Supplementary Material

A. Implementation details

In this section, we provide implementation details that are not included in Section 4.1.

CIFAR-100-LT. To set up a fair comparison, we used the same random seed to make CIFAR-100-LT, and followed the implementation of [1]. We trained ResNet-32 [7] by SGD optimizer with a momentum of 0.9, and a weight decay of 2×10^{-4}. As in [1], we used simple data augmentation [7] by padding 4 pixels on each side and applying horizontal flipping or random cropping to 32×32 size. We trained for 200 epochs and used a linear warm-up of the learning rate [6] in the first five epochs. The learning rate was initialized as 0.1, and it was decayed at the 160th and 180th epoch by 0.01. The model was trained with a batch size of 128 on a single GTX 1080Ti. We turned off CMO for the last three epochs in order to finetune the model in the original input space.

For experiments in Table 11, we use the same strategy as for {CMO w/ Mixup}. For {CMO w/ Gaussian Blur} and {CMO w/ Color Jitter}, which do not mix two images, we divided classes into two groups: the majority and the minority. Then, for the minority group, we augmented the data with color jitter and gaussian blur, respectively. We set brightness to 0.5 and hue to 0.3 for color jitter, and set kernel size as $(5, 7)$ and sigma as $(0.1, 5)$ for Gaussian blur using the PyTorch [9] implemented functions.

ImageNet-LT. For ImageNet-LT, we followed most of the details from [11]. As in [11], we performed simple horizontal flips, color jittering, and took random crops 224×224 in size. We used ResNet-50 as a backbone network. The networks were trained with a batch size of 256 on 4 GTX 1080Ti GPUs for 100 epochs using SGD and an initial learning rate of 0.1; this rate decayed by 0.1 at 60 epochs and 80 epochs.

iNaturalist 2018. For iNaturalist 2018, we used the same data augmentation method as for ImageNet-LT. Multiple backbone networks were experimented on iNaturalist 2018, including ResNet-50, ResNet-101, ResNet-152 [7], and Wide ResNet-50 [12]. All backbone networks were trained with a batch size of 512 on 8 Tesla V100 GPUs for 200 epochs using SGD at an initial learning rate of 0.1; this rate decayed by 0.1 at 75 epochs and 160 epochs. Experiments were implemented and evaluated on the NAVER Smart Machine Learning (NSML) [8] platform.

B. Ablation studies

B.1. Comparison with oversampling methods

We compare CMO with other oversampling methods for performance improvement on CIFAR-100 with imbalance ratio 50 and 10 in Table 1. As in the imbalance ratio of 100, our method consistently improves performance in all long-tailed recognition methods.

Table 1. **Comparison against baselines on CIFAR-100-LT** Results with classification accuracy (%) of ResNet-32. The best results are marked in bold.

Imbalance ratio	50				10			
Method	Vanilla	+ROS [10]	+Remix [2]	+CMO	Vanilla	+ROS [10]	+Remix [2]	+CMO
CE	44.0 (+0.0)	39.7 (-4.3)	45.0 (+1.0)	**48.3 (+4.3)**	56.4 (+0.0)	55.6 (-0.8)	58.7 (+2.3)	**59.5 (+3.1)**
CE-DRW [1]	45.6 (+0.0)	41.3 (-4.3)	49.5 (+3.9)	**50.9 (+5.3)**	57.9 (+0.0)	56.4 (-1.5)	59.2 (+1.3)	**61.7 (+3.8)**
LDAM-DRW [1]	47.9 (+0.0)	38.3 (-9.6)	48.8 (+0.9)	**51.7 (+3.8)**	57.3 (+0.0)	53.9 (-3.4)	55.9 (-1.4)	**58.4 (+1.1)**
RIDE [11]	51.4 (+0.0)	31.3 (-20.1)	47.9 (-3.5)	**53.0 (+1.6)**	59.8 (+0.0)	49.4 (-10.4)	59.5 (-0.3)	**60.2 (+0.4)**

B.2. Results on longer training epochs

We evaluate CMO using the same setting from PaCo [5]. That is, we train the network for 400 epochs and use AutoAugment [3] on CIFAR-100-LT. For iNaturalist2018, RandAugment [4] is applied. Table 2, 3 reveals that {BS + CMO} surpasses PaCo in most cases, and achieves a new state-of-the-art performance. These results demonstrate the effectiveness of CMO, despite its simplicity.

Imbalance ratio	100	50	10
BS*	50.8	54.2	63.0
PaCo [5]*	**52.0**	56.0	64.2
BS + CMO	51.7	**56.7**	**65.3**

Table 2. **Classification Accuracy on CIFAR-100-LT with different imbalance ratios.** We train ResNet-32 with AutoAugment [3] in 400 epochs. $*$ is from [5] The best results are marked in bold.

	All	Many	Med	Few
BS*	71.8	**72.3**	72.6	71.7
PaCo [5]*	73.2	70.3	73.2	73.6
BS + CMO	**74.0**	71.9	**74.2**	**74.2**

Table 3. **Classification Accuracy on iNaturalist2018.** We train ResNet-50 for 400 epochs with RandAugment [4]. "$*$" indicates the results are from [5]. The best results are marked in bold.

B.3. Impact of α

We evaluate the impact of the hyperparameter $alpha$ in Figure 1. The classification accuracy according to different $\alpha \in \{0.1, 0.25, 0.5, 1.0, 2.0, 4.0\}$ is plotted. CMO improves the baseline accuracy (38.6%) in all cases. The best performance is achieved when $\alpha = 1.0$.

B.4. Computational cost

One of the biggest advantages of our method is its low computational cost. CMO only requires to load an additional batch of data from the minor-class-weighted loader. We measure the training time per batch on ImageNet-LT (see Table 4). While CE takes **0.355s**, CE+CMO takes **0.369s**, which is only an increase of 3.94%.

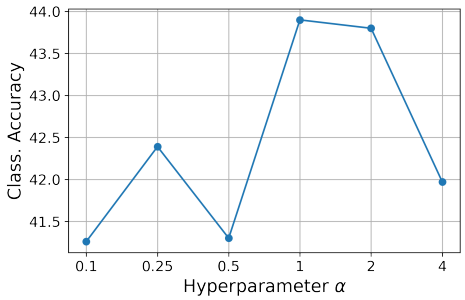

Figure 1. **Impact of** α on CIFAR-100-LT with an imbalance ratio of 100.

	CE	CE + CMO
Training Time (s)	0.355	0.369

Table 4. **Training time on ImageNet-LT.**

C. Pseudo-code of Context-rich Minority Oversampling

We present the PyTorch-syle pseudo-code of CMO algorithm in Algorithm 1. Note that CMO is easy to implement with just a few lines that are easily applicable to any loss, networks, or algorithms. Thus, CMO can be a very practical and effective solution for handling imbalanced dataset.

Algorithm 1 PyTorch-style pseudo-code for CMO

```
# original_loader: data loader from original data distribution
# weighted_loader: data loader from minor-class-weighted distribution
# model: any backbone network such as ResNet or multi-branch networks (RIDE)
# loss: any loss such as CE, LDAM, balanced softmax, RIDE loss

for epoch in Epochs:
   # load a batch for background images from original data dist.
   for x_b, y_b in original_loader:
       # load a batch for foreground from minor-class-weighted dist.
       x_f, y_f = next(weighted_loader)

       # get coordinate for random binary mask
       lambda = np.random.uniform(0,1)
       cx = np.random.randint(W) # W: width of images
       cy = np.random.randint(H) # H: height of images
       bbx1 = np.clip(cx - int(W * np.sqrt(1. - lambda))//2,0,W)
       bbx2 = np.clip(cx + int(W * np.sqrt(1. - lambda))//2,0,W)
       bby1 = np.clip(cy - int(H * np.sqrt(1. - lambda))//2,0,H)
       bby2 = np.clip(cy + int(H * np.sqrt(1. - lambda))//2,0,H)

       # get minor-oversampled images
       x_b[:, :, bbx1:bbx2, bby1:bby2] = x_f[:, :, bbx1:bbx2, bby1:bby2]
       lambda = 1 - ((bbx2 - bbx1) * (bby2 - bby1) / (W * H))# adjust lambda

       # output (x_f is attached to x_b)
       output = model(x_b)

       # loss
       losses = loss(output, y_b) * lambda + loss(output, y_f) * (1. - lambda)

       # optimization step
       losses.backward()
       optimizer.step()
```

References

[1] Kaidi Cao, Colin Wei, Adrien Gaidon, Nikos Arechiga, and Tengyu Ma. Learning imbalanced datasets with label-distribution-aware margin loss. In *Advances in Neural Information Processing Systems*, 2019.

[2] Hsin-Ping Chou, Shih-Chieh Chang, Jia-Yu Pan, Wei Wei, and Da-Cheng Juan. Remix: Rebalanced mixup. In *Computer Vision – ECCV 2020 Workshops*, 2020.

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

[4] Ekin Dogus Cubuk, Barret Zoph, Jon Shlens, and Quoc Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Advances in Neural Information Processing Systems*, 2020.

[5] Jiequan Cui, Zhisheng Zhong, Shu Liu, Bei Yu, and Jiaya Jia. Parametric contrastive learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[6] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, 2017.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[8] Hanjoo Kim, Minkyu Kim, Dongjoo Seo, Jinwoong Kim, Heungseok Park, Soeun Park, Hyunwoo Jo, KyungHyun Kim, Youngil Yang, Youngkwan Kim, et al. Nsml: Meet the mlaas platform with a real-world case study. *arXiv preprint arXiv:1810.09957*, 2018.

[9] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[10] Jason Van Hulse, Taghi M. Khoshgoftaar, and Amri Napolitano. Experimental perspectives on learning from imbalanced data. In *Proceedings of the 24th International Conference on Machine Learning*, 2007.

[11] Xudong Wang, Long Lian, Zhongqi Miao, Ziwei Liu, and Stella Yu. Long-tailed recognition by routing diverse distribution-aware experts. In *International Conference on Learning Representations*, 2021.

[12] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2016.