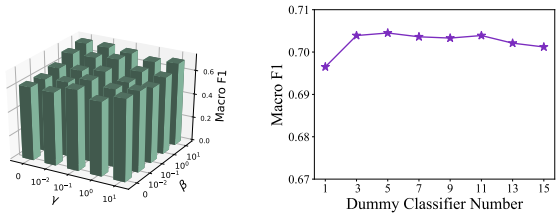
Supplementary Material for Learning Placeholders for Open-Set Recognition

Da-Wei Zhou Han-Jia Ye† De-Chuan Zhan
State Key Laboratory for Novel Software Technology, Nanjing University
{zhoudw, yehj}@lamda.nju.edu.cn, zhandc@nju.edu.cn

Abstract

*Open-set recognition is proposed to maintain classification performance on known classes and reject unknowns. We proposed to learn PlaceholdeRs for Open-SEt Recognition (*PROSER*), which prepares for the unknown classes by allocating placeholders for both data and classifier. In the supplementary, we describe the concrete settings for all the problems in the main paper together with additional experimental results. We also carefully discuss the advantages and disadvantages of vanilla mixup versus manifold mixup in novel instances generating.*



(a) Trade-off parameter (b) Dummy classifier number

Figure 1. Sensitivity of hyper-parameters on CIFAR100 dataset. We report macro F1 between 15 known classes and unknown.

1. Additional Experimental Results

This section introduces some additional experiment results and then gives the implementation details.

1.1. Sensitivity about Hyper-Parameters

In this section, we conduct experiments to explore the influence of hyper-parameters with the CIFAR100 dataset. The implementation details are the same as the ablations of the main paper. We choose 15 classes out of 100 as known ones, and another 15 out of 85 are open-set categories, making known-unknown ratio $1:1$. The performance is measured with macro F1 over 15 known classes and unknown.

In the main paper, the overall loss is described as:

$$l_{total} = l_1 + \gamma * l_2 \,, \tag{1}$$

where $l_1 = \sum_{(\mathbf{x},y)\in\mathcal{D}_{tr}} \ell(\hat{f}(\mathbf{x}), y) + \beta\ell(\hat{f}(\mathbf{x}) \setminus y, K+1)$ is the classifier placeholder loss, and $l_2 = \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{D}_{tr}} \ell([W, \hat{\boldsymbol{w}}]^\top \phi_{post}(\tilde{\mathbf{x}}_{pre}), K+1)$ is the data placeholder loss, yielding three different parts. In this section, we report the effects of these hyper-parameters β and γ as well as dummy classifier number C in Figure 1. Considering the trade-off parameters are adopted to balance the loss of each parts, we tune them in range of $\{0, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}\}$. As a result, we can get 25 results, corresponding to each combination from the set.

Note that we have provided an ablation study in the main paper, where 'Mixup' stands for only equipping the plain CNN with data placeholders, *i.e.*, $\gamma \to \infty$. Correspondingly, 'Dummy' stands for only training dummy classifiers, *i.e.*, $\gamma = 0$. The results in Figure 1(a) are consistent with the former conclusions that only employ part of the placeholder, *i.e.*, data or classifier, is not enough to produce the best performance. Additionally, we can observe that $\beta = 1, \gamma = 0.1$ leads to the best performance of the current task, which means a combination of data and classifier placeholders can jointly improve the model's performance. This also guides the hyper-parameters setup in other tasks. We also show the influence of classifier placeholder number C in Figure 1(b). Since we arrange 15 classes as the open-set class, we tune it in the range of $\{1, 3, \ldots, 15\}$. The results validate that multiple dummy classifiers increase the diversity of classifier placeholders, and can match novel patterns with the nearest classifier. However, learning too many dummy classifiers does not help open-set recognition, which shows a decline when $C > 11$.

1.2. Running Time Comparison

[9] point out that generative-based methods need more time in model training and instance generating. As a result,

†Correspondence to: Han-Jia Ye (yehj@lamda.nju.edu.cn)

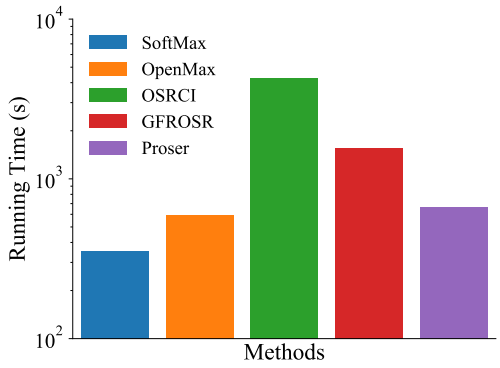


Figure 2. Running time comparison for different methods on MNIST dataset. The y-axis is in the logarithmic scale.

it takes more time to implement these methods in real-world applications. We conduct experiments on MNIST, and compare to [3, 6, 1] as well as softmax in terms of the training time. The results are reported in Figure 2. The running time of generative methods includes training generative models and novel instance generation.

From Figure 2, we can tell that PROSER is of the same order of magnitude with Softmax and OpenMax [1]. In comparison, generative-based methods OSRCI [3] and GFROSR [6] consume much more time than PROSER. OSRCI needs to generate counterfactual images as novelty, and augment the initial dataset with these generated open-set instances, which consumes the most time. GFROSR trains an extra generative model to reconstruct images, and the reconstructed images are fed into the classification model, which consumes the second-most time. Since we only generate open-set classes with manifold mixup, the mixup loss is based on mixed embedding, which means we do not need more steps in training novel patterns. As a result, PROSER can generate novel classes and train models efficiently.

1.3. Manifold Mixup VS Vanilla Mixup

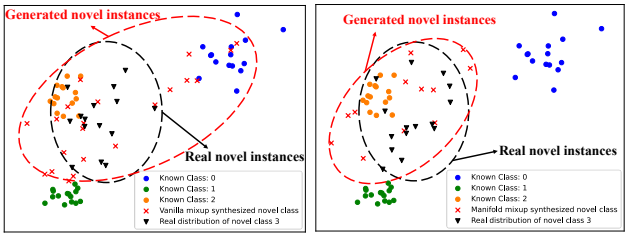
In the main paper, we discuss the pros and cons of manifold mixup [8] and the reason we do not adopt vanilla mixup in the input space. In this section, we give the detailed pseudo code for manifold mixup for data placeholders and the performance comparison between manifold mixup and vanilla mixup [10, 7].

The guideline of generating data placeholders with manifold mixup is shown in Algorithm 1. Comparing to vanilla training, where mini-batch instances are fed into the model to forward passing and conduct back-propagation, our proposed method consumes the same complexity of forwarding and backward passing. Line 1 forward the mini-batch with the pre-embedding module, and get the middle representation of the original batch. These middle-representations are

Algorithm 1 Manifold mixup for data placeholders

Input: Embedding module $\phi(\cdot)$, which can be decomposed of pre-embedding $\phi_{pre}(\cdot)$ and post-embedding $\phi_{post}(\cdot)$;
Closed-set mini-batch: $\mathcal{D}_{tr} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{B}$;
Output: Updated classifier $\hat{f}$;

1: Calculate the pre-embeddings of this mini-batch $\phi_{pre}(\mathcal{D}_{tr})$;
2: Shuffle the mini-batch with random order, and get shuffled (embedding,label) pair $\mathcal{D}_{\text{shuffle}} = \{(\phi_{pre}(\hat{\mathbf{x}}_i), \hat{y}_i)\}_{i=1}^{B}$;
3: **for** $i = 1, \cdots, B$ **do**
4: Mask the pairs of the same class, *i.e.*, $y_i = \hat{y}_i$;
5: **end for**
6: Sample λ from Beta distribution;
7: Calculate the manifold mixup pre-embeddings $\tilde{\mathbf{x}}_{pre}$ with unmasked pairs, *i.e.*, data placeholders;
8: Calculate the post-embeddings of $\tilde{\mathbf{x}}_{pre}$, *i.e.*, $\phi_{post}(\tilde{\mathbf{x}}_{pre})$;
9: Calculate the manifold mixup loss ← Eq. 7;



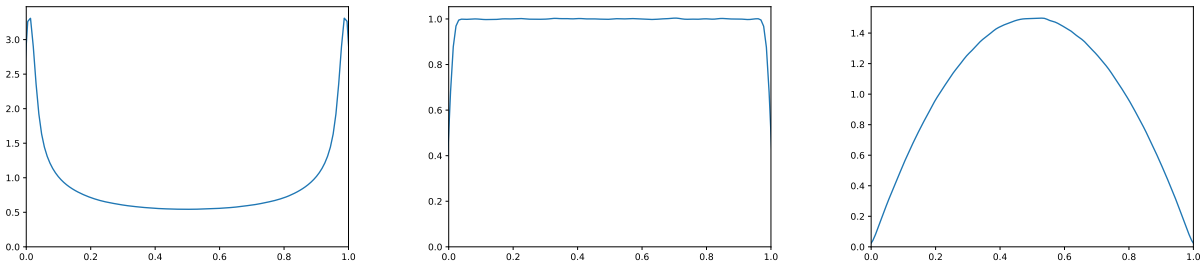
(a) Vanilla mixup (b) Manifold mixup

Figure 3. Visualization of generated novel instances and real distribution of novel instances. Left: novel instances generated by vanilla mixup. Right: novel instances generated by manifold mixup. Red crosses stand for generated novel instances, colored dots stand for known class instances, and black crosses stand for real distribution of novel class. The generated space of vanilla mixup covers known space, which may hurt the learning of embedding space.

then shuffled with random order to form $\mathcal{D}_{\text{shuffle}}$, as shown in Line 2. To avoid mixing two instances from the same class, we mask the pairs of the same class with Line 4, and then conduct mixup to generate data placeholders in Line 7.

Note that the size of mixed embeddings $\tilde{\mathbf{x}}_{pre}$ should be no more than B, since we only combine unmasked instances. These data placeholders are then fed into the post-embedding module to get the ultimate representation in Line 8. As a result, the total forward and backward consumption is no more than vanilla training.

We also test the performance comparison between manifold mixup and vanilla mixup, *i.e.*, replace the ϕ_{pre} into identity mapping and mixup in the input space. As we stated in the main paper, we argue that manifold mixup is optimiz-



(a) $\alpha = 0.4$ (b) $\alpha = 1$ (c) $\alpha = 2$

Figure 4. Kernel density estimation of Beta distribution when α varies.

Table 1. Ablation study over manifold mixup and vanilla mixup, the configurations are the same as in Figure 1, where 15 classes are selected from CIFAR100 as known classes and another 15 are open-set classes.

α	0.4	1	1.5	2
Vanilla Mixup	62.3	64.4	63.9	64.1
Manifold Mixup	**63.0**	**68.6**	**70.0**	**70.7**

able, and its benefits help the process of novelty generation. [8] proved that manifold mixup can move the decision boundary away from the data in all directions, which results in a compact embedding space. With the help of these data placeholders, the embeddings of known classes would be much tighter, thus leaving more place for embeddings of unknown classes. Moreover, we conduct ablations to validate the effectiveness of manifold mixup in Table 1. The experiment configuration is the same as Figure 1, and we tune different α in the Beta distribution to choose the best hyper-parameter α. Since the mixup percentage λ is influenced by α, we show the kernel density estimation with different α in Figure 4. We can infer that $\alpha < 1$ tends to sample λ near 0 or 1, while $\alpha > 1$ tends to sample λ near 0.5. Correspondingly, $\alpha = 1$ would result in a uniform distribution.

Let us return to our intuition where we try to synthesis novel patterns with known instances. Considering the places beside one class should not be a new class, while the middle point between two classes is often low-confidence area, an intuitive way we seek to mimic novel classes is to use $\alpha > 1$, as shown in Figure 4(c). To our relief, the results in Table 1 also validate the assumption that $\alpha = 2$ leads to the best performance. The results also indicate that compared to vanilla mixup, manifold mixup leads to a more compact embedding space, which boosts open-set recognition. The results also guide the setup of α in all experiments. We adopt $\alpha = 2$ in all experiments in the main paper without tuning the best task-specific value.

We also show the embedding of generated novel instances in Figure 3. We conduct experiments under the same setting as the visualization experiment part in the main paper, and show the embedding of generated instances by vanilla mixup and manifold mixup. The β parameter is the same between these two methods, and we can infer from Figure 3 that manifold mixup generates instances more similarly than the vanilla method. Besides, the generated space of vanilla mixup is much more than novel space, which also involves known space. As a result, utilizing vanilla mixup may destroy the learned embedding space to some extent.

2. Experiment Implementation

In this part, we introduce the implementation details, *i.e.*, the full results of unknown detection, hyper-parameter selection, model optimization, and dataset configuration.

2.1. Unknown Detection Results

In the main paper, we report the averaged AUC of unknown detection tasks. We simulate the sampling process over five trials [3] to report the mean and standard deviation, and the full results are shown in Table 2. We report the baseline performance from [6, 9, 3]. Note that N.R. in the table means that the original paper did not report the standard deviation.

2.2. Implementation Details of PROSER.

We employ the same backbone architecture as [6, 3]. PROSER is trained with SGD with momentum of 0.9, and the initial learning rate is set to 0.001 in the experiment. We fix the batch size to 128 for all datasets. As we discussed in the hyper-parameter part, we set $\beta = 1, \gamma = 0.1$, and the number of classifier placeholders is set to 5 for all datasets. The calibration $bias$ is obtained by ensuring 95% validation data be recognized as known. The α parameter in Beta distribution is set to 2 for all datasets. We conduct the experiment on Nvidia RTX 2080-Ti GPU with Pytorch 1.6.0 [5].

Table 2. Unknown detection performance in terms of AUC (mean±std). Results are averaged among five randomized trials. N. R. means the original work did not provide a particular value.

Methods	SVHN	CIFAR10	CIFAR+10	CIFAR+50	Tiny-ImageNet
Softmax	88.6 ± 1.4	67.7 ± 3.8	81.6 ± N. R.	80.5 ± N. R.	57.7 ± N. R.
OpenMax [1]	89.4 ± 1.3	69.5 ± 4.4	81.7 ± N. R.	79.6 ± N. R.	57.6 ± N. R.
G-OpenMax [2]	89.6 ± 1.7	67.5 ± 4.4	82.7 ± N. R.	81.9 ± N. R.	58.0 ± N. R.
OSRCI [3]	91.0 ± 1.0	69.9 ± 3.8	83.8 ± N. R.	82.7 ± N. R.	58.6 ± N. R.
C2AE [4]	89.2 ± 1.3	71.1 ± 0.8	81.0 ± 0.5	80.3 ± 0.0	58.1 ± 1.9
CROSR [9]	89.9 ± 1.8	N. R.	N. R.	N. R.	58.9 ± N. R.
GFROSR [6]	93.5 ± 1.8	83.1 ± 3.9	91.5 ± 0.2	91.3 ± 0.2	64.7 ± 1.2
PROSER	**94.3 ± 0.6**	**89.1 ± 1.6**	**96.0 ± 0.4**	**95.3 ± 0.3**	**69.3 ± 0.5**

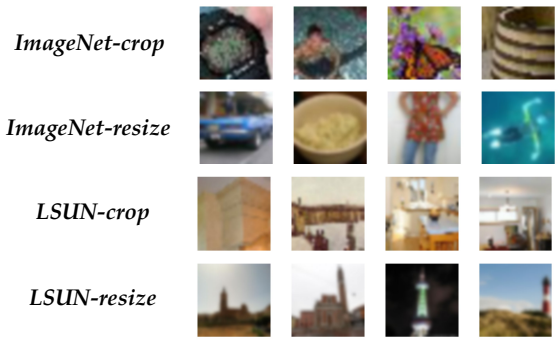


Figure 5. Dataset example of CIFAR10 open-set recognition, each line stands for the open-set class in main paper.

2.3. Dataset Configuration.

We also show the dataset example of CIFAR10 open-set recognition tasks in the main paper in Figure 5. The 'crop' datasets is part of the original picture, and 'resize' datasets is the full original picture resized into 32*32 pixel. As a result, detecting outliers from 'crop' datasets is easier than that of 'resize' datasets. This is consistent with macro-F1 results in the main paper.

References

[1] Abhijit Bendale and Terrance E Boult. Towards open set deep networks. In *CVPR*, pages 1563–1572, 2016. 2, 4

[2] Zongyuan Ge, Sergey Demyanov, Zetao Chen, and Rahil Garnavi. Generative openmax for multi-class open set classification. In *BMVC*, 2017. 4

[3] Lawrence Neal, Matthew Olson, Xiaoli Fern, Weng-Keen Wong, and Fuxin Li. Open set learning with counterfactual images. In *ECCV*, pages 613–628, 2018. 2, 3, 4

[4] Poojan Oza and Vishal M Patel. C2ae: Class conditioned auto-encoder for open-set recognition. In *CVPR*, pages 2307–2316, 2019. 4

[5] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8026–8037, 2019. 3

[6] Pramuditha Perera, Vlad I Morariu, Rajiv Jain, Varun Manjunatha, Curtis Wigington, Vicente Ordonez, and Vishal M Patel. Generative-discriminative feature representations for open-set recognition. In *CVPR*, pages 11814–11823, 2020. 2, 3, 4

[7] Yuji Tokozume, Yoshitaka Ushiku, and Tatsuya Harada. Between-class learning for image classification. In *CVPR*, pages 5486–5494, 2018. 2

[8] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In *ICML*, pages 6438–6447, 2019. 2, 3

[9] Ryota Yoshihashi, Wen Shao, Rei Kawakami, Shaodi You, Makoto Iida, and Takeshi Naemura. Classification-reconstruction learning for open-set recognition. In *CVPR*, pages 4016–4025, 2019. 1, 3, 4

[10] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018. 2