PatchUp: A Regularization Technique for Convolutional Neural Networks

Mojtaba Faramarzi1,2, Mohammad Amini1,3, Akilesh Badrinaaraayanan1,2, Vikas Verma1,2,4

Sarath Chandar1,5,6

1Mila - Quebec AI Institute, 2University of Montreal, 3McGill University, 4Aalto University, Finland, 5École Polytechnique de Montréal, 6Canada CIFAR AI Chair

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 like Mixup and CutMix. 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 CIFAR-10, CIFAR-100, and SVHN datasets with PreactResnet18, PreactResnet34, and WideResnet-28-10 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 better generalization to affine transformations of 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 [1], image classification [1–3], speech recognition [4] and natural language understanding [5, 6]. 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 [7, 8]. 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 [9]. 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 [10]. Earlier work in this area has been more focused on the data-independent techniques such as Dropout [11], SpatialDropout [12], and DropBlock [13]. Dropout performs well on fully connected layers [14]. However, it is less effective on convolutional layers [15]. 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 [15] addresses this issue by dropping the entire feature map from a convolutional layer. DropBlock [13] further improves SpatialDropout by dropping random continuous feature blocks from feature maps instead of dropping the entire feature map in the convolutional layers.
Recent works show that data-dependent regularizers can achieve better generalization for CNN models. Mixup [16], 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:

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

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 [10]; 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, Mai et al. [17] used a meta-learning approach to learn \( \lambda \) with a lower possibility of causing such collisions. However, this meta-learning approach adds significant computation complexity. ManifoldMixup [18] 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.

Different from the interpolation based regularizers discussed above, cutout [19] 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 [20] 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. CutMix also improves the generalization performance of a very deep and wide CNN model such as PyramidNet. According to the CutMix paper, applying CutMix at the latent space, Feature CutMix, is not as effective as applying CutMix in the input space [20].

In this work, we propose PatchUp which is a regularization technique 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 experiments verify that Hard PatchUp achieves a better generalization performance compared to other state-of-the-art regularization techniques for CNNs such as Mixup, cutout, CutMix and ManifoldMixup on CIFAR-10, CIFAR-100, and SVHN datasets. Soft PatchUp achieves the second-best performance on CIFAR-10, CIFAR-100 with PreactResnet18, PreactResnet34, and WideResnet-28-10 models while achieving comparable results to ManifoldMixup on SVHN with PreactResnet18 and PreactResnet34. Furthermore, PatchUp provides significant improvements in the generalization on deformed images and better robustness against Fast Gradient Sign Method (FGSM) adversarial attack.

## 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 [18]. 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.

### 2.1 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 [13]. 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, \( \text{block}_\text{size} \). Hence, the altering probabilities are
Figure 1: PatchUp process for two hidden representations associated with two samples randomly selected in the mini-batch \((a, b)\). \(X_1 = g^i_k(a)\) and \(X_2 = g^i_k(b)\) 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.

readjusted using the following formula \[13\]:

\[
\gamma_{\text{adj}} = \frac{\gamma \times \text{area of feature map}}{\text{area of block} \times \text{valid region to build block}}
\] (2)

where the area of the feature map and block are the \(\text{feat\_size}^2\) and \(\text{block\_size}^2\), 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 \(\text{Bernoulli}(\gamma_{\text{adj}})\). 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 \(\text{block\_size}\) 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. Figure 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).

2.2 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:

\[
\phi_{\text{hard}}(g_k(x_i), g_k(x_j)) = M \odot g_k(x_i) + (1 - M) \odot g_k(x_j),
\]

(3)

where \(\odot\) is known as the element-wise multiplication operation and \(M\) is the binary mask described in section 2.1.

To define Soft PatchUp operation, we first define the mixing operation for any two vectors \(a\) and \(b\) as follows:

\[
\text{Mix}_\lambda(a, b) = \lambda \cdot a + (1 - \lambda) \cdot b,
\]

(4)

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

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

(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. Consequently, it controls the strength of interpolation \[16\]. Both PatchUp operations are illustrated in Figure 1.

2.3 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.
Again, 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:

\[
L(f) = \mathbb{E}_{(x_i, y_i) \sim p} \mathbb{E}_{(x_j, y_j) \sim p} \mathbb{E}_{\lambda \sim \text{Beta}(\alpha, \alpha)} \mathbb{E}_{k \sim \mathcal{S}} \text{Mix}_{\rho_\text{free}}[(\ell(f_k(\phi_k), y_i), \ell(f_k(\phi_k), Y)) + \ell(f_k(\phi_k), W(y_i, y_j))].
\] (6)

where \(p_\text{free}\) 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. \(\phi\) is defined as follows:

- for \(\text{Hard PatchUp}\), \(Y = y_j\), and
- for \(\text{Soft PatchUp}\), \(Y = \text{Mix}_\lambda(y_i, y_j)\).

\(W\) is applied randomly, \(\lambda\) is applied randomly. To apply PatchUp to the input space, only the Hard PatchUp operation is used since swapping in the input space provides better generalization compared to mixing \([20]\). Only one random rectangular patch is selected in the input space (similar to CutMix) because the PatchUp binary mask is too strong for the input space, which has only three channels, compared to hidden layers in which each layer has numerous channels.

### 2.4 PatchUp in Input Space

When \(k = 0\), PatchUp only gets applied to the input space. To apply PatchUp to the input space, only the Hard PatchUp operation is used since swapping in the input space provides better generalization compared to mixing \([20]\). Only one random rectangular patch is selected in the input space (similar to CutMix) because the PatchUp binary mask is too strong for the input space, which has only three channels, compared to hidden layers in which each layer has numerous channels.

### 3 Relation to Other Methods

**PatchUp Vs. ManifoldMixup:** Both PatchUp and ManifoldMixup try to improve the generalization of a model by combining the latent representations of a pair of examples. ManifoldMixup combines two hidden representations by using the mixing operation defined in Equation 4 which produces a new latent representation in a linear way for a pair of two hidden representations. PatchUp uses a more complex approach to find a combination of two hidden representations, ensuring that a more diverse subspace of the hidden space gets explored. To understand the behaviour and the limitation that exist in the ManifoldMixup, assume that we have a 3D hidden space representation as illustrated in figure [2]. Figure [2] 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. On the other hand, 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. Appendix C provides the mathematical and experimental justifications to show the comparative advantage of PatchUp over ManifoldMixup.
PatchUp Vs. CutMix: The CutMix strategy is to cut and fill some parts of the selected pairs instead of using interpolation for creating a new sample in the input space. Therefore, the CutMix method has less potential for a manifold intrusion problem, however, CutMix may still suffer from a manifold intrusion problem. Figure 3 shows two samples with small portions that correspond to their labels. CutMix cuts and fills the rectangular parts of the selected image randomly. 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 fraction of the images that is swapped. In contrast, these scenarios cannot occur in PatchUp since it works in the hidden representation space. 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. Figure A.6 shows an example of CutMix Mask and PatchUp mask in input space and hidden representation space, respectively.

Feature-CutMix applies CutMix in the latent space. According to Yun et al. [20], Feature-CutMix is not as effective as CutMix. Both the learning objective of PatchUp, as well as the binary mask selection are different from Feature-CutMix.

4 Experiments

To evaluate the generalization improvements that PatchUp can provide with either Hard or Soft PatchUp, we applied PatchUp to image classification tasks on CIFAR-10, CIFAR-100 [21], and SVHN [22] datasets with PreActResNet18, PreActResNet34, and WideResNet-28-10 models. We used the same set of base hyper-parameters for all the models to be able to compare and evaluate the generalization improvements due to different regularization methods in a fair way. Appendix D explains the experiment setup and the hyper-parameter tuning processes in detail. PatchUp adds $\text{patchup\_prob}$, $\gamma$ and $\text{block\_size}$ to the set of hyper-parameters. $\text{patchup\_prob}$ is the probability that PatchUp is performed for a given mini-batch. We set $\alpha$ to 2 in PatchUp. And, based on the hyper-parameter tuning described in Appendix D, 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.

4.1 Generalization on Image Classification

Table 1 shows the comparison of the generalization performance of PatchUp with five other state-of-the-art regularization techniques on the CIFAR-10 and CIFAR-100 datasets. Our experiments show that Hard PatchUp leads to a lower test error for all the models on both CIFAR-10 and CIFAR-100. Soft PatchUp has the second-best result for this task. From Table 1, we see that PatchUp provides a significant improvement over previous state-of-the-art regularization techniques across different architectures. We observed that Soft PatchUp has the second-best performance across all the architectures indicating the potential that Soft PatchUp is beneficial in fine-grained classification. Table 2 shows that Hard PatchUp achieves the best top-1 error across the different models. Our experiments show that ManifoldMixup has the second-best performance for PreActResNet18 on SVHN. Soft PatchUp also achieves the second-best performance for WideResNet-28-10 on SVHN. It is also worth noting that Soft PatchUp performs reasonably well and is comparable to ManifoldMixup for PreActResNet34 on SVHN.

4.2 Generalization on Deformed Images

Regularization methods aim to improve the generalization of a model to unseen data. Applying affine transformations on the test set can provide novel deformed data that can be used to evaluate and compare the minimality and sufficiency of the representations learned by models with state-of-the-art regularization techniques [18]. We trained PreActResNet34 and WideResNet-28-10 on the CIFAR100 dataset. And then, we created deformed test sets from CIFAR100 by applying random rotations, random shearings, and different rescalings. Table 3 shows that PatchUp provides the best performance on affine transformed test sets and better generalization in PreActResNet34. Table 6 in Appendix E illustrates that the quality of representations is improved by PatchUp and it also shows

\footnote{The code to reproduce all the results is available at https://github.com/chandar-lab/PatchUp}
Table 1: Image classification task error rates on CIFAR-10 and CIFAR-100. We run experiments five times to report the mean and the standard deviation of errors and neg-log-likelihoods. Best performance result is shown in bold, second best is underlined. The lower number is better.

| Model               | PreActResNet18 | PreActResNet34 | WideResNet-28-10 |
|---------------------|----------------|----------------|------------------|
| No Mixup            | 4.800 ± 0.135  | 0.184 ± 0.004  | 22.342 ± 0.269   |
| Input Mixup (α = 1) | 4.618 ± 0.141  | 0.150 ± 0.012  | 21.950 ± 0.180   |
| ManifoldMixup (α = 1.5) | 3.809 ± 0.048 | 0.147 ± 0.016  | 23.386 ± 0.185   |
| Cutout              | 4.218 ± 0.046  | 0.158 ± 0.005  | 21.396 ± 0.384   |
| DropBlock           | 5.038 ± 0.147  | 0.185 ± 0.005  | 25.022 ± 0.259   |
| CutMix              | 3.518 ± 0.098  | 0.131 ± 0.002  | 22.184 ± 0.176   |
| Soft PatchUp        | 2.956 ± 0.119  | 0.169 ± 0.031  | 19.950 ± 0.180   |
| Hard PatchUp        | 2.918 ± 0.131  | 0.146 ± 0.078  | 19.120 ± 0.172   |

Table 2: Error rates comparison on SVHN. We run experiments five times to report the mean and the standard deviation of errors and neg-log-likelihoods. Best performance result is shown in bold, second best is underlined. The lower number is better.

| Model               | PreActResNet18 | PreActResNet34 | WideResNet-28-10 |
|---------------------|----------------|----------------|------------------|
| No Mixup            | 4.640 ± 0.099  | 0.204 ± 0.004  | 21.950 ± 0.180   |
| Input Mixup (α = 1) | 4.260 ± 0.075  | 0.175 ± 0.004  | 20.000 ± 0.140   |
| ManifoldMixup (α = 1.5) | 2.926 ± 0.062 | 0.124 ± 0.004  | 18.724 ± 0.305   |
| Cutout              | 3.600 ± 0.141  | 0.150 ± 0.012  | 22.420 ± 0.075   |
| DropBlock           | 4.950 ± 0.188  | 0.221 ± 0.010  | 23.744 ± 0.125   |
| CutMix              | 3.332 ± 0.071  | 0.142 ± 0.004  | 19.944 ± 0.141   |
| Soft PatchUp        | 2.570 ± 0.062  | 0.108 ± 0.005  | 18.630 ± 0.153   |
| Hard PatchUp        | 2.534 ± 0.048  | 0.108 ± 0.005  | 17.692 ± 0.125   |

4.3 Robustness to Adversarial Examples

Since neural networks are trained based on Empirical Risk Minimization (ERM), slight changes in the data distribution have a significant effect on the model performance [23][10]. Such unseen data used to confuse the models are known as adversarial examples. Certain data-dependent regularization techniques can alleviate such fragility to adversarial examples by training the models with interpolated data. Therefore, the robustness of a regularized model to adversarial examples can be considered as a criterion for comparison [16][18][29]. To evaluate the robustness of PatchUp against adversarial attacks, we compared the performance of PreActResNet18, PreActResNet34, and WideResNet-28-10 on CIFAR-10 and CIFAR-100 with adversarial examples created by the FGSM attack described in [24]. We compared the performance of WideResNet-28-10 for SVHN against the FGSM attack. Figure [A.1] in the appendix shows further comparisons. Based on the results, we can see that Soft PatchUp is more robust to adversarial attacks when compared to other regularization methods. Hard PatchUp and ManifoldMixup achieve better generalization in deformed test sets on WideResNet-28-10. Generalization is significantly improved by PatchUp, as are the quality of representations learned by PatchUp as demonstrated by this experiment.
To study the effect of the state-of-the-art regularization techniques on the activations in the residual blocks, we compared the mean magnitude of feature activations in the residual blocks following [19] in WideResNet28-10 for the test set in CIFAR-100. We first train the models with regularization techniques and then calculate the magnitudes of activations in the validation set. The higher mean magnitude of features shows that the models tried to produce a wider variety of features in the residual blocks [19]. Our WideResNet28-10 has a conv2d module followed by three residual blocks. For this ablation study, we selected $k$ randomly such that $k \in \{1, 2, 3\}$. Therefore, we apply the ManifoldMixup and PatchUp in either input space, first conv2d, first residual block, or second residual block. Figure 5 illustrates the comparison of ManifoldMixup, cutout, CutMix, Soft PatchUp, and Hard PatchUp. Figures 5a and 5b illustrate that PatchUp produces more diverse features in the layers where we apply PatchUp. In Appendix C, Figure A.12 shows this ablation study results in first conv2d, first residual block, second residual block and third residual block. 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 experiment also shows that producing a wide variety of features can be an advantage for a model. However, according to our experiments, a larger magnitude of activations does not always mean better performance. Figure 5c shows that for ManifoldMixup, the mean magnitude of the feature activations is less than other approaches. But, it performs better than cutout and CutMix in image classification, affine transformations, and FGSM attacks.

4.4 Effect on Activations

To study the effect of the state-of-the-art regularization techniques on the activations in the residual blocks, we compared the mean magnitude of feature activations in the residual blocks following [19] in WideResNet28-10 for the test set in CIFAR-100. We repeated each test for five trained models to report the mean and the standard deviation of errors. Best performance result is shown in bold, second best is underlined. The lower number is better.

| Transformation | cutout | CutMix | ManifoldMixup | Soft PatchUp | Hard PatchUp |
|----------------|--------|--------|---------------|--------------|--------------|
| Rotate (-20, 20) | 37.444 ± 0.526 | 35.418 ± 0.328 | 35.444 ± 0.572 | 31.136 ± 0.524 | **30.406 ± 0.520** |
| Rotate (-40, 40) | 58.752 ± 0.995 | 57.830 ± 0.586 | 54.424 ± 0.946 | 53.422 ± 0.429 | **49.956 ± 0.798** |
| Shear (-28.6, 28.6) | 36.552 ± 0.487 | 34.148 ± 0.473 | 33.150 ± 0.416 | **28.984 ± 0.497** | 29.574 ± 0.410 |
| Shear (-57.3, 57.3) | 57.736 ± 0.574 | 53.640 ± 0.587 | 55.444 ± 0.683 | **49.102 ± 0.532** | 50.318 ± 0.616 |
| Scale (0.6) | 72.994 ± 1.231 | 54.304 ± 1.268 | 78.998 ± 1.126 | **46.246 ± 1.204** | 50.062 ± 2.062 |
| Scale (0.8) | 35.092 ± 0.857 | 29.380 ± 0.577 | 34.624 ± 0.370 | **23.942 ± 0.212** | 25.338 ± 0.328 |
| Scale (1.2) | 42.310 ± 0.706 | 49.522 ± 2.035 | 41.322 ± 0.638 | 43.414 ± 0.652 | **38.002 ± 0.703** |
| Scale (1.4) | 69.404 ± 0.901 | 78.664 ± 1.854 | **65.938 ± 0.751** | 77.068 ± 1.189 | 66.338 ± 1.219 |

(a) Comparison on PreActResNet18 for CIFAR-100. 
(b) Comparison on WideResNet28-10 for SVHN.

Figure 4: Robustness to FGSM attack. Plots are based on the mean and standard deviation of the accuracy of five trained models for each method against FGSM attack. The x-axis represents $\epsilon$ which is the magnitude that controls the perturbation.

The second-best performance in most experiments. 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.
4.5 Significance of loss terms

PatchUp uses the loss that is introduced in Equation 5. We can paraphrase the PatchUp learning objective for this ablation study as follow:

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

(9)

where

\[
L_1 = \text{Mix}_{p_{\phi}}[\ell(f_k(\phi_k), y_i), \ell(f_k(\phi_k), Y)],
\]

(10)

\[
L_2 = \ell(f_k(\phi_k), W(y_i, y_j)),
\]

(11)

This section is an ablation study to show the effect of \( L_1 \) and \( L_2 \) in PatchUp loss. Table 4 shows the error rate on the validation set for WideResNet-28-10 on CIFAR-100. This study shows that the summation of the \( L_1 \) and \( L_2 \) reduces error rate by \( .1\% \) in both Soft PatchUp and Hard PatchUp.

Table 4: The error rate on the validation set on CIFAR-100 for WideResNet-28-10 with Hard PatchUp and Soft PatchUp. The result is the mean and standard deviation of the experiment for five runs. A smaller number indicates better performance.

| Simple WideResNet-28-10 Error Rate: 23.256 ± 0.586 |
|-----------------------------------------------|
| Error rates with \( L_1 \) Error rates with \( L_2 \) Error rates with \( L(f) \) |
| Soft PatchUp 16.856 ± 0.666 16.865 ± 0.339 16.75 ± 0.291 |
| Hard PatchUp 16.135 ± 0.229 16.79 ± 0.457 16.02 ± 0.358 |

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

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc., 2012.

[2] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, Advances in Neural Information Processing Systems 28, pages 91–99. Curran Associates, Inc., 2015.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. CoRR, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[4] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine, 29(6):82–97, Nov 2012. ISSN 1558-0792. doi: 10.1109/MSP.2012.2205597.

[5] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS’14, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. CoRR, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

[7] Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 233–242. JMLR. org, 2017.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

[9] A. Achille and S. Soatto. Information dropout: Learning optimal representations through noisy computation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 40(12):2897–2905, Dec 2018. ISSN 1939-3539. doi: 10.1109/TPAMI.2017.2784440.

[10] Hongyu Guo, Yongyi Mao, and Richong Zhang. Mixup as locally linear out-of-manifold regularization. CoRR, abs/1809.02499, 2018. URL http://arxiv.org/abs/1809.02499.

[11] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15:1929–1958, 06 2014.

[12] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. CoRR, abs/1411.4280, 2014. URL http://arxiv.org/abs/1411.4280.

[13] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. Dropblock: A regularization method for convolutional networks. CoRR, abs/1810.12890, 2018. URL http://arxiv.org/abs/1810.12890.

[14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.
[15] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. CoRR, abs/1411.4280, 2014. URL http://arxiv.org/abs/1411.4280.

[16] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization, 2017.

[17] Zhijun Mai, Guosheng Hu, Dexiong Chen, Fumin Shen, and Heng Tao Shen. Metamixup: Learning adaptive interpolation policy of mixup with meta-learning, 2019.

[18] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pages 6438–6447, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/verma19a.html.

[19] Terrance DeVries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout, 2017.

[20] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features, 2019.

[21] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[22] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011, 2011.

[23] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199, 2013.

[24] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2014.

[25] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. CoRR, abs/1503.02406, 2015. URL http://arxiv.org/abs/1503.02406.
Appendices

A Algorithm

In this appendix, we provide a detailed algorithm for implementing PatchUp. As with most regularization techniques, PatchUp also has two modes (either inference or training). It also needs the combining type (either Soft PatchUp or Hard PatchUp), γ, and block_size. This algorithm shows how PatchUp generates a new hidden representation from \((g_k(x_i), y_i)\) and \((g_k(x_j), y_j)\). Lines 4 to 9 in the algorithm are the binary mask creation process used in both Soft PatchUp and Hard PatchUp.

Algorithm 1 PatchUp

Input:

\((g_k(x_i), y_i)\): the hidden representation for the sample \((x_i, y_i)\) at layer \(k\).
\((g_k(x_j), y_j)\): the hidden representation for the sample \((x_j, y_j)\) at layer \(k\).

mode: either inference or training.
mixing_type: soft or hard.
γ: the probability of altering a feature.
block_size: the size of each block in the binary mask.

Output

\(y_i, y_j\): original labels for samples \(i\) and \(j\).
\(H'\): the new hidden representation computed by PatchUp.
\(p_u\): The portion of the feature maps that remained unchanged.
\(Y\): the target corresponding to the changed features.
\(W\): re-weighted target according to the interpolation policy.

1: if mode == Inference then
2: return \((g_k(x_i), y_i), (g_k(x_j), y_j)\)
3: end if
4: kernel_size ← \((\text{block size}, \text{block size})\)
5: stride ← \((1, 1)\)
6: padding ← \((\frac{\text{block size}}{2}, \frac{\text{block size}}{2})\)
7: \(\gamma_{\text{adj}}\) ← adjust \(\gamma\) using (2)
8: \(\text{holes} \leftarrow \max\_\text{pool2d}(\text{Bernoulli}(\gamma_{\text{adj}}), \text{kernel size}, \text{stride}, \text{padding})\)
9: \(\text{Mask} \leftarrow 1 - \text{holes}\)
10: \(\text{unchanged} \leftarrow \text{Mask} \odot g_k(x_i)\)
11: \(p_u\) ← calculate the portion of changed features map.
12: Patch_i ← \(\text{holes} \odot g_k(x_i)\)
13: Patch_j ← \(\text{holes} \odot g_k(x_j)\)
14: if \(\text{mixing type} == \text{hard}\) then
15: Patch_i ← Patch_j
16: \(Y \leftarrow y_j\)
17: \(W \leftarrow W_{\text{hard}}(y_i, y_j)\) using (8)
18: else if \(\text{mixing type} == \text{soft}\) then
19: \(\lambda \sim \text{Beta}(\alpha, \alpha)\)
20: \(Y \leftarrow \text{Mix}_\lambda(y_i, y_j)\)
21: \(W \leftarrow W_{\text{soft}}(y_i, y_j)\) using (8)
22: Patch_i ← \(\text{Mix}_\lambda(\text{Patch}_i, \text{Patch}_j)\)
23: end if
24: \(H' \leftarrow \text{unchanged} + \text{Patch}_i\)
25: return \(y_i, y_j, H', p_u, Y, W\)

Figure A.7 briefly illustrates and summarizes the binary mask creation process in PatchUp. Lines 11 to 25 correspond to the interpolation and combination of hidden representations in the mini-batch in PatchUp. Figure A.6 compares the masks generated by PatchUp and CutMix.
Figure A.6: Mask sampling in PatchUp is applied in the hidden state, compared to CutMix which is applied in the input space. Red areas show the blocks that should be altered.

Figure A.7: PatchUp mask creation process \((\text{block}\_\text{size} = 5)\). The left matrix shows the process of feature selection from feature maps. By using a \(\max\_\text{pool}2d\) function, we can create blocks around selected features. The \(\max\_\text{pool}2d\) function uses \(\text{stride} = (1, 1)\), \(\text{kernel}\_\text{size} = (\text{block}\_\text{size}, \text{block}\_\text{size})\), and \(\text{padding} = (\frac{\text{block}\_\text{size}}{2}, \frac{\text{block}\_\text{size}}{2})\). Red and blue points are 1 and 0 in the generated binary mask, respectively.

Figure A.8: The comparison of \(\rho\) for flattened hidden representations of a mini-batch of samples at the second residual block (layer \(k = 3\)) of WideResNet-28-10 with corresponding regularization method.

B Why random \(k\)?

PatchUp applies block-level regularization at a randomly selected hidden representation layer \(k\). The Information Bottleneck (IB) principle, introduced by Tishby and Zaslavsky \([25]\), gives a formal intuition for selecting \(k\) randomly. First, let us encapsulate the layers of the network into blocks where each block could contain more than one layer. Let \(g_k\) be the \(k\)-th block of layers. In this case, sequential blocks share the information as a hidden representation to the next block of layers, sequentially. We can consider this case as a Markov chain of the block of layers as follows:

\[
x \to g_1(x) \to g_2(x) \to g_3(x).
\]  

In this scenario, the sequential communication between the intermediate hidden representations are considered to be an information bottleneck. Therefore,

\[
I(g_k(x); g_2(x)) < I(g_2(x); g_1(x)) < I(g_1(x); x),
\]

where \(I(g_k(x); g_{k-1}(x))\) is the mutual information between the \(k\)-th and \((k - 1)\)-th layer. If \(g_3(x)\) has enough information to represent \(x\), then applying regularization techniques in \(g_3(x)\) will provide a better generalization to unseen data. However, most of the current state-of-the-art CNN models contain residual connections which break the Markov chain described above (since information can skip the \(g_3\) layer). One solution to this challenge is to randomly select a residual block and apply regularization techniques like ManifoldMixup or PatchUp.

C PatchUp Interpolation Policy Effect

Assume that \(H_1\) and \(H_2\) are flattened hidden representations of two examples produced at layer \(k\). And, \(H\) is the flattened interpolated hidden representation of these two paired samples at layer \(k\). First, we calculate the cosine distance of the pairs \((H_2, H_1), (H_1, H),\) and \((H_2, H)\). Reversing the cosine of these cosine similarities give the angular distance between each pair of vectors denoted as...
with the same labels.

As discussed in section 3, ManifoldMixup can provide interpolated hidden representation only in a limited space. This section describes the hyper-parameters of each model in Table 5 following the hyper-parameter setup from ManifoldMixup [18] experiments in order to create a fair comparison. First, we performed hyper-parameter tuning for the PatchUp to achieve the best validation performance. Then we ran all the experiments five times, reporting the mean and standard deviation of errors and negative log likelihoods for the selected models. We let models train for defined epochs and checkpoint the best model in terms of validation performance during the training. In our study, we used PreActResNet18, PreActResNet34, and WideResNet-28-10 models. Table 5 shows the hyper-parameters used for training the models.

PatchUp adds patchup_prob, γ and block_size as hyper-parameters. patchup_prob is the probability that the PatchUp operation is performed for a given mini-batch, i.e if there are N mini-batches and patchup_prob is p, PatchUp is performed in p fraction of N mini-batches. γ and block_size are described in section 2. We tuned the PatchUp hyper-parameter on CIFAR-10 with the PreActResNet18. To create a validation set, we split 10% of training samples into a validation set. We set α to 2 in PatchUp. For Soft PatchUp, we set patchup_prob to 1.0 and applied PatchUp to all mini-batches in training. Then, we did a grid search by varying γ from 0.45 to 0.9 and block_size...
(a) Impact of $\alpha$ in ManifoldMixup approach.
(b) Impact of cutmix_prob in CutMix approach.
(c) Impact of $\gamma$, block_size with patchup_prob as 1.0 for Soft PatchUp.
(d) Impact of $\gamma$, block_size with patchup_prob as 0.7 for Hard PatchUp.

Figure A.10: Impact of hyper-parameters $\gamma$, block_size and patchup_prob on error rates in the CIFAR-10 validation set for PreActResNet18. We repeated each job three times to collect the mean and the standard deviation of errors. Marked points are the mean of the error rate in the validation set. And, the shadow shows the bootstrapping of results for each hyper-parameter setting. The lower numbers on the y-axes correspond to better performance.

Table 5: The hyper-parameters used for each model to compare the effect of each regularization technique. The learning rate is denoted as $lr$. And, $lr$ is multiplied at each learning rate schedule step by the step factor.

| Model          | lr  | lr steps | step factor | Epochs |
|----------------|-----|----------|-------------|--------|
| PreactResnet18 | 0.1 | 500-1000-1500 | 0.1         | 2000   |
| PreactResnet34 | 0.1 | 500-1000-1500 | 0.1         | 2000   |
| WideResnet-28-10 | 0.1 | 200-300     | 0.1         | 400    |

from 3 to 9. We found that $\gamma$ of 0.75 and block_size of 7 work best for Soft PatchUp as shown in figure A.10c. Similarly, for Hard PatchUp, we set patchup_prob to 0.7 and performed a grid search by varying $\gamma$ from 0.2 to 0.6 and block_size from 3 to 9. We found that block_size of 7 and $\gamma$ of 0.5 yield the best results for Hard PatchUp as shown in figure A.10d. Figure A.10a shows that ManifoldMixup with ($\alpha = 1.5$) achieves the best validation performance. For cutout, we used the same hyper-parameters proposed in [19], setting cutout to 16 for CIFAR10, 8 for CIFAR100, and 20 for SVHN following [19]. Figure A.10b shows that CutMix achieves its best validation performance in PreActResNet18 in CIFAR-10 with cutmix_prob = 0.4. Furthermore, DropBlock achieves its best validation performance on this task by setting the block size and $\gamma$ to 7 and 0.9, respectively [13].
Table 6: Error rates in the test set on samples subject to affine transformations for WideResNet-28-10 trained on CIFAR-100 with indicated regularization method. We repeated each test for five trained models to report the mean and the standard deviation of errors. Best performance result is shown in bold, second best is underlined. The lower number is better.

| Transformation | cutout              | CutMix               | ManifoldMixup | Soft PatchUp | Hard PatchUp |
|---------------|---------------------|----------------------|---------------|--------------|--------------|
| Rotate (-20, 20) | 36.162 ± 0.633 | 34.236 ± 0.785 | 35.774 ± 0.621 | **31.282 ± 0.622** | 31.340 ± 0.318 |
| Rotate (-40, 40) | 57.220 ± 0.549 | 56.512 ± 0.752 | 56.610 ± 0.877 | **52.014 ± 0.916** | 52.804 ± 0.576 |
| Shear (-28.6, 28.6) | 33.482 ± 0.463 | 31.770 ± 0.312 | 32.300 ± 0.317 | **30.898 ± 0.836** | **28.426 ± 0.430** |
| Shear (-57.3, 57.3) | 53.328 ± 0.587 | 50.618 ± 0.552 | 52.366 ± 0.170 | 51.908 ± 0.632 | **48.334 ± 0.631** |
| Scale (0.6) | 56.770 ± 0.376 | 45.980 ± 0.404 | 63.924 ± 2.160 | 52.648 ± 0.616 | 46.924 ± 1.035 |
| Scale (0.8) | 30.550 ± 0.611 | 26.818 ± 0.328 | 29.012 ± 0.372 | 27.188 ± 0.507 | **23.840 ± 0.535** |
| Scale (1.2) | 47.268 ± 0.639 | 51.258 ± 0.817 | **41.644 ± 0.846** | 42.108 ± 0.985 | 43.370 ± 1.223 |
| Scale (1.4) | 79.000 ± 0.933 | 82.562 ± 0.575 | 72.752 ± 0.846 | **70.970 ± 1.433** | 77.370 ± 1.457 |

E  Generalization on Deformed Images

We created the deformed test sets from CIFAR100, as described in Section 4.2. Table 6 shows improved quality of representations learned by a WideResNet-28-10 model regularized by PatchUp on CIFAR-100 deformed test sets. The significant improvements in generalization provided by PatchUp in this experiment shows the high quality of representations learned with PatchUp.

F  Robustness to Adversarial Examples

The adversarial attacks refer to small and unrecognizable perturbations on the input images that can mislead deep learning models [24][8]. One approach to creating adversarial examples is using the Fast Gradient Sign Method (FGSM), also known as a white-box attack [24]. FGSM creates examples by adding small perturbations to the original examples. Once a regularized model is trained then FGSM creates adversarial example as follows [24]:

\[ x' = x + \epsilon \times \text{sign}(\nabla_x J(\theta, x, y)) \]  

(15)

where \( x' \) is an adversarial example, \( x \) is the original example, \( y \) is the ground truth label for \( x \), and \( J(\theta, x, y) \) is the loss of the model with parameters of \( \theta \). \( \epsilon \) controls the perturbation.

Our experiments show the effectiveness of Soft PatchUp against the attacks in most cases. However, Hard PatchUp performed well against the FGSM attack only on PreActResNet34 for CIFAR-100. Figure A.11 shows the comparison of the state-of-the-art regularization techniques’ effect on model robustness against the FGSM attack.

G  Analysis of PatchUp’s Effect on Activations

In our implementation WideResNet28-10 has a conv2d module followed by three residual blocks. Figure A.12 illustrates the comparison of ManifoldMixup, cutout, CutMix, Soft PatchUp, and Hard PatchUp. Figure A.12A, A.12B, and A.12C show that PatchUp produces more variety of features in layers that we apply PatchUp on.
Figure A.11: Robustness to the FGSM attack, known as a white-box attack. We repeated each test for five trained models to report the mean and the standard deviation of the accuracy of each method against the FGSM attack. The higher values on the y-axes show the robustness of the model against the attack. And, $\epsilon$ is the magnitude that controls the perturbation.
Figure A.12: The effect of the state-of-the-art regularization techniques on activations in WideResNet28-10 for CIFAR100 test set. Each curve is the magnitude of feature activations, sorted by descending value, and averaged over all test samples for each method. The higher magnitude shows a wider variety of the produced features by the model at each block.

(a) Comparison on the first convolution module.
(b) Comparison on 1st Residual Block.
(c) Comparison on 2nd Residual Block.
(d) Comparison on 3rd Residual Block.