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

Mojtaba Faramarzi¹, ², Mohammad Amini¹, ³, Akilesh Badrinaaraaayanan¹, ², Vikas Verma¹, ², ⁴, Sarath Chandar¹, ⁵

¹Mila - Quebec AI Institute
²University of Montreal
³McGill University
⁴Aalto University, Finland
⁵École 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.

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 \( \lambda \sim \text{Beta}(\alpha, \alpha) \) 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 \( \lambda \) in Mixup is sampled from
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 a prior distribution, 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 a prior distribution, 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 a prior distribution, leading to two training samples which have the same inputs but different targets.

Different from the interpolation-based regularizers discussed above, Cutout ( DeVries and Taylor 2017) drops the contiguous regions from the input feature map. 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 $\gamma$ 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, \textit{block$_{size}$}. Hence, the altering probabilities are readjusted using the following formula (Ghiasi, Lin, and Le 2018):

$$\gamma_{a_j} = \frac{\gamma \times (\text{feature map's area})}{(\text{block$_{size}$ area}) \times (\text{valid region to build block})} \tag{2}$$

where the area of the feature map and block are the \textit{feat$_{size}$} and \textit{block$_{size}$}, respectively, and the valid region to build the block is $(\text{feat$_{size}$} - \text{block$_{size}$} + 1)^2$. For each feature in the feature map, we sample from Bernoulli$(\gamma_{a_j})$. 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 \textit{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 \textit{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.

\textbf{PatchUp Operation}

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

Consider two samples $x_i$ and $x_j$. The \textit{Hard PatchUp} operation at layer $k$ is defined as follows:

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

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

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

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

$$\phi_{\text{soft}}(g_k(x_i), g_k(x_j)) = M \odot g_k(x_i) + \text{Mix}_\lambda[(1 - M) \odot g_k(x_i), (1 - M) \odot g_k(x_j)], \tag{5}$$

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

\textbf{Learning Objective}

After applying the \textit{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 \textit{PatchUp} after the $k$-th layer. Mathematically, the CNN with \textit{PatchUp} minimizes the following loss function:

$$L(f) = \mathbb{E}_{(x_i, y_i) \sim P_{\mathcal{X}}} \mathbb{E}_{(x_j, y_j) \sim P_{\mathcal{X}}} \mathbb{E}_{\mathcal{S} \sim \text{Beta}(\alpha, \alpha)} \mathbb{E}_{k \sim \mathcal{S}} \text{Mix}_{p_a}[\ell(f_k(\phi_k), y_i), \ell(f_k(\phi_k), Y)] + \ell(f_k(\phi_k), W(y_i, y_j)), \tag{6}$$

where $p_a$ is the fraction of the unchanged features from feature maps in $g_k(x_i)$ and $\mathcal{S}$ is the set of layers where \textit{PatchUp} is applied randomly. $\phi$ is \$\phi_{\text{hard}}$ for \textit{Hard PatchUp} and $\phi_{\text{soft}}$ for \textit{Soft PatchUp}.

$Y$ is the target corresponding to the changed features. In the case of \textit{Hard PatchUp}, $Y = y_j$, and in the case of \textit{Soft PatchUp}, $Y = \text{Mix}_\lambda(y_i, y_j)$. $W(y_i, y_j)$ calculates the re-weighted target according to the interpolation policy for $y_i$ and $y_j$. $W$ for \textit{Hard PatchUp} and \textit{Soft PatchUp} is defined as follows:

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

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

The \textit{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).

\textbf{PatchUp in Input Space}

By setting $k = 0$, we can apply \textit{PatchUp} to only the input space. When we apply \textit{PatchUp} to the input space, only the \textit{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 \textit{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”).

\textbf{3 Relation to Similar Methods}

\textit{PatchUp Vs. ManifoldMixup}: \textit{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. \textit{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 \textit{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, \textit{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 $\lambda$ sampled from the beta distribution. Fig.2 is a simple diagrammatic description of how
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, CIFAR100 (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, ResNet512, 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 $\alpha$ to 2 in PatchUp. PatchUp has $\text{patchup\_prob}$, $\gamma$ and $\text{block\_size}$ as additional hyperparameters. $\text{patchup\_prob}$ 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 $\text{patchup\_prob}$, $\gamma$, and $\text{block\_size}$ as 0.7, 0.5, and 7, respectively. Soft PatchUp achieves the best performance with $\text{patchup\_prob}$, $\gamma$, and $\text{block\_size}$ as 1.0, 0.75, and 7, respectively.

### 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 provides 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 $\alpha$ when we have more training classes. It is notable that using the same $\alpha$, that is used in CIFAR or SVHN, leads to worst performance than No-Mixup in Tiny-
Table 1: Classification on Tiny-ImageNet. Best performance result is shown in bold, second best is underlined (five times).

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.

Table 3: Classification error rates 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 $E_{c}^{s}$ where $s$ is the level of severity and $c$ is corruption type such that $E_{c} = \sum_{s=1}^{5} E_{c}^{s}$ (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
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 $\text{PGD}_{L_{\infty}}$ (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.

Table 5: Robust Accuracy of WRN-28-10 in the Tiny-ImageNet dataset against adversarial 7-steps attacks with $\epsilon = \frac{8}{255}$. The $\alpha$ 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).

<table>
<thead>
<tr>
<th>Methods</th>
<th>DeepFool</th>
<th>DDN</th>
<th>CW</th>
<th>PGD$<em>{L</em>{\infty}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Mixup</td>
<td>0.17 ± 0.01</td>
<td>0.18 ± 0.01</td>
<td>0.18 ± 0.01</td>
<td>0.17 ± 0.01</td>
</tr>
<tr>
<td>Input Mixup</td>
<td>0.19 ± 0.01</td>
<td>0.20 ± 0.01</td>
<td>0.20 ± 0.01</td>
<td>0.19 ± 0.01</td>
</tr>
<tr>
<td>ManifoldMixup</td>
<td>0.20 ± 0.01</td>
<td>0.20 ± 0.01</td>
<td>0.20 ± 0.01</td>
<td>0.19 ± 0.01</td>
</tr>
<tr>
<td>Cutout</td>
<td>0.18 ± 0.01</td>
<td>0.18 ± 0.01</td>
<td>0.19 ± 0.01</td>
<td>0.18 ± 0.01</td>
</tr>
<tr>
<td>DropBlock</td>
<td>0.17 ± 0.01</td>
<td>0.17 ± 0.01</td>
<td>0.17 ± 0.01</td>
<td>0.17 ± 0.01</td>
</tr>
<tr>
<td>CutMix</td>
<td>0.19 ± 0.01</td>
<td>0.19 ± 0.01</td>
<td>0.19 ± 0.01</td>
<td>0.19 ± 0.01</td>
</tr>
<tr>
<td>Puzzle Mix</td>
<td>0.20 ± 0.01</td>
<td>0.20 ± 0.01</td>
<td>0.20 ± 0.01</td>
<td>0.19 ± 0.01</td>
</tr>
<tr>
<td>Hard PatchUp</td>
<td>0.18 ± 0.01</td>
<td>0.19 ± 0.01</td>
<td>0.19 ± 0.01</td>
<td>0.18 ± 0.01</td>
</tr>
</tbody>
</table>

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 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:

$$L(f) = E(x, y) \sum_p E(x, y) \sum_p E_{k\sim Beta(\alpha, \alpha)} E_{k\sim S}(L_1 + L_2)$$

(9)

where $L_1 = \text{Mix}_{p_0}[\ell(f_k(\phi_k), y), \ell(f_k(\phi_k), Y)]$ and $L_2 = \ell(f_k(\phi_k), W(y_1, y_j))$. 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.

<table>
<thead>
<tr>
<th>Simple</th>
<th>Error Rate:</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRN-28-10</td>
<td>23.26 ± 0.59</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error with L1</th>
<th>Error with L2</th>
<th>Error with $L(f)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soft PatchUp</td>
<td>16.86 ± 0.67</td>
<td>16.87 ± 0.34</td>
</tr>
<tr>
<td>Hard PatchUp</td>
<td>16.14 ± 0.23</td>
<td>16.79 ± 0.46</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>layer</th>
<th>Val Error</th>
<th>Test Error</th>
<th>Test NLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18.43 ± 0.44</td>
<td>17.86 ± 0.16</td>
<td>0.73 ± 0.01</td>
</tr>
<tr>
<td>2</td>
<td>22.54 ± 0.80</td>
<td>21.42 ± 0.28</td>
<td>0.85 ± 0.01</td>
</tr>
<tr>
<td>3</td>
<td>26.17 ± 0.50</td>
<td>25.25 ± 0.14</td>
<td>1.14 ± 0.03</td>
</tr>
</tbody>
</table>

Random selection

| 16.38 ± 0.47 | 16.13 ± 0.20 | 0.66 ± 0.02 |

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

| PatchUp Masks in $k = 0$ | 16.98 ± 0.34 | 16.90 ± 0.02 | 0.67 ± 0.01 |

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


