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 (Gai and Ghahramani 2016) and ZoneOut (Krueger et al. 2016), Information Dropout (Achille and Soatto 2018), SpatialDropout (Tompson et al. 2014a, 2014c), 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. 2014a) 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...
MetaMixUp (Mai et al. 2019) used the same inputs but different targets. To overcome the manifold intrusion problem, MetaMixUp uses 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 computational 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. appx.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, $\text{block\_size}$. Hence, the altering probabilities are readjusted using the following formula (Ghiassi, Lin, and Le [2018]):

$$\gamma_{adj} = \frac{\gamma \times (\text{feature\_map\_area})}{(\text{block\_size}) \times (\text{valid\_region\_to\_build\_block})} \quad (2)$$

where the area of the feature map and block are the $\text{feature\_size}^2$ and $\text{block\_size}^2$, respectively, and the valid region to build the block is $\frac{\text{feature\_size} - \text{block\_size} + 1}{2}$. For each feature in the feature map, we sample from $\text{Bernoulli}(\gamma_{adj})$. 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 $\text{block\_size}$ 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. 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. 10 and 9 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:

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

where $\odot$ is known as the element-wise multiplication operation and $M$ 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:

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

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]). 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:

$$L(f) = \mathbb{E}_{(x_i, y_i) \sim P} \mathbb{E}_{(x_j, y_j) \sim P} \mathbb{E}_{\lambda \sim \text{Beta}(\alpha, \alpha)} \mathbb{E}_{S \sim S} \left[ \text{Mix}_\mu[p_u \ell(f_k(\phi_k), y), \ell(f_k(\phi_k), Y)] + \ell(f_k(\phi_k), W(y, y_j)) \right] \quad (6)$$

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

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

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

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

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 3).

**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 2).

### 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 1. 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 $\lambda$ sampled from the beta distribution.
This section presents the results of applying PatchUp to image classification tasks using various benchmark datasets such as CIFAR10, CIFAR100 (Krizhevsky, 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). We used the same set of base hyper-parameters for all the models for a thorough and fair comparison. The details of the 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 hyper-parameters. \( \text{patchup\_prob} \) 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 which helps to outperform other methods by 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).

| Method                  | ResNet101 | ResNet152 | WideResnet-28-10 |
|-------------------------|-----------|-----------|------------------|
|                         | Error     | Loss      | Error            | Loss     |
| No Mixup                | 46.18±0.49| 2.37±0.03 | 45.22±0.32       | 2.25±0.08|
| Input Mixup (α = 1)     | 44.93±0.34| 2.25±0.05 | 44.65±0.11       | 2.23±0.04|
| ManifoldMixup (α = 2)   | 44.51±0.14| 2.25±0.01 | 44.13±0.31       | 2.23±0.02|
| Cutout                  | 45.78±0.35| 2.35±0.01 | 45.23±0.19       | 2.34±0.02|
| DropBlock               | 46.98±0.53| 2.27±0.10 | 45.82±0.22       | 2.17±0.01|
| CutMix                  | 42.94±0.38| 2.11±0.01 | 41.71±0.71       | 2.08±0.03|
| Puzzle Mix              | 42.25±0.19| 2.13±0.01 | 42.13±0.31       | 2.13±0.03|
| Soft PatchUp            | 38.68±0.34| 1.87±0.01 | 38.11±0.29       | 1.84±0.02|
| Hard PatchUp            | 40.55±0.15| 1.94±0.01 | 40.31±0.18       | 1.93±0.01|

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.

| PreActResNet18          | CIFAR-10  | CIFAR-100 | SVHN   |
|-------------------------|-----------|-----------|--------|
|                         | Test Error| Test Error| Test Error|
| No Mixup                | 4.80±0.14 | 24.62±0.36| 3.04±0.09|
| Input Mixup (α = 1)     | 3.63±0.20 | 22.00±0.32| 2.93±0.22|
| ManifoldMixup (α = 1.5) | 3.39±0.05 | 21.40±0.38| 2.44±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            | 2.95±0.12 | 19.95±0.18| 2.55±0.06|
| Hard PatchUp            | 2.92±0.13 | 19.12±0.17| 2.29±0.08|

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) **.

| 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            | 21.06           | 5.57            |
| Hard PatchUp            | 20.75           | 5.29            |

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 $E_s^c$ where $s$ is the level of severity and $c$ is corruption type such that $E_s^c = \sum_{s=1}^{5} E_s^c$ (Hendrycks and Dietterich 2019). Fig4 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 a deformed test from CIFAR100
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-approx. 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 PGDL∞ (Madry et al. 2019) attacks. 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.

| 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 | 31.14 ± 0.52 | **30.41 ± 0.52** |
| Rotate (-40, 40) | 58.75 ± 0.10 | 57.83 ± 0.59 | 54.42 ± 0.95 | 54.04 ± 0.80 | 54.49 ± 0.39 | **49.96 ± 0.80** |
| Shear (-28.6, 28.6) | 36.55 ± 0.49 | 34.15 ± 0.47 | 34.15 ± 0.42 | 31.63 ± 0.69 | **28.98 ± 0.50** | 29.57 ± 0.41 |
| Shear (-57.3, 57.3) | 57.74 ± 0.57 | 53.64 ± 0.59 | 55.44 ± 0.68 | 52.49 ± 0.39 | **49.10 ± 0.53** | 50.32 ± 0.62 |
| Scale (0.6) | 72.99 ± 1.23 | 54.30 ± 1.27 | 78.99 ± 1.13 | 59.70 ± 1.24 | **46.25 ± 1.20** | 50.06 ± 2.69 |
| Scale (0.8) | 35.09 ± 0.86 | 29.38 ± 0.58 | 34.62 ± 0.37 | 30.37 ± 0.36 | **23.94 ± 0.21** | 25.34 ± 0.33 |
| Scale (1.2) | 42.31 ± 0.71 | 49.52 ± 2.04 | 41.32 ± 0.64 | 47.40 ± 1.44 | 43.41 ± 0.65 | **38.01 ± 0.70** |
| Scale (1.4) | 69.40 ± 0.90 | 78.66 ± 1.85 | **65.94 ± 0.75** | 77.59 ± 0.97 | 77.07 ± 1.19 | 66.34 ± 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.

**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-approx. 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 PGDL∞ (Madry et al. 2019) attacks. 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 | PGDL∞ |
|------------------|----------|-----|----|--------|
| No Mixup         | 0.17 ± 0.01 | 0.18 ± 0.01 | 0.18 ± 0.01 | 0.17 ± 0.01 |
| Input Mixup      | 0.19 ± 0.01 | 0.20 ± 0.01 | 0.20 ± 0.01 | 0.19 ± 0.01 |
| ManifoldMixup    | 0.20 ± 0.01 | 0.20 ± 0.01 | 0.20 ± 0.01 | 0.19 ± 0.01 |
| Cutout           | 0.18 ± 0.01 | 0.185 ± 0.01 | 0.19 ± 0.01 | 0.18 ± 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 | 0.20 ± 0.01 | 0.20 ± 0.01 | 0.19 ± 0.01 |
| Hard PatchUp     | 0.18 ± 0.01 | 0.19 ± 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 $\alpha$ is 0.2 for Mixup, ManifoldMixup, and Puzzle Mix. The 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

![Diagram](image-url)
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, and 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 the 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 to better performance. Fig. 6 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 follows:

\[
L(f) = \mathbb{E}_{(x, y) \sim P} \mathbb{E}_{(x, y') \sim P} \mathbb{E}_{A \sim \text{Beta}(\alpha, \alpha)} \mathbb{E}_{k \sim \mathcal{S}} (L_1 + L_2)
\]

(9)

where \( L_1 = \text{Mix}_{p_k} [\ell(f_k(\phi_k), y_i), \ell(f_k(\phi_k), Y)] \) 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 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 the PatchUp mask in the input space.

![Figure 6: 1st (left) and 2st (right) Residual Block.](image)

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.

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

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

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

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.
6 Acknowledgments

We would like to acknowledge Compute Canada and Calcul Quebec for providing the 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 Down-sampled 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 Salience and Local Statistics for Optimal Mixup. arXiv:2009.06962.

Kingma, D. P.; Salimans, T.; and Welling, M. 2015. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28: 2575–2583.

Kingma, D. P.; and Welling, M. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Krizhevsky, A. 2009. Learning Multiple Layers of Features from Tiny Images.

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 rns 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
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, NIPS’14, 3104–3112. 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.

Tishby, N.; and Zaslavsky, N. 2015. Deep Learning and the Information Bottleneck Principle. CoRR, abs/1503.02406.

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.

Vernna, 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.
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), $\gamma$, 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.
- $\gamma$: 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 $\text{mode} == \text{Inference}$ then
2: return $(g_k(x_i), y_i), (g_k(x_j), y_j)$
3: end if
4: kernel_size $\leftarrow$ (block_size, block_size)
5: stride $\leftarrow$ (1, 1)
6: padding $\leftarrow$ (block_size - stride) / 2
7: $\gamma_{adj}$ $\leftarrow$ adjust $\gamma$ using (2)
8: holes $\leftarrow$ max_pool2d(Bernoulli($\gamma_{adj}$), kernel_size, stride, padding)
9: Mask $\leftarrow$ 1 - holes
10: unchanged $\leftarrow$ Mask $\circ$ $g_k(x_i)$
11: $p_u$ $\leftarrow$ calculate the portion of changed features map.
12: $\text{Patch}_i$ $\leftarrow$ holes $\circ$ $g_k(x_i)$
13: $\text{Patch}_j$ $\leftarrow$ holes $\circ$ $g_k(x_j)$
14: if mixing_type $==$ hard then
15: $\text{Patch}_i$ $\leftarrow$ $\text{Patch}_j$
16: $Y$ $\leftarrow$ $y_j$
17: $W$ $\leftarrow$ $W_{\text{hard}}(y_i, y_j)$ using (8)
18: else if mixing_type $==$ soft then
19: $\lambda$ $\sim$ Beta($\alpha$, $\alpha$)
20: $Y$ $\leftarrow$ Mix$_\lambda(y_i, y_j)$
21: $W$ $\leftarrow$ $W_{\text{soft}}(y_i, y_j)$ using (8)
22: $\text{Patch}_i$ $\leftarrow$ Mix$_\lambda(\text{Patch}_i, \text{Patch}_j)$
23: end if
24: $H'$ $\leftarrow$ unchanged + $\text{Patch}_i$
25: return $y_i, y_j, H', p_u, Y, W$

Figure 10 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 9 compares the masks generated by PatchUp and CutMix.

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] (2015), 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:

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

(10)

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

\[ I(g_3(x); g_2(x)) < I(g_2(x); g_1(x)) < I(g_1(x); x), \]  

(11)

Figure 9: 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.

Let us define \( \angle \rho = (\angle \alpha_1 + \angle \alpha_2) - \angle \theta \) and \( \angle \rho' = |\angle \alpha_1 - \angle \alpha_2| - \angle \theta \). According to the triangle inequality principle, either \( \angle \rho \) or \( \angle \rho' \) will be zero if, and only if, \( \mathcal{H} \in \mathcal{S} \). Figure 12 illustrates three possible scenarios for two paired 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.

C PatchUp Interpolation Policy Effect

Assume that \( \mathcal{H}_1 \) and \( \mathcal{H}_2 \) are flattened hidden representations of two examples produced at layer \( k \). And, \( \mathcal{H} \) is the flattened interpolated hidden representation of these two paired samples at layer \( k \). First, we calculate the cosine distance of the pairs (\( \mathcal{H}_2, \mathcal{H}_1 \)), (\( \mathcal{H}_1, \mathcal{H} \)), and (\( \mathcal{H}_2, \mathcal{H} \)). Reversing the cosine of these cosine similarities give the angular distance between each pair of vectors denoted as \( \angle \alpha_1 \), \( \angle \alpha_2 \), and \( \angle \theta \), respectively. There is always a surface that contains \( \mathcal{H}_2 \) and \( \mathcal{H}_1 \) denoted as \( \mathcal{S} \). Mathematically, we have:

\[ \angle \alpha_1 = \cos^{-1}\left(\frac{\mathcal{H}_1 \cdot \mathcal{H}}{\|\mathcal{H}_1\| \|\mathcal{H}\|}\right) \quad & \quad \angle \alpha_2 = \cos^{-1}\left(\frac{\mathcal{H}_2 \cdot \mathcal{H}}{\|\mathcal{H}_2\| \|\mathcal{H}\|}\right) \quad & \quad \angle \theta = \cos^{-1}\left(\frac{\mathcal{H}_2 \cdot \mathcal{H}_1}{\|\mathcal{H}_2\| \|\mathcal{H}_1\|}\right), \]  

(12)
hidden representations and their flattened interpolated hidden representations. \( \angle \rho \) and \( \angle \rho' \) are zero for the left and right figures, respectively. We try to empirically show that \( H_2, H_1, \) and \( H \) always lie in the same surface \( S \) and \( H \) lies between \( H_2 \) and \( H_1 \) in ManifoldMixup. This means that \( \angle \rho = 0 \) for ManifoldMixup because of its linear interpolation policy. The middle figure in [12] is the case that both \( \angle \rho \) and \( \angle \rho' \) are not equal to zero. This figure shows that one possible situation is that flattened interpolated hidden representation does not lie in the surface \( S \). Our goal is to produce the interpolated hidden representation that lies in all possible places towards all dimensions in the hidden space.

![Image](54x546 to 215x657)

Figure 12: \( H_1 \) and \( H_2 \) are the flattened hidden representations. \( H \) is the flattened interpolated hidden representation that can be produced by either ManifoldMixup, Soft PatchUp or Hard PatchUp.

As discussed in section [3], ManifoldMixup can provide interpolated hidden representation only in a limited space. However, Soft PatchUp and Hard PatchUp can produce a wide variety of interpolated hidden representations towards different dimensions. To support that, in WideResNet-28-10, for a mini-batch of 100 samples, we calculated the \( \angle \rho \) for the flattened interpolated hidden representation produced by ManifoldMixup, Hard PatchUp, and Soft PatchUp at the second residual block (layer \( k = 3 \)) with the same interpolation policy (\( \lambda = 4, \gamma = .5, \) and block.size = 7) for both Soft PatchUp and Hard PatchUp for all samples in the mini-batch. The swarmplot [11] shows all \( \angle \rho \) for the mini-batch are equal to zero in ManifoldMixup, which empirically supports our hypothesis. However, PatchUp produces more diverse interpolated hidden representations towards all dimensions in the hidden space. It is worth mentioning that few \( \angle \rho \) that are equal to zero in Soft PatchUp and Hard PatchUp belong to the interpolated hidden representation that was constructed from the pairs with the same labels.

**D PatchUp Experiment Setup and Hyper-parameter Tuning**

We follow the ManifoldMixup in our experiment setup, we use SGD with 0.9 Nesterov momentum as an optimizer, mini-batch of 100, and weight decay of 1e-4 [Verma et al. 2019].

This section describes the hyper-parameters of each model in table [9] following the hyper-parameter setup from Manifold-Mixup [Verma et al. 2019] 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 [9] shows the hyper-parameters used for training the models.

Table 8: Training time comparison between Puzzle Mix and PatchUp using one GPU (V100) with 4 CPU cores and 32G of memory in our experiment setups. The reported time is rounded up to one hour.

|                   | CIFAR (hours) | SVHN (hours) | Tiny-ImageNet (hours) |
|-------------------|---------------|--------------|-----------------------|
|                   | PatchUp       | Puzzle Mix   | PatchUp               | Puzzle Mix   | PatchUp | Puzzle Mix |
| PreActResNet18    | 18            | 53           | 18                    | 53           | -       | -          |
| PreActResNet34    | 24            | 64           | 24                    | 60           | -       | -          |
| WideResNet-28-10  | 12            | 26           | 12                    | 32           | 23      | 52         |

PatchUp adds \texttt{patchup.prob}, \( \gamma \) and \texttt{block.size} as hyper-parameters. \texttt{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 \texttt{patchup.prob} is \( p \), PatchUp is performed in \( p \) fraction of \( N \) mini-batches. \( \gamma \) and \texttt{block.size} are described in section [2]. We tuned the PatchUp hyper-parameter on CIFAR10 with the PreActResNet18. To create a validation set, we split 10% of training samples into a validation set. We set \( \alpha \) to 2 in PatchUp. For Soft PatchUp, we set \texttt{patchup.prob} to 1.0 and applied PatchUp to all mini-batches in training. Then, we did a grid search by varying \( \gamma \) from 0.45 to 0.9 and \texttt{block.size} from 3 to 9. We found that \( \gamma \) of 0.75 and \texttt{block.size} of 7 work best
(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 13: Impact of hyper-parameters $\gamma$, block_size and patchup_prob on error rates in the CIFAR10 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.
According to the Eq. 4, łunchanged part of the feature maps (when considering the tight competition between regularization methods. For Soft PatchUp as shown in figure 13a, Similarly, for Hard PatchUp, we set patchup_prob to 0.7 and performed a grid search by varying γ from 0.2 to 0.6 and block_size from 3 to 9. We found that block_size of 7 and γ of 0.5 yield the best results for Hard PatchUp as shown in figure 13d. Figure 13a shows that ManifoldMixup with (α = 1.5) achieves the best validation performance. For cutout, we used the same hyper-parameters proposed in (DeVries and Taylor 2017), setting cutout to 16 for CIFAR10, 8 for CIFAR100, and 20 for SVHN following (DeVries and Taylor 2017). Figure 13a shows that CutMix achieves its best validation performance in PreActResNet18 in CIFAR10 with cutmix_prob = 0.4. Furthermore, DropBlock achieves its best validation performance on this task by setting the block size and γ to 7 and 0.9, respectively (Ghiasi, Lin, and Le 2018).

As per Table 7, applying the CutMix Binary Mask in input space but still using the PatchUp gamma as a hyperparameter will prevent such large semantic changes in the latent space. However, gamma cannot avoid such negative effects in input space. As per Table 7, applying the CutMix Binary Mask in input space but still using the PatchUp learning objective leads to a performance improvement of 0.836 when training WRN-28-10 on CIFAR-100, which is significant when considering the tight competition between regularization methods.

### E Further explanation on PatchUp and its learning objective:

According to the Eq. 8, $Mix_{p_u}$ is computed as follows:

$$Mix_{p_u}[a, b] = p_u a + (1 - p_u) b$$

In the first term in equation 8 we take a convex combination of the following two loss functions: one with respect to the unchanged part of the feature maps ($\ell(f_k(\phi_k), y_i)$) and the other with respect to the interpolated part ($\ell(f_k(\phi_k), \lambda Y)$). Please note that the loss function for the interpolated part will change based on whether we do Soft or Hard PatchUp. Therefore, we can expand the PatchUp learning objective as follows:

$$L(f) = E_{(x_i, y_i) \sim P} E_{(x_j, y_j) \sim P} E_{\lambda \sim \text{Beta}(\alpha, \alpha)} E_{k \sim S} Mix_{p_u} [\ell(f_k(\phi_k), y_i), \ell(f_k(\phi_k), \lambda Y)] + \ell(f_k(\phi_k), \lambda W(y_i, y_j))$$

Based on the operation defined in section 2, we can substitute and expand the learning objective for Hard PatchUp as follows:

(For Soft PatchUp the process will be similar.) In the Hard PatchUp:

$$Y = y_j$$

And according to Equation 8

$$W_{hard}(y_i, y_j) = Mix_{p_u}(y_i, y_j) = p_u y_i + (1 - p_u) y_j$$

$$Mix_{p_u}[\ell(f_k(\phi_k), y_i), \ell(f_k(\phi_k), Y)] = p_u \ell(f_k(\phi_k), y_i) + (1 - p_u) \ell(f_k(\phi_k), y_j)$$

Therefore, we can conclude the learning objective for Hard PatchUp is defined as follows:

$$L(f) = p_u \ell(f_k(\phi_k), y_i) + (1 - p_u) \ell(f_k(\phi_k), y_j) + \ell(f_k(\phi_k), p_u y_i + (1 - p_u) y_j)$$

where the $\phi_{hard}(g_k(x_i), g_k(x_j)) = M \odot g_k(x_i) + (1 - M) \odot g_k(x_j)$. Hard PatchUp is a fully differentiable operation. The masks for the hard PatchUp are sampled outside the computation graph and passed as input to the architecture. Hence the entire architecture is differentiable end-to-end. This is similar to the reparameterization trick that was proposed in (Kingma and Welling 2013). A similar trick was also applied in Variational Dropout (Kingma, Salimans, and Welling 2015) and other

| 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    |

Table 9: 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 step factor.
similar methods as well. Similarly, we can substitute the operations \((Mix\) and \(\phi_{soft}\) defined in Equation 3 and 5) to expand the learning objective of Soft PatchUp.

PatchUp works by applying a structured noisy process to the representations at a random point in the forward pass (random layer). The model then learns to recognize the structure in the noisy representations created in the forward pass by back-propagating through the model with the gradients from the PatchUp learning objective. Since all operations are differentiable and involve either swapping or mixing latent spaces (similar to the reparameterization trick), we will not have any non-differentiability issues when calculating the gradients. Hence, deriving the backprop manually is not necessary.

\section*{F Generalization on Deformed Images}

Table 10: 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. The best performance result is shown in bold, second best is underlined. The lower number is better.

| Transformation | cutout | CutMix | ManifoldMixup | Puzzle Mix | Soft PatchUp | Hard PatchUp |
|----------------|--------|--------|---------------|------------|--------------|--------------|
| Rotate (-20, 20) | 36.162 ± 0.633 | 34.236 ± 0.785 | 35.774 ± 0.621 | 28.35 ± 0.561 | 31.282 ± 0.622 | 31.340 ± 0.318 |
| Rotate (-40, 40) | 57.220 ± 0.549 | 56.512 ± 0.752 | 56.610 ± 0.877 | 51.49 ± 1.028 | 52.014 ± 0.916 | 53.804 ± 0.576 |
| Shear (-28.6, 28.6) | 33.482 ± 0.463 | 31.770 ± 0.312 | 32.300 ± 0.317 | 27.178 ± 0.256 | 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 | 47.414 ± 0.482 | 51.908 ± 0.632 | 48.334 ± 0.631 |
| Scale (0.6) | 56.770 ± 0.376 | 45.980 ± 0.404 | 63.924 ± 2.160 | 46.354 ± 0.869 | 52.648 ± 0.616 | 46.924 ± 1.035 |
| Scale (0.8) | 30.550 ± 0.611 | 26.818 ± 0.328 | 29.012 ± 0.372 | 24.428 ± 0.171 | 27.188 ± 0.597 | 23.840 ± 0.535 |
| Scale (1.2) | 47.268 ± 0.639 | 51.258 ± 0.817 | 41.644 ± 0.846 | 49.18 ± 1.046 | 42.108 ± 0.985 | 43.370 ± 1.223 |
| Scale (1.4) | 79.000 ± 0.933 | 82.562 ± 0.575 | 72.752 ± 0.846 | 82.152 ± 0.41 | 70.970 ± 1.433 | 77.370 ± 1.457 |

We created the deformed test sets from CIFAR100, as described in Section 4. Table 10 shows improved quality of representations learned with PatchUp compared to other methods. Similarly, we can substitute the operations \((Mix\) and \(\phi_{soft}\) defined in Equation 3 and 5) to expand the learning objective of Soft PatchUp.

\section*{G Robustness to Adversarial Examples}

The adversarial attacks refer to small and unrecognizable perturbations on the input images that can mislead deep learning models (Goodfellow, Shlens, and Szegedy 2014; Goodfellow, Bengio, and Courville 2016). One approach to creating adversarial examples is using the Fast Gradient Sign Method (FGSM), also known as a white-box attack (Goodfellow, Shlens, and Szegedy 2014). FGSM creates examples by adding small perturbations to the original examples. Once a regularized model is trained then FGSM creates adversarial examples as follows (Goodfellow, Shlens, and Szegedy 2014):

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

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 CIFAR100. Figure 14 shows the comparison of the state-of-the-art regularization techniques’ effect on model robustness against the FGSM attack.

\section*{H Analysis of PatchUp’s Effect on Activations}

In our implementation WideResNet28-10 has a conv2d module followed by three residual blocks. Figure 15 illustrates the comparison of ManifoldMixup, cutout, CutMix, Soft PatchUp, and Hard PatchUp. Figure 15a, 15b, and 15c show that PatchUp produces more variety of features in layers that we apply PatchUp on.

\section*{I More Details on Image Classification results}

This section contains the image classification error and Test NLL on CIFAR10, CIFAR100, and SVHN datasets. Tables 12 and 11 shows Test error rates and test NLL on CIFAR10, CIFAR100, and SVHN datasets using PreActResNet18, PreActResNet34, and WideResNet-28-10 models.

When we have a data-scarce scenario, high-capacity models tend to memorize the examples. This leads to models with reasonable testing performance but that are very sensitive to small changes in test distribution as shown in Tiny-ImageNet-C experiments. Tiny-ImageNet has 200 classes while CIFAR-100 has 100 classes, and both datasets have 500 training examples per class. Our experiment shows that on the CIFAR-100 (the smaller dataset), all the data-dependent regularization methods tend to perform well. When we switch to Tiny-ImageNet (the larger dataset), the (non-PatchUp) data-dependent regularization methods are still effective, but not necessarily close to the best method (PatchUp) as shown in Table 12 and 11.
Table 11: Error rates comparison on SVHN. We run experiments five times to report the mean and the standard deviation. The best performance result is shown in bold, second best is underlined. The lower number is better.

|                   | PreActResNet18 | PreActResNet34 | WideResnet-28-10 |
|-------------------|----------------|----------------|------------------|
|                   | Error          | Loss           | Error            | Loss           | Error            | Loss           |
| No Mixup          | 3.035 ± 0.092  | 0.138 ± 0.004  | 3.087 ± 0.009    | 0.164 ± 0.008  | 2.833 ± 0.081   | 0.137 ± 0.008  |
| Input Mixup (α = 1) | 2.930 ± 0.221  | 0.233 ± 0.016  | 2.855 ± 0.096    | 0.223 ± 0.024  | 2.643 ± 0.161   | 0.207 ± 0.041  |
| ManifoldMixup (α = 1.5) | 2.436 ± 0.056  | 0.157 ± 0.062  | 2.423 ± 0.428    | 0.146 ± 0.064  | 2.425 ± 0.101   | 0.157 ± 0.029  |
| Cutout            | 2.794 ± 0.121  | 0.122 ± 0.010  | 2.654 ± 0.152    | 0.114 ± 0.007  | 2.475 ± 0.148   | 0.109 ± 0.009  |
| DropBlock         | 2.961 ± 0.111  | 0.134 ± 0.005  | 3.101 ± 0.083    | 0.158 ± 0.006  | 2.732 ± 0.055   | 0.132 ± 0.002  |
| CutMix            | 3.040 ± 0.054  | 0.135 ± 0.031  | 2.658 ± 0.049    | 0.121 ± 0.005  | 2.433 ± 0.045   | 0.110 ± 0.003  |
| Puzzle Mix        | 2.657 ± 0.042  | 0.113 ± 0.003  | 2.451 ± 0.075    | 0.111 ± 0.002  | 2.432 ± 0.068   | 0.107 ± 0.002  |
| Soft PatchUp      | 2.551 ± 0.056  | 0.129 ± 0.023  | 2.467 ± 0.081    | 0.111 ± 0.005  | **2.081 ± 0.066** | 0.111 ± 0.010 |
| Hard PatchUp      | **2.286 ± 0.084** | 0.107 ± 0.007  | **2.123 ± 0.024** | 0.101 ± 0.007  | 2.088 ± 0.061   | 0.105 ± 0.012  |

Table 12: Image classification task error rates on CIFAR-10 and CIFAR-100. We run experiments five times to report the mean and the standard deviation. The best performance result is shown in bold, second best is underlined. The lower number is better.
Figure 14: 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 15: 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.
Figure 16: Robustness to Tiny-ImageNet Common Corruptions (Tiny-ImageNet-C). We repeated each test for five trained models to report the mean and the standard. The lower values on the y-axes show the robustness of the model against the input common corruptions. The y-axis is the sum of error rates for each category. And, the x-axis represents the list of corruptions in Tiny-ImageNet-C.