Mixed Samples Data Augmentation with Replacing Latent Vector Components in Normalizing Flow

Genki Osada[1,2], Budrul Ahsan[3], and Takashi Nishide[2]

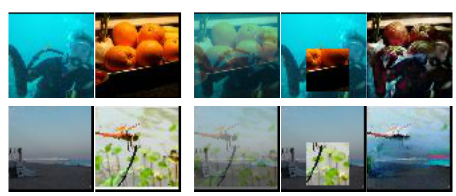
[1]LINE Corporation, Japan
[2]University of Tsukuba, Japan
[3]IBM Japan

Abstract

Data augmentation mixing two samples has been acknowledged as an effective regularization method for various deep neural network models. Given that images mixed by popular methods (e.g., MixUp and CutMix) are unnatural to the human eye, we hypothesized that generating more natural images could achieve better performance as data augmentation. To verify this, we propose a new mixing method that synthesizes images in which two source images coexist naturally. Our method performs a mixing operation in latent space through a normalizing flow, and the key is how to mix two latent vectors. We preliminarily observed that there exists a dependency between the dimensions in input space and those in latent space in transformation with normalizing flows. Based on this observation, we designed our mixing scheme in latent space. We show that our method yields visually natural augmented images and improves classification performance.

1 Introduction

Data augmentation methods that generate new training data by mixing two source samples have been increasingly popular to regularize deep neural network models. We refer to those methods as mixed samples data augmentation (MDA). Even basic methods such as MixUp [1, 2] and CutMix [3] are widely acknowledged for their effectiveness. However, their success is somewhat surprising, given that the images generated by these methods are unnatural to the human eye. (see Fig. 1.) Considering that input images at the test time do not contain such unnaturalness, we hypothesize that if we can generate mixed samples more naturally, it would become more effective as augmented data.



Source-1 Source-2 MixUp CutMix LS-Mix

Figure 1: Visual comparison of mixing methods. Unlike MixUp and CutMix, our method, LS-Mix, generates natural mixed images.

We thus propose an MDA method that synthesizes images, in which two source images coexist naturally by being stitched together without producing artifacts. We perform mixing operations in latent space instead of input data space using lossless invertible transformation with normalizing flows (NFs) [4, 5, 6]. How to mix two latent vectors is the key to generating natural images. As we will describe in Section 3, we found a dependency between the dimensions of a latent vector and the pixels of its corresponding image. By utilizing that dependency, we design a mixing scheme that naturally stitches part of two source images while preserving their original appearance to the maximum extent. We call our method Latent space Sequential Mix (LS-Mix). The overview of

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

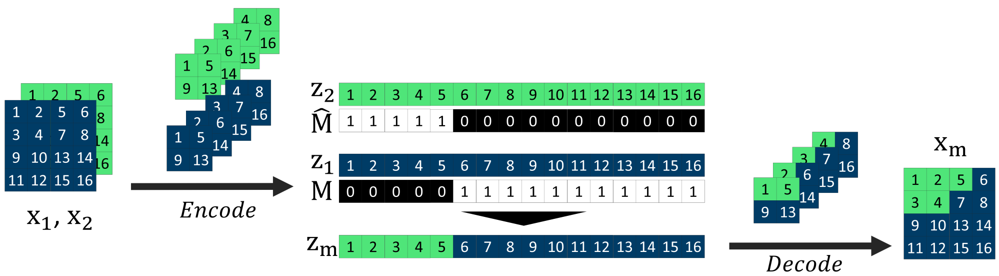


Figure 2: Illustration of our method, LS-Mix. Mixed image $\mathbf{x}_\mathrm{m}$ is generated from two source images $\mathbf{x}_1$ and $\mathbf{x}_2$, shape of which is $[4, 4, 1]$, i.e., dimensionality $D = 16$. *Encode* and *Decode* in figure correspond to invertible function $g()$ and $g^{-1}()$, respectively. In Encode process, squeezing with `space_to_depth` operation transforms $\mathbf{x}_1$ and $\mathbf{x}_2$ into $[1, 1, 16]$ shape tensors, $\mathbf{z}_1$ and $\mathbf{z}_2$, via $[2, 2, 4]$ tensors. Let mixture rate λ be $\frac{5}{16}$, first 5 components of $\mathbf{M}$ are set to 0, 1 otherwise, and mixed latent vector $\mathbf{z}_\mathrm{m}$ is generated with it, and $\widehat{\mathbf{M}}$ denotes $\mathbb{1} - \mathbf{M}$ in figure. In Decode process, inverse operation of squeezing, `depth_to_space`, transforms $\mathbf{z}_\mathrm{m}$ with shape of $[1, 1, 16]$ into $\mathbf{x}_\mathrm{m}$ of $[4, 4, 1]$ via $[2, 2, 4]$ tensor.

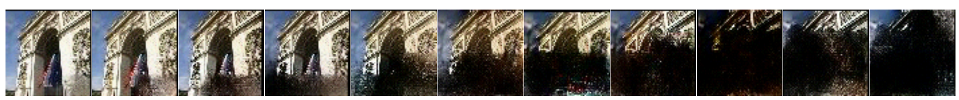


Figure 3: Observation of squeezing dependency. Decoded images $\mathbf{x}' = g^{-1}(\mathbf{z}')$ are shown where $\mathbf{z}'$ is made by replacing last $\lfloor \lambda D \rceil$ components of latent vector $\mathbf{z} = g(\mathbf{x}) \in \mathbb{R}^D$ with 0-value and $\mathbf{x}$ denotes original image. λ increases as $\frac{1}{12}, \frac{2}{12}, \ldots, \frac{11}{12}$ and corresponding $\mathbf{x}'$ are shown left to right.

LS-Mix is shown in Fig. 2, and the images generated by LS-Mix are shown in Fig. 1. In this short paper, we perform the evaluation on SVHN, CIFAR-10, CIFAR-100, and TinyImageNet, and show that LS-Mix outperforms MixUp, CutMix, and other methods that perform mixing in latent space.

2 Preliminary

We use normalizing flows (NFs), $g(\cdot)$, for the invertible transformation between the input space $\mathcal{X}$ and the latent space $\mathcal{Z}$. An NF produces clear reconstructed images because of its invertibility, unlike VAE [7, 8], whose reconstructed images are known to become blurred [9, 10]. See Appendix A for NFs. The architecture we use is Glow [11]. Our proposed method utilizes the mechanism called *squeezing*, which is one of the building blocks of NFs.

Squeezing. While the shape of images $\mathbf{x}$ is $[h, w, c]$ with spatial dimensions $h \times w$ and channel dimension c, the latent vectors $\mathbf{z} = g(\mathbf{x})$ mapped with a NF are D dimensional flat vectors with no spatial structure, where $D = h \times w \times c$. NFs perform the conversion of shapes between $\mathbf{x}$ and $\mathbf{z}$ by an operation called *squeezing*. The squeezing is typically done with iterative use of `space_to_depth` operation which converts the shape of inputs from $[h, w, c]$ to $[\frac{h}{2}, \frac{w}{2}, 4c]$ until it becomes $[1, 1, D]$, which are treated as $D(= 1 \times 1 \times D)$ dimensional flat vectors in the implementation. In the inverse transformation g^{-1} that transforms $\mathbf{z}$ back to $\mathbf{x}$, the opposite conversion, `depth_to_space` operation is performed. How exactly the `space_to_depth` and `depth_to_space` operations perform the folding of the spatial dimensions into the channel dimension depends on the implementation, but typically it is implemented to work in a manner equal to `tf.nn.space_to_depth` and `tf.nn.depth_to_spaceAPI` in TensorFlow, which we use in our experiments. We depict the process of `space_to_depth` and `depth_to_space` in Fig. 2.

3 Our Method

Motivating observation: squeezing dependency. The key to synthesizing natural images is how to mix two latent vectors. We preliminarily observed that in the transformation with NFs, there

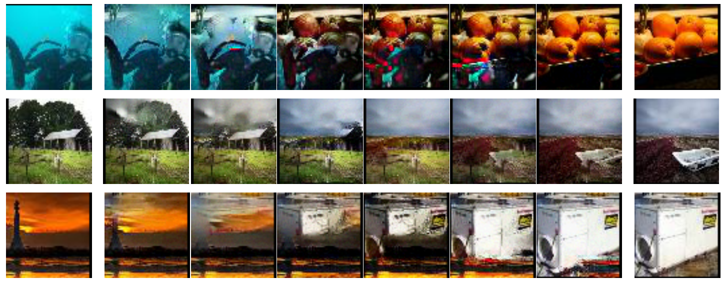


Figure 4: Mixed images with our method, LS-Mix. Columns at both ends are source images, and images in middle are mixture of them with mixing rate $\lambda = \{\frac{2}{16}, \frac{5}{16}, \frac{7}{16}, \frac{9}{16}, \frac{11}{16}, \frac{14}{16}\}$.

exists a dependency between the dimensions in $\mathcal{X}$ and $\mathcal{Z}$, which we call *squeezing dependency*. We experimentally observed how the generated image $\mathbf{x} = g^{-1}(\mathbf{z})$ changes when some consecutive components of latent vector $\mathbf{z}$ are replaced with 0-value. Fig. 3 shows that as the number of consecutive components replaced with 0 gradually increases, the generated image becomes closer to being all black. We can see that the transition to the black image follows not the way that the entire image gradually becomes darker but instead the way that a small black area initially appearing in the lower right corner expands to cover the entire image. We identified that the order of the black area expansion is in accordance with the alignment order of the `depth_to_space` conversion in the squeezing operation described in Section 2. We thus interpreted that this dependency comes from the order of squeezing applied during the training of the NF model. We note that what we did in this experiment corresponds to the case where all elements of $\mathbf{z}_1$ in Fig. 2 were replaced by 0. From this observation, it is shown that consecutive components in $\mathbf{z}$ correspond to consecutive areas in $\mathbf{x} = g^{-1}(\mathbf{z})$. It suggests that manipulating the latent vectors following the squeezing dependency allows us to generate an image preserving a specific region of the source images. Based on this observation, we designed our mixing method.

Method. We call our method Latent space Sequential Mix (LS-Mix). LS-Mix works as follows: mapping two source inputs $\mathbf{x}_1$ and $\mathbf{x}_2$ to the latent space $\mathcal{Z}$, we obtain the corresponding latent vectors, $\mathbf{z}_1 = g(\mathbf{x}_1)$ and $\mathbf{z}_2 = g(\mathbf{x}_2)$. The mixing operation is done as

$$\mathbf{z}_{\text{m}} = \mathbf{M} \otimes g(\mathbf{x}_1) + (\mathbb{1} - \mathbf{M}) \otimes g(\mathbf{x}_2) \tag{1}$$

where $\mathbf{M} \in \{0, 1\}^D$ is a binary mask, D is the dimensionality of $\mathcal{Z}$ (and $\mathcal{X}$), $\mathbb{1}$ is a D-dimensional vector in which all elements are 1, and $\otimes$ is an element-wise product. The key of our method is how to create $\mathbf{M}$, and following the squeezing dependency, we simply set the first $\lfloor \lambda D \rceil$ components of $\mathbf{M}$ to 0 and 1 otherwise, according to the mixing rate λ. As the value of λ increases from 0 to 1, the size of the area that comes from $\mathbf{x}_2$ via $\mathbf{z}_2$ increases, following the order of `depth_to_space` operation. After mixing, mapping $\mathbf{z}_{\text{m}}$ back to $\mathcal{X}$ by g^{-1}, we obtain synthesized image $\mathbf{x}_{\text{m}} = g^{-1}(\mathbf{z}_{\text{m}})$. Fig. 2 illustrates the case of $D = 16$ and $\lambda = \frac{5}{16}$. Like the blue and green regions on $\mathbf{x}_{\text{m}}$ in Fig. 2, LS-Mix performs mixing in such a way that the parts of each source image are embedded spatially continuously in the generated $\mathbf{x}_{\text{m}}$. Indeed, other mixing ways can be thought of, and we evaluate four alternatives, including Linear interpolation and Bernoulli mixup (Bern-Mix) [12], in Sections 4 and 5.

4 Experiments

Preparation. We evaluated LS-Mix in classification accuracies, comparing to the *baseline* model (trained without MDA), MixUp, and CutMix. The datasets we used are SVHN [13], CIFAR-10/100 [14], and TinyImageNet [15], the details of which are shown in Appendix B.1. Following the previous works such as [16], we used two architectures for the classifier, the 13-layer CNN (CNN-13) and Wide-ResNet-28-10 (WRN-28-10) [1] [17]. The CNN-13 has been used in the literature such as [18, 19, 20, 21, 22, 10], whose architecture is described in Appendix B.2. We first trained only the Glow model separately from the classifier for each dataset, and we use the same model throughout all the experiments of classifier training. For the model parameters and the training settings, we would

[1] github.com/tensorflow/models/tree/master/research/autoaugment

Table 1: Error rates (%) with CNN-13. Best three are shown in bold.

	SVHN	CIFAR-10
baseline	2.46	5.00
MixUp	2.43	4.33
CutMix	2.69	4.40
Linear Intrpl. ($\alpha = 0.2$)	2.34	5.04
Linear Intrpl. ($\alpha = 0.4$)	2.33	4.94
Linear Intrpl. ($\alpha = 0.6$)	2.36	4.89
BernMix ($\alpha = 0.2$)	2.44	4.35
BernMix ($\alpha = 0.4$)	2.40	4.53
BernMix ($\alpha = 0.6$)	2.56	4.65
(ours)		
LS-Mix ($\alpha = 0.2$)	**2.27**	**4.21**
LS-Mix ($\alpha = 0.4$)	**2.25**	**4.01**
LS-Mix ($\alpha = 0.6$)	**2.31**	**3.99**

Table 2: Error rates (%) with WRN-28-10 in format of 'mean $\pm$ std'. Each experiment was run 3 times.

	CIFAR-10	CIFAR-100	TinyImageNet
baseline	3.61 ± 0.102	17.63 ± 0.143	33.57 ± 0.232
MixUp	2.61 ± 0.044	16.26 ± 0.117	32.81 ± 0.183
CutMix	2.57 ± 0.181	16.19 ± 0.470	31.92 ± 0.549
LS-Mix	$\mathbf{2.45} \pm 0.061$	$\mathbf{15.55} \pm 0.190$	$\mathbf{31.15} \pm 0.292$

Table 3: Error rates (%) for combination methods with WRN-28-10. Each experiment was run 3 times.

	CIFAR-10	CIFAR-100	TinyImageNet
MixUp + CutMix	2.03 ± 0.170	15.35 ± 0.290	31.01 ± 0.309
MixUp + LS-Mix	2.13 ± 0.108	15.19 ± 0.274	32.59 ± 0.295
CutMix + LS-Mix	$\mathbf{1.00} \pm 0.195$	$\mathbf{11.99} \pm 0.231$	$\mathbf{30.70} \pm 0.347$

like to refer the reader to Appendix B. MDA methods, including LS-Mix, have a hyper-parameter $\alpha \in (0, 1)$, based on which the mixing rate λ is sampled as $\lambda \sim \text{Beta}(\alpha, \alpha)$ where $\text{Beta}(\cdot)$ is a beta distribution. We tested $\alpha \in (0.1, 1.0)$ for MixUp, the full results of which are shown in Table 6 in Appendix C, and we picked the best ones, $\alpha = 0.7$, through the experiments. For CutMix, we used $\alpha = 1.0$ according to [3]. We run the experiments on a single NVIDIA Quadro P5000 GPU.

Results. We first evaluated LS-Mix with different α using CNN-13 classifier on SVHN and CIFAR-10. To compare the mixing scheme employed in LS-Mix with different mixing ways in $\mathcal{Z}$, we also evaluated two other methods that perform mixing in $\mathcal{Z}$, Linear interpolation (*Linear Intrpl*) and Bernoulli mixup (*Bern-Mix*) [12], Linear Intrpl performs linear interpolation in $\mathcal{Z}$. Mixing by Bern-Mix is written as Eq. 1, but unlike LS-Mix, the binary mask $\mathbf{M}$ in Bern-Mix is made at random by sampling from Bernoulli(λ) distribution. The results are shown in Table 1. The LS-Mix achieved the best results regardless of the value of α. Linear Intrpl and Bern-Mix were less effective than MixUp and CutMix on CIFAR-10. We present the synthesized images in Appendix E. We found that the images produced by Linear Intrpl are almost totally darkened entirely when λ is in a neighborhood around 0.5. The phenomenon of all-gray images concentrated in the center of $\mathcal{Z}$ has been reported in [23], and the result of Linear Intrpl is considered to be the same phenomenon. Also, we saw that Bern-Mix produces unnatural images, as described in [12].

Next, we evaluated the methods with WRN-28-10 classifier on CIFAR-10, CIFAR-100, and Tiny-ImameNet. We set α to 0.4 for LS-Mix, 0.7 for MixUp, and 1.0 for CutMix on all datasets. The results in Table 2 show that LS-Mix also outperformed other methods.

We also tested the performance of the combined use of LS-Mix, MixUp, and CutMix. As shown in Table 3, the combined use always improves the performance. In particular, LS-Mix + CutMix achieved remarkable improvement.

5 Discussion and Conclusion

Aside from Linear Intrpl and Bern-Mix, alternative mixing schemes in the latent space are possible. We introduced two more alternatives, *Swap-Mix* and *Reverse-Mix*. The mixing schemes and synthesized images are presented in Appendix D. We found that those two yield images drastically deformed from the source image and that their performance as MDA is far worse than LS-Mix, as shown in Table 4. This ablation study implies that complying with the squeezing dependency, as in LS-Mix, is important when performing mixes that replace components of the latent vectors.

Table 4: Error rates (%) with CNN-13. α is 0.4 for three mix methods.

	SVHN	CIFAR-10
baseline	2.46	5.00
Swap-Mix	4.39	5.20
Reverse-Mix	4.21	4.60
LS-Mix	**2.25**	**4.01**

Unlike other mixing methods in latent space (i.e., Linear Intrpl, Bern-Mix [12], Swap-Mix, and Reverse-Mix), LS-Mix is designed to preserve the original structure of both source images as much as possible and to stitch them naturally. That enables LS-Mix to yield natural images, and we empirically showed that such images work better as data augmentation. Although it is necessary to evaluate its effectiveness for large-size images such as ImageNet [15] in the future, this short paper demonstrated that the LS-Mix is promising as an MDA method.

Acknowledgements

The authors would like to thank Eric Jang for insightful discussions.

References

[1] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

[2] Yuji Tokozume, Yoshitaka Ushiku, and Tatsuya Harada. Between-class learning for image classification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[3] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[4] Oren Rippel and Ryan Prescott Adams. High-dimensional probability estimation with deep density models. *arXiv preprint arXiv:1302.5125*, 2013.

[5] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1530–1538, Lille, France, 07–09 Jul 2015. PMLR.

[6] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. In *International Conference on Learning Representations*, 2017.

[7] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

[8] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1278–1286, Bejing, China, 22–24 Jun 2014. PMLR.

[9] Huaibo Huang, zhihang li, Ran He, Zhenan Sun, and Tieniu Tan. Introvae: Introspective variational autoencoders for photographic image synthesis. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[10] Genki Osada, Budrul Ahsan, Revoti Prasad Bora, and Takashi Nishide. Regularization with latent space virtual adversarial training. In *Computer Vision – ECCV 2020*, pages 565–581, Cham, 2020. Springer International Publishing.

[11] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems 31*, pages 10215–10224. Curran Associates, Inc., 2018.

[12] Christopher Beckham, Sina Honari, Vikas Verma, Alex M Lamb, Farnoosh Ghadiri, R Devon Hjelm, Yoshua Bengio, and Chris Pal. On adversarial mixup resynthesis. In *Advances in Neural Information Processing Systems 32*, pages 4346–4357. Curran Associates, Inc., 2019.

[13] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[14] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

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

[16] Vikas Verma, Alex Lamb, Juho Kannala, Yoshua Bengio, and David Lopez-Paz. Interpolation consistency training for semi-supervised learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 3635–3641. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[17] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press, September 2016.

[18] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. Distributional smoothing with virtual adversarial training. In *International Conference on Learning Representations*, 2016.

[19] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in Neural Information Processing Systems 30*, pages 1195–1204. Curran Associates, Inc., 2017.

[20] Zihang Dai, Zhilin Yang, Fan Yang, William W Cohen, and Ruslan R Salakhutdinov. Good semi-supervised learning that requires a bad gan. In *Advances in Neural Information Processing Systems 30*, pages 6510–6520. Curran Associates, Inc., 2017.

[21] Ben Athiwaratkun, Marc Finzi, Pavel Izmailov, and Andrew Gordon Wilson. There are many consistent explanations of unlabeled data: Why you should average. In *International Conference on Learning Representations*, 2019.

[22] Sungrae Park, JunKeon Park, Su-Jin Shin, and Il-Chul Moon. Adversarial dropout for supervised and semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[23] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, and Balaji Lakshminarayanan. Detecting out-of-distribution inputs to deep generative models using typicality, 2020.

[24] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? In *International Conference on Learning Representations*, 2019.

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[26] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[27] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[28] Krzysztof Kolasinski. *An implementation of the GLOW paper and simple normalizing flows lib*, 2018.

[29] Jens Behrmann, Paul Vicol, Kuan-Chieh Wang, Roger Grosse, and Joern-Henrik Jacobsen. Understanding and mitigating exploding inverses in invertible neural networks. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 1792–1800. PMLR, 13–15 Apr 2021.

[30] Rob Cornish, Anthony Caterini, George Deligiannidis, and Arnaud Doucet. Relaxing bijectivity constraints with continuously indexed normalising flows. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2133–2143. PMLR, 13–18 Jul 2020.

[31] Alexandre Verine, Yann Chevaleyre, Fabrice Rossi, and benjamin negrevergne. On the expressivity of bi-lipschitz normalizing flows. In *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*, 2021.

A Normalizing Flow

Let $g(\cdot)$ be an invertible function and $\mathbf{h}_0$ and $\mathbf{h}_1$ be random variables of equal dimensionality. With the change of variables rule, a transformation $\mathbf{h}_1 = g(\mathbf{h}_0)$ can be written as the change in the probability density function (pdf): $p(\mathbf{h}_0) = p(\mathbf{h}_1)|\det(d\mathbf{h}_1/d\mathbf{h}_0)|$. Defining $\mathbf{h}_0 := \mathbf{x}$ and $\mathbf{h}_T := \mathbf{z}$, T-times repetition of this transformation, $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_T$, yields $\log p(\mathbf{x}) = \log p(\mathbf{z}) + \sum_{t=1}^{T} \log|\det(d\mathbf{h}_t/d\mathbf{h}_{t-1})|$, which gives us a invertible map between an image $\mathbf{x}$ and a correspondent latent vector $\mathbf{z}$. The target distribution $p(\mathbf{z})$, i.e., the latent space $\mathcal{Z}$, can be set to arbitrary distribution, but we choose a standard Gaussian $\mathcal{N}(0, \mathbf{I})$. Due to an invertible mapping, the dimensionality of $\mathcal{Z}$ is equal to that of the input space $\mathcal{X}$, which we denote as D. Through the training the flow model learns $g()$ so that it transforms the distribution of training images $p(\mathbf{x})$ into $\mathcal{N}(0, \mathbf{I})$. There are several types of flow models, but we use Glow [11], which is the most popular and has been used in many applications such as [24, 10].

A.1 Factor Out

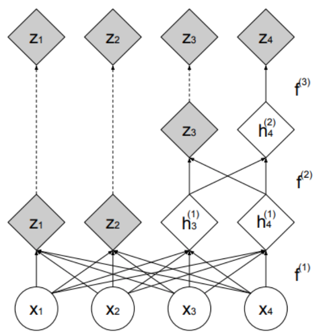


Figure 5: Illustration of factor out cited from [6]. f^i in figure corresponds to g^i in text.

To reduce computation cost and memory usage, flow models, including Glow, typically employ the mechanism called *factor out* [6, 11]. Fig. 5 shows the illustration cited from [6], and it is formalized as

$$(\mathbf{z}^{(i+1)}, \mathbf{h}^{(i+1)}) = g^{(i)}(\mathbf{h}^{(i)}) \tag{2}$$

$$\mathbf{z}^{(L+1)} = g^{(L)}(\mathbf{h}^{(L)}) \tag{3}$$

$$\mathbf{z} = (\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(L+1)}) \tag{4}$$

where L is the number of factor out operations. Also, $\mathbf{z}^{(i)}$ (and $\mathbf{h}^{(i)}$) and $g^{(i)}$ indicate the outputs of i-th operation and i-th transform function, respectively. For $i < L + 1$, the dimension of the outputs of $g^{(i)}$ are split in half (Eq. (2)). The splitting is not applied for $i = L$ (Eq. (3)). All $\mathbf{z}^{(i)}$ which have been factored out at different timing are concatenated to obtain the final output $\mathbf{z}$ (Eq. (4)). Thus, the components of $\mathbf{z}$ are gradually stacked in $L + 1$ times. For example, when $L = 4$ for 64×64 size RGB image datasets, $\mathbf{z}$ is composed of 5 levels as $[\mathbf{z}^1, \mathbf{z}^2, \mathbf{z}^3, \mathbf{z}^4, \mathbf{z}^5] = \mathbf{z}$, with dimensions of 6144, 3072, 1536, 768, and 768, respectively.

B Experimental Setup

B.1 Datasets

The SVHN dataset [13] consists of 32×32 pixel RGB images of real-world house numbers, having 10 classes. The CIFAR-10 dataset [14] also consists of 32×32 pixel RGB natural images in 10 different classes. Similarly, CIFAR-100 [14] has 100 classes. The TinyImageNet dataset [15] consists of 64×64 pixel RGB natural images of 200 classes. The numbers of training/test images are $73,257$/$26,032$ for SVHN, $50,000$/$10,000$ for CIFAR-10 and CIFAR-100, $100,000$/$10,000$ for TinyImageNet, respectively. We adopt the standard data-augmentation: random 2×2 translation to both datasets and horizontal flips to CIFAR-10/100 and TinyImageNet. The same augmentation is applied to the training of Glow.

B.2 Architecture of CNN-13 Classifier

Table 5: Architecture of CNN-13 classifier. BNorm stands for batch normalization. Slopes of all Leaky ReLU are set to 0.1.

Input: 32×32 RGB image	8: 2×2 max-pool, dropout 0.5
1: 3×3 conv. 128 same padding, BNorm, lReLU	9: 3×3 conv. 512 valid padding, BNorm, lReLU
2: 3×3 conv. 128 same padding, BNorm, lReLU	10: 1×1 conv. 256 BNorm, lReLU
3: 3×3 conv. 128 same padding, BNorm, lReLU	11: 1×1 conv. 128 BNorm, lReLU
4: 2×2 max-pool, dropout 0.5	12: Global average pool $6 \times 6 \rightarrow 1 \times 1$
5: 3×3 conv. 256 same padding, BNorm, lReLU	13: Fully connected $128 \rightarrow 10$
6: 3×3 conv. 256 same padding, BNorm, lReLU	14: BNorm (only for SVHN)
7: 3×3 conv. 256 same padding, BNorm, lReLU	15: Softmax

The architecture of CNN-13 is shown in Table 5.

B.3 Hyper-Parameters

Classifiers. We used the Adam optimizer [25] for the CNN-13 with the momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. For WRN-28-10, we use stochastic gradient descent with Nesterov momentum of 0.9. We trained with 200, 300, and 120 epochs for SVHN, CIFAR-10/100, and TinyImageNet, respectively. For CNN-13, the learning rate starts with 0.001 and exponentially decays with a rate 0.97 at every 2 epochs after the first 60,000 and 184,000 updates for SVHN and CIFAR-10, respectively. For WRN-28-10, we use a cosine learning decay, which is used in [26], starting with 0.1. The size of a mini-batch is 128 for CNN-13 and 50 for WRN-28-10.

Glow. We used the Adam optimizer with the momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The learning rate starts with 0.0001, and we trained $8,200$ iterations for both datasets, with batch size 256 for SVHN and CIFAR-10/100 and 16 for TinyImageNet, respectively. There are two major parameters to design the architecture: the depth of flow K and the number of factoring out operations L. For SVHN, CIFAR-10 and CIFAR-100, we chose $K = 32$ and $L = 3$. For TinyImageNet, we chose $K = 48$ and $L = 4$. The channel width of convolutions is 128 for all datasets. In accordance with [11], we train the Glow model on 5-bit images converted from the original 8-bit for high fidelity. We also would like to refer to our experimental code for the details. [2]

C Results of MixUp

Table 6: Error rates (%) of MixUp with CNN-13 on SVHN and CIFAR-10.

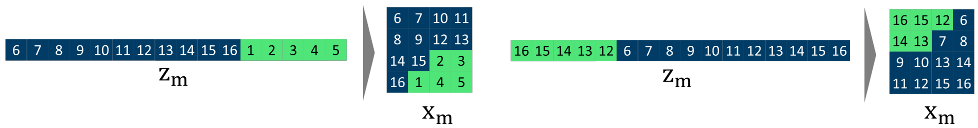
	α									
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
SVHN	2.51	2.53	2.58	2.54	2.54	2.48	**2.43**	2.54	2.46	2.56
CIFAR-10	4.92	4.65	4.65	4.36	4.65	4.40	**4.33**	4.47	4.44	**4.33**

We tested MixUp with different $\alpha \in (0.1, 1.0)$, and the results are shown in Table 6. For both datasets, the setting $\alpha = 0.7$ was the best, although $\alpha = 1.0$ achieved the same result on CIFAR-10.

D Alternative Mixture Methods in Latent Space

We introduce two alternative mixing schemes in the latent space, *Swap-Mix* and *Reverse-Mix*. The two alternative methods are depicted in Fig. 6 in correspondence with Fig. 2. The Swap-Mix creates the mask $\mathbf{M}$ in the same way as LS-Mix, but the components taken from each $\mathbf{z}_1$ and $\mathbf{z}_2$ are combined in the reverse order from LS-Mix. The Reverse-Mix sorts the components of $\mathbf{z}_1$ in reverse order before the selection of components is done. The images mixed by them are shown in Fig. 7, which

[2] We used TensorFlow 1.13. [27] and the experiments were run on NVIDIA Quadro P5000. The code for Glow is based on [28].



(a) Swap-Mix (b) Reverse-Mix

Figure 6: Illustration of alternative methods corresponding to one of LS-Mix in Fig. 2. In Swap-Mix, order of green and blue in $\mathbf{z}_{\mathrm{m}}$ is reversed from LS-Mix. In Reverse-Mix, order of green components in $\mathbf{z}_{\mathrm{m}}$ is reversed from LS-Mix. Consequently, ways to embed source images into $\mathbf{x}_{\mathrm{m}}$ differ.

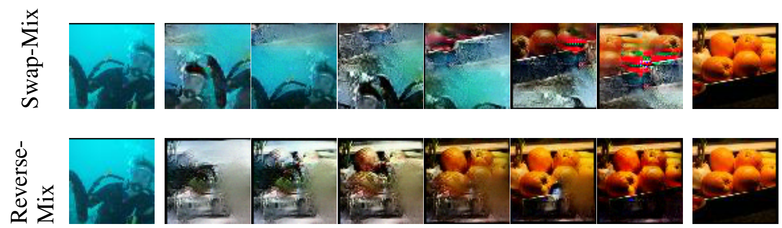
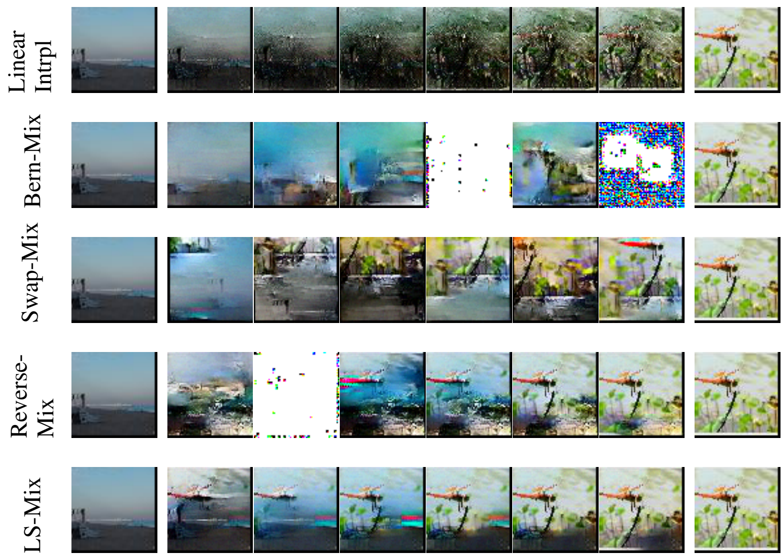


Figure 7: Mixed images with alternative methods. Images at both ends are source images which are same as Fig. 4. Images in middle are mixture of them with mixing rate with $\lambda = \{\frac{2}{16}, \frac{5}{16}, \frac{7}{16}, \frac{9}{16}, \frac{11}{16}, \frac{14}{16}\}$.

are drastically deformed while retaining the remnants of their source images. The results showed that Swap-Mix and Reverse-Mix are much less effective than LS-Mix as shown in Table 4. In fact, their performances are often even worse than the *baseline* model which was trained without any MDA. The poor performance of Swap-Mix and Reverse-Mix is probably caused by the fact that their mixed images deviate too much from the source images, as shown in Fig. 7. As Swap-Mix displaces the position of the source images horizontally and vertically on stitching, this effect is probably too strong for data augmentation. On the other hand, Reverse-Mix showed that replacing the latent components in reverse order immediately results in incomprehensible images to humans, regardless of mixing, as the images with $\lambda = \{\frac{2}{16}, \frac{5}{16}, \frac{7}{16}\}$ shown in Fig. 7. Moreover, we also found that, unlike LS-Mix, these methods often generated entire corrupted images as shown in Appendix E. The same corruptions were found in the images with Bern-Mix, and those are probably what is referred to as *inverse explosion* in [29], which is caused by the numerical errors that occur mainly due to a high Lipschitz constant of g and g^{-1} [30, 31]. We conjecture that for images that deviate from the training data more than a certain level, the flow model becomes unstable, i.e., Lipschitz constants become large, which leads to the generation of corrupted images due to inverse explosion.

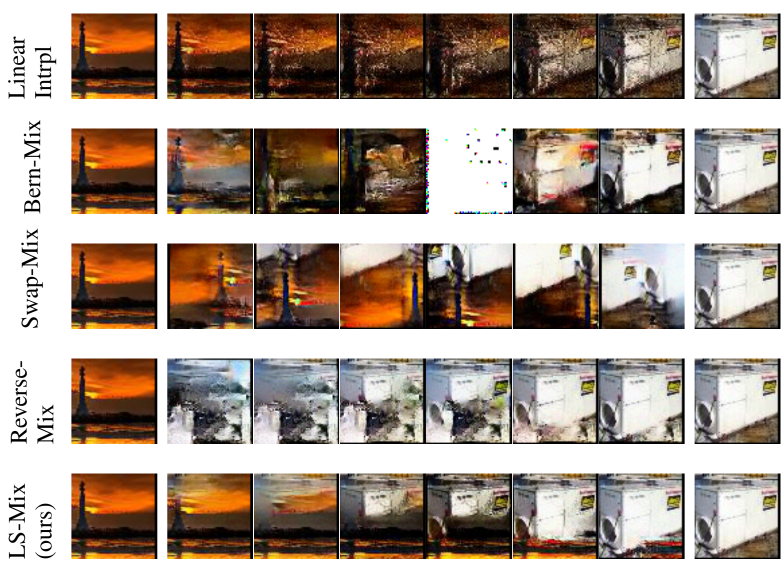
E More Samples

We show more samples as follows. We see that the mixed images with LS-Mix are more natural and stable than those of other methods. Columns at both ends are the source images, and the images in middle are mixture of them with mixing rate $\lambda = \{\frac{2}{16}, \frac{5}{16}, \frac{7}{16}, \frac{9}{16}, \frac{11}{16}, \frac{14}{16}\}$.


Linear Intrpl
Bern-Mix
Swap-Mix
Reverse-Mix
LS-Mix


Linear Intrpl
Bern-Mix
Swap-Mix
Reverse-Mix
LS-Mix (ours)


Linear Intrpl
Bern-Mix
Swap-Mix
Reverse-Mix
LS-Mix (ours)


Linear Intrpl
Bern-Mix
Swap-Mix
Reverse-Mix
LS-Mix (ours)