A. Details of Motivating Experiment in Figure 2

In Table 8, we described all the selected classes from ImageNet to construct OS_{oracle}, for each target dataset. For clarity, we also visualized examples for each selected category in Figures 7–10. Also, Table 9 summarizes the exact numbers of the motivating experiment results.

Target (X)	Selected Classes for OS_{oracle}
Aircraft	airliner, airship, American egret, crane, space shuttle, spoonbill, warplane, white stork
Cars	ambulance, beach wagon, cab, convertible, jeep, limousine, minivan, Model T, racer, sports car
Pet	basset, beagle, boxer, cocker spaniel, Chihuahua, Egyptian cat, English setter, German short-haired pointer, Great Pyrenees, keeshond, Leonberg, miniature pinscher, Newfoundland, Persian cat, Pomeranian, pug, Saint Bernard, Samoyed, Scotch terrier, Siamese cat, soft-coated wheaten terrier, Staffordshire bullterrier, tiger cat, Yorkshire terrier
Birds	albatross, bee eater, brambling, bulbul, chickadee, goldfinch, house finch, hummingbird, indigo bunting, jacamar, jay, junco, kite, magpie, pelican, quail, red-backed sandpiper, red-breasted merganser, redshank, robin

Table 8. The class list of OS_{oracle} according to each target dataset (X).

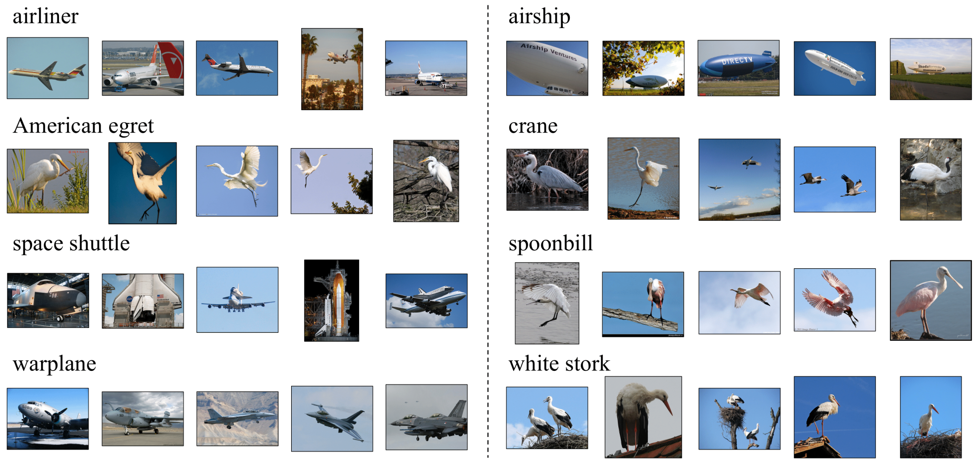


Figure 7. Visualization of examples whose classes belong to OS_{oracle} (X = FGVC-Aircraft).

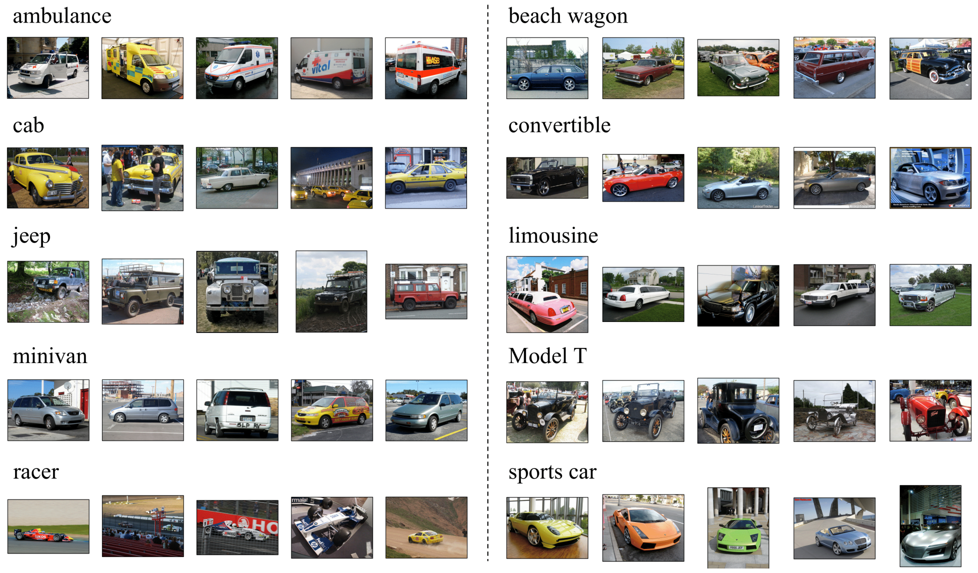


Figure 8. Visualization of examples whose classes belong to OS_{oracle} (X = Stanford Cars).

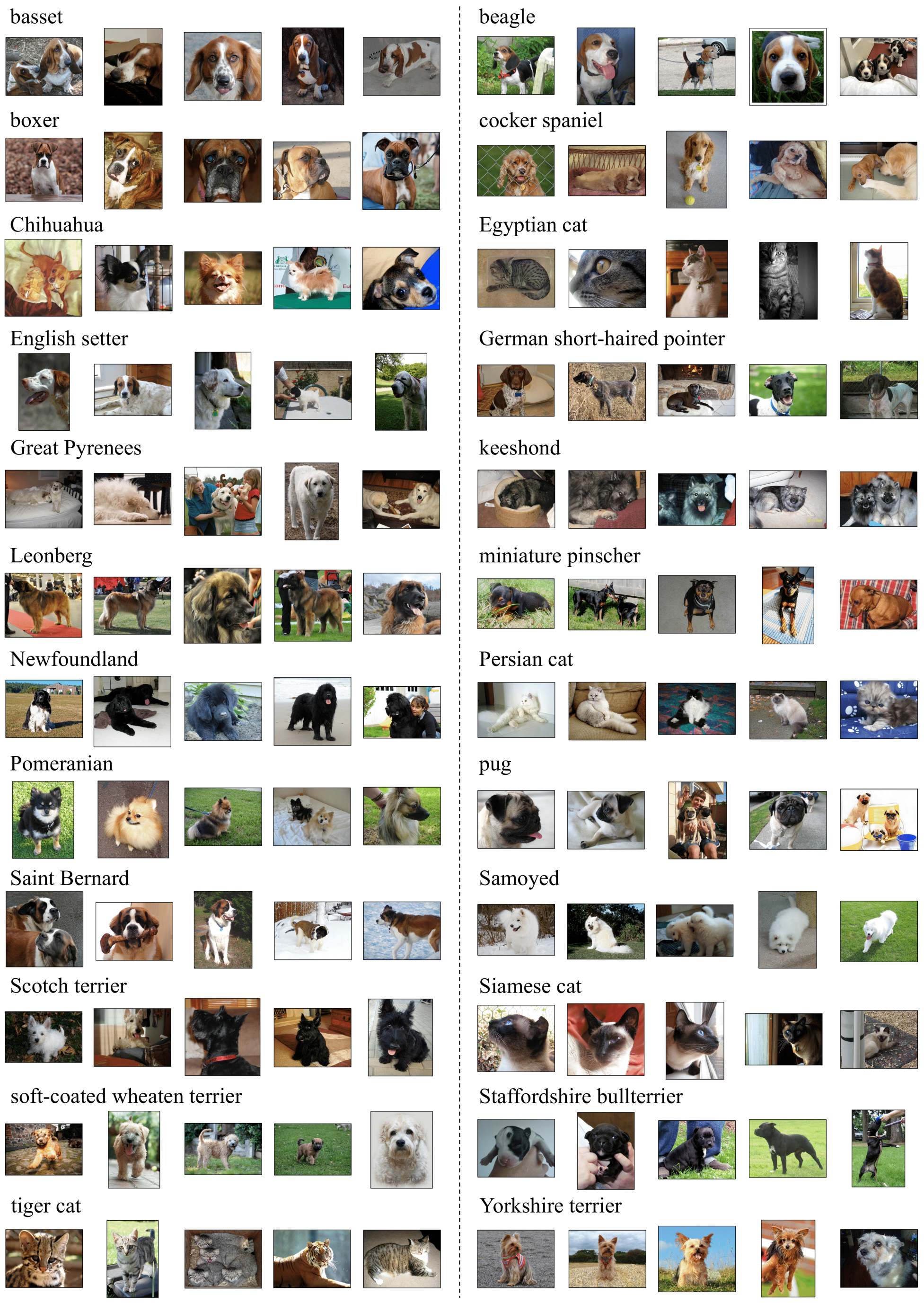


Figure 9. Visualization of examples whose classes belong to OS_{oracle} (X = Oxford-IIIT Pet).

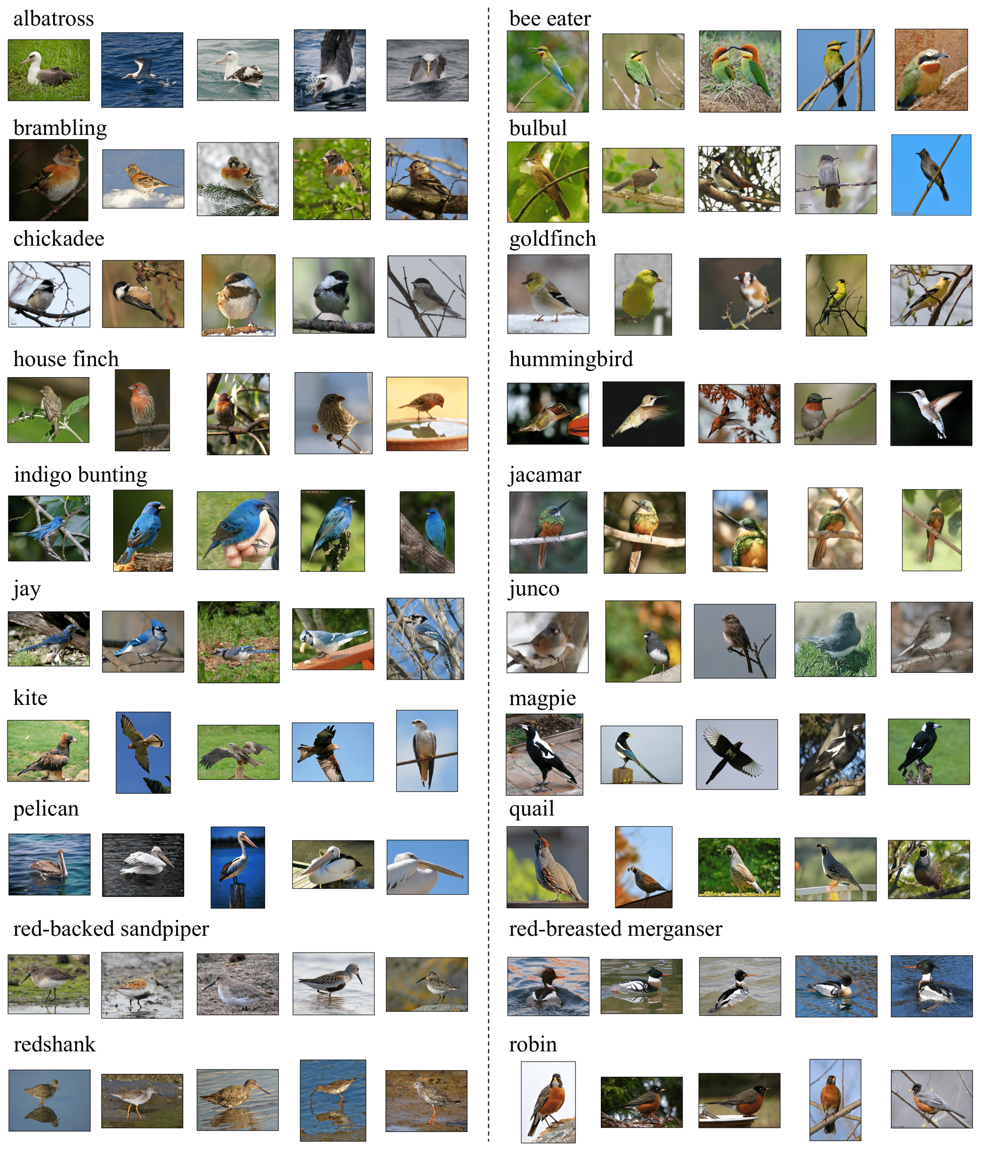


Figure 10. Visualization of examples whose classes belong to OS_{oracle} (X = Caltech-UCSD Birds).

	Target dataset (X) and its number of samples							
pretraining	FGVC-Aircraft		Stanford Cars		Oxford-IIIT Pet		Caltech-UCSD Birds	
X	46.56%	(6.7K)	55.42%	(8.1K)	59.23%	(3.7K)	29.27%	(6.0K)
OS	41.50%	(1.3M)	41.86%	(1.3M)	67.66%	(1.3M)	33.21%	(1.3M)
X+OS	39.88%	(1.3M)	42.92%	(1.3M)	68.22%	(1.3M)	32.88%	(1.3M)
X+OS_{rand}	48.24%	(19.5K)	49.26%	(20.9K)	64.27%	(16.5K)	31.90%	(18.8K)
$\boldsymbol{X}$**+**$\boldsymbol{OS}_{\textbf{oracle}}$	**48.78%**	(17.1K)	**57.56%**	(21.1K)	**80.73%**	(34.9K)	**38.26%**	(32.0K)

Table 9. Linear evaluation performance with sample size for each pretraining dataset. ImageNet is used as the open-set, and the random sampling portion is 1% of $|OS|$.

B. Experimental Settings

We used eleven fine-grained datasets and four open-sets in the main experiments. We summarized the dataset configurations in the following Table 10. We followed the linear evaluation protocol used in recent SSL literature [14, 24]: pretraining the encoder and fine-tuning only the classifier with the frozen encoder part.

B.1. Datasets

Since the SSL pretraining does not rely on any label information, we incorporate the train and validation set of the original benchmarks, such as FGVC-Aircraft, Oxford 102 Flower, Describable Textures, and Food 11, to enlarge the number of training samples. Besides, in the case of CelebAMask-HQ, we exclude identities that contain less than 15 images. The original number of 30,000 images thus have reduced to 4,263, and the number of identities have reduced from 6,217 to 307.

Dataset	Train #	Test #	Class #
ImageNet [20]	1,281,167	50,000	1,000
Microsoft COCO [42]	118,287	5,000	80
iNaturalist 2021-mini [67]	500,000	100,000	10,000
Places365 [82]	8,026,628	328,500	365
FGVC-Aircraft [47]	6,667	3,333	100
Stanford Cars [35]	8,144	8,041	196
Oxford-IIIT Pet [53]	3,680	3,669	37
Caltech-UCSD Birds [74]	5,990	5,790	200
Stanford Dogs [32]	12,000	8,580	120
Oxford 102 Flower [49]	2,040	6,149	102
Stanford 40 Actions [77]	4,000	5,532	40
MIT-67 Indoor Scene Recognition [54]	5,360	1,340	67
Describable Textures (DTD) [18]	3,760	1,880	47
CelebAMask-HQ [36]	4,263	1,215	307
Food 11 [59]	13,296	3,347	11

Table 10. Datasets we used and their configurations. The upper part corresponds to open-sets and the lower part corresponds to fine-grained target datasets. Although every open-set contains its test set and designated classes, we have only used an unlabeled train set.

B.2. Encoder Pretraining

Basic settings: We set the cost of pretraining to 8 GPU days (4 GPUs $\times$ 2 days) since the train set sizes vary depending on the pretraining dataset. For example, we pretrained the model for longer epochs of 5K on small-scale fine-grained datasets, because one epoch is a few hundred times shorter in iteration than an epoch of *OS*. Meanwhile, although it took more than 8 GPU days, we trained 200 epochs for ImageNet open-set (*OS*) and X+*OS* experiments for full pretraining.

For SimCore pretraining, every 1% sampling last 5K epochs, but every 5% sampling last 2K epochs due to the excess of 8 GPU days. Since SimCore with a stopping criterion had different sampling ratios according to the target dataset, in this case, we trained the model for $\min(5\text{K}, 10\text{K}/p)$ epochs. If the open-set changes, we increased or decreased the epochs by the ratio between the open-set size and ImageNet size, *i.e.*, X+Places365$_{\text{SimCore}}$ uses $\times$6.3 shorter epochs than X+ImageNet$_{\text{SimCore}}$. Also, when using SimCore with a stopping criterion, a budget limit is not necessarily needed. Nonetheless, in case there are sufficiently large number of core-samples in the open-set, we set the budget to $\times$50 of the target dataset size to see the effect of a small coreset. Note that we used the stopping threshold τ as 0.95 in default.

SimCLR method: When pretraining the encoder with the SimCLR method, SGD optimizer is used with an initial learning rate of 0.1 and ℓ_2 regularization parameter 1e-4, with 512 batch size. The learning rate was scheduled by cosine annealing [44], decayed to zero on the last epoch. We followed the same hyperparameters, the temperature of 0.07 and the projection dimension of 128, as in the original paper [14]. For data augmentation, we used a random resized crop, horizontal flip, color jitter, and random gray-scale.

BYOL method: We used an Adam optimizer [34] with a learning rate of 1e-3 and ℓ_2 regularization parameter 1e-6. BYOL method is highly sensitive to the exponential moving average (EMA) value of a momentum encoder [28]. Therefore, we followed the same EMA scheduler as in the original implementation [24], of which the momentum starts from 0.996 to 1. We should also note that we did not use the EMA scheduler for the experiments with short epochs.

SwAV method: We mostly followed the same settings as the SimCLR method, such as an SGD optimizer with a learning rate of 0.1 and weight decay of 1e-4. In the case of pretraining on ImageNet, we used the temperature value of 0.1 and an epsilon value of 0.02, while we froze the 3,000-dimensional prototypes for one epoch. For every other cases, we froze the 100-dimensional prototypes for ten epochs.

DINO method: We mostly followed the pretraining setups as in the original paper [11]. We used an AdamW optimizer [45] with a learning rate of 1e-3. Cosine annealing and warmup were used during the 2% of total epochs. We scheduled an ℓ_2 regularization parameter from 0.04 to 0.4, and the EMA momentum was increased from 0.996 to 1.0. Note that the ℓ_2 regularization had not been applied to any biases and normalization parameters. Besides, we scheduled the teacher temperature from 0.04 to 0.07 for the 40% of total epochs and set the student temperature as 0.1. The center momentum was 0.9, and we set the dimension of projector as 65,536. As a data augmentation technique, we used additional four local croppings [10] (96×96). The encoder's last layer was frozen during the first 10 epochs. These numerous hyperparameters were highly fit to the ImageNet pretraining, whereas they did not seem effective in several fine-grained datasets. In practice, the stopping threshold τ of 0.6 was optimal for the DINO method, which can be attributed to the large dimension of the projector.

MAE method: We used an AdamW optimizer with a learning rate of 3e-4, cosine annealing, and warmup epochs of 100. Furthermore, we used the constant ℓ_2 regularization parameter of 0.05. As in the original paper [27], we set the mask ratio as 75%, and we normalized the pixel values when calculating the reconstruction loss.

B.3. Fine-Tuning for Linear Evaluation

We fine-tuned a linear classifier for 100 epochs and searched the optimal learning rate among five logarithmically spaced values from 1 to 10^2. We decayed the learning rate by 0.1 at 60 and 80 epochs, and any regularization techniques, such as weight decay, were not used.

B.4. Open-Set Semi-Supervised and Webly Supervised Learning

SimCore fine-tuning: For SimCore fine-tuning baselines, we fine-tuned the SimCore pretrained models for 200 epochs with the batch size of 256. We used an SGD optimizer with ℓ_2 regularization parameter of 1e-4. The learning rate started from the value of 3e-2 and was decayed by a cosine annealing scheduler.

OpenSemi framework: Under the open-set semi-supervised learning (OpenSemi) framework, we used an SGD optimizer with a learning rate of 3e-2, ℓ_2 regularization parameter of 1e-4, and cosine annealing. In default, we trained the random initialized model for 128 epochs with iteration steps of 512 and the batch size of 64. For the models pretrained by SimCore, we used the shorter epochs of 32.

For the Self-Training algorithm [61], we used the temperature of 1.0 and set the coefficient of the unlabeled loss as 0.5. We trained a teacher network from scratch for 500 epochs and fine-tuned the SimCore initialized model for 100 epochs.

For the OpenMatch algorithm [56], we set the coefficient of open-set entropy minimization (OEM) loss and soft open-set consistency regularization (SOCR) loss as 0.1 and 0.5, respectively. Besides, we set the epochs to start FixMatch training as 20 for random initialized models and 5 for SimCore pretrained models.

WeblySup framework: We used an SGD optimizer with a learning rate of 3e-2, ℓ_2 regularization parameter of 1e-4, and cosine annealing. For the Co-Teaching algorithm [25], we used the batch size of 256 and the forget ratio of 0.2. In addition, we followed the values of other hyperparameters as in Han *et al*. We set the epochs as 1,000 for training from scratch, and 200 for the experiments of SimCore initialization. Note that we modified the learning rate and forget ratio to 0.1 for Cars dataset.

For the DivideMix method [38], we trained the model with 400 epochs and the batch size of 128. We set the probability threshold as 0.2 and warmup epochs as 30 for every dataset. Since the DivideMix algorithm includes the training schemes of MixMatch [7], we used the default hyperparameters of the MixMatch algorithm, such as the unlabeled loss coefficient of 75 and alpha value of 0.75.

B.5. Downstream Tasks

For the object detection task, we used an Adam optimizer with a learning rate of 1e-4 and the batch size of 16. Besides, we set the total epochs and IoU threshold of non-maximum suppression as 30 and 0.2, respectively. We should note that the backbone network of DeepLabV3+ is frozen during fine-tuning, and we trained an FPN network from scratch.

For the semantic segmentation task, we have also frozen the SSL pretrained ResNet-50 encoder of RetinaNet. We used an SGD optimizer with the batch size of 256, ℓ_2 regularization parameter of 1e-4, and the Nesterov momentum. The optimal learning rate is chosen among five logarithmically spaced values from 1e-1 to 1e-3.

C. Sampling Ratios of SimCore Experiments

The coreset denotes a subset of the open-set that is semantically similar to the target dataset. Since we measure this similarity on the feature space generated by a retrieval model, the sampling ratio with a stopping criterion could vary depending on several factors, such as model architectures, SSL losses, open-set, or target dataset. That is, an image pair can be recognized as similar or dissimilar according to how they are represented.

The exact values of sampling ratios in our experiments are summarized in Tables 11–12. In effect, SimCLR with any architecture samples less than 1% as the coreset of Cars, but BYOL with ResNet50 samples over 30% of the open-set. Although the proper sampling ratios were different across each method, SimCore has consistently improved performances with the selected samples.

method	architecture	Aircraft	Cars	Pet	Birds
SimCLR	EfficientNet	0.59%	0.37%	2.16%	1.16%
SimCLR	ResNet18	0.31%	0.35%	0.96%	1.09%
SimCLR	ResNeXt50	0.79%	0.96%	14.36%	13.17%
SimCLR	ResNet101	0.64%	0.65%	8.51%	11.37%
BYOL	ResNet50	4.78%	31.78%	14.36%	23.38%
SwAV	ResNet50	26.02%	31.78%	14.36%	23.38%
DINO	ViT-Ti/16	12.19%	2.51%	14.36%	21.41%
MAE	ViT-B/16	3.23%	1.11%	8.47%	5.69%

Table 11. Sampling ratios of the SimCore experiments in Table 3 and Table 4.

Furthermore, SimCore samples the reasonable amount with any open-set samples, *e.g.*, sampling number of iNaturalist (32K) vs. Places365 (268K) in Indoor. Along with Figure 3 and Table 5, the performance gain of SimCore is correlated with the semantic similarity between X and OS, suggesting the benefit of suitable open-sets.

OS for SimCore	Pet	Birds	Action	Indoor	Aircraft	Cars
COCO (0.12M)	29.07%	29.60%	59.88%	37.49%	-	-
iNaturalist (0.5M)	22.05%	13.03%	21.87%	6.31%	-	-
Places365 (8M)	2.29%	3.73%	2.49%	3.34%	-	-
ImageNet (1.3M)	14.36%	13.68%	15.61%	13.46%	1.03%	0.95%
`ALL` (9.9M)	1.85%	3.02%	2.01%	2.70%	0.20%	0.43%
WebVision (2.4M)	7.52%	12.24%	8.18%	10.96%	0.39%	0.76%
WebFG-496 (0.05M)	-	34.47%	-	-	25.32%	40.21%

Table 12. Sampling ratios of the SimCore experiments in Figure 3 and Table 5. The ratios are calculated based on the size of each open-set.

D. Implementation Details of Feature Distribution Analysis

Figure 5 in Section 4.3 visualizes feature distribution of each pretraining dataset. We extracted the feature embeddings with the retrieval model (*i.e.*, pretrained encoder on the target dataset for short epochs). Then, we reduced all representations to two-dimensional vectors via t-SNE [66]. These representation vectors are distributed on a unit ring by Gaussian kernel density estimation, as introduced in [71].

In detail, we visualized the feature distributions of OS, X, and the coreset sampled from OS. For the visualization of OS, ImageNet in this case, we randomly selected 10% of OS for faster convergence of t-SNE. For the coreset, SimCore samples 1% of OS. Unlike in Aircraft and Cars, in Pet and Birds, the distribution of X features are more overlapped with that of OS features. This implies high semantic similarity between the open-set (ImageNet) and the target dataset (Pet or Birds).

E. Sensitivity Study

E.1. Stopping Threshold

The stopping criterion in SimCore algorithm requires a threshold value τ. In default, we used $\tau = 0.95$ throughout the experiments. Here, we investigate the sensitivity of SimCore performance to its stopping threshold value. In Table 13, we summarized both sampling ratios and classification accuracies by varying the stopping threshold. We could observe that the τ value of 0.95 generally performs well in both SimCLR and MAE.

	SimCLR with ResNet50						MAE with ViT-B/16					
target	$\tau = 0.99$		$\boldsymbol{\tau = 0.95}$		$\tau = 0.9$		$\tau = 0.99$		$\boldsymbol{\tau = 0.95}$		$\tau = 0.9$	
Aircraft	0.21%	46.7	1.03%	48.3	5.27%	47.6	0.26%	49.5	3.23%	48.1	17.18%	52.9
Cars	0.24%	56.5	0.95%	60.3	4.52%	52.5	0.29%	62.1	1.11%	52.4	5.22%	62.9
Pet	1.96%	79.8	14.36%	79.7	14.36%	79.7	0.45%	52.5	8.47%	77.8	14.36%	75.8
Birds	0.71%	36.2	13.68%	37.7	23.38%	37.7	0.43%	31.1	5.69%	42.1	23.38%	35.3

Table 13. Sensitivity study to the stopping threshold of SimCore. For each threshold value, first column indicates the sampling ratio according to the threshold, and second column indicates the corresponding linear evaluation accuracy.

E.2. Retrieval Model

Prior to the coreset sampling, we pretrain the encoder on a target dataset for short epochs, 1K in our experiments. We refer this model, used in the coreset selection, to a *retrieval model*. This is necessary because we should measure the similarity between the features from the target dataset and from the open-set. Figure 11 presents SimCore performances, given different pretraining epochs of retrieval models. For comparison, the sampling ratio is set to $p = 1\%$, and the random sampling strategy is also included.

As a result, 0.5K ep. model performed marginally worse than other SimCore models, whereas 1K ep. was comparable to 5K ep. model. Still, every SimCore models outperformed random sampling or without *OS*. We should note that 1K ep. pretraining takes up a small portion of the entire training process. On the basis of Pet, 1K pretraining of the retrieval model only costs 4.3% of the total iterations.

However, one can still have a concern on the training cost for the retrieval model. To address this, we conducted an additional experiment, where the models are pretrained on only target datasets for the same number of iterations as our SimCore with the sampling ratio of 1%. For example, we trained the models on Pet for 18,925 epochs to exactly match the iterations to SimCore 1% experiment. Interestingly, we obtained the results as follows, compared to SimCore: 52.26% in Aircraft (+3.81%), 52.97% in Cars (−6.03%), 60.80% in Pet (−16.33%), and 30.59% in Birds (−5.97%). Based on these results, we have confirmed the efficacy of the sampled coreset, as simply increasing the pretraining epochs does not result in the performance improvement.

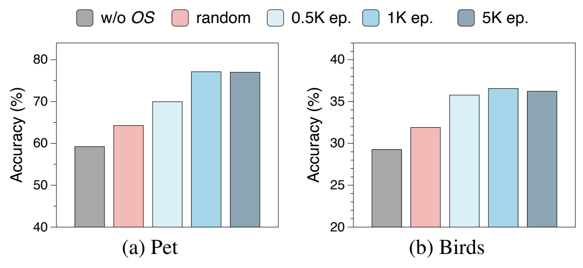
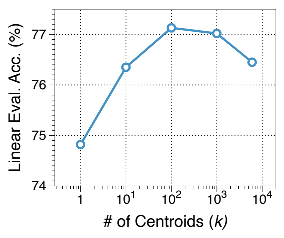


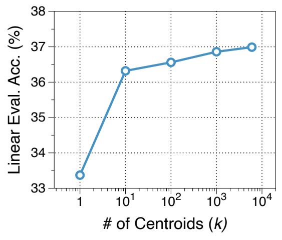
Figure 11. Performance comparisons with different pretraining epochs of retrieval models. "w/o OS": training on X for 5K epochs. "random": random sampling followed by 5K epochs training on X+OS_{rand}. "N ep.": N epochs pretraining on X before the coreset sampling, followed by 5K epochs training on X+OS_{SimCore}.

E.3. Number of Clusters

To reduce the complexity, we have used 100 centroids to calculate the $\hat{f}(\mathcal{S})$ value after k-means clustering. Here, we show that SimCore is robust to the number of k. Figure 12 shows SimCore performance results according to different k values, {1, 10, 10^2, 10^3, $|X|$}, with 1% coreset sampling from ImageNet. Except for the single centroid case, SimCore exhibited overall high accuracies.



(a) Pet: $|X|$ = 3,680



(b) Birds: $|X|$ = 5,990

Figure 12. Sensitivity study of the SimCore performance according to the number of centroids. The maximum feasible centroid number is 3,680 and 5,990 for Pet and Birds, respectively.

F. Additional Experiments

F.1. Two-Stage SSL Pretraining

If a good initialization pretrained on OS is available, we could further pretrain this model on either X ($OS \rightarrow X$) or the union of X and the coreset ($OS \rightarrow X + OS_{\text{SimCore}}$). Therefore, we additionally experimented with the checkpoint of the official code of SimCLR [14], which is already pretrained on the ImageNet dataset. Here, we used an Adam optimizer with a learning rate of 1e-3 and reduced the epochs to 20% compared to training from scratch. Table 14 summarizes the evaluation results of linear probing for three pretraining schemes. In practice, our SimCore outperforms baselines in 10 out of 11 fine-grained datasets. This demonstrates that a well-pretrained model can also be effectively exploited through two-stage pretraining schemes with the SimCore algorithm.

pretrain	Aircraft	Cars	Pet	Birds	Dogs	Flowers	Action	Indoor	Textures	Faces	Food
OS	41.90	45.79	78.35	41.95	61.32	89.67	68.67	68.04	71.81	53.91	88.05
$OS \rightarrow X$	55.77	53.07	77.46	39.97	61.53	91.29	66.25	72.02	72.66	**61.48**	92.41
$OS \rightarrow X + OS_{\text{SimCore}}$	**55.83**	**57.03**	**84.10**	**44.99**	**68.44**	**91.45**	**72.31**	**75.20**	**73.19**	59.09	**92.68**

Table 14. Comparisons between different pretraining schemes on eleven fine-grained datasets. The models are all initialized from the SimCLR's checkpoint that is already pretrained on ImageNet.

F.2. Sampling Strategy

In our default SimCore sampling, we sampled the coreset only once after the training of the retrieval model. In practice, the sampling procedure only takes $\sim$20 minutes, mostly for computing the pairwise similarity (note that SSL pretraining takes $\sim$2 days). There are several available variations on this sampling strategy. One naïve way is to sample a coreset several times during the pretraining, since the model evolves and thus retrieves different coresets throughout the training.

In this sense, we resampled the coreset ($p = 1\%$) three times during 5K epochs training, while maintaining the entire cost for pretraining. As a result, this sampling strategy yielded the performances of 49.99% in Aircraft (+1.54%), 57.63% in Cars ($-$1.37%), 79.75% in Pet (+2.62%), and 38.08% in Birds (+1.52%). Thus, exploring the coreset sampling strategies may be a future work that potentially improves the performance of SimCore.

F.3. Additional Fine-Tuning Schemes

Semi-supervised learning: Table 15 presents the experimental results on fine-tuning using various semi-supervised learning methods, including MixMatch [7], ReMixMatch [6], FixMatch [60], and FlexMatch [80]. We would note that the semi-supervised learning approach in Table 7b has simply fine-tuned models with a small portion of labeled data, following the learning protocols described in [14, 24]. In terms of implementation details, we used the default hyperparameter settings for each method and fine-tuned models for 32 epochs, with iteration steps of 512. In most cases, we found that our SimCore pretraining approach showed the significant performance improvements, regardless of semi-supervised learning methods. This suggests that a self-supervised pretraining with coreset selection, followed by the use of semi-supervised algorithms, can be an effective approach when only a small amount of labeled data is available.

method	pretrain	Aircraft	Cars	Pet	Birds	method	pretrain	Aircraft	Cars	Pet	Birds
MixMatch	X	37.2	38.1	62.0	21.9	FixMatch	X	30.4	**33.8**	52.8	13.7
MixMatch	*OS*	44.3	30.5	73.7	24.4	FixMatch	*OS*	25.1	21.4	63.8	13.3
MixMatch	**SimCore**	**45.1**	**38.6**	**74.3**	**26.0**	FixMatch	**SimCore**	**31.2**	30.5	**67.3**	**13.8**
ReMixMatch	X	48.6	55.5	76.8	34.5	FlexMatch	X	36.9	**43.2**	54.9	**16.7**
ReMixMatch	*OS*	**57.2**	44.5	79.9	33.1	FlexMatch	*OS*	24.2	6.8	65.4	13.4
ReMixMatch	**SimCore**	56.0	**56.4**	**82.6**	**36.7**	FlexMatch	**SimCore**	**41.9**	38.0	**71.8**	10.7

Table 15. Fine-tuning performances with various semi-supervised learning methods. We assumed that only 10% of the data were annotated for semi-supervised fine-tuning.

Active learning: We conducted additional experiments on various active learning methods for fine-tuning the SSL-pretrained models. Table 16 presents the experimental results with four standard active selection methods, such as Random, Entropy [69], Coreset [58], and BADGE [3]. We randomly selected the first 10% of the data and sequentially queried 10% ratio using each active learning algorithm. After annotating the queried samples, we fine-tuned models for 100 epochs, starting from the checkpoint of the previous active learning round. As an effective initialization, our SimCore pretraining approach outperformed both pretraining baselines using either X or *OS*, in the active learning fine-tuning schemes as well.

	Aircraft				Cars				Pet				Birds			
method	10%	20%	30%	40%	10%	20%	30%	40%	10%	20%	30%	40%	10%	20%	30%	40%
pretrain: X																
Random	28.8	45.2	50.0	55.3	26.2	52.8	63.4	71.0	47.2	56.8	59.4	64.8	13.5	24.8	31.9	41.4
Entropy	-	38.9	46.2	53.8	-	47.3	62.9	72.4	-	57.6	60.4	66.2	-	25.5	32.6	40.5
Coreset	-	45.2	51.4	56.3	-	53.3	66.3	74.3	-	57.6	62.7	66.5	-	24.2	31.7	40.2
BADGE	-	45.3	51.3	56.1	-	51.9	**66.6**	73.7	-	58.8	61.4	66.9	-	25.7	31.9	40.4
pretrain: *OS*																
Random	20.7	36.0	45.5	54.8	10.9	32.5	47.1	59.5	35.0	62.0	68.3	72.8	11.2	20.7	20.6	30.1
Entropy	-	34.1	43.3	52.0	-	29.0	45.7	58.6	-	61.9	69.5	75.1	-	21.8	30.3	39.3
Coreset	-	35.3	44.5	51.8	-	27.2	43.4	56.1	-	60.3	71.1	75.4	-	19.9	28.1	38.4
BADGE	-	35.9	46.5	53.6	-	29.5	45.0	58.3	-	62.4	70.4	76.0	-	20.8	30.0	39.7
pretrain: **SimCore**																
Random	33.7	49.6	57.6	62.8	25.2	54.3	63.9	71.5	50.5	66.2	71.6	75.5	8.1	24.7	32.1	**43.0**
Entropy	-	43.4	53.4	62.7	-	49.0	64.2	73.2	-	65.5	71.0	76.4	-	**25.8**	32.7	42.8
Coreset	-	50.1	57.5	63.5	-	52.1	64.9	73.9	-	67.7	72.9	**78.2**	-	23.5	31.3	41.0
BADGE	-	**50.9**	**58.7**	**65.0**	-	**53.4**	66.5	**74.5**	-	**70.0**	**75.2**	77.8	-	24.8	**32.9**	42.4

Table 16. Fine-tuning performances with various active learning methods.

G. Comparisons with Hard Negative Mining

SimCore can be thought of as hard sample mining from the open-set. In SSL literature, hard negative mining (HNM) is a well-known technique for leveraging the informative sample features. Robinson *et al.* [55] proposed an implicit method for mining the negatives that are similar to an anchor (z_i in Eq. 1), while Wang *et al.* [70] suggested explicit sampling to inflict large gradient penalties.

Table 17 compares SimCore with the existing HNM approaches. Hard negative sampling (HNS) and explicit HNS (EHNS) slightly improve the performance; however, both still use all open-set samples, some of which may not be relevant to the target. In other words, they employ hard negatives for every open-set anchors, which may reduce the performance due to the distribution mismatch to the target dataset.

SimCore, on the other hand, retrieves only the coreset, reducing the need for an additional negative mining in the loss function. Nonetheless, SimCore can be applied to any SSL method including HNM-based losses. Thus, we have tried using HNS loss after the explicit coreset sampling, achieving a small gain in the Birds dataset.

method	pretrain data	hard (exp.)	hard (imp.)	Pet	Birds
SimCLR [14]	X	✗	✗	58.2	25.9
HNS [55]	X+OS	✗	✓	63.6	30.0
EHNS [70]	X+OS	✓	✗	62.9	27.9
EHNS-m [70]	X+OS (mem.)	✓	✗	53.4	18.7
SimCore (ours)	X+coreset	✓	✗	**72.7**	**31.3**
SimCore + HNS	X+coreset	✓	✓	69.3	**33.1**

Table 17. Comparisons with hard negative mining techniques. We compared each method by the usage of explicit or implicit hard negative sampling and the pretraining data. ResNet18 is used for the above experiments.

Explicit HNS with memory bank: We have compared SimCore with previous hard negative mining techniques: HNS [55] and EHNS [70]. However, they only slightly improved the performance because they still use all open-set samples as an anchor. To this end, we modified the sampling strategy of EHNS to better align with our conception.

Specifically, because EHNS is an explicit sampling method based on the similarities with an anchor, we can only use target data samples as the anchor and utilize hard negatives in the open-set. There are two ways to make it available:

1. We can sample a minibatch from X+OS as before, but only calculates the EHNS loss with the target anchors.
2. We can sample a minibatch only from X. The negative pairs are from other target sample's features and open-set features, where the open-set features are saved in a memory bank [28].

For the first strategy, the size difference between OS and X is so large that there are not enough target samples in a minibatch, which prevented the model from being converged. The second strategy is more reasonable in that a minibatch is comprised of only X samples, while the explicit hard negative sampling is done with other target features and open-set features.

Table 17 presents the result of EHNS with memory bank (EHNS-m), showing that EHNS-m is not much effective. This is because negative pairs from the memory bank do not impose any gradients–only the target samples are directly involved in learning. To make the pretraining effective, therefore, it is crucial to sample a coreset and employ the entire coreset samples into the training.