PatchUp: A Feature-Space Block-Level Regularization Technique for Convolutional Neural Networks

Mojtaba Faramarzi[1, 2], **Mohammad Amini**[1, 3], **Akilesh Badrinaaraayanan**[1, 2], **Vikas Verma**[1, 2, 4], **Sarath Chandar** [1, 5]

[1]Mila - Quebec AI Institute
[2]University of Montreal
[3]McGill University
[4]Aalto Univeristy, Finland
[5]École Polytechnique de Montréal

Abstract

Large capacity deep learning models are often prone to a high generalization gap when trained with a limited amount of labeled training data. A recent class of methods to address this problem uses various ways to construct a new training sample by mixing a pair (or more) of training samples. We propose PatchUp, a hidden state block-level regularization technique for Convolutional Neural Networks (CNNs), that is applied on selected contiguous blocks of feature maps from a random pair of samples. Our approach improves the robustness of CNN models against the manifold intrusion problem that may occur in other state-of-the-art mixing approaches. Moreover, since we are mixing the contiguous block of features in the hidden space, which has more dimensions than the input space, we obtain more diverse samples for training towards different dimensions. Our experiments on CIFAR10/100, SVHN, Tiny-ImageNet, and ImageNet using ResNet architectures including PreActResnet18/34, WRN-28-10, ResNet101/152 models show that PatchUp improves upon, or equals, the performance of current state-of-the-art regularizers for CNNs. We also show that PatchUp can provide a better generalization to deformed samples and is more robust against adversarial attacks.

1 Introduction

Deep Learning (DL), particularly deep Convolutional Neural Networks (CNNs) have achieved exceptional performance in many machine learning tasks, including object recognition (Krizhevsky, Sutskever, and Hinton 2012), image classification (Krizhevsky, Sutskever, and Hinton 2012; Ren et al. 2015; He et al. 2015), speech recognition (Hinton et al. 2012) and natural language understanding (Sutskever, Vinyals, and Le 2014; Vaswani et al. 2017). However, in a very deep and wide network, the network has a tendency to memorize the samples, which yields poor generalization for data outside of the training data distribution (Arpit et al. 2017; Goodfellow, Bengio, and Courville 2016). To address this issue, noisy computation is often employed during the training, making the model more robust against invariant samples and thus improving the generalization of the model (Achille and Soatto 2018). This idea is exploited in several state-of-the-art regularization techniques.

Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Such noisy computation based regularization techniques can be categorized into data-dependent and data-independent techniques (Guo, Mao, and Zhang 2018). Earlier work in this area has been more focused on the data-independent techniques such as Dropout (Srivastava et al. 2014), Variational Dropout (Gal and Ghahramani 2016) and ZoneOut (Krueger et al. 2016), Information Dropout (Achille and Soatto 2018), SpatialDropout (Tompson et al. 2014a), and DropBlock (Ghiasi, Lin, and Le 2018). Dropout performs well on fully connected layers (Srivastava et al. 2014). However, it is less effective on convolutional layers (Tompson et al. 2014b). One of the reasons for the lack of success of dropout on CNN layers is perhaps that the activation units in the convolutional layers are correlated, thus despite dropping some of the activation units, information can still flow through these layers. SpatialDropout (Tompson et al. 2014b) addresses this issue by dropping the entire feature map from a convolutional layer. DropBlock (Ghiasi, Lin, and Le 2018) further improves SpatialDropout by dropping random continuous feature blocks from feature maps instead of dropping the entire feature map in the convolutional layers.

Data-augmentation also is a data-dependent solution to improve the generalization of a model. Choosing the best augmentation policy is challenging. AutoAugment (Cubuk et al. 2019) finds the best augmentation policies using reinforcement learning with huge computation overhead. AugMix (Hendrycks et al. 2020) reduces this overhead by using stochasticity and diverse augmentations and adding a Jensen-Shannon Divergence consistency loss to training loss. Recent works show that data-dependent regularizers can achieve better generalization for CNN models. Mixup (Zhang et al. 2017), one such data-dependent regularizer, synthesizes additional training examples by interpolating random pairs of inputs x_i, x_j and their corresponding labels y_i, y_j as:

$$\tilde{x} = \lambda x_i + (1-\lambda)x_j \quad \text{and} \quad \tilde{y} = \lambda y_i + (1-\lambda)y_j, \quad (1)$$

where $\lambda \in [0, 1]$ is sampled from a Beta distribution such that $\lambda \sim Beta(\alpha, \alpha)$ and $(\tilde{x},\ \tilde{y})$ is the new example. By using these types of synthetic samples, Mixup encourages the model to behave linearly in-between the training samples. The mixing coefficient λ in Mixup is sampled from

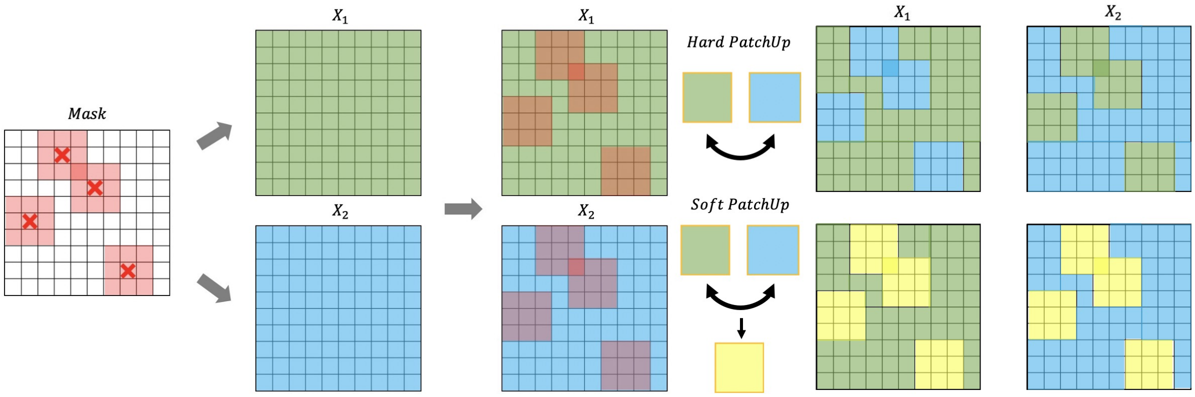


Figure 1: *PatchUp* process for two hidden representations associated with two samples randomly selected in the mini-batch $(a,\ b)$. $X_1 = g_k^{(i)}(a)$ and $X_2 = g_k^{(i)}(b)$ where i is the feature map index. Right top shows *Hard PatchUp* output and the right bottom shows the interpolated samples with *Soft PatchUp*. The yellow continuous blocks represent the interpolated selected blocks.

a prior distribution. This may lead to the *manifold intrusion problem* (Guo, Mao, and Zhang 2018): the mixed synthetic example may *collide* (i.e. have the same value in the input space) with other examples in the training data, essentially leading to two training samples which have the same inputs but different targets. To overcome the manifold intrusion problem, MetaMixUp (Mai et al. 2019) used a meta-learning approach to learn λ with a lower possibility of causing such collisions. However, this meta-learning approach adds significant computation complexity. ManifoldMixup (Verma et al. 2019) attempts to avoid the manifold intrusion problem by interpolating the hidden states (instead of input states) of a randomly chosen layer at every training update. Recently, Puzzle Mix (Kim, Choo, and Song 2020) explicitly exploits an optimized masking strategy for the Input Mixup. It uses the saliency information and the underlying statistics of pair of images to avoid manifold intrusion problems at each batch training step. Puzzle Mix adds computation overhead at training time to find an optimal mask policy while improving the performance of the model in comparison to Mixup and ManifoldMixup.

Different from the interpolation-based regularizers discussed above, Cutout (DeVries and Taylor 2017) drops the contiguous regions from the image in the input space. This kind of noise encourages the network to learn the full context of the images instead of overfitting to the small set of visual features. CutMix (Yun et al. 2019) is another data-dependent regularization technique that cuts and fills rectangular shape parts from two randomly selected pairs in a mini-batch instead of interpolating two selected pairs completely. Applying CutMix at the input space improves the generalization of the CNN model by spreading the focus of the model across all places in the input instead of just a small region or a small set of intermediate activations. According to the CutMix paper, applying CutMix at the latent space, Feature CutMix, is not as effective as applying CutMix in the input space (Yun et al. 2019).

In this work, we introduce a feature-space block-level data-dependent regularization that operates in the hidden space by masking out contiguous blocks of the feature map of a random pair of samples, and then either mixes (*Soft PatchUp*) or swaps (*Hard PatchUp*) these selected contiguous blocks. Our regularization method does not incur significant computational overhead for CNNs during training. *PatchUp* improves the generalization of ResNet architectures on image classification task (on CIFAR-10, CIFAR-100, SVHN, and Tiny-ImageNet), deformed images classification, and against adversarial attacks. It also helps a CNN model to produce a wider variety of features in the residual blocks compared to other state-of-the-art regularization methods for CNNs such as Mixup, Cutout, CutMix, ManifoldMixup, and Puzzle Mix.

2 PatchUp

PatchUp is a hidden state block-level regularization technique that can be used after any convolutional layer in CNN models. Given a deep neural network $f(x)$ where x is the input, let g_k be the k-th convolutional layer. The network $f(x)$ can be represented as $f(x) = f_k(g_k(x))$ where g_k is the mapping from the input data to the hidden representation at layer k and f_k is the mapping from the hidden representation at layer k to the output (Verma et al. 2019). In every training step, *PatchUp* applies block-level regularization at a randomly selected convolutional layer k from a set of intermediate convolutional layers. appendix-B gives a formal intuition for selecting k randomly.

Binary Mask Creation

Once a convolutional layer k is chosen, the next step is to create a binary mask M (of the same size as the feature map in layer k) that will be used to *PatchUp* a pair of examples in the space of $g_k(x)$. The mask creation process is similar to that of DropBlock (Ghiasi, Lin, and Le 2018). The idea is to select contiguous blocks of features from the feature map that will be either mixed or swapped with the same features in another example. To do so, we first select a set of features that can be altered (mixed or swapped). This is done by using the hyper-parameter γ which decides the probability of altering a feature. When we alter a feature, we also alter a square block of features centered around that feature which is controlled by the side length of this square block, $block\_size$. Hence, the altering probabilities are readjusted using the following formula (Ghiasi, Lin, and Le 2018):

$$\gamma_{adj} = \frac{\gamma \times \textit{(feature map's area)}}{\textit{(block's area)} \times \textit{(valid region to build block)}}, \quad (2)$$

where the area of the feature map and block are the $feat\_size^2$ and $block\_size^2$, respectively, and the valid region to build the block is $(feat\_size - block\_size + 1)^2$. For each feature in the feature map, we sample from $Bernoulli(\gamma_{adj})$. If the result of this sampling for feature f_{ij} is 0, then $M_{ij} = 1$. If the result of this sampling for f_{ij} is 1, then the entire square region in the mask with the center M_{ij} and the width and height of the square of $block\_size$ is set to 0. Note that these feature blocks to be altered can overlap which will result in more complex block structures than just squares. The block structures created are called patches. Fig.1 illustrates an example mask used by *PatchUp*. The mask M has 1 for features outside the patches (which are not altered) and 0 for features inside the patches (which are altered). See Fig. 9 and 8 in Appendix for more details.

PatchUp Operation

Once the mask is created, we can use the mask to select patches from the feature maps and either swap these patches (*Hard PatchUp*) or mix them (*Soft PatchUp*).

Consider two samples x_i and x_j. The *Hard PatchUp* operation at layer k is defined as follows:

$$\phi_{\mathbf{hard}}(g_k(x_i), g_k(x_j)) = \mathbf{M} \odot g_k(x_i) + (\mathbf{1} - \mathbf{M}) \odot g_k(x_j), \quad (3)$$

where $\odot$ is known as the element-wise multiplication operation and $\mathbf{M}$ is the binary mask described in section 2. To define *Soft PatchUp* operation, we first define the mixing operation for any two vectors a and b as follows:

$$\text{Mix}_{\lambda}(a,\ b) = \lambda \cdot a + (1 - \lambda) \cdot b, \quad (4)$$

where $\lambda \in [0, 1]$ is the mixing coefficient. Thus, the *Soft PatchUp* operation at layer k is defined as follows:

$$\begin{aligned}\phi_{\mathbf{soft}}(g_k(x_i), g_k(x_j)) =& \mathbf{M} \odot g_k(x_i) + \text{Mix}_{\lambda}[((\mathbf{1} - \mathbf{M}) \\ & \odot g_k(x_i)),\ ((\mathbf{1} - \mathbf{M}) \odot g_k(x_j))],\end{aligned} \quad (5)$$

where λ in the range of $[0,\ 1]$ is sampled from a Beta distribution such that $\lambda \sim Beta(\alpha, \alpha)$. α controls the shape of the Beta distribution. Hence, it controls the strength of interpolation (Zhang et al. 2017). *PatchUp* operations are illustrated in Fig. 1 (see more details in Algorithm 1 in Appendix).

Learning Objective

After applying the *PatchUp* operation, the CNN model continues the forward pass from layer k to the last layer in the model. The output of the model is used for the learning objective, including the loss minimization process and updating the model parameters accordingly. Consider the example pairs (x_i, y_i) and (x_j, y_j). Let $\phi_k = \phi(g_k(x_i), g_k(x_j))$ be the output of *PatchUp* after the k-th layer. Mathematically, the CNN with *PatchUp* minimizes the following loss function:

$$\begin{aligned}L(f) =& \mathbb{E}_{(x_i,\ y_i)\sim P}\ \mathbb{E}_{(x_j,\ y_j)\sim P}\ \mathbb{E}_{\lambda\sim \text{Beta}(\alpha,\ \alpha)}\, \mathbb{E}_{k\sim\mathcal{S}} \\ & \text{Mix}_{p_u}[\ell(f_k(\phi_k),\ y_i)\ , \ell(f_k(\phi_k),\ Y)] \\ & + \ell(f_k(\phi_k),\ W(y_i,\ y_j)),\end{aligned} \quad (6)$$

where p_u is the fraction of the unchanged features from feature maps in $g_k(x_i)$ and $\mathcal{S}$ is the set of layers where *PatchUp* is applied randomly. ϕ is ϕ_{hard} for *Hard PatchUp* and ϕ_{soft} for *Soft PatchUp*.

Y is the target corresponding to the changed features. In the case of *Hard PatchUp*, $Y = y_j$ and in the case of *Soft PatchUp*, $Y = Mix_{\lambda}(y_i, y_j)$. $W(y_i, y_j)$ calculates the reweighted target according to the interpolation policy for y_i and y_j. W for *Hard PatchUp* and *Soft PatchUp* is defined as follows:

$$W_{hard}(y_i, y_j) = \text{Mix}_{p_u}(y_i, y_j) \quad (7)$$

$$W_{soft}(y_i, y_j) = \text{Mix}_{p_u}(y_i,\ \text{Mix}_{\lambda}(y_i, y_j)). \quad (8)$$

The *PatchUp* loss function has two terms where the first term is inspired from the CutMix loss function and the second term is inspired from the MixUp loss function (more detail in Appendix-E).

PatchUp in Input Space

By setting $k = 0$, we can apply *PatchUp* to only the input space. When we apply *PatchUp* to the input space, only the *Hard PatchUp* operation is used, this is due to the reason that, as shown in (Yun et al. 2019), swapping in the input space provides better generalization compared to mixing. Furthermore, we select only one random rectangular patch in the input space (similar to CutMix) because the *PatchUp* binary mask is potentially too strong for the input space, which has only three channels, compared to hidden layers in which each layer can have a larger number of channels (more detail in section "4").

3 Relation to Similar Methods

***PatchUp* Vs. ManifoldMixup:** *PatchUp* and ManifoldMixup improve the generalization of a model by combining the latent representations of a pair of examples. ManifoldMixup linearly mixes two hidden representations using Equation 4. *PatchUp* uses a more complex approach ensuring that a more diverse subspace of the hidden space gets explored. To understand the behaviour and the limitation that exists in the ManifoldMixup, assume that we have a 3D hidden space representation as illustrated in Fig.2. It presents the possible combinations of hidden representations explored via ManifoldMixup and *PatchUp*. Blue dots represent real hidden representation samples. ManifoldMixup can produce new samples that lie directly on the orange lines which connect the blue point pairs due to its linear interpolation strategy. But, *PatchUp* can select various points in all dimensions, and can also select points extremely close to the orange lines. The proximity to the orange lines depends on the selected pairs and λ sampled from the beta distribution. Fig.2 is a simple diagrammatic description of how

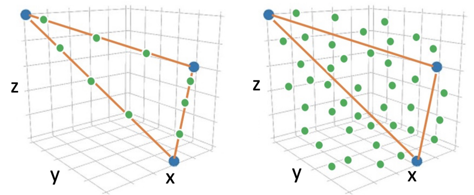


Figure 2: Left: ManifoldMixup interpolated samples for any combination of the three blue hidden states selected only from along orange line. Right: *PatchUp* can produce interpolated hidden representations for these three hidden states in almost all possible places in all dimensions except the samples which lie directly on the orange lines.

Figure 3: The two possible block selections from CutMix for two samples (cat and dog) with a large background. Swapping a similar part of the background or an essential element correlated to the label in the selected images can have a negative effect on the CutMix learning objective.

PatchUp constructs more diverse samples. Appendix-C provides a mathematical and real experimental justification.

***PatchUp* Vs. CutMix:** The CutMix cuts and fills the rectangular parts of the randomly selected pairs instead of using interpolation for creating a new sample in the input space. Therefore, CutMix has less potential for a manifold intrusion problem, however, CutMix may still suffer from a manifold intrusion problem. Fig.3 shows two samples with small portions that correspond to their labels. In this example, if only the parts within the yellow bounding boxes are swapped, then the label does not change. However, if the parts within the white bounding boxes are swapped, then the entire label is swapped. In both scenarios, CutMix only learns the interpolated target based on the fractions of the swapped part. In contrast, these scenarios are less likely to occur in *PatchUp* since it works in the hidden representation space most of the time. Another difference between CutMix and *PatchUp* is how the masks are created. *PatchUp* can create arbitrarily shaped masks while CutMix masks can only be rectangular. Fig. 8 (appx.) shows an example of CutMix and *PatchUp* masks in input space and hidden representation space, respectively. CutMix is more effective than Feature-CutMix that applies CutMix in the latent space (Yun et al. 2019). The learning objective of *PatchUp* and the binary mask selection are both different from that of Feature-CutMix.

4 Experiments

This section presents the results of applying *PatchUp* to image classification tasks using various benchmark datasets such as CIFAR10, CIFfAR100 (Krizhevsky, Hinton et al. 2009), SVHN (the standard version with 73257 training samples) (Netzer et al. 2011), Tiny-ImageNet (Chrabaszcz, Loshchilov, and Hutter 2017), and with various benchmark architectures such as PreActResNet18/34 (He et al. 2016), and ResNet101, ResNet152, and WideResNet-28-10 (WRN-28-10) (Zagoruyko and Komodakis 2017)[1]. We used the same set of base hyper-parameters for all the models for a thorough and fair comparison. The details of experimental setup and the hyper-parameter tuning are given in appendix-E. We set α to 2 in *PatchUp*. *PatchUp* has $patchup\_prob$, γ and $block\_size$ as additional hyper-parameters. $patchup\_prob$ is the probability that *PatchUp* is performed for a given mini-batch. Based on our hyper-parameter tuning, *Hard PatchUp* yields the best performance with $patchup\_prob$, γ, and $block\_size$ as 0.7, 0.5, and 7, respectively. *Soft PatchUp* achieves the best performance with $patchup\_prob$, γ, and $block\_size$ as 1.0, 0.75, and 7, respectively.

[1]The code is available: https://github.com/chandar-lab/PatchUp

Generalization on Image Classification

Table 2 shows the comparison of the generalization performance of *PatchUp* with six recently proposed mixing based or feature-level methods on the CIFAR10/100, and SVHN datasets. Since Puzzle Mix clearly showed that both CutMix and Puzzle Mix perform better than AugMix (Kim, Choo, and Song 2020), we excluded it from our experiments. Tables 12 and 11 in appendix-I show test errors and NLLs. Our experiments show that *PatchUp* leads to a lower test error for all the models on CIFAR, SVHN, and Tiny-ImageNet with a large margin. Specifically, *Soft PatchUp* outperforms other methods on Tiny-ImageNet dataset using ResNet101/152, and WRN-28-10 followed by *Hard PatchUp*. As explained in appendix-C and shown in Fig. 10, both *Soft* and *Hard PatchUp* produce a wide variety of interpolated hidden representations towards different dimensions. However, *Soft PatchUp* behaves more conservatively that helps to outperform other methods with a large margin in the case of a limited number of training samples per class and having more targets.

Hard PatchUp provides the best performance in the CIFAR and *Soft PatchUp* achieves the second-best performance except on the CIFAR10 with WRN-28-10 where Puzzle Mix provides the second-best performance. In the SVHN, ManifoldMixup achieves the second-best performance in PreActResNet18 and 34 where *Hard PatchUp* provide the lowest top-1 error. *Soft PatchUp* performs reasonably well and comparable to ManifoldMixup for PreActResNet34 on SVHN and leads to a lower test error followed by *Hard PatchUp* for WRN-28-10 in the SVHN. We observe that the Mixup, ManifoldMixup, and Puzzle Mix are sensitive to the α when we have more training classes. It is notable that using the same α, that is used in CIFAR or SVHN, leads to worst performance than No-Mixup in Tiny-

	ResNet101		ResNet152		WideResnet-28-10	
	Error	Loss	Error	Loss	Eror	Loss
No Mixup	46.18 ± 0.49	2.37 ± 0.03	45.22 ± 0.32	2.28 ± 0.08	36.76 ± 0.25	1.89 ± 0.02
Input Mixup ($\alpha = 1$)	44.93 ± 0.34	2.25 ± 0.05	44.65 ± 0.11	2.23 ± 0.04	36.50 ± 0.29	1.88 ± 0.02
ManifoldMixup (α= 2)	44.54 ± 0.14	2.25 ± 0.01	44.13 ± 0.31	2.23 ± 0.02	35.96 ± 0.64	1.83 ± 0.06
Cutout	45.78 ± 0.35	2.35 ± 0.01	45.23 ± 0.19	2.34 ± 0.02	35.85 ± 0.11	1.83 ± 0.01
DropBlock	46.98 ± 0.53	2.27 ± 0.10	45.82 ± 0.22	2.17 ± 0.01	36.97 ± 0.38	1.89 ± 0.02
CutMix	42.04 ± 0.38	2.11 ± 0.01	41.71 ± 0.71	2.08 ± 0.03	35.81 ± 0.18	1.75 ± 0.08
Puzzle Mix	42.25 ± 0.19	2.13 ± 0.01	42.13 ± 0.31	2.13 ± 0.03	33.43 ± 0.22	1.65 ± 0.05
Soft PatchUp	$\mathbf{38.68 \pm 0.34}$	1.87 ± 0.01	$\mathbf{38.11 \pm 0.29}$	1.84 ± 0.02	$\mathbf{29.81 \pm 0.24}$	1.46 ± 0.04
Hard PatchUp	$\underline{40.55 \pm 0.15}$	1.94 ± 0.01	$\underline{40.31 \pm 0.18}$	1.93 ± 0.01	$\underline{33.12 \pm 0.11}$	1.57 ± 0.05

Table 1: Classification on Tiny-ImageNet. Best performance result is shown in bold, second best is underlined (five times).

ImageNet (Table 1) where others are almost stable (more details in the Appendix). PatchUp and other methods reach the reported performance with WRN-28-10 model on Tiny-ImageNet after about 23 hours of training using one GPU (V100). However, Puzzle Mix needs 53 hours for training (more in Table 8-appx.).

PreActResNet18	**CIFAR-10** Test Error	**CIFAR-100** Test Error	**SVHN** Test Error
No Mixup	4.80 ± 0.14	24.62 ± 0.36	3.04 ± 0.09
Input Mixup (α = 1)	3.63 ± 0.20	22.33 ± 0.32	2.93 ± 0.22
ManifoldMixup (α = 1.5)	3.39 ± 0.05	21.40 ± 0.38	$\underline{2.44 \pm 0.06}$
Cutout	4.22 ± 0.05	23.39 ± 0.19	2.79 ± 0.12
DropBlock	5.04 ± 0.15	25.02 ± 0.26	2.96 ± 0.11
CutMix	3.52 ± 0.90	22.18 ± 0.18	3.04 ± 0.05
Puzzle Mix	3.16 ± 0.11	20.65 ± 0.21	2.66 ± 0.04
Soft PatchUp	$\underline{2.95 \pm 0.12}$	$\underline{19.95 \pm 0.18}$	2.55 ± 0.06
Hard PatchUp	$\mathbf{2.92 \pm 0.13}$	$\mathbf{19.12 \pm 0.17}$	$\mathbf{2.29 \pm 0.08}$
PreActResNet34			
No Mixup	4.64 ± 0.10	23.34 ± 0.27	3.09 ± 0.66
Input Mixup (α = 1)	3.26 ± 0.08	21.00 ± 0.44	2.86 ± 0.10
ManifoldMixup (α = 1.5)	2.93 ± 0.06	18.72 ± 0.31	$\underline{2.42 \pm 0.43}$
Cutout	3.69 ± 0.14	22.42 ± 0.08	2.65 ± 0.15
DropBlock	4.95 ± 0.19	23.74 ± 0.13	3.10 ± 0.08
CutMix	3.33 ± 0.07	19.94 ± 0.14	2.66 ± 0.05
Puzzle Mix	2.99 ± 0.07	19.97 ± 0.23	2.45 ± 0.08
Soft PatchUp	$\underline{2.57 \pm 0.06}$	$\underline{18.63 \pm 0.15}$	2.47 ± 0.08
Hard PatchUp	$\mathbf{2.53 \pm 0.05}$	$\mathbf{17.69 \pm 0.13}$	$\mathbf{2.12 \pm 0.02}$
WideResNet-28-10			
No Mixup	4.24 ± 0.14	22.44 ± 0.23	2.83 ± 0.08
Input Mixup (α = 1)	3.27 ± 0.35	18.73 ± 0.15	2.64 ± 0.16
ManifoldMixup (α = 1.5)	3.25 ± 0.18	18.35 ± 0.38	2.43 ± 0.10
Cutout	3.13 ± 0.12	20.16 ± 0.35	2.48 ± 0.15
DropBlock	4.18 ± 0.07	22.36 ± 0.15	2.73 ± 0.06
CutMix	3.15 ± 0.12	18.32 ± 0.19	2.43 ± 0.05
Puzzle Mix	$\underline{2.56 \pm 0.07}$	17.53 ± 0.22	2.43 ± 0.07
Soft PatchUp	2.61 ± 0.05	$\underline{16.73 \pm 0.11}$	$\mathbf{2.08 \pm 0.07}$
Hard PatchUp	$\mathbf{2.53 \pm 0.07}$	$\mathbf{16.13 \pm 0.19}$	$\underline{2.09 \pm 0.06}$

Table 2: Image classification error rates on CIFAR10/100 and SVHN (five runs). The best performance result is shown in bold, the second-best is underlined. The lower is better.

Since ManifoldMixup and Puzzle Mix show that they perform better than No-Mixup and Input Mixup on affine transformation and against adversarial attacks (Verma et al. 2019), we exclude No-Mixup and Input Mixup for the tasks in the following sections. Table 3 shows that *PatchUp* achieves a better error rate compared to other methods in the ImageNet2012 dataset (Russakovsky et al. 2015). To have a fair comparison, we used the same experiment setup proposed in the CutMix paper (300 epochs). For *Soft PatchUp* we set the gamma, patchup_block, and patchup_prob to 0.6, 7, and 1.0, respectively. For Hard *PatchUp* we set them to 0.5, 7, and 0.6, respectively.

Method	Top-1 Error (%)	Top-5 Error (%)
Vanilla*	23.68	7.05
Input Mixup*	22.58	6.40
Cutout*	22.93	6.66
ManifoldMixup*	22.50	6.21
FeatureCutMix*	21.80	6.06
CutMix*	21.40	5.92
PuzzleMix**	21.24	5.71
Soft PatchUp	$\underline{21.06}$	$\underline{5.57}$
Hard PatchUp	$\mathbf{20.75}$	$\mathbf{5.29}$

Table 3: Classification error rates on on the ILSVRC2012 (ImageNet2012) dataset. We include results from (Yun et al. 2019)* and (Kim, Choo, and Song 2020)**.

Robustness to Common Corruptions

The common corruption benchmark helps to evaluate the robustness of models against the input corruptions (Hendrycks and Dietterich 2019). It uses the 75 corruptions in 15 categories such that each has five levels of severity. We compare the methods robustness in Tiny-ImageNet-C for ResNet101/152, and WRN-28-10. So, we compute the sum of error denoted as E_c^f where s is the level of severity and c is corruption type such that $\mathrm{E}_c^f = \sum_{s=1}^{5} E_{s,c}^f$ (Hendrycks and Dietterich 2019). Fig.4 shows *Soft PatchUp* leads the best performance in Tiny-ImageNet-C and *Hard PatchUp* achieves the second-best. Figures 15a and b in Appendix show the comparison results in ResNet101 and 152.

Generalization on Deformed Images

Affine transformations on the test set provide novel deformed data that can be used to evaluate the robustness of a method on out-of-distribution samples (Verma et al. 2019). We trained PreActResNet34 and WRN-28-10 on the CIFAR100. Then, we created deformed test from CIFAR100

Transformation	cutout	CutMix	ManifoldMixup	Puzzle Mix	*Soft PatchUp*	*Hard PatchUp*
Rotate (-20, 20)	37.45 ± 0.53	35.42 ± 0.33	35.44 ± 0.57	31.70 ± 0.66	$\underline{31.14 \pm 0.52}$	$\mathbf{30.41 \pm 0.52}$
Rotate (-40, 40)	58.75 ± 0.10	57.83 ± 0.59	54.42 ± 0.95	54.04 ± 0.80	$\underline{53.42 \pm 0.42}$	$\mathbf{49.96 \pm 0.80}$
Shear (-28.6, 28.6)	36.55 ± 0.49	34.15 ± 0.47	34.15 ± 0.42	31.63 ± 0.69	$\mathbf{28.98 \pm 0.50}$	$\underline{29.57 \pm 0.41}$
Shear (-57.3, 57.3)	57.74 ± 0.57	53.64 ± 0.59	55.44 ± 0.68	52.49 ± 0.39	$\mathbf{49.10 \pm 0.53}$	$\underline{50.32 \pm 0.62}$
Scale (0.6)	72.99 ± 1.23	54.30 ± 1.27	78.99 ± 1.13	59.70 ± 1.24	$\mathbf{46.25 \pm 1.20}$	$\underline{50.06 \pm 2.69}$
Scale (0.8)	35.09 ± 0.86	29.38 ± 0.58	34.62 ± 0.37	30.37 ± 0.36	$\mathbf{23.94 \pm 0.21}$	$\underline{25.34 \pm 0.33}$
Scale (1.2)	42.31 ± 0.71	49.52 ± 2.04	$\underline{41.32 \pm 0.64}$	47.40 ± 1.44	43.41 ± 0.65	$\mathbf{38.01 \pm 0.70}$
Scale (1.4)	69.40 ± 0.90	78.66 ± 1.85	$\mathbf{65.94 \pm 0.75}$	77.59 ± 0.97	77.07 ± 1.19	$\underline{66.34 \pm 1.22}$

Table 4: Error rates in the test set on samples subject to affine transformations for PreActResNet34 trained on CIFAR100 (repeated each test for five trained models). Best in bold, second best is underlined. The lower number is better.

by applying some affine transformations. Table 4 shows that *PatchUp* provides the best performance on affine transformed test and better generalization in PreActResNet34. Table 10 (F-appx.) shows that the quality of representations is improved by *PatchUp* and it shows better generalization in deformed test on WRN-28-10. Generalization is significantly improved by *PatchUp* over existing methods by a large margin, as is the quality of representations.

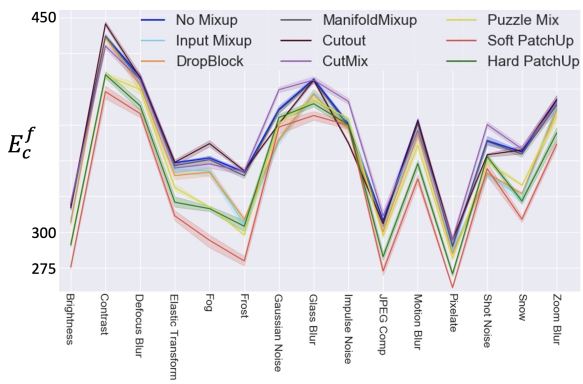
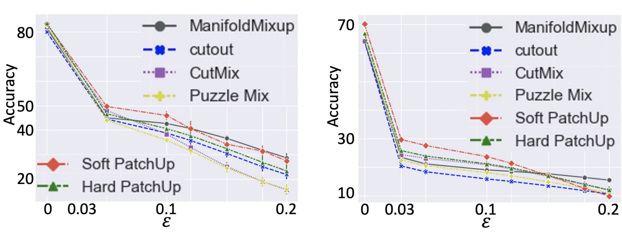


Figure 4: Errors in Tiny-ImageNet-C for WRN-28-10. The y-axis is the sum of error rates for each category. The x-axis represents the corruptions. The lower is better (five runs).



(a) CIFAR100. (b) Tiny-ImageNet.

Figure 5: WRN-28-10 robustness to FGSM attack (five runs). The y-axis is the accuracy against FGSM attack. The x-axis is the magnitude that controls the perturbation (ϵ).

Robustness to Adversarial Examples

Neural networks, trained with ERM, are often vulnerable to adversarial examples (Szegedy et al. 2013). Certain data-dependent methods can alleviate such fragility to adversarial examples by training the models with interpolated data. So, the robustness of a regularized model to adversarial examples can be considered as a criterion for comparison (Zhang et al. 2017; Verma et al. 2019). Fig.5 compares the performance of the methods on CIFAR100 and Tiny-ImageNet with adversarial examples created by the FGSM attack described in (Goodfellow, Shlens, and Szegedy 2014). Fig 13-appx. contains further comparison on PreActResNet18/34 and WRN-28-10 for CIFAR10 and SVHN with FGSM attacks. Table 5 shows the robust accuracy (in the range of $[0, 1]$) for the Foolbox benchmark (Rauber, Brendel, and Bethge 2018) against the 7-steps DeepFool (Moosavi-Dezfooli, Fawzi, and Frossard 2016), Decoupled Direction and Norm (DDN) (Rony et al. 2019), Carlini-Wagner (CW) (Carlini and Wagner 2017), and PGD_{L_∞} (Madry et al. 2019) attacks with $\epsilon = \frac{8}{255}$. We observe that *PatchUp* is more robust to adversarial attacks compared to other methods. While *Hard PatchUp* achieves better performance in terms of classification accuracy, *Soft PatchUp* seems to trade-off a slight loss of accuracy in order to achieve more robustness.

Methods	DeepFool	DDN	CW	PGD_{L_∞}
No Mixup	0.17 ± 0.01	0.18 ± 0.01	0.18 ± 0.01	0.17 ± 0.01
Input Mixup	$\underline{0.19 \pm 0.01}$	0.20 ± 0.01	0.20 ± 0.01	0.19 ± 0.01
ManifoldMixup	$\underline{\mathbf{0.20 \pm 0.01}}$	$\mathbf{0.20 \pm 0.01}$	$\underline{0.20 \pm 0.01}$	$\underline{0.19 \pm 0.01}$
Cutout	0.18 ± 0.01	0.185 ± 0.01	$\underline{0.19 \pm 0.01}$	$\underline{0.18 \pm 0.01}$
DropBlock	0.18 ± 0.01	0.186 ± 0.01	0.19 ± 0.01	0.18 ± 0.01
CutMix	0.17 ± 0.01	0.171 ± 0.01	0.17 ± 0.01	0.17 ± 0.01
Puzzle Mix	0.19 ± 0.01	0.191 ± 0.01	0.19 ± 0.01	0.19 ± 0.01
Soft PatchUp	0.19 ± 0.01	$\underline{0.20 \pm 0.01}$	$\mathbf{0.20 \pm 0.01}$	$\mathbf{0.19 \pm 0.01}$
Hard PatchUp	0.18 ± 0.01	$\underline{0.19 \pm 0.01}$	0.19 ± 0.01	0.18 ± 0.01

Table 5: Robust Accuracy of WRN-28-10 in the Tiny-ImageNet dataset against adversarial 7-steps attacks with $\epsilon = \frac{8}{255}$. The α is 0.2 for Mixup, ManifoldMixup, and Puzzle Mix. Best performance result is shown in bold, second best is underlined. The higher number is better (five runs).

Effect on Activations

To study the effect of the methods on the activations in the residual blocks, we compared the mean magnitude of feature activations in the residual blocks following (DeVries and Taylor 2017) in WRN-28-10 for CIFAR100 test set. We train the models with each method and then calculate the magnitudes of activations in the test set. The higher mean magnitude of features shows that the models tried to produce a wider variety of features in the residual blocks (DeVries and Taylor 2017). Our WRN-28-10 has a conv2d module followed by three residual blocks. We selected k randomly such that $k \in \{0, 1, 2, 3\}$. And, we apply the ManifoldMixup and *PatchUp* in either input space, the first conv2d, the first or second residual blocks (results in Fig.7). Figure 6 shows

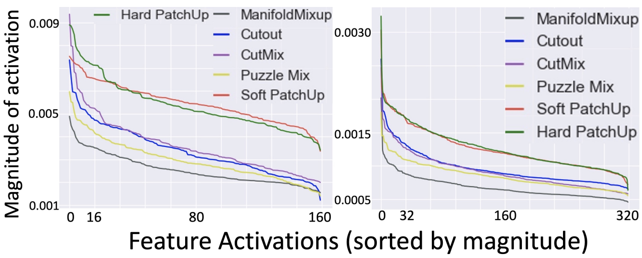


Figure 6: 1st (left) and 2st (right) Residual Block.

Figure 7: The effect of the methods on activations in WRN-28-10 for CIFAR100 test set on the first and second Residual Blocks. Each curve is the magnitude of the feature activations, sorted in descending value, and averaged over all test samples. The higher magnitude indicates a wider variety of the produced features by the model at each block.

that *PatchUp* produces more diverse features in the layers where we apply *PatchUp*. Fig. 14-appx. shows the results in first conv2d, first residual, second residual and third residual blocks. Since we are not applying the *PatchUp* in the third residual block, the mean magnitude of the feature activations are below, but very close to, Cutout and CutMix. This also shows that producing a wide variety of features can be an advantage for a model. But, according to our experiments, a larger magnitude of activations does not always lead better performance. Fig.7 shows that for ManifoldMixup, the mean magnitude of the feature activations is less than others. But, it performs better than Cutout and CutMix in most tasks.

Significance of loss terms and analysis of *k*

PatchUp uses the loss that is introduced in Equation 6. We can paraphrase the *PatchUp* learning objective for this ablation study as follow:

$$L(f) = \mathbb{E}_{(x_i,y_i)_{\sim P}} \mathbb{E}_{(x_j,y_j)_{\sim P}} \mathbb{E}_{\lambda \sim \text{Beta}(\alpha,\alpha)} \mathbb{E}_{k \sim \mathcal{S}} \left(L_1 + L_2\right) \tag{9}$$

where $L_1 = \text{Mix}_{p_u}[\ell(f_k(\phi_k), y_i), \ell(f_k(\phi_k), Y)]$ and $L_2 = \ell(f_k(\phi_k), W(y_i, y_j))$. We also show the effect of L_1 and L_2 in *PatchUp* loss. Table 6 shows the error rate on the validation set for WRN-28-10 on CIFAR100. This shows the summation of the L_1 and L_2 reduces error rate by $.1\%$ in *PatchUp*. We conducted an experiment to show the importance of random layer selection in PatchUp. Table 7 shows the contribution of the random selection of the layer in the overall performance of the method. In the left-most column 1/2/3 refers to PatchUp being applied to only one layer (more details in the section “B” in appx.).

As noted in section “2”, the PatchUp mask is “too strong” for the input space. Fig.8 shows that the PatchUp mask often drastically destroys the semantic concepts in the input images. Thus, we select one random rectangular patch in the input space (similar to CutMix). However, the learning objective in (k = 0) is still the PatchUp objective that is different from CutMix. The last row in table 7 shows the negative effect of applying PatchUp mask in the input space.

Simple WRN-28-10	Error Rate: 23.26 ± 0.59		
	Error with $L1$	Error with $L2$	Error with $L(f)$
Soft PatchUp	16.86 ± 0.67	16.87 ± 0.34	$\mathbf{16.75 \pm 0.29}$
Hard PatchUp	16.14 ± 0.23	16.79 ± 0.46	$\mathbf{16.02 \pm 0.36}$

Table 6: The validation error on CIFAR100 for WRN-28-10 with *Hard and Soft PatchUp*. The lower is better (five runs).

layer	Val Error	Test Error	Test NLL
1	18.43 ± 0.44	17.86 ± 0.16	0.73 ± 0.01
2	22.54 ± 0.80	21.42 ± 0.28	0.85 ± 0.01
3	26.17 ± 0.50	25.25 ± 0.14	1.14 ± 0.03
Random selection	$\mathbf{16.38 \pm 0.47}$	$\mathbf{16.13 \pm 0.20}$	$\mathbf{0.66 \pm 0.02}$
PatchUp Masks in $\mathbf{k = 0}$	16.98 ± 0.34	$16.90 \pm 0.0.21$	0.67 ± 0.01

Table 7: WRN-28-10 using *Hard PatchUp* on CIFAR100 (five runs).

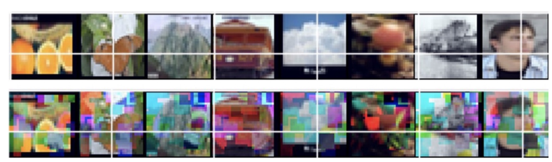


Figure 8: (top) Original samples. (bottom) Hard PatchUp output using PatchUp Binary Mask on input images.

5 Conclusion

We presented *PatchUp*, a simple and efficient regularizer scheme for CNNs that alleviates some of the drawbacks of the previous mixing-based regularizers. Our experimental results show that with the proposed approach, *PatchUp*, we can achieve state-of-the-art results on image classification tasks across different architectures and datasets. Similar to previous mixing based approaches, our approach also has the advantage of avoiding any added computational overhead. The strong test accuracy achieved by *PatchUp*, with no additional computational overhead, makes it particularly appealing for practical applications.

Acknowledgments

We would like to acknowledge Compute Canada and Calcul Quebec for providing computing resources used in this work. The authors would also like to thank Damien Scieur, Hannah Alsdurf, Alexia Jolicoeur-Martineau, and Yassine Yaakoubi for reviewing the manuscript. SC is supported by a Canada CIFAR AI Chair and an NSERC Discovery Grant.

References

Achille, A.; and Soatto, S. 2018. Information Dropout: Learning Optimal Representations Through Noisy Computation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12): 2897–2905.

Arpit, D.; Jastrzebski, S.; Ballas, N.; Krueger, D.; Bengio, E.; Kanwal, M. S.; Maharaj, T.; Fischer, A.; Courville, A.; Bengio, Y.; et al. 2017. A closer look at memorization in deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 233–242. JMLR. org.

Carlini, N.; and Wagner, D. 2017. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, 39–57. IEEE.

Chrabaszcz, P.; Loshchilov, I.; and Hutter, F. 2017. A Downsampled Variant of ImageNet as an Alternative to the CIFAR datasets. arXiv:1707.08819.

Cubuk, E. D.; Zoph, B.; Mane, D.; Vasudevan, V.; and Le, Q. V. 2019. AutoAugment: Learning Augmentation Policies from Data. arXiv:1805.09501.

DeVries, T.; and Taylor, G. W. 2017. Improved Regularization of Convolutional Neural Networks with Cutout. arXiv:1708.04552.

Gal, Y.; and Ghahramani, Z. 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, 1019–1027.

Ghiasi, G.; Lin, T.; and Le, Q. V. 2018. DropBlock: A regularization method for convolutional networks. *CoRR*, abs/1810.12890.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep learning*. MIT press.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and Harnessing Adversarial Examples. arXiv:1412.6572.

Guo, H.; Mao, Y.; and Zhang, R. 2018. MixUp as Locally Linear Out-Of-Manifold Regularization. *CoRR*, abs/1809.02499.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep Residual Learning for Image Recognition. *CoRR*, abs/1512.03385.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Identity Mappings in Deep Residual Networks. arXiv:1603.05027.

Hendrycks, D.; and Dietterich, T. 2019. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. arXiv:1903.12261.

Hendrycks, D.; Mu, N.; Cubuk, E. D.; Zoph, B.; Gilmer, J.; and Lakshminarayanan, B. 2020. AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty. arXiv:1912.02781.

Hinton, G.; Deng, L.; Yu, D.; Dahl, G. E.; Mohamed, A.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T. N.; and Kingsbury, B. 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 29(6): 82–97.

Kim, J.-H.; Choo, W.; and Song, H. O. 2020. Puzzle Mix: Exploiting Saliency and Local Statistics for Optimal Mixup. arXiv:2009.06962.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning Multiple Layers of Features from Tiny Images. *Master's thesis, Department of Computer Science, University of Toronto*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In Pereira, F.; Burges, C. J. C.; Bottou, L.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 25*, 1097–1105. Curran Associates, Inc.

Krueger, D.; Maharaj, T.; Kramár, J.; Pezeshki, M.; Ballas, N.; Ke, N. R.; Goyal, A.; Bengio, Y.; Courville, A.; and Pal, C. 2016. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*.

Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2019. Towards Deep Learning Models Resistant to Adversarial Attacks. arXiv:1706.06083.

Mai, Z.; Hu, G.; Chen, D.; Shen, F.; and Shen, H. T. 2019. MetaMixUp: Learning Adaptive Interpolation Policy of MixUp with Meta-Learning. arXiv:1908.10059.

Moosavi-Dezfooli, S.-M.; Fawzi, A.; and Frossard, P. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2574–2582.

Netzer, Y.; Wang, T.; Coates, A.; Bissacco, A.; Wu, B.; and Ng, A. Y. 2011. Reading Digits in Natural Images with Unsupervised Feature Learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.

Rauber, J.; Brendel, W.; and Bethge, M. 2018. Foolbox: A Python toolbox to benchmark the robustness of machine learning models. arXiv:1707.04131.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In Cortes, C.; Lawrence, N. D.; Lee, D. D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 28*, 91–99. Curran Associates, Inc.

Rony, J.; Hafemann, L. G.; Oliveira, L. S.; Ayed, I. B.; Sabourin, R.; and Granger, E. 2019. Decoupling Direction and Norm for Efficient Gradient-Based L2 Adversarial Attacks and Defenses. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.;

Berg, A. C.; and Fei-Fei, L. 2015. ImageNet Large Scale Visual Recognition Challenge. arXiv:1409.0575.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15: 1929–1958.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to Sequence Learning with Neural Networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS14, 31043112. Cambridge, MA, USA: MIT Press.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Tompson, J.; Goroshin, R.; Jain, A.; LeCun, Y.; and Bregler, C. 2014a. Efficient Object Localization Using Convolutional Networks. *CoRR*, abs/1411.4280.

Tompson, J.; Goroshin, R.; Jain, A.; LeCun, Y.; and Bregler, C. 2014b. Efficient Object Localization Using Convolutional Networks. *CoRR*, abs/1411.4280.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention Is All You Need. *CoRR*, abs/1706.03762.

Verma, V.; Lamb, A.; Beckham, C.; Najafi, A.; Mitliagkas, I.; Lopez-Paz, D.; and Bengio, Y. 2019. Manifold Mixup: Better Representations by Interpolating Hidden States. In Chaudhuri, K.; and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 6438–6447. Long Beach, California, USA: PMLR.

Yun, S.; Han, D.; Oh, S. J.; Chun, S.; Choe, J.; and Yoo, Y. 2019. CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features. arXiv:1905.04899.

Zagoruyko, S.; and Komodakis, N. 2017. Wide Residual Networks. arXiv:1605.07146.

Zhang, H.; Cisse, M.; Dauphin, Y. N.; and Lopez-Paz, D. 2017. mixup: Beyond Empirical Risk Minimization. arXiv:1710.09412.