

APPENDIX A NOISY SAMPLE VISUALIZATION

In Figure 3 we show example images randomly chosen from the out-of-distribution samples filtered out by our method. In Figure 4 we show random examples where their pseudo-labels are different from the original training labels. By visual examination, we observe that our method can remove OOD samples and correct noisy labels at a high success rate.

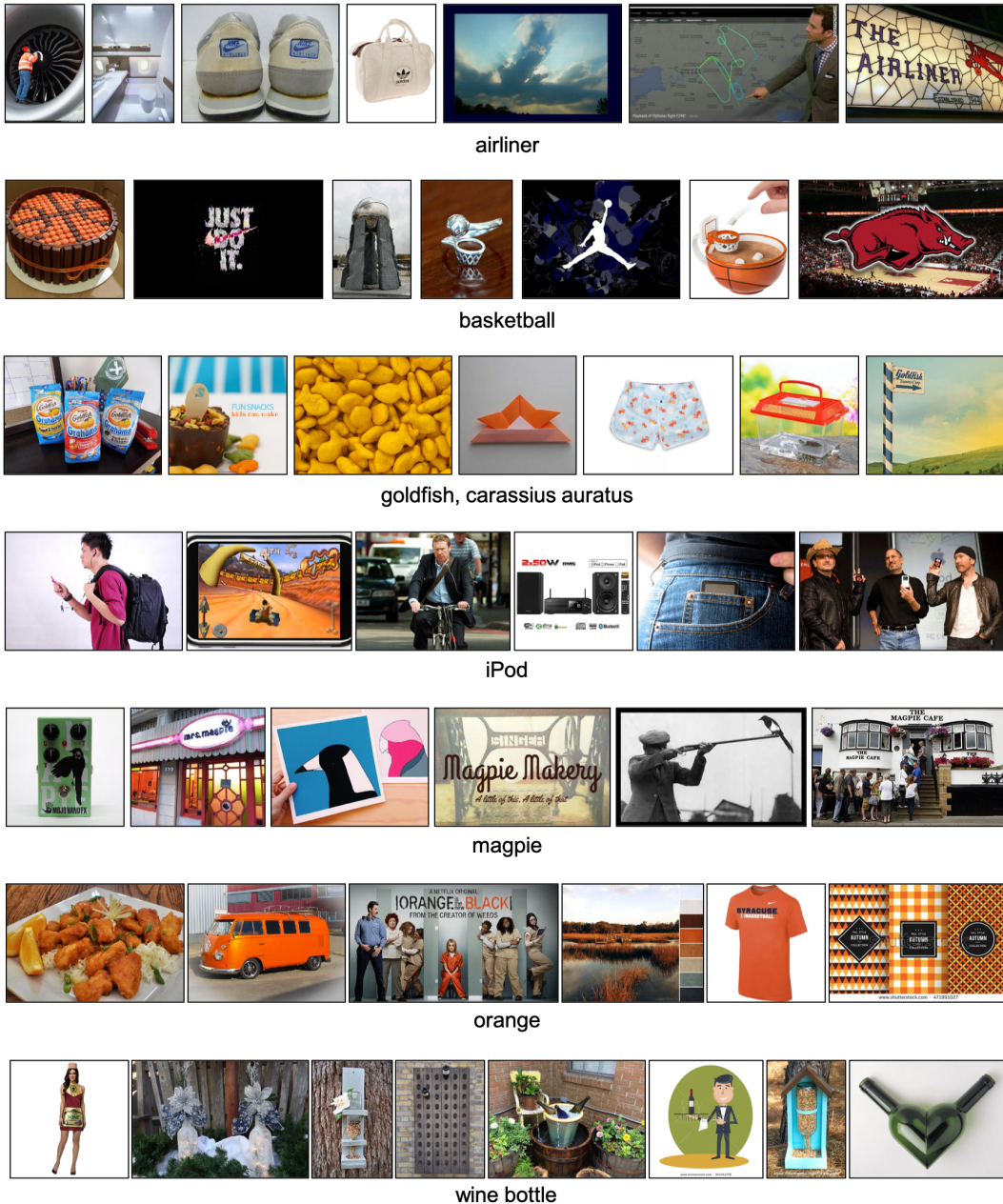


Figure 3: Examples of randomly selected out-of-distribution samples filtered out by our method. The original training labels are shown below the images.

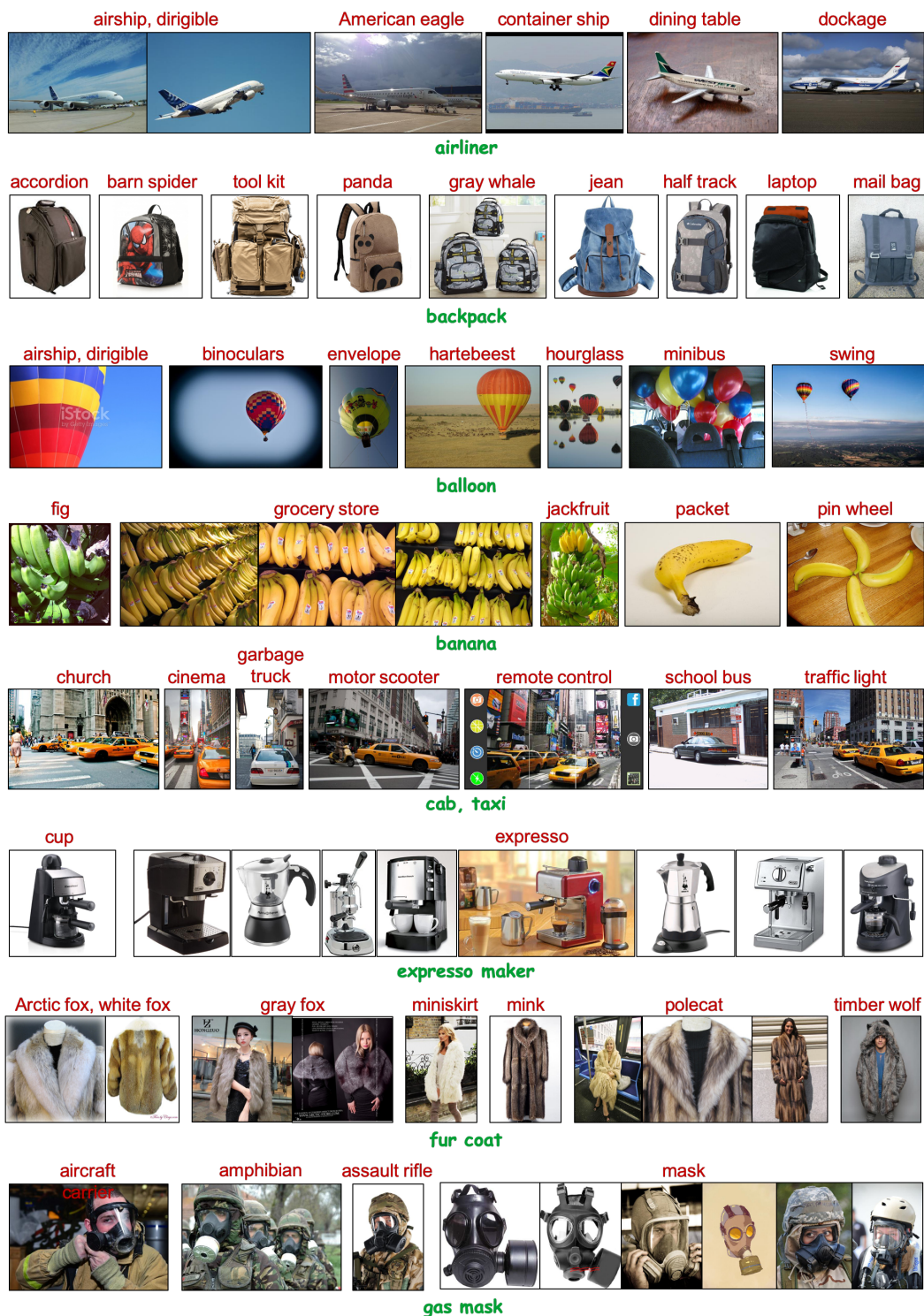


Figure 4: Examples of randomly selected samples with noisy labels corrected by our method. The original training labels are shown in red and corrected pseudo-labels are shown in green.

APPENDIX B PSEUDO-CODE OF MOPro

Algorithm 1 summarizes the proposed method.

Algorithm 1: MoPro’s main algorithm.

```

1 Input: number of classes  $K$ , temperature  $\tau$ , threshold  $T$ , momentum  $m$ , encoder network  $f(\cdot)$ ,
  projection network  $g(\cdot)$ , classifier  $h(\cdot)$ , momentum encoder  $g'(f'(\cdot))$ .
2 for  $\{(x_i, y_i)\}_{i=1}^b$  in loader do           // load a minibatch of noisy training data
3   for  $i \in \{1, \dots, b\}$  do
4      $\tilde{x}_i = \text{weak\_aug}(x_i)$                        // weak augmentation
5      $\tilde{x}'_i = \text{strong\_aug}(x_i)$                    // strong augmentation
6      $v_i = f(\tilde{x}_i)$                                // representation
7      $z_i = g(v_i)$                                // normalized low-dimensional embedding
8      $z_i = g'(f'(\tilde{x}'_i))$                          // momentum embedding
9      $p_i = h(v_i)$                                // class prediction
10     $s_i = \{s_i^k\}_{k=1}^K, s_i^k = \frac{\exp(z_i \cdot c_k / \tau)}{\sum_{k=1}^K \exp(z_i \cdot c_k / \tau)}$  // prototypical score
    // noise correction
11     $q_i = (p_i + s_i) / 2$                          // soft pseudo-label
12    if  $\max_k q_i^k > T$  then
13       $\hat{y}_i = \arg \max_k q_i^k$ 
14    else if  $q_i^{y_i} > 1/K$  then
15       $\hat{y}_i = y_i$ 
16    else
17       $\hat{y}_i = \text{OOD}$ 
18    end
    // calculate losses
19     $\mathcal{L}_{\text{ins}}^i = -\log \frac{\exp(z_i \cdot z'_i / \tau)}{\sum_{r=0}^R \exp(z_i \cdot z'_r / \tau)}$  // instance contrastive loss
20    if  $\hat{y}_i$  is not OOD then
21       $\mathcal{L}_{\text{pro}}^i = -\log \frac{\exp(z_i \cdot c_{\hat{y}_i} / \tau)}{\sum_{k=1}^K \exp(z_i \cdot c_k / \tau)}$  // prototypical contrastive loss
22       $\mathcal{L}_{\text{ce}}^i = -\log(p_i^{\hat{y}_i})$  // cross entropy loss
23    else
24       $\mathcal{L}_{\text{pro}}^i = \mathcal{L}_{\text{ce}}^i = 0$ 
25    end
    // update momentum prototypes
26     $c_{\hat{y}_i} \leftarrow \text{Normalize}(m c_{\hat{y}_i} + (1 - m) z_i)$ 
27  end
28   $\mathcal{L} = \sum_{i=1}^b (\mathcal{L}_{\text{ce}}^i + \mathcal{L}_{\text{pro}}^i + \mathcal{L}_{\text{ins}}^i)$  // total loss
29  update networks  $f, g, h$  to minimize  $\mathcal{L}$ .
30 end

```

APPENDIX C TRANSFER LEARNING IMPLEMENTATION DETAILS

For low-shot image classification on Places and VOC, we follow the procedure in Li et al. (2020b) and train linear SVMs on the global average pooling features of ResNet-50. We preprocess all images by resizing to 256 pixels along the shorter side and taking a 224×224 center crop. The SVMs are implemented in the LIBLINEAR (Fan et al., 2008) package.

For low-resource finetuning on ImageNet, we adopt different finetuning strategy for different versions of WebVision pretrained models. For WebVision V0.5 and V1.0, since they contain the same 1000 classes as ImageNet, we finetune the entire model including the classification layer. We train with SGD, using a batch size of 256, a momentum of 0.9, a weight decay of 0, and a learning rate of 0.005. We train for 40 epochs, and drop the learning rate by 0.2 at 15 and 30 epochs. For WebVision 2.0, since it contains 5000 classes, we randomly initialize a new classification layer with 1000 output

dimension, and finetune the model end-to-end. We train for 50 epochs, using a learning rate of 0.01, which is dropped by 0.1 at 20 and 40 epochs.

For object detection and instance segmentation on COCO, we adopt the same setup in MoCo (He et al., 2019), using Detectron2 (Girshick et al., 2018) codebase. The image scale is in [640, 800] pixels during training and is 800 at inference. We fine-tune all layers end-to-end. We finetune on the train2017 set ($\sim 118k$ images) and evaluate on val2017.

APPENDIX D STANDARD DEVIATION FOR LOW-SHOT CLASSIFICATION

Table 7 reports the standard deviation for the low-shot image classification experiment in Section 5.1

Method	Pretrain dataset	VOC07				Places205			
		$k=1$	$k=2$	$k=4$	$k=8$	$k=1$	$k=2$	$k=4$	$k=8$
CE (Sup.)	ImageNet	54.3 \pm 4.8	67.8 \pm 4.4	73.9 \pm 0.9	79.6 \pm 0.8	14.9 \pm 1.3	21.0 \pm 0.3	26.9 \pm 0.6	32.1 \pm 0.4
MoPro	WebVision-V1.0	59.5 \pm 5.2	71.3 \pm 2.2	76.5 \pm 1.1	81.4 \pm 0.6	16.9 \pm 1.3	23.2 \pm 0.3	29.2 \pm 0.6	34.5 \pm 0.3
MoPro	WebVision-V2.0	64.8 \pm 6.7	74.8 \pm 2.6	79.9 \pm 1.4	83.9 \pm 1.0	22.2 \pm 1.3	29.2 \pm 0.5	35.6 \pm 0.7	40.9 \pm 0.3

Table 7: Low-shot image classification experiments. Mean and standard deviation are calculated across 5 runs.