertical Concat.” and “Horizontal Concat.”, each with their own randomly chosen λ . Then, *mixup* is applied with these two images as input. This has the effect of producing an image where the top-left is from x_1 , the bottom-right is from x_2 , and the top-right and bottom-left are mixed between the two with different mixing coefficients. All together, with $\lambda_1, \lambda_2, \lambda_3 \sim \text{Beta}(\alpha, \alpha)$, we have that $\tilde{x}(r, c)$ is equal to:

$$\begin{aligned}
 x_1(r, c), & & \text{if } r \leq \lambda_1 H \wedge c \leq \lambda_2 W \\
 \lambda_3 x_1(r, c) + (1 - \lambda_3)x_2(r, c), & & \text{if } r \leq \lambda_1 H \wedge c > \lambda_2 W \\
 (1 - \lambda_3)x_1(r, c) + \lambda_3 x_2(r, c), & & \text{if } r > \lambda_1 H \wedge c \leq \lambda_2 W \\
 x_2(r, c) & & \text{if } r > \lambda_1 H \wedge c > \lambda_2 W
 \end{aligned} \tag{2.7}$$

The label \tilde{y} is determined in a straightforward manner based on the rules for label generation in “Vertical Concat.”, “Horizontal Concat.”, and *mixup*, and can be thought of as the expected fraction of its value a random pixel takes from x_1 vs x_2 .

VH-BC+. Rather than combining the outputs of “Vertical Concat.” and “Horizontal Concat.” with *mixup*, in “VH-BC+” they are combined with BC+. While it is tempting to think of this as simply replacing *mixup* with BC+, there is a subtle implementation detail: when to subtract the mean for each image. There are two options: directly before applying BC+, *i.e.* after performing “Vertical Concat.” and “Horizontal Concat.”, or before producing either of the concatenated images. We argue that the latter is correct — if an output of one of the concatenation methods is made into a zero-mean image, then it will have relatively little effect, as the image will still be clearly made of two distinct parts, even only based on first-order statistics (the mean). However, if the original two images are zero-meaned before either of the concatenation methods, then each concatenated image will still be zero mean in expectation, but the boundary between the two will no longer be as easily discerned. Indeed, we tested both methods in initial experiments, and while we found both to perform reasonably, ultimately the latter was slightly better on average.

Random Square. In this method, a random square within x_1 is replaced with a portion of x_2 . This method is inspired by Cutout [14] and random erasing data augmentation [126], but instead of

replacing the subimage with 0, we replace it with part of a different image. As in Cutout, the size of the square is a hyperparameter, which we set to 16 pixels. One subtle implementation choice with this method is which region in x_2 to use as a replacement — in our implementation, we use a portion of x_2 picked randomly among all regions of the appropriate size, which we found slightly better than using the region in x_2 directly corresponding to the replaced region of x_1 .

Random Column Interval. This method is a slight generalization of “Horizontal Concat.” — a random interval of columns is picked and that part of image x_1 is replaced with columns in x_2 . The difference between this method and “Horizontal Concat.” is that this column interval need not begin with the first column. The interval in this method is picked by sampling the lower bound of the interval uniformly, with the upper bound then sampled uniformly between all possible remaining upper bounds.

Random Row Interval. This method is identical to “Random Column Interval” but is applied to a random interval of rows instead of columns.

Random Rows. For each row in the output image \tilde{x} , this method randomly samples whether to take the row from x_1 or x_2 , where the probability of choosing the corresponding row in x_1 is given by λ . As before, \tilde{y} is determined based on the fraction of rows that were taken from x_1 compared with x_2 . One interpretation of this method is as a higher-frequency variant of “Vertical Concat.” in that rows are still taken either entirely from x_1 or x_2 , but with this method they may alternate between x_1 and x_2 , possibly many times, rather than being grouped into a single large block of rows.

Random Columns. This method is identical to “Random Rows” but samples columns instead of rows.

Random Pixels. This method is similar to “Random Rows” but samples each pixel separately. That is, after first sampling λ , a matrix of size $W \times H$ is created, consisting of numbers drawn uniformly from $[0, 1]$, and converted to a binary matrix via comparison with λ . This is interpreted as a boolean mask which can be element-wise multiplied with x_1 and x_2 in order to efficiently compute the output. The label \tilde{y} can also be easily determined using the expected value of the mask.

Random Elements. This method is similar to “Random Rows” but samples each element of the image separately. That is, when the image is represented as a $H \times W \times 3$ tensor (using RGB), each element in the tensor is randomly sampled from the corresponding value in either x_1 or x_2 . As with “Random Pixels”, this can be efficiently computed by using a $H \times W \times 3$ tensor of random numbers.

Noisy Mixup. Normally, in *mixup* [124], a single $\lambda \sim \text{Beta}(\alpha, \alpha)$ is sampled and then used across the entire image (or $\lambda \sim U[0, 1]$ for BC [111]). However, in order to produce an output with an expected label of $\lambda y_1 + (1 - \lambda)y_2$, there is no need for λ to be the same across an entire image – instead, as long as its expectation is the same, the same label applies. In this method, we first sample λ in the same fashion, but then for each pixel identified by a row r and column c we add random zero-centered noise to the mixing coefficient: $\lambda_{r,c} = \lambda + \ell_{r,c}$ where $\ell_{r,c} \sim N(0, \sigma^2)$, with σ^2 a hyperparameter that we set to $\sqrt{0.025} \approx 0.16$, indicating adding a small amount of pixel-wise data-dependent noise. It is also useful to constrain $\lambda_{r,c}$ to lie in the range $[0, 1]$, *i.e.* $\lambda_{r,c} = \max(\min(\lambda + \ell_{r,c}, 1), 0)$. Though this method is nearly linear, we find that it makes for an interesting comparison experimentally with the strictly linear methods *mixup* [124] and BC [111].

2.4 Experiments

2.4.1 Implementation Details

In all experiments, we perform mixed-example data augmentation by directly pairing together two examples at a time, rather than doing it on a batch-by-batch basis [124]. While this has the potential to slow down data processing due to twice as much I/O and non-vectorized operations, it is somewhat simpler to develop with, especially for the slightly more involved data generation methods. Furthermore, we found that we were still able to generate data fast enough for models to use via effective use of dataset caching, which keeps I/O to a minimum. Initial experiments furthermore suggested that this does not affect accuracy. We also found that it was crucial to do other types of data augmentation (*e.g.* random cropping and flips) before applying any type of mixed-example data augmentation, with differences in accuracy greater than 1% on CIFAR-10, but do not currently have an explanation for why this is important, an implementation detail we also found shared in all existing open-source code of prior work. All experiments were done on a desktop with two Nvidia Geforce GTX 1080 Ti GPUs. We now list dataset-specific implementation details.

CIFAR 10/100. We perform the bulk of our experiments on the CIFAR-10 and CIFAR-100 datasets [58], the primary test bed used by prior work [124, 111]. Following [124], we conduct our experiments using the pre-activation ResNet-18 [36], which we re-implemented in TensorFlow [1]. This network architecture has the advantage of having a relatively high accuracy (*e.g.* 5.4% error on CIFAR-10) while taking only 2 hours for a complete training run with any of the methods. Furthermore, previous work [124, 111] has already shown strong correlations in improvements across model architectures. We also note that this ResNet variant is slightly different from the official pre-activation ResNet-18, attaining somewhat higher accuracy.

Following both Zhang *et al.* [124] and Tokozume *et al.* [111], we use $\alpha = 1$ where applicable, which results in a uniform distribution for λ , though this parameter can in principle be tuned based on the extent of overfitting. On CIFAR-10 we use a weight decay of 10^{-4} for all mixed-example methods and $5 \cdot 10^{-4}$ for the baseline ResNet-18, and on CIFAR-100 we use a weight decay of $5 \cdot 10^{-4}$ for all methods on CIFAR-100, which we found necessary in order to reproduce prior work [124], making a difference of slightly more than 1% in final accuracy.

Our learning rate schedule follows [36], in which minibatches are of size 128, the learning rate starts at .01 for a warm-up period of 400 steps, increases to 0.1, then decays by a factor of 10 after 32,000, 48,000, and 70,000 steps. In practice, we noticed that this learning rate strategy can be somewhat unstable, with losses spiking up at the 400-step transition, after which models failed to recover well and ended up with a few percent lower accuracy than they would otherwise. Though the extent of this problem depended on the method, for reproducibility we have taken the practice of running 20 copies of each model for three epochs ($\approx 1,200$ steps) and then only continuing the three models with the lowest loss values, which tended to not observe dramatic spikes in loss. This both improved final performance and reduced training variability.

Caltech-256. For experiments on Caltech-256 [29] we used the Inception-v3 [109] architecture with the default “Inception” preprocessing, resulting in a 299×299 pixel image input to the model. We used the default weight decay for Inception models of $4 \cdot 10^{-5}$ and a batch size of 64 for all methods. For the baseline model we used a learning rate of .03, decayed by a factor of 10 when validation performance saturated, which occurred after 20,000 and 26,000 steps. The learning rate additionally had a warm-up period of 2,000 steps, during which time we increased it at a log-linear rate from $3 \cdot 10^{-5}$ to .03. All other methods had a similar warm-up phase and initial learning rate, but with learning rate decays by a factor of 10 after 45,000 and 57,000 steps, echoing the observation in prior work [124, 111]

Table 2.1: Experimental results on CIFAR-10. All numbers are the average across three training runs, measured at the final step of training, and methods are ordered by performance. Numbers for the baseline ResNet-18 model, *mixup*, and BC+ are from our TensorFlow re-implementation. Italicized method names and performances indicate methods which performed better than either existing state-of-the-art mixed-example method.

	Method	Error (%)
CIFAR-10	ResNet-18	5.4
	<i>mixup</i> [124]	4.3
	BC+[111]	4.2
	Rand. Elems.	6.2
	Rand. Pixels	5.7
	Rand. Col. Int.	5.1
	Rand. Cols	4.8
	Horiz. Concat.	4.7
	Rand. Rows	4.6
	Noisy Mixup	4.5
	Rand. Row. Int.	4.5
	Vert. Concat.	4.4
	Mixed. Concat.	4.4
	Rand. Square.	4.3
	<i>Rand. 2×2</i>	<i>4.1</i>
	<i>VH-BC+</i>	<i>3.8</i>
<i>VH-Mixup</i>	<i>3.8</i>	

that mixed-example data augmentation can take longer to train, particularly with larger models. For “BC+” and “VH-BC+”, we found it important to add a small constant when determining the standard deviation of each image in order to avoid numerical issues that made the loss diverge.

2.4.2 Results

CIFAR-10. CIFAR-10 [58] consists of 60,000 images of size 32×32 pixels, split evenly among 10 categories, with 50,000 training images and 10,000 test images, and is a standard test bed for training of small-scale deep learning models, having been used extensively in related work [124, 111]. Results on CIFAR-10 are shown in Table 2.1. A few trends are immediately apparent:

First, we examine the central question of our work: is linearity required for mixed-example data augmentation to be successful? Our experimental results answer this clearly: *linearity is not required*

for effective mixed-example data augmentation. Rather, the space of useful mixed-example data augmentation appears to be much larger than realized in previous work [124, 111, 49] — with the exception of “Rand. Pixels” and “Rand. Elems”, all other mixed-example techniques improved upon the baseline ResNet. Even the simplest of methods, “Horiz. Concat” and “Vert Concat”, improved upon the baseline significantly, and are perhaps the least similar to prior work of the methods considered.

While linearity may not be a requirement in order for a method to improve upon baseline performance, is it required among the most effective methods? Again, our results indicate that this need not be the case. While “VH-BC+” and “VH-Mixup” have some element of linearity in portions of the image, “Rand. 2×2 ”, despite containing no element-wise weighted averaging at all, was just as useful a form of data augmentation as *BC+* and *mixup*, even slightly outperforming them in the sample set of runs we conducted. While we agree with previous work that linearity on its own can be a fruitful inductive bias, it is by no means necessary.

Can we get the best of both worlds by combining the insights of linearity as an inductive bias with non-linear types of mixed-example data augmentation? Two of the methods we explored, “VH-Mixup” and “VH-BC+”, do just that, and in fact both were able to outperform all other approaches, setting a new state of the art for mixed-example data augmentation. This result is particularly promising due to the nascency of mixed-example approaches and the general applicability to a wide range of tasks (for the methods in this chapter, tasks in computer vision).

Last, in an effort to learn more about which aspects of such augmentation methods are useful, it is worth remarking on the methods that did not work well as negative examples. In particular, “Rand. Elems” and “Rand. Pixels” both worked worse than the baseline of doing no mixed-example data augmentation. These methods have a commonality: by treating every pixel differently, they exhibit the tendency to introduce high-frequency signals in the data, *i.e.* introducing a signal that varies rapidly from pixel to pixel. We hypothesize that this type of data augmentation makes it more difficult for models to capture local details within images, forcing them to rely more on low-frequency content and limiting their ability to properly learn from all available signals. In a similar vein, we also note that “Noisy Mixup”, although it worked reasonably well, was not even as effective as *mixup* without any modifications, which we also attribute to the addition of high-frequency content.

CIFAR-100. CIFAR-100 [58] is a 100-class companion of CIFAR-10 with otherwise similar properties. We present our results on CIFAR-100 in Table 2.2 Trends on CIFAR-100 were largely

Table 2.2: Experimental results on CIFAR-100. As in CIFAR-10, all numbers are the average across three training runs, measured at the final step of training, and methods are ordered by performance. Italics indicates better than existing state-of-the-art mixed-example methods.

	Method	Error (%)
CIFAR-100	ResNet-18	23.6
	<i>mixup</i> [124]	21.3
	BC+[111]	21.1
	Rand. Elems.	24.2
	Rand. Pixels	24.0
	Rand. Cols	22.4
	Noisy Mixup	21.8
	Horiz. Concat.	21.7
	Rand. Col. Int.	21.4
	<i>Rand. Square.</i>	20.9
	<i>Rand. Rows</i>	20.9
	<i>Mixed. Concat.</i>	20.9
	<i>Vert. Concat.</i>	20.8
	<i>Rand. 2 × 2</i>	20.4
	<i>Rand. Row. Int.</i>	20.1
	<i>VH-BC+</i>	19.9
<i>VH-Mixup</i>	19.7	

similar to results on CIFAR-10, with the best and worst methods consistent, though the ordering in between changed somewhat. One clear difference, though, is that many more of our exploratory methods outperformed prior work on CIFAR-100 (8 for CIFAR-100 vs 3 for CIFAR-10). While we do not offer any compelling hypothesis for this change, it provides evidence that at least some minor differences in effectiveness between each mixed-example method are likely to be data-dependent.

A further point worth noting, shared across both CIFAR-10 and CIFAR-100, is that row-based methods performed better than their column-based counterparts: “Vert. Concat” outperformed “Horiz. Concat”, “Rand. Rows” outperformed “Rand. Cols”, and “Rand. Row. Int” improved upon “Rand. Col. Int”. This potentially indicates the importance of keeping horizontal information intact when doing data augmentation, a finding reminiscent of much older work in picking horizontally-shaped spatial pooling grids [62].

Table 2.3: Experimental results on Caltech-256. Results are determined by a single run of model training, with evaluation checkpoints picked based on maximum validation performance.

	Method	Error (%)
Caltech-256	Inception-v3	51.4
	<i>mixup</i> [124]	42.7
	BC+[111]	42.6
	VH-Mixup	43.7
	<i>VH-BC+</i>	40.3

Caltech-256. In order to test methods for mixed-example data augmentation on larger, more real-world images, we additionally evaluate on Caltech-256 [29], a dataset of 256 categories. Since there is no predefined training, validation, or test splits, we constructed splits by randomly taking 40 images from each category for training, 10 for validation, and 30 for testing, resulting in splits of size 10,240, 2560, and 7,680, respectively. While smaller than other datasets of large natural images, *e.g.* ImageNet [86], experiments are also much more tractable, taking roughly 10 hours on average when training from scratch, compared with an estimated 25 days to run a single ImageNet experiment, which is prohibitively long. For this set of experiments, we focused our analysis on the best-performing methods from CIFAR-10 and 100, “VH-Mixup” and “VH-BC+”, in addition to the three baselines. Results are presented in Table 2.3.

Most noticeably, we found that *all* mixed-example data augmentation methods were able to improve performance over the baseline Inception-v3 network. The effect is dramatic, with improvements of up to 10% in accuracy above baseline. This highlights the strength of mixed-example methods, particularly with high-capacity models, a finding that strengthens results from prior work [124, 111].

Within the set of mixed-example methods, though, ordering is less obvious — while it is clear that “VH-BC+” was particularly successful, the reason by which it was so much better than “VH-Mixup” remains mysterious. It is also worth noting that confidence intervals for these experiments are somewhat wide: a 95% confidence interval due to data sampling alone is roughly $\pm 1\%$ at current accuracy levels, and the true interval is likely larger due to additional run-to-run variance from random initialization and data processing. Despite these limitations in measurement, we see these results as highly encouraging for applied tasks where data may be limited and performance is critical.

2.5 Conclusion

In this chapter we explored the space of mixed-example data augmentation, in the process generalizing and improving upon recent work [124, 111, 49]. We sought to determine whether linearity was necessary in order for mixed-example data augmentation to be effective, and in the process of answering that question, found a surprisingly large spectrum of non-linear methods that resulted in improvements over models trained with standard augmentation methods. Our methods, though specific to image-based tasks, are straightforward to implement and do not require any hyperparameter tuning beyond those in existing methods [124, 111]. Though we considered a variety of methods in this chapter, the field of mixed-example data augmentation is still in its early stages, and we postulate that it is likely even more effective methods exist. We hope that our explorations inspire further research in the area.

Key questions for future research include developing an understanding for *why* mixed-example data augmentation works and determining which specific properties of such augmentation methods are useful. We have shown that it is possible to combine the strengths of multiple approaches, but it remains unclear what the limits of mixed-example data augmentation are. On a lower level, it would also be interesting to understand the relationship between mixed-example data augmentation and other more traditional forms of data augmentation. For example, we found the puzzling behavior that mixed-example data augmentation is only effective when performed after other forms of data augmentation, and even this simple detail eludes current understanding.

One disadvantage of our approach is that, unlike some prior work [124, 111], our methods operate only on images. While this is true, we believe it is likely that domain-specific approaches such as ours can be made for other problems, such as speech [39] or natural language processing [9]. Furthermore, we believe that such approaches are actually most important for domain-specific tasks, such as robotics [65], which also tend to be the most data-starved and in need of improved methodology and further research.

3. Logit Regularization for Adversarial Robustness

We now turn our attention to the end-to-end behavior of trained models, examining the worst-case performance of neural networks via small perturbations of their input, known as *adversarial examples* [110]. In this chapter, we investigate a recent method for defending against adversarial examples, and demonstrate that much of its benefit can be attributed to a simpler phenomenon, logit regularization. Using this understanding, we extend the logit regularization technique used, improving its effectiveness against two different types of adversarial example attacks.

3.1 Introduction

Neural networks, despite their high performance on a variety of tasks, can be brittle. Given data intentionally chosen to trick them, many deep learning models suffer extremely low performance. This type of data, commonly referred to as *adversarial examples*, represent a security threat to any machine learning system where an attacker has the ability to choose data input to a model, potentially allowing the attacker to control a model's behavior.

At the same time, applications of computer vision are pervasive, with future applications including autonomous driving and medical diagnostics. While these use cases of vision are exciting in their potential for societal good, they also have the potential to be grave threats when behaving erroneously, with undesired behavior able to cause harm to both their users and creators. It is therefore of critical importance to understand how to defend against such adversarial attacks, both to prevent these systems from failing and to prevent malicious actors from exploiting any vulnerabilities they may have. Though

challenging, this is nonetheless an urgent need, as it is already known that attacks targeting systems for autonomous driving and medical diagnostics are possible [18, 19].

Today, adversarial examples are typically created by small, carefully chosen transformations of data that models otherwise have high performance on. While this is primarily due to the ease of experimentation with existing datasets [24], the full threat of adversarial examples is indeed only limited by the ability and creativity of an attacker’s example generation process – for example, even relatively basic research has shown the potential for adversarial attacks in the physical world [61], with more attacks being found on a regular basis.

Even with the limited threat models considered in current research, performance on adversarially chosen examples can be dramatically worse than unperturbed data. One canonical example is the CIFAR-10 image classification task [58], where white-box accuracy (accuracy when an adversary has access to the entire model structure and weights) on adversarially chosen examples is lower than 50%, even for the most robust defenses known today [73, 54], while unperturbed accuracy can be as high as 98.5% [10], a $30\times$ difference in misclassification rate. On larger tasks, such as ImageNet [86], the difference is even bigger, as no model is known to be robust to any but the weakest of all adversarial attacks.

Current defenses against adversarial examples generally come in one of a few flavors. Perhaps the most common approach is to generate adversarial examples as part of the training procedure and explicitly train on them, known as “adversarial training”. Another approach is to transform the model’s input representation in a way that thwarts an attacker’s adversarial example construction mechanism. While these methods can be effective, care must be taken to make sure that they are not merely obfuscating gradients [3]. Last, generative models can be built to model the original data distribution, recognizing when the input data is out of sample and potentially correcting it [100, 88]. Of these, arguably the most robust defenses today follow the adversarial training paradigm, of which adversarial logit pairing [54] is the most recent incarnation, extending the adversarial training work of Madry *et al.* [73] by incorporating an additional loss term to make the logits (the pre-softmax values that are the result of the final fully-connected layer in a network) of an unperturbed and adversarial example more similar. This similarity provides an additional training signal to encourage model predictions on adversarial examples to not differ much from the original examples, decreasing the negative effect of adversarial examples.

In this chapter, we show that adversarial logit pairing derives a large fraction of its benefits from regularizing the model’s logits toward zero, which we demonstrate through simple and easy to understand theoretical arguments in addition to empirical demonstration. Investigating this phenomenon

further, we examine two alternatives for logit regularization, finding that both result in improved robustness to adversarial examples, sometimes surprisingly so – for example, using the right amount of label smoothing [109] can result in greater than 40% robustness to a 10-step projected gradient descent (PGD) attack [73] on CIFAR-10 while training only on the original, unperturbed training examples, and is also a compelling black-box defense. We then present an alternative formulation of adversarial logit pairing that separates the logit pairing and logit regularization effects, improving the defense. The end result of these investigations is a defense that outperforms state-of-the-art approaches for PGD-based adversaries on CIFAR-10 for both white-box and black-box attacks, while requiring little to no computational overhead on top of adversarial training.

3.2 Overview of Adversarial Training

Before proceeding with our analysis, we review existing work on adversarial training for context. While adversarial examples have been examined in the machine learning community in some capacity for many years [12], their study has drawn a sharp focus in the current renaissance of deep learning, starting with Szegedy *et al.* [110] and Goodfellow *et al.* [26], particularly in the context of computer vision. In Goodfellow *et al.* [26], adversarial training is presented as training with a weighted loss between an original and adversarial example, *i.e.* with a loss of

$$\tilde{J}(\theta, x, y) = \frac{1}{m} \sum_{i=1}^m \alpha J(\theta, x^{(i)}, y^{(i)}) + (1 - \alpha) J(\theta, g(x^{(i)}), y^{(i)}) \quad (3.1)$$

where $g(x)$ is a function representing the adversarial example generation process, originally presented as $g(x) = x + \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y))$, α is a weighting term between the original and adversarial examples typically set to 0.5, θ are the model parameters to learn, J is a cross-entropy loss, m is the dataset size, $x^{(i)}$ is the i th input example, and $y^{(i)}$ is its label. Due to the use of a single signed gradient with respect to the input example, this method was termed the “fast gradient sign method” (FGSM), requiring a single additional forward and backward pass of the network to create. Kurakin *et al.* [61] extended FGSM into a multi-step attack, iteratively adjusting the perturbation applied to the input example through several rounds of FGSM. This was also the first attack that could be described

as a variant of projected gradient descent (PGD), where the adversarial perturbation is initialized to zero. Both of these approaches primarily target an L^∞ threat model, where the L^∞ norm between the original and adversarial example is constrained to a small value. By keeping the L^∞ norm small, it is assumed that the adversarial example will have the same correct label as the original example, *i.e.* that the perturbation is small enough to still be easily recognizable as the original category, an assumption that allows for research in the field without requiring the manual annotation of every new adversarial example.

Madry *et al.* [73] built upon these works by initializing the search process for the adversarial perturbation randomly, and is among the strongest attacks currently available. Although only a slight modification of Kurakin *et al.* [61], this detail is critical – with a zero initialization it is easy to become robust only at existing training points, thus causing a “gradient masking” effect [3]. Through extensive experiments, they showed that even performing PGD with a single random initialization is able to approximate the strongest adversary found with current first-order methods, and doing adversarial training with this attack resulted in the most robust model yet. However, as with multi-step FGSM, performing adversarial training with this approach can be rather expensive, taking an order of magnitude longer than standard training. Specifically, PGD-based adversarial training requires $N + 1$ forward and backward passes of the model, where N is the number of PGD iterations, and is typically on the order of 5 to 20 [73].

Improving on PGD-based adversarial training, Kannan *et al.* [54] introduced adversarial logit pairing (ALP), which adds a term to the adversarial training loss function that encourages the model to have similar logits for original and adversarial examples:

$$\begin{aligned} \tilde{J}(\theta, x, y) = \frac{1}{m} \sum_{i=1}^m \alpha J(\theta, x^{(i)}, y^{(i)}) + \\ (1 - \alpha) J(\theta, g(x^{(i)}), y^{(i)}) + \\ \lambda L(f(x^{(i)}; \theta), f(g(x^{(i)}); \theta)). \end{aligned} \tag{3.2}$$

where L was set to an L^2 loss and $f(x, \theta)$ returns the logits of the model corresponding to example x . Adversarial logit pairing has the motivation of increasing the amount of structure given to the model in the learning process by encouraging the model to have similar prediction patterns on the original and adversarial examples, a process reminiscent of distillation [40].

Kannan *et al.* [54] also studied a baseline version of ALP, called “clean logit pairing”, which paired randomly chosen unperturbed examples together. Surprisingly, this worked reasonably well, inspiring them to experiment with a similar idea they call “clean logit squeezing”, regularizing the L^2 norm of the model’s logits, which worked even more effectively, though this idea itself was not combined with adversarial training. It is this aspect of the work that is most related to what we study in this chapter.

Last, it is worth noting work examining the reproducibility of adversarial logit pairing [15]. In [15], ALP was found to not actually be robust on ImageNet to multi-step white-box attacks with a large number of iterations, continuing the trend of no models being robust on ImageNet. However, the improved robustness of ALP on smaller datasets that are more commonly used was not refuted – we also find this to be the case, providing further experimental evidence with attacks of up to 1,000 steps. Thus, we believe that ALP shows some promise in advancing our understanding and effectiveness in defending against adversarial examples.

3.3 Adversarial Logit Pairing and Logit Regularization

We now show how adversarial logit pairing [54] acts as a logit regularizer. For notational convenience, denote $\ell_c^{(i)}$ as the logit of the model for class c on example i in its original, unperturbed form, and $\tilde{\ell}_c^{(i)}$ as the logit for the corresponding adversarial example. The logit pairing term in adversarial logit pairing is a simple L^2 loss:

$$L = \frac{1}{2}(\ell_c^{(i)} - \tilde{\ell}_c^{(i)})^2 \tag{3.3}$$

While it is obvious that minimizing this term will have the effect of making the original and adversarial logits more similar in some capacity, what precise effect does it have on the model during training? To examine the effect of such a loss in gradient-based training, the dominant training paradigm for almost all computer vision models today, we can look at the gradient of this loss term with respect to the logits themselves:

$$\frac{\partial L}{\partial \ell_c^{(i)}} = \ell_c^{(i)} - \tilde{\ell}_c^{(i)} \tag{3.4}$$

$$\frac{\partial L}{\partial \tilde{\ell}_c^{(i)}} = \tilde{\ell}_c^{(i)} - \ell_c^{(i)} \tag{3.5}$$

Under the assumption that the adversarial example moves the model’s predictions away from the correct label (as should be the case with any reasonable adversarial example, such as an untargeted PGD-based attack), we will have that $\ell_c^{(i)} > \tilde{\ell}_c^{(i)}$ when $c = y^{(i)}$ is the correct category, and $\ell_c^{(i)} < \tilde{\ell}_c^{(i)}$ otherwise¹. Keeping in mind that model updates move in the direction opposite of the gradient, then the update to the model’s weights will attempt to make the original logits smaller and the adversarial logits larger when $c = y^{(i)}$ and will otherwise attempt to make the original logits larger and the adversarial logits smaller.

However, it is not sufficient to examine this in isolation, as logit pairing is only one component of a typical adversarial loss: it must be considered in the context of the adversarial training loss \tilde{J} – in particular, the cross-entropy loss used in \tilde{J} for the adversarial example $g(x^{(i)})$ already encourages the adversarial logits to be higher for the correct category and smaller for all incorrect categories, and furthermore the scale of the loss \tilde{J} typically is an order of magnitude larger than the adversarial pairing loss. Thus, we argue that the main effect of adversarial logit pairing is actually in the remaining two types of updates, encouraging the logits of the original example to be smaller for the correct category and larger for all incorrect categories – an effect which is essentially regularizing model logits in a manner similar to “logit squeezing” [54] or label smoothing [109].

Examining this further, we can also take a different perspective by explicitly incorporating the *scale* of the logits in the logit pairing term. If we factor out a shared scale factor γ from each logit, the logit pairing term has the form

$$L = \frac{1}{2}(\gamma\ell_c^{(i)} - \gamma\tilde{\ell}_c^{(i)})^2 \tag{3.6}$$

implying that

$$\frac{\partial L}{\partial \gamma} = \gamma(\ell_c^{(i)} - \tilde{\ell}_c^{(i)})^2, \tag{3.7}$$

Since $(\ell_c^{(i)} - \tilde{\ell}_c^{(i)})^2$ is non-negative, this means that the model will always attempt to update the scale γ of the logits in the opposite direction of its sign, which is necessarily an update moving γ toward zero so long as the logits were not identical – in fact, if this were the only term in the loss, then it is easy to see that $\gamma = 0$ would be a global minimizer of the loss. However, in practice this effect is partially counterbalanced by the adversarial training term, which requires that logits across different categories be different in order to minimize its cross-entropy loss.

¹While it is theoretically possible that the logits for an incorrect category are smaller in a loss-maximizing adversarial example, increasing their value, all else equal, will result in a higher loss, so $\ell_c^{(i)} < \tilde{\ell}_c^{(i)}$ will typically hold.

Given this interpretation, in this chapter we now explore four key questions: 1) Experimental verification of our analysis. In practice, how much of a logit regularization effect does ALP have? 2) Do other forms of logit regularization have similar effects on adversarial robustness? If so, then an entire family of methods for improving adversarial robustness will have been found. 3) Is it possible to decouple adversarial logit pairing explicitly into a form where the effect of logit regularization and pairing can be disentangled? 4) Finally, using these insights, can we discover even more robust models?

3.3.1 Experimental Evidence

Perhaps the most straightforward way to test our hypothesis is to examine the logits of a model trained with ALP vs one trained with standard adversarial training. If our hypothesis is true, then the model trained with ALP will have logits that are generally smaller in magnitude. We present the results of this experiment in Figure 3.1(left), using an 18-layer ResNet [35] classifier trained on CIFAR-10 [58] as our experimental testbed (see Sec. 3.5.1 for details).

It is indeed the case that the logits for a model trained with ALP are of smaller magnitude than those of a model trained with PGD, with a variance reduction of the logits from 8.31 to 4.02 on the original clean (non-adversarial) test data². Though this provides evidence that ALP *does* have the effect of regularizing logits, this data alone is not sufficient to determine if logit regularization is a key mechanism in ALP’s improved adversarial robustness.

To answer this, we examine if standard adversarial training can be improved by explicitly regularizing the logits. If adversarial robustness can be improved, but similar improvements can *not* be made to ALP, then at least some of the benefits of ALP can be attributed strictly to logit regularization. We present the results of this experiment in Figure 3.1(right), implemented using the “logit squeezing” form of regularization (L^2 -regularization on the logits).

We find that incorporating regularization on model logits is able to recover slightly more than half of the total improvement from logit pairing, with a unimodal distribution – too little regularization has only a small effect, and too much regularization approaches the point of being harmful. In contrast, when added to a model already trained with ALP, regularizing the logits does not lead to any improvement at all, and in fact hurts performance at all levels of regularization strength, likely due to the combination of explicit logit regularization and the implicit regularization happening from ALP overpowering the

²The corresponding distributions on a set of PGD-based adversarial examples are very similar.

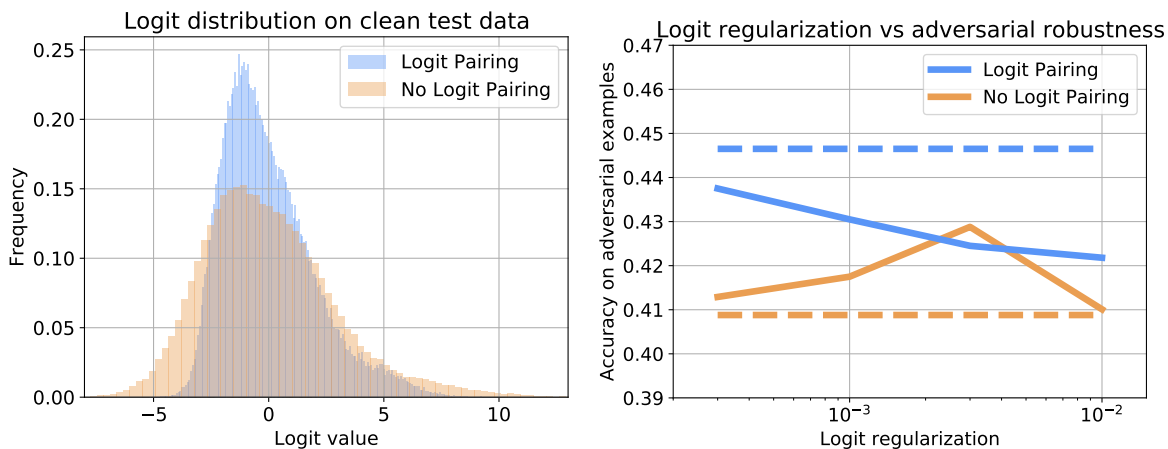


Figure 3.1: Left: Distribution of logits on clean test data for models trained with and without logit pairing. Right: Performance against a 10-step PGD attack for models trained with varying amounts of logit regularization, with and without logit pairing. Dashed lines indicate accuracy on adversarial examples without any logit regularization at all.

cross-entropy loss of adversarial training. This evidence makes clear that one of the key improvements from logit pairing is due to a logit regularization effect.

We would like to emphasize that these results are not meant to diminish ALP in any sense – our goals are to investigate the mechanism by which it works and explore if it can be generalized or improved. Thus, given these results, it is worth examining other methods that have an effect of regularizing logits in order to tell whether this is a more general phenomenon.

3.4 Other forms of logit regularization

Label Smoothing. Label smoothing is the process of replacing the one-hot training distribution of labels with a softer distribution, where the probability of the correct class has been smoothed out onto the incorrect classes [109]. Concretely, label smoothing uses the target distribution:

$$p_c^{(i)} = \begin{cases} 1 - s & c = y^{(i)} \\ \frac{s}{C-1} & c \neq y^{(i)} \end{cases} \quad (3.8)$$

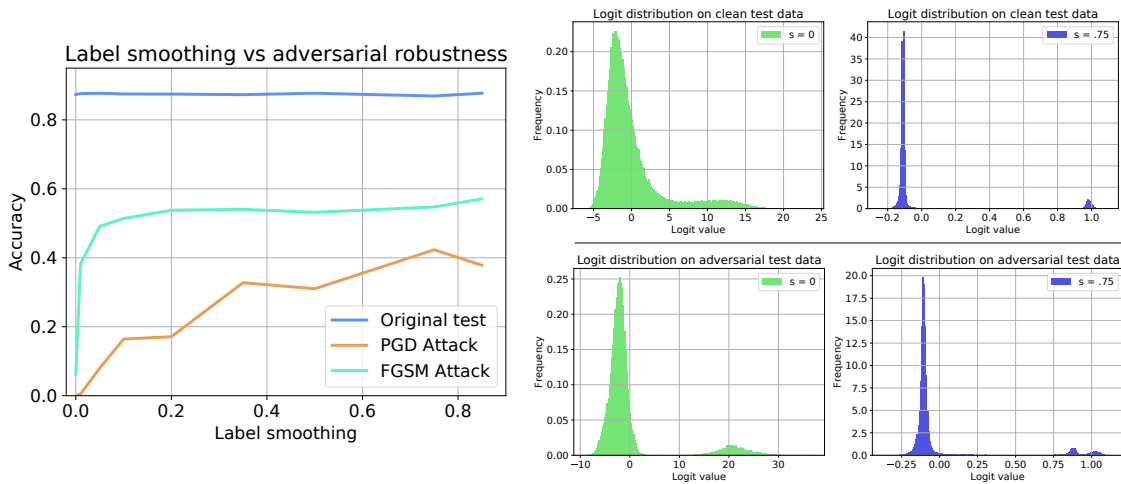


Figure 3.2: Left: Clean (“original test”) and adversarial accuracy on CIFAR-10 as a function of the label smoothing strength. Note that models for this figure were trained exclusively on the original training data – no adversarial examples were involved in the training procedure. Top-Right: Logit distribution of model trained with no label smoothing (middle) and with label smoothing of $s = .75$ (right), evaluated on the original test images, showing the regularization effect that label smoothing has on the logit distribution. Bottom-Right: Logit distributions when evaluated on PGD-based adversarial examples, showing how the regularization effect of label smoothing also applies to adversarial examples.

where $p_c^{(i)}$ is the target probability for class c for example i , the number of categories is denoted by C , and $s \in [0, 1 - \frac{1}{C}]$ is the smoothing strength. Label smoothing was originally introduced as a form of regularization, designed to prevent models from being too confident about training examples, and had the goal of improved generalization. Furthermore, it can be easily implemented as a preprocessing step on the labels, and does not affect model training time in any significant way. Interestingly, Kurakin *et al.* [61] found that incorporating a small amount of label smoothing present in a model trained on ImageNet actually *decreased* adversarial robustness roughly by 1%. Here we find a different effect.

In Figure 3.2(left) we show the effect label smoothing has on the performance of a model trained only on clean (*i.e.* non-adversarial) training data. Very surprisingly, models trained with label smoothing only (*i.e.* no other defenses against adversarial examples) were nearly as robust to this 10-step PGD attack as models trained with PGD-based adversarial training or adversarial logit pairing, both of which take an order of magnitude more time to train – though in fairness we note that when PGD and

ALP-based models are trained only on adversarial examples rather than a mixture of clean and adversarial data, their robustness exceeds this performance by around 5% (experiments not shown in figures). Furthermore, this benefit of label smoothing comes at no significant loss in accuracy on unperturbed test data, while generally adversarial training tends to trade off original vs adversarial performance. Another curiosity is that adding in any label smoothing at all dramatically improves robustness to FGSM-based adversaries (adding label smoothing of $s = .01$ brings accuracy up from 6.1% to 38.3%; experiment not shown in figures), while PGD-based attacks saw much more gradual improvement with label smoothing strength. While remarkable, this property of label smoothing on the loss surface eludes understanding, warranting further research.

Examining the logits themselves (Figure 3.2, right), we see a striking difference between the models – the model trained with label smoothing both has a dramatically smaller dynamic range of logits – roughly 1.2 vs. 20, a 16-fold decrease – and also presents a much more bimodal logit distribution than the model trained without label smoothing. In other words, it has learned to predict extremely consistent values for logits, a property that may contribute to its adversarial robustness. We observed that this behavior held for all positive values of s , with a stronger effect the higher s was.

This behavior can be explained: when trained with no label smoothing, the cross-entropy loss used in most models encourages model output to be as close to a one-hot distribution as possible, predicting a probability of 1 for the correct category and 0 for all other categories. When viewed as logits instead of probabilities, this corresponds to pushing the logits for the correct and incorrect categories as far apart as possible. However, models trained with label smoothing are instead encouraged to produce a soft distribution, with no probabilities too close to either 0 or 1, corresponding to a bounded target logit difference which gets smaller with increasing s .

Additional experiments involving label smoothing are given in Section 3.5.

Mixed-Example Data Augmentation Recently, a new form of data augmentation was found that stands in contrast to standard label-preserving data augmentation. Mixed-example data augmentation consists of combining different training examples together, dramatically altering both the appearance of the training examples and their labels. Introduced concurrently by multiple groups [124, 49, 111], these types of data augmentation typically have the form of element-wise weighted averaging of two input examples (typically images), with the training label also determined as a weighted average of the original two training labels (represented as one-hot vectors). Besides making target labels soft (*i.e.* not 1-of- K) during training time, these methods also encourage models to behave linearly between

examples, which may improve robustness to out of sample data. Interestingly, Zhang *et al.* [124] found that this type of data augmentation improved robustness to FGSM-based attacks on ImageNet [86], but Kannan *et al.* [54] found that the method did not improve robustness against a targeted attack with a stronger PGD-based adversary.

Experimentally, we found evidence agreeing with both conclusions – when applying *mixup* [124], we found a sizeable increase in robustness to FGSM adversaries, going from 6.1% on CIFAR-10 by training without *mixup* to 30.8% with *mixup*, but did not observe a significant change when evaluated against a PGD-based adversary. While robustness to a PGD adversary with only 5 steps increased by a tiny amount (from 0.0% to 0.5%), robustness to a 10-step PGD adversary remained at 0%. In our experiments, we use *VH-mixup*, the slightly improved version of *mixup* introduced by Summers and Dinneen [103].

3.4.1 Decoupling Adversarial Logit Pairing

We have now considered alternate methods by which logits can be regularized. Now, we examine how they interact with the logit regularization effect of adversarial logit pairing. Doing so requires decoupling the logit pairing and logit regularization effects of ALP.

In adversarial logit pairing [54], the logit pairing term is implemented as an L^2 loss:

$$L(f(x^{(i)}; \theta), f(g(x^{(i)}); \theta)) = \|f(x^{(i)}) - f(g(x^{(i)}))\|^2, \quad (3.9)$$

though other losses such as an L^1 or Huber loss are also possible. Expanding this creates a form that makes the pairing and regularization terms evident:

$$\begin{aligned} L(f(x^{(i)}; \theta), f(g(x^{(i)}); \theta)) = \\ \|f(x^{(i)})\|^2 - 2f(x^{(i)})^T f(g(x^{(i)})) + \|f(g(x^{(i)}))\|^2, \end{aligned} \quad (3.10)$$

where the first and third terms are explicit logit regularization terms on $f(x^{(i)})$ and $f(g(x^{(i)}))$, and the logit pairing effect is only determined by the middle inner product. While using an L^2 loss is a natural loss for regularization purposes, the pairing term can be improved by considering a more general form:

$$L(f(x^{(i)}; \theta), f(g(x^{(i)}); \theta)) = h(f(x^{(i)}), f(g(x^{(i)}))) + \beta(\|f(x^{(i)})\|^2 + \|f(g(x^{(i)}))\|^2), \quad (3.11)$$

where h has the express goal of making the logits more similar (with as little logit regularization as possible), and the regularization terms have been grouped with a controllable weighting factor. There are several natural choices for h , such as the Jensen-Shannon divergence, a cosine similarity, or any similarity metric that does not have a significant regularization effect. We have found that simply taking the cross entropy between the distributions induced by the logits was effective – depending on the actual values of the logits, this can either still have a mild squeezing effect (if the logits are very different), a mild expanding effect (if the logits are very similar), or something in between.

One implementation detail worth noting is that it can be difficult to reason about and set the relative strengths of the pairing loss and adversarial training loss. To that end, we set the strength of the pairing loss h as a constant fraction of the adversarial loss, implemented by setting the coefficient of the loss as a constant multiplied by a non-differentiable version of the ratio between the losses.

By decomposing adversarial logit pairing explicitly into logit pairing and logit regularization terms in this way, adversarial robustness to a 10-step PGD attack improves by an absolute 1.9% over ALP, or 5.6% over PGD-based adversarial training.

3.5 Additional experiments

In this section we present additional experiments on three datasets: The primary two datasets are CIFAR-10 and CIFAR-100, which are datasets for 10-way and 100-way classification, respectively, each with 50,000 examples, on which we evaluate both white-box and black-box adversarial attacks. White-box attacks are ones in which the adversary has full access to a model’s architecture and weights, while black-box attacks are ones in which this information is hidden to the adversary. Additionally, we evaluate on the Street View House Numbers (SVHN) dataset [79], which is significantly larger in the number of training examples (604,388) and whose 10 classes have an uneven distribution, with the most common class roughly 3 times more common than the least common class.

Table 3.1: White-box accuracy of models on CIFAR-10.

Training Method	Adversary				
	Natural	FGSM	PGD (5 steps)	PGD (10 steps)	PGD (20 steps)
Regular Training	87.4%	6.1%	0.0%	0.0%	0.0%
Label Smoothing	86.9%	54.7%	49.4%	41.7%	34.4%
PGD [73]	75.8%	50.5%	51.0%	46.1%	45.3%
ALP [54]	74.0%	52.6%	53.6%	49.1%	48.5%
LRM (ours)	68.5%	52.8%	53.8%	51.4%	51.0%

3.5.1 Implementation Details

In the experiments throughout this chapter on CIFAR-10/100, we used an 18-layer ResNet [35], equivalent to the “simple” model of Madry *et al.* [73], with a weight decay of $2 \cdot 10^{-4}$ and a momentum optimizer with strength of 0.9. Standard data augmentation of random crops and horizontal flips was used. After a warm up period of 5 epochs, the learning rate peaked at 0.1 and decayed by a factor of 10 at 100 and 150 epochs, training for a total of 200 epochs for models not trained on adversarial examples and 101 epochs for models using adversarial training – adversarial accuracy tends to increase for a brief period of time after a learning rate decay, then quickly drop by a small amount, an empirical finding also echoed by Schmidt *et al.* [91]. The minibatch size was 128.

Adversarial examples were constrained to a maximum L^∞ norm of .03, and all PGD-based attacks used a step size of 0.0078. For our implementation of adversarial logit pairing, on CIFAR-10 we used a pairing coefficient of 0.5 as recommended by [54], and found a larger coefficient of 5.0 more effective on CIFAR-100.

On SVHN, a smaller 8-layer ResNet was used for computational efficiency due to the scale of the dataset, which is a considerable challenge to overcome with adversarial training. Models were trained for 101 epochs, with a learning rate of .001 for the first 5 epochs, .01 until epoch 80, and .001 afterward. Data augmentation consisted of random crops after an initial 4 pixel padding. Adversarial attacks on SVHN were performed with an L^∞ constraint on the perturbation of 12/255 with a step size of 3/255. All adversarial attacks were constructed using the CleverHans library [82], implemented in TensorFlow [1], and all experiments were done on two Nvidia Geforce GTX 1080 Ti GPUs.

3.5.2 Towards a more robust model

Given these forms of logit regularization, perhaps the most natural question is whether they can be combined to create an even more robust model. Thus, in this section we focus exclusively on making a model (and comparable baselines) as robust as possible to PGD-based attacks. In particular, for baseline methods (PGD-based adversarial training [73] and adversarial logit pairing [54]), we opt to train exclusively on adversarial examples, effectively setting $\alpha = 0$ in Equation 3.1, which roughly trades off 4 – 5% accuracy for clean test examples for a similar gain in adversarial performance.

To combine the logit regularization methods together, on CIFAR-10 and CIFAR-100 we use a modest amount of label smoothing ($s = 0.1$) and use VH-mixup [103] on the input examples. For the logit pairing formulation presented in Section 3.4.1, we found different hyperparameters worked best on our two evaluation datasets. On CIFAR-10, we set $\beta = 10^{-3}$, and set the ratio between the adversarial training loss and the pairing loss to 0.125, which focuses the loss on keeping adversarial and original examples similar. On CIFAR-100, the more challenging dataset, we use $\beta = 3 \cdot 10^{-5}$ and a ratio of 0.95, which allows the network to focus more on fitting the data while still maintaining a balance with defending against adversarial examples. On SVHN, we set $s = 0.2$, use $\beta = 10^{-4}$, employ regular mixup [124] (which is more suitable for the imagery of SVHN), and use a ratio of 0.95, similar to CIFAR-100. We note that these parameters were not tuned much due to resource constraints. We refer to this combination simply as LRM (“Logit Regularization Methods”).

CIFAR-10 White-box performance on CIFAR-10 is shown in Table 3.1. LRM, achieves the highest level of adversarial robustness of the methods considered for all PGD-based attacks, and to the best of our knowledge represents the most robust method on CIFAR-10 to date. However, like other adversarial defenses, this comes at the cost of performance on the original test set, which makes sense – from the perspective of adversarial training, a clean test image is simply the center of the set of feasible adversarial examples. Nonetheless, it is interesting that the tradeoff between adversarial and non-adversarial performance can continue to be pushed further, with the optimal value of that tradeoff dependent on application, *i.e.* whether worst-case performance is more important than performance on the original unperturbed examples.

Next, black-box performance is shown in Table 3.2. As is standard in most black-box evaluations of adversarial defenses, this is performed by generating adversarial examples with one model (the “Source”) and evaluating them on a separate independently trained model (the “Target”). In this exper-

Table 3.2: Black-box accuracy of models on CIFAR-10.

Target	Source				
	Regular Training	Label Smoothing	PGD	ALP	LRM (ours)
Regular Training	26.8%	32.0%	75.1%	73.9%	67.3%
Label Smoothing	66.7%	67.3%	75.6%	74.2%	67.6%
PGD [73]	69.8%	69.1%	58.0%	57.9%	55.7%
ALP [54]	69.4%	68.8%	60.8%	59.3%	56.4%
LRM (ours)	71.5%	70.9%	60.5%	59.4%	54.3%

Table 3.3: White-box accuracy of models on CIFAR-100.

Training Method	Adversary				
	Natural	FGSM	PGD (5 steps)	PGD (10 steps)	PGD (20 steps)
Regular Training	59.1%	2.3%	0.1%	0.0%	0.0%
Label Smoothing	55.0%	11.5%	2.3%	0.9%	0.2%
PGD [73]	50.2%	26.2%	27.2%	23.9%	23.6%
ALP [54]	44.9%	27.3%	28.3%	26.0%	25.9%
LRM (ours)	43.8%	28.1%	29.1%	26.8%	26.6%

iment, we use a 10-step PGD attack to generate adversarial examples. As found in other works [73], the success of a black-box attack depends both on how similar the training procedure was between the source and target models and on the strength of the source model – for example, LRM, uniformly results in a stronger black-box attack than ALP [54], which itself is a uniformly stronger black-box attack than adversarial training with PGD [73]. As such, using LRM, as the source mildly damages the black-box defenses of PGD and ALP.

Interestingly, label smoothing was fairly effective as a black-box defense, being among the most robust models across all different sources. In particular, label smoothing had the highest minimum performance across sources (over 10% higher than any other method), which is particularly surprising given its near-zero cost compared to the adversarially-trained models.

CIFAR-100 White-box performance on CIFAR-100 is presented in Table 3.3. Again, we find that LRM achieves the highest level of adversarial robustness to all adversarial attacks, with ALP also strictly better than PGD-based adversarial training. Interesting, though, we find that label smoothing

completely fails to all attacks on CIFAR-100, behaving almost completely differently than on CIFAR-10. Although examining this was not the goal of our work, this does highlight the importance of evaluating proposed adversarial defenses on multiple datasets.

The corresponding results for black-box attacks on CIFAR-100 are shown in Table 3.4, where we again find clear differences across datasets. This time, the difference is that all methods perform much more similarly to one another, with the exception of clear differences of transferring from PGD-trained models to non-adversarially trained models.

SVHN We demonstrate performance against white-box attacks on SVHN in Table 3.5. Despite the large differences in dataset scale, image type, and imbalance in class frequencies, we again find that our method, LRM, is the most robust of all approaches on every adversarial attack, with patterns in the performance of each defense very similar to the other datasets, providing additional evidence that our methods and insights are generalizable.

3.5.3 Evaluating Stronger Attacks

Evaluating adversarial defenses is difficult to do correctly – since evaluating against any attack merely provides an upper bound on adversarial robustness, it is critical to evaluate on the strongest attacks available to make the bound as tight as possible. Furthermore, care must be taken to avoid gradient masking or obfuscated gradients [3], which can lead to a false sense of security.

Here we evaluate white-box performance on CIFAR-10 with two very strong attacks: a 1,000-step PGD adversary (the same attack that ALP succumbed to on ImageNet), and SPSA [113], a gradient-free attack that is effective at uncovering gradient masking. Results are given in Table 3.6. Note that SPSA is evaluated against a representative 1,000-image sample of the evaluation set for efficiency, since a full evaluation would take roughly 90 hours, and that we use the same evaluation settings as provided in [113].

We find nearly no difference when going from a 20-step to a 1,000-step PGD attack for all methods except for label smoothing, which loses most of its robustness. This suggests that label smoothing, while providing only a mild amount of worst-case adversarial robustness, can actually make the adversarial optimization problem much more challenging, which we believe is also the underlying reason for its effectiveness against black-box attacks. Based on this conjecture, we also evaluated label smoothing

Table 3.4: Black-box accuracy of models on CIFAR-100.

Target	Source				
	Regular Training	Label Smoothing	PGD	ALP	LRM (ours)
Regular Training	33.3%	31.7%	50.1%	44.7%	43.8%
Label Smoothing	37.0%	31.6%	50.0%	44.8%	43.9%
PGD [73]	37.6%	34.0%	33.4%	32.5%	32.4%
ALP [54]	39.7%	36.6%	33.6%	31.5%	32.1%
LRM (ours)	37.7%	33.4%	34.5%	33.0%	31.3%

Table 3.5: White-box accuracy of models on SVHN.

Training Method	Adversary				
	Natural	FGSM	PGD (5 steps)	PGD (10 steps)	PGD (20 steps)
Regular Training	96.7%	14.8%	0.0%	0.0%	0.0%
Label Smoothing	97.0%	50.8%	15.1%	4.4%	1.3%
PGD [73]	85.3%	50.5%	47.3%	39.6%	38.0%
ALP [54]	82.5%	49.6%	47.1%	40.0%	38.6%
LRM (ours)	83.6%	51.5%	48.4%	40.9%	39.4%

as a black-box defense with a 1,000-step PGD attack, where we have found a much smaller drop in performance, going from 67.3% to 60.8%, confirming that label smoothing still has its place in black-box defenses. The exact mechanism by which label smoothing makes the search for adversarial examples more difficult, however, remains elusive, which we think is an interesting avenue for further research.

On the other hand, an SPSA attack removes some of the difference in robustness between PGD, ALP, and LRM. While this illustrates that ALP and LRM are likely doing some type of gradient masking in a way that PGD cannot detect, even with 1,000 iterations, it also illustrates that there is a real gain in adversarial robustness even when considering strong gradient-free attacks.

3.6 Conclusion

In this chapter, we have shown the usefulness of logit regularization for improving the robustness of neural models of computer vision to adversarial examples. We first presented an analysis of adversarial logit pairing, showing that roughly half of its improvement over adversarial training can be attributed

Table 3.6: Evaluating models against the strongest white-box attacks on CIFAR-10. SPSA is evaluated on a 1,000-image (10%) subsample, and a 20-step PGD attack is provided for context.

Training Method	Adversary		
	PGD (20 steps)	PGD (1,000 steps)	SPSA [113]
Regular Training	0.0%	0.0%	0.0%
Label Smoothing	34.4%	7.2%	8.9%
PGD [73]	45.3%	45.2%	45.4%
ALP [54]	48.5%	48.3%	46.1%
LRM (ours)	51.0%	51.1%	47.9%

to a non-obvious logit regularization effect. Based on this, we investigated two other forms of logit regularization, demonstrating the benefits of both, and then presented an alternative method for adversarial logit pairing that more cleanly decouples the logit pairing and logit regularization effects while also improving performance.

By combining these logit regularization techniques together, we were able to create both a stronger defense against white-box PGD-based attacks and also a stronger attack against PGD-based defenses, both of which come at almost no additional cost to PGD-based adversarial training. We also demonstrated the surprising strength of label smoothing as a black-box defense and its paradoxical weakness to highly-optimized white-box attacks.

We anticipate that future work will push the limits of logit regularization even further to improve defenses against adversarial examples, possibly drawing on techniques originally devised for other purposes [84]. We also hope that these investigations will yield insights into training adversarially-robust models without the overhead of multi-step adversarial training, an obstacle that has made it challenge to scale up adversarial defenses to larger datasets without very large computational budgets.

4. Improving Batch Normalization

We now focus our attention on a different key component of modern deep learning, normalization layers. In this chapter we predominantly study Batch Normalization [51], used extremely commonly in computer vision. As in previous chapters, our goal is to advance the understanding of a key topic in deep learning (Batch Normalization) and use the understanding to improve the technique itself. Here, we identify four improvements to the generic form of Batch Normalization and the circumstances under which they work, yielding performance gains across a wide range of settings while requiring no additional computation during training. Most notably this includes a previously unstudied discrepancy between its training and inference behavior.

4.1 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 [58, 33, 65, 106, 28]. A key component of many neural networks is the use of normalization layers such as Batch Normalization [51], Group Normalization [121], or Layer Normalization [4], 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 [90], 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 [98, 125], custom nonlinearities [56], or other more complicated techniques [122]. Even in situations where training without normalization layers is possible, their usage can still aid generalization [125]. 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 chapter we identify four techniques 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, a technique designed for very large-batch multi-GPU training [42], 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-batch 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.

4.2 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:

$$\hat{x}_i = \gamma \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta \quad (4.1)$$

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 [51], the pioneering work in normalization layers, defined μ_i and σ_i^2 to be 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, because it is desirable to make inference behavior independent of inference batch statistics (so that the output of a model during inference time for a given example does not depend on which other examples are in the inference batch), the mean and variance are replaced with a moving average of the mean and variance observed during training time. This is typically done using an exponential moving average, *e.g.* $\bar{\mu}_i^{\text{new}} = 0.9999\bar{\mu}_i^{\text{old}} + 0.0001\mu_i$, where $\bar{\mu}_i$ is the moving average of the mean and μ_i is the mean for the current training batch. The effectiveness of Batch Normalization is undeniable, playing a key role in nearly all state-of-the-art convolutional neural networks since its discovery [109, 107, 35, 36, 127, 128, 44, 43, 89]. 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 [51], recent work has instead proposed that its true effectiveness stems from making the optimization landscape smoother [90].

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 [4], which has found use in many natural language processing tasks [116], 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 [114], 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 [121] 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 [121].

Besides these most common approaches, many other forms of normalization also exist: Weight Normalization [87] 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. Decorrelated Batch Normalization [47] performs ZCA whitening [58] in its normalization layer, and Iterative Normalization [48] makes it more efficient via a Newton iteration approach. Cho and Lee analyze the weights in Batch Normalization from the perspective of a Riemannian manifold, yielding new optimization and regularization methods that utilize the manifold’s geometry.

Targeting the small batch problem, Batch Renormalization [50] uses 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. Guo *et al.* tackle the small batch setting by aggregating normalization statistics over multiple forward passes.

Recently, Switchable Normalization [68] 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 differentially, 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. This results in Switchable Normalization obtaining somewhat worse performance than Group Normalization for small batch sizes. Differentiable Dynamic Normalization [69] fixes the latter point, learning an even more flexible normalization layer. Beyond these, there are many approaches we omit for lack of space [66, 13, 41, 56, 122, 125].

4.3 Improving Normalization

In this section we detail four methods for improving Batch Normalization. We also refer readers to the Appendix (Sec. A.3) 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 experi-

mental results, which aids in understanding each of the approaches as they are presented. We experiment with four standard image-centric datasets in this section: CIFAR-100, SVHN, Caltech-256, and ImageNet, and report results on validation datasets in order to fully describe each approach without contaminating test-set results. We give results on test sets, and experimental details in Sec. 4.4.

4.3.1 Inference Example Weighing

Batch Normalization has a disparity in function between training and inference: As previously noted, Batch Normalization calculates its normalization statistics (*i.e.* mean and variances) over each mini-batch 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 [4, 114, 87, 121, 50]. 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 shown¹ that the minimum possible output of a Batch Normalization layer is:

$$\min_{x_0, \dots, x_{B-1}} \gamma \frac{x_0 - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta = -\gamma\sqrt{B-1} + \beta \quad (4.2)$$

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. Beyond being a theoretical bound, in practice real networks actually do exceed this bound during inference: the output range of a network with Batch Normalization is wider during inference than during training (see Sec. A.2 in Appendix).

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

¹Proof in Appendix Sec. A.1

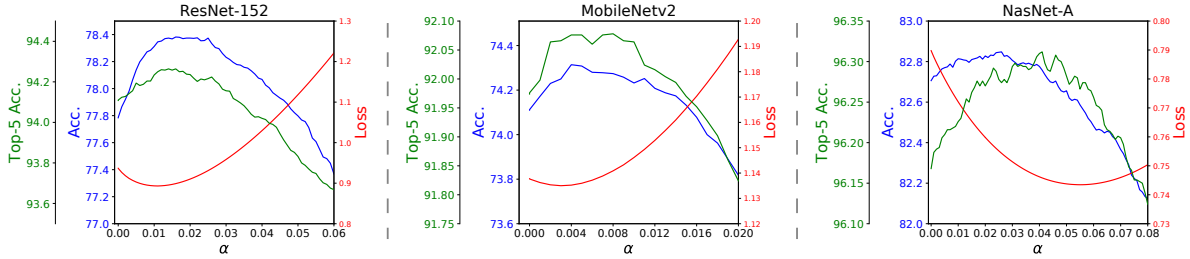


Figure 4.1: Effect of the example-weighting hyperparameter α on ImageNet for ResNet-152, MobileNetV2, and NASNet-A, measuring top-1 and top-5 accuracies and the cross-entropy loss.

m_{x^2} the corresponding moving average over x^2 , we apply the following normalization:

$$\begin{aligned}
 \mu_i &= \alpha E[x_i] + (1 - \alpha)m_x \\
 \sigma_i^2 &= (\alpha E[x_i^2] + (1 - \alpha)m_{x^2}) - \mu_i^2 \\
 \hat{x}_i &= \gamma \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta
 \end{aligned} \tag{4.3}$$

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

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 $\alpha = \frac{1}{B}$, since during training time an example occupies exactly $\frac{1}{B}$ of the batch. 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 (see subsequent experiments for evidence). 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. 4.1 we show the effect of α on the ImageNet ILSVRC 2012 validation set [86] for three diverse models: ResNet-

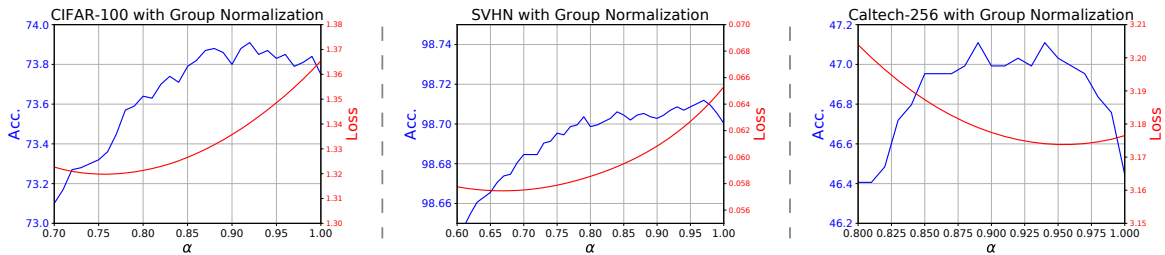


Figure 4.2: Effect of the example-weighting hyperparameter α for models trained with Group Normalization on CIFAR-100, SVHN, and Caltech-256.

152 [36], MobileNetV2 [89], and NASNet-A Large [128]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 large effects 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. We refer the reader to the Appendix Sec. A.4 for additional plots with larger ranges of α .

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 [121]. We demonstrate this in Fig. 4.2, 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. Since adding in any information from the moving averages at all represents a clear difference from the training setting of Group Normalization, this finding is counterintuitive. 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 with the application of data augmentation, 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 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.

²Models obtained from [96].

Last, we also note very recent work [99] which examines a similar approach for incorporating the statistics of an example during inference time, using per-layer weights and optimizing with a more involved procedure that encourages similar outputs to the training distribution.

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.

4.3.2 Ghost Batch Normalization for Medium Batch Sizes

Ghost Batch Normalization, a technique originally developed for training with very large batch sizes across many accelerators (*e.g.* GPUs) [42], consists of calculating normalization statistics on 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

$$\begin{aligned}\mu_i &= \frac{1}{B'} \sum_{j=1}^B x_j \left[\frac{jB'}{B} \right] = \left[\frac{iB'}{B} \right] \\ \sigma_i^2 &= \frac{1}{B'} \sum_{j=1}^B x_j^2 \left[\frac{jB'}{B} \right] = \left[\frac{iB'}{B} \right] - \mu_i^2\end{aligned}\tag{4.4}$$

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 [42], and has since been used by subsequent research rigorously studying the large-batch regime [93]. Here, we show that it can also be useful in the medium-batch setting³.

Why might Ghost Batch Normalization be useful? One reason is its power as a regularizer: due to the stochasticity in normalization statistics caused by the random selection of minibatches during training, Batch Normalization causes the representation of a training example to randomly change every time it appears in a different batch of data. By decreasing the number of examples that the normalization statistics are calculated over, Ghost Batch Normalization increases the variability of these changes, thereby increasing the amount of regularization. Based on this hypothesis, we would

³We experiment with batch sizes up to 128 in this chapter.

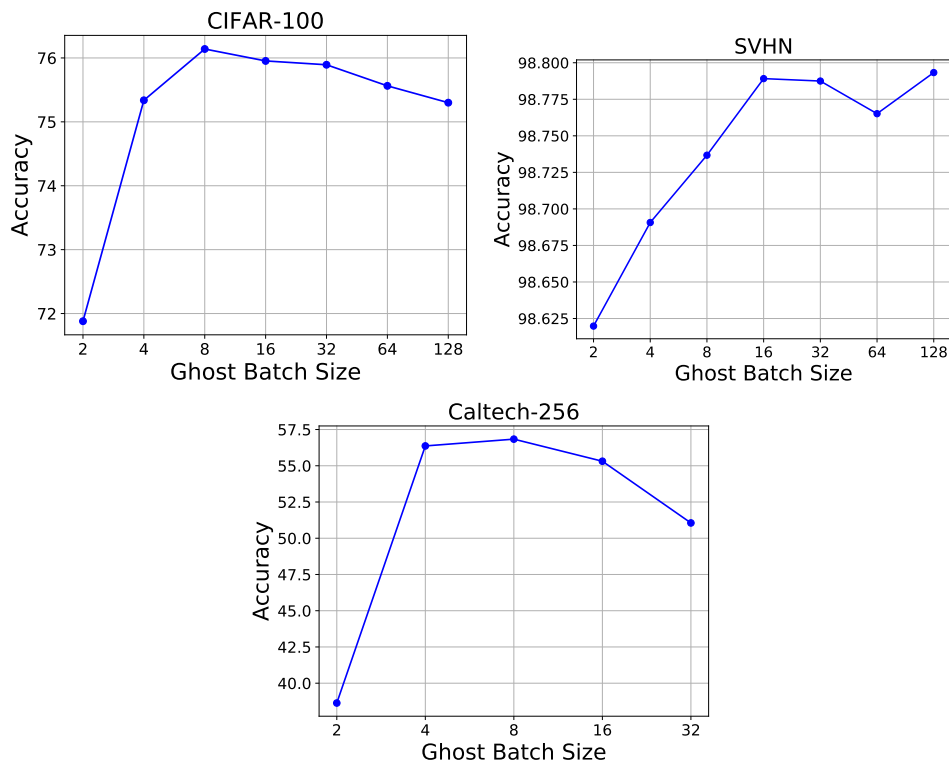


Figure 4.3: Accuracy vs. Ghost Batch Normalization size for CIFAR-100, SVHN, and Caltech-256.

expect to see a unimodal effect of the Ghost Batch Normalization size B' on model performance — a large value of B' would offer somewhat diminished performance as a weaker regularizer, a very low value of B' would have excess regularization and lead to poor performance, and an intermediate value would offer the best tradeoff of regularization strength.

We confirm this intuition in Fig. 4.3. Surprisingly, just using this one simple technique was capable of improving performance by 5.8% on Caltech-256 and 0.84% on CIFAR-100, which is remarkable given it has no additional cost during training. 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.

As a bonus, Ghost Batch Normalization has a synergistic effect with inference example weighing — it has the effect of making each example more important in calculating its own normalization statistics

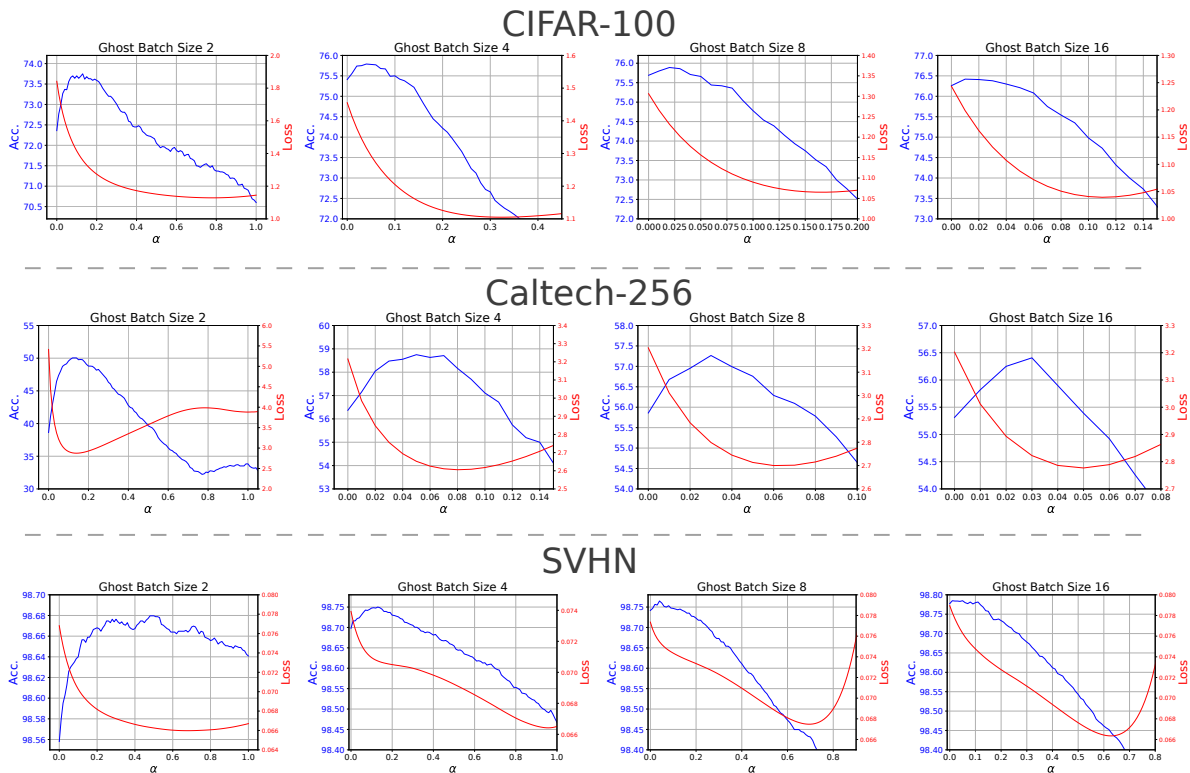


Figure 4.4: The complementary effects of Inference Example Weighing (Sec. 4.3.1) and Ghost Batch Normalization (Sec. 4.3.2) on CIFAR-100, SVHN, and Caltech-256.

μ_i and σ_i^2 . This importance is even greater the smaller that B' is, and it is precisely this effect that inference example weighing is designed to correct for. We show these results in Fig. 4.4, where we find increasing gain from inference example weighing as B' is made smaller, a gain that is in addition to the benefits of Ghost Batch Normalization itself. Interestingly, these examples also demonstrate that accuracy and cross-entropy, the most commonly-used classification loss, are only partially correlated, with the optimal values for the inference example weight α sometimes differing wildly between the two (e.g. for SVHN).

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.

4.3.3 Batch Normalization and Weight Decay

Weight decay [60] is a regularization technique that multiplies the weight of a neural network after each weight update from gradient descent by a factor of $1 - \delta$, and has a complex interaction with Batch Normalization. At first, it may even 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 $\epsilon = 0$ in the normalization layer). However, 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 [41, 115].

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 — Wu and He briefly mention weight decay with these parameters, where weight decay was used when training from scratch, but not fine-tuning, two other works [27, 35] have this form of weight decay explicitly turned off, and He *et al.* encourage disabling weight decay on γ and β , but ultimately find diminished performance by doing so.

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 [35, 36] have such paths for many of their normalization layers due to the chaining of skip-connections. However, Inception-style networks [109, 107] 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 hypothesis 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 since the model was originally hardly overfitting at all.

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.

4.3.4 Generalizing Batch and Group Normalization for Small Batches

While Batch Normalization is very effective in the medium to large-batch setting, it still suffers when not enough examples are available to calculate reliable normalization statistics. Although we have shown that techniques such as Inference Example Weighing (Sec. 4.3.1) can help significantly with this, it is still only a partial solution. At the same time, Group Normalization [121] 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. 4.3.2 and Sec. 4.3.3, 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%). 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.

4.4 Additional Experiments

4.4.1 Experimental Details

All results in Sec. 4.3 were performed on the validation datasets of each respective dataset (this section examines test set performance after hyperparameters have been optimized). Of the six datasets we experiment with, only ImageNet [86] and Flowers-102 [80] have their own pre-defined validation split, so we constructed validation splits for the other datasets as follows: for CIFAR-100 [58], 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 [79], we similarly split the 604,388 non-test images in an 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 for testing. For CUB-2011, we used 25% of the given training data as a validation set.

The model used for CIFAR-100 and SVHN was ResNet-18 [36, 35] with 64, 128, 256, and 512 filters across blocks. For Caltech-256, a much larger Inception-v3 [109] model was used, and we additionally experiment with ResNet-152 [36] on Flowers-102 and CUB-2011 in Sec. 4.4.3. All experiments were done on two Nvidia Geforce GTX 1080 Ti GPUs.

4.4.2 Combining All Four: 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, with a batch size $B = 128$. We plot results for CIFAR-100 and Caltech-256 in Fig. 4.5 (a), comparing against Group Normalization and an idealized Batch Normalization with constant performance across batch sizes (simulating if the problematic dependence of Batch Norm on the batch size were completely solved). On CIFAR-100, we see improvements against the best available baseline across all batch sizes.

For medium to large batch sizes ($B \geq 4$), improvements are driven by the combination of Ghost Batch Normalization (Sec. 4.3.2), Inference Example Weighing (Sec. 4.3.1), and weight decay introduced on γ and β (Sec. 4.3.3). To aid in distinguishing between these effects, we also plot the impact of Ghost Batch Normalization alone, which we find particularly impactful as long as the batch size isn't too small ($B > 2$). Turning to very small batch sizes, 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. 4.3.4), 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 $B = 2$.

4.4.3 Transfer Learning

We also show the applicability of these approaches in the context of transfer learning, which we demonstrate on the Flowers-102 [80] and CUB-2011 [118] datasets via fine-tuning a ResNet-152 model from ImageNet. These tasks present several challenges: 1) the Flowers-102 data only contains 10 images per category in the training set (and CUB-2011 only 30 examples per class), 2) pre-training models on ImageNet is a very strong form of prior knowledge, and despite the small dataset size may heavily reduce the regularization effects of some of the techniques, and 3) we examine the setting of pre-training with generic ImageNet models trained *without* any of these modifications, which gives an advantage to both the generic Batch Normalization and Group Normalization, for which pre-trained models exist.

We plot results in Fig. 4.5 (b), where we find remarkable qualitative agreement of our non-transfer learning results to this setting, despite the challenges. In total, on Flowers-102 our techniques were able to improve upon Batch Normalization by 2.4% (from 91.0% to 93.4% top-1 accuracy, a 27% relative

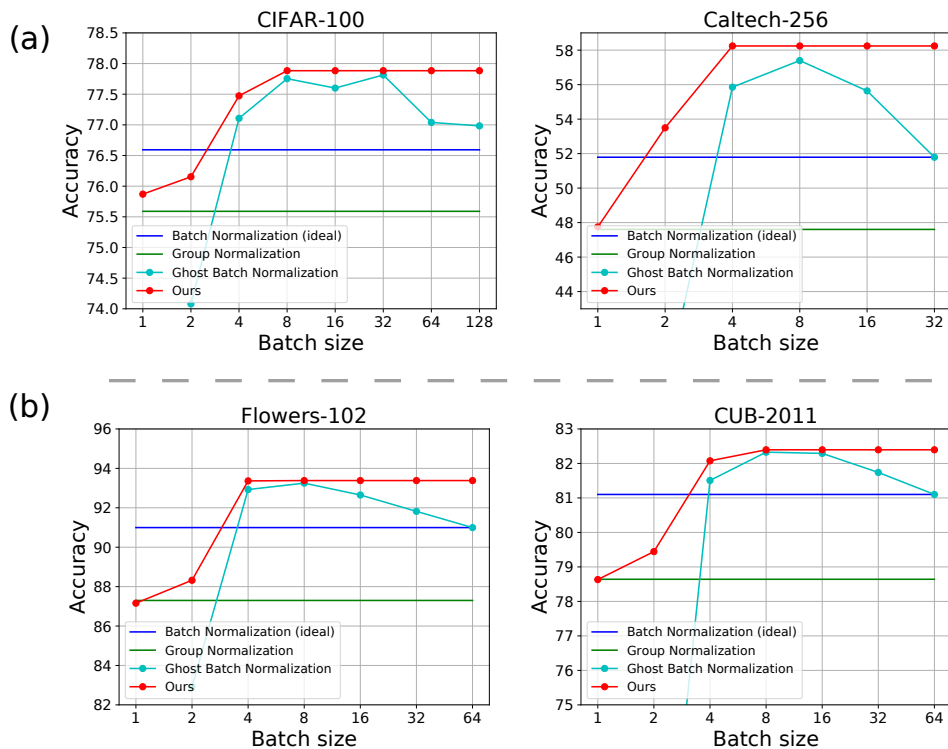


Figure 4.5: Total performance changes across batch sizes for CIFAR-100 and Caltech-256 (a) when training from scratch, incorporating all proposed improvements to Batch Normalization. On the bottom (b) is the same on Flowers-102 and CUB-2011, which employs transfer learning via fine-tuning from ImageNet. Also shown within each plot is the performance of Group Normalization, an idealized Batch Normalization that scales perfectly across batch sizes, and Ghost Batch Normalization (Sec. 4.3.2) by itself, for which the x-axis represents the Ghost Batch Size B' .

reduction in error), and upon Group Normalization by 6.1% (from 87.3%, a 48% relative reduction in error). On CUB-2011, which has more training data, we improved upon Batch Normalization by 1.4% (from 81.1% to 82.4%) and Group Normalization by 3.8% (from 78.6%).

We anticipate that even further improvements might arise by additionally pre-training models with some of these techniques (particularly Ghost Batch Normalization), as we were able to see a large impact (roughly 5%) on Group Normalization by pre-training with a Group Normalization-based model instead of Batch Normalization.

Table 4.1: Accuracy on CIFAR-100 with non-i.i.d. minibatches. B' refers to the Ghost Batch Normalization size (equivalent to the batch size for Batch Normalization and Batch Renormalization), and “Batch Group Norm.” refers to our approach in Sec. 4.3.4. “Inf. Ex. Weight: Off” refers to using only the moving averages for normalization statistics (*i.e.* $\alpha = 0$), while “On” refers to tuning α based on the validation set.

Method	B'	Inf. Ex. Weight: Off	Inf. Ex. Weight: On
Batch Norm	128	40.1	62.2
Batch ReNorm	128	69.0	69.0
Ghost Batch Norm	64	42.3	50.8
	32	57.8	70.9
	16	64.3	72.2
	8	68.7	72.0
	4	70.4	71.5
	2	68.4	71.4
Batch Group Norm.	2	75.2	76.1

4.4.4 Non-i.i.d. minibatches

An implicit assumption in Batch Normalization is that training examples are sampled independently, so that minibatch normalization statistics all follow roughly the same distribution and training statistics are faithfully represented in the moving averages. However, in applications where training batches are *not* sampled i.i.d., such as metric learning [81, 76] or hard negative mining [95], violating this assumption may lead to undesired consequences in the model. Here, we test our approaches in this challenging setting.

Following Batch Renormalization [50], we study the case where examples in a minibatch are sampled from a small number of classes — specifically, we consider CIFAR-100, and study the extreme case where each minibatch ($B = 128$) is comprised of examples from only four random categories (sampled with replacement), each of which is represented with 32 examples in the minibatch. We present results for Batch Normalization, Batch Renormalization, our generalization of Batch and Group Normalization from Sec. 4.3.4 (“Batch Group Norm.”), and the full interaction of Ghost Batch Normalization and Inference Example Weighing in Table 4.1. In this challenging setting, Inference Example Weighing, Ghost Batch Normalization, and Batch Group Norm all have large effect, in many cases halving the error rate of Batch Normalization. For example, Inference Example Weighing was able to reduce the error rate by 20% without any retraining, and tuning Ghost Batch Normalization, even with-

out any inference modifications, was just as effective as Batch Renormalization, a technique partially designed for the non-i.i.d. case. Even further, Batch Group Normalization was hardly affected at all by the non-i.i.d. training distribution (76.1 vs 76.2 for i.i.d.). Last, it is interesting to note that Inference Example Weighing had practically no effect on Batch Renormalization (improvement $\leq 0.1\%$), confirming Batch Renormalization's effect in making models more robust to the use of training vs moving average normalization statistics.

4.5 Conclusion

In this chapter, we have demonstrated four improvements to Batch Normalization applicable to 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.

5. Nondeterminism and Instability

In this final chapter, we examine an issue present and implicit in each previous chapter: the role of nondeterminism in the optimization process of neural networks. The direct effect of nondeterminism is an increase in uncertainty levels of performance, which makes it difficult to validate model improvements. Here, we develop a protocol for testing the effects of nondeterminism, and demonstrate that all sources of nondeterminism considered have similar effects across a wide range of model diversity measures. Then, we identify the instability of model training as the key factor creating this effect, and develop two approaches that reduce the effects of instability on run-to-run variability.

5.1 Introduction

Consider this common scenario: you have a baseline “current best” model, and are trying to improve it. One of your experiments has produced a model whose metrics are slightly better than the baseline. Yet you have your reservations — how do you know the improvement is “real” and not due to run-to-run variability?

Similarly, consider hyperparameter optimization, in which many possible values exist for a set of hyperparameters, with minor differences in performance between them. How do you pick the best hyperparameters, and how can you be sure that you’ve actually picked wisely?

In both scenarios, the standard practice is to train multiple independent copies of your model to understand its variability. While this helps address the problem, it is extremely wasteful, using more computing power, increasing the time required for effective research, and making reproducibility difficult, all while still leaving some uncertainty.

Ultimately, the source of this problem is nondeterminism in model optimization — randomized

components of model training that cause each run to produce different models with their own performance characteristics, even when given the same input. Nondeterminism itself occurs due to many factors: while the most salient source is the random initialization of parameters, other sources exist, including random shuffling of training data, stochasticity in data augmentation, explicit random operations (e.g. dropout [101]), asynchronous training [85], and even nondeterminism in low-level libraries such as cuDNN [8]. While it is often possible to set random seeds to control for these factors and make training deterministic (*i.e.* produce the same model weights every time training is done), doing so risks overfitting to a given set of random seeds and may also lead to slower training (*e.g.* using cuDNN in a deterministic mode is slower than when running nondeterministically).

Despite the clear impact nondeterminism has on the efficacy of modeling, relatively little attention has been paid towards understanding its mechanisms. In this chapter, we establish an experimental protocol for analyzing the impact of nondeterminism in model training, allowing us to quantify the independent effect of each source of nondeterminism. In doing so, we make a surprising discovery: each source has nearly the same effect on the variability of final model performance. Further, we find each source produces models of similar diversity, as measured by correlations between model predictions, functional changes in model performance while ensembling, and state-of-the-art methods of model similarity [57]. To emphasize one particularly interesting result: nondeterminism in low-level libraries like cuDNN can matter just as much with respect to model diversity and variability as varying the entire network initialization.

We explain this mystery by demonstrating that it can be attributed to *instability* in optimizing neural networks — when training with SGD-like approaches, we show that small changes to initial parameters result in large changes to final parameter values. In fact, the instabilities in the optimization process are extreme: *changing a single weight by the smallest possible amount within machine precision ($\sim 6 \cdot 10^{-11}$) produces nearly as much variability as all other sources combined*. Therefore, any source of nondeterminism with any effect at all on model weights inherits at least this level of variability.

Last, we present promising results in reducing the effects of instability on run-to-run variability. While we find that many approaches result in no apparent change, we propose and demonstrate two approaches that reduce model variability without any increase in model training time: accelerated model ensembling and test-time augmentation. Together, these provide the first encouraging signs for the tractability of this problem.

5.2 Related Work

Nondeterminism. Relatively little prior work has studied the effects of nondeterminism on model optimization. Within reinforcement learning, nondeterminism is recognized as a significant barrier to reproducibility and evaluating progress in the field [78, 37, 52, 71]. In the setting of supervised learning, though, the focus of this chapter, the problem is much less studied. Madhyastha and Jain [72] aggregate all sources of nondeterminism together into a single random seed and analyze the variability of model attention and accuracy as a function of it across various NLP datasets. They also propose a method for reducing this variability (see Sec. B.4 for details of our reproduction attempt). More common in the field, results across multiple random seeds are reported [16], but the precise nature of nondeterminism’s influence on variability goes unstudied.

Instability. We use the term “stability” analogously to numerical stability [38], in which a stable algorithm is one for which the final output (converged model) does not vary much as the input (initial parameters) are changed. In other contexts, the term “stability” has been used both in learning theory [6] and in reference to vanishing and exploding gradients [32].

5.3 Nondeterminism

Many sources of nondeterminism exist in neural network optimization, each of affects the variability of trained models. We begin with a very brief overview:

Parameter Initialization. When training a model, parameters without preset values are initialized randomly according to a given distribution, *e.g.* a zero-mean Gaussian with variance determined by the number of input connections to the layer [25, 34].

Data Shuffling. In stochastic gradient descent, the gradient is approximated on a random subset of examples, commonly implemented by using small batches of data iteratively in a shuffled training dataset [5]. Shuffling may happen either once, before training, or in between each epoch of training, the variant we use in this chapter.

Data Augmentation. A common practice, data augmentation refers to randomly altering each training example to artificially expand the training dataset [94]. For example, randomly flipping images encourages invariance to left/right orientation.

Stochastic Regularization. Some types of regularization, such as Dropout [101], take the form of stochastic operations internal to a model during training. Other instances of this include DropConnect [119] and variable length backpropagation through time [75], among many others. When not controlled for, these random operations can make training nondeterministic.

Low-level Operations. Often overlooked, many libraries that deep learning frameworks are built on, such as cuDNN [8] typically run nondeterministically for performance reasons. This nondeterminism is small — in one test we performed it caused an output difference of 0.003%. In the case of cuDNN, the library we test, it is possible to disable nondeterministic behavior at a speed penalty on the order of $\sim 15\%$. However, unlike nondeterminism sources, it is not possible to “seed” this; it is only possible to turn it on or off.

5.3.1 Protocol for Testing Effects of Nondeterminism

Performance Variability. Our protocol for testing the effects of sources of nondeterminism is based on properly controlling for each source. Formally, suppose there are N sources of nondeterminism, with source i controlled by seed S_i . To test the effect of source i , we keep all values $\{S_j\}_{j \neq i}$ set to a constant, and vary S_i with R different values, where R is the number of independent training runs performed. For sources of nondeterminism which cannot be effectively seeded, such as cuDNN, we indicate one of these values as the deterministic value, which it must be set to when varying the other sources of nondeterminism.

For example, denote S_1 the seed for random parameter initialization, S_2 for training data shuffling, and S_3 for cuDNN, where $S_3 = 1$ is the deterministic value for cuDNN. To test the effect of random parameter initialization, with a budget of $R = 100$ training runs, we set S_3 to the deterministic value of 1, S_2 to an arbitrary constant (typically 1 for simplicity), and test 100 different values of S_1 . All together, this corresponds to training models for each of $(S_1, S_2, S_3) \in \{(i, 1, 1)\}_{i=1}^{100}$. To measure variability of a particular evaluation metric (*e.g.* cross-entropy or accuracy, the standard metrics used for classification), we calculate the standard deviation (across all $R = 100$ models) of the metric. Note

that it is also possible to test the effects of several sources of nondeterminism in tandem this way, *e.g.* by considering $(S_1, S_2, S_3) \in \{(i, i, 0)\}_{i=1}^R$ to measure the joint effect of all three sources in this example, though it is more challenging to measure more precise interactions between several sources without running experiments for all possible combinations of random seeds.

Representation Diversity. We also examine differences in the *representation* of trained models, complementary to variability in test set performance — this allows us to differentiate cases where two sources of nondeterminism have similar performance variability but actually produce models with disparate amounts of representational similarity. In order to rigorously examine this, we consider four distinct analyses of the functional behavior of models:

The first and simplest metric we consider is the average disagreement between pairs of models, with higher disagreement corresponding to higher diversity and variability. In contrast to our other metrics, this considers only the argmax of a model’s predictions (*i.e.* the index of the prediction with highest output value), which makes it the most limited but also the most interpretable of the group. This metric has also been used recently to compare similarity in the context of network ensembles [20].

Second, we consider the average correlation between the predictions of two models, *i.e.* the expectation (across pairs of models from the same nondeterminism source), of the correlation of predictions, calculated across examples and classes. Concretely, for a classification task, the predicted logits from each of R models are flattened into vectors of length $N * C$ (with N test examples and C classes), and we calculate the mean correlation coefficient of the predictions across all $\binom{R}{2}$ pairs of models. We use Spearman’s ρ for the correlation coefficient, but note that other metrics are possible and yield similar conclusions. For this metric, a lower score indicates a more diverse set of models. This heuristic accounts for model outputs beyond the argmax, allowing one to examine similarities in the full output distribution of models.

The third analysis we perform examines the change in performance in ensembling two models from the same source of nondeterminism. Intuitively, if a pair of models are completely redundant, then ensembling them would result in no change in performance. However, if models actually learn different representations, then ensembling should on average create an improvement (unless there is a great difference in performance between the two, which is not the case in our experiments varying random seeds), with a greater improvement the greater the diversity in a set of models. Denoting by $f(S_i)$ some particular evaluation metric f calculated on the predictions of model S_i , and $\frac{S_i+S_j}{2}$ the ensemble of models S_i and S_j , this change is formally:

$$\frac{1}{\binom{R}{2}} \sum_{i=1}^R \sum_{j=i+1}^R \left(f\left(\frac{S_i + S_j}{2}\right) - \frac{f(S_i) + f(S_j)}{2} \right) \quad (5.1)$$

Last, for a more detailed view of learned representations internal to a network, we consider a state-of-the-art method for measuring the similarity of neural network representations, centered kernel alignment (CKA) [57], which has previously been used to analyze models trained with different random initializations, widths, and even entirely different architectures. Computing CKA between a pair of models results in a score bounded above by 1, where 1 is perfect representational similarity between models, and lower corresponds to greater dissimilarity. We use the linear version of CKA, which Kornblith *et al.* found to perform similarly to more complicated RBF kernels.

5.3.2 Experiments in Image Classification

We begin our study of nondeterminism with the fundamental task of image classification. We execute our protocol with CIFAR-10 [58] as a testbed, a 10-way classification dataset with 50,000 training images of resolution 32×32 pixels and 10,000 for testing. In these initial experiments, we use a 14-layer ResNet model [35], trained with a cosine learning rate decay [67] for 500 epochs with a maximum learning rate of .40, three epochs of learning rate warmup, a batch size of 512, momentum of 0.9, and weight decay of $5 \cdot 10^{-4}$, obtaining a baseline accuracy of 90.0%. Data augmentation consists of random crops and horizontal flips. All experiments were done on two NVIDIA Tesla V100 GPUs with `pytorch` [83].

We show the results of our protocol in this setting in Table 5.1. Across all measures of performance variability and representation diversity, what we find is surprising and clear — while there are slight differences, each source of nondeterminism has very similar effects on the variability of final trained models. In fact, random parameter initialization, arguably the most salient form of nondeterminism, does not stand out based on any of these metrics, and even combinations of multiple sources of nondeterminism produce remarkably little difference — all are within a maximum of 20% (relative) of each other. No nondeterminism source or combination of sources is even a factor of two more important than any other across any metric.

Turning toward CKA and representational diversity on intermediate layers in the model, we plot average CKA values across 6 representative layers in Fig. 5.1, done for pairwise combinations of 25

Table 5.1: The effect of each source of nondeterminism and several combinations of nondeterminism sources for ResNet-14 on CIFAR-10. The second and third columns give the standard deviation of accuracy and cross-entropy across 100 runs, varying only the nondeterminism source (700 trained models total). Also given are error bars, corresponding to the standard deviation of each standard deviation. The fourth, fifth, and sixth columns give the average percentage of examples models disagree on, the average pairwise Spearman’s correlation coefficient between predictions, and the average change in accuracy from ensembling two models, respectively (Sec. 5.3.1).

Nondeterminism Source	Accuracy SD (%)	Cross-Entropy SD	Pairwise Disagree (%)	Pairwise Corr.	Ensemble Δ (%)
Parameter Initialization	0.23 ± 0.02	0.0074 ± 0.0005	10.7	0.872	1.82
Data Shuffling	0.25 ± 0.02	0.0082 ± 0.0005	10.6	0.871	1.81
Data Augmentation	0.23 ± 0.02	0.0072 ± 0.0005	10.7	0.872	1.83
cuDNN	0.22 ± 0.01	0.0083 ± 0.0007	10.5	0.873	1.76
Data Shuffling + cuDNN	0.21 ± 0.01	0.0077 ± 0.0005	10.6	0.871	1.80
Data Shuffling + Aug. + cuDNN	0.22 ± 0.01	0.0074 ± 0.0005	10.7	0.871	1.84
All Nondeterminism Sources	0.26 ± 0.02	0.0072 ± 0.0005	10.7	0.871	1.82

Table 5.2: The effect of each source of nondeterminism for a QRNN on Penn Treebank; 100 runs per row. Note that lower PPL is better for language modeling tasks, so changes in PPL from ensembling are negative.

Nondeterminism Source	PPL SD	Pairwise Disagree (%)	Ensemble PPL Δ
Parameter Initialization	0.20 ± 0.01	17.3	-2.07
Stochastic Operations	0.19 ± 0.01	17.3	-2.08
All Nondeterminism Sources	0.18 ± 0.01	17.4	-2.07

models (due to the cost of CKA). Consistent with other analyses, CKA reveals that while some differences in average representational similarity exist between nondeterminism sources, particularly in the output of the first residual block (differences of up to 3.5%), by and large these differences are small, easily dwarfed in size by differences across layers (differences of up to 13%).

5.3.3 Experiments in Language Modeling

Here we show that this phenomenon is not unique to image classification by applying the same experimental protocol to language modeling. For these experiments, we employ a small quasi-recurrent neural network (QRNN) [7] on Penn Treebank [74], using the publicly available code of [75]. This

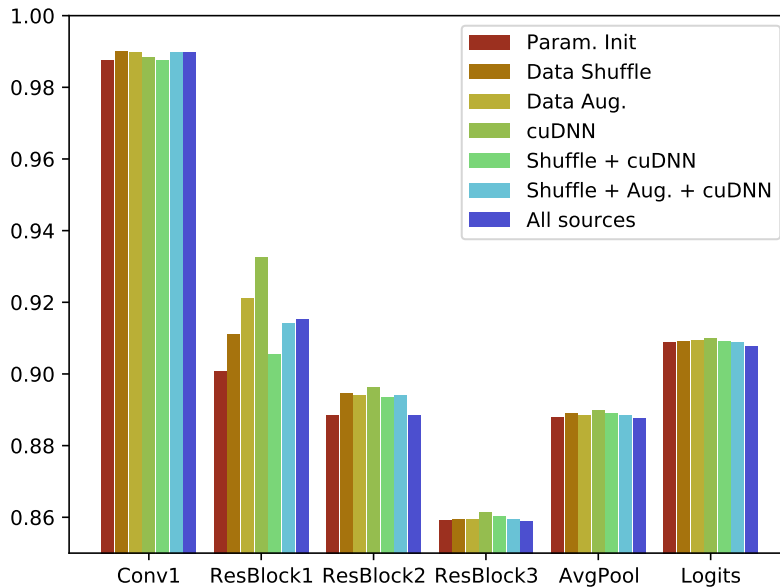


Figure 5.1: Average CKA representation similarity [57] for pairs of ResNet-14 models on CIFAR-10 across nondeterminism sources and a variety of network layers.

model uses a 256-dimensional word embedding, 512 hidden units per layer, and 3 layers of recurrent units, obtaining a perplexity (PPL) of 75.49 on the Penn Treebank test set.

For this task, two sources of nondeterminism are relevant: random parameter initialization, and stochastic operations, including a variation of dropout and variable length backpropagation through time, which share a common seed. To measure performance variability, PPL is the most widely-accepted metric, and for diversity in representation we focus on only two metrics (pairwise disagreement and benefits from ensembling) because CKA was not designed for variable-length input and standard computing libraries [117] are not efficient enough to calculate $O(R^2)$ correlation coefficients with such large inputs.

We show results in Table 5.2, where we find almost no difference across all diversity metrics, showing the phenomenon generalizes beyond image classification and ResNets.

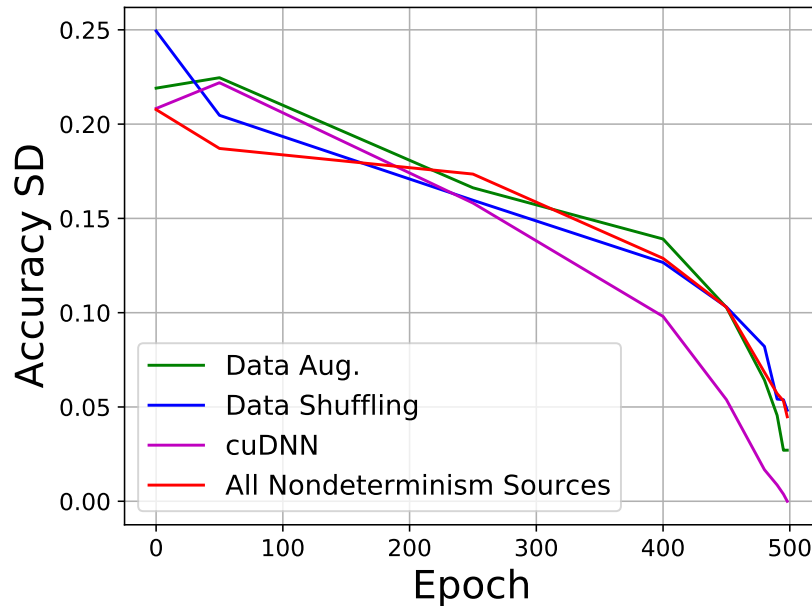


Figure 5.2: The effect of the onset of nondeterminism on the variability of accuracy in converged models. Each point corresponds to training 100 models deterministically for a certain number of epochs (x-axis), then enabling a given source of nondeterminism by varying its seed starting from that epoch and continuing through to the end of training. Epochs evaluated are $\{0, 50, 250, 400, 450, 480, 490, 495, 498\}$, with a linear interpolation shown.

5.3.4 Nondeterminism Throughout Training

One hypothesis for the this phenomenon’s cause is the sensitivity of optimization in the initial phase of learning, which recent work has demonstrated in other contexts [2, 21]. With our experimental protocol, this is straightforward to test: If this were the case, then training models identically for the first N epochs and only then introducing nondeterminism would result in a significantly less variability in trained models, measured across any metric of choice. Furthermore, by varying N , we can actually determine *when* in training each source of nondeterminism has its effect (for sources that vary over the course of training, *i.e.* not random parameter initialization).

We perform this experiment for the ResNet-14 model on CIFAR-10 in Fig. 5.2, where we find that the beginning of training is *not* particularly sensitive to nondeterminism. Instead, model variability is nearly as high when enabling nondeterminism even after 50 epochs, and we see only a gradual reduction in final model variability as the onset of nondeterminism is moved later and later.

Table 5.3: The effect of instability — randomly changing a single weight by one bit during initialization for ResNet-14 on CIFAR-10.

Nondeterminism Source	Accuracy SD (%)	Cross-Entropy SD	Pairwise Disagree (%)	Pairwise Corr.	Ensemble Δ (%)
Random Bit Change	0.21 ± 0.01	0.0068 ± 0.0004	10.6	0.874	1.82

5.4 Instability

Why does each source of nondeterminism have similar effects on model variability? To answer this question, we find the smallest possible change that produces the same amount of variability, in the process demonstrating the *instability* in optimizing neural networks.

5.4.1 Instability and Nondeterminism

To demonstrate, we perform a simple experiment: First we deterministically train a simple ResNet-14 model on CIFAR-10, achieving a test cross-entropy of 0.3519 and accuracy of 90.0%. Then, we train another model in an identical fashion, with exactly equal settings for all sources of nondeterminism, but one extremely small change: randomly pick a single weight in the first layer and change its value by the smallest possible amount in a 32-bit floating point representation, *i.e.* an addition or subtraction of a single bit in the least-significant digit. As an example, this could change a value from -0.0066514308 to -0.0066514313 , a difference on the order of $5 \cdot 10^{-10}$.

What happens when we optimize this model, different from the original by only a single bit? By the end of the first epoch of training, with the learning rate still warming up, the new model already differs in accuracy by 0.18% (25.74% vs 25.56%). In one more epoch the difference is a larger 2.33% (33.45% vs 31.12%), and after three epochs, the difference is a staggering 10.42% (41.27% vs 30.85%). Finally, at the end of training the model weights converge, the new model obtained an accuracy of 90.12% and a cross-entropy of 0.34335, substantially different from the original despite only a tiny change in initialization. For another perspective, viewing the optimization process end-to-end, with the initial parameters as the input and a given performance metric as the output, we have demonstrated a condition number $\frac{\|\delta f\|}{\|\delta x\|}$ of $1.8 \cdot 10^7$ for cross-entropy and $2.6 \cdot 10^8$ for accuracy.

We can more rigorously test this using our protocol from Sec. 5.3 — this time, our source of nondeterminism is randomly picking a different weight to change in each model training run, then

Table 5.4: The effect of instability for a QRNN on Penn Treebank. Also see Table 5.2 for comparison.

Nondeterminism Source	PPL SD	Pairwise Disagree (%)	Ensemble PPL Δ
Random Bit Change	0.19 ± 0.01	17.7	-2.07

either incrementing or decrementing it to the next available floating-point value. We show the results in Table 5.3 for image classification on CIFAR-10 (*c.f.* Table 5.1 for comparison) and Table 5.4 for language modeling on Penn Treebank (*c.f.* Table 5.2), where we find that even this small change produces roughly as much variability in model performance as every other source of nondeterminism.

From this, it is easy to see why every other source of nondeterminism has similar effects — so long as nondeterminism produces any change in model weights, whether by changing the input slightly, altering the gradient in some way, or any other effect, it will produce *at least* as much model variability as caused by the instability of model optimization.

5.4.2 Instability and Depth

Instability occurs in networks of more than a single layer.

Due to convexity, linear models optimized with a cross-entropy loss and an appropriate learning rate schedule always converge to a global minimum. However, in practice we find an even stronger property: when initial weights are modified by a single bit, beyond simply converging to the same final value, the entire optimization trajectory stays close to that of an unperturbed model, never differing by more than a vanishingly small amount. At convergence, a set of linear models trained in this way with only single random bit changes had a final accuracy SD of 0 (*i.e.* no changes in any test set predictions) and cross-entropy SD of $\sim 1 \cdot 10^{-7}$, far below that of any deeper model.

In contrast, instability occurs as soon as a single hidden layer was added, with an accuracy SD of 0.28 and cross-entropy SD of 0.0051 for a model with a fully-connected hidden layer, and an accuracy SD of 0.14 and cross-entropy SD of 0.0022 when the hidden layer is convolutional, both a factor of 10,000 greater than the linear model. See Sec. B.1 and Sec. B.2 for full details and a visualization of the effects of instability during training.

5.5 Reducing Variability

Here we identify and demonstrate two approaches that partially mitigate the variability caused by non-determinism and instability. See the Sec. B.4 for learnings on approaches which *did not* reduce variability.

Accelerated Ensembling. As previously mentioned, the standard practice for mitigating run-to-run variability is to train multiple independent copies of a model, gaining a more robust performance estimate by measuring a metric of interest over multiple trials. Ensembling is a similar alternative approach, which shares the intuition of multiple independent training runs, but differs in that the predictions themselves are averaged and the performance of the ensembled model itself is measured. However, since ensembling still requires training multiple model copies, it does not reduce the burden caused by nondeterminism and instability.

To that end, we propose the use of *accelerated* ensembling techniques to reduce variability. Accelerated ensembling is a recent research direction with a goal of mimicking a full model ensemble but only requiring one training run, with a variety of techniques to achieve this goal under research [45, 22, 120]. While such techniques typically underperform regular ensembles, which are composed out of models trained independently, the nature of their accelerated training can reduce variability without incurring additional cost during training.

The approach we focus on is Snapshot Ensembles [45], which uses a cyclic learning rate schedule, creating ensemble members when the learning rate goes to 0 (before rising again due to the cyclic learning rate).

In Table 5.5, we compare a snapshot ensemble (“Acc. Ens.”) with 5 cycles in its learning rate (*i.e.* model snapshots are taken after every 100 epochs of training) to ordinary ensembling on CIFAR-10 with all sources of nondeterminism enabled. Despite training only a single model, the accelerated ensemble had variability in accuracy and cross-entropy comparable to an ensemble of two independently-trained models, with other metrics comparable to those of even larger ensembles. Across measures, accelerated ensembling reduces variability by an average of 48% relative.

Test-Time Data Augmentation. Test-time data augmentation (TTA) is the practice of augmenting test set examples using data augmentation, averaging model predictions made on each augmented

Table 5.5: Comparison of single and ensemble model variability on CIFAR-10 with proposed methods for reducing the effects of nondeterminism. For standard ensembles, N denotes the number of constituent models, “Acc. Ens.” uses the Snapshot Ensemble method of accelerated ensembling, and [Single|Acc. Ens.]/[Flip|Flip-Crop]-TTA use either horizontal flips or flips and crops for test-time augmentation on top of either regular single models or an accelerated ensemble. Also shown is the training time and average relative reduction in variability across metrics compared to the baseline “Single Model”. All results are based on 100 runs of model training.

Model	Training Cost	Accuracy SD (%)	Cross-Entropy SD	Pairwise Disagree (%)	Pairwise Corr.	Ensemble Δ (%)	Variability Reduction
Single Model	1 \times	0.26 \pm 0.02	0.0072 \pm 0.0005	10.7	0.871	1.82	<i>n/a</i>
Ensemble ($N = 2$)	2 \times	0.19 \pm 0.02	0.0044 \pm 0.0004	6.9	0.929	0.89	39%
Ensemble ($N = 3$)	3 \times	0.15 \pm 0.02	0.0033 \pm 0.0005	5.5	0.951	0.59	55%
Ensemble ($N = 4$)	4 \times	0.17 \pm 0.02	0.0030 \pm 0.0004	4.6	0.963	0.43	60%
Ensemble ($N = 5$)	5 \times	0.12 \pm 0.02	0.0028 \pm 0.0004	4.1	0.970	0.34	67%
Acc. Ens.	1 \times	0.19 \pm 0.02	0.0044 \pm 0.0003	6.1	0.957	0.63	48%
Single/Flip-TTA	1 \times	0.24 \pm 0.02	0.0061 \pm 0.0005	8.2	0.905	1.20	21%
Single/Flip-Crop-TTA	1 \times	0.19 \pm 0.01	0.0049 \pm 0.0004	6.9	0.922	0.92	37%
Acc. Ens./Flip-TTA	1 \times	0.15 \pm 0.01	0.0039 \pm 0.0003	5.0	0.967	0.45	58%
Acc. Ens./Flip-Crop-TTA	1 \times	0.16 \pm 0.01	0.0033 \pm 0.0002	4.6	0.972	0.38	61%

example, and is typically used to improve generalization [108]. TTA can be thought of as a form of ensembling in data-space, since predictions of different augmented examples are ensembled together, providing a parallel to the model-space averaging of standard ensembling, which partially mitigated the variability due to nondeterminism.

In Table 5.5 (last four rows), we show results on CIFAR-10 with horizontal flip TTA and image cropping TTA (details in Sec. B.3), and also experiment with combining accelerated ensembling with TTA. Simple flip TTA reduces variability across all metrics (21% relative reduction on average), and adding crops pushes this to 37%. Combined with accelerated model ensembling, this reduces variability by 61%, while maintaining the same training budget as the baseline “Single Model”.

5.6 Generalization Experiments

In this section we detail additional experiments to show the generalization of our results on nondeterminism, instability, and methods to reduce variability to other datasets (MNIST, ImageNet) and model architectures. We compile our main generalization results in Table 5.6, with additional results in Appendix B.

Table 5.6: Generalization experiments of nondeterminism and instability with other architectures on CIFAR-10, ImageNet, and MNIST. For CIFAR-10 and MNIST, each row is computed from the statistics of 100 trained models, and for ImageNet, each row is computed from 30 trained models. Within each section the most relevant comparisons to make are between “Random Bit Change” and “All Nondeterminism Sources” to evaluate instability, and between “All Nondeterminism Sources”, “Acc. Ens.”, and each TTA method to evaluate the efficacy of our proposals to mitigate the effects of nondeterminism and instability (all TTA models have all sources of nondeterminism enabled). Notation follows Tables 5.1 and 5.5.

Nondeterminism Source	Accuracy SD (%)	Cross-Entropy SD	Pairwise Disagree (%)	Pairwise Corr.	Ensemble Δ (%)
CIFAR-10: ResNet-6					
Parameter Initialization	0.50 ± 0.04	0.0117 ± 0.0010	20.0	0.925	2.17
All Nondeterminism Sources	0.43 ± 0.03	0.0106 ± 0.0007	20.1	0.924	2.17
Random Bit Change	0.41 ± 0.02	0.0094 ± 0.0006	19.8	0.925	2.12
Single/Flip-Crop-TTA	0.44 ± 0.03	0.0096 ± 0.0006	15.6	0.949	1.41
Acc. Ens.	0.45 ± 0.03	0.0104 ± 0.0007	14.0	0.963	0.99
Acc. Ens./Flip-Crop-TTA	0.43 ± 0.03	0.0096 ± 0.0006	11.6	0.973	0.71
CIFAR-10: ResNet-18					
Parameter Initialization	0.15 ± 0.01	0.0067 ± 0.0005	4.7	0.814	0.71
All Nondeterminism Sources	0.18 ± 0.01	0.0073 ± 0.0005	4.8	0.808	0.75
Random Bit Change	0.13 ± 0.01	0.0060 ± 0.0005	4.7	0.830	0.73
Single/Flip-Crop-TTA	0.14 ± 0.01	0.0047 ± 0.0003	3.4	0.851	0.41
Acc. Ens.	0.13 ± 0.01	0.0038 ± 0.0003	2.9	0.884	0.31
Acc. Ens./Flip-Crop-TTA	0.11 ± 0.01	0.0029 ± 0.0002	2.2	0.909	0.19
CIFAR-10: ShuffleNetv2-50%					
Parameter Initialization	0.22 ± 0.01	0.0112 ± 0.0007	8.4	0.696	1.38
All Nondeterminism Sources	0.22 ± 0.02	0.0123 ± 0.0008	8.4	0.692	1.40
Random Bit Change	0.21 ± 0.01	0.0107 ± 0.0006	8.3	0.695	1.36
Single/Flip-Crop-TTA	0.18 ± 0.01	0.0093 ± 0.0007	6.5	0.762	0.90
Acc. Ens.	0.18 ± 0.01	0.0067 ± 0.0005	5.0	0.930	0.52
Acc. Ens./Flip-Crop-TTA	0.15 ± 0.01	0.0051 ± 0.0004	4.1	0.948	0.35
CIFAR-10: VGG-11					
Parameter Initialization	0.20 ± 0.01	0.0063 ± 0.0004	6.6	0.807	0.91
All Nondeterminism Sources	0.18 ± 0.01	0.0065 ± 0.0004	6.6	0.806	0.94
Random Bit Change	0.16 ± 0.01	0.0060 ± 0.0004	6.5	0.811	0.89
Single/Flip-Crop-TTA	0.15 ± 0.01	0.0042 ± 0.0003	4.2	0.892	0.36
Acc. Ens.	0.13 ± 0.01	0.0041 ± 0.0003	4.1	0.914	0.39
Acc. Ens./Flip-Crop-TTA	0.11 ± 0.01	0.0026 ± 0.0002	2.8	0.951	0.17
MNIST					
Parameter Initialization	0.047 ± 0.0036	0.0024 ± 0.0001	0.54	0.941	0.064
All Nondeterminism Sources	0.046 ± 0.0032	0.0022 ± 0.0001	0.56	0.939	0.068
Random Bit Change	0.035 ± 0.0026	0.0011 ± 0.0001	0.30	0.989	0.011
Single/Crop-TTA	0.039 ± 0.0025	0.0016 ± 0.0001	0.38	0.953	0.037
Acc. Ens.	0.050 ± 0.0031	0.0019 ± 0.0001	0.55	0.943	0.064
Acc. Ens./Crop-TTA	0.046 ± 0.0028	0.0013 ± 0.0001	0.40	0.956	0.039
ImageNet: ResNet-50 (5 epoch)					
70					
All Nondeterminism Sources	0.27 ± 0.03	0.0625 ± 0.0142	26.3	0.933	1.34
Random Bit Change	0.34 ± 0.05	0.0935 ± 0.0313	24.7	0.944	1.18
Single/Flip-TTA	0.27 ± 0.03	0.0604 ± 0.0137	23.6	0.942	1.04
Single/Crop-TTA	0.26 ± 0.03	0.0653 ± 0.0152	24.4	0.939	1.17
Single/Flip-Crop-TTA	0.26 ± 0.03	0.0636 ± 0.0146	22.3	0.946	0.95

CIFAR-10. On CIFAR-10, in addition to the ResNet-14 employed throughout this chapter, we experiment with a smaller 6-layer variant, larger 18-layer variant, VGG-11 [98], and a 50%-capacity ShuffleNetv2 [70]. As shown in Table 5.6, the observations around instability and its relationship to nondeterminism generally hold for these architectures, with a close correspondence between the magnitude of effects for a random bit change and each of the five metrics considered.

Turning towards our proposals (Sec. 5.5) for mitigating the effects of nondeterminism and instability on model variability, we find across all model architectures that both accelerated ensembling and test-time augmentation reduce variability across nearly all metrics, with perhaps larger relative reductions for larger models and the pairwise metrics. Only the intersection of the smallest model (ResNet-6) and metrics of performance variability (Accuracy SD and Cross-Entropy SD) was there no benefit.

MNIST. Experiments on MNIST [64], allow us to test whether our observations hold for tasks with very high accuracy — 99.14% for our relatively simple baseline model, which has two convolution and fully-connected layers. As before, we find similar effects of nondeterminism for parameter initialization and all nondeterminism sources, including a comparable effect (albeit smaller) from a single random bit change, highlighting that the instability of training extends even to datasets where the goal is simpler and model performance is higher. Of note, though, is the relatively smaller effect of a single bit change on pairwise metrics of diversity, further suggesting that the magnitude of instability might be at least partially related to the interplay of model architecture, capacity, and degree of overfitting.

In terms of the mitigations against variability, only test-time augmentation appeared to significantly help. For MNIST, the only augmentation employed was cropping, with a small 1-pixel padding (models were trained with no data augmentation). While the fact that accelerated ensembling did not result in improvements is not particularly important in practice (since MNIST models are fast to train), it is an interesting result, which we hypothesize is also related to the degree of overfitting (similar to ResNet-6 on CIFAR-10).

ImageNet. For ImageNet, it is computationally prohibitive to run full training on the large number of models needed for a truly rigorous evaluation of nondeterminism and instability, so we perform an approximate analysis by training multiple ResNet-50 models for 5 epochs, obtaining an average top-1 accuracy of 56.8% on the ImageNet validation set — on par with a fully-trained AlexNet [59], though considerably lower than a ResNet-50 trained to completion.

Of particular note, we find evidence supporting instability, with “Random Bit Change” having comparable levels of variability compared to models trained with all nondeterminism sources, despite only 5 epochs of training. For reducing variability, we focus our efforts on TTA due to the short training time, where we find modest improvements for both flipping-based TTA and crop-based TTA on pairwise metrics, with no clear pattern for Accuracy and Cross-Entropy SD (which have large error bars).

5.7 Conclusion

In this chapter, we have shown two surprising facts: First, though conventional wisdom holds that run-to-run variability in model performance is primarily determined by random parameter initialization, many sources of nondeterminism actually result in similar levels of variability. Second, a key driver of this phenomenon is the instability of model optimization, in which changes on the order of 10^{-10} in a single weight at initialization can have as much effect as reinitializing all weights to completely random values. We have also identified two approaches for reducing the variability in model performance and representation without incurring any additional training cost.

We hope that our work sheds light on a complex phenomenon that affects all deep learning researchers and inspires further research on improving optimization stability.

6. Conclusion

In this thesis, we have examined a range of key topics in deep learning, furthering the field’s understanding of each topic and using the understanding to generate technical advances that improve the state of the art. In data augmentation, we have disproven the primary hypothesis for the efficacy of mixed-example approaches, introducing new superior methods along the way. For adversarial robustness, we have shown that logit pairing methods derive much of their benefit from logit regularization, extending and improving the approach via other logit regularization methods. Considering batch normalization, we developed an understanding of its behavior during training and inference, and introduced four unique changes that improve models using batch normalization across a variety of settings. Finally, we examined the effects of nondeterminism on final trained models, showed that different sources of nondeterminism produce similar diversity in models, identified that this is due to instability in model optimization, and proposed two methods for reducing the effects of instability.

Though we have advanced the field’s understanding of these topics, further exciting research is still possible and valuable. For example, developing a perfect data augmentation approach would have nearly unlimited benefit, as it would allow training of models with essentially arbitrarily large amounts of data. Alternatively, although we have improved adversarial robustness, a big gap still exists between the worst-case and average-case performance of trained models. When it comes to nondeterminism, while we have provided the first promising results at reducing run-to-run variability in model performance, further advances are likely possible that would serve to reduce the computational demands of iterating on deep learning model development. Even more generally, as long as progress continues to be made in deep learning as a field, the need for deeper understanding will remain.

A. Appendix: Improving Batch Normalization

A.1 Proof of Batch Normalization Output Bounds

Here we present a proof of Eq. 4.2. 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

$$\min_{x_0, \dots, x_{B-1}} \gamma \frac{x_0 - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta = -\gamma\sqrt{B-1} + \beta \quad (\text{A.1})$$

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

$$\min_{x_0, \dots, x_{B-1}} \frac{x_0 - \frac{1}{B} \sum_{i=0}^{B-1} x_i}{\sqrt{\frac{1}{B} \sum_{i=0}^{B-1} (x_i - \frac{1}{B} \sum_{j=0}^{B-1} x_j)^2 + \epsilon}} = -\sqrt{B-1} \quad (\text{A.2})$$

From here we assume without loss of generality that $x_0 = 0$ – since the output of Batch Normalization is invariant to an additive constant on all x_i , we can subtract x_0 from all x_i and maintain the same value. We also assume that all $x_i \geq 0$, then frame the minimum as a bound

$$\frac{-\frac{1}{B} \sum_{i=0}^{B-1} x_i}{\sqrt{\frac{1}{B} \sum_{i=0}^{B-1} (x_i - \frac{1}{B} \sum_{j=0}^{B-1} x_j)^2 + \epsilon}} \geq -\sqrt{B-1} \quad (\text{A.3})$$

$$-\frac{1}{B} \sum_{i=0}^{B-1} x_i \geq -\sqrt{\frac{B-1}{B} \sum_{i=0}^{B-1} \left(x_i - \frac{1}{B} \sum_{j=0}^{B-1} x_j \right)^2} + \epsilon \quad (\text{A.4})$$

$$-\frac{1}{B} \sum_{i=0}^{B-1} x_i \geq -\sqrt{\frac{B-1}{B} \sum_{i=0}^{B-1} \left(x_i^2 - \frac{2x_i}{B} \sum_{j=0}^{B-1} x_j + \frac{1}{B^2} \left(\sum_{j=0}^{B-1} x_j \right)^2 \right)} + \epsilon \quad (\text{A.5})$$

$$\sum_{i=0}^{B-1} x_i \leq \sqrt{\sum_{i=0}^{B-1} \left((B-1)Bx_i^2 - 2(B-1)x_i \sum_{j=0}^{B-1} x_j + \frac{B-1}{B} \left(\sum_{j=0}^{B-1} x_j \right)^2 \right)} + \epsilon \quad (\text{A.6})$$

$$\sum_{i=0}^{B-1} x_i \leq \sqrt{(B-1)B \sum_{i=0}^{B-1} x_i^2 - 2(B-1) \left(\sum_{i=0}^{B-1} x_i \right)^2 + (B-1) \left(\sum_{i=0}^{B-1} x_i \right)^2} + \epsilon \quad (\text{A.7})$$

$$\sum_{i=0}^{B-1} x_i \leq \sqrt{(B-1)B \sum_{i=0}^{B-1} x_i^2 - (B-1) \left(\sum_{i=0}^{B-1} x_i \right)^2} + \epsilon \quad (\text{A.8})$$

$$\left(\sum_{i=0}^{B-1} x_i \right)^2 \leq (B-1)B \sum_{i=0}^{B-1} x_i^2 - (B-1) \left(\sum_{i=0}^{B-1} x_i \right)^2 + \epsilon \quad (\text{A.9})$$

$$B \left(\sum_{i=0}^{B-1} x_i \right)^2 \leq (B-1)B \sum_{i=0}^{B-1} x_i^2 + \epsilon \quad (\text{A.10})$$

$$\left(\sum_{i=0}^{B-1} x_i \right)^2 \leq (B-1) \sum_{i=0}^{B-1} x_i^2 + \epsilon \quad (\text{A.11})$$

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

$$\left(\sum_{i=1}^{B-1} x_i \right)^2 \leq (B-1) \sum_{i=1}^{B-1} x_i^2 \quad (\text{A.12})$$

With a change of variables, we have the more general

$$\left(\sum_{i=0}^{N-1} x_i \right)^2 \leq N \sum_{i=0}^{N-1} x_i^2 \quad (\text{A.13})$$

$$\frac{1}{N^2} \left(\sum_{i=0}^{N-1} x_i \right)^2 \leq \frac{1}{N} \sum_{i=0}^{N-1} x_i^2 \quad (\text{A.14})$$

$$\mathbb{E}[x]^2 \leq \mathbb{E}[x^2] \quad (\text{A.15})$$

$$\mathbb{E}[x^2] - \mathbb{E}[x]^2 \geq 0 \quad (\text{A.16})$$

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:

$$\frac{-\frac{1}{B} \sum_{i=0}^{B-1} x_i}{\sqrt{\frac{1}{B} \sum_{i=0}^{B-1} (x_i - \frac{1}{B} \sum_{j=0}^{B-1} x_j)^2 + \epsilon}} \quad (\text{A.17})$$

$$\frac{-\frac{1}{B} \sum_{i=1}^{B-1} a}{\sqrt{\frac{1}{B} \left(\frac{(B-1)^2}{B^2} a^2 + \sum_{i=1}^{B-1} (a - \frac{B-1}{B} a)^2 \right) + \epsilon}} \quad (\text{A.18})$$

$$\frac{-\frac{B-1}{B} a}{\sqrt{\frac{1}{B} \left(\frac{(B-1)^2}{B^2} a^2 + (B-1) a^2 \left(1 - \frac{2(B-1)}{B} + \frac{(B-1)^2}{B^2} \right) \right) + \epsilon}} \quad (\text{A.19})$$

$$\frac{-(B-1)a}{B \sqrt{\frac{a^2(B-1)}{B} \left(\frac{B-1}{B^2} + 1 - 2\frac{B-1}{B} + \frac{(B-1)^2}{B^2} \right) + \epsilon}} \quad (\text{A.20})$$

$$\frac{-(B-1)a}{\sqrt{a^2(B-1) \left(\frac{B-1}{B} + B - 2B + 2 + \frac{(B-1)^2}{B} \right) + \epsilon}} \quad (\text{A.21})$$

$$\frac{-(B-1)a}{\sqrt{a^2(B-1) \left(\frac{B^2 - 2B + 1 + B - 1 - B^2 + 2B}{B} \right) + \epsilon}} \quad (\text{A.22})$$

$$\frac{-(B-1)a}{\sqrt{a^2(B-1) + \epsilon}} \quad (\text{A.23})$$

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

$$\frac{-(B-1)a}{a\sqrt{(B-1)}} \tag{A.24}$$

which is simply

$$-\sqrt{(B-1)} \tag{A.25}$$

completing the proof.

A.2 Empirical Evidence of Batch Normalization Output Bounds

In Fig. A.1 we show the observed output ranges for the last Batch Normalization layer in our CIFAR-10 network (spatial resolution: 4×4), plotting both the range during training and at inference time on the CIFAR-10 test set. Different values of B were obtained by using different Ghost Batch Normalization sizes, keeping in mind that B is determined by the product of the batch size and spatial dimensions.

At large values of B , it is unlikely that any network obtains a value even close to the bound of Eq. 4.2, but as B gets smaller, the output range of the network during training becomes smaller in magnitude, eventually being nearly tight with our bound — for example, for $\log(B) = 5$, the theoretical minimum is -5.57 , while the network obtained a minimum of -5.30 . However, the maximum and minimum values obtained during inference on the test set show no clear pattern as B changes, and are not subject to the training time bound, which is particularly noticeable for small values of B , where values fall outside the training-time bounds.

A.3 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

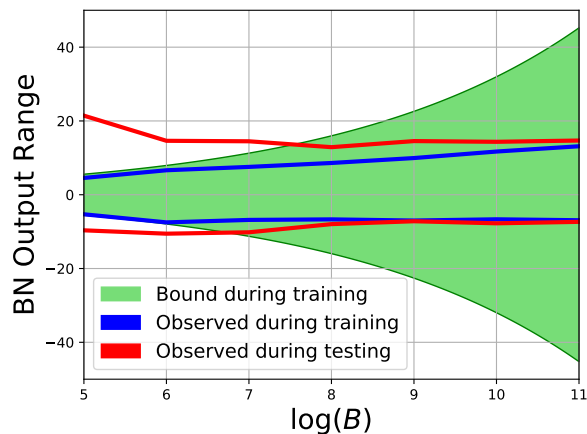


Figure A.1: Range of output values obtained during inference on the CIFAR-10 test set, compared with the range observed during training and the bound of Eq. 4.2. See text for details.

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 than 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.

A.4 Supplemental Inference Example Weighing Plots

In Figures A.2, A.3, and A.4 we present plots corresponding to Figures 4.1, 4.2, and 4.4 of the main text, with larger ranges of the inference weight α . In the main text, we restricted the range of α to values which showed off the tradeoff of α versus performance at a reasonably local scale, and these figures show a larger scale for completeness in characterizing model behavior. While this behavior can largely be extrapolated from the behavior for a smaller range of α , there are some interesting trends.

On ImageNet A.2, we see that only a small amount of inference example weighing is necessary to get most of its benefit, and setting α to larger values corresponds to a regime quite different than in training, smoothly decaying model performance as α becomes less and less appropriate. Similarly, when applying inference example weighing to Group Normalization (Fig. A.3, while performance intuitively decays as α moves farther and farther away from 1, a surprisingly large range of values for α result in similar performance to Group Normalization, especially on SVHN. Lastly, when comparing the effect of α on models trained with Ghost Batch Normalization (Fig. A.4, we clearly see that the optimal value for α is decreasing with respect to the Ghost Batch Normalization size, with the possible unusual exception of optimizing for loss on SVHN.

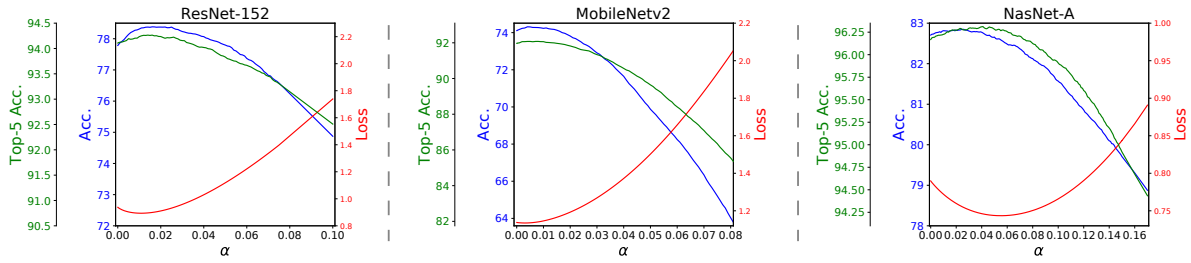


Figure A.2: Effect of the example-weighting hyperparameter α on ImageNet; supplemental version of Fig. 4.1 with a larger range of α .

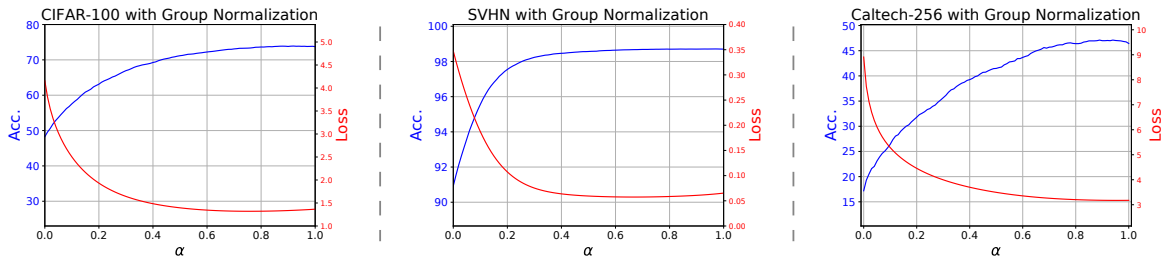


Figure A.3: Effect of the example-weighting hyperparameter α for models trained with Group Normalization on CIFAR-100, SVHN, and Caltech-256; supplemental version of Fig. 4.2 with a larger range of α .

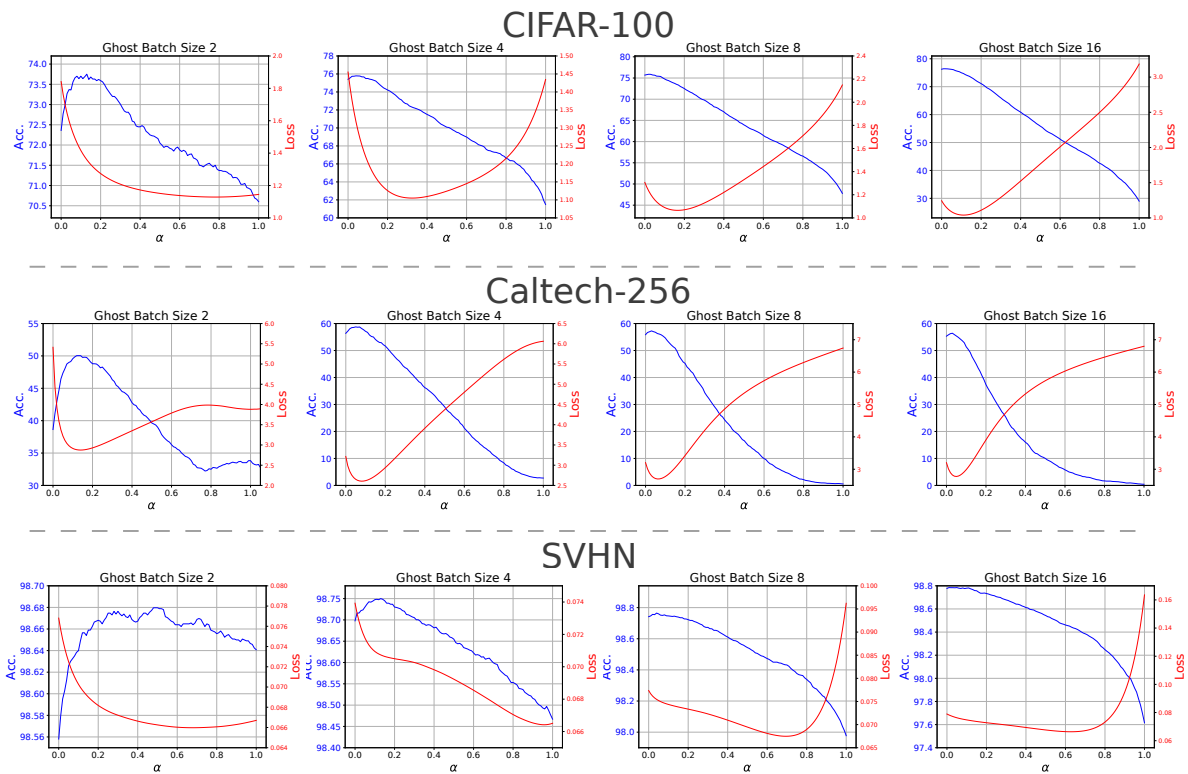


Figure A.4: The complementary effects of Inference Example Weighing and Ghost Batch Normalization on CIFAR-100, SVHN, and Caltech-256; supplemental version of Fig. 4.4 with a larger range of α .

B. Appendix: Nondeterminism and Instability

B.1 Linear, 2-Layer, and ResNet-10 Results

In Table B.1 we include results for linear networks, 2-layer networks (1 hidden layer), and a ResNet-10 on CIFAR-10. As noted in Sec. 5.4.2, parameter initialization generally has less effect for linear models, and random bit changes in particular have nearly no effect on linear models, highlighting the stability of SGD in optimizing linear models. Also of note is the relative smaller effect of a single bit change for a 2-layer network where the hidden layer is convolutional — still much larger than for the linear model, but significantly smaller than for any other non-linear model. This suggests a similar effect to what was previously observed on MNIST (Table 5.6) in that degree of instability might be related to the interplay of model, dataset, and the degree of overfitting.

Otherwise, these results follow the previous results, wherein each source of nondeterminism has roughly the same effect as each other, and a majority of this is due to the instability of optimization, evidenced by the high variability in models with only random bit changes at initialization. Test-time augmentation also remains effective in reducing model variability as compared to “All Nondeterminism Sources”, the setting TTA is applied to.

B.2 Impact of Random Bit Changes Over Time

In Fig. B.1 we plot the effect of a random bit change for a linear and single on CIFAR-10, illustrating the effect of instability as described in Sec. 5.4. In the first few epochs of training, we observe that the

Table B.1: Linear, 2-layer, and ResNet-10 experiments on CIFAR-10.

Nondeterminism Source	Accuracy SD (%)	Cross-Entropy SD	Pairwise Disagree (%)	Pairwise Corr.	Ensemble Δ (%)
CIFAR-10: Linear model					
Parameter Initialization	$0.03 \pm 2e-3$	$0.0002 \pm 1e-5$	0.5	0.997	-4e-3
All Nondeterminism Sources	0.10 ± 0.01	$0.0007 \pm 4e-5$	5.5	0.996	0.06
Random Bit Change	0.00 ± 0.00	$1e-7 \pm 1e-8$	0.0	1.000	0.00
Single/Flip-Crop-TTA	$0.05 \pm 3e-3$	$0.0001 \pm 1e-5$	0.9	0.998	-3e-3
CIFAR-10: One hidden layer (fully-connected)					
Parameter Initialization	0.31 ± 0.02	0.0051 ± 0.0003	24.2	0.941	1.38
All Nondeterminism Sources	0.30 ± 0.02	0.0054 ± 0.0004	24.9	0.937	1.49
Random Bit Change	0.28 ± 0.02	0.0051 ± 0.0004	23.4	0.945	1.32
Single/Flip-Crop-TTA	0.15 ± 0.01	0.0017 ± 0.0001	7.1	0.993	0.15
CIFAR-10: One hidden layer (convolutional)					
Parameter Initialization	0.26 ± 0.02	0.0040 ± 0.0003	12.5	0.974	0.64
All Nondeterminism Sources	0.22 ± 0.01	0.0042 ± 0.0003	12.7	0.973	0.68
Random Bit Change	0.14 ± 0.01	0.0022 ± 0.0002	6.4	0.993	0.18
Single/Flip-Crop-TTA	0.19 ± 0.01	0.0033 ± 0.0002	7.5	0.989	0.24
CIFAR-10: ResNet-10					
Parameter Initialization	0.23 ± 0.01	0.0060 ± 0.0003	13.7	0.912	2.13
All Nondeterminism Sources	0.23 ± 0.01	0.0065 ± 0.0004	13.6	0.911	2.13
Random Bit Change	0.25 ± 0.02	0.0065 ± 0.0005	13.5	0.913	2.08
Single/Flip-Crop-TTA	0.24 ± 0.02	0.0047 ± 0.0003	9.5	0.943	1.22
Acc. Ens.	0.23 ± 0.01	0.0047 ± 0.0003	8.8	0.962	0.96
Acc. Ens./Flip-Crop-TTA	0.18 ± 0.01	0.0035 ± 0.0002	6.7	0.973	0.58

standard deviation and range of cross-entropy for the model with one hidden layer quickly grows, only eventually decreasing much later in training as the model’s parameters converges toward their final values. On the other hand, for linear models, the standard deviation consistently remains 5 or more orders of magnitude lower throughout training.

B.3 Test-Time Augmentation Details

CIFAR-10. On CIFAR-10, in addition to TTA with horizontal flipping (*i.e.* ensembling model predictions on the original image with its horizontally-flipped version), we also used a form of TTA with cropping. Our usage of crop-based TTA was based on the version of cropping used as data augmentation during model training, in which each image was zero-padded by four pixels along each side, after which a random 32×32 crop was drawn. For TTA, we use all of these possible crops, along with their horizontally-flipped versions, to make a total of $162 ((4 \cdot 2 + 1)^2 \cdot 2)$ augmented versions of the original image.

ImageNet. On ImageNet, the standard evaluation protocol we use for our experiments first resizes each image to have its smaller side have length 256, after which the center 224×224 crop is taken. For TTA, besides the horizontal image flipping used in CIFAR-10, we also experimented with crops (‘Crop-TTA’ and ‘Flip-Crop-TTA’) as follows: after each image is resized to have its smaller side length 256, a central 256×256 crop is taken, and then 9 crops of size 224×224 are taken in a 3×3 grid, starting from the top-left, and where the spacing between crops in the grid is 16 pixels.

B.4 Approaches that don’t reduce instability

In the process of finding an approach that reduces run-to-run variability of models (Sec. 5.5), we experimented with many approaches which all failed to make a dent in improving variability and stability. For the benefit of the field, here we provide our experiences with these approaches which did not succeed in improving stability, despite the intuitive arguments for why they might help.

Learning rate and duration of training. Noticing that the effects of nondeterminism seemed to accumulate during the course of training (Fig. 5.2), it seemed reasonable that varying the learning rate

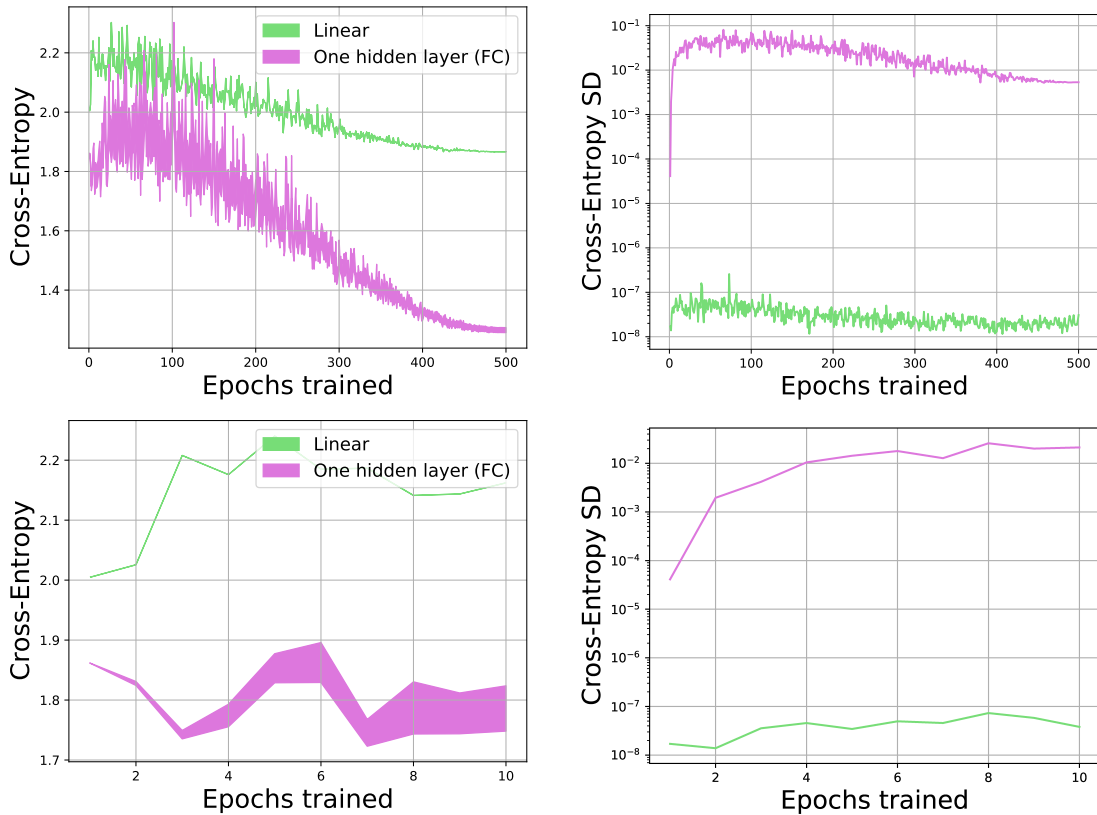


Figure B.1: The impact of a random bit change during initialization for linear models vs 2-layer models with a single fully-connected hidden layer, where row 1 considers the full 500 epochs of training, and row 2 zooms in on the first 10 epochs. The left column of each row gives the range of cross-entropy values for 100 models in the middle 95th percentile of cross-entropy, plotted at each epoch. The right column of each row presents the standard deviation of these models.

or duration of training might have an effect. However, varying the duration of training from anywhere between 50 and 2,000 epochs on CIFAR-10 all produced models with a similar variance in performance as the results in the rest of this work (which used 500 epochs), even though the absolute performance differed by up to $\sim 2\%$.

We show these results in Table B.2. In general, increasing the number of epochs or changing the learning rate did not change the variability in performance (Accuracy SD; Cross-Entropy SD) much, with only a very slight increase in variability as the number of epochs grew to extremely large values (*i.e.* 2,000 epochs). There were slightly larger changes in pairwise representation-based metrics, where training longer again increased run-to-run variability. However, none of these attempts actually reduced variability; they only served to potentially make it larger.

As part of these experiments, we also verified the effects of instability with only 200 epochs of training and the effectiveness of accelerated ensembling techniques (“Acc. Ens.”) with this reduced training time, given in the last three rows of Table B.2.

Using a different optimizer. Since instability and nondeterminism are both a property of optimization, it is conceivable that use of a different optimizer might be able to lessen the degree of instability in model training. We experimented with SGD using various values of momentum, ranging from 0 for pure SGD to 0.999 for a momentum-driven optimizer, but none succeeded in reduce instability. In addition, we experimented with Adam [55], picked as a representative of the class of adaptive learning rate algorithms, but this, too, had no effect on stability.

Aggressive Stochastic Weight Averaging. Inspired by the success found by Madhyastha and Jain, we tried Aggressive Stochastic Weight Averaging (ASWA), a variant of SWA [53]. However, we could not get the model to converge to a reasonable degree of performance with the original formulation due to update sizes that decreased too rapidly, and though we were able to modify it to converge successfully, the output variance remained as high as the other models.

Gradient Clipping. With the intuition that instability might be caused by spurious gradients of large magnitude, we experimented with clipping the norm of gradients (using `pytorch`’s implementation of `torch.nn.utils.clip_grad.clip_grad_norm_`). Like other approaches, though, this had no effect on model variability.

Table B.2: Experiments varying the learning rate and number of epochs for ResNet-14 on CIFAR-10. In each row, the experimental setting is abbreviated by [sources of nondeterminism]/[maximum learning rate]/[number of epochs] N =[number of models trained], with the exception of the last row, which is a Snapshot ensemble but otherwise follows the same format.

Setting	Accuracy SD (%)	Cross-Entropy SD	Pairwise Disagree (%)	Pairwise Corr.	Ensemble Δ (%)
All Sources/.40/50 ($N=20$)	0.31 ± 0.04	0.0070 ± 0.0007	11.4	0.922	1.54
All Sources/.40/100 ($N=20$)	0.26 ± 0.04	0.0068 ± 0.0015	10.8	0.909	1.71
All Sources/.40/250 ($N=20$)	0.19 ± 0.03	0.0054 ± 0.0009	10.7	0.889	1.78
All Sources/.40/500 ($N=100$)	0.26 ± 0.02	0.0072 ± 0.0005	10.7	0.871	1.82
All Sources/.40/2000 ($N=20$)	0.24 ± 0.02	0.0096 ± 0.0013	11.2	0.828	2.08
Shuffle/.40/500 ($N=100$)	0.25 ± 0.02	0.0082 ± 0.0005	10.6	0.871	1.81
Shuffle/.20/500 ($N=100$)	0.23 ± 0.02	0.0071 ± 0.0005	11.0	0.858	1.95
Shuffle/.20/1000 ($N=100$)	0.21 ± 0.02	0.0088 ± 0.0005	11.1	0.837	2.02
Shuffle/.10/500 ($N=100$)	0.20 ± 0.01	0.0076 ± 0.0005	11.6	0.845	2.08
Shuffle/.10/2000 ($N=100$)	0.24 ± 0.02	0.0100 ± 0.0006	11.6	0.801	2.19
Param. Init/.40/500 ($N=100$)	0.23 ± 0.02	0.0074 ± 0.0005	10.7	0.872	1.82
Param. Init/.20/500 ($N=100$)	0.23 ± 0.02	0.0084 ± 0.0005	11.0	0.859	1.97
Param. Init/.20/1000 ($N=100$)	0.25 ± 0.02	0.0095 ± 0.0007	11.1	0.836	2.06
Param. Init/.10/500 ($N=100$)	0.26 ± 0.02	0.0083 ± 0.0005	11.7	0.844	2.13
Param. Init/.10/2000 ($N=100$)	0.22 ± 0.01	0.0093 ± 0.0008	11.6	0.800	2.18
All Sources/.40/200 ($N=100$)	0.23 ± 0.02	0.0076 ± 0.0004	10.6	0.895	1.75
Random Bit/.40/200 ($N=100$)	0.21 ± 0.01	0.0067 ± 0.0004	10.3	0.897	1.70
Acc. Ens./All Sources/.40/200 ($N=100$)	0.21 ± 0.01	0.0046 ± 0.0003	6.6	0.963	0.68

Table B.3: The effects of accelerated model ensembling on Penn Treebank; 100 runs per row. “Acc. Ens.” indicates accelerated ensembling, and the trailing number in each setting name is the number of epochs models are trained for.

Setting	PPL SD	Pairwise Disagree (%)	Ensemble PPL Δ
All Nondeterminism Sources/500	0.18 ± 0.01	17.4	-2.07
Acc. Ens./All Sources/500	0.21 ± 0.02	13.7	-1.33
All Nondeterminism Sources/1000	0.17 ± 0.01	17.6	-2.08
Acc. Ens./All Sources/1000	0.16 ± 0.01	14.1	-1.34

Weight Augmentation. A very experimental approach, to reduce instability we experimented with taking an averaged gradient around the current set of parameters at each step, approximated by sampling a random weight offset before doing a forward or backward pass through the model. Intuitively, this might encourage optimization to not be too sensitive to the current value of weights; however, in practice this simply didn’t affect the variance or stability of the model.

B.5 Accelerated Ensembling in Language Modeling

Although the accelerated ensembling technique we employed in Sec. 5.5, Snapshot Ensembles [45], were only designed for image classification, we have also experimented with their usage for language modeling (see Sec. 5.3.3 for problem setup). We present results in Table B.3, where we additionally compare models trained for 500 and 1000 epochs. In both cases, accelerated ensembling resulted in lower model variability when considering pairwise metrics (reducing the fraction of tokens models disagreed on and reducing the PPL improvement from ensembling). However, the variability in PPL was more mixed, and in fact we note that the accelerated ensembles actually had higher average PPL than their counterparts (*e.g.* 75.0 for the accelerated ensemble vs 73.0 for the regular model), indicating that alternative accelerated ensembling techniques may be warranted for language modeling.

B.6 Subtleties in Evaluation

While we have done our best to make our experimental protocol straightforward and easy to interpret, one subtlety arises when interpreting results. First, recall that in our example from Sec. 5.3.1, testing

the effects of random initialization corresponded to training models for $(S_1, S_2, S_3) \in \{(i, 1, 1)\}_{i=1}^R$, where S_1 is the seed for random initialization, S_2 is the seed for training data shuffling, and S_3 is set to 1 to indicate the deterministic mode for cuDNN. The subtlety arises in that the resulting distribution of $(S_1, S_2, S_3) \in \{(i, 1, 1)\}_{i=1}^R$ is not necessarily the same as the distribution where S_2 is set to a different arbitrary constant value, *e.g.* $S_2 = 2$. Due to this, there may be minor discrepancies when comparing the diversity in performance between two different sources of nondeterminism (albeit unlikely to change general conclusions). Combined with the natural sampling variability implicit in only training a finite number of models, this can lead to paradoxical results such as the standard deviation for a particular metric being slightly higher for a random bit change as compared to an entirely different random parameter initialization. While we have separately validated that the general conclusions of our results hold when varying a few of these constant factors (*i.e.* running experiments where S_2 is set to 2 and 3, in this example), it is difficult to resolve the discrepancy entirely without models according to the full cross-product of random seeds, which is prohibitive due to the exponential number of required computation.

References

- [1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.
- [2] Alessandro Achille, Matteo Rovere, and Stefano Soatto. Critical learning periods in deep neural networks. In *International Conference on Learning Representations*, 2019.
- [3] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning*, 2018.
- [4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.
- [5] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.
- [6] Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of machine learning research*, 2(Mar):499–526, 2002.
- [7] James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural networks. *arXiv preprint arXiv:1611.01576*, 2016.
- [8] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

- [9] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.
- [10] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.
- [11] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
- [12] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–108. ACM, 2004.
- [13] Lucas Deecke, Iain Murray, and Hakan Bilen. Mode normalization. In *International Conference on Learning Representations*.
- [14] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.
- [15] Logan Engstrom, Andrew Ilyas, and Anish Athalye. Evaluating and understanding the robustness of adversarial logit pairing. *arXiv preprint arXiv:1807.10272*, 2018.
- [16] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.
- [17] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115, 2017.
- [18] Kevin Eykholt, Ivan Evtimov, Earlene Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1625–1634, 2018.

- [19] Samuel G. Finlayson, John D. Bowers, Joichi Ito, Jonathan L. Zittrain, Andrew L. Beam, and Isaac S. Kohane. Adversarial attacks on medical machine learning. *Science*, 363(6433):1287–1289, 2019.
- [20] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*, 2019.
- [21] Jonathan Frankle, David J Schwab, and Ari S Morcos. The early phase of neural network training. In *International Conference on Learning Representations*, 2020.
- [22] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems*, pages 8789–8798, 2018.
- [23] Xavier Gastaldi. Shake-shake regularization. *arXiv preprint arXiv:1705.07485*, 2017.
- [24] Justin Gilmer, Ryan P Adams, Ian Goodfellow, David Andersen, and George E Dahl. Motivating the rules of the game for adversarial example research. *arXiv preprint arXiv:1807.06732*, 2018.
- [25] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.
- [26] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.
- [27] 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.
- [28] 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.
- [29] Gregory Griffin, Alex Holub, and Pietro Perona. Caltech-256 object category dataset. 2007.
- [30] Yanyang Gu, Jun Zhou, and Bin Qian. Melanoma detection based on mahalanobis distance learning and constrained graph regularized nonnegative matrix factorization. In *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*, pages 797–805. IEEE, 2017.

- [31] Varun Gulshan, Lily Peng, Marc Coram, Martin C Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, et al. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316(22):2402–2410, 2016.
- [32] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2017.
- [33] 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.
- [34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
- [35] 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.
- [36] 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.
- [37] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [38] Nicholas J Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002.
- [39] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.
- [40] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

- [41] 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.
- [42] 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.
- [43] 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.
- [44] 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.
- [45] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.
- [46] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, pages 646–661. Springer, 2016.
- [47] Lei Huang, Dawei Yang, Bo Lang, and Jia Deng. Decorrelated batch normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 791–800, 2018.
- [48] Lei Huang, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Iterative normalization: Beyond standardization towards efficient whitening. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4874–4883, 2019.
- [49] Hiroshi Inoue. Data augmentation by pairing samples for images classification. *arXiv preprint arXiv:1801.02929*, 2018.
- [50] Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Advances in Neural Information Processing Systems*, pages 1945–1953, 2017.

- [51] 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.
- [52] Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.
- [53] Pavel Izmailov, Dmitrii Podoprikin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.
- [54] Harini Kannan, Alexey Kurakin, and Ian Goodfellow. Adversarial logit pairing. *arXiv preprint arXiv:1803.06373*, 2018.
- [55] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [56] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980, 2017.
- [57] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. *arXiv preprint arXiv:1905.00414*, 2019.
- [58] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.
- [59] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [60] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.
- [61] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.
- [62] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *null*, pages 2169–2178. IEEE, 2006.

- [63] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
- [64] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [65] 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.
- [66] 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.
- [67] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.
- [68] Ping Luo, Jiamin Ren, and Zhanglin Peng. Differentiable learning-to-normalize via switchable normalization. In *International Conference on Learning Representations*.
- [69] Ping Luo, Peng Zhanglin, Shao Wenqi, Zhang Ruimao, Ren Jiamin, and Wu Lingyun. Differentiable dynamic normalization for learning deep representation. In *International Conference on Machine Learning*, pages 4203–4211, 2019.
- [70] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.
- [71] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.
- [72] Pranava Madhyastha and Rishabh Jain. On model stability as a function of random seed. In *Conference on Computational Natural Language Learning*, pages 929–939, 2019.
- [73] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

- [74] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. 1993.
- [75] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.
- [76] Yair Movshovitz-Attias, Alexander Toshev, Thomas K Leung, Sergey Ioffe, and Saurabh Singh. No fuss distance metric learning using proxies. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 360–368, 2017.
- [77] Asim Munawar, Phongtharin Vinayavekhin, and Giovanni De Magistris. Spatio-temporal anomaly detection for industrial robots through prediction in unsupervised feature space. In *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*, pages 1017–1025. IEEE, 2017.
- [78] Prabhat Nagarajan, Garrett Warnell, and Peter Stone. The impact of nondeterminism on reproducibility in deep reinforcement learning. 2018.
- [79] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.
- [80] M-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.
- [81] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4004–4012, 2016.
- [82] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. Technical report on the cleverhans v2.1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*, 2018.

- [83] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.
- [84] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. In *International Conference on Learning Representations*, 2017.
- [85] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.
- [86] 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.
- [87] 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.
- [88] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. In *International Conference on Learning Representations*, 2018.
- [89] 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.
- [90] 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.
- [91] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. In *Advances in Neural Information Processing Systems*, 2018.

- [92] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, 2020.
- [93] 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.
- [94] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.
- [95] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 761–769, 2016.
- [96] N. Silberman and S. Guadarrama. Tensorflow-slim image classification model library. <https://github.com/tensorflow/models/tree/master/research/slim>.
- [97] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.
- [98] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*.
- [99] Saurabh Singh and Abhinav Shrivastava. Evalnorm: Estimating batch normalization statistics for evaluation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3633–3641, 2019.
- [100] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In *International Conference on Learning Representations*, 2018.
- [101] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

- [102] Cecilia Summers and Michael J Dinneen. Improved adversarial robustness via logit regularization methods. *arXiv preprint arXiv:1906.03749*, 2019.
- [103] Cecilia Summers and Michael J Dinneen. Improved mixed-example data augmentation. In *IEEE Winter Conference on Applications of Computer Vision*, 2019.
- [104] Cecilia Summers and Michael J Dinneen. Four things everyone should know to improve batch normalization. In *International Conference on Learning Representations*, 2020.
- [105] Cecilia Summers and Michael J Dinneen. Nondeterminism and instability in neural network optimization. In *International Conference on Machine Learning*, 2021.
- [106] 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.
- [107] 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.
- [108] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [109] 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.
- [110] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.
- [111] Yuji Tokozume, Yoshitaka Ushiku, and Tatsuya Harada. Between-class learning for image classification. In *Computer Vision and Pattern Recognition*, 2018.
- [112] Yuji Tokozume, Yoshitaka Ushiku, and Tatsuya Harada. Learning from between-class examples for deep sound recognition. In *International Conference on Learning Representations*, 2018.

- [113] Jonathan Uesato, Brendan O’Donoghue, Aaron van den Oord, and Pushmeet Kohli. Adversarial risk and the dangers of evaluating against weak attacks. *arXiv preprint arXiv:1802.05666*, 2018.
- [114] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.
- [115] Twan van Laarhoven. L2 regularization versus batch and weight normalization. *arXiv preprint arXiv:1706.05350*, 2017.
- [116] 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.
- [117] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.
- [118] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.
- [119] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066, 2013.
- [120] Yeming Wen, Dustin Tran, and Jimmy Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. In *International Conference on Learning Representations*, 2020.
- [121] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.
- [122] 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.
- [123] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017.

- [124] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.
- [125] Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. In *International Conference on Learning Representations*.
- [126] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896*, 2017.
- [127] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*.
- [128] 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.