Université de Montréal

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

par

Mojtaba Faramarzi

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

July 23, 2021

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

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

présenté par

Mojtaba Faramarzi

a été évalué par un jury composé des personnes suivantes :

Aaron Courville
(président-rapporteur)

Sarath Chandar Anbil Parthipan
(directeur de recherche)

Claude Frasson
(membre du jury)

Résumé

Les modèles d'apprentissage profond à large capacité ont souvent tendance à présenter de hauts écarts de généralisation lorsqu'ils sont entrainés avec une quantité limitée de données étiquetées. Dans ce cas, des réseaux de neurones très profonds et larges auront tendance à mémoriser les échantillons de données et donc ils risquent d'être vulnérables lors d'un léger décalage dans la distribution des données au moment de tester. Ce problème produit une généralisation pauvre lors de changements dans la répartition des données au moment du test. Pour surmonter ce problème, certaines méthodes basées sur la dépendance et l'indépendance de données ont été proposées. Une récente classe de méthodes efficaces pour aborder ce problème utilise plusieurs manières de contruire un nouvel échantillon d'entrainement, en mixant une paire (ou plusieurs) échantillons d'entrainement.

Dans cette thèse, nous introduisons *PatchUp*, une régularisation de l'espace des caractéristiques au niveau des blocs dépendant des données qui opère dans l'espace caché en masquant des blocs contigus parmi les caractéristiques mappées, sélectionnés parmi une paire aléatoire d'échantillons, puis en mixant (*Soft PatchUp*) ou en échangeant (*Hard PatchUp*) les blocs contigus sélectionnés. Notre méthode de régularisation n'ajoute pas de surcharge de calcul significative au CNN pendant l'entrainement du modèle. Notre approche améliore la robustesse des modèles CNN face au problème d'intrusion du collecteur qui pourrait apparaitre dans d'autres approches de mixage telles que Mixup et CutMix. De plus, vu que nous mixons des blocs contigus de caractéristiques dans l'espace caché, qui a plus de dimensions que l'espace d'entrée, nous obtenons des échantillons plus diversifiés pour entrainer vers différentes dimensions. Nos expériences sur les ensembles de données CIFAR-10, CIFAR-100, SVHN et Tiny-ImageNet avec des architectures ResNet telles que PreActResnet18, PreActResnet34, WideResnet-28-10, ResNet101 et ResNet152 montrent que PatchUp dépasse ou égalise les performances de méthodes de régularisation pour CNN considérée comme état de l'art actuel. Nous montrons aussi que PatchUp peut fournir une meilleure généralisation pour des transformations affines d'échantillons et est plus robuste face à des attaques d'exemples contradictoires. PatchUp aide aussi les modèles CNN à produire une plus grande variété de caractéristiques dans les blocs résiduels en comparaison avec les méthodes de pointe de régularisation pour CNN telles que Mixup, Cutout, CutMix, ManifoldMixup et Puzzle Mix.

Mots clés: Apprentissage en profondeur, Réseau Neuronal Convolutif, Généralisation, Régularisation, Techniques de régularisation dépendantes et indépendantes des données, Robustesse aux attaques adverses.

Abstract

Large capacity deep learning models are often prone to a high generalization gap when trained with a limited amount of labeled training data. And, in this case, very deep and wide networks have a tendency to memorize the samples, and therefore they might be vulnerable under a slight distribution shift at testing time. This problem yields poor generalization for data outside of the training data distribution. To overcome this issue some data-dependent and data-independent methods have been proposed. A recent class of successful methods to address this problem uses various ways to construct a new training sample by mixing a pair (or more) of training samples.

In this thesis, we introduce *PatchUp*, 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. 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, SVHN, and Tiny-ImageNet datasets using ResNet architectures including PreActResnet18, PreActResnet34, WideResnet-28-10, ResNet101, and ResNet152 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 affine transformations of samples and is more robust against adversarial attacks. PatchUp 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.

Key words: Deep Learning, Convolutional Neural Network, Generalization, Regularization, Data-dependent and Data-independent Regularization Techniques, Robustness to Adversarial Attacks.

Contents

List of tables

List of figures

List of Abbreviations

DL	Deep Learning
CNN	Convolutional Neural Network
CIFAR	Canadian Institute for Advanced Research
SVHN	The Street View House Numbers
FGSM	Fast Gradient Sign Method
PGD	Projected Gradient Descent
CW	Carlini-Wagner attack
DDN	Decoupled Direction and Norm attack

Remerciements

To you.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor, Dr. Sarath Chandar, for providing guidance and feedback throughout my study. I am grateful to him for accepting me as his student and providing a great chance to learn the technical concepts and personal skills.

I would like to thank my primary collaborators: Mohammad Amini, Akilesh Badrinaaraayanan, and Vikas Verma, and Sarath Chandar. Chapters 3 and 4 in this thesis are joint research work with them. I would also like to thank you, Doriane Olewicki, Darshan Patil, Louis Clouâtre, and Paul-Aymeric McRae for reviewing the thesis.

I am grateful to be part of Mila, the Quebec AI Institute. I would like to thank the following Mila faculty members for their technical advice and several exciting conversations over some projects and their guidance in the last year: Irina Rish, Pouya Bashivan, and Samira Ebrahimi Kahou.

I want to thank all my teammates in ChandarLab who participated in weekly meetings and for all their effort to make the ChandarLab an excellent lab for research and a fun place to learn new things.

I want to acknowledge Compute Canada and Calcul Quebec for providing computing resources used in this work.

This thesis would not have been possible without the support of my friends and family!

Chapter 1

Introduction

Deep Learning (DL), particularly deep Convolutional Neural Networks (CNNs) have achieved exceptional performance in many machine learning tasks, including object recognition [29], image classification [17, 29, 46], speech recognition [23], and natural language understanding [59, 64]. However, in a 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 [2, 14]. Poor generalization or *overfitting* problem might happen for a high-capacity model that performs very well on training data but not in unseen data at evaluation time [7]. There are several approaches to tackle this problem.

This thesis aims to introduce a new data-dependent regularization that leads the CNN model to prevent overfitting and reduce the generalization gap. Therefore, it helps CNN models generalize better on unseen data and against adversarial attacks.

1.1. Machine Learning

According to Arthur Samuel, Machine Learning (ML) is the field of study that gives computers the ability to learn without being explicitly programmed. Another definition for machine learning was proposed by Tom Mitchell [39] that indicates that machine learning is the study of computer algorithms that improve automatically through experience and by the use of data.

There are different types of learning algorithms including supervised learning, unsupervised learning, semi-supervised learning, self-supervised learning, and reinforcement learning. This thesis has a focus on supervised learning algorithms, particularly deep learning algorithms and convolutional neural networks. In supervised learning, the model is supervised using samples and the targets associated with each sample. In supervised learning, the datasets that are given to the learner have the correct target clearly defined, and therefore the goal of the model is to find a relationship between the input and the learning target.

Regression and classification problems are two main problems that supervised learning algorithms try to solve. In a regression problem, the model has to predict continuous-valued output while in classification, the model has to predict discrete-valued output.

1.2. Convolutional Neural Networks

Deep learning facilitates data representation and hierarchical learning through some sequential layers of abstraction [32, 50]. Deep learning models learn automatically and extract features from raw data using forward and backward propagations. These automatically extracted features in higher levels of abstraction are constructed from lower level features in the hierarchy [33]. Deep learning models outperform many complex tasks, particularly in the vision domain. Convolutional neural networks are deep learning models that contain at least one convolutional layer. Such networks are usually constructed by stacking some convolutional layers, pooling layers, and fully connected layers. It is worth mentioning that most of the learning time complexity of CNN models belong to the layers before dense layers, and most of the memory complexity is because of the memory requirement for keeping the dense layer parameters. A CNN expects to receive a tensor that represents the raw data which is often in three color channels. The input image has H rows, W columns, and three channels (R, G, B). Input tensors with higher dimensions will be treated in the same way in the CNN models. These input data will be processed in CNN models one layer after another. The following section explains the convolutional operation as a vital operation in the convolutional layer.

1.2.1. Convolutional Layer

Before the surge of deep learning and CNN models, filters were hand-designed by domain experts, which were then applied to an image to result in a feature map. Convolutional operation refers to convolving the filter on the image. Figure 1.1 shows convolving a vertical filter on the image to extract the vertical edges on the image. While hand-crafted features can provide good performance in some datasets, they are highly data-dependent and hence will not generalize to other datasets. CNN architectures take advantage of convolution operation and automatically extract features and filters using forward and backward propagation. CNNs start with a random uniform initialization of the filters at the beginning of the learning process and then find each layer's best filters using either batch gradient descent or stochastic gradient descent. Some hyperparameters that affect the convolutional operation include

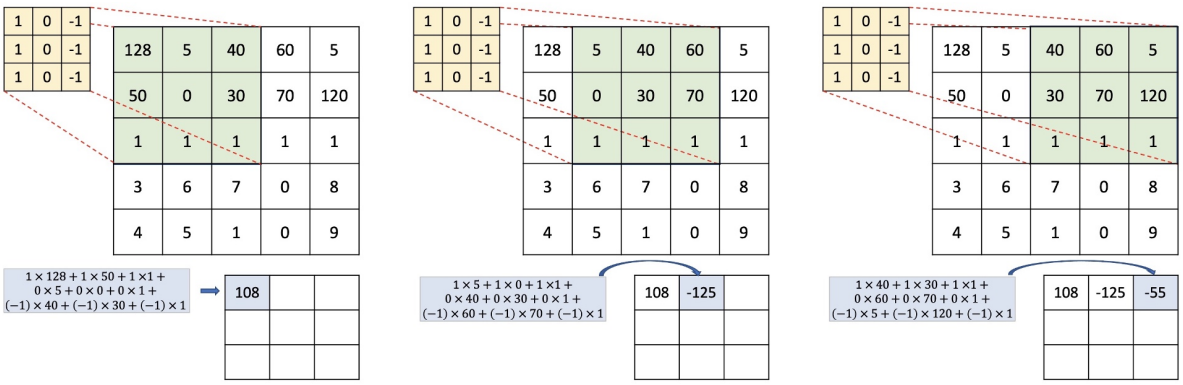


Fig. 1.1. Convolutional operation on a simple input using a vertical edge filter. The filter convolved on the input with the stride of one.

stride, padding, and dilation. Following is a summary of the definition of each of these hyperparameters.

Stride controls how the filter convolves and moves over the input tensor. In figure 1.1, the filter convolves around the input volume by shifting one unit at a time. Using a smaller stride value on convolutional operation helps to process more the center of the image, while a larger stride leads to considering the pixels close to the edges [75].

Padding is an additional layer of pixels that we can add to the border of an image. It indicates the number of pixels added to an input tensor before a convolutional operation gets applied. SAME and VALID paddings are two types of padding. Valid padding implies no padding at all. Therefore, the input image will remain in an unaltered shape. The same padding also refers to the case that p layers will be added to the input tensor's border such that the output tensor has the same dimensions as the input tensor.

Since a convolutional operation uses the middle pixels multiple times, the information in the middle of images will have more effect in extracted filters than the pixels on corners and edges. Adding padding around the input tensor helps preserve the border and corner information in the feature extraction process, as the information in the middle of the input tensor. If the padding is set to zero, then every pixel value that is added will be set to zero. Figure 1.2 shows the padding that is used to prevent losing the border pixels information.

Assume a convolutional layer receives input tensor for N samples in the mini-batch such that each sample has C_{in} channels. The input tensor can be presented as $(N, C_{in}, H_{in}, W_{in})$. After applying a convolutional operation on each channel we expect to have an output presented as $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ where C_{out} indicates the number of output channels after

applying a filter with kernel size of K. The width and height of output (O_w, O_h) for each channel is computed as follows:

$$O_h = \left\lfloor \frac{I_h + 2p_0 - d_0 \times (K_0 - 1) - 1}{S_0} + 1 \right\rfloor \quad And \quad O_w = \left\lfloor \frac{I_w + 2p_1 - d_1 \times (K_1 - 1) - 1}{S_1} + 1 \right\rfloor, \tag{1.2.1}$$

where p, d, and K denote the padding, dilatation, and the filter kernel respectively. Padding indices represent the padding on each side. Dilation indices indicate the height and width spacing between kernel elements. The default value for simple and straightforward convolution operation is one. If dilation is set to d as an integer number more than one, convolutions operation will get applied such that there will be $d - 1$ skipped cells between each filter. Figure 1.2 shows both dilation and padding effect in convolutional operation. Dilated convolution can achieve about 5% performance improvement on the test-set using adapted VGG-16 [56] model for dense predication [72].

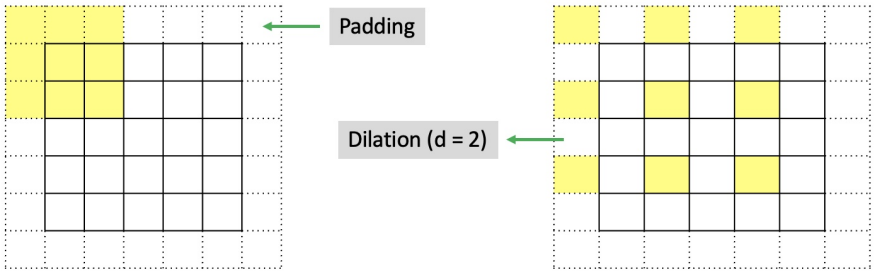


Fig. 1.2. Dilation and Padding.

A convolutional layer consists of convolutional operations, each acting on a different slice of the input tensor from the previous convolutional layer or on the image's tensor directly. Figure 1.3 illustrates a simple CNN model that is used for house numbers digit (SVHN dataset) classification. The proposed architecture leaded to 94.85% accuracy on 2012 while human performance to solve such task is 98.0% [52].

A CNN model is a multistage architecture that can automatically extract features. Figure 1.3 shows multiple stages that work as a part of a CNN model sequentially. In this architecture, the arrays of information that flow from one stage to another are called feature maps. A ConvNet has several consecutive layers that are separated into multiple stages. Each layer can be either a convolution layer for extracting features, a feature pooling layer, or a non-linearity layer. The convolutional network remarkably outperforms a broad spectrum of perceptual tasks in comparison to simple MLP models. Like most supervised learning methods, training a CNN model as an efficient and accurate model requires a large number

of labeled training samples. While CNN models are still essential methods in vision tasks, some recent works tried to propose alternative architectures like Capsule Networks [49]. CNN models can not find the orientational and relative spatial relationships between the elements in an image that capsule networks tries to capture. However, they are still far from the CNN models' performance in large datasets like ImageNet [24].

The performance of CNNs depends on the samples we have in the training set, and a slight shift in data distribution drastically reduces the CNN models' performance. That makes CNN models very naive against even week adversarial attacks. CNNs can overfit or underfit the data depending on the hyper-parameters, model structures, and training data. [12]. In this thesis, we study some regularization techniques that prevent overfitting in the CNN model and improve the CNN model's generalization. For a proof of concept, we consider the ResNet Architecture in the experimental setup. However, the proposed method is applicable in other CNN models with different architectures without any required changes. The following section gives an overview of the generalization and regularization techniques.

1.3. Generalization and Regularization

A model overfits the training data when the model performs well on the training data but provides a poor performance on the evaluation data (unseen data) [14]. In other words, there is a considerable gap between the model's performance on training data and the model performance on the evaluation dataset. This gap is also known as the generalization gap. Figure 1.4 illustrates the generalization gap on a model performance on training in Tiny-ImageNet dataset using WideResNet-28-10 model after 400 training epochs. The generalization gap in this example is about 40%, which indicates the model completely overfitted to the training dataset.

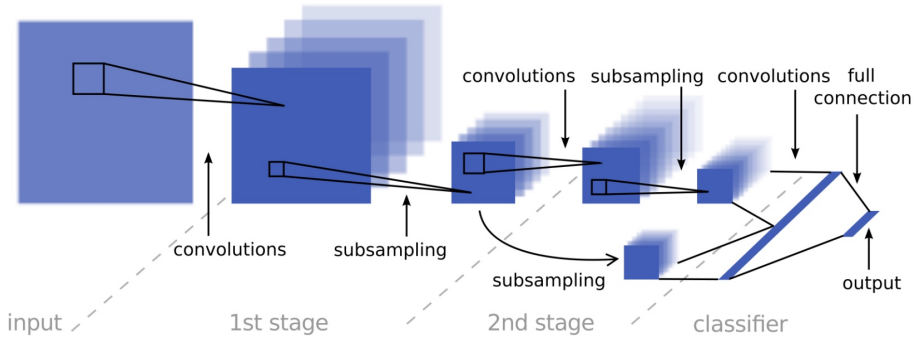


Fig. 1.3. A simple CNN model for house numbers digit (SVHN dataset) classification [52].

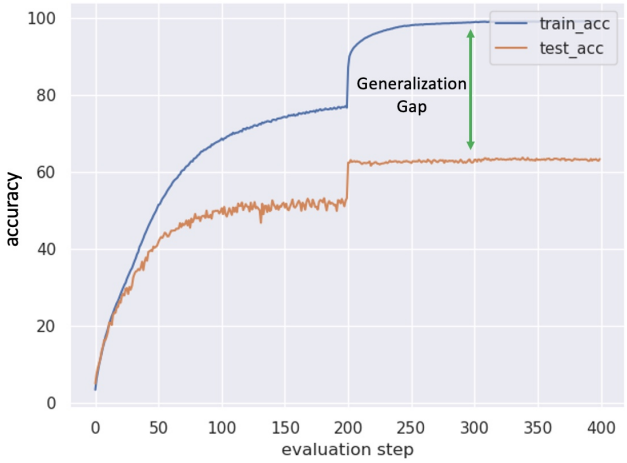


Fig. 1.4. The blue line shows the model's performance in the training set, and the orange line shows the accuracy of the model at test time. The green arrow illustrates the generalization gap between training and unseen data.

Several techniques are proposed to reduce the generalization gap and avoid overfitting. A straightforward approach to avoid overfitting problem is to reduce the model's capacity up to the point that the model still performs well in training but not by memorizing the training samples. This can be done by using network-reduction methods such as pre-pruning, post-pruning, and model compression [70].

Another approach to improve the generalization of the model is by expanding the training data. In supervised learning, the training data and annotations that are used for supervising at training time and the quality of this information have significant effects on the performance of the model on unseen data. Data expansion methods include some effective approaches such as exploiting more training data [14, 58], randomly expanding training sets [26], adding some random noise to the existing dataset [70], and producing some new data based on the distribution of the existing data set [70, 71].

Two other approaches for improving a model's generalization are applying early stopping and parameter norm penalties [14]. Early stopping avoids overfitting and, as a result, reduces the generalization gap. However, understanding when we should apply early stopping is another challenge that one should consider [42].

In the parameter norm penalties approach, L1 regularization and L2 regularization (Weight Decay) are two well-known methods that have been used to improve the generalization of the model. L1 regularization adds minimizing the sum of the absolute value of the parameters into the learning objective of the model, and L2 regularization or weight decays push the parameters closer to the origin by adding minimizing $\frac{1}{2}\|\boldsymbol{w}\|_2^2$ to the learning

objective [14]. Since the performance of a model is dependent on the model parameters, controlling them using regularization techniques can be beneficial.

Regularization techniques can be categorized into two main categories, including explicit regularization and implicit regularization techniques. The goal of the explicit regularization technique is to reduce the representational capacity of the model. The implicit regularization technique affects the model's sufficient capacity instead of reducing the model's representational capacity using optimization techniques or other tricks [21].

Recently, adding some noisy computation while training has shown some improvement in the model generalization. Noisy computation is often employed during the training, making the model more robust against invariant samples and thus improving the generalization of the model [1]. 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 methods [16]. Earlier work in this area has been more focused on the data-independent techniques such as Dropout [57], Variational Dropout [11] and ZoneOut [30], Information Dropout [1], SpatialDropout [62], and DropBlock [13]. Dropout performs well on fully connected layers [57]. However, it is less effective on convolutional layers [63]. 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 [63] 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.

Data-augmentation also is a data-dependent solution to improve the generalization of a model. Choosing the best augmentation policy is challenging. AutoAugment [8] finds the best augmentation policies using reinforcement learning with huge computation overhead. AugMix [20] 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 [76], one such data-dependent regularizer, synthesizes additional training examples by linearly interpolating random pairs of inputs such that the linear interpolation coefficient λ is sampled from a Beta distribution. By using these types of synthetic samples, Mixup encourages the model to behave linearly in-between the training samples.

The mixing coefficient λ in Mixup is sampled from a prior distribution. This may lead to the *manifold intrusion problem* [16]: the mixed synthetic example may collide (i.e., have the same value in the input space) with other training data examples, essentially leading

to two training samples that have the same inputs but different targets. To overcome the manifold intrusion problem, MetaMixUp [38] used a meta-learning approach to learn λ with a lower possibility of causing such collisions. However, this meta-learning approach adds significant computation complexity. ManifoldMixup [65] 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 [27] has explicitly exploited an optimized masking strategy for the Input Mixup. It uses the saliency information and the underlying statistics of pair of images to avoid manifold intrusion problems at each batch training step. Puzzle Mix adds computation overhead at training time to find an optimal mask policy while improving the model's performance compared to Mixup and ManifoldMixup.

Unlike the interpolation-based regularizers discussed above, Cutout [9] 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 [73] is another data-dependent regularization technique that cuts and fills rectangular-shaped 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 [73].

1.4. Contributions

The key contribution of this thesis is introducing *PatchUp*, a feature-space, block-level, data-dependent regularization technique that operates in the hidden space of CNN models. We propose masking out contiguous blocks of the feature map of a random pair of samples, and then either mix (*Soft PatchUp*) or swap (*Hard PatchUp*) these selected blocks to obtain more diverse samples for training towards different dimensions and therefore achieve a better image classification performance and more robustness against adversarial examples and deformed images. Our regularization method does not incur significant computational overhead for CNNs during training.

1.5. Thesis Layout

The rest of the thesis is organized as follows. Chapter 2 explains the modern regularization techniques that can improve the generalization of a CNN model. Chapter 3 introduces *PatchUp*, 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. Chapter 4 contains all the experimental results, ablation studies, and analysis on *PatchUp*. Chapter 5 concludes the thesis and outlines future research directions.

Chapter 2

Regularization Techniques for CNNs

Regularization methods help prevent overfitting to the training set distribution and improve the generalization performance in machine learning models [31]. The goal is to make the model robust to previously unseen data, assuming the unseen data is drawn from the same distribution as the training data. Regularization schema can be categorized into data-dependent and data-independent techniques [16]. In this chapter, we summarize some state-of-the-art regularization methods from both approaches that have been used for Convolutional Neural Networks (CNNs).

2.1. Data-dependant Regularization Methods

Data-dependent regularization uses the structure of the data to constrain the model parameter space [16]. This includes data augmentation methods [34, 53, 55], adversarial training schemes [15], and some state-of-the-art regularization methods that work either in input space or in latent representation space.

The straightforward regularization approach uses data augmentation methods that improve the model's generalization performance by providing more diverse data. Data augmentation methods create diverse data by randomly augmenting the original data [4, 29, 54]. The most common augmentations that have been used for training CNNs are translating the image by a few pixels, randomly cropping, resizing, and flipping the image horizontally. Apart from these traditional transformations, one could also use generative adversarial networks [25, 35, 45, 67, 69] and texture transfer [10, 22] for data augmentation. Since data augmentation methods are considered data-dependent regularization methods, choosing the best data augmentation method is a challenging task since the effectiveness of the selected augmentation methods depends on the structure of the data. Recent works [8, 20] show that chaining and combining augmentation policies can improve the effectiveness of the data augmentation method. However, finding such chained or combined policies introduces additional

challenges. This section contains a summary of some popular data-dependent regularization methods that improve the model performance by exploiting the data structure.

2.1.1. AutoAugment

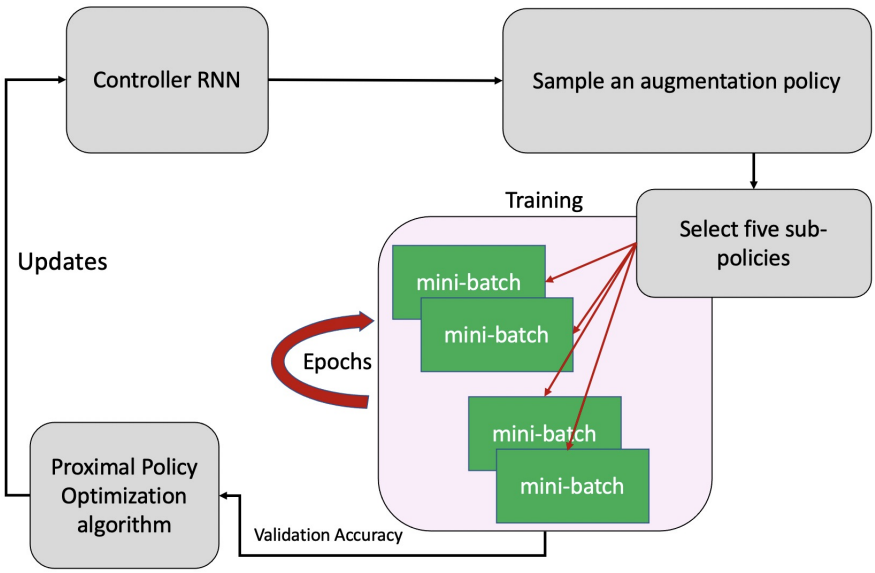


Fig. 2.1. Controller RNN samples a policy and trains model using five sub-policies created based on the selected policy. Then, the searching algorithm receives the validation accuracy as a reward signal and updates the controller RNN.

As mentioned before, searching for an augmentation strategy that provides better performance is a time-consuming task that can be automated. AutoAugment introduces a reinforcement learning (RL) approach to automatically find a better data augmentation strategy for a given data and model [8]. AutoAugment has two components: a controller that is a simple RNN model and a Proximal Policy Optimization (PPO) [51] algorithm that uses the validation accuracy to select the best policy.

The search space is limited to 5 sub-policies. Users can design the search space by creating each sub-policy in their experiment. Each sub-policy is constructed using two data augmentations that can be applied in sequence. AutoAugment calls each data augmentation as an image operation. The probability and the magnitude of the operation are two hyperparameters that are related to each operation. The Controller RNN is responsible for sampling an augmentation strategy at each time step, training the model using the selected data augmentation policy, and computing the validation accuracy. Then, the PPO component uses the computed validation accuracy as a reward to update the controller using reinforcement

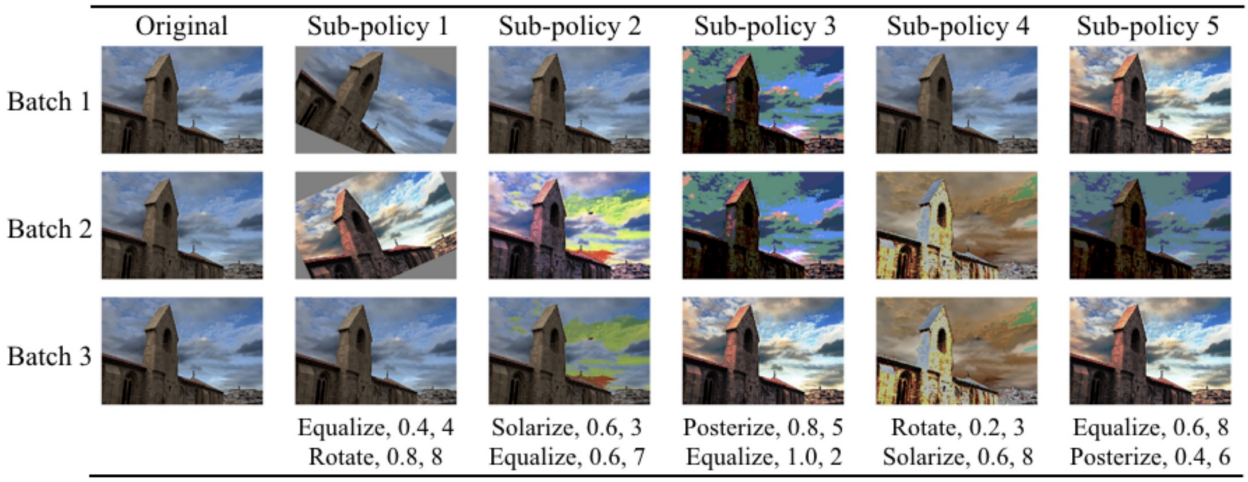


Fig. 2.2. For each example in the mini-batch, one of the 5 sub-policies is chosen uniformly random to augment the image. Last row specifies the sub-policy that is applied in each images. First number indicates the probability of applying each operation and the second number shows the magnitude of the operation out of 10 (the image is borrowed from AutoAugment paper [8]).

learning [51]. After selecting a policy, the controller RNN uses the sub-policies created based on the selected policy for training each of the mini-batches. Each policy has five sub-policies that are selected uniformly at random to augment images in a mini-batch. Each sub-policy consists of two operations, as well as a probability of actually calling each operation and the magnitude with which each operation is applied. Figure 2.1 shows the overview of the AutoAugment method and Figure 2.2 illustrates how the sub-policies are applied on images for each mini-batch. As figure 2.2 shows, five sub-policies that are designed for the ImageNet dataset. For example, the first sub-policy consists of two image operations, Equalize and Rotate, that each has two hyperparameters. It also shows in 80% of the case (0.8), the operation will be applied with a strength of 8 out of 10.

AutoAugment provides significant improvement in performance on several image recognition benchmarks. However, its drawback is that to achieve such an improvement, it requires a massive number of GPU hours for computation, even for small datasets [36].

2.1.2. AugMix

AutoAugment improves model performance by searching and finding the best augmentation strategy during training. However, this adds a huge computational overhead. AugMix [20] reduces this overhead by using stochasticity, diverse augmentations, and by adding a Jensen-Shannon Divergence consistency loss to the training loss.

AugMix mixes a chain of augmentation operations that are used in AutoAugment. It samples k augmentation chains, each composed of one to three randomly selected augmentation strategies. AugMix then mixes the augmentation chains to create final augmented images. To select the k-dimensional vector of k augmentation chains, AugMix uses the $Dirichlet(\alpha, \ldots, \alpha)$ distribution as a mixing policy. After constructing the augmented image by mixing the chain of augmented images, AugMix linearly interpolates the augmented images with the original image using a mixing coefficient that is randomly sampled from the $Beta(\alpha, \alpha)$ Distribution.

Several sources of stochasticities are applied to construct the final images, including the operation selection, the severity of the operations, and the coefficient of linear interpolations. Algorithm 1 explains the AugMix algorithm in detail. After creating two augmented samples from the original image (as declared in lines 14 and 15 in Algorithm 1), AugMix defines the loss function by adding the Jensen-Shannon divergence to the classification loss (line 16). Therefore, the loss function is computed as follow [20]:

$$\mathcal{L}(p_{\text{orig}}, y) + \lambda \, \text{JS}(p_{\text{orig}}; p_{\text{augmix1}}; p_{\text{augmix2}}). \tag{2.1.1}$$

where

$$\text{JS}\left(p_{\text{orig}}\,; p_{\text{augmix1}}\,; p_{\text{augmix2}}\,\right) = \frac{1}{3}\left(\text{KL}\left[p_{\text{orig}}\,\|M\right] + \text{KL}\left[p_{\text{augmix 1}}\|M\right] + \text{KL}\left[p_{\text{augmix 2}}\|M\right]\right) \tag{2.1.2}$$

$$M = \frac{1}{3}\left(p_{\text{orig}}\; + p_{\text{augmix 1}} + p_{\text{augmix 2}}\right) \tag{2.1.3}$$

2.1.3. Mixup

Mixup [76] synthesizes additional training examples by interpolating random pairs of inputs x_i, x_j and their corresponding labels y_i, y_j as:

$$\tilde{x} = \lambda x_i + (1-\lambda)x_j \quad \text{and} \quad \tilde{y} = \lambda y_i + (1-\lambda)y_j, \tag{2.1.4}$$

where $\lambda \in [0,1]$ is sampled from a Beta distribution such that $\lambda \sim Beta(\alpha, \alpha)$ and $(\tilde{x}, \tilde{y})$ is the new example. By using these types of synthetic samples, Mixup encourages the model to behave linearly in-between the training samples. We know that the goal of the learning function f that maps input $x \in X$ to target $y \in Y$ is minimizing the empirical risk using the following loss function:

$$L(f) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(y_i, f\left(x_i\right)\right) \tag{2.1.5}$$

Algorithm 1 AugMix Pseudocode (source of the algorithm is the AugMix paper. [20]).

Input:

Model $\hat{p}$, Classification Loss $\mathcal{L}$, Image x_{orig}, Operations $\mathcal{O} = \{\text{rotate}, \ldots, \text{posterize}\}$

Output:

loss (Loss Output)

function AugmentAndMix(x_{orig}, $k = 3$,$\alpha = 1$)
  Fill x_{aug} with zeros
  Sample mixing weights $(w_1, w_2, \ldots, w_k) \sim \text{Dirichlet}(\alpha,\alpha,\ldots,\alpha)$
  for $i = 1, \ldots, k$ **do**
    Sample operations $\text{op}_1, \text{op}_2, \text{op}_3 \sim \mathcal{O}$
    Compose operations with varying depth $\text{op}_{12} = \text{op}_2 \circ \text{op}_1$ and $\text{op}_{123} = \text{op}_3 \circ \text{op}_2 \circ \text{op}_1$
    Sample uniformly from one of these operations chain $\sim \{\text{op}_1, \text{op}_{12}, \text{op}_{123}\}$
    $x_{\text{aug}} \mathrel{+}= w_i \cdot \text{chain}(x_{\text{orig}})$
  end for
  Sample weight $m \sim \text{Beta}(\alpha, \alpha)$
  Interpolate with rule $x_{\text{augmix}} = m x_{\text{orig}} + (1 - m) x_{\text{aug}}$
  return x_{augmix}
end function
$x_{\text{augmix1}} = \text{AugmentAndMix}(x_{\text{orig}})$ # x_{augmix1} is stochastically generated
$x_{\text{augmix2}} = \text{AugmentAndMix}(x_{\text{orig}})$ # $x_{\text{augmix1}} \neq x_{\text{augmix2}}$
loss $= \mathcal{L}(\hat{p}(y \mid x_{\text{orig}}), y) + \lambda$ Jensen-Shannon$(\hat{p}(y \mid x_{\text{orig}}); \hat{p}(y|x_{\text{augmix1}}); \hat{p}(y|x_{\text{augmix2}}))$
return loss

where n is the total number of training examples.

Since Mixup linearly interpolates the pair of input samples and therefore their corresponding targets in supervised learning algorithms, it should minimize the empirical vicinal risk as follow:

$$L(f) = \frac{1}{m} \sum_{i=1}^{m} \ell\left(\tilde{y}_i, f\left(\tilde{x}_i\right)\right) \tag{2.1.6}$$

where m is the number of samples sampled from the generic vicinal distribution, called mixup denoted as $\mu\left(\tilde{x}, \tilde{y} \mid x_i, y_i\right)$ which is computed as follows:

$$\mu\left(\tilde{x}, \tilde{y} \mid x_i, y_i\right) = \frac{1}{n} \sum_{j}^{n} \mathop{\mathbb{E}}_{\lambda} \left[\delta\left(\tilde{x} = \lambda \cdot x_i + (1 - \lambda) \cdot x_j, \tilde{y} = \lambda \cdot y_i + (1 - \lambda) \cdot y_j\right)\right] \tag{2.1.7}$$

Mixup regularizes a neural network model by generating new augmented examples which have a linear relation to existing examples in the dataset [76]. To this end, it can provide some kind of adversarial-like examples besides the existing samples in the training set. Mixup is an effective regularization approach that helps to generalize better in deep learning models [76]. The mixing coefficient λ in Mixup is sampled from a prior distribution. This may lead to

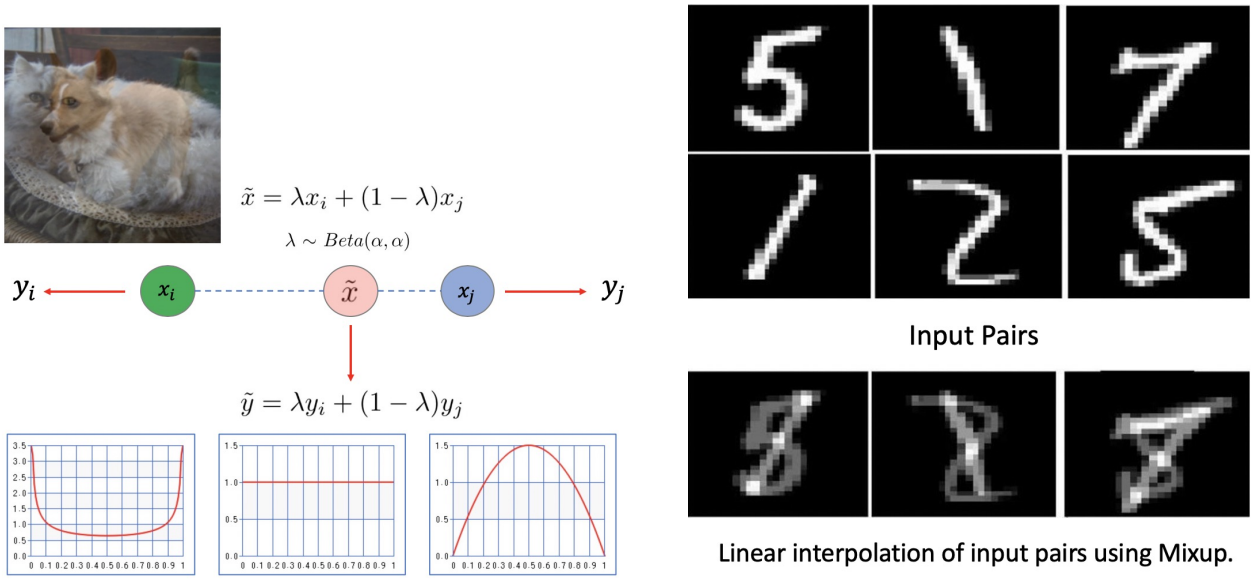


Fig. 2.3. (Left) Mixup overview that shows the linear interpolation of a random pair of samples (a dog and a cat as shown in the figure) using λ as a coefficient of interpolation that is sampled from the Beta Distribution. Three curves show the beta distribution for α equal to .5, 1, and 2 from left to right, respectively. (Right, borrowed from [16]) Manifold intrusion for three pairs of samples from the MNIST dataset. The mixed samples intrude with class 8.

the *manifold intrusion problem* [16]: 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. Figure 2.3 shows an example of manifold intrusion problem for three pairs of samples from the MNIST dataset.

2.1.4. MetaMixup

In the Mixup method, the random selection of λ from the range of [0, 1] might cause the manifold intrusion problem which would lead the model to predict incorrect labels for the interpolated inputs. To overcome the manifold intrusion problem, MetaMixUp [38] 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.

MetaMixup initially uses a random value for λ, generated by a prior distribution for each minibatch. Then, it uses the validation set to tune the λ with one step of meta-learning training. In doing so, it learns λ^*, which is the optimized λ that alleviates manifold intrusion by decreasing the negative effect of the Mixup interpolation. Since, the λ^* has to be in the

range of [0, 1], MetaMixup uses a sigmoid function to map the optimized λ to the range of [0, 1]. Algorithm 2 explains the MetaMixup process in detail.

Algorithm 2 MetaMixUp (source of the algorithm is the MetaMixUp paper [38])

Input:
Training data $\mathcal{D}$, validation $\mathcal{D}_v$.
Output:
Deep neural network $\Phi(\theta)$ and λ^*.
Parameters:
Deep neural network $\Phi(\theta)$, batch size B, learning rate η, step size α.

for $t = 1, 2, \ldots, Iter_{max}$ **do**
  Shuffle training set $\mathcal{D}$;
  for $n = 1, \ldots, \frac{|\mathcal{D}|}{B}$ **do**
    Fetch mini-batch $\bar{D}$ from $\mathcal{D}$;
    Fetch mini-batch $\bar{D}_v$ from $\mathcal{D}_v$;
    Random initialize $\lambda = \{\lambda_i\}_{i=1}^{B}$;
    Turn network to meta stage $\Phi(\theta) \rightarrow \Phi^{'}(\theta)$;
    MixUp examples with λ to construct $\tilde{\mathcal{D}}$;

    Update $\theta^{'} = \theta - \eta \nabla_\theta \ell(\Phi^{'}(\theta), \tilde{\mathcal{D}})$;
    Update $\lambda^* = \lambda - \alpha \nabla_\lambda \ell(\Phi^{'}(\theta^{'}), \mathcal{D}_v)$;
    MixUp examples with updated λ^* to reconstruct $\tilde{\mathcal{D}}$;
    Update nework $\theta := \theta - \eta \nabla_\theta \ell(\Phi(\theta), \tilde{\mathcal{D}})$;
  end for
end for
return $\Phi(\theta)$ and λ^*

2.1.5. Cutout

Different from the interpolation-based regularizers discussed above, Cutout [9] drops the contiguous regions from the image in the input space. Figure 2.4 (right) shows the Cutout at input space. Cutout is inspired by intuitions from dropout. Dropout performs well on fully connected layers, but it is less effective on convolutional layers. Since convolutional layers have much fewer parameters than fully connected layers, they require less regularization. Furthermore, with convolutional layers, neighboring pixels in images share much information. By dropping pixels from images, these pixels' neighbors will likely still pass the dropped pixels' information. Because of these two reasons, dropout has a smaller effect on improving the performance of models and increasing their robustness.

Cutout tries to provide a similar effect as dropout to improve the generalization of the deep learning models but operates at the input stage of the network rather than in the hidden layers [9]. Cutout drops a contiguous section of input image instead of random individual pixels. This allows Cutout to drop and remove the randomly selected region in all subsequent feature maps. In doing so, it forces the network to focus on the entire input and be less dependent on a specific section of features. The strength of this approach is removing the transition of vicinity information of dropped points to the next stage by dropping randomly contiguous selected points not at the intermediate level but at the input stage [9]. Cutout encourages the model to produce a wider variety of features when making predictions instead of relying on the presence of a limited number of features.

Cutout shows that the size of the cutout region is a more important hyperparameter than the shape. Therefore, for simplicity, they conduct their experiment with a square-shaped patch as the cutout region. Figure 2.4 (left) shows the effects of different patch sizes on validation accuracy in the WideResNet-28-10 [74] model for CIFAR10 and CIFAR100 image classification tasks. The red line is the baseline or the vanilla model performance. It also shows that for ten classes, the best value for the patch size is 16. However, with the increased number of classes, the best patch size is eight. Cutout observes that it is important for the model to receive some images where a large portion of the image is not dropped during training. The cutout operation improves the model performance without adding a computational overhead in training. It is can easily be applied on the CPU and mixed with other augmentation strategies.

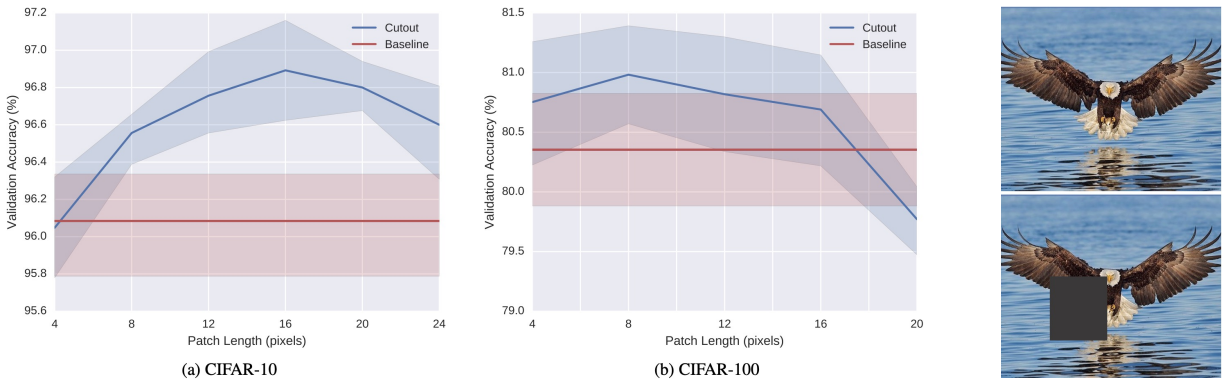


Fig. 2.4. (Left) Patch size effect comparison on CIFAR dataset in validation accuracy in WideResNet-28-10 model [9]. (Right) Cutout drops the contiguous regions from the image.

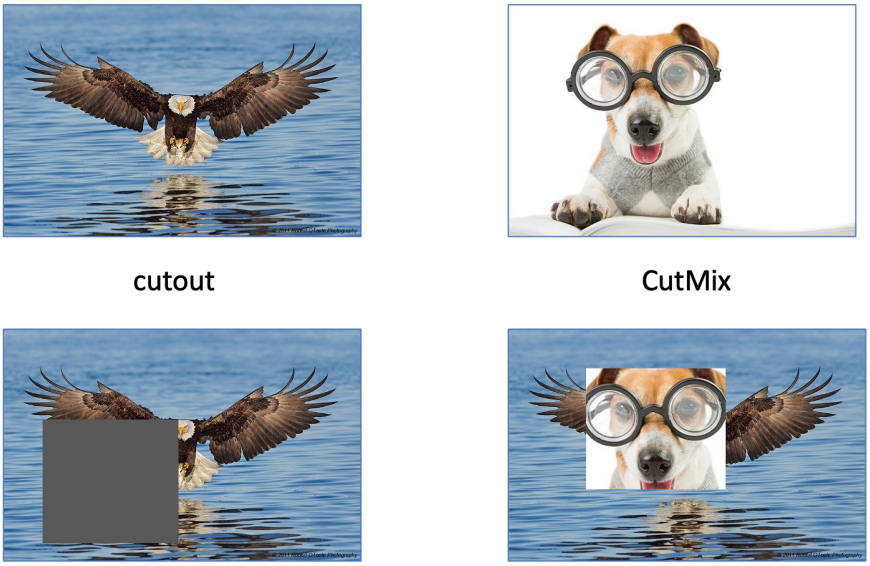


Fig. 2.5. CutMix comparison with Cotout. While Cutout just drops the selected patch, CutMix replaces the selected patch with a different patch from another image.

2.1.6. CutMix

CutMix [73] is another data-dependent regularization technique that cuts and replaces a rectangular shaped part from an image with a randomly selected patch from another image in the mini-batch, instead of interpolating the two selected images 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 [73]. CutMix alleviates the manifold intrusion problem by cutting and filling some parts of the pair instead of interpolating two inputs completely [73]. CutMix also provides a promising positive effect on the generalization of a very deep and wide CNN model such as PyramidNet [73].

CutMix generates a new training sample from two input samples (x_A, x_B) with (y_A, y_B) as their associated labels. The new sample is created using the following combining operation:

$$\tilde{x} = \mathbf{M} \odot x_A + (\mathbf{1} - \mathbf{M}) \odot x_B$$
$$\tilde{y} = \lambda y_A + (1 - \lambda) y_B$$

In the combining operation $\mathbf{M} \in \{0,1\}^{W \times H}$ is a binary mask that defines the drop out and fill in part of two images. As, figure 2.5 shows, CutMix selects a bounding-box from the second input image and fill it in the same position in the first image.

The bounding-box $\mathbf{B} = (r_x, r_y, r_w, r_h)$ region is sampled uniformly in the following way:.

$$r_x \sim \text{ Unif } (0, W), r_w = W\sqrt{1-\lambda}$$
$$r_y \sim \text{ Unif } (0, H), \quad r_h = H\sqrt{1-\lambda}$$

Since the cropped area ratio is $\frac{r_w r_h}{WH} = 1 - \lambda$, CutMix defines the interpolated target by $\tilde{y} = \lambda y_A + (1-\lambda) y_B$.

Since parts of the pair of images are swapped the loss function of the CutMix should be modified accordingly. Mathematically, CutMix minimizes:

$$L(f) = \mathbb{E}_{(x,\, y)\sim P}\ \mathbb{E}_{(x',\, y')\sim P}\ \mathbb{E}_{\lambda\sim \text{Beta}(\alpha,\, \alpha)}\, \text{Mix}_{\lambda=p_u}(\ell(f(\tilde{x}),\, y), \ell(f(\tilde{x}),\, y')). \tag{2.1.8}$$

where p_u is the portion of the unchanged part of the input and $\tilde{x}$ is the image produced using binary mask described in CutMix approach denoted as $\mathbf{M}$ that indicates where to swap the two input data in their three-colored channels [73]. The performance of the CutMix approach depends heavily on the W and H (the size of input images). There are no experiments that show this approach does not depend on those parameters. One positive point from CutMix is that it relaxed the alpha and set it to one. Algorithm 3 explains the CutMix algorithm in detail [73].

2.1.7. ManifoldMixup

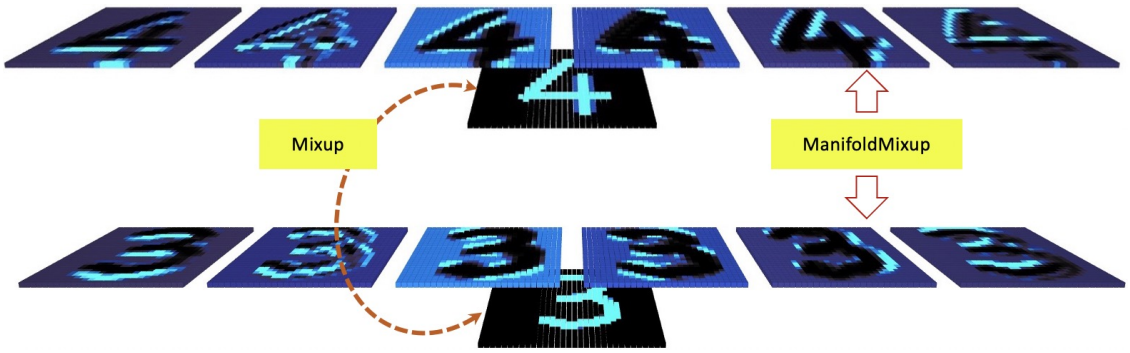


Fig. 2.6. ManifoldMixup interpolates the hidden states of a randomly chosen layer including the input layer at every training update. However, Mixup interpolation get applied only the input space.

ManifoldMixup [65] 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. Figure 2.6 shows the main differences between ManifoldMixup and Mixup. ManifoldMixup provides smoother decision boundaries at multiple levels of representation and

Algorithm 3 CutMix (source of the algorithm is the CutMix paper [73]).

Input:

input: samples from a mini-batch in N×C×W×H size tensor.
target: target of samples in a mini-batch in N×K size tensor.

```
if mode == training then
  input_s, target_s = shuffle_minibatch(input, target) # CutMix starts here.
  lambda = Unif(0,1)
  r_x = Unif(0,W)
  r_y = Unif(0,H)
  r_w = Sqrt(1 - lambda)
  r_h = Sqrt(1 - lambda)
  x1 = Round(Clip(r_x - r_w / 2, min=0))
  x2 = Round(Clip(r_x + r_w / 2, max=W))
  y1 = Round(Clip(r_y - r_h / 2, min=0))
  y2 = Round(Clip(r_y + r_h / 2, min=H))
  input[:, :, x1:x2, y1:y2] = input_s[:, :, x1:x2, y1:y2]
  lambda = 1 - (x2-x1)*(y2-y1)/(W*H) # Adjust lambda to the exact area ratio.
  target = lambda * target + (1 - lambda) * target_s # CutMix ends.
end if
output = model_forward(input)
loss = compute_loss(output, target)
model_update()
```

improves the log-likelihood on the test set. Therefore, the model is more robust against adversarial examples and has better generalization for unseen data.

In ManifoldMixup a neural network is defined as $f(x) = f_k(g_k(x))$, where g_k maps the input data to the hidden representation at layer k. Then, the hidden representation is mapped by f_k to the output f(x). In this approach, the k is randomly selected first. It denotes the layer whose output will be used in the ManifoldMixup algorithm. After that, two samples (x, y) and $(x\prime, y\prime)$ from a mini-batch go through the forward process until the selected layer. ManifoldMixup processes two random data mini-batches as usual until reaching layer k. At this stage, we are going to apply g_k. To compute the output, we have to perform the Mixup approach to the outputs of $(g_k(x),y)$ and $(g_k(x\prime),y\prime)$.

$$(\tilde{g}_k, \tilde{y}) := \left(Mix_\lambda\left(g_k(x), g_k\left(x'\right)\right), Mix_\lambda\left(y, y'\right)\right) \tag{2.1.9}$$

where $\lambda \sim Beta(\alpha, \alpha)$. Finally, we continue the forward process to the end of network and calculate the gradients of all parameters of the network in the back-propagation phase. The

loss of the ManifoldMixup [65] is as follows:

$$L(f) = \mathbb{E}_{(x,y)\sim P}\ \mathbb{E}_{(x\prime,y\prime)\sim P}\ \mathbb{E}_{\lambda\sim \mathrm{Beta}(\alpha,\alpha)}\ \mathbb{E}_{k\sim \mathcal{S}}\, \ell(f_k(\mathrm{Mix}_\lambda(g_k(x), g_k(x\prime))), \mathrm{Mix}_\lambda(y, y\prime)). \quad (2.1.10)$$

In this approach, k and λ can be selected randomly for each minibatch. ManifoldMixup takes advantage of Mixup by performing the Mixup for input samples of mini-batches. Manifold Mixup can lead to flatter class-specific representations that could not be provided by other regularizers. Since ManifoldMixup pushes the boundary decision away, the model has robust behavior in adversarial examples. In general, ManifoldMixup delivers better performance, faster learning convergence, and a smaller gap in training and testing performance.

2.1.8. Puzzle Mix

Recently, Puzzle Mix [27] has explicitly exploited an optimized masking strategy for the Input Mixup. It uses the saliency information and the underlying statistics of pairs of images to avoid manifold intrusion problems at each batch training step. Puzzle Mix considers an optimization problem alternating between the multi-label objectives for interpolated pairs of images for optimal mixing mask using the saliency information for an optimal transport objective. To avoid the manifold intrusion problem after the mixing operation, Puzzle Mix utilizes the saliency information to find the main object in each image and then uses this information to transport the object before applying the mixing procedure to avoid such a mix that leads to manifold intrusion problem. Puzzle Mix adds computation overhead at training time to find an optimal mask policy while improving the model's performance compared to Mixup and ManifoldMixup.

2.2. Data-independent Regularization Methods

The second category of regularization methods is the data-independent regularization approach. Data-independent regularization improves model performance in unseen data without exploiting the structure or the distribution of data [16]. The most well-known methods in this direction are dropout and all its various forms, including Variational Dropout [11], ZoneOut [30], and other dropout methods proposed specifically to improve CNN models' performance. This section contains a summary of DropBlock, a state-of-the-art Convolutional dropout technique.

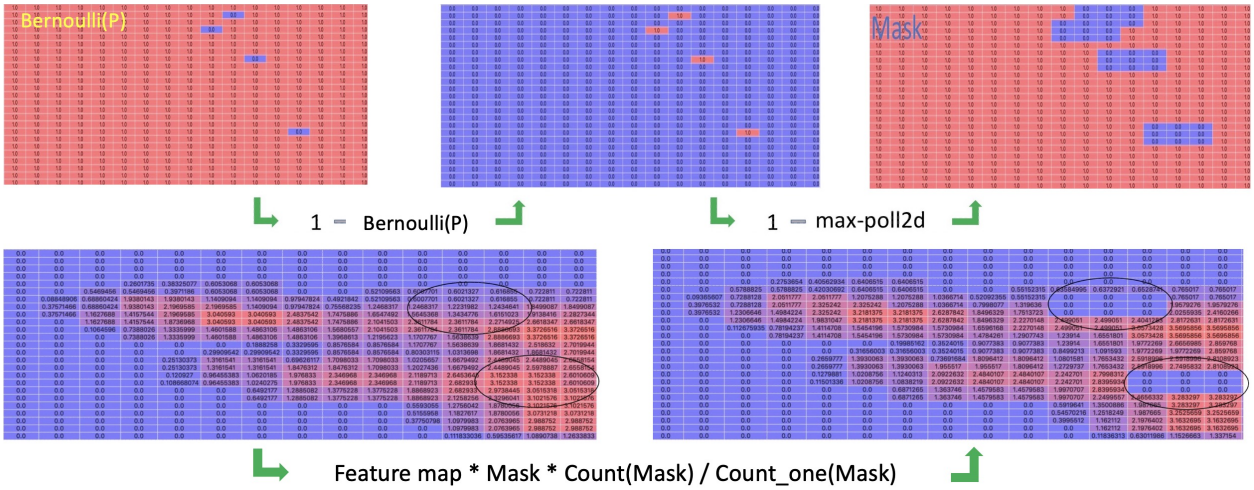


Fig. 2.7. Mask sampling in DropBlock and the process of dropping the continuous block of features from the feature map. DropBlock creates a Binnary mask using Bernoulli distribution. The P parameter for each feature in the feature map is computed as γ in DropBlock. The process started from the top left to the right.

2.2.1. DropBblock

While dropout performs well on fully connected layers [57], it is less effective on convolutional layers [63]. 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 [63] addresses this issue by dropping the entire feature map from a convolutional layer. DropBlock [13] further improves SpatialDropout by dropping random contiguous feature blocks from feature maps instead of dropping the entire feature map in the convolutional layers. DropBlock works as a regularization method similar to dropout but specifically designed for convolutional blocks. Pseudocode of DropBlock is shown in Algorithm 4. DropBlock uses two hyperparameters: $block\_size$, which represents the size of the block to be dropped and γ, which controls how many activation units to drop. Figure 2.7 illustrates the summary of dropping continuous blocks of features from the future map.

DropBlock explicitly sets γ that controls the number of features to drop. Suppose that we want to keep every activation unit with the probability of $keep\_prob$. In dropout [57] the binary mask will be sampled with the Bernoulli distribution with mean $(1 - keep\_prob)$. However, to account for the fact that every zero entry in the mask will be expanded by $block\_size^2$ and that the blocks will be fully contained in the feature map, we need to adjust γ accordingly when we sample the initial binary mask. DropBlock computes γ as

follows:

$$\gamma = \frac{1 - keep\_prob}{block\_size^2} \frac{feat\_size^2}{(feat\_size - block\_size + 1)^2} \tag{2.2.1}$$

where *keep_prob* is the probability of keeping a feature in normal dropout. The space to select a feature is limited to $(feat\_size - block\_size + 1)^2$ where $feat\_size$ is the size of feature map. DropBlock provides a binary mask such that there might be some overlap in the candidate dropping-regions. Therefore, Equation 2.2.1 gives an approximation of how many activation units to drop.

Algorithm 4 DropBlock (source of the algorithm is the DropBlock paper [13]).

Input:

A: output activations of a layer.
block_size: the size of each block in the binary mask.
γ: the probability of altering a feature.
mode: either *inference* or *training*.

Output:

A': output activations of a layer after dropping some continuous blocks of features.

1: **if** *mode* == *Inference* **then**
2: return A
3: **end if**
4: $M_{i,j} \sim Bernoulli(\gamma)$ # Randomly sample mask M
5: For each zero position $M_{i,j}$, create a spatial square mask with the center being $M_{i,j}$, the width, height being *block_size* and set all the values of M in the square to be zero.
6: $A = A \times M$ # Apply the mask
7: $A' = A \times \text{count}(M)/\text{count\_ones}(M)$ # Normalize the features
8: **return** A'

Chapter 3

PatchUp

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

Submitted to CVPR 2021.

Contribution:

- I came up with the idea under the supervision of Prof. Sarath Chandar. Prof. Sarath Chandar provided significant support and help for the further development of the idea.
- Experiment setups and analysis were designed by me and Prof. Sarath Chandar with valuable feedback from Vikas Verma.
- I wrote all the code for algorithms and experiments reported in this thesis.
- Mohammad Amini and Akilesh Badrinaaraayanan helped me in running experiments and collecting results.
- I also wrote the entire paper with valuable help from Prof. Sarath Chandar, Vikas Verma, and Mohammad Amini.

Affiliation:

- Mojtaba Faramarzi: Mila - Quebec AI Institute, Université de Montréal.
- Mohammad Amini: Mila - Quebec AI Institute, McGill University.
- Akilesh Badrinaaraayanan: Mila - Quebec AI Institute, Université de Montréal.
- Vikas Verma: Mila - Quebec AI Institute, Université de Montréal, Aalto Univeristy, Finland.
- Sarath Chandar: Mila - Quebec AI Institute, École Polytechnique de Montréal, Canada CIFAR AI Chair.

In this chapter, we introduce *PatchUp*, 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.

3.1. 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 [65]. In every training step, *PatchUp* applies block-level regularization at a randomly selected convolutional layer k from a set of intermediate convolutional layers. Section-4.10 gives a formal intuition for selecting k randomly.

3.1.1. Binary Mask Creation

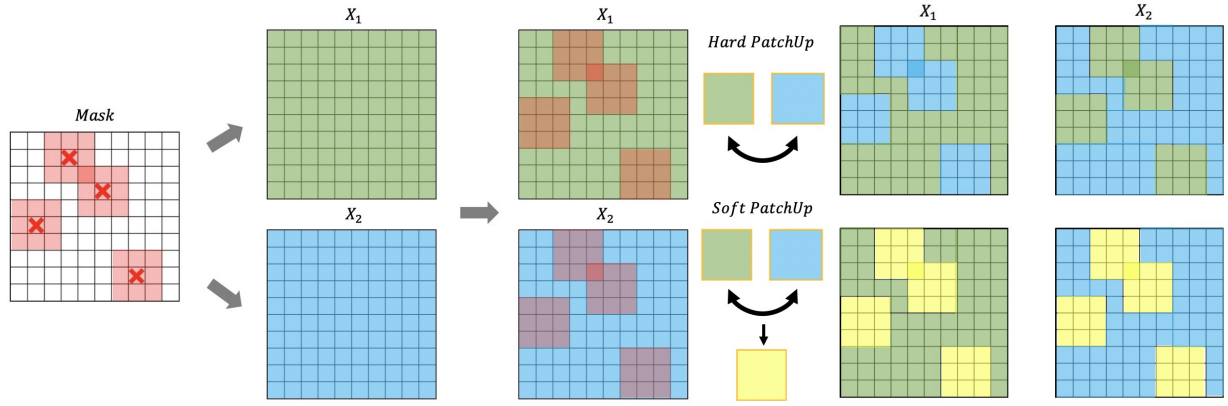


Fig. 3.1. *PatchUp* process for two hidden representations associated with two samples randomly selected in the mini-batch (a, b). $X_1 = g_k^{(i)}(a)$ and $X_2 = g_k^{(i)}(b)$ where i is the feature map index. Right top shows *Hard PatchUp* output and the right bottom shows the interpolated samples with *Soft PatchUp*. The yellow continuous blocks represent the interpolated selected blocks.

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 γ which decides the probability of altering a feature. When we alter a feature, we also alter a square block of features centered around that feature which is controlled by the side length of this square block, *block_size*. Hence, the altering probabilities are readjusted using the following formula [13]:

$$\gamma_{adj} = \frac{\gamma \times \textit{(feature map's area)}}{\textit{(block's area)} \times \textit{(valid region to build block)}}. \tag{3.1.1}$$

where the area of the feature map and block are the $feat\_size^2$ and $block\_size^2$, respectively, and the valid region to build the block is $(feat\_size - block\_size + 1)^2$.

For each feature in the feature map, we sample from $Bernoulli(\gamma_{adj})$. If the result of this sampling for feature f_{ij} is 0, then $M_{ij} = 1$. If the result of this sampling for f_{ij} is 1, then the entire square region in the mask with the center M_{ij} and the width and height of the square of *block_size* is set to 0. Note that these feature blocks to be altered can overlap which will result in more complex block structures than just squares. The block structures created are called patches. Figure-3.1 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).

3.1.2. 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:

$$\phi_{\mathbf{hard}}(g_k(x_i), g_k(x_j)) = \mathbf{M} \odot g_k(x_i) + (\mathbf{1} - \mathbf{M}) \odot g_k(x_j), \tag{3.1.2}$$

where $\odot$ is known as the element-wise multiplication operation and $\mathbf{M}$ is the binary mask described in section 3.1.1.

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

$$\text{Mix}_\lambda(a, b) = \lambda \cdot a + (1 - \lambda) \cdot b, \tag{3.1.3}$$

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

$$\phi_{\mathbf{soft}}(g_k(x_i), g_k(x_j)) = \mathbf{M} \odot g_k(x_i) + \text{Mix}_\lambda[((\mathbf{1} - \mathbf{M}) \odot g_k(x_i)), ((\mathbf{1} - \mathbf{M}) \odot g_k(x_j))], \quad (3.1.4)$$

where λ in the range of [0, 1] is sampled from a Beta distribution such that $\lambda \sim Beta(\alpha, \alpha)$. α controls the shape of the Beta distribution. Consequently, it controls the strength of interpolation [76]. Both *PatchUp* operations are illustrated in Figure 3.1.

3.1.3. 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:

$$\begin{aligned} L(f) = \mathbb{E}_{(x_i,\, y_i)\sim P}\ \mathbb{E}_{(x_j,\, y_j)\sim P}\ \mathbb{E}_{\lambda\sim \text{Beta}(\alpha,\, \alpha)}\, \mathbb{E}_{k\sim \mathcal{S}}\, \text{Mix}_{p_u}[\ell(f_k(\phi_k),\, y_i)\ , \ell(f_k(\phi_k),\, Y)] \\ + \ell(f_k(\phi_k),\, W(y_i,\, y_j)), \end{aligned} \quad (3.1.5)$$

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

Y is the target corresponding to the changed features. In the case of *Hard PatchUp*, $Y = y_j$ and in the case of *Soft PatchUp*, $Y = Mix_\lambda(y_i, y_j)$. $W(y_i, y_j)$ calculates the 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:

$$W_{hard}(y_i, y_j) = \text{Mix}_{p_u}(y_i, y_j), \quad (3.1.6)$$

$$W_{soft}(y_i, y_j) = \text{Mix}_{p_u}(y_i,\, \text{Mix}_\lambda(y_i, y_j)). \quad (3.1.7)$$

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.

3.1.4. Algorithm

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,

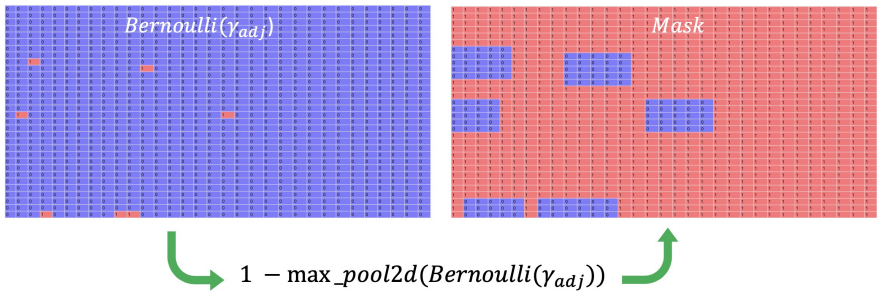
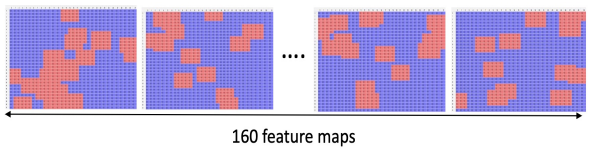
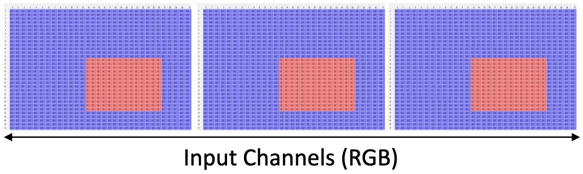


Fig. 3.2. *PatchUp* mask creation process ($block\_size = 5$). The left matrix shows the process of feature selection from feature maps. By using a max_pool2d function, we can create blocks around selected features. The max_pool2d function uses $stride = (1, 1)$, $kernel\_size = (block\_size, block\_size)$, and $padding = (\frac{block\_size}{2}, \frac{block\_size}{2})$. Red and blue points are 1 and 0 in the generated binary mask, respectively.

and $block\_size$. Algorithm-5 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*. Figure 3.2 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 3.3 compares the masks generated by *PatchUp* and CutMix.



(a) Mask sampling in *PatchUp*.



(b) Mask sampling in CutMix.

Fig. 3.3. 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.

3.1.5. 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 [73], 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 larger number of channels (more details in section 4.9).

Algorithm 5 ***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.

if $mode == Inference$ **then**
  return $(g_k(x_i), y_i)$, $(g_k(x_j), y_j)$
end if
$kernel\_size \leftarrow (block\_size, block\_size)$
$stride \leftarrow (1, 1)$
$padding \leftarrow (\frac{block\_size}{2}, \frac{block\_size}{2})$
$\gamma_{adj} \leftarrow$ adjust γ using (3.1.1)
$holes \leftarrow max\_pool2d(Bernoulli(\gamma_{adj}), kernel\_size, stride, padding)$
$Mask \leftarrow 1 - holes$
$unchanged \leftarrow Mask \odot g_k(x_i)$
$p_u \leftarrow$ calculate the portion of changed features map.
$Patch_i \leftarrow holes \odot g_k(x_i)$
$Patch_j \leftarrow holes \odot g_k(x_j)$
if $mixing\_type == hard$ **then**
  $Patch_i \leftarrow Patch_j$
  $Y \leftarrow y_j$
  $W \leftarrow W_{hard}(y_i, y_j)$ using (3.1.7)
else if $mixing\_type == soft$ **then**
  $\lambda \sim Beta(\alpha, \alpha)$
  $Y \leftarrow Mix_\lambda(y_i, y_j)$
  $W \leftarrow W_{soft}(y_i, y_j)$ using (3.1.7)
  $Patch_i \leftarrow Mix_\lambda(Patch_i, Patch_j)$
end if
$H' \leftarrow unchanged + Patch_i$
return y_i, y_j, H', p_u, Y, W

3.2. Relation to Other Methods

3.2.1. *PatchUp* Vs. ManifoldMixup:

Both *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 3.1.3. *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 figure 3.4. It presents the possible combinations of hidden representations explored via ManifoldMixup and *PatchUp*. Blue dots represent real hidden representation samples. ManifoldMixup can produce new samples that lie directly on the orange lines which connect the blue point pairs due to its linear interpolation strategy. But, *PatchUp* can select various points in all dimensions, and can also select points extremely close to the orange lines. The proximity to the orange lines depends on the selected pairs and λ sampled from the beta distribution. Figure 3.4 is a simple diagrammatic description of how PatchUp constructs more diverse samples. Section-4.8 provides a mathematical and real experimental justification for this discussion.

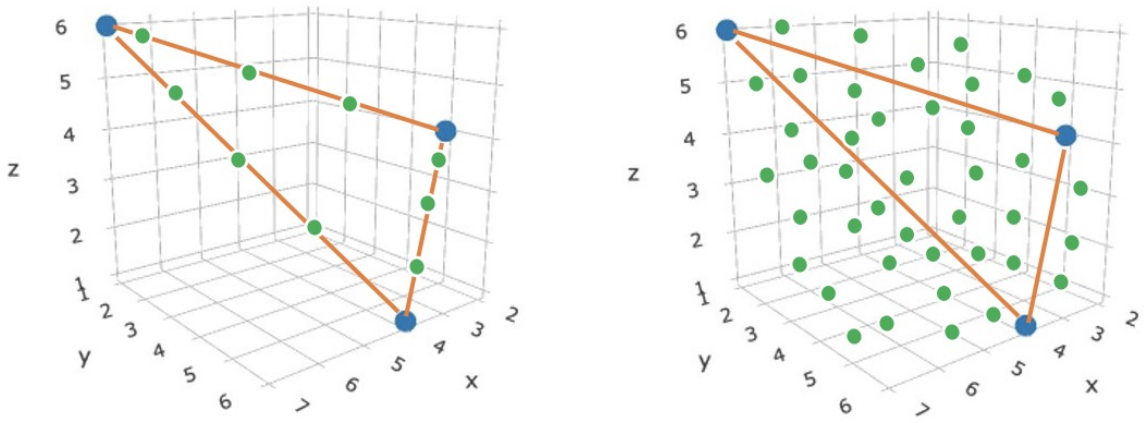


Fig. 3.4. Left: ManifoldMixup interpolated samples for any combination of the three blue hidden states selected only from along orange line. Right: *PatchUp* can produce interpolated hidden representations for these three hidden states in almost all possible places in all dimensions except the samples which lie directly on the orange lines.

3.2.2. *PatchUp* Vs. CutMix:

The CutMix cuts and fills the rectangular parts of the randomly selected pairs instead of using interpolation for creating a new sample in the input space. Therefore, CutMix has less potential for a manifold intrusion problem, however, CutMix may still suffer from a manifold intrusion problem. Figure 3.5 shows two samples with small portions that correspond to their labels. In this example, if only the parts within the yellow bounding boxes are swapped, then the label does not change. However, if the parts within the white bounding boxes are swapped, then the entire label is swapped. In both scenarios, CutMix only learns the interpolated target based on the fractions of the swapped part. In contrast, these scenarios are is less likely to occur in *PatchUp* since it works in the hidden representation space most of the time. Another difference between CutMix and *PatchUp* is how the masks are created. *PatchUp* can create arbitrarily shaped masks while CutMix masks can only be rectangular. Figure 3.3 shows an example of CutMix Mask and *PatchUp* mask in input space and hidden representation space, respectively. CutMix is more effective than Feature-CutMix that applies CutMix in the latent space [73]. And, both the learning objective of *PatchUp*, as well as the binary mask selection are different from Feature-CutMix.

Fig. 3.5. The two possible block selections from CutMix for two samples (cat and dog) with a large background. Swapping a similar part of the background or an essential element correlated to the label in the selected images can have a negative effect on the CutMix learning objective.

3.3. Conclusion

As we discussed in previous sections, most state-of-the-art data-dependent regularization techniques can cause manifold intrusion problems, particularly the approaches applied in input space such as Mixup and CutMix. ManifoldMixup tried to avoid the manifold intrusion problem by applying the regularization at latent space. ManifoldMixup also attempted to push the decision boundary away by exploring linear interpolation of hidden representations at randomly selected layers. Since ManifoldMixup interpolates the whole part of two hidden representations, the limited representations that can regularize the model are explored. To improve the CNN models' performance and their generalization on unseen data, this chapter introduced Patch up as a feature-space block-level regularization technique for the CNN models. Our experiments in the next chapter show that it can improve the CNN models' performance and provide better robustness against adversarial attacks and better generalization on a dataset that has a small shift from the training set.

Chapter 4

Experiments

In this chapter, we present the results[1] of applying *PatchUp* to image classification tasks using various benchmark datasets such as CIFAR-10, CIFAR-100 [28], SVHN (the standard version with 73257 training samples) [41], Tiny-ImageNet [6] datasets, and with various benchmark architectures such as PreActResNet18, PreActResNet34 [18], and ResNet101, ResNet152, and WideResNet-28-10 [74] models.

4.1. Data

CIFAR-10: The CIFAR-10 dataset [28] has been used to evaluate image classification tasks since 2009. It consists of 60000 color samples divided into 50000 training and 10000 test samples in size of 32×32 for 10 classes. CIFAR-10 dataset contains 6000 samples per class.

CIFAR-100: Similar to CIFAR-10, CIFAR-100 [28] dataset also has 60,000 colour images of size 32×32 but the samples grouped into 100 classes, each class containing 500 training, 100 testing images and requires more fine-grained classification.

SVHN: The Street View House Numbers (SVHN) dataset [41] contains a total of 630,420 colour images with a resolution of 32×32 pixels. The dataset consists of 73,257 training images and 26,032 test images that images are obtained from house numbers in Google Street View images.

Tiny-ImageNet: The Tiny ImageNet dataset [6] is a modified subset of the original ImageNet dataset [48] that contains 200 classes of ImageNet dataset, with 100,000 training samples and 10,000 testing examples. We reserved 10,000 samples of the training set as a validation set. The resolution of the images in the Tiny-ImageNet dataset is 64x64 pixels that make it more challenging to extract information from samples. Having a lower resolution in samples makes the classification task much more challenging [6].

[1]The code to reproduce all the results is available at https://github.com/chandar-lab/PatchUp.

4.2. Experiment Setup and Hyper-parameter Tuning

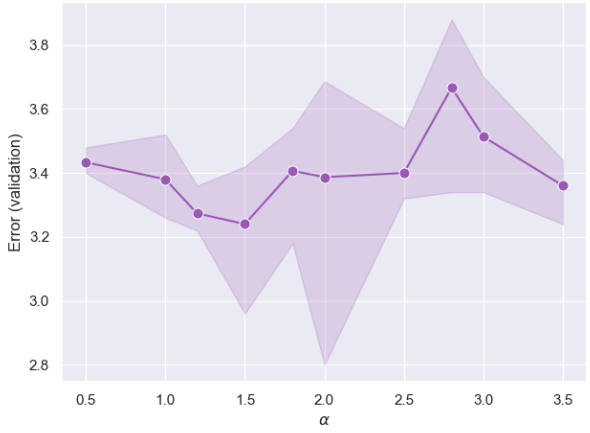
We follow ManifoldMixup in our experimental setup. We use SGD with 0.9 Nesterov momentum as an optimizer, mini-batch of 100, and weight decay of 1e-4 [65]. This section describes the hyper-parameters of each model in table 4.1 following the hyper-parameter setup from ManifoldMixup [65] experiments to 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.

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

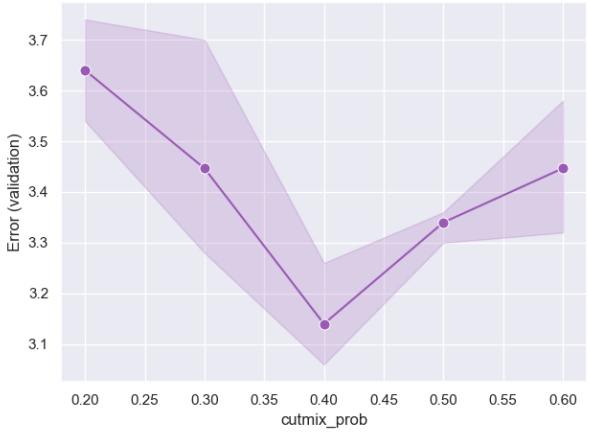
Table 4.1. The hyper-parameters used for each model to compare the effect of each regularization technique. The learning rate is denoted as *lr*. *lr* is multiplied at each learning rate schedule step by the step factor.

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 3.2. 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* from 3 to 9. We found that γ of 0.75 and *block_size* of 7 work best for *Soft PatchUp* as shown in figure 4.1c. 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 4.1d. Figure 4.1a shows that ManifoldMixup with ($\alpha = 1.5$) achieves the best validation performance. For cutout, we used the same hyper-parameters proposed in [9], setting cutout to 16 for CIFAR10, 8 for CIFAR100, and 20 for SVHN following [9]. Figure 4.1b shows that CutMix achieves

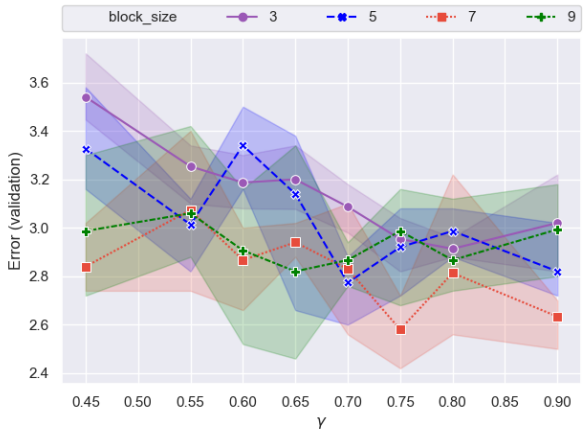
its best validation performance in PreActResNet18 in CIFAR-10 with *cutmix_prob* = 0.4. Furthermore, DropBlock achieves its best validation performance on this task by setting the block size and γ to 7 and 0.9, respectively [13].



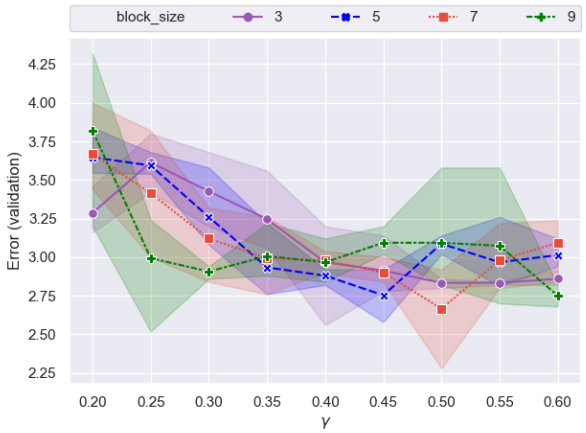
(a) Impact of α in ManifoldMixup approach.



(b) *cutmix_prob*'s impact on CutMix.



(c) Impact of γ, *block_size* with *patchup_prob* as 1.0 for *Soft PatchUp*.



(d) Impact of γ, *block_size* with *patchup_prob* as 0.7 for *Hard PatchUp*.

Fig. 4.1. Impact of hyper-parameters γ, *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 4.2. Image classification error rates on Tiny-ImageNet. We run experiments five times to report the mean and the standard deviation. Best performance result is shown in bold, second best is underlined. The lower number is better.

	ResNet101		ResNet152		WideResNet-28-10	
	Error	Loss	Error	Loss	Eror	Loss
No Mixup	46.184 ± 0.495	2.369 ± 0.027	45.222 ± 0.321	2.280 ± 0.083	36.756 ± 0.245	1.885 ± 0.023
Input Mixup ($\alpha = 1$)	44.930 ± 0.338	2.252 ± 0.044	44.645 ± 0.113	2.233 ± 0.040	36.504 ± 0.286	1.876 ± 0.018
ManifoldMixup (α= 2)	44.538 ± 0.139	2.247 ± 0.011	44.128 ± 0.313	2.231 ± 0.022	35.964 ± 0.644	1.829 ± 0.064
Cutout	45.782 ± 0.346	2.349 ± 0.012	45.234 ± 0.193	2.336 ± 0.017	35.850 ± 0.114	1.834 ± 0.007
DropBlock	46.978 ± 0.531	2.273 ± 0.095	45.818 ± 0.220	2.171 ± 0.007	36.972 ± 0.378	1.893 ± 0.015
CutMix	42.042 ± 0.376	2.108 ± 0.012	41.714 ± 0.711	2.084 ± 0.029	35.812 ± 0.180	1.748 ± 0.079
Puzzle Mix	42.250 ± 0.193	2.132 ± 0.011	42.126 ± 0.306	2.125 ± 0.029	33.428 ± 0.216	1.651 ± 0.050
Soft PatchUp	$\mathbf{38.676 \pm 0.340}$	1.865 ± 0.007	$\mathbf{38.112 \pm 0.287}$	1.840 ± 0.016	$\mathbf{29.812 \pm 0.240}$	1.459 ± 0.042
Hard PatchUp	$\underline{40.552 \pm 0.151}$	1.938 ± 0.007	$\underline{40.314 \pm 0.175}$	1.927 ± 0.009	$\underline{33.116 \pm 0.113}$	1.569 ± 0.053

Table 4.3. Error rates comparison on SVHN. We run experiments five times to report the mean and the standard deviation. 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	Eror	Loss
No Mixup	3.035 ± 0.092	0.138 ± 0.004	3.087 ± 0.659	0.164 ± 0.008	2.833 ± 0.081	0.137 ± 0.008
Input Mixup ($\alpha = 1$)	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= 2)	$\underline{2.436 \pm 0.056}$	0.157 ± 0.062	$\underline{2.423 \pm 0.428}$	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	$\mathbf{2.081 \pm 0.066}$	0.111 ± 0.010
Hard PatchUp	$\mathbf{2.286 \pm 0.084}$	0.107 ± 0.004	$\mathbf{2.123 \pm 0.024}$	0.101 ± 0.007	$\underline{2.088 \pm 0.061}$	0.105 ± 0.012

4.3. Generalization on Image Classification

Table 4.4 shows the comparison of the generalization performance of *PatchUp* with six recently proposed mixing based or feature-level regularization methods on the CIFAR-10 and CIFAR-100 datasets. Since Puzzle Mix clearly showed that both CutMix and Puzzle Mix perform better than AugMix [27], we excluded it from our experiments. Our experiments show that *PatchUp* leads to a lower test error for all the models on CIFAR (Table 4.4), SVHN (Table 4.3), and Tiny-ImageNet (Table 4.2) with a large margin. Specifically, *Soft PatchUp* outperforms other methods on Tiny-ImageNet dataset using ResNet101/152, and

WideResNet-28-10 followed by *Hard PatchUp*. As explained in Section-4.8 and shown in Figure 4.8, both *Soft* and *Hard PatchUp* produce a wide variety of interpolated hidden representations towards different dimensions. However, *Soft PatchUp* behaves more conservatively that helps outperform other methods with a large margin in the case of a limited number of training samples per class and having more targets.

Table 4.4. Image classification task error rates on CIFAR-10 and CIFAR-100. We run experiments five times to report the mean and the standard deviation. Best performance result is shown in bold, second best is underlined. The lower number is better.

PreActResNet18	Test Error (%)	Test NLL
No Mixup	4.800 ± 0.135	0.184 ± 0.004
Input Mixup ($\alpha = 1$)	3.628 ± 0.201	0.192 ± 0.012
ManifoldMixup ($\alpha = 1.5$)	3.388 ± 0.048	0.147 ± 0.016
Cutout	4.218 ± 0.046	0.158 ± 0.005
DropBlock	5.038 ± 0.147	0.185 ± 0.005
CutMix	3.518 ± 0.898	0.131 ± 0.002
Puzzle Mix	3.155 ± 0.110	0.119 ± 0.004
Soft PatchUp	$\underline{2.956 \pm 0.119}$	0.169 ± 0.031
Hard PatchUp	$\mathbf{2.918 \pm 0.131}$	0.146 ± 0.718
PreActResNet34		
No Mixup	4.640 ± 0.099	0.204 ± 0.004
Input Mixup ($\alpha = 1$)	3.260 ± 0.075	0.175 ± 0.004
ManifoldMixup ($\alpha = 1.5$)	2.926 ± 0.062	0.124 ± 0.004
Cutout	3.690 ± 0.141	0.150 ± 0.012
DropBlock	4.950 ± 0.188	0.221 ± 0.010
CutMix	3.332 ± 0.071	0.142 ± 0.004
Puzzle Mix	2.996 ± 0.069	0.125 ± 0.006
Soft PatchUp	$\underline{2.570 \pm 0.062}$	0.108 ± 0.005
Hard PatchUp	$\mathbf{2.534 \pm 0.048}$	0.108 ± 0.005
WideResNet-28-10		
No Mixup	4.244 ± 0.142	0.162 ± 0.011
Input Mixup ($\alpha = 1$)	3.272 ± 0.353	0.191 ± 0.018
ManifoldMixup ($\alpha = 1.5$)	3.252 ± 0.183	0.155 ± 0.034
Cutout	3.134 ± 0.119	0.122 ± 0.005
DropBlock	4.182 ± 0.069	0.157 ± 0.003
CutMix	3.148 ± 0.118	0.126 ± 0.004
Puzzle Mix	$\underline{2.562 \pm 0.074}$	0.098 ± 0.002
Soft PatchUp	2.606 ± 0.052	0.132 ± 0.029
Hard PatchUp	$\mathbf{2.528 \pm 0.065}$	0.114 ± 0.014

Comparison on CIFAR-10

PreActResNet18	Test Error (%)	Test NLL
No Mixup	24.622 ± 0.358	1.062 ± 0.017
Input Mixup ($\alpha = 1$)	22.326 ± 0.323	1.011 ± 0.012
ManifoldMixup ($\alpha = 1.5$)	21.396 ± 0.384	0.931 ± 0.008
Cutout	23.386 ± 0.185	1.004 ± 0.004
DropBlock	25.022 ± 0.259	1.067 ± 0.016
CutMix	22.184 ± 0.176	0.949 ± 0.012
Puzzle Mix	20.649 ± 0.214	0.857 ± 0.013
Soft PatchUp	$\underline{19.950 \pm 0.180}$	0.833 ± 0.005
Hard PatchUp	$\mathbf{19.120 \pm 0.172}$	0.748 ± 0.013
PreActResNet34		
No Mixup	23.342 ± 0.269	1.103 ± 0.006
Input Mixup ($\alpha = 1$)	21.000 ± 0.440	0.950 ± 0.019
ManifoldMixup ($\alpha = 1.5$)	18.724 ± 0.305	0.810 ± 0.008
Cutout	22.420 ± 0.075	1.043 ± 0.001
DropBlock	23.744 ± 0.125	1.113 ± 0.007
CutMix	19.944 ± 0.141	0.907 ± 0.008
Puzzle Mix	19.974 ± 0.225	0.893 ± 0.022
Soft PatchUp	$\underline{18.630 \pm 0.153}$	0.816 ± 0.016
Hard PatchUp	$\mathbf{17.692 \pm 0.125}$	0.758 ± 0.016
WideResNet-28-10		
No Mixup	22.442 ± 0.226	1.065 ± 0.010
Input Mixup ($\alpha = 1$)	18.726 ± 0.149	0.854 ± 0.013
ManifoldMixup ($\alpha = 1.5$)	18.352 ± 0.378	0.833 ± 0.023
Cutout	20.164 ± 0.351	0.931 ± 0.016
DropBlock	22.364 ± 0.149	1.049 ± 0.013
CutMix	18.316 ± 0.185	0.839 ± 0.020
Puzzle Mix	17.528 ± 0.224	0.757 ± 0.006
Soft PatchUp	$\underline{16.726 \pm 0.110}$	0.722 ± 0.017
Hard PatchUp	$\mathbf{16.134 \pm 0.197}$	0.660 ± 0.017

Comparison on CIFAR-100

Hard PatchUp provides the best performance in the CIFAR and *Soft PatchUp* achieves the second-best performance except on the CIFAR-10 with WideResNet-28-10 where Puzzle Mix provides the second-best performance. In the SVHN dataset ManifoldMixup achieves

the second-best performance in PreActResNet18 and PreActResNet34 where *Hard PatchUp* provide the lowest top-1 error. *Soft PatchUp* performs reasonably well and comparable to ManifoldMixup for PreActResNet34 on SVHN and leads to a lower test error followed by *Hard PatchUp* for WideResNet-28-10 in the SVHN dataset. We observe that the Mixup, ManifoldMixup, and Puzzle Mix are sensitive to the α when we have more training classes. We tried to fix hyper-parameter for all experiments. Notably, using the same α used in CIFAR or SVHN leads to worse performance than No-Mixup in Tiny-ImageNet, where others are almost stable. Since ManifoldMixup and Puzzle Mix show that they perform better than No-Mixup and Input Mixup on affine transformation and against adversarial attacks [27, 65], we exclude experiments on No-Mixup and Input Mixup for the tasks in the following sections.

4.4. Robustness to Common Corruptions

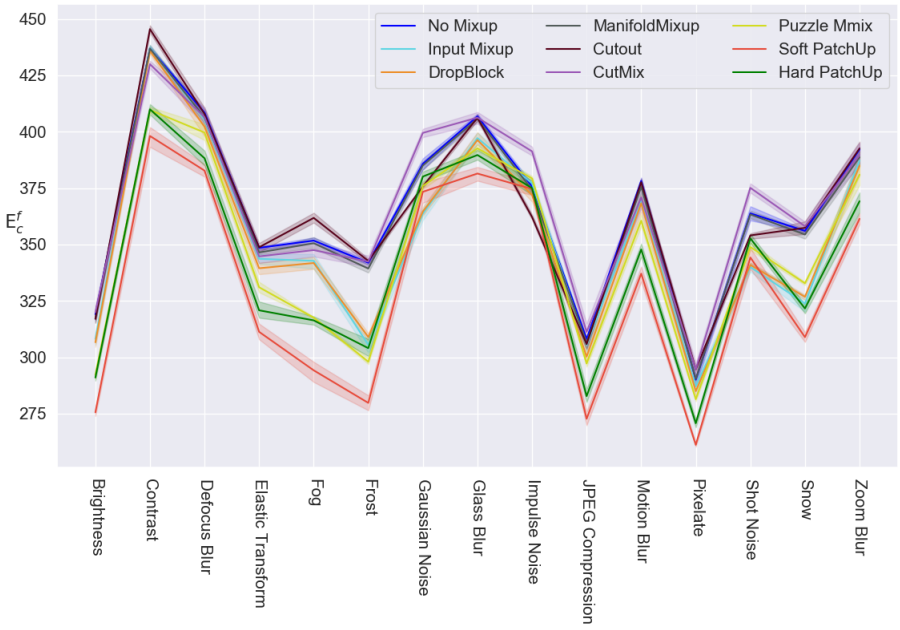
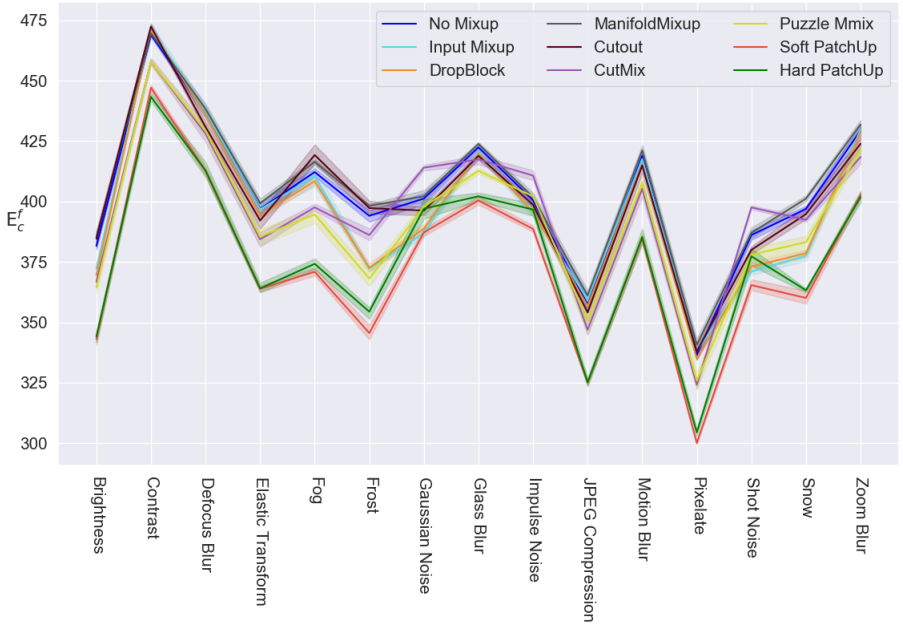
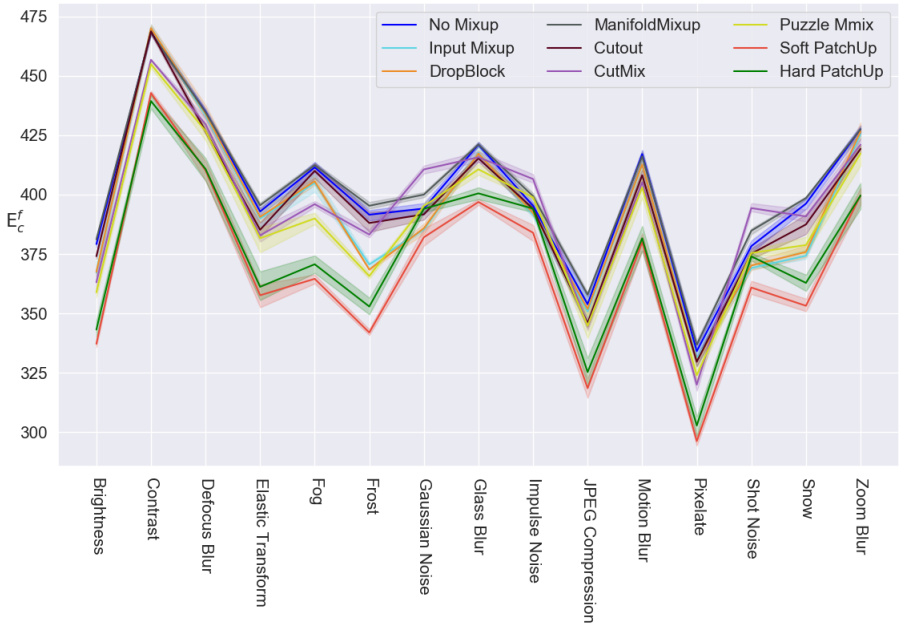


Fig. 4.2. Robustness of WideResNet-28-10 to Tiny-ImageNet Common Corruptions (Tiny-ImageNet-C). We repeated each test for five trained models to report the mean and the standard. The lower values on the y-axes show the robustness of the model against the input common corruptions. The y-axis is the sum of error rates for each category. And, the x-axis represents the list of corruptions in Tiny-ImageNet-C.



(a) Comparison on ResNet101.



(b) Comparison on on ResNet152.

Fig. 4.3. 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.

The common corruption benchmark helps to evaluate the robustness of models against the input corruptions [19]. It uses the 75 corruptions in 15 categories such that each category has five levels of severity. We compare the methods robustness in Tiny-ImageNet-C dataset for ResNet101, ResNet152, and WideResNet-28-10 models. So, we compute the sum of error rates denoted as E_c^f where s is the level of severity and c is corruption type such that $\mathrm{E}_c^f = \sum_{s=1}^{5} E_{s,c}^f$ [19]. Figure 4.2 shows *Soft PatchUp* leads the best performance in Tiny-ImageNet-C and *Hard PatchUp* achieves the second-best performance in the WideResNet-28-10. Figures 4.3a and 4.3b show the comparison results in ResNet101, and ResNet152.

4.5. Generalization on Deformed Images

Affine transformations on the test set provide novel deformed data samples that can be used to evaluate the robustness of a method on out-of-distribution samples [65]. We trained PreActResNet34 and WideResNet-28-10 on the CIFAR100 dataset. Then, we created deformed test sets from CIFAR100 by applying some affine transformations. Table 4.6 shows that *PatchUp* provides the best performance on affine transformed test sets and better generalization in PreActResNet34 (The lower number is better).

Table 4.5. 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.

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	$\mathbf{31.282 \pm 0.622}$	$\underline{31.340 \pm 0.318}$
Rotate (-40, 40)	57.220 ± 0.549	56.512 ± 0.752	56.610 ± 0.877	51.49 ± 1.028	$\mathbf{52.014 \pm 0.916}$	$\underline{53.804 \pm 0.576}$
Shear (-28.6, 28.6)	33.482 ± 0.463	31.770 ± 0.312	32.300 ± 0.317	27.178 ± 0.256	$\underline{30.898 \pm 0.836}$	$\mathbf{28.426 \pm 0.430}$
Shear (-57.3, 57.3)	53.328 ± 0.587	$\underline{50.618 \pm 0.552}$	52.366 ± 0.170	47.414 ± 0.482	51.908 ± 0.632	$\mathbf{48.334 \pm 0.631}$
Scale (0.6)	56.770 ± 0.376	$\mathbf{45.980 \pm 0.404}$	63.924 ± 2.160	46.354 ± 0.869	52.648 ± 0.616	$\underline{46.924 \pm 1.035}$
Scale (0.8)	30.550 ± 0.611	$\underline{26.818 \pm 0.328}$	29.012 ± 0.372	24.428 ± 0.171	27.188 ± 0.597	$\mathbf{23.840 \pm 0.535}$
Scale (1.2)	47.268 ± 0.639	51.258 ± 0.817	$\mathbf{41.644 \pm 0.846}$	49.18 ± 1.046	$\underline{42.108 \pm 0.985}$	43.370 ± 1.223
Scale (1.4)	79.000 ± 0.933	82.562 ± 0.575	$\underline{72.752 \pm 0.846}$	82.152 ± 0.41	$\mathbf{70.970 \pm 1.433}$	77.370 ± 1.457

Table 4.5 illustrates that the quality of representations is improved by *PatchUp* and it also shows better generalization in deformed test sets on WideResNet-28-10 (The lower number is better).

Table 4.6. Error rates in the test set on samples subject to affine transformations for PreActResNet34 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.

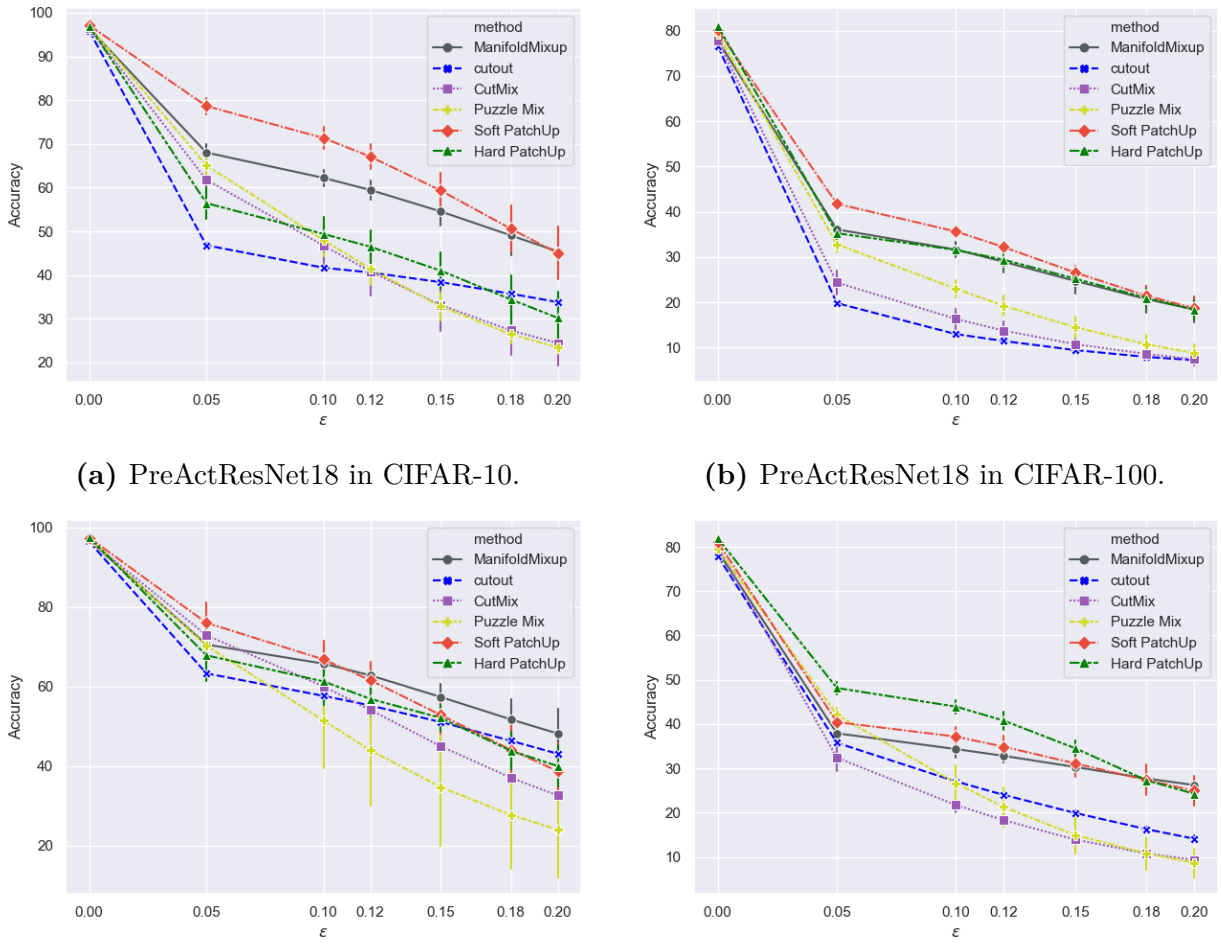
Transformation	cutout	CutMix	ManifoldMixup	Puzzle Mix	*Soft PatchUp*	*Hard PatchUp*
Rotate (-20, 20)	37.448 ± 0.526	35.418 ± 0.328	35.444 ± 0.572	31.698 ± 0.664	$\underline{31.136 \pm 0.524}$	$\mathbf{30.406 \pm 0.520}$
Rotate (-40, 40)	58.752 ± 0.995	57.830 ± 0.586	54.424 ± 0.946	54.042 ± 0.800	$\underline{53.422 \pm 0.420}$	$\mathbf{49.956 \pm 0.798}$
Shear (-28.6, 28.6)	36.552 ± 0.487	34.148 ± 0.473	34.150 ± 0.416	31.628 ± 0.688	$\mathbf{28.984 \pm 0.497}$	$\underline{29.574 \pm 0.410}$
Shear (-57.3, 57.3)	57.736 ± 0.574	53.640 ± 0.587	55.444 ± 0.683	52.492 ± 0.390	$\mathbf{49.102 \pm 0.532}$	$\underline{50.318 \pm 0.616}$
Scale (0.6)	72.994 ± 1.231	54.304 ± 1.268	78.998 ± 1.126	59.696 ± 1.242	$\mathbf{46.246 \pm 1.204}$	$\underline{50.062 \pm 2.692}$
Scale (0.8)	35.092 ± 0.857	29.380 ± 0.577	34.624 ± 0.370	30.366 ± 0.359	$\mathbf{23.942 \pm 0.212}$	$\underline{25.338 \pm 0.328}$
Scale (1.2)	42.310 ± 0.706	49.522 ± 2.035	$\underline{41.322 \pm 0.638}$	47.38 ± 1.443	43.414 ± 0.652	$\mathbf{38.002 \pm 0.703}$
Scale (1.4)	69.404 ± 0.901	78.664 ± 1.854	$\mathbf{65.938 \pm 0.751}$	77.586 ± 0.967	77.068 ± 1.189	$\underline{66.338 \pm 1.219}$

4.6. Robustness to Adversarial Examples

Neural networks trained with Empirical Risk Minimization (ERM) are often vulnerable to adversarial examples [60, 76]. 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 [65, 73, 76].

Table 4.7. Robust Accuracy of WideResNet-28-10 in the Tiny-ImageNet dataset against adversarial 7-steps attacks with $\epsilon = \frac{16}{255}$. Best performance result is shown in bold, second best is underlined. The higher number is better (repeated five times).

Methods	DeepFool	DDN	CW	PGD_{L_∞}
No Mixup	0.171 ± 0.007	0.177 ± 0.005	0.177 ± 0.005	0.171 ± 0.004
Input Mixup	$\underline{0.193 \pm 0.002}$	0.195 ± 0.001	0.195 ± 0.001	0.190 ± 0.001
ManifoldMixup	$\mathbf{0.197 \pm 0.003}$	$\mathbf{0.198 \pm 0.002}$	$\underline{0.197 \pm 0.002}$	$\underline{0.191 \pm 0.002}$
Cutout	0.183 ± 0.007	0.185 ± 0.005	0.185 ± 0.005	0.180 ± 0.005
DropBlock	0.181 ± 0.004	0.186 ± 0.001	0.186 ± 0.001	0.181 ± 0.002
CutMix	0.168 ± 0.004	0.171 ± 0.003	0.172 ± 0.003	0.171 ± 0.003
Puzzle Mix	0.187 ± 0.002	0.191 ± 0.001	0.191 ± 0.002	0.188 ± 0.001
Soft PatchUp	0.187 ± 0.002	$\underline{0.196 \pm 0.002}$	$\mathbf{0.198 \pm 0.002}$	$\mathbf{0.192 \pm 0.002}$
Hard PatchUp	0.179 ± 0.004	0.186 ± 0.003	0.186 ± 0.003	0.182 ± 0.004



(a) PreActResNet18 in CIFAR-10.

(b) PreActResNet18 in CIFAR-100.

(c) PreActResNet34 in CIFAR-10.

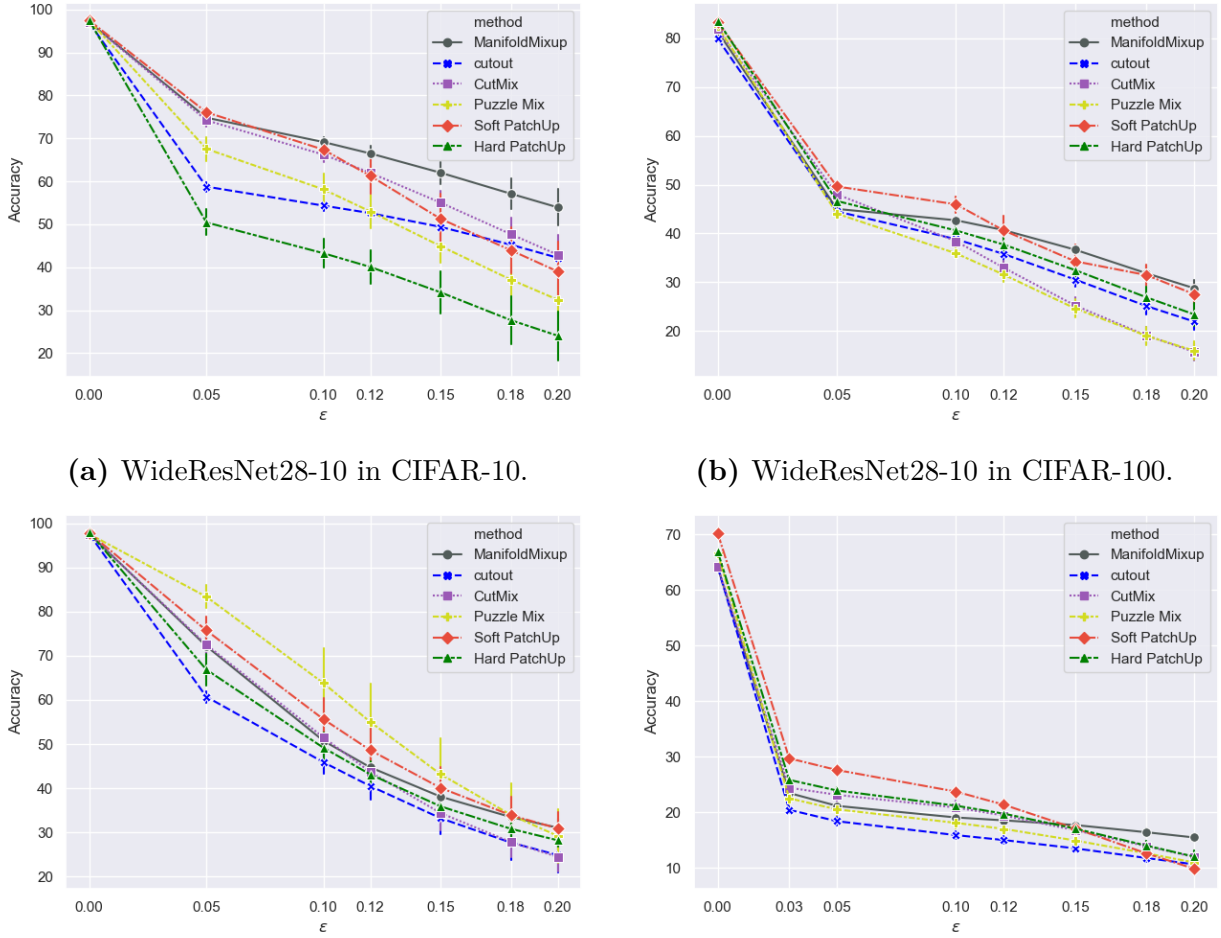
(d) PreActResNet34 in CIFAR-100.

Fig. 4.4. Robustness of PreActResNet18 and PreActResNet34 models in CIFAR-10 and CIFAR-100 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 each method's accuracy against the FGSM attack. The higher values on the y-axes show the robustness of the model against the attack. And, ϵ is the magnitude that controls the perturbation.

The adversarial attacks refer to small and unrecognizable perturbations on the input images that can mislead deep learning models [14, 15]. One approach to creating adversarial examples is using the Fast Gradient Sign Method (FGSM), also known as a white-box attack [15]. FGSM creates perturbed samples by adding small perturbations to the original samples. Once a regularized model is trained, then FSGM creates adversarial examples as follows [15]:

$$x' = x + \epsilon \times \text{sign}\left(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y)\right). \tag{4.6.1}$$

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



(a) WideResNet28-10 in CIFAR-10.

(b) WideResNet28-10 in CIFAR-100.

(c) WideResNet28-10 in SVHN.

(d) WideResNet28-10 in Tiny-ImageNet.

Fig. 4.5. Robustness of WideResNet28-10 model in CIFAR-10, CIFAR-100, SVHN, and Tiny-ImageNet 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 each method's accuracy against the FGSM attack. The higher values on the y-axes show the robustness of the model against the attack. And, ϵ is the magnitude that controls the perturbation.

Figures 4.4 and 4.5 shows the compared performance of the methods for PreActResNet18, PreActResNet34, and WideResNet-28-10 models in CIFAR-10, CIFAR-100, SVHN, and Tiny-ImageNet datasets against the FGSM attacks described in [15]. Table 4.7 also shows the robust accuracy (in the range of $[0, 1]$) for the Foolbox benchmark [44] against the

7-steps DeepFool [40], Decoupled Direction and Norm (DDN) [47], Carlini-Wagner (CW) [5], and PGD_{L_∞} [37] attacks with $\epsilon = \frac{8}{255}$.

We observe that *PatchUp* is more robust to adversarial attacks when compared to other regularization methods. While *Hard PatchUp* achieves better performance in terms of classification accuracy, *Soft PatchUp* seems to trade-off a slight loss of accuracy to achieve more robustness.

4.7. Effect on Activations

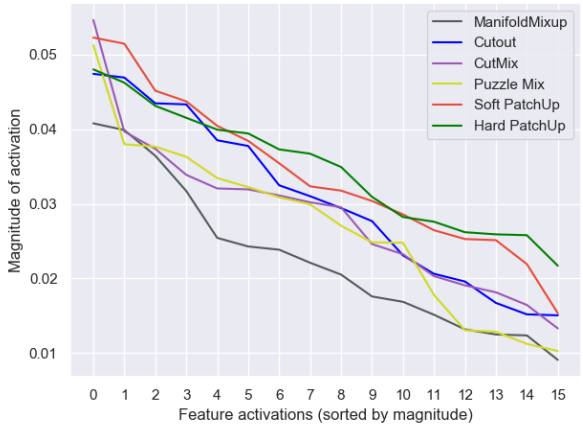
Cutout introduced an ablation study to compare the regularization techniques. Cutout compares the average magnitude of feature activations in different layers of the model. According to the Cutout, the higher average magnitude of feature activations shows that the model uses a wider variety of features when making predictions, rather than relying on the presence of a smaller number of features [9]. Having a wider variety of features can capture more detail from input images and provide rich representation.

However, our experiment shows that having a higher average magnitude of feature activations on a regularized model could be considered a strength for a model and, therefore, for the regularization method. Still, it is not sufficient to consider a regularization method as the best regularization technique. This section shows that *PatchUp* provides a higher average magnitude of feature activations in the layers that *PatchUp* are applied at training time.

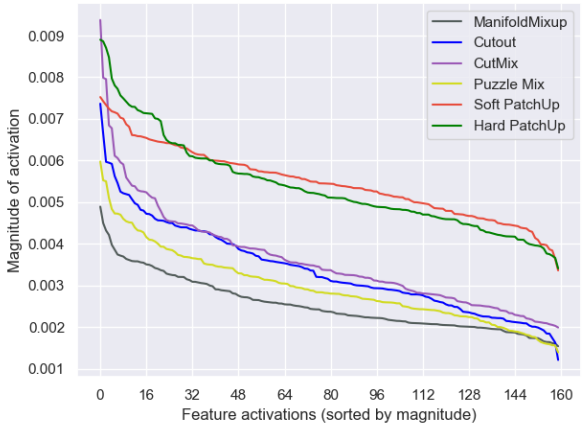
To study the effect of the methods on the activations in the residual blocks, we compared the mean magnitude of feature activations in the residual blocks following [9] in WideResNet-28-10 for CIFAR-100 test set. We first train the models with each method and then calculate the magnitudes of activations in the test set. And, the higher mean magnitude of features shows that the models tried to produce a wider variety of features in the residual blocks [9].

Our WideResNet-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, first conv2d, first or second residual blocks.

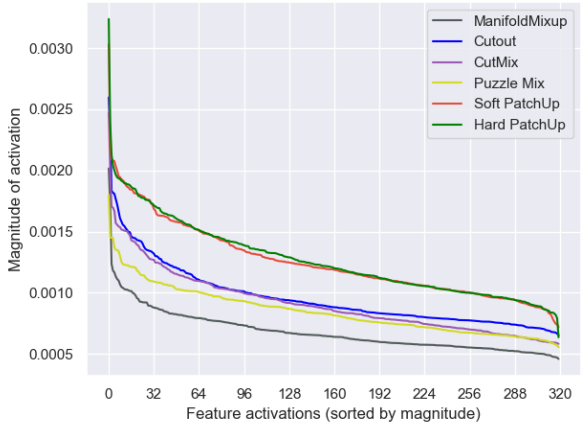
Figures 4.6a, 4.6b, and 4.6c illustrate that *PatchUp* produces more diverse features in the layers where we apply *PatchUp*. Since we are not applying the *PatchUp* in the third residual block, the mean magnitude of the feature activations are below, but very close to, Cutout and CutMix as shown in Figure 4.6d. 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 4.6



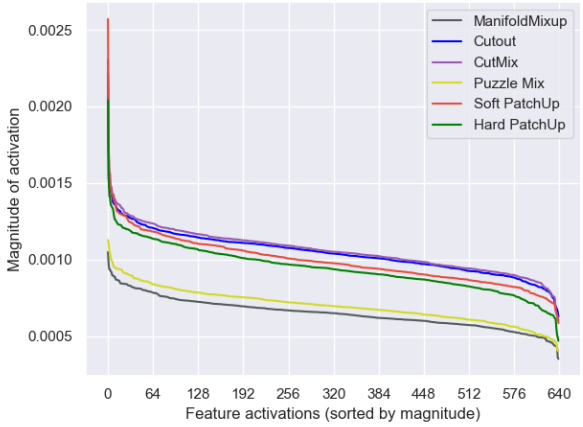
(a) Comparison on the first convolution module.



(b) Comparison on 1st Residual Block.



(c) Comparison on 2nd Residual Block.



(d) Comparison on 3rd Residual Block.

Fig. 4.6. 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.

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. In our implementation WideResNet28-10 has a conv2d module followed by three residual blocks. Figure 4.6 illustrates the comparison of ManifoldMixup, cutout, CutMix, *Soft PatchUp*, and *Hard PatchUp*.

4.8. *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 $\angle\theta$, $\angle\alpha_1$, and $\angle\alpha_2$, respectively. There is always a surface that contains $\mathcal{H}_2$ and $\mathcal{H}_1$ denoted as $\mathcal{S}$. Mathematically, we have:

$$\angle\alpha_1 = \cos^{-1}\left(\frac{\mathcal{H}_1 \cdot \mathcal{H}}{\|\mathcal{H}_1\|\|\mathcal{H}\|}\right) \;\&\; \angle\alpha_2 = \cos^{-1}\left(\frac{\mathcal{H}_2 \cdot \mathcal{H}}{\|\mathcal{H}_2\|\|\mathcal{H}\|}\right) \;\&\; \angle\theta = \cos^{-1}\left(\frac{\mathcal{H}_2 \cdot \mathcal{H}_1}{\|\mathcal{H}_2\|\|\mathcal{H}_1\|}\right), \tag{4.8.1}$$

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 4.7 illustrates three possible scenarios for two paired flattened 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 $\mathcal{H}_2$, $\mathcal{H}_1$, and $\mathcal{H}$ always lie in the same surface $\mathcal{S}$ and $\mathcal{H}$ lies between $\mathcal{H}_2$ and $\mathcal{H}_1$ in ManifoldMixup. This means that $\angle\rho = 0$ for ManifoldMixup because of its linear interpolation policy. The middle figure in 4.7 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 $\mathcal{S}$. Our goal is to produce the interpolated hidden representation that lies in all possible places towards all dimensions in the hidden space.

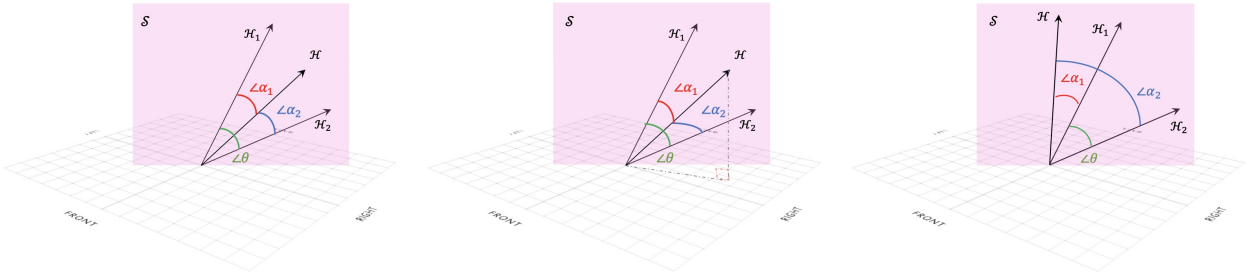


Fig. 4.7. $\mathcal{H}_1$ and $\mathcal{H}_2$ are the flattened hidden representations. $\mathcal{H}$ is the flattened interpolated hidden representation that can be produced by either ManifoldMixup, *Soft PatchUp* or *Hard PatchUp*.

As discussed in section 3.2.1, 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

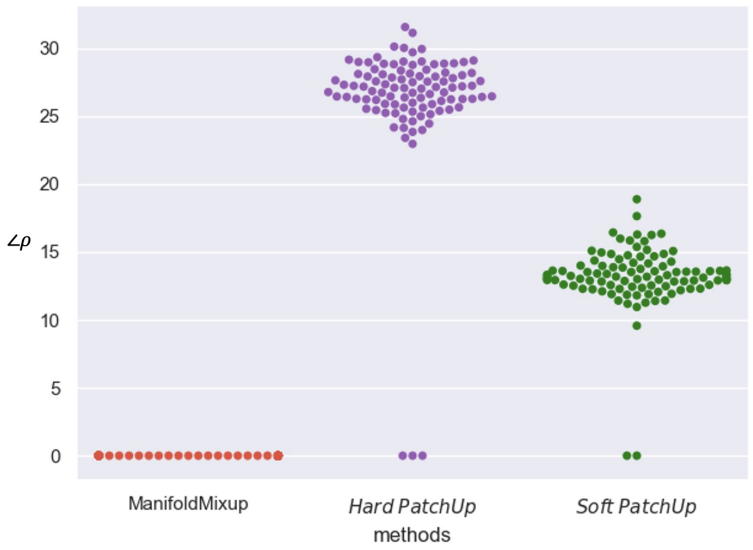


Fig. 4.8. The comparison of $\angle\rho$ for flattened hidden representations of a mini-batch of samples at the second residual block (layer $k = 3$) of WideResNet-28-10 with corresponding regularization method.

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 4.8 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.

4.9. Significance of loss terms

PatchUp uses the loss that is introduced in Equation 3.1.5. We can paraphrase the *PatchUp* learning objective for this ablation study as follow:

$$L(f) = \mathbb{E}_{(x_i,\, y_i)\sim P}\ \mathbb{E}_{(x_j,\, y_j)\sim P}\ \mathbb{E}_{\lambda\sim \mathrm{Beta}(\alpha,\, \alpha)}\, \mathbb{E}_{k\sim \mathcal{S}}(L_1 + L_2), \tag{4.9.1}$$

where $L_1 = \mathrm{Mix}_{p_u}[\ell(f_k(\phi_k), y_i), \ell(f_k(\phi_k), Y)]$ and $L_2 = \ell(f_k(\phi_k), W(y_i, y_j))$. We also show the effect of L_1 and L_2 in *PatchUp* loss. Table 4.8 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 *PatchUp*.

Table 4.8. The validation error rate on CIFAR-100 for WideResNet-28-10 with *Hard and Soft PatchUp*. The result is the mean and standard deviation of the experiment for five runs. A smaller number indicates better performance.

Simple WRN-28-10	Error Rate: 23.256 ± 0.586		
	Error rates with $L1$	Error rates with $L2$	Error rates with $L(f)$
Soft PatchUp	16.856 ± 0.666	16.865 ± 0.339	$\mathbf{16.75 \pm 0.291}$
Hard PatchUp	16.135 ± 0.229	16.79 ± 0.457	$\mathbf{16.02 \pm 0.358}$

4.10. Why random k?

PatchUp applies block-level regularization at a randomly selected hidden representation layer k. The Information Bottleneck (IB) principle [61] gives a formal intuition for selecting k randomly. First, let us encapsulate the network layers 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:

$$x \rightarrow g_1(x) \rightarrow g_2(x) \rightarrow g_3(x). \tag{4.10.1}$$

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

$$I(g_3(x);\ g_2(x)) < I(g_2(x);\ g_1(x)) < I(g_1(x);\ x), \tag{4.10.2}$$

where $I(g_k(x);\ g_{k-1}(x))$ is the mutual information between the k-th and $(k-1)$-th layer.

If $g_{k=3}(x)$ has enough information to represent x, then applying regularization techniques in $g_{k=3}(x)$ will provide a better generalization to unseen data. However, most current state-of-the-art CNN models contain residual connections that break the Markov chain described above (since information can skip the $g_{k=3}$ layer). One solution to this challenge is to randomly select a residual block and apply regularization techniques like ManifoldMixup or *PatchUp*.

We also conducted an experiment to show the importance of random layer selection in PatchUp. Table 4.9 shows the contribution of the random selection of the layer in the overall performance of the method. Worth mentioning that applying ManifoldMixup or *PatchUp* at every layer will also be considered a very strong regularization that hurts the model's performance. As Table 4.9 shows, if we apply the *PatchUp* only in the fixed layer it might

either overfit or twists the manifold. In the left-most column 1/2/3 refers to PatchUp being applied to only one layer (more details in Section-4.10). As noted in section 3.1.5, the PatchUp mask is "too strong" for the input space, which has only three channels. Figure 4.9 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 4.9 shows the negative effect of applying PatchUp mask in the input space.

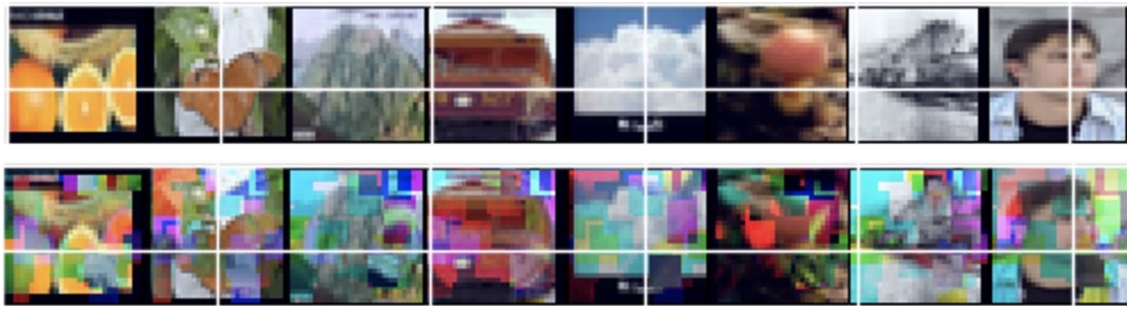


Fig. 4.9. (top) Original samples. (bottom) Hard PatchUp output using PatchUp Binary Mask on input images.

Table 4.9. WRN-28-10 using Hard PatchUp on CIFAR-100. repeated five times and reported the mean and std.

layer	Validation Error (%)	Test Error (%)	Test NLL
1	18.428 ± 0.441	17.864 ± 0.158	0734 ± 0.014
2	22.536 ± 0.799	21.418 ± 0.278	0.854 ± 0.008
3	26.172 ± 0.497	25.254 ± 0.139	1.138 ± 0.033
Random selection	$\mathbf{16.382 \pm 0.473}$	$\mathbf{16.134 \pm 0.197}$	$\mathbf{0.66 \pm 0.017}$
PatchUp Masks in k = 0	16.976 ± 0.342	$16.97 \pm 0.0.206$	0.671 ± 0.002

Chapter 5

Conclusion

We presented *PatchUp*, an efficient and straightforward regularizer scheme for CNNs that alleviates some of the drawbacks of the previous mixing-based regularizers. *PatchUp* exploits either mixing or swapping a masked out contiguous blocks of the feature map of a random pair of samples at randomly selected layers to avoid manifold intrusion problem. Like previous mixing-based approaches, our approach also has the advantage of preventing any added computational overhead.

We show that the *PatchUp* provides a comprehensive variety of interpolated hidden representations towards different dimensions and therefore a better generalization and robustness against adversarial attacks including FGSM, DDN, CW, and PGD_∞ attacks. *PatchUp* as a data-dependent regularization operates in the hidden space in a CNN model during training. Our experimental results show that with the proposed approach, *PatchUp*, we can achieve state-of-the-art results on image classification and deformed image classification tasks across different models with the ResNet architecture in CIFAR-10, CIFAR-100, SVHN, and Tiny-ImageNet datasets.

Our empirical results show that *PatchUp* as a data-dependent regularization method that enforces CNN models to produce minimal and more robust features. To this end, the strong test accuracy improvements achieved by *PatchUp*, with no additional computational overhead, makes it particularly appealing for practical applications.

5.1. Future Work

PatchUp selects the layer randomly in ResNet architecture, making sense for the ResNet architecture due to the skip connection. However, finding a more efficient and adaptive way to select an intermediate layer at each training step could be considered potential future work. Another line of work could be comparing PatchUp with other regularization techniques on models that use for Point Clouds object classification or segmentation. Point

Clouds data is a 3D ShapeNets used to detect objects using LiDAR technology. ModelNet40 [68] as a 3D Representation for Volumetric Shapes can be used as a benchmark to see the effect of PatchUp on CNN layers in PointNet [43], PCNN [3], and Dynamic Graph CNN (DGCNN) [66].

References

[1] A. Achille et S. Soatto : Information dropout: Learning optimal representations through noisy computation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 40(12):2897–2905, Dec 2018.

[2] 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.

[3] Matan Atzmon, Haggai Maron et Yaron Lipman : Point convolutional neural networks by extension operators, 2018.

[4] Henry S Baird, Horst Bunke et Kazuhiko Yamamoto : Structured document image analysis. Springer Science & Business Media, 2012.

[5] Nicholas Carlini et David Wagner : Towards evaluating the robustness of neural networks. In 2017 ieee symposium on security and privacy (sp), pages 39–57. IEEE, 2017.

[6] Patryk Chrabaszcz, Ilya Loshchilov et Frank Hutter : A downsampled variant of imagenet as an alternative to the cifar datasets, 2017.

[7] Michael Cogswell, Faruk Ahmed, Ross Girshick, Larry Zitnick et Dhruv Batra : Reducing overfitting in deep networks by decorrelating representations, 2016.

[8] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan et Quoc V. Le : Autoaugment: Learning augmentation policies from data, 2019.

[9] Terrance DeVries et Graham W. Taylor : Improved regularization of convolutional neural networks with cutout, 2017.

[10] Alexei A Efros et William T Freeman : Image quilting for texture synthesis and transfer. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pages 341–346, 2001.

[11] Yarin Gal et Zoubin Ghahramani : A theoretically grounded application of dropout in recurrent neural networks. In Advances in neural information processing systems, pages 1019–1027, 2016.

[12] Andrei Dmitri Gavrilov, Alex Jordache, Maya Vasdani et Jack Deng : Preventing model overfitting and underfitting in convolutional neural networks. International Journal of Software Science and Computational Intelligence (IJSSCI), 10(4):19–28, 2018.

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

[14] Ian Goodfellow, Yoshua Bengio et Aaron Courville : Deep learning. MIT press, 2016.

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

[16] Hongyu Guo, Yongyi Mao et Richong Zhang : Mixup as locally linear out-of-manifold regularization. CoRR, abs/1809.02499, 2018.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun : Deep residual learning for image recognition. CoRR, abs/1512.03385, 2015.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun : Identity mappings in deep residual networks, 2016.

[19] Dan Hendrycks et Thomas Dietterich : Benchmarking neural network robustness to common corruptions and perturbations, 2019.

[20] Dan Hendrycks, Norman Mu, Ekin D. Cubuk, Barret Zoph, Justin Gilmer et Balaji Lakshminarayanan : Augmix: A simple data processing method to improve robustness and uncertainty, 2020.

[21] Alex Hernández-García et Peter König : Data augmentation instead of explicit regularization, 2020.

[22] Aaron Hertzmann, Charles E Jacobs, Nuria Oliver, Brian Curless et David H Salesin : Image analogies. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pages 327–340, 2001.

[23] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath et 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.

[24] Geoffrey Hinton : How to represent part-whole hierarchies in a neural network, 2021.

[25] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou et Alexei A Efros : Image-to-image translation with conditional adversarial networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1125–1134, 2017.

[26] George N Karystinos et Dimitrios A Pados : On overfitting, generalization, and randomly expanded training sets. IEEE Transactions on Neural Networks, 11(5):1050–1057, 2000.

[27] Jang-Hyun Kim, Wonho Choo et Hyun Oh Song : Puzzle mix: Exploiting saliency and local statistics for optimal mixup, 2020.

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

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

[30] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville et Chris Pal : Zoneout: Regularizing rnns by randomly preserving hidden activations. arXiv preprint arXiv:1606.01305, 2016.

[31] Jan Kukačka, Vladimir Golkov et Daniel Cremers : Regularization for deep learning: A taxonomy. arXiv preprint arXiv:1710.10686, 2017.

[32] Yann LeCun, Yoshua Bengio et al. : Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks, 3361(10):1995, 1995.

[33] Yann LeCun, Yoshua Bengio et Geoffrey Hinton : Deep learning. nature, 521(7553): 436–444, 2015.

[34] Yann LeCun, Léon Bottou, Yoshua Bengio et Patrick Haffner : Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[35] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang et al. : Photo-realistic single image super-resolution using a generative adversarial network. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4681–4690, 2017.

[36] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim et Sungwoong Kim : Fast autoaugment, 2019.

[37] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras et Adrian Vladu : Towards deep learning models resistant to adversarial attacks, 2019.

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

[39] Tom M Mitchell et al. : Machine learning. 1997.

[40] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi et Pascal Frossard : Deepfool: a simple and accurate method to fool deep neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2574–2582, 2016.

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

[42] Lutz Prechelt : Early stopping-but when? In Neural Networks: Tricks of the trade, pages 55–69. Springer, 1998.

[43] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su et Leonidas J. Guibas : Frustum pointnets for 3d object detection from rgb-d data, 2018.

[44] Jonas Rauber, Wieland Brendel et Matthias Bethge : Foolbox: A python toolbox to benchmark the robustness of machine learning models, 2018.

[45] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele et Honglak Lee : Generative adversarial text to image synthesis. arXiv preprint arXiv:1605.05396, 2016.

[46] Shaoqing Ren, Kaiming He, Ross Girshick et 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 et R. Garnett, éditeurs : Advances in Neural Information Processing Systems 28, pages 91–99. Curran Associates, Inc., 2015.

[47] Jerome Rony, Luiz G. Hafemann, Luiz S. Oliveira, Ismail Ben Ayed, Robert Sabourin et Eric Granger : Decoupling direction and norm for efficient gradient-based l2 adversarial attacks and defenses. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

[48] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg et Li Fei-Fei : ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV), 115(3):211–252, 2015.

[49] Sara Sabour, Nicholas Frosst et Geoffrey E Hinton : Dynamic routing between capsules, 2017.

[50] Jürgen Schmidhuber : Deep learning in neural networks: An overview. Neural networks, 61:85–117, 2015.

[51] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford et Oleg Klimov : Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[52] Pierre Sermanet, Soumith Chintala et Yann LeCun : Convolutional neural networks applied to house numbers digit classification, 2012.

[53] Patrice Y Simard, Yann A LeCun, John S Denker et Bernard Victorri : Transformation invariance in pattern recognition—tangent distance and tangent propagation. In Neural networks: tricks of the trade, pages 239–274. Springer, 1998.

[54] Patrice Y Simard, David Steinkraus, John C Platt et al. : Best practices for convolutional neural networks applied to visual document analysis. In Icdar, volume 3, 2003.

[55] Karen Simonyan et Andrew Zisserman : Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[56] Karen Simonyan et Andrew Zisserman : Very deep convolutional networks for large-scale image recognition, 2015.

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

[58] Yi Sun, Xiaogang Wang et Xiaoou Tang : Deep learning face representation from predicting 10,000 classes. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1891–1898, 2014.

[59] Ilya Sutskever, Oriol Vinyals et 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.

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

[61] Naftali Tishby et Noga Zaslavsky : Deep learning and the information bottleneck principle. CoRR, abs/1503.02406, 2015.

[62] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun et Christoph Bregler : Efficient object localization using convolutional networks. CoRR, abs/1411.4280, 2014.

[63] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun et Christoph Bregler : Efficient object localization using convolutional networks. CoRR, abs/1411.4280, 2014.

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

[65] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz et Yoshua Bengio : Manifold mixup: Better representations by interpolating hidden states. In Kamalika Chaudhuri et Ruslan Salakhutdinov, éditeurs : Proceedings of the 36th International Conference on Machine Learning, volume 97 de Proceedings of Machine Learning Research, pages 6438–6447, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[66] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein et Justin M. Solomon : Dynamic graph cnn for learning on point clouds, 2019.

[67] Huikai Wu, Shuai Zheng, Junge Zhang et Kaiqi Huang : Gp-gan: Towards realistic high-resolution image blending. In Proceedings of the 27th ACM International Conference on Multimedia, pages 2487–2495, 2019.

[68] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang et Jianxiong Xiao : 3d shapenets: A deep representation for volumetric shapes, 2015.

[69] Raymond A Yeh, Chen Chen, Teck Yian Lim, Alexander G Schwing, Mark Hasegawa-Johnson et Minh N Do : Semantic image inpainting with deep generative models. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 5485–5493, 2017.

[70] Xue Ying : An overview of overfitting and its solutions. In Journal of Physics: Conference Series, volume 1168, page 022022. IOP Publishing, 2019.

[71] Kevin Y Yip et Mark Gerstein : Training set expansion: an approach to improving the reconstruction of biological networks from limited and uneven reliable interactions. Bioinformatics, 25(2):243–250, 2009.

[72] Fisher Yu et Vladlen Koltun : Multi-scale context aggregation by dilated convolutions, 2016.

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

[74] Sergey Zagoruyko et Nikos Komodakis : Wide residual networks, 2017.

[75] Luiz Zaniolo et Oge Marques : On the use of variable stride in convolutional neural networks. Multimedia Tools and Applications, 79(19):13581–13598, 2020.

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