MIXFEAT: MIX FEATURE IN LATENT SPACE LEARNS DISCRIMINATIVE SPACE

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep learning methods perform well in various tasks. However, the over-fitting problem remains, where the performance decreases for unknown data. We here provide a novel method named *MixFeat*, which directly makes the latent space discriminative. MixFeat mixes two feature maps in each latent space and uses one of their labels for learning. We report improved results obtained using existing network models with MixFeat on CIFAR-10/100 datasets. In addition, we show that MixFeat effectively reduces the over-fitting problem even in the case that the training dataset is small or contains errors. We argue that MixFeat is complementary with existing methods that mix both images and labels, in that MixFeat is suitable for discrimination tasks while existing methods are suitable for regression tasks. MixFeat is easy to implement and can be added to various network models without additional computational cost in the inference phase.

1 INTRODUCTION

Deep neural networks (LeCun et al., 1998) have performed well for various tasks, such as image recognition (Krizhevsky et al., 2012; Simonyan & Zisserman, 2015; He et al., 2016a;b; Han et al., 2017; Huang et al., 2017), object detection (Ren et al., 2015; Redmon et al., 2016), and semantic segmentation (Chen et al., 2018; Badrinarayanan et al., 2017).

One remaining problem with training deep neural networks is the over-fitting of training data, despite many methods having been proposed to solve this problem; e.g., the dropout method (Srivastava et al., 2014) drops randomly selected elements of feature maps, the mixup method (Zhang et al., 2018a) and between-class learning (Tokozume et al., 2018) mix pairs of training images and labels, and the manifold mixup method (Verma et al., 2018) mixes pairs of training feature maps on a randomly selected latent space and labels. We propose a method named MixFeat to learn discriminative features in each latent space directly.

The main contributions of this paper are as follows.

- We propose the MixFeat scheme to reduce the over-fitting problem by mixing two feature maps in latent space and making the latent space discriminative without any additional computational cost in the inference phase.
- We present a guideline for judging whether labels should be mixed when mixing features for an individual purpose.
- We conduct extensive experiments to demonstrate the effectiveness of the generalization of MixFeat.

MixFeat is easy to implement and can be added to various neural network models. It has the potential to be applied for various discrimination tasks, such as object detection, semantic segmentation, and anomaly detection. MixFeat is described in the next section.

2 MIXFEAT

2.1 OVERVIEW

We consider the method of avoiding over-fitting by making features discriminative in the latent space of neural networks. Training the perturbed sample to output the same inference as the pure sample enlarge Fisher's criterion (Fisher, 1936) (*i.e.*, the ratio of the between-class distance to the within-class variance). It is therefore conceivable to learn the perturbed samples when learning the discriminative latent space. However, a perturbation independent of the given examples is inefficient because the latent space is extremely high-dimensional and dynamically changes during learning. We consider that the perturbation should be determined according to the subspace spanned by a plurality of samples in the latent space. For simplification, we adopt a partial plane spanned by the origin, the base example, and another example, as shown in Fig 5(a). We now propose the method MixFeat, which mixes two feature maps in each latent space to learn the discriminative latent space.

2.2 MIXING ONLY FEATURES OR BOTH FEATURES AND LABELS

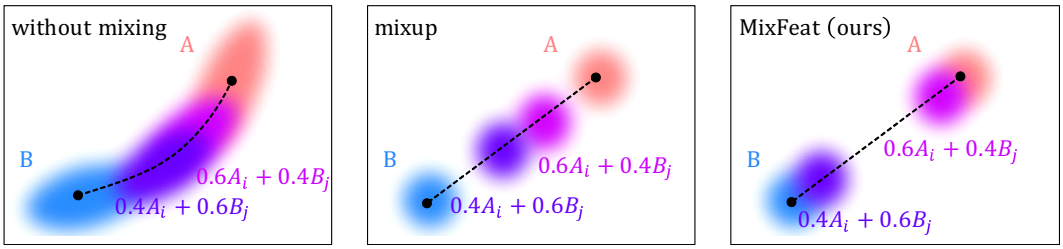

Figure 1: The class distributions in the learned latent spaces, where A and B respectively denote the distributions of classes A and B while A_i and B_j respectively denote examples in classes A and B. The mixed features are distributed regressively in the mixup method while the mixed features are distributed discriminatively in MixFeat.

In related works, methods that mix two images and labels have recently been proposed. The mixup method (Zhang et al., 2018a), between-class learning (BCL) (Tokozume et al., 2018) and manifold mixup method (Verma et al., 2018) mix two training examples both in features and labels with random weights, and they have been reported to reduce over-fitting appreciably.

We argue that mixing methods are able to constrain the feature distribution, which cannot be achieved by training without mixing, and the obtained feature space greatly differs between MixFeat and the mixup family of methods. Figure 1 shows the assumed class distributions in learned feature spaces. Without mixing, the feature distribution of the mixed features becomes large and largely overlaps the feature distributions of classes A and B and other mixing ratio features (Fig. 1(left)). Mixing methods penalize this situation in different ways. The Mixup family allow the model to output the mixing ratio and any overlap becomes small, and the latent space thus becomes regressive (Fig. 1(center)). MixFeat lets the model output the same inference between the mixed sample and pure sample, and only the overlap between classes becomes small, such that the latent space becomes more discriminative than that in the case of the mixup method (Fig. 1(right)). It is thus confirmed that these mixing methods can be used properly according to the purpose, which is regressive or discriminative.

2.3 COMPUTATION OF MIXFEAT

The process of MixFeat is shown in Fig. 2. Let $\oplus$ denote the addition operator and $\odot$ denote the sample-wise product.

The forward training pass, as shown in Fig. 2(a), is described as

$$\boldsymbol{Y} = \boldsymbol{X} + (\mathbf{a} \odot \boldsymbol{X} + \mathbf{b} \odot F(\boldsymbol{X})), \tag{1}$$

$$\text{where } \mathbf{a} = \mathbf{r}\cos\boldsymbol{\theta},\ \mathbf{b} = \mathbf{r}\sin\boldsymbol{\theta},\ \mathbf{r} \sim N(0, \sigma^2),\ \boldsymbol{\theta} \sim U(-\pi, \pi),$$

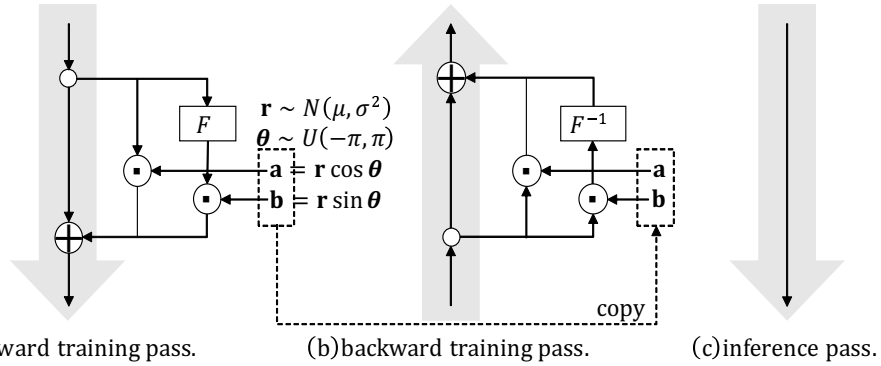

Figure 2: Computational passes of MixFeat. Let $\oplus$ denote the addition operator and $\odot$ denote the sample-wise product. $\mathbf{r}$ is a random value vector sampled from a Gaussian distribution $N(0, \sigma^2)$ while $\boldsymbol{\theta}$ is a random-value vector sampled from a uniform distribution $U(-\pi, \pi)$, with an element being associated with a feature map in the mini-batch. Furthermore, F denotes the random sort operation along the example axis to the input tensor, F^{-1} denotes the restoring order operation along the example axis to the input tensor, and (*copy*) denotes copying the vector from the forward training pass. The inference phase returns the input tensor as it is.

where the first term $\boldsymbol{X}$ is the input mini-batch tensor, the second term $\mathbf{a} \odot \boldsymbol{X} + \mathbf{b} \odot F(\boldsymbol{X})$ is the perturbation, $\boldsymbol{Y}$ denotes the output mini-batch tensors of MixFeat, $\mathbf{1}$ is a vector of ones of appropriate length, $\mathbf{r}$ is a Gaussian random vector with $N(0, \sigma^2)$, and $\boldsymbol{\theta}$ is a uniform random vector with $U(-\pi, \pi)$, with an element being associated with an example in the mini-batch on the latent space. Furthermore, $F(\boldsymbol{X})$ denotes the randomly sorted $\boldsymbol{X}$ along the example axis. An appropriate value for σ is discussed later.

The backward training pass, shown in Fig.2(b), is calculated as

$$\boldsymbol{G_X} = \boldsymbol{G_Y} + (\mathbf{a} \odot \boldsymbol{G_Y} + F^{-1}(\mathbf{b} \odot \boldsymbol{G_Y})), \tag{2}$$

where $\boldsymbol{G_X}$ and $\boldsymbol{G_Y}$ respectively denote the partial derivatives of the final output loss function with respect to $\boldsymbol{X}$ and $\boldsymbol{Y}$, $F^{-1}(\cdot)$ denotes the inverse operation of $F(\cdot)$, which restores the order of examples before the random sorting, and vectors $\mathbf{a}$ and $\mathbf{b}$ are copies of $\mathbf{a}$ and $\mathbf{b}$ in the forward training pass (*copy* in Fig. 2).

During inference, as shown in Fig. 2(c), the perturbation branches are not necessary for inference and can be removed:

$$\boldsymbol{Y} = \boldsymbol{X}. \tag{3}$$

Equation (3) indicates the MixFeat layer in the inference phase returns the input as it is; i.e., the MixFeat layer can be simply removed from the inference phase and it thus does not have any additional computational cost at this point.

3 Experiments

3.1 CIFAR-10 and 100 datasets

The following experiments were conducted on the CIFAR-10 and -100 datasets (Krizhevsky & Hinton, 2009). The two CIFAR datasets consist of RGB natural images comprising 32×32 pixels. CIFAR-10 consists of images drawn from 10 classes while CIFAR-100 is drawn from 100 classes. The CIFAR-10 and -100 datasets respectively contain 50,000 training images and 10,000 test images. In our experiments, the input images of the CIFAR-10 and -100 datasets were processed adopting the following conventional augmentation process (Krizhevsky et al., 2012; Simonyan & Zisserman, 2015). The original image of 32×32 pixels was color-normalized and then horizontally flipped with 50% probability. It was then zero-padded to a size of 40×40 pixels and randomly cropped to an image of 32×32 pixels.

All models were trained employing back-propagation and a stochastic gradient descent with Nesterov momentum (Sutskever et al., 2013). We adopted the weight initialization introduced by He et al. (2015). A single graphics processing unit (GeForce GTX Titan X or GeForce GTX 1080 Ti) was used for each training. The initial learning rate was set to 0.05 and decayed by a factor of 0.1 at the half and three-quarter points of the overall training process (300 epochs), following Huang et al. (2017). In addition, we used a weight decay (Krogh & Hertz, 1992) of 5×10^{-4}, momentum of 0.9, and batch size of 128.

We compared the performance of the proposed MixFeat method with another over-fitting avoidance method: the mixup method (Zhang et al., 2018a). We trained ResNet (pre-activation version) (He et al., 2016b), DenseNet, DenseNetBC (Huang et al., 2017), PyramidNet (Han et al., 2017). For the mixup method, the beta distribution parameter $\alpha = 0.2$ was used. For PyramidNet, the initial learning rate was set to 0.01 and a batch size of 32 was used depending on the memory limitation of the graphics processing unit. We implemented the methods using Chainer v4.4.0 (Tokui et al., 2015).

Results are given in Table 1. The results obtained with the mixup method were consistently better than those obtained with the vanilla model, which is a model that does not avoid over-fitting. Ultimately, the best performance was obtained when MixFeat was adopted for all network models. The best results on the CIFAR-10 and -100 datasets were 2.92% and 16.03% for 272-layer PyramidNet. These results demonstrate that MixFeat improves the performance of various network models.

The training and test error curves obtained with and without MixFeat are shown in Fig. 3. The test error rate training with MixFeat shows better convergence than the rate training without MixFeat. In addition, the training and testing error curves for training with MixFeat are closer together than those for training without MixFeat, which shows that MixFeat reduces over-fitting.

Table 1: Benchmark results on the CIFAR-10 and -100 datasets. We show the average and standard errors for five trials. Here, (k) and (α) respectively denote the network's growth rate in DenseNet and PyramidNet. The order of the layers in a module are the same in each model: BN-ReLU-Conv (ResNet and DenseNetBC repeat this order twice for each module). *Vanilla* indicates that nothing was done to avoid over-fitting, *mixup* (Zhang et al., 2018a) is used as the learning scheme and *MixFeat* is located immediately after each convolution layer. Overall, MixFeat improves the performance of all convolutional neural network models.

				Test error rate (%)		
Dataset	Model	Depth	#Params	Vanilla	Mixup	MixFeat(ours)
CIFAR-10	ResNet	20	0.3M	7.33 ± 0.09	7.00 ± 0.19	$\mathbf{6.54 \pm 0.24}$
	ResNet	110	1.7M	5.18 ± 0.10	5.01 ± 0.15	$\mathbf{4.78 \pm 0.18}$
	ResNet(BottleNeck)	164	1.7M	4.61 ± 0.07	4.38 ± 0.08	$\mathbf{3.78 \pm 0.04}$
	DenseNet(k=12)	40	1.0M	5.59 ± 0.20	5.20 ± 0.24	$\mathbf{4.82 \pm 0.10}$
	DenseNet-BC(k=12)	100	0.8M	4.66 ± 0.12	4.58 ± 0.11	$\mathbf{3.91 \pm 0.13}$
	PyramidNet(α=200)	272	26.0M	3.64 ± 0.15	-	$\mathbf{2.92 \pm 0.06}$
CIFAR-100	ResNet	20	0.3M	31.39 ± 0.26	30.44 ± 0.38	$\mathbf{29.67 \pm 0.23}$
	ResNet	110	1.7M	24.59 ± 0.13	24.17 ± 0.18	$\mathbf{23.58 \pm 0.46}$
	ResNet(BottleNeck)	164	1.7M	21.31 ± 0.23	20.58 ± 0.26	$\mathbf{19.92 \pm 0.24}$
	DenseNet(k=12)	40	1.0M	26.40 ± 0.23	24.68 ± 0.28	$\mathbf{23.31 \pm 0.28}$
	DenseNet-BC(k=12)	100	0.8M	22.98 ± 0.13	21.89 ± 0.27	$\mathbf{20.09 \pm 0.16}$
	PyramidNet(α=200)	272	26.0M	17.74 ± 0.23	-	$\mathbf{16.03 \pm 0.15}$

3.2 Performance of avoiding over-fitting

We present experimental results that demonstrate how well MixFeat avoids over-fitting.

3.2.1 Incorrect labels in the training dataset

Incorrect labels in the training dataset worsen the test error rates because of over-fitting. We thus compared the test error rates with and without MixFeat while changing the ratio of incorrect labels

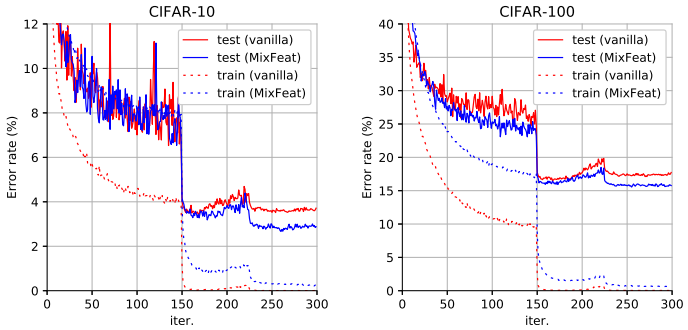

Figure 3: Training and test error curves for PyramidNet ($depth = 272, \alpha = 200$). **Left:** Training curve on CIFAR-10. **Right:** Training curve on CIFAR-100. The discrepancy between the training and test curves is suppressed with MixFeat.

in the training dataset. Results are shown in Fig. 4. As shown in Fig. 4 (Left), increasing the ratio of incorrect labels in the training data greatly magnifies the error rate without MixFeat whereas the increase in the error rate is considerably suppressed when MixFeat is used. As shown in Fig. 4 (Center and Right), the test curves worsen drastically after the peak without MixFeat whereas the test curves are kept low by MixFeat.

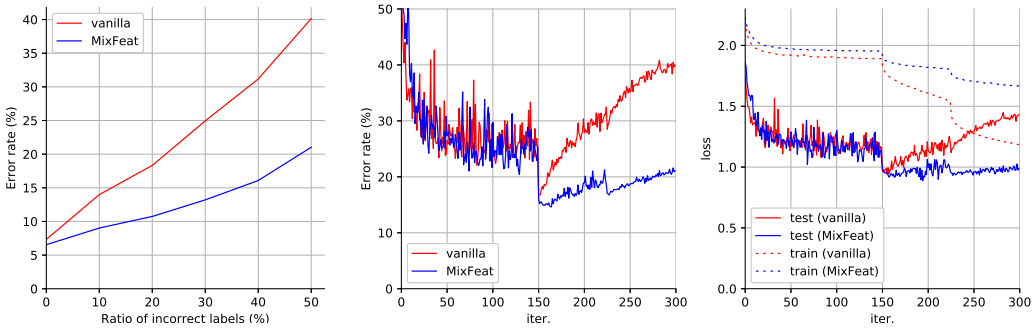

Figure 4: **Left:** Test error (%) results for an increasing number of incorrect labels in the training dataset with and without MixFeat on CIFAR-10 using 20-layer ResNets (pre-activation). The increase in the error rate with the increasing number of incorrect labels is suppressed with MixFeat. **Center:** Test error curves for the training dataset with 50% incorrect labels with and without MixFeat. **Right:** Training and test loss curves for the training dataset with 50% incorrect labels with and without MixFeat. The test curves worsen drastically after the peak without MixFeat whereas they are kept low by MixFeat.

3.2.2 Reducing the size of the training dataset

In general, reducing the size of the training dataset results in over-fitting. We thus compared the test error rates with and without MixFeat while reducing the size of the training dataset. In this experiment, the number of parameter update iterations was made the same by increasing the number of epochs in inverse proportion to the training dataset size. Figure 6 (Left) shows that reducing the number of training data greatly increases the error rate without MixFeat whereas the increase in the error rate is considerably suppressed when MixFeat is used.

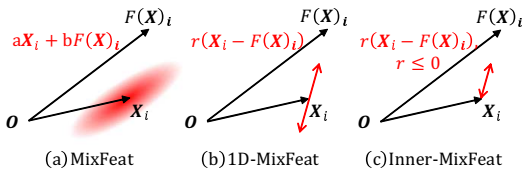

Figure 5: Distribution of various MixFeat perturbations. From left to right, (a) MixFeat, (c) 1D-MixFeat and (d) inner-MixFeat.

The results of these experiments indicate that MixFeat prevents deep neural networks from over-fitting the training data.

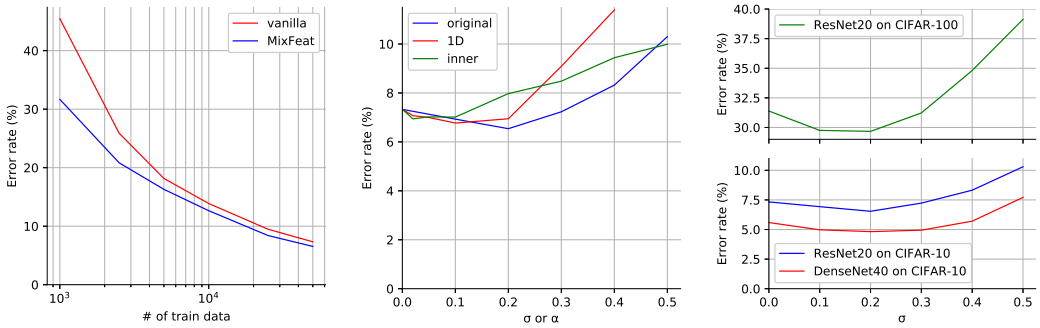

Figure 6: **Left:** Test error (%) results when reducing the training dataset size with and without MixFeat on CIFAR-10 using 20-layer ResNets (pre-activation). The increase in the error rate with data reduction is suppressed with MixFeat. **Center:** Comparison of the results of original MixFeat, 1D-MixFeat and inner-MixFeat with changing hyperparameter σ on CIFAR-10 using 20-layer ResNets (pre-activation). The original MixFeat has the highest performance. **Right:** Comparison of the results for various hyperparameter σ values in MixFeat for various settings. The best value is $\sigma = 0.2$ in each setting.

3.3 Ablation analysis

3.3.1 Dimensions and direction of the distribution

We can simply modify MixFeat as a one-dimensional version (1D-MixFeat) and inner-division version (inner-MixFeat) as shown in Fig. 5. The forward training pass of 1D-MixFeat is described as

$$\boldsymbol{Y} = \boldsymbol{X} + \mathbf{r} \odot (\boldsymbol{X} - F(\boldsymbol{X})) \tag{4}$$

while the backward training pass is described as

$$\boldsymbol{G_X} = \boldsymbol{G_Y} + (\mathbf{r} \odot \boldsymbol{G_Y} - F^{-1}(\mathbf{r} \odot \boldsymbol{G_Y})), \tag{5}$$

and the inference phase is the same as that of the original MixFeat.

Inner-MixFeat follows the mixing concept of the mixup method and BCL. The forward training pass is described as

$$\boldsymbol{Y} = \boldsymbol{X} + \mathbf{r}' \odot (\boldsymbol{X} - F(\boldsymbol{X})) \quad \text{where} \quad \mathbf{r}' = |B(\alpha, \alpha) - 0.5| - 0.5, \tag{6}$$

where $\mathbf{r}'$ is the mixing ratio based on a random vector with a beta distribution $B(\alpha, \alpha)$ following the mixup method. Note that we modified the mixing ratio for consistency between the major component of the mixed image and the label. The backward training pass is described as

$$\boldsymbol{G_X} = \boldsymbol{G_Y} + (\mathbf{r}' \odot \boldsymbol{G_Y} - F^{-1}(\mathbf{r}' \odot \boldsymbol{G_Y})) \tag{7}$$

and the inference phase is

$$\boldsymbol{Y} = (1 + \mathbb{E}\mathbf{r}')\boldsymbol{X}, \tag{8}$$

where $\mathbb{E}\mathbf{r}'$ denotes the expected value of $\mathbf{r}'$. Figure 6 (Center) compares the test error rate with changing hyperparameter σ for the original MixFeat, 1D-MixFeat and inner-MixFeat. The best results for each variation of MixFeat were $6.54\%(\sigma = 0.2)$ for the original MixFeat, $6.77\%(\sigma = 0.1)$ for 1D-MixFeat, and $6.94\%(\sigma = 0.02)$ for inner-MixFeat. The original MixFeat thus has the highest performance.

3.3.2 Location of MixFeat in the network

We investigated the location of MixFeat in a commonly used pre-activation unit (He et al., 2016b; Huang et al., 2017), which consists of convolution (*Conv*), batch normalization (*BN*) (Ioffe & Szegedy, 2015), and a rectified linear unit (*ReLU*) (Nair & Hinton, 2010), referred to as *-BN-ReLU-Conv-*. Table 2 shows that MixFeat is performed regardless of the location but the best location is after convolution. We therefore place MixFeat directly after each convolution.

3.3.3 Reasonable hyperparameter value σ

We investigated the standard deviation σ of the distribution of $\mathbf{r}$ in Eq. (1). This investigation thus elucidates the optimal range of the distribution for effective perturbation. Figure 6 (Right) compares the results. Here, a value of σ that is too small does not improve the performance while a value of σ that is too large decreases the performance appreciably. That is to say, the majority of the components of $\boldsymbol{Y}$ are replaced from the input tensor to the perturbation tensor if $|\mathbf{r}|$ is too large, and this range of values thus does not work well. Although σ cannot be theoretically determined, $\sigma = 0.2$ is the experimentally determined optimal hyperparameter.

Table 2: Comparison of the MixFeat location in a *-(1)-BN-(2)-ReLU-(3)-Conv-(4)-* preactivation unit. MixFeat is performed regardless of the location but the best location is after convolution.

20-layer ResNet on CIFAR-10	
MixFeat location	Error rate (%)
(no MixFeat)	7.33
(1)	6.74
(2)	6.85
(3)	6.66
(4)	**6.54**

4 Relationship with previous work

We discuss the relationship between our approach and the others that reduce the over-fitting of training data. Our approach is related to a series of approaches based on perturbing training data. We argue the differences between our approach and the others as follows.

Data augmentation methods for input images are widely used. The conventional data augmentation method (Krizhevsky et al., 2012; Simonyan & Zisserman, 2015) adds perturbations to the input images through geometric or value transformations. Cutout (DeVries & Taylor, 2017b) and random erasing (Zhong et al., 2017) methods overwrite elements in randomly selected rectangular regions with zeros or random values. These methods are intuitive and easily adjustable. Reasonable perturbation on the input image is independent of our method and a synergistic effect can be expected when using these methods together with our method, MixFeat.

"Drop" perturbations are used for regularization and/or convergence acceleration. Dropout (Srivastava et al., 2014) drops randomly selected elements of feature maps, Dropconnect (Wan et al., 2013) drops randomly selected network connections and ResDrop (Huang et al., 2016) drops randomly selected residual paths in ResNets (He et al., 2016a). It seems these methods have not been used much in recent times owing to their poor compatibility with batch normalization or complicated implementation. However, these methods can be used with MixFeat if needed.

"Shake" methods calculate randomly weighted sums of parallel network branches. Shake-shake (Gastaldi, 2017) mixes the identity map and two residual branches with i.i.d. random weights for forward and backward passes. ShakeDrop (Yamada et al., 2018) mixes the identity map and one residual branch with independent (and not identically) random weights for forward and backward pass. They acquire an ensemble effect in each "shake" block. However, these methods worsen convergence speeds as reported in the cited study. The study reported 1800-epoch training was better on CIFAR datasets, compared with 300-epoch training as a popular setting. This result is presumed to be due to the parallel network branches not being on the same mapping, which is different from the case for MixFeat.

The following methods mix two images. Sample pairing (Inoue, 2018) repeats two learning phases alternately, with one phase learning the average of two images and one of their labels and the other phase learning the input image and label as they are. Augmentation in feature space (DeVries & Taylor, 2017a) mixes two neighboring images in a pretrained feature space to expand the distribution of the feature maps. MixFeat is considered an extension of these methods to a dynamic latent space.

Methods that mix two images and labels have recently been proposed. Mixup (Zhang et al., 2018a) and BCL (Tokozume et al., 2018) mix two training examples (an image and label) with random weights, and they are reported to reduce over-fitting appreciably. The manifold mixup method (Verma et al., 2018) mixes two training examples as does the mixup method but mixes feature maps randomly selected from some predetermined latent space instead of images. The difference between MixFeat and these methods is described in 2.2.

Manifold adversarial training (MAT) (Zhang et al., 2018b) learns the most sensitive adversarial examples in each feature map through two-step learning for each mini batch. In the first step, the most sensitive direction for each example in the given mini batch is found. In the second step, each

example is shifted in the most sensitive direction in each feature map. MAT has the same purpose as our method in that it makes feature maps discriminative, but the magnitude of the perturbation cannot be determined reasonable because the perturbation vector is determined from only the sensitive direction. Moreover, MAT increases the training time because of the two-step learning process.

5 VISUALIZATION

Finally, we visualize the feature distributions learned with vanilla and MixFeat methods in Fig. 7. We trained a six-layer neural network with or without MixFeat on two-class toy-data that have a two-dimensional checkerboard distribution. The vanilla network architecture is a three-fold stack of $\{\text{fc}(20) \rightarrow \text{tanh} \rightarrow \text{fc}(2)\}$ while the MixFeat network architecture is a three-fold stack of $\{\text{fc}(20) \rightarrow \text{tanh} \rightarrow \text{MixFeat} \rightarrow \text{fc}(2)\}$, where $\text{fc}(k)$ denotes the k-way fully connected layer. The figure shows that the features obtained with MixFeat are discriminatively distributed even after hidden layers while the features obtained without MixFeat are discriminatively distributed only in the output. We conjecture that is why the classification performance improved with MixFeat.

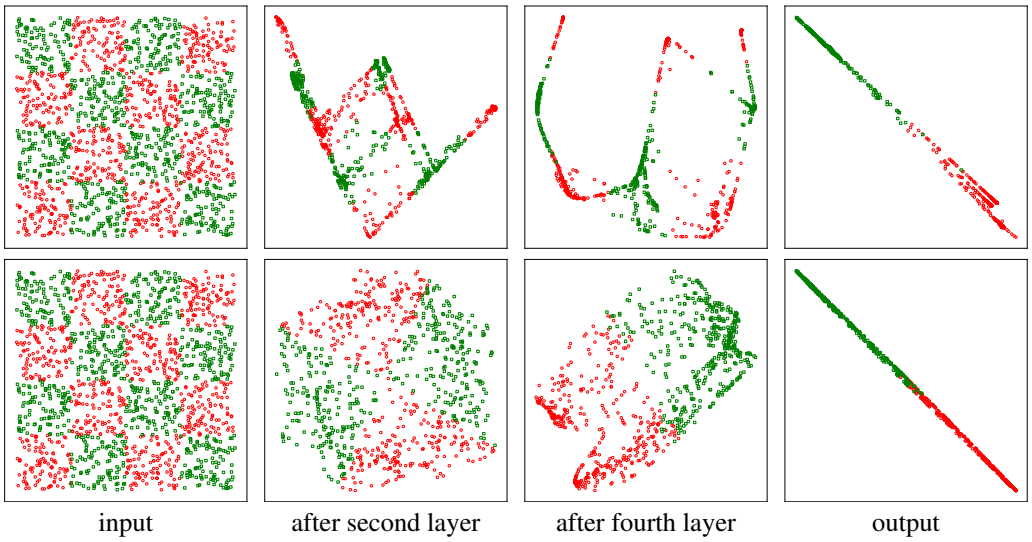

Figure 7: Visualization of feature distributions after 1200-epoch training with a six-layer neural network on two-dimensional toy-data. **Top row** shows the results without MixFeat, **Bottom row** shows the results with MixFeat. From left to right, the input distribution, intermediate distributions after second and fourth layers and output distribution. The distributions obtained with learning with MixFeat are more discriminative for each class at each depth.

6 CONCLUSIONS

We proposed a novel method named MixFeat, which mixes two feature maps in latent space to avoid over-fitting in training deep neural networks. As a result, the mixed feature reasonably expands the feature distribution in each latent space and makes the latent space discriminative to improve generalization performance. Our experimental results show that MixFeat appreciably improves the generalization performance. We discussed the relationship between our approach and a series of previously reported approaches and recommended the proper use of the MixFeat and mixup methods according to the task being discriminative or regressive. Further studies are needed to extend MixFeat to tasks that use only small mini-batches, such as object detection or semantic segmentation. Because the MixFeat module can be easily added to various network models without additional computational cost in the inference phase, we believe that it will be the de facto standard for methods of reducing over-fitting.

REFERENCES

Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *TPAMI*, 39(12):2481–2495, 2017.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 40(4):834–848, 2018.

Terrance DeVries and Graham W Taylor. Dataset augmentation in feature space. In *ICLR Workshop*, 2017a.

Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017b.

Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7 (2):179–188, 1936.

Xavier Gastaldi. Shake-shake regularization. In *ICLR Workshop*, 2017.

Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. In *CVPR*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016b.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *CVPR*, 2017.

Hiroshi Inoue. Data augmentation by pairing samples for images classification. *arXiv preprint arXiv:1801.02929*, 2018.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Tech Report*, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *NIPS*, 1992.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *IEEE*, 1998.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.

Yuji Tokozume, Yoshitaka Ushiku, and Tatsuya Harada. Between-class learning for image classification. In *CVPR*, 2018.

Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *NIPS Workshop on Machine Learning Systems*, 2015.

Vikas Verma, Alex Lamb, Christopher Beckham, Aaron Courville, Ioannis Mitliagkis, and Yoshua Bengio. Manifold mixup: Encouraging meaningful on-manifold interpolation as a regularizer. *arXiv preprint arXiv:1806.05236v1*, 2018.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *ICML*, 2013.

Yoshihiro Yamada, Masakazu Iwamura, and Koichi Kise. Shakedrop regularization. In *ICLR Workshop*, 2018.

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018a.

Shufei Zhang, Kaizhu Huang, Jianke Zhu, and Yang Liu. Manifold adversarial learning. *arXiv preprint arXiv:1807.05832*, 2018b.

Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896*, 2017.