MixUp as Locally Linear Out-of-Manifold Regularization

Hongyu Guo
National Research Council Canada
1200 Montreal Road, Ottawa
hongyu.guo@nrc-cnrc.gc.ca

Yongyi Mao
School of Electrical Engineering & Computer Science
University of Ottawa, Ottawa, Ontario
yymao@eecs.uottawa.ca

Richong Zhang
BDBC, School of Computer Science and Engineering
Beihang University, Beijing, China
zhangrc@act.buaa.edu.cn

Abstract

MixUp (Zhang et al. 2017) is a recently proposed data-augmentation scheme, which linearly interpolates a random pair of training examples and correspondingly the one-hot representations of their labels. Training deep neural networks with such additional data is shown capable of significantly improving the predictive accuracy of the current art. The power of MixUp, however, is primarily established empirically and its working and effectiveness have not been explained in any depth. In this paper, we develop an understanding for MixUp as a form of "out-of-manifold regularization", which imposes certain "local linearity" constraints on the model's input space beyond the data manifold. This analysis enables us to identify a limitation of MixUp, which we call "manifold intrusion". In a nutshell, manifold intrusion in MixUp is a form of under-fitting resulting from conflicts between the synthetic labels of the mixed-up examples and the labels of original training data. Such a phenomenon usually happens when the parameters controlling the generation of mixing policies are not sufficiently fine-tuned on the training data. To address this issue, we propose a novel adaptive version of MixUp, where the mixing policies are automatically learned from the data using an additional network and objective function designed to avoid manifold intrusion. The proposed regularizer, AdaMixUp, is empirically evaluated on several benchmark datasets. Extensive experiments demonstrate that AdaMixUp improves upon MixUp when applied to the current art of deep classification models.

Introduction

Deep learning techniques have achieved profound success in many challenging real-world applications, including image recognition (Krizhevsky, Sutskever, and Hinton 2012; He et al. 2016), speech recognition (Hinton et al. 2012; Graves, Mohamed, and Hinton 2013), and machine translation (Sutskever, Vinyals, and Le 2014; Bahdanau, Cho, and Bengio 2014), amongst many others (Goodfellow et al. 2014; Silver et al. 2016; Kipf and Welling 2016). A recently proposed such technique, MixUp (Zhang et al. 2017), is a simple and yet very effective data-augmentation approach to enhance the performance of deep classification models. Through linearly interpolating random data sample pairs and their training targets, MixUp generates a synthetic set of examples and use these examples to augment the training set. In (Zhang et al. 2017), MixUp is shown to dramatically improve the predictive accuracy of the current art of deep neural networks.

Despite its demonstrated effectiveness, the power of MixUp is mostly established empirically. To date, the working of MixUp has not been well explained. In addition, the mixing (i.e., interpolation) policies in MixUp are controlled by a global hyper-parameter α, which needs to be tuned by trial and error on the data set. This on one hand makes MixUp inconvenient to use, and on the other hand, adds to the mystery what role such a parameter controls and how to tune it properly.

issue 1

The study of this current paper is motivated by developing a deeper understanding of MixUp, specifically pertaining to why it works. To that end, we formulate MixUp as a new form of regularization. Specifically, we categorize regularization schemes as *data-independent* regularization and *data-dependent* regularization. Data-independent regularization imposes constraints on the model without exploiting the structure (e.g., the distribution) of data; typical examples include penalizing various forms of the norm of the network parameters (e.g., weight decay (Hanson and Pratt 1988)) or dropout (Srivastava et al. 2014). Data-dependent regularization constrains the parameter space of the model in a way that depends on the structure of the data; previous examples of such schemes include, data augmentation schemes (Simard et al. 1998; Lecun et al. 1998; Simonyan and Zisserman 2014), adversarial training schemes (e.g., (Goodfellow, Shlens, and Szegedy 2014)), and some information bottleneck based regularization schemes (e.g., (Alemi et al. 2016)). In this paper, we show that MixUp is also a data-dependent regularization scheme in the sense that the imposed constraints on the model explicitly exploit

regulariza
tion

the data distribution. But different from all previous data-dependent regularization schemes, MixUp imposes its constraints by making use of the regions in the input space of the model that are outside of the data manifold. Identifying the constraints imposed by MixUp with a set of "locally linear" constraints, we call such a data-dependent regularization scheme a "locally linear out-of-manifold" regularization scheme.

Under this perspective, we identify an intrinsic problem in MixUp, which we refer to as "manifold intrusion". Briefly, manifold intrusion occurs when a mixed example collides with a real example in the data manifold, but is given a soft label that is different from the label of the real example. Figure 1 (left) shows some examples of manifold intrusion. In the figure, the mixed images (top row) look close to a hand-written "8" (which indeed live in the data set) and yet MixUp assigns a soft-label that is not "8". For example, the soft-label of the top-left image is given as the half-half mix of labels "1" and "5". We explain in this work that manifold intrusion necessarily results in under-fitting and degradation of the model performance. This is confirmed by our experiments, where we see MixUp with an inappropriate hyper-parameter α actually deteriorates the model performance (Figure 1, right). Indeed, with MixUp, one has to select hyper-parameter α carefully via trial and error, in order to enhance the base model.

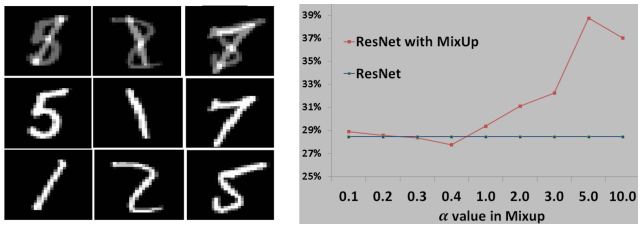

Figure 1: Left: Linearly interpolated images (top row) from the original images (bottom two rows). Right: Performance of MixUp on a reduced Cifar100 data set (reduced to containing 20% of data samples) vs various values of α.

Built on our understanding and motivated by the potential manifold intrusion in MixUp, we then propose a generalized and adaptive version of MixUp, which we call AdaMixUp. On one hand, AdaMixUp extends the mixing policies in the standard MixUp from 2-fold to higher-fold; more importantly it adaptively learns good policy regions from the data and automatically avoids manifold intrusion. Our experiments on several image classification tasks show that AdaMixUp significantly improves the current art of deep classification models and outperforms the standard MixUp by a large margin.

Preliminaries

In a standard classification setting, let $\mathcal{X}$ denote the vector space in which each example x lives. Throughout the paper, the elements of $\mathcal{X}$ will be treated as vectors and all vectors in this paper are taken as column vectors.

Let $\mathcal{Y}$ denote the set of all class labels. The objective of our classification problem is to develop a classifier which assigns every example a label in $\mathcal{Y}$.

It is worth noting however that not every $x \in \mathcal{X}$ is a valid example. That is, some x cannot be associated to a label in $\mathcal{Y}$. Specifically, the set of all valid examples is only a subset of $\mathcal{X}$, which we refer to as the *data manifold*, or in short, the manifold, and denote it by $\mathcal{M}$.

The following hypothesis is adopted for nearly every classification problem.

Hypothesis 1 (Basic Hypothesis) *Each $x \in \mathcal{M}$ has a* unique *label in $\mathcal{Y}$. We use $g(x)$ to denote the label of x, where g is a function mapping $\mathcal{M}$ onto $\mathcal{Y}$. We stress that on $\mathcal{X}$ outside $\mathcal{M}$, g is not defined.*

In the classification problem, we are given a subset $\mathcal{D} \subset \mathcal{M}$ as the training examples, where for each $x \in \mathcal{D}$, $g(x)$ is given.

A distribution over a set of m elements will be denoted by a vector in $\mathbb{R}^m$. It is well known that the set of all such distributions form the *probability simplex* in $\mathbb{R}^m$, which we will denote by $\mathbb{S}_m$. The set of all distributions on $\mathcal{Y}$ is obviously a probability simplex, but we will give it a separate notation $\mathcal{P}(\mathcal{Y})$ for distinction. Specifically, each distribution in $\mathcal{P}(\mathcal{Y})$ is regarded as a "soft label", whereas each distribution in other probability simplexes potentially serves as a set of coefficients, which will be used to mix training examples via a convex combination. For example, a Bernoulli distribution $[\alpha, 1-\alpha]^T$ as a distribution in $\mathbb{S}_2$ may potentially be used to mix two examples $x, x' \in \mathcal{X}$ by $\alpha x + (1-\alpha)x'$. Thus the distributions in these simplexes will be referred to as a *mixing policy*, or simply *policy*. Any subset Λ of these probability simplex with non-zero Lebesgue measure will then be called a *policy region*. A distribution or policy will be called *degenerate* if it is a distribution containing a single non-zero probability mass.

单纯形

Let $\mathcal{F}(\mathcal{X}, \mathcal{Y})$ denote the space of all functions mapping $\mathcal{X}$ to $\mathcal{P}(\mathcal{Y})$. A *discriminative classification model* (such as a neural network model) that outputs a predictive distribution over $\mathcal{Y}$ for an example $x \in \mathcal{X}$ can be characterized as a subset $\mathcal{H} \subset \mathcal{F}(\mathcal{X}, \mathcal{Y})$

Given the model $\mathcal{H}$, the objective of learning is then finding a member function $H \in \mathcal{H}$ which hopefully, for each $x \in \mathcal{M}$, returns a predictive distribution that puts highest probability on $g(x)$. Usually such an H is found by optimizing a well-defined loss function $\mathcal{L}_{\mathcal{D}}(H)$ over all H in the space $\mathcal{H}$

$$\widehat{H} := \arg\min_{H \in \mathcal{H}} \mathcal{L}_{\mathcal{D}}(H). \tag{1}$$

We note that we subscript the loss with $\mathcal{D}$ to emphasize its dependency on the training data $\mathcal{D}$.

Regularization

It is well known that when the space $\mathcal{H}$ is large (for example, high dimensional), or when the model is too complex, the model tends to overfit and generalize poorly. The main techniques to cure such a problem is *regularization*.

Regularization, in our view, may refer to any technique that effectively reduces the model capacity or reduces the

space $\mathcal{H}$. In this paper, we wish to divide regularization into two categories: *data-dependent* and *data-independent*.

Data-independent regularization is the usual notion of regularization. Typical such techniques directly impose certain constraint, say, $C(H) < A$, on the functions $H \in \mathcal{H}$. This turns the optimization problem (1) to a constrained optimization problem, which, under the Lagrangian formulation, can be expressed by

$$\widehat{H} := \arg\min_{H\in\mathcal{H}} \{\mathcal{L}_{\mathcal{D}}(H) + \beta C(H)\} \tag{2}$$

For example, the weight-decay regularization is a data-independent regularization, which essentially imposes a L2-norm constraint on the model parameters. Dropout (Srivastava et al. 2014) can be regarded as another example of such a regularizer, which can be understood as reducing the model capacity in a Bayesian style (Gal and Ghahramani 2015) and which has an equivalent loss function similar to the form of (2). A key feature in data-independent regularization is that the training data do not enter the regularization term $C(H)$ in such a scheme.

Data-dependent regularization imposes constraints or makes additional assumptions on $\mathcal{H}$ with respect to the training data $\mathcal{D}$.

The most well known such techniques are data augmentation. Specifically, in a data augmentation scheme, another set $\mathcal{D}' \subset \mathcal{X}$, *believed to be inside* $\mathcal{M}$ but beyond the training set $\mathcal{D}$, are introduced for training. For example, if $\mathcal{D}$ is a set of images, $\mathcal{D}'$ can be obtained by rotating each image in $\mathcal{D}$; the label $g(x)$ of an image x in $\mathcal{D}'$ is taken as the label of the image x before rotation. Data augmentation turns the optimization problem (1) into

$$\widehat{H} := \arg\min_{H\in\mathcal{H}} \{\mathcal{L}_{\mathcal{D}}(H) + \mathcal{L}_{\mathcal{D}'}(H)\} \tag{3}$$

where $\mathcal{L}_{\mathcal{D}'}$ is the same loss function but on data $\mathcal{D}'$. Aligning (3) with (2), one can equivalently regard data augmentation as imposing a constraint on the space $\mathcal{H}$ (hence a regularization scheme), and such a constraint, depending on the data set $\mathcal{D}$, is precisely due to the hypothesis that $\mathcal{D}' \subset \mathcal{M}$ and $g(\cdot)$ is rotation-invariant.

Another data-dependent regularization scheme is adversarial training (see, e.g., (Goodfellow, Shlens, and Szegedy 2014)). In such a scheme, for each training example x, an adversarial example x' is found, and it is assumed that x' has the same label as x. The adversarial examples are then also used for training purpose. Although delicate procedures have been proposed to find adversarial examples, the found adversarial examples are used in the same way as the additional data set $\mathcal{D}'$ in data augmentation.

In both data augmentation and adversarial training, the constraints imposed on the label function g is in the data manifold; the constraints essentially specify how function g should behave on additional points in the manifold $\mathcal{M}$. Next we will argue that MixUp may be viewed as another data-dependent regularization scheme, which introduces constraints on g outside the manifold $\mathcal{M}$.

Locally Linear Out-of-Manifold Regularization

For each $y \in \mathcal{Y}$, let $\delta_y \in \mathcal{P}(\mathcal{Y})$ be the single-point-mass (namely, degenerate) distribution on $\mathcal{Y}$ which puts all the probability mass on y. Now we associate with g a function $G : \mathcal{X} \to \mathcal{P}(\mathcal{Y})$ satisfying

$$G(x) := \delta_{g(x)}, \text{for any } x \in \mathcal{M}. \tag{4}$$

Obviously, on the data manifold $\mathcal{M}$, G is simply a representation of g in $\mathcal{F}(\mathcal{X}, \mathcal{Y})$. Additionally any $\mathcal{G}$ satisfying the above equation is as perfect as g for any valid example. Thus, any such function G can be taken as a "ground-truth" classifier.

Then the classification problem can be reformulated as constructing a model $\mathcal{H} \subset \mathcal{F}(\mathcal{X}, \mathcal{Y})$ and finding a member $H \in \mathcal{H}$ that well approximates a function G satisfying (4).

Noting that the condition (4) on G is only within $\mathcal{M}$, and it imposes no condition on the space $\mathcal{X}$ outside $\mathcal{M}$. However, the functions in $\mathcal{H}$ are defined on $\mathcal{X}$, well beyond $\mathcal{M}$. Thus, if one chooses to regularize the model in a data-dependent manner, there is a lots of degrees of freedom beyond $\mathcal{M}$ which one can exploit. We will show that the MixUp strategy presented in (Zhang et al. 2017) can be seen as such a technique.

To begin, for any subset $\Omega \subseteq \mathcal{X}$, we will use $\Omega^{(k)}$ to denote the set of all k-column matrices in which each column is a point (vector) in Ω. This also defines notations $\mathcal{M}^{(k)}$ and $\mathcal{D}^{(k)}$; however with these latter two notations, we require each matrix in $\mathcal{M}^{(k)}$ and $\mathcal{D}^{(k)}$ satisfy an additional condition, namely, *the labels of the columns of the matrix must be all distinct*.

Let k be a given integer not less than 2. Let $\Lambda \subseteq \mathbb{S}_k$ be a policy region, and $\mathbf{X} \in \mathcal{X}^{(k)}$. We say that a function $F \in \mathcal{F}(\mathcal{X}, \mathcal{Y})$ is Λ-*linear with respect to* $\mathbf{X}$ if for any $\lambda \in \Lambda$

$$F(\mathbf{X}\lambda) = F(\mathbf{X})\lambda$$

where $F(\cdot)$, when having its domain extended to matrices, acts on matrix $\mathbf{X}$ column-wise. We also refer to such a notion of linearity as a *k-fold local linearity*. We note that if we allow Λ to $\mathbb{R}^k$, then $\mathbb{R}^k$-linearity with respect to all $\mathbf{X} \in \mathcal{X}^{(k)}$ for every integer $k \geq 1$ implies the standard linearity.

Standard MixUp

Hypothesis 2 (Standard MixUp Hypothesis) *There exists a policy region $\Lambda \subset \mathbb{S}^2$ such that for any matrix $\mathbf{X} \in \mathcal{M}^{(2)}$, the function G is Λ-linear with respect to $\mathbf{X}$.*

We note that this hypothesis only assumes a fold-2 local linearity and a global choice of Λ. If Hypothesis 2 holds and one is given Λ, then one can implementing the hypothesis as a constraint on $\mathcal{H}$, by imposing on the optimization problem the following constraint.

> Standard MixUp Constraint:
> Every $H \in \mathcal{H}$ is Λ-linear w.r.t. every $\mathbf{X} \in \mathcal{D}^{(2)}$,

This gives rise a regularization scheme, which can be implemented by repeating the following process: draw a random pair data points[1] in $\mathcal{D}$ to form $\mathbf{X}$, and draw a random λ from Λ; take $\mathbf{X}\lambda$ as an additional training example and $G(\mathbf{X})\lambda$ as its training target.

This scheme, generating more training examples, is essentially the standard MixUp scheme (Zhang et al. 2017).

In this paper, we however point out that one needs to caution with such a regularization scheme.

Lemma 1 *Let Λ be a policy region in $\mathbb{S}_2$ and $\mathbf{X} \in \mathcal{M}^{(2)}$. If there is a non-degenerate $\lambda \in \Lambda$ with $\mathbf{X}\lambda \in \mathcal{M}$, then Hypothesis 2 can not hold with this choice Λ*[2].

Proof: The condition of the lemma, stated more precisely in Footnote 2 suggests that there is a region $\Lambda' \subseteq \Lambda$ with non-zero Lebesgue measure such that every $\lambda \in \Lambda'$ is non-degenerate and $\mathbf{X}\lambda \in \mathcal{M}$. The fact that $\mathbf{X}\lambda \in \mathcal{M}$ suggests, by Hypothesis 1, that $g(\mathbf{X}\lambda) \in \mathcal{Y}$, or $\mathcal{G}(\mathbf{X}\lambda)$ is a single-point-mass distribution for every $\lambda \in \Lambda'$.

To induce a contradiction, suppose that Hypothesis 2 holds for Λ, namely, G is Λ-linear with respect to every $\mathbf{X} \in \mathcal{M}^{(2)}$. Then $\mathcal{G}(\mathbf{X}\lambda) = \mathcal{G}(\mathbf{X})\lambda$ for every $\lambda \in \Lambda' \subseteq \Lambda$. Since the two columns (points) in $\mathbf{X}$ have different labels, the two columns of $G(\mathbf{X})$ must be two different one-hot vectors (degenerate distributions). But every $\lambda \in \Lambda'$ is non-degenerate; as a consequence, every $G(\mathbf{X})\lambda$ must be non-degenerate. This contradicts the previous argument that $\mathcal{G}(\mathbf{X}\lambda)$ is a single-point-mass. Therefore Hypothesis 2 fails on a region in Λ with non-zero measure. □

Geometrically, the failure of Hypothesis 2 in this lemma is due to that the mixing x_1 and x_2 by λ causes the mixed point $\mathbf{X}\lambda$ to "intrude" into $\mathcal{M}$ and "collide" with a point, say, x' in $\mathcal{M}$. We call such a phenomenon *manifold intrusion*. Note that in this case, $\mathbf{X}\lambda$ and x' are in fact the same point, but they have different labels. Obviously, when such intrusion occurs, regularization using $\mathbf{X}$ and λ will contradict with the original training data. This essentially induces a form of *under-fitting*.

Figure 2 depicts the effect of MixUp on constraining the space $\mathcal{H}$, where the region $\mathcal{G}$ denotes the space of functions in $\mathcal{F}(\mathcal{X}, \mathcal{Y})$ satisfying (4).

When the space $\mathcal{H}$ is large, there is a large region of $\mathcal{H}$ satisfying (4). This gives a large intersection of $\mathcal{H}$ and $\mathcal{G}$, each member of which is a perfect classifier on the training set but performing questionably on the testing set. Thus the model without MixUp tends to result in large prediction variances or over-fitting (left figure).

When MixUp is applied and manifold intrusion does not occur (middle figure), the MixUp constraint effectively reduces the space $\mathcal{H}$. Since the MixUp constraint does not conflict (4) (at least explicitly), the reduced $\mathcal{H}$ would contain members compatible with (4). Thus the intersection of $\mathcal{G}$ and $\mathcal{H}$ reduces while remaining non-empty. Each member in the intersection is a perfect classifier on the training set and the mixed examples. But the significantly reduced intersection gives rise to significantly lower prediction variances on the testing set, thereby serving to regularize the model and reduce over-fitting.

When the application of MixUp results in manifold intrusion (right figure), there is no member in $\mathcal{H}$ that satisfies both the MixUp constraint and the condition (4). The intersection of $\mathcal{H}$ and $\mathcal{G}$ then becomes empty. In this case, no member in $\mathcal{H}$ can fit both the original training data and the (intruding) mixed examples. This gives rise to under-fitting and high prediction biases.

From the view point of restricting the space $\mathcal{H}$, one wishes to make Λ as large as possible. But this lemma sets a limit on the possible Λ that makes Hypothesis 2 hold. Due to this lemma, we need to assure that the Λ we choose does not cause such an "intrusion".

Let Λ^* denote the largest policy region in $\mathbb{S}_2$ such that manifold intrusion does not occur. In the original MixUp, recall that inappropriate hyper-parameter result in no gain and only negative impact. We believe that much of this is due to manifold intrusion. On the other hand, the search of good hyper-parameter α in the original MixUp can be seen as searching for Λ^* by trial and error.

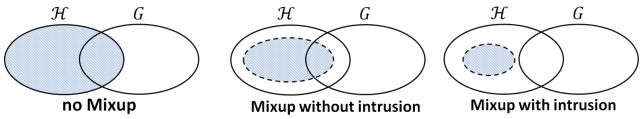

Figure 2: Effect of MixUp on constraining $\mathcal{H}$

In this paper, we generalize the standard MixUp to regularization with higher-fold local linearities, where Λ^* (more precisely its generalization) is learned from the data.

Adaptive MixUp (AdaMixUp)

Hypothesis 3 (Generalized MixUp Hypothesis) *For any integer k not less than 2 and not greater than some $k_{\max}$, and for any matrix $\mathbf{X} \in \mathcal{M}^{(k)}$, there exists a policy region $\Lambda \subset \mathbb{S}^k$ such that the function G is Λ-linear w.r.t. $\mathbf{X}$. Note that here Λ depends on $\mathbf{X}$, which we will denote by $\Lambda(\mathbf{X})$.*

If this hypothesis holds, and if for every $k \in \{2, 3, \ldots, k_{\max}\}$ and every $\mathbf{X} \in \mathcal{M}^{(k)}$, $\Lambda(\mathbf{X})$ in the hypothesis is given, then the hypothesis can be implemented as the following constraint on $\mathcal{H}$.

> Generalized MixUp Constraint:
> For every $k \in \{2, 3, \ldots, k_{\max}\}$, every $\mathbf{X} \in \mathcal{D}^{(k)}$ and every $\Lambda(\mathbf{X})$, every $H \in \mathcal{H}$ is $\Lambda(\mathbf{X})$-linear w.r.t. $\mathbf{X}$.

Imposing such a constraint in training provides another regularization scheme, which will be one ingredient of the proposed AdaMixUp scheme. We will present AdaMixUp in the next section, after developing its other ingredients.

[1] Although we have required in the definition of $\mathcal{D}^{(2)}$ that two points in matrix $\mathbf{X}$ should have different labels, this needs not to be rigorously followed in implementation. Relaxing this condition in fact is expected to decrease the chance of manifold intrusion. Similar comments apply to the AdaMixUp later.

[2] More precisely, in order to make the hypothesis fail, instead of requiring a single non-degenerate λ, we in fact require a subset of Λ having non-zero measure. But we abbreviate it here for simplicity. Similar comments apply to Lemma. 2.

Lemma 2 *Let $k \in \{2, 3, \ldots, k_{\max}\}$ and $\mathbf{X} \in \mathcal{M}^{(k)}$. Suppose that Λ is a policy region in $\mathbb{S}_k$. If there is a non-degenerate $\lambda \in \Lambda$ with $\mathbf{X}\lambda \in \mathcal{M}$, then Hypothesis 3 with $\Lambda(\mathbf{X}) = \Lambda$ can not hold.*

This lemma parallels Lemma 1 and can be proved similarly. Lemma 2 similarly suggests that when the imposed constraint results in intrusion into the manifold $\mathcal{M}$, Hypothesis 3 necessarily fails. The lemma also sets a limit for each $\Lambda(\mathbf{X})$. For each $\mathbf{X}$, we will denote the largest $\Lambda(\mathbf{X})$ by $\Lambda^*(\mathbf{X})$. The following result immediately follows.

Lemma 3 *For any $\mathbf{X} \in \mathcal{M}^{(2)}$, $\Lambda^* \subseteq \Lambda^*(\mathbf{X})$.*

Proof: By the definitions of Λ^* and $\Lambda^*(\mathbf{X})$, $\Lambda^* = \bigcap_{\mathbf{X} \in \mathcal{M}^{(2)}} \Lambda^*(\mathbf{X})$. Thus $\Lambda^* \subseteq \Lambda^*(\mathbf{X})$. □

In reality, one expects that Λ^* is strictly contained in $\Lambda^*(\mathbf{X})$. That is, even when we take $k_{\max} = 2$, the Generalized MixUp Hypothesis imposes a stronger constraint than the Standard MixUp Hypothesis, and hence can provide a stronger regularization without causing under-fitting. When $k_{\max} > 2$, the Generalized MixUp Hypothesis imposes additional constraints, which further regularizes the model and improves generalization.

AdaMixUp From here on, we stay in the Generalized MixUp regime, namely assuming Hypothesis 3 holds. We now explore the possibility of learning $\Lambda(\mathbf{X})$ for each $\mathbf{X}$.

Suppose that each $\Lambda^*(\mathbf{X}) \subset \mathbb{S}_k$ can be sufficiently parametrized by $\pi_k(\mathbf{X})$ in some space Π. Then $\pi_k(\cdot)$ can be regarded as a function mapping $\mathcal{M}^{(k)}$ to Π. Then learning each $\Lambda(\mathbf{X})$ reduces to learning the function $\pi_k(\cdot)$. We now consider using a neural network to represent the function π_k. This network will be referred to as a *policy region generator*. With a slight abuse of notation, we will use $\pi_k(\mathbf{X})$ to denote any candidate choice of $\Lambda^*(\mathbf{X}) \subset \mathbb{S}_k$ generated by the network $\pi_k(\cdot)$.

To train the networks $\{\pi_k\}$, consider the construction of another network $\varphi(\cdot)$. The network takes any $x \in \mathcal{X}$ as input and outputs a predictive distribution on whether the x lies in manifold $\mathcal{M}$.

This network, which we refer to as an *intrusion discriminator*, aims at distinguishing elements inside the manifold $\mathcal{M}$ and those outside. By assuring that the virtual examples obtained by mixing with the policies in $\pi_k(\mathbf{X})$ are outside the manifold and the original examples are inside the manifold, the supervising signal trains both the networks $\{\pi_k\}$ and φ. More details are given below.

For a given $x \in \mathcal{X}$, let $p(\cdot|x;\varphi)$ be the predictive distribution on $\{1, 0\}$ generated by network φ, where 1 denotes x is outside $\mathcal{M}$. The loss function for training $\{\pi_k\}$ and φ is given by the following cross-entropy loss, which we call the "intrusion loss":

$$\mathcal{L}_{\text{intr}} := \frac{1}{k_{\max} - 1} \sum_{k=2}^{k_{\max}} \mathbb{E}_{\mathbf{X} \sim \mathcal{D}^{(k)}, \lambda \sim \pi_k(\mathbf{X})} \log p(1|\mathbf{X}\lambda;\varphi) + \mathbb{E}_{x \sim \mathcal{D}} \log p(0|x;\varphi) \tag{5}$$

This intrusion loss $\mathcal{L}_{\text{intr}}$, depending on both π_k's and φ will be responsible for providing training signals for π_k's and φ.

Given π_k's, let $\mathcal{D}'$ be the set of all the virtual examples $\mathbf{X}\lambda$ that can be obtained by drawing an $\mathbf{X}$ from $\mathcal{D}^{(k)}$ for some k and mixing with some policy λ in region $\pi_k(\mathbf{X})$. Let the "MixUp Loss" $\mathcal{L}_{\mathcal{D}'}$ be the average cross-entropy loss for the virtual examples $\mathbf{X}\lambda$ in $\mathcal{D}'$ with respect to their respective soft labels $G(\mathbf{X})\lambda$. Then we arrive at an overall loss function

$$\mathcal{L}_{\text{total}} := \mathcal{L}_{\mathcal{D}}(H) + \mathcal{L}_{\mathcal{D}'}(H, \{\pi_k\}) + \mathcal{L}_{\text{intr}}(\{\pi_k\}, \varphi) \tag{6}$$

Training the networks H, $\{\pi_k\}$ and φ on $\mathcal{L}_{\text{total}}$ then gives rise to the proposed AdaMixUp framework. Comparing the loss in (6) with the objective function in (2), one can equate $\mathcal{L}_{\mathcal{D}'}(H, \{\pi_k\}) + \mathcal{L}_{\text{intr}}(\{\pi_k\}, \varphi)$ in (6) with regularization term $\beta C(H)$ in (2). Specifically, here $\mathcal{L}_{\mathcal{D}'}(H, \{\pi_k\})$ serves to regularize the model and $\mathcal{L}_{\text{intr}}(\{\pi_k\}, \varphi)$ serves to prevent the model from "over-regularization", namely, intrusion or under-fitting. This justifies AdaMixUp (and the standard MixUp) as a regularization technique. The difference between AdaMixUp (6) and the standard regularization (2) is however two fold: first, unlike the standard regularization which is data-independent, AdaMixUp depends on the dataset; on the other hand, AdaMixUp contains parameters, which are adaptively learned from the dataset $\mathcal{D}$.

Given that each π_k has sufficient capacity and properly trained, in theory $\pi_k(\mathbf{X})$ can approximate $\Lambda^*(\mathbf{X})$ well. Then if the φ also has sufficient capacity, the intrusion loss $\mathcal{L}_{\text{intr}}$ can be driven to near zero. In practice however, one usually need to reduce the capacities of π_k's and φ. Then the intrusion loss may be seen as an approximate measure of the extent to which the data allows for mix up without causing intrusion under the architectural choices of π_k's and φ.

Implementation of AdaMixUp

Implementation of $\{\pi_k\}$

Instead of constructing $k_{\max} - 1$ networks π_k's, we in fact implement these networks recursively using a single network π_2. For this purpose, we recursively parametrize $\Lambda(\mathbf{X}) \in \mathbb{S}_k$, which we will rewrite as $\Lambda_k(\mathbf{X})$ for clarity.

For an $\mathbf{X} \in \mathcal{X}^{(k)}$, we may express it as $[\mathbf{X}^{(k-1)}, x_k]$, where $\mathbf{X}^{(k-1)}$ is the sub-matrix of $\mathbf{X}$ containing the first $k-1$ columns. Then, for $k > 2$, we parametrize

$$\Lambda_k(\mathbf{X}) := \left\{ \begin{bmatrix} \gamma\lambda \\ 1-\gamma \end{bmatrix} : \lambda \in \Lambda_{k-1}\left(\mathbf{X}^{(k-1)}\right), \begin{bmatrix} \gamma \\ 1-\gamma \end{bmatrix} \in \Lambda_2\left([\lambda\mathbf{X}^{(k-1)}, x_k]\right) \right\} \tag{7}$$

This then allows the use of single network π_2 to express $\Lambda_k(\mathbf{X})$ for all $\mathbf{X}$ and all k.

Specifically, a Fold-2 mixing policy is parameterized by a single number γ. For each input $\mathbf{X} \in \mathcal{X}^{(2)}$, π_2 returns two positive values α and Δ with $\alpha + \Delta \leq 1$, which specifies $\Lambda(\mathbf{X})$ as the interval $(\alpha, \alpha + \Delta)$ as the range for γ. To this end, the last layer of the π_2 network is implemented as a softmax function, which generates a triplet of values $(\alpha, \Delta, \alpha')$ with sum of one, where the third element α' is discarded. Using network π_2, we generate a set of Fold-2 mixed examples by repeatedly drawing a pair of samples in $\mathcal{D}$ and mixing them using a random policy drawn from policy region generated by π_2 on the pair of samples.

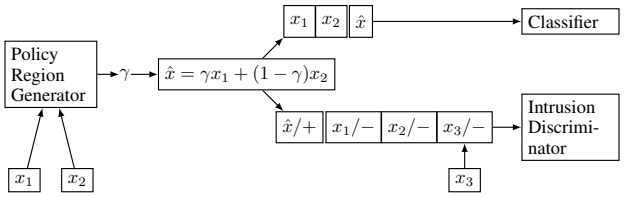

Figure 3: Fold-2 AdaMixUp for a single triplet (x_1, x_2, x_3). Each batch is implemented to contain multiple such triplets. "+" and "-" indicate positive and negative examples, respectively.

To generate mixed samples with an increased maximum fold number, we consider the original examples and previously generated mixed examples as the new training set $\mathcal{D}_{\text{new}}$. We then repeat the following process: draw a pair of samples, one from $\mathcal{D}_{\text{new}}$ and the other from $\mathcal{D}$; submit the pair to network π_2 to generate a policy region; draw a random policy from the region and obtain a new sample by mixing the pair of samples using the drawn policy.

Reparametrization Trick

To allow the gradient signal to back-propagate through the policy sampler, a reparametrization trick similar to that of (Kingma and Welling 2013) is used. Specifically, drawing γ from $(\alpha, \alpha + \Delta)$ is implemented by drawing ϵ from the uniform distribution over $(0, 1)$, then let $\gamma := \Delta \cdot \epsilon + \alpha$.

Implementation of $\varphi(\cdot)$

The network φ is a neural network that classifies original examples from the mixed examples. In our implementation, φ shares with the classifier network all but the final layers. The final layer of φ is a simple logistic regression binary classifier.

Training

The network φ and π_2 are trained jointly, where we iterate over minimizing $\mathcal{L}_{\mathcal{D}} + \mathcal{L}_{\mathcal{D}'}$ and minimizing $\mathcal{L}_{\text{intr}}$ in (6) via mini-batched SGD. A Fold-2 AdaMixUp is illustrated in Figure 3.

Experiments

Data Sets and Experimental Settings

We evaluate AdaMixUp on eight benchmarks. *MNIST* is the popular digit (1-10) recognition dataset with 60,000 training and 10,000 test gray-level, 784-dimensional images. *Fashion* is an image recognition dataset having the same scale as MNIST, containing 10 classes of fashion product pictures. *SVHN* is the Google street view house numbers recognition data set with 73,257 digits (1-10) 32x32 color images for training, 26,032 for testing, and 531,131 additional, easier samples. We did not use the additional images. *Cifar10* is an image classification dataset with 10 classes, 50,000 training and 10,000 test samples. *Cifar100* is similar to CIFAR10 but with 100 classes and 600 images each. *Cifar10-S* and *Cifar100-S* are respectively Cifar10 and Cifar100 reduced to containing only 20% of the training samples. *ImageNet-R* is the ImageNet-2012 classification dataset (Russakovsky et al. 2014) with 1.3 million training images, 50,000 validation images, and 1,000 classes. We follow the data processing approaches used in Mixup (Zhang et al. 2017), except that the crop size is 100x100 instead of 224x224 due to our limited computational resources. We report both top-1 and top-5 error rates.

In our experiments, the Intrusion Discriminator, Classifier, and Policy Region Generator have the same network architecture, and the first two share the same set of parameters. We test AdaMixUp on two types of baseline networks: a three layers CNN as implemented in (Wu and others 2016) as the baseline for easier tasks MNIST and Fashion, and a ResNet-18 as implemented in (Zagoruyko and Komodakis 2016) for the other six more difficult tasks. All models examined are trained using mini-batched backprop, as specified in (Wu and others 2016) and (Zagoruyko and Komodakis 2016), for 400 epochs. Each reported performance value (accuracy or error rate) is the median of the performance values obtained in the final 10 epochs.

Predictive Performance

Our first experiment compares Fold-2 AdaMixUp (i.e., mixing image pairs) to two networks: the baseline networks (i.e., 3-layer CNN for MNIST and Fashion and ResNet-18 for the rest datasets) and MixUp on the baseline networks. We present the error rates obtained by the Baseline, MixUp, and AdaMixUp, along with the relative error reduction of AdaMixUp over the Baseline, in Table 1.

Table 1 indicates that the AdaMixUp outperforms both the Baseline and MixUp on all the eight testing datasets. Also, the relative error reduction of AdaMixUp over the Baseline is at least 5.7%, and with a large margin in some cases. For example, for the SVHN and Cifar10 datasets, the relative error reduction achieved by AdaMixUp are over 30%.

Table 1 also suggests that, with the default α value, MixUp failed to improve the baseline systems' accuracy on four out of the eight datasets, namely the MNIST, Cifar10-S, Cifar100-S, and Imagenet-R datasets, as underlined in Table 1, suggesting over-regularization or under-fitting.

Data Set	Baseline	MixUp	Ada MixUp	Relative Impro. (%)
mnist	0.52	0.57	0.49	5.77
fashion	7.37	6.92	6.21	15.74
svhn	4.50	3.80	3.12	30.67
cifar10	5.53	4.24	3.52	36.35
cifar100	25.6	21.14	20.97	18.09
cifar10-S	7.68	7.88	6.85	10.81
cifar100-S	28.47	29.39	26.72	6.15
ImageNet-R top1	53.00	54.89	49.17	7.22
ImageNet-R top5	29.41	31.02	25.78	12.34

Table 1: Error rates (%) obtained by the testing methods.

In Table 2, we also present the α and Δ values learned by the Policy Region Generator, along with the losses of both the Intrusion Discriminator and the Classifier of the

AdaMixUp method. Results in Table 2 indicate that the values of α and Δ vary for different datasets; the former ranges from 0.4 to 0.9 but the range for Δ is much smaller, with values between 0.010 and 0.035. Noting that the intrusion losses are fairly close to 0 on all the datasets, suggesting that these datasets well support Fold-2 AdaMixUp under the proposed structure of Policy Region Generator.

Ada values	α	Δ	$\mathcal{L}_{\text{intr}}$	$\mathcal{L}_{\mathcal{D}} + \mathcal{L}_{\mathcal{D}'}$
mnist	0.497	0.010	0.002	0.201
fashion	0.511	0.011	0.003	0.290
svhn	0.924	0.011	0.002	0.116
cifar10	0.484	0.029	0.003	0.371
cifar100	0.484	0.034	0.002	0.679
cifar10-S	0.486	0.027	0.003	0.479
cifar100-S	0.482	0.035	0.007	0.712
ImageNet-R	0.493	0.004	0.038	2.382

Table 2: Fold-2 AdaMixUp: Mean α, Δ, and training losses.

Training Characteristics

Figure 4 depicts the behavior of Policy Region Generator in Fold-2 AdaMixUp over training iterations. It appears that the Policy Region Generator initially explores a wide range of policy space before settling down at around 40K iterations (when the means of α and Δ both stabilize, at 0.48 and 0.03 respectively).

Figure 5 shows the average of the drawn mixing policy γ in Fold-2 AdaMixUp during training iterations and the corresponding classification performance of the model. The range of γ appears to stabilize at around 40K iterations (left plot), consistent with the observation in Figure 4. Interestingly, also at the same time, the classification error drops and begins to stabilize (right plot).

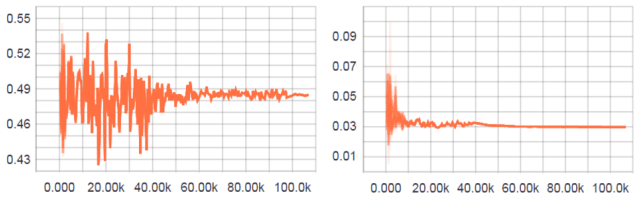

Figure 4: The mean of α (left) and the mean of Δ (right) in Fold-2 AdaMixUp on Cifar10.

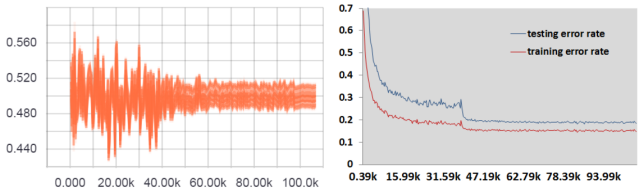

Figure 5: Fold-2 AdaMixUp on Cifar10: Mean of mixing policy γ (left) and training/testing error rates (right).

Figures 6 shows some typical mixed images in Fold-2 AdaMixUp and their corresponding original images in MNIST. We note that these mixed images obviously distinguish themselves from the original images. Thus they do not intrude into the data manifold.

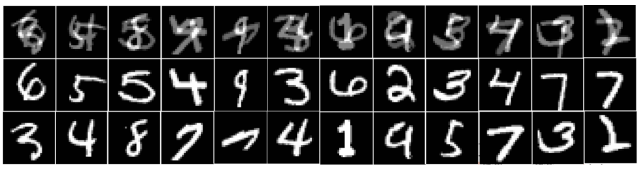

Figure 6: Mixed images (top row) in Fold-2 AdaMixUp from the original images (bottom 2 rows) in MNIST.

Impact of Mixing Fold

Table 3 presents the error rates obtained by Fold-3 and Fold-4 AdaMixUp on the seven testing datasets (excluding ImageNet due to limited computational resources). We see that increasing the mixing fold from 2 to higher values, the performance of AdaMixUp improves. This is due to stronger constraints are imposed to the model and hence results in stronger regularization. One however expects that imposing stronger constraints makes the model more difficult to fit and further increasing mixing fold may eventually lead to diminishing gain and even under-fitting. This is hinted by Table 4, where the averages of learned mixing parameters and the training losses of the Fold-3 AdaMixUp are presented.

In Table 4, we see that the intrusion losses of Fold-3 AdaMixUp are much higher than those of Fold-2 AdaMixUp (Table 2). This implies that in higher-fold AdaMixUp, it is more difficult for the Policy Region Generator to carve a good policy region that is not regarded by the Intrusion Discriminator as causing intrusion. This difficulty is also suggested by the high α_2 values in Table 4 (often close to 1). They indicate that in these cases, mixing only occurs "slightly".

AdaMixUp	Fold-2	Fold-3	Fold-4
mnist	0.49	0.42	0.41
fashion	6.21	5.88	5.71
svhn	3.12	2.98	2.91
cifar10	3.52	3.25	3.21
cifar100	20.97	20.68	20.29
cifar10-S	6.85	6.14	5.96
cifar100-S	26.72	25.72	25.49

Table 3: Error rates (%) of Fold-3 and Fold-4 AdaMixUp.

Sensitivity to Model Capacity

We vary the number of filters on each layer of the ResNet-18 with half quarter, quarter, half, and three-quarter of the original number of filters (denoted as base filter), and present the test error rates on Cifar100 in Figure 7 (green curves).

The green curves in Figure 7 indicate that the Fold-2 AdaMixUp benefits from large number of filters deployed:

Fold-3	$\alpha 1/\Delta 1$	$\alpha 2/\Delta 2$	$\mathcal{L}_{\text{intr}}$	$\mathcal{L}_{\mathcal{D}} + \mathcal{L}_{\mathcal{D}'}$
mnist	0.533/0.013	0.662/0.013	0.002	0.337
fashion	0.517/0.015	0.678/0.019	0.024	0.327
svhn	0.637/0.011	0.943/0.020	0.028	0.482
cifar10	0.516/0.028	0.938/0.041	0.003	0.403
cifar100	0.121/0.010	0.959/0.015	0.004	0.305
cifar10-S	0.504/0.039	0.849/0.068	0.028	0.505
cifar100-S	0.639/0.011	0.943/0.018	0.023	0.542

Table 4: Fold-3 AdaMixUp: average training losses and policy region parameters. (α_1, Δ_1): initial Fold-2 mixing parameter; (α_2, Δ_2): further mixing parameter.

the accuracy keeps improving while enlarging the number of filters. On the contrary, the ResNet-18 received dramatically accuracy improvement before the number of filters is half of the base but obtained no further improvement after.

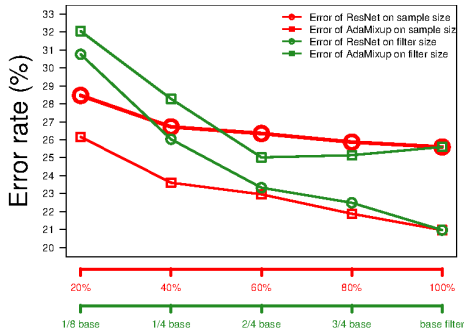

Figure 7: Error rates obtained by ResNet-18 and AdaMixUp on Cifar100, when varying the number of filters (green curves) and varying the size of the samples (red curves).

Effect of Data Size

We down sample the Cifar100 data with 20%, 40%, 60% and 80% of the training samples, and present the test error rates obtained by Fold-2 AdaMixUp in Figure 7 (red curves).

The red curves in Figure 7 indicate that more training samples help with both the ResNet-18 and AdaMixUp, but the accuracy gap between the two methods are widening while increasing the training sample size.

Benefit of the Intrusion Discriminator

Table 5 lists the test error rates obtained by the Fold-2 AdaMixUp with ResNet-18 on the Cifar10 and Cifar100 data, but without the Intrusion Discriminator. Table 5 shows

Data Set	Baseline ResNet-18	Ada MixUp	AdaMixUp w/o Intru. Discr.
cifar10	5.53	3.52	3.83
cifar100	25.6	20.97	24.75

Table 5: Error rates obtained by Fold-2 AdaMixUp without the Intrusion Discriminator on the Cifar10 and Cifar100.

that excluding the Intrusion Discriminator component, the AdaMixUp method was still able to improve the accuracy over the baseline, but obtained much lower accuracy than that of including the Intrusion Discriminator unit.

Interpolating on Hidden Layer

We also evaluate an alternative structure for AdaMixUp, where mixing happens on the layer before the final softmax layer in ResNet-18. Our results suggest that the test errors of the Fold-2 AdaMixUp increased dramatically due to the high cost of the Intrusion Discriminator. The error rates increased from 20.97% to 22.21% and from 3.52% to 4.94%, respectively for Cifar100 and Cifar10. Notably, both have large intrusion loss of around 0.49, which may due to the fact that perceptual similarity for images in the hidden embedding space may not as distinguishable as that in the input space. As a result, the Policy Generator failed to find good mixing policies to avoid colliding into the data manifold.

Related Work

Common data augmentation methods have been designed based on substantial domain knowledge (Lecun et al. 1998; Simonyan and Zisserman 2014; Amodei et al. 2015; Zhong et al. 2017), relied on specific network architectures (Gastaldi 2017; Devries and Taylor 2017; Yamada, Iwamura, and Kise 2018), or leveraged feedback signals to search the optimal augmentation strategies (Lemley, Bazrafkan, and Corcoran 2017; Cubuk et al. 2018). Our method excludes those requirements, and only leverages a simple linear interpolation for data augmentation.

Our work closely relates to approaches linearly interpolating examples and labels (Zhang et al. 2017; DeVries and W. Taylor 2017; Verma et al. 2018; Tokozume, Ushiku, and Harada 2017). Nevertheless, these approaches depends on the correct user-predefined mixing policies. Also, their interpolation typically lies along the lines of sample pairs. On the contrary, our approach automatically learns the mixing policy regions and benefits from mixing multiple images.

Conclusions and Outlook

This work justifies the effectiveness of MixUp from the perspective of out-of-manifold regularization. We also identify the inherent problem with standard MixUp in causing manifold intrusion. This motivates us to develop AdaMixUp, which generalizes MixUp to higher-fold mixing policies and automatically learns the policy regions that avoid manifold intrusion.

To us, this work is only the beginning of a rich research theme. The justification of MixUp and AdaMixUp we provide thus far is primarily qualitative. It is desirable to quantitatively characterize the generalization capability of AdaMixUp. Moreover, the out-of-manifold regularization perspective potentially opens a door to new regularization techniques. Beyond local linearization, we believe that there is a rich space of other possibilities.

Acknowledgments

The authors wish to thank the anonymous reviewers for their valuable comments, some of which are likely to help further-

ing this work.

This work is supported partly by China 973 program (2015CB358700), by the National Natural Science Foundation of China (61772059), and by the Beijing Advanced Innovation Center for Big Data and Brain Computing.

References

Alemi, A. A.; Fischer, I.; Dillon, J. V.; and Murphy, K. 2016. Deep variational information bottleneck. *CoRR* abs/1612.00410.

Amodei, D.; Anubhai, R.; Battenberg, E.; Case, C.; Casper, J.; Catanzaro, B.; Chen, J.; Chrzanowski, M.; Coates, A.; Diamos, G.; Elsen, E.; Engel, J.; Fan, L.; Fougner, C.; Han, T.; Hannun, A. Y.; Jun, B.; LeGresley, P.; Lin, L.; Narang, S.; Ng, A. Y.; Ozair, S.; Prenger, R.; Raiman, J.; Satheesh, S.; Seetapun, D.; Sengupta, S.; Wang, Y.; Wang, Z.; Wang, C.; Xiao, B.; Yogatama, D.; Zhan, J.; and Zhu, Z. 2015. Deep speech 2: End-to-end speech recognition in english and mandarin. *CoRR* abs/1512.02595.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *CoRR* abs/1409.0473.

Cubuk, E. D.; Zoph, B.; Mané, D.; Vasudevan, V.; and Le, Q. V. 2018. Autoaugment: Learning augmentation policies from data. *CoRR* abs/1805.09501.

Devries, T., and Taylor, G. W. 2017. Improved regularization of convolutional neural networks with cutout. *CoRR* abs/1708.04552.

DeVries, T., and W. Taylor, G. 2017. Dataset augmentation in feature space. *CoRR*.

Gal, Y., and Ghahramani, Z. 2015. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *CoRR* abs/1506.02142.

Gastaldi, X. 2017. Shake-shake regularization. *CoRR* abs/1705.07485.

Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A. C.; and Bengio, Y. 2014. Generative adversarial nets. In *NIPS*, 2672–2680.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and harnessing adversarial examples. *CoRR* abs/1412.6572.

Graves, A.; Mohamed, A.; and Hinton, G. E. 2013. Speech recognition with deep recurrent neural networks. *CoRR* abs/1303.5778.

Hanson, S. J., and Pratt, L. Y. 1988. Comparing biases for minimal network construction with back-propagation. In *NIPS1988*, 177–185.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Identity mappings in deep residual networks. *CoRR* abs/1603.05027.

Hinton, G.; Deng, L.; Yu, D.; Dahl, G.; rahman Mohamed, A.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T.; and Kingsbury, B. 2012. Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine*.

Kingma, D. P., and Welling, M. 2013. Auto-encoding variational bayes. *CoRR* abs/1312.6114.

Kipf, T. N., and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *CoRR* abs/1609.02907.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*, 1097–1105. USA: Curran Associates Inc.

Lecun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

Lemley, J.; Bazrafkan, S.; and Corcoran, P. 2017. Smart augmentation - learning an optimal data augmentation strategy. *CoRR* abs/1703.08383.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M. S.; Berg, A. C.; and Li, F. 2014. Imagenet large scale visual recognition challenge. *CoRR* abs/1409.0575.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T. P.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Simard, P.; LeCun, Y.; Denker, J. S.; and Victorri, B. 1998. Transformation invariance in pattern recognition-tangent distance and tangent propagation. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, 239–27.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *CoRR* abs/1409.1556.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15(1):1929–1958.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. *CoRR* abs/1409.3215.

Tokozume, Y.; Ushiku, Y.; and Harada, T. 2017. Between-class learning for image classification. *CoRR* abs/1711.10284.

Verma, V.; Lamb, A.; Beckham, C.; Courville, A.; Mitliagkas, I.; and Bengio, Y. 2018. Manifold mixup: Encouraging meaningful on-manifold interpolation as a regularizer. *CoRR*.

Wu, Y., et al. 2016. Tensorpack. https://github.com/tensorpack/.

Yamada, Y.; Iwamura, M.; and Kise, K. 2018. Shakedrop regularization. *CoRR* abs/1802.02375.

Zagoruyko, S., and Komodakis, N. 2016. Url https://github.com/szagoruyko/ wide-residual-networks.

Zhang, H.; Cissé, M.; Dauphin, Y. N.; and Lopez-Paz, D. 2017. mixup: Beyond empirical risk minimization.

Zhong, Z.; Zheng, L.; Kang, G.; Li, S.; and Yang, Y. 2017. Random erasing data augmentation. *CoRR* abs/1708.04896.