Supplementary Material
CAPro: Webly Supervised Learning with Cross-modality Aligned Prototypes

Yulei Qin[1] **Xingyu Chen**[2] **Yunhang Shen**[1] **Chaoyou Fu**[1]

Yun Gu[3] **Ke Li**[1] **Xing Sun**[1] **Rongrong Ji**[4]

[1]Tencent YouTu Lab [2]ByteDance
[3]Shanghai Jiao Tong University [4]Xiamen University
yuleiqin@tencent.com

S1 Datasets Details

S1.1 WebVision1k

It contains 2.4M web images collected from Google and Flickr, which share the same 1k category names with ImageNet1k [1]. For each example, we use all available description, title, and tag in its metadata for raw text preparation. Besides, we follow [2, 3] to use the subset of WebVision–**Google500** for ablation studies in consideration of lower GPU resource and time consumption without losing generalization. It contains 0.48M images from Google with randomly chosen 500 categories. The testing set of ImageNet1k and its subset ImageNet500 are involved as well for evaluation.

S1.2 NUS-WIDE (Web)

It contains 0.26M web images from Flickr with 5k unique user tags. Each example is manually annotated with multiple labels within 81 concepts that are filtered out of the 5k tags. It also provides weak labels (an official web version) by checking if each of the 81 category name appears in user tags of every example. Almost 50% of the web labels are wrong and 50% of the true labels are missing in tags [4]. Since tags contain phrases without delimiters, we split phrases based on unigram frequencies [5] for raw text preparation. We follow [3] to train models with weakly-labeled training set (*a.k.a.*, NUS-WIDE Web) and validate them on the clean testing set.

S2 Mathmatical Notations

In this section, we present the description of all math notations in the manuscript (see Table S1).

S3 Implementation and Training Details

S3.1 General Settings

In consideration of performance and efficiency, we adopt ResNet-50 [6] and MiniLM-L6 [7] as image and text encoders by default. The training settings are listed in Table S2. In general, we refer to [8] for: batch size is 256; optimizer is SGD; learning rate linearly rises with 10 warm-up epochs and decays by cosine schedule. Specifically, although the benefit of an increased batch size (*e.g.*, 1024) has been validated [3, 9], we follow the standard batch size setting of 256 on WebVision1k for two

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

Table S1: List of symbols.

Symbol	Description
$\mathbf{x}_i$	a web image indexed with i
$\mathbf{t}_i$	textual metadata associated with $\mathbf{x}_i$
y_i	web label associated with $\mathbf{x}_i$
y_i^*	ground-truth label associated with $\mathbf{x}_i$
N	the total number of images in a web dataset
C	the total number of categories
D	a web dataset
θ_e	parameters of image encoder
θ_c	parameters of classifier
$\mathcal{F}(\theta_e;\theta_c)$	a deep model with image encoder and classifier
d_v	dimension of the visual features
$\mathbf{v}_i$	visual features of $\mathbf{x}_i$ extracted by image encoder, $\mathbf{v}_i \in \mathbb{R}^{d_v}$
d_t	dimension of the textual embeddings
$\mathbf{s}_i$	textual embeddings of $\mathbf{t}_i$ extracted by text encoder, $\mathbf{s}_i \in \mathbb{R}^{d_t}$
$\mathbf{t}^c$	category definition of the c-th class
$\mathbf{s}^c$	textual embeddings of $\mathbf{t}^c$ extracted by text encoder, $\mathbf{s}^c \in \mathbb{R}^{d_t}$
$\mathbf{p}_i$	predictions on $\mathbf{v}_i$ from classifier, $\mathbf{p}_i \in \mathbb{R}^C$, $\mathbf{p}_{i(k)}$ denotes its k-th element
d_p	dimension of the visual embeddings
$\mathbf{z}_i$	low-dimensional embeddings of $\mathbf{v}_i$ after projection, $\mathbf{z}_i \in \mathbb{R}^{d_p}$
$\tilde{\mathbf{v}}_i$	reconstructed visual features from $\mathbf{z}_i$, $\tilde{\mathbf{v}}_i \in \mathbb{R}^{d_v}$
$\mathbf{q}_i$	predictions on $\mathbf{z}_i$ from auxiliary classifier, $\mathbf{q}_i \in \mathbb{R}^C$
Q	the size of dictionary, namely the length of queue
k	number of nearest neighbors (NN) in k-NN and k-reciprocal-NN
$\mathcal{V}$	vertices, nodes
$\mathcal{E}$	edges
$\mathcal{G}$	graph, $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$
A	adjacency matrix
$\mathcal{N}(\mathbf{x}_i, k)$	k-NN sets of $\mathbf{x}_i$
$\mathcal{R}(\mathbf{x}_i, k)$	k-reciprocal-NN sets of $\mathbf{x}_i$
$d(\mathbf{v}_i, \mathbf{v}_j)$	distance between $\mathbf{v}_i$ and $\mathbf{v}_j$
$d^*(\mathbf{v}_i, \mathbf{v}_j)$	refined distance between $\mathbf{v}_i$ and $\mathbf{v}_j$
$V_{\mathbf{v}_i,\mathbf{v}_j}$	k-reciprocal feature encoding the distance between $\mathbf{v}_i$ and $\mathbf{v}_j$
$\mathbf{S}$	concatenated textual embeddings, $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_N), \mathbf{S} \in \mathbb{R}^{N \times d_t}$
I_N	identity matrix
$\tilde{A}$	adjacency matrix A with self-connection I_N
$\tilde{D}_{ii}$	degree matrix of $\tilde{A}$
$\hat{\mathbf{S}}$	denoised $\mathbf{S}$ after smoothing
$\hat{\mathbf{s}}_i$	denoised $\mathbf{s}_i$ after smoothing
K	top-K selected examples with visual-semantic alignment
σ_K^c	the K-th smallest distance $d^*(\hat{\mathbf{s}}_i, \mathbf{s}^c)$ in the c-th class
D_K^c	top-K set of the c-th class
D_K	sets of top-K examples from all classes
$\hat{\mathbf{z}}^c$	unnormalized visual prototype of the c-th class
$\mathbf{z}^c$	normalized visual prototype of the c-th class
τ	temperature coefficient
α	weight between self-prediction and prototype-instance similarity
$\mathbf{o}_i$	fused, comprehensive output for label correction, $\mathbf{o}_i \in \mathbb{R}^C$
$\mathbf{r}_i$	prototype-instance similarity, $\mathbf{r}_i \in \mathbb{R}^C$, $\mathbf{r}_{i(k)}$ denotes its k-th element
γ	threshold for label correction
m_p	momentum coefficient
$\mathbf{b}_i$	collective bootstrapping target on $\mathbf{z}_i$
w_{ij}	weight of contribution from $\mathbf{z}'_j$ in the dictionary for $\mathbf{b}_i$
λ^{bts}	weight for collective bootstrapping loss
λ^{prj}	weight for projection and reconstruction losses
λ^{pro}	weight for prototypical contrastive loss
λ^{ins}	weight for instance contrastive loss

Table S2: List of hyper-parameters for training settings.

Settings	WebVision1k/Google500	NUS-WIDE
Optimizer	SGD	
Optimizer momentum	0.9	
Optimizer weight decay	1×10^{-4}	
Batch size	256	
Step1 pre-training scheduler	Cosine decay with linear warm-up	
Step1 pre-training warm-up epochs	5/10	10
Step1 pre-training learning rate	1×10^{-1}	2×10^{-3}
Step1 pre-training epochs with encoders frozen	0	
Step1 pre-training epochs in total	120	100
Step2 training scheduler	Cosine decay with linear warm-up	
Step2 training warm-up epochs	5/10	10
Step2 training learning rate	1×10^{-1}	2×10^{-3}
Step2 training epochs with encoders frozen	5/10	10
Step2 training epochs in total	60/120	100
Step3 fine-tuning scheduler	Cosine decay	
Step3 fine-tuning warm-up epochs	0	
Step3 fine-tuning learning rate	1×10^{-4}	2×10^{-5}
Step3 fine-tuning epochs with encoders frozen	15/20	20
Step3 fine-tuning epochs in total	15/20	20
Image encoder (by default)	ResNet-50 [6]	
Text encoder (by default)	MiniLM-L6 [7]	
λ^{prj}	1	
λ^{pro}	1	
λ^{ins}	1	
λ^{bts}	0.1	
m_p	0.999	
d_p	128	
τ	0.1	
α	0.5	
top-K	50	
Q	8192	2048
γ	0.6	0.9
k-NN/k-reciprocal-NN	5	10

reasons: 1) comparability with most of the previous methods; 2) limited computing resources with 8 GPUs (a mini-batch size of 32 on each GPU for a batch size of 256 in total).

We empirically set $\lambda^{prj} = 1$, $\lambda^{pro} = 1$, $\lambda^{ins} = 1$, $\lambda^{bts} = 0.1$, $m_p = 0.999$, $d_p = 128$, $\tau = 0.1$, $\alpha = 0.5$, and $K = 50$ by default. Their optimal values require meticulous fine-tuning of each hyper-parameter, which is beyond consideration of the present study.

Data augmentation (random cropping, resacling, and horizontal flipping) is applied on the inputs to the query encoder while stronger augmentation, including color jittering and blurring [10]), is added on those to the key encoder.

Experiments are conducted on a CentOS 7 workstation with an Intel 8255C CPU, 377 GB Mem, and 8 NVIDIA V100 GPUs. The training of CAPro on WebVision1k, Google500, and NUS-WIDE respectively costs about ten days, three days, and one day under the environment mentioned above.

S3.2 WebVision1k-only Implementation

We refer to [8] for $Q = 8192$ and $\gamma = 0.6$. The learning rate is 0.1 and the number of epochs is 120. Given the complexity of graph construction, we use $k = 5$ for both text denoising and matching.

S3.3 NUS-WIDE (Web)-only Implementation

In view of the dataset scale, we set $Q = 2048$ with a learning rate of 2×10^{-3} and a total of 100 epochs. $k = 10$ is chosen since user tags are noiser and sparser in NUS-WIDE.

To support multi-label learning, it is necessary, but not yet enough, to simply replace the softmax-based cross-entropy losses with sigmoid-based binary cross-entropy losses. The premise of prototypical contrastive learning does not hold true anymore because one instance can be simultaneously engaged in formation of multiple clusters, which violates the exclusivity inherited in softmax activation. Our experiments demonstrate that, only by projecting $\mathbf{v}_i$ into compact subspaces specific to each class, can we properly learn the decision boundary to continue prototypical learning. Technically, we set C additional fully-connected (FC) layers after the projector to respectively map $\mathbf{z}_i$ into $\tilde{\mathbf{z}}_{i,c} \in \mathbb{R}^{d_p}, c = 1, 2, ..., C$. For the c-th class, both positive and negative prototypes $\tilde{\mathbf{z}}^{c+}, \tilde{\mathbf{z}}^{c-}$ are shaped accordingly for the contrast against $\tilde{\mathbf{z}}_{i,c}$. Such operation can be viewed as magnifying class-indicative contents via recombination of the shared embedding $\mathbf{z}_i$. Another minor modification is required on noise removal, where the output $\mathbf{o}_{i,c}$ is fused independently for the c-th class for binary separation. Considering the overwhelming negative instances, we set $\gamma = 0.9$ to avoid deviation by majorities in label rectification. Discussion on the hyper-parameter γ can be found in Sec. S4.2.

S3.4 Training Steps

CAPro adopts a three-step training pipeline. It employs a pre-training step to learn common visual patterns from the original web dataset D. Then, the training starts with instance-prototype and instance-wise contrastive learning, collective bootstrapping, and on-line noise removal. Finally, given the trained model, off-line noise removal is performed on D for data cleaning and we fine-tune the classifier alone with the cleaned dataset.

Step1 pre-training We perform visual pre-training on ResNet-50 with cross-entropy and projection-reconstruction losses. At this time, we only use original web labels to train the model to learn common visual description, which lays foundation for the subsequent prototype initialization in step2. Note that the **pre-trained** model is also the **vanilla** method in our experiments.

Step2 training All components are initialized with the pre-trained parameters in step1. As shown in Table S2, we keep the encoders frozen with warm-up epochs. This helps stabilize training to avoid the prototypical embeddings, which are initialized at the beginning by averaging top-K semantically-correct examples, being perturbed drastically. In this step, we re-train the model with all losses. Apart from classification, we perform instance-wise and instance-prototype contrastive learning, collective bootstrapping, noise removal, and prototype update.

Step3 fine-tuning We follow MoPro [8] to perform noise removal on the training dataset D. The trained model in step2 is used to correct labels or discard OOD examples with our control flow. Then, we keep encoders frozen and fine-tune the classifier alone with the cleaned set for better performance. Note that such **fine-tuned** model is exactly the **CAPro** in experiments.

We also prepare Algo. S1 to explicitly explain the entire training process.

S4 Ablation Study

S4.1 Effect of Text Enhancement

Figs. S1 and S2 present additional qualitative comparison for selecting instances with potentially-correct semantics. For WebVision, noiser categories are chosen to validate the effectiveness of text enhancement by smoothing and reranking. We can observe that due to the problem of polysemy, a majority of the retrieved images are irrelevant to the correct semantics and simple k-NN-based smoothing in [3] can hardly handle such situation. In contrast, our text enhancement helps pinpoint, not perfect but comparatively reliable, web instances that share similar semantics (*e.g.*, metalwork in *Nail*). Besides, we also sample three categories from the noiser NUS-WIDE to double-check the effectiveness of text enhancement. For example, in the category of airport, direct matching of user tag embeddings to the textual prototype returns a few close-up images of warcrafts, which has nothing to do with *Airport*. On the contrary, our text enhancement helps to select the truly matched instances.

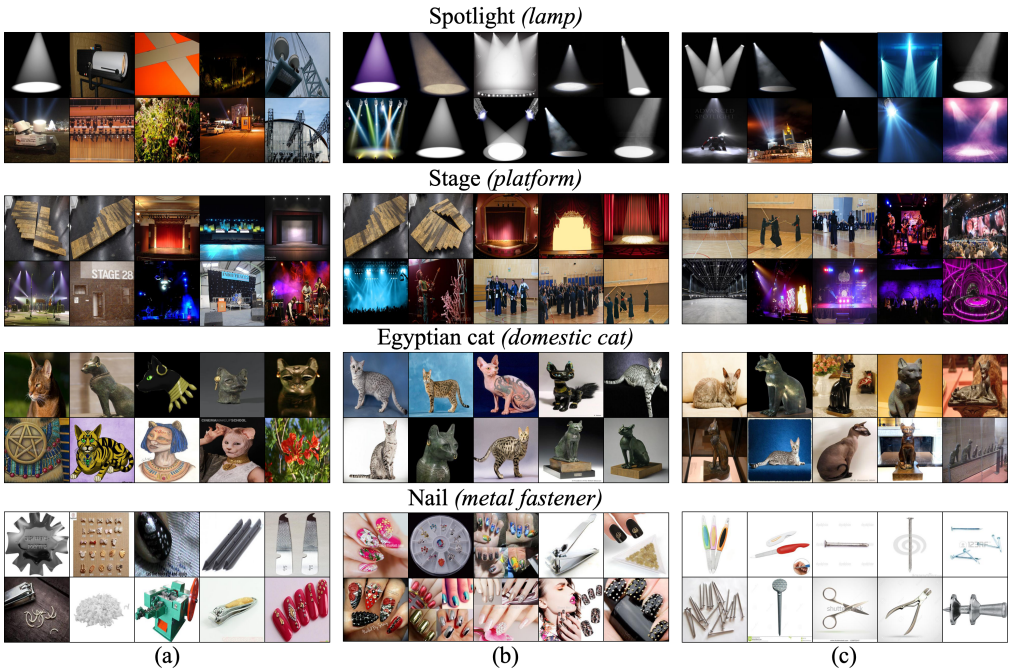

Figure S1: Top-matched WebVision1k instances are chosen: (a) without text enhancement, (b) with text enhancement in VSGraph [3], and (c) with our text enhancement.

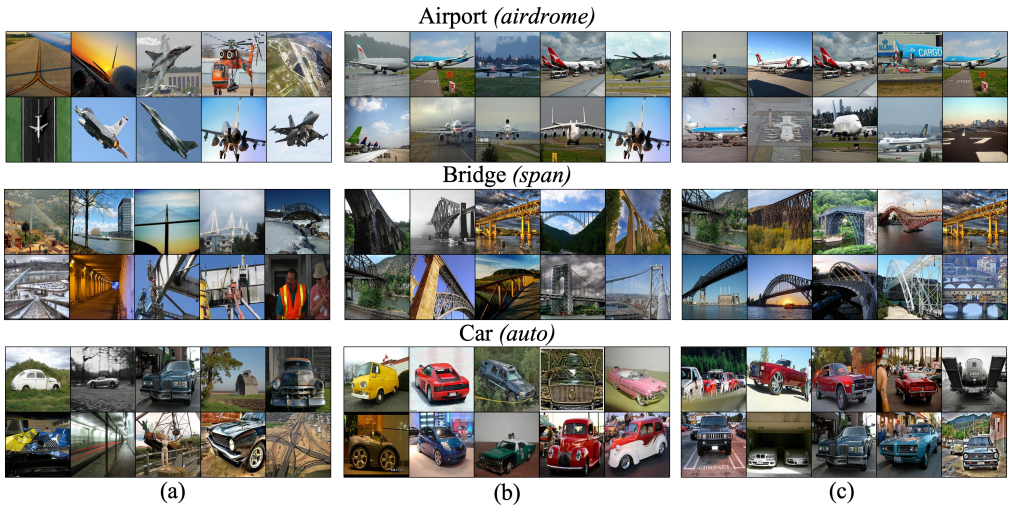

Figure S2: Top-matched NUS-WIDE (Web) instances are chosen: (a) without text enhancement, (b) with text enhancement in VSGraph [3], and (c) with our text enhancement.

Algorithm S1: CAPro's training procedure.

Data: Web images and their associated texts and labels $D = \{(\mathbf{x}_i, \mathbf{t}_i, y_i)\}_{i=1}^{N}$.

Step1 pre-training
for $(\mathbf{x}_i, y_i) \in D$ **do**
  $\mathcal{L}_i = \mathcal{L}_i^{\text{cls}} + \mathcal{L}_i^{\text{prj}}$;
  Update image encoder, classifier, projector, and reconstructor to minimize $\mathcal{L}_i$;
end
Step2 training
for $(\mathbf{x}_i, \mathbf{t}_i, y_i) \in D$ **do**
  Extract $\mathbf{v}_i$ from $\mathbf{x}_i$ via the image encoder;
  Extract $\mathbf{s}_i$ from $\mathbf{t}_i$ via the text encoder;
end
Build k-reciprocal-NN graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ with $\{\mathbf{v}_i\}_{i=1}^{N}$;
Enhance text embeddings from $\mathbf{s}_i$ to $\hat{\mathbf{s}}_i$ via graph convolution on $\mathcal{G}$;
for $c \in \{1, 2, ..., C\}$ **do**
  Extract $\mathbf{s}^c$ from $\mathbf{t}^c$ via the text encoder;
  for $i \in \{1, 2, ..., N | y_i = c\}$ **do**
    Match textual instances $\mathbf{s}_i$ to prototypes $\mathbf{s}^c$ to obtain visual anchors D_K^c;
  end
end
Initialize visual prototypes with D_K;
for $(\mathbf{x}_i, y_i) \in D$ **do**
  $\mathcal{L}_i = (1 - \lambda^{\text{bts}})\mathcal{L}_i^{\text{cls}} + \lambda^{\text{bts}}\mathcal{L}_i^{\text{bts}} + \lambda^{\text{prj}}\mathcal{L}_i^{\text{prj}} + \lambda^{\text{pro}}\mathcal{L}_i^{\text{pro}} + \lambda^{\text{ins}}\mathcal{L}_i^{\text{ins}}$;
  Update image encoder, classifier, and projector to minimize $\mathcal{L}_i$;
  Refine y_i to $\hat{y}_i$ to remove noise;
end
Step3 fine-tuning
for $(\mathbf{x}_i, \hat{y}_i) \in D$ **do**
  $\mathcal{L}_i = \mathcal{L}_i^{\text{cls}}$;
  Update classifier to minimize $\mathcal{L}_i$;
end

Table S3: Effect of γ on CAPro without collective bootstrapping.

γ	Reference Provider	Google500 Top1	Google500 Top5	ImageNet500 Top1	ImageNet500 Top5	NUS-WIDE C-F1	NUS-WIDE O-F1	NUS-WIDE mAP
0.6	×	72.0	88.0	66.9	85.4	8.3	9.1	6.9
0.8	×	71.2	87.7	65.9	84.8	–	–	–
0.9	×	–	–	–	–	39.2	44.4	46.8

S4.2 Effect of γ on Noise Removal

Table S3 reports the influence of γ when collective bootstrapping is not applied. We follow MoPro [8] to validate two γ candidates: $\gamma = 0.6$ and $\gamma = 0.8$. We find that $\gamma = 0.6$ works the best on Google500. As γ increases, it becomes more difficult for the model to correct labels and thereafter label-flipping errors may still exist. As suggested by MoPro, the optimum γ is related to the percentage of noise in web datasets. For noiser datasets, γ should be decreased for the model to correct wrong labels at an early stage. Otherwise, overfitting might occur and weaken prototypical representation learning. The fine-tuning of γ requires elaborate experiments, which is beyond the scope of the present study.

For multi-label learning on NUS-WIDE, the optimum γ is not only related to the noise level but also to the ratio of the number of positive examples to that of negative examples. Since the negative instances in each class exceeds positive ones by more than one order of magnitude, decreasing γ will easily make the model to classify any instance as negative. Once the model overfits the overwhelming negative examples, valid positive supervision sources would only come from the top-K matched examples with cross-modality alignment, which degrades generalizability greatly. In this case, we should keep a stricter threshold $\gamma = 0.9$ to only allow confident label rectification.

Table S4: Effect of prototype update frequency on CAPro. By default, we update visual prototypes every epoch using high-quality examples in each mini-batch. For 0-epoch per update, we do not introduce additional high-quality web examples to polish prototypes, but only update them with the top-K matched semantically-correct examples with their latest visual embeddings.

# Epochs	Google500		ImageNet500		NUS-WIDE		
per update	Top1	Top5	Top1	Top5	C-F1	O-F1	mAP
0	75.5	91.1	71.6	88.8	39.2	44.4	47.2
1 (by default)	76.0	91.3	72.0	89.2	39.3	45.4	48.0
5	75.9	91.2	71.8	89.2	39.6	45.0	47.6
10	76.0	91.2	71.7	89.1	39.3	45.8	48.2

S4.3 Effect of Prototype Update Frequency

By default, we update prototypes by examples in a mini-batch for every epoch. We additionally perform ablation study on the update frequency where comparison is conducted between: 1) 0-epoch, 2) 1-epoch (by default), 3) 5-epoch, and 4) 10-epoch. For 0-epoch update, we **do not** update prototypes with embeddings from other high-quality web examples. Instead, prototypes are **renewed** with the latest embeddings only from the top-K matched examples to avoid being out-of-date. For 5-epoch and 10-epoch update, we polish prototypes every 5 and 10 epochs, respectively.

Table S4 reports the effect of prototype update frequency on CAPro. On the Google500 and NUS-WIDE datasets, if prototypes are only formed by limited clean examples, their generalization across domain becomes poor and thus causes performance drop of 0.45% (top1) and 0.6% on average, respectively. For Google500, reduced update frequency generally causes lower performance, meaning that the prototypes should keep refreshed for large-scale datasets. For NUS-WIDE, if the update frequency decreases a bit (5-epoch), the model improves on C-F1 but underperforms a little on O-F1 and mAP. With the 10-epoch frequency, we surprisingly find that results are improved on all evaluation metrics. One possible explanation is that the delayed prototype update can help stabilize training at an early stage, but the optimal frequency might be subject to dataset scale and noise level.

Table S5: Effect of noise removal policy on CAPro. We compare with MoPro to show the effectiveness of keeping labels of top-K matched semantically-correct examples unchanged.

Noise Removal	Google500		ImageNet500		NUS-WIDE		
policy	Top1	Top5	Top1	Top5	C-F1	O-F1	mAP
MoPro [8]	75.8	91.1	71.7	89.0	38.8	42.2	47.2
CAPro (ours)	76.0	91.3	72.0	89.2	39.3	45.4	48.0

S4.4 Effect of Noise Removal Policy

Table S5 reports the effect of noise removal policy on CAPro. Our label adjustment policy is inspired from the Eq. (5) of MoPro [8] but differs in that we keep labels of the top-K matched examples selected in cross-modality alignment unchanged. Therefore, if we replace our noise removal policy with the MoPro one, we actually allow the deep model to get rid of the guidance from top-K examples. These selected top-K examples can be altered or even discarded by the MoPro-style policy.

It turns out that without such enforcement, the performance dropped under both single-label and multi-label scenarios. The possible reason behind is that, due to the overwhelming noise (*e.g.*, the semantic-misalignment noise) in certain categories, the model itself cannot keep representation learning robust to noise even with a good start. The labels of top-K samples will be prone to the noisy majority, which invalidates prototypical learning. Besides, the superiority of CAPro over MoPro-style update also substantiates the purity and correctness of the selected examples.

S5 Analysis on Open-Set Recognition

We provide detailed analysis on CAPro for open-set recognition. Fig. S3 presents the per-class F1 (C-F1), per-class Precision (C-P), and per-class Recall (C-R). We compare five methods for ablation

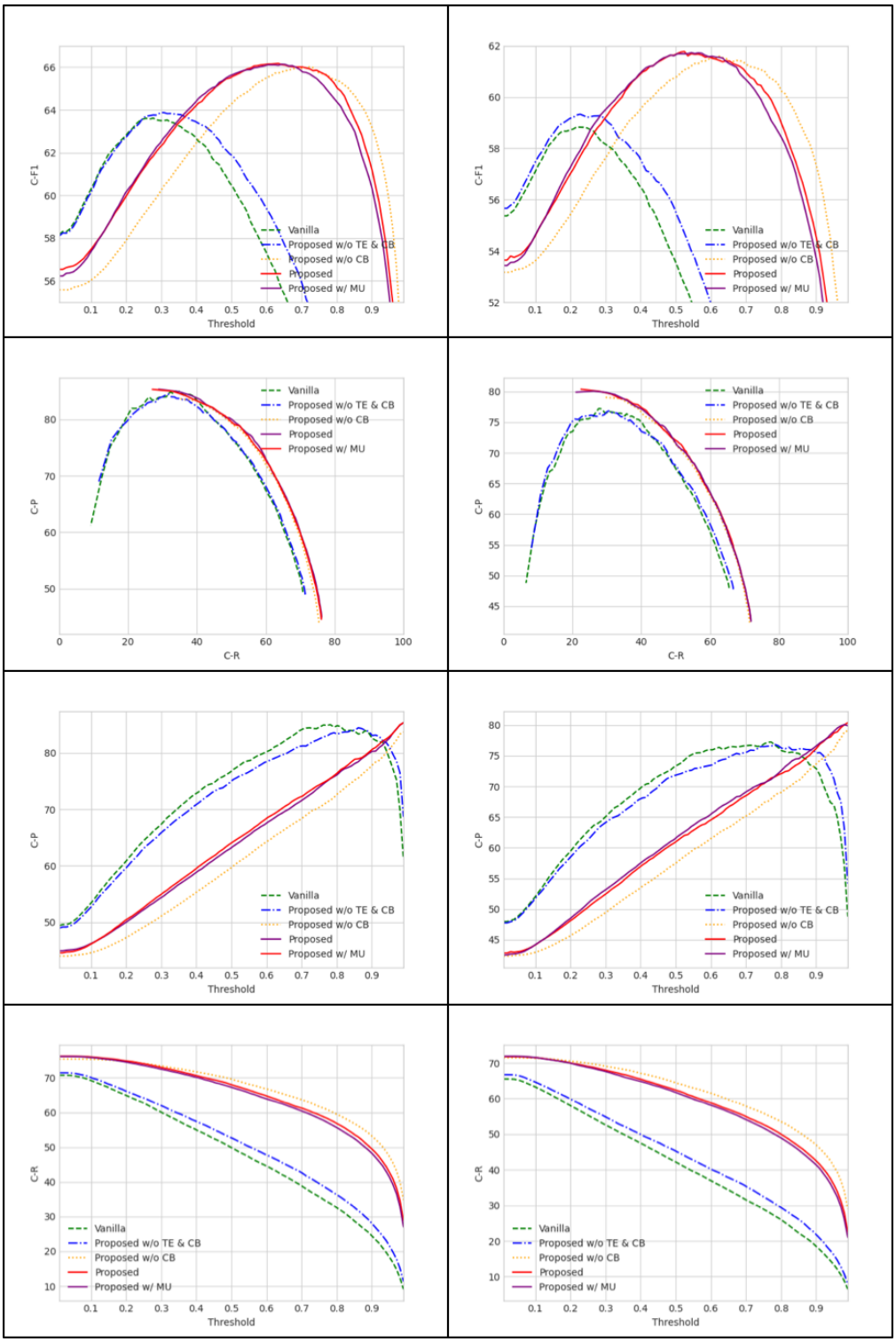

Figure S3: Effect of threshold for open-set recognition on (a) WebVision1k and (b) ImageNet1k. TE, CB, and MU respectively refer to our text enhancement, collective bootstrapping, and mix-up. Examples with prediction confidence lower than the threshold will be classified as open-set category.

study including: 1) the vanilla method, 2) CAPro without text enhancement (TE) & collective bootstrapping (CB), 3) CAPro without CB, 4) CAPro, and 5) CAPro with mix-up (MU).

We vary the confidence threshold from 0 to 1 with an interval of 0.01 to comprehensively measure the performance of CAPro. For each example, if the highest model prediction confidence is below the threshold, the example will be classified as the open-set category. Otherwise, the example will be classified into one of the known categories. We train methods on the Google500 training set and validate them on WebVision1k and ImageNet1k testing sets. Examples from the remaining 500 novel categories should all fall into the open-set category.

It can be observed that compared with vanilla method, CAPro enjoys a much higher recall but a relatively lower precision. It means that our CAPro is more confident about its prediction and a threshold around 0.6 would end up with an optimal C-F1 over 66%. The precision-recall curve reflects that each key component does improve open-set recognition. Note that due to the limited sampling of confidence threshold, the precision-recall curve is not spanning across the entire axes. However, the tendency of curves confirms the effectiveness of our components.

S6 Guidelines on Tuning of Hyper-Parameters

Loss weights of λ^{bts}, λ^{prj}, λ^{pro}, and λ^{ins} First, for the total objective, we follow MoPro [8] to use $\lambda^{\text{pro}} = 1$ and $\lambda^{\text{ins}} = 1$. Out of simplicity, we also use $\lambda^{\text{prj}} = 1$ as default.

Second, we would like to explain the effect of λ^{pro}, λ^{ins}, and λ^{prj} on regularzation. A larger λ^{pro} may pull instances too close totheir prototypes, which "shrinks" class clusters in the embedding space. A larger λ^{ins} will enforce stronger visual discriminability between two instances. It may cause two examples from the same category to differ greatly and thereafter downgrades the visual prototype update and class cluster regularization. A larger λ^{prj} improves the reconstruction quality of $\tilde{\mathbf{v}}_i$, which encourages $\mathbf{z}_i$ to retain more information of $\mathbf{v}_i$ in the embedding space. The projection-reconstruction loss is only involved in the pre-training stage (see Algo. S1), and therefore λ^{prj} will not affect the prototypical and instance-wise contrastive learning in the following stage.

Third, for one's custom web datasets, we suggest that λ^{pro}, λ^{ins}, and λ^{prj} should be tuned according to the performance results under three settings: 1)$\lambda^{\text{pro}} = 0$ vs. $\lambda^{\text{pro}} = 1$; 2)$\lambda^{\text{ins}} = 0$ vs. $\lambda^{\text{ins}} = 1$; 3)$\lambda^{\text{prj}} = 0$ vs. $\lambda^{\text{prj}} = 1$.

According to our experiments on both single-label and multi-label datasets, the default settings of $\lambda^{\text{pro}} = 1$, $\lambda^{\text{ins}} = 1$, and $\lambda^{\text{prj}} = 1$ should work well on most cases.

For λ^{bts}, we suggest 0.1 would achieve a balance between the individual and collective label references. A much larger value may cause over smoothing and over-regularization on visual learning.

Threshold γ For γ on single-label datasets, its value is related to the percentage of noise in datasets. For WebVision1k and Google500 (34% noise [11]), $\gamma = 0.6$ works better than $\gamma = 0.8$. For one's own web dataset, if the noise ratio is larger, γ should be tuned lower so that wrong labels could be corrected at an earlier stage before overfitting. For γ on multi-label datasets, its value is related to both the percentage of noise and the number ratio of positive-negative samples. For NUS-WIDE (50% noise [3] and 0.02 avg. ratio of positive-negative examples), $\gamma = 0.9$ works better than $\gamma = 0.6$. For one's own web dataset, if the noise ratio is smaller and the positive over negative ratio is smaller, γ should be tuned higher so that hard positive samples will not be easily discarded to avoid underfitting.

Prototype Update Frequency For the update frequency, its value is related to the dataset scale and noise level. For WebVision1k and Google500, visual prototypes should be updated per epoch to improve their diversity, which better handles the domain gap between web and realistic datasets. For NUS-WIDE, the update frequency could be reduced to stabilize training, where the prototypes can be prone to the overwhelming negative examples in each category.

Top-K For top-K, its value is related to the percentage of noise. If the noise ratio is less than 30%, K should be set higher than 50 to include more diverse examples.

Others The current settings of other hyper-parameters (see Table S2) work well. For one's own dataset, we believe these values can be set as starting points and finetuned accordingly. Among

all hyper-parameters, our ablation results show that for λ^{bts}, γ, and top-K, their values do affect performance and should be set following the rules mentioned above (such as the dataset scale, the noise ratio, and the positive-negative ratio). For the remaining hyper-parameters such as the prototype update frequency, we do not observe significant fluctuation. In other words, the model is robust to these hyper-parameters.

S7 Failure Cases

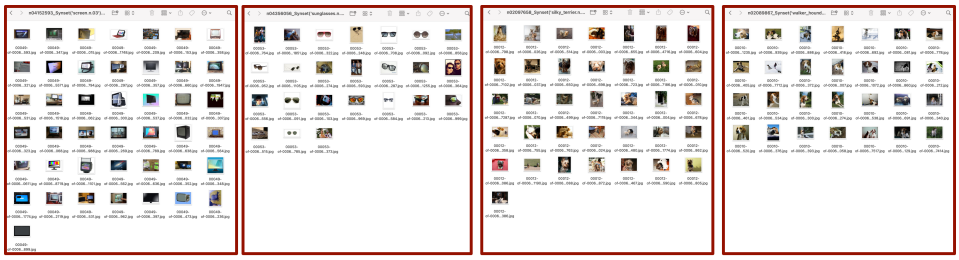

Figure S4: Failure cases of our CAPro on certain classes where the simple vanilla baseline achieves better performance.

We provide failure cases of our CaPro on the WebVision1k dataset (see Fig. S4). Our observations are summarized as the following.

First, CAPro can handle fine-grained categories on WebVision1k. The introduction of atypicals increases the risk of noise. For generalization on anomalies or rarities, one solution is to choose both top-K and randomly sampled instances.

Second, for WebVision1k, both MoPro and CAPro underperform the vanilla baseline (optimized only by the cross-entropy loss) on a total of 387 and 312 classes, respectively. Top5 failures of classes include screen, sunGlasses, bellCote, ballPlayer, and popBottle. For ImageNet1k, MoPro and CAPro underperform the vanilla on a total of 450 and 358 classes, respectively. Top-5 failures of classes include silkyTerrier, walkerHound, academicGown, standardSchnauzer, and bellCote.

We also provide interesting findings below:

First, the domain gap exists between web and realistic datasets as the top 5 failure cases on the WebVision1k and ImageNet1k testing sets are quite different.

Second, the vanilla method tends to overfit the training set so that it outperforms on highly similar concepts such as screen vs. monitor and sunGlasses vs. sunGlass.

Third, mistakes on silky vs. yorkshire Terrier and walker vs. englishFox hound are ascribed to over-regularization. The inter-class relationship might be used for class-wise adjustment.

S8 Computational Complexity

Table S6: The parameters and GFLOPs of different encoders.

Encoders	Number of Parameters	GFLOPs
R50 [6]	25M	3.8
MiniLM [7]	22M	4.7
XLNet [12]	110M	29
GPT-Neo [13]	1.3B	3400

First, we present the number of parameters and GFLOPs for the image and text encoders in Table S7.

Second, we present the cost of the text enhancement of our CAPro with respect to its performance gains. Here, N denotes the number of all nodes in the visual graph. k is the number of neighbors per

Table S7: The cost of the text enhancement of our CAPro with respect to its performance gains.

Text Encoding	Text Enhancement	Cost	Google500 Top1	ImageNet500 Top1
MiniLM [7]	VSGraph [3]	$O(N^2 d_v)$+$O(N{d_v}^2 + Nkd_v)$	72.0	66.9
	Ours	$+O(3Nkd_v)$+$O(4k)$+$O(4klog(4k))$	+3.5	+4.6
XLNet [12]	VSGraph [3]	$O(N^2 d_v)$+$O(N{d_v}^2 + Nkd_v)$	71.6	66.8
	Ours	$+O(3Nkd_v)$+$O(4k)$+$O(4klog(4k))$	+3.8	+4.7
GPT-Neo [13]	VSGraph [3]	$O(N^2 d_v)$+$O(N{d_v}^2 + Nkd_v)$	72.0	67.2
	Ours	$+O(3Nkd_v)$+$O(4k)$+$O(4klog(4k))$	+3.7	+4.4

node and d_v is the dimension of the feature $\mathbf{v}$. With $k = 5$ and $k = 10$ respectively for WebVision1k and NUS-WIDE, our improvement over VSGraph is worthy at the expense of such a low cost.

Table S8: The cost of the reference provider of our CAPro with respect to its performance gains.

Text Encoding	Reference Provider	Cost	Google500 Top1	ImageNet500 Top1
MiniLM [7]	Mix-up [14]	–	75.7	71.4
	NCR [9]	$O(m^2(d_v + C))$	-0.2	+0.1
	Our CB	$O(mQ(d_p + C))$	+0.3	+0.6
MiniLM [7]	Bootstrap [15]	–	75.5	71.3
	NCR [9]	$O(m^2(d_v + C))$	+0	+0.2
	Our CB	$O(mQ(d_p + C))$	+0.5	+0.7
MiniLM [7]	LabelSmooth [16]	–	75.4	71.2
	NCR [9]	$O(m^2(d_v + C))$	+0.1	+0.3
	Our CB	$O(mQ(d_p + C))$	+0.6	+0.8
MiniLM [7]	SCC [2]	–	73.8	70.2
	NCR [9]	$O(m^2(d_v + C))$	+1.7	+1.3
	Our CB	$O(mQ(d_p + C))$	+2.2	+1.8

Third, we present the cost of the reference provider of our CAPro with respect to its performance gains. Here, m is the batch size, d_v is the dimension of $\mathbf{v}$, Q is the size of dictionary, d_p is the dimension of z, and C is the number of classes. The common reference provider techniques such as the Mix-up, Bootstrapping, label smoothing, and SCC do not incur significant overhead. Operations of our CB are fast to compute for moderate $m = 256$, $Q = 8192$, $d_p = 128$, and $C = 1000$ since PyTorch supports efficient matrix multiplication on GPUs. Besides, compared with NCR, our $d_p = 128$ is 16x smaller than d_v=2048 in NCR, and our $m = 256$ is 4x smaller than $m = 1024$ in NCR. For WebVision1k, our cost is 1.35x smaller than NCR. For NUS-WIDE, our cost is 20.37x smaller than NCR. It is reasonable to conclude that our CB is more efficient and effective than NCR.

Finally, it is noted that the text encoding and text enhancement methods are performed off-line and executed only once. They do not participate in network optimization. Besides, the pretrained text encoders are only used for inference under 1 V100 GPU. Therefore, the additional cost is acceptable in return for semantically-correct web images.

References

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 248–255. IEEE, 2009.

[2] Jingkang Yang, Litong Feng, Weirong Chen, Xiaopeng Yan, Huabin Zheng, Ping Luo, and Wayne Zhang. Webly supervised image classification with self-contained confidence. In *European Conference on Computer Vision*, pages 779–795. Springer, 2020.

[3] Jingkang Yang, Weirong Chen, Litong Feng, Xiaopeng Yan, Huabin Zheng, and Wayne Zhang. Webly supervised image classification with metadata: Automatic noisy label correction via visual-semantic graph. In *Proceedings of ACM International Conference on Multimedia*, pages 83–91, 2020.

[4] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. NUS-WIDE: a real-world web image database from national university of singapore. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, pages 1–9, 2009.

[5] Jorge Ramón Fonseca Cacho, Ben Cisneros, and Kazem Taghva. Building a wikipedia n-gram corpus. In *Proceedings of SAI Intelligent Systems Conference*, pages 277–294. Springer, 2021.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 770–778, 2016.

[7] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33:5776–5788, 2020.

[8] Junnan Li, Caiming Xiong, and Steven Hoi. Mopro: Webly supervised learning with momentum prototypes. In *International Conference on Learning Representations*, 2020.

[9] Ahmet Iscen, Jack Valmadre, Anurag Arnab, and Cordelia Schmid. Learning with neighbor consistency for noisy labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4672–4681, 2022.

[10] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[11] Wen Li, Limin Wang, Wei Li, Eirikur Agustsson, and Luc Van Gool. Webvision database: Visual learning and understanding from web data. *arXiv preprint arXiv:1708.02862*, 2017.

[12] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in Neural Information Processing Systems*, 32, 2019.

[13] Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, March 2021. URL `https://doi.org/10.5281/zenodo.5297715`. If you use this software, please cite it using these metadata.

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

[15] Scott E Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. Training deep neural networks on noisy labels with bootstrapping. In *International Conference on Learning Representations (Workshop)*, 2015.

[16] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. When does label smoothing help? *Advances in Neural Information Processing Systems*, 32, 2019.