JUNE 18-22, 2023 CVPR VANCOUVER, CANADA

Coreset Sampling from Open-Set for Fine-Grained Self-Supervised Learning

The IEEE/CVF Conference on Computer Vision and Pattern Recognition 2023

Sungnyun Kim*, Sangmin Bae*, Se-Young Yun

(*: equal contribution)

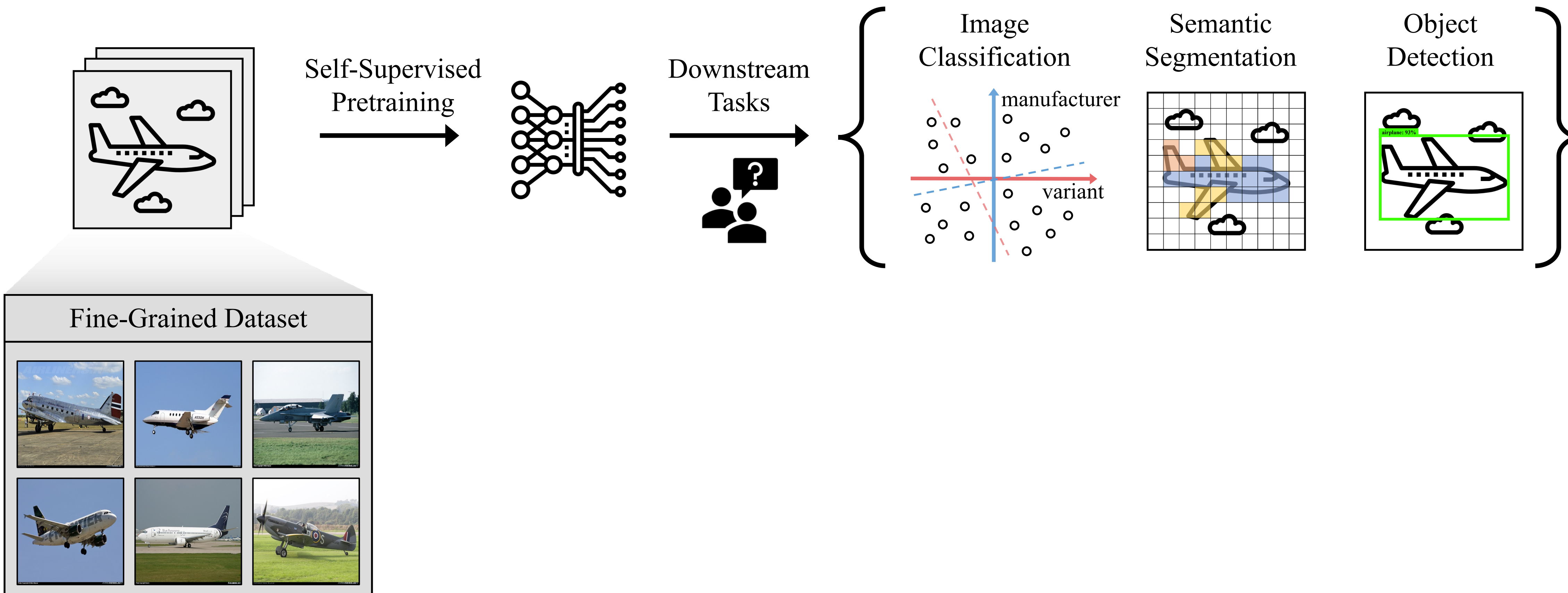
Poster ID: **TUE-PM-326**

KAIST AI Kim Jaechul Graduate School

OSI Optimization and Statistical Inference LAB

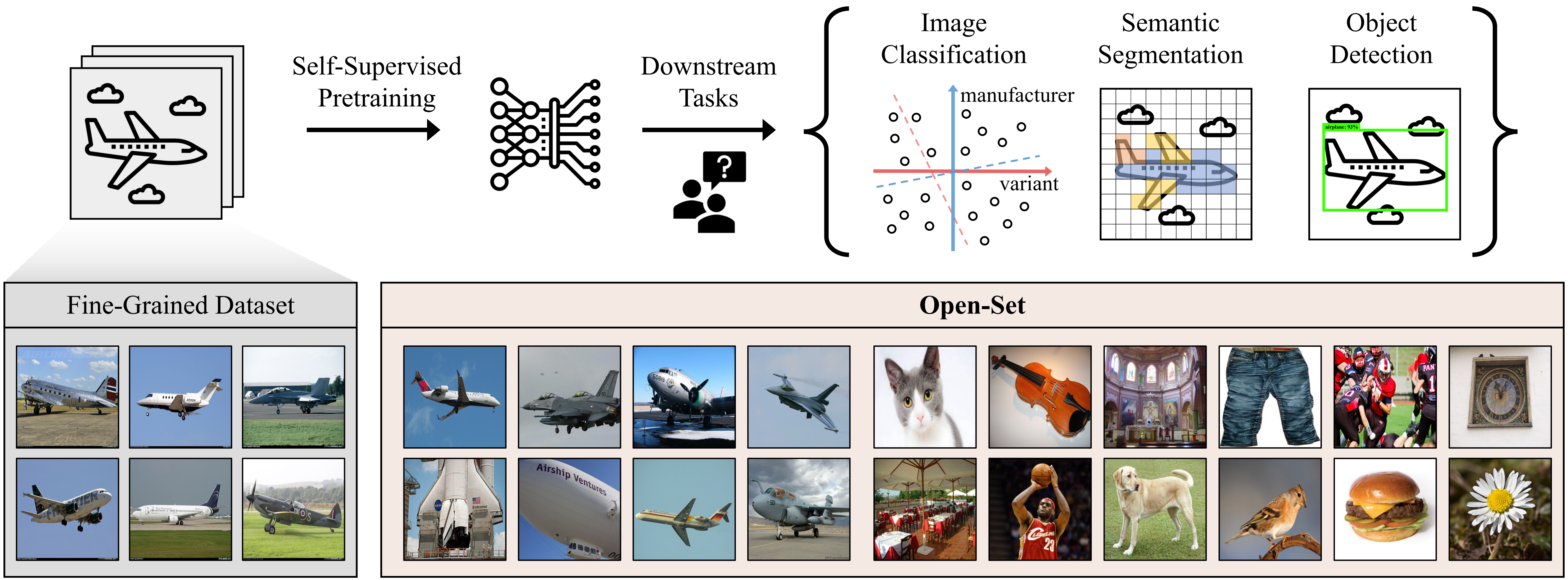
Highlights of the Paper

Self-Supervised Learning (SSL) is a promising approach for fine-grained tasks.



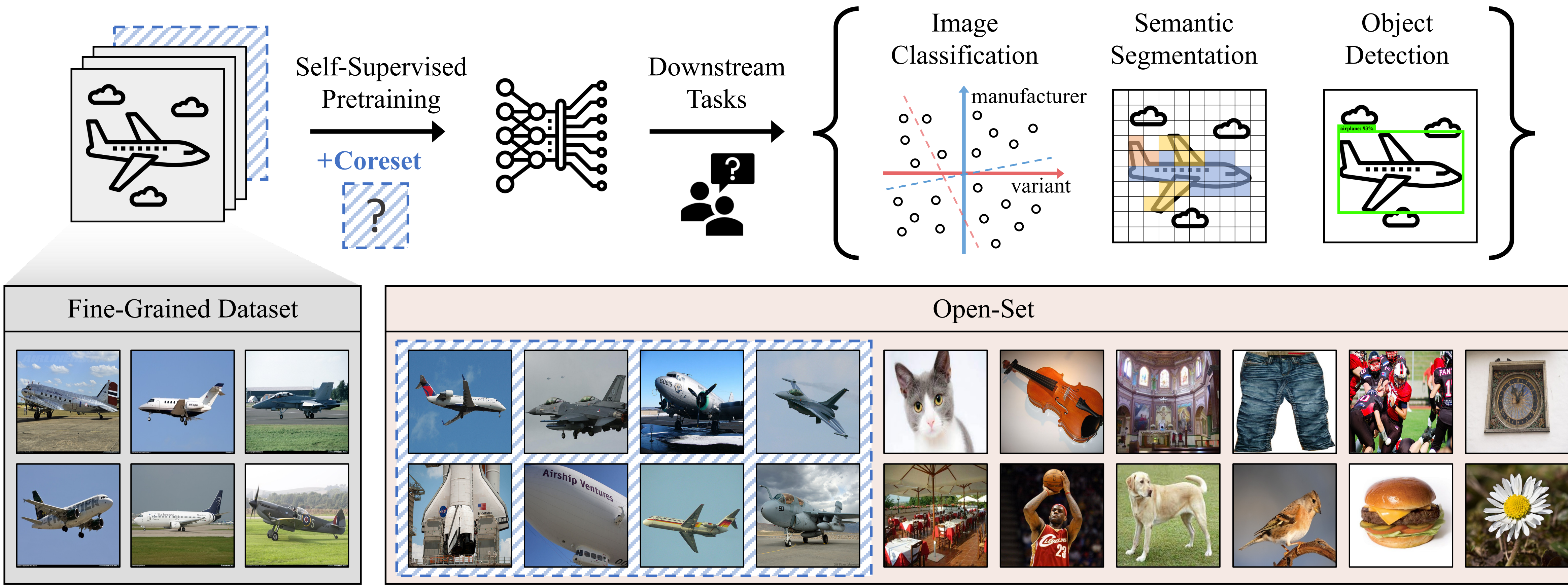
Highlights of the Paper

We can assume an **Open-Set** to build a versatile model.



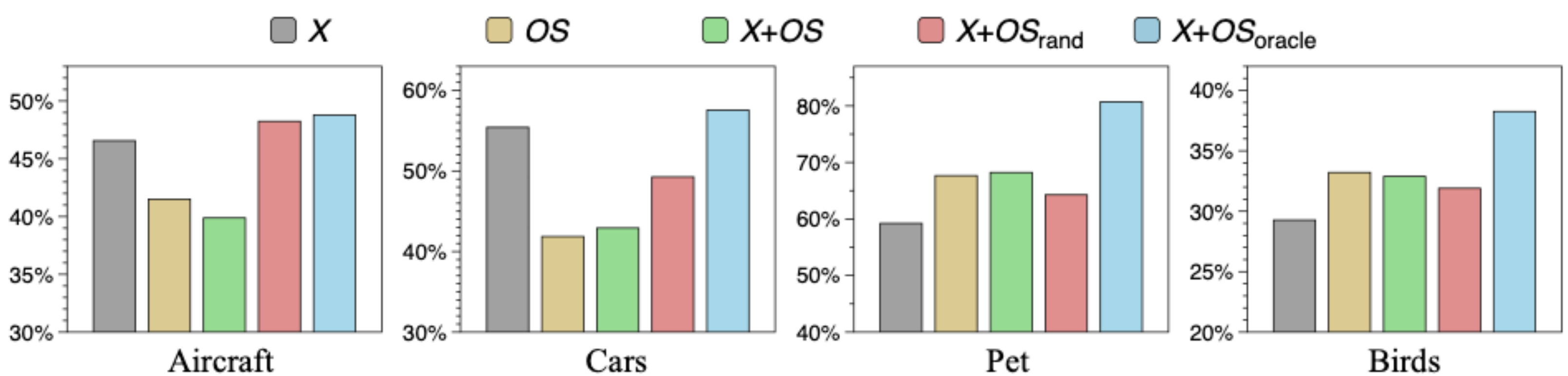
Highlights of the Paper

We propose a novel and simple coreset sampling algorithm, **SimCore**.



Motivating Experiments

Addressing distribution mismatch between fine-grained dataset and open-set is a critical issue.



Target (X)	Classes for OS_{oracle} (#)
Aircraft	airliner, warplane, ... (8)
Cars	convertible, jeep, ... (10)
Pet	Persian cat, beagle, ... (24)
Birds	goldfinch, junco, ... (20)

1. SSL on the **open-set** (OS) does not always outperform SSL on **target dataset** (X).
2. Selecting **relevant class samples** (OS_{oracle}) show the significant performance gains.

Problem Setting

We first propose a realistic OpenSSL task, where sampling the coreset is important on SSL performance.

Task [ref.]	Problem Setting	Train (Labeled)	Train (Unlabeled)	Test	Definition of OS/CS	Main Goal
Novel Class Discovery [26, 30, 81]	test data consist of only novel classes	seen	-	novel	-	cluster novel classes in test dataset
Open-Set Recognition [5, 12, 57, 68]	test set contains seen and novel classes	seen	-	seen + novel	[OS] test dataset containing seen and novel classes	reject instances from novel classes at test time
Webly Sup. [15, 39, 62]	train data contains web-crawled noisy samples	partially noisy	-	seen	[OS] web-crawled train dataset containing noisy samples	robustly train instances with corrupted labels
Open-Set Semi-Sup. [17, 33, 51, 56, 61]	unlabeled train data contain novel classes	seen	seen + novel	seen	[OS] training dataset containing seen and novel classes	train a robust model while regularizing novel classes
Open-World Semi-Sup. [4, 8, 9]	train and test data contain novel classes	seen	seen + novel	seen + novel	[OS] dataset containing seen and novel classes	discover novel classes and assign samples at test time
Open-Set Annotation [50]	unlabeled data pool contains novel classes	seen	seen* + novel*	seen	[OS] unlabeled data pool with seen and novel classes	aim to query seen classes from unlabeled data pool
Coreset Selection in AL [58, 72]	query instances to be annotated	seen	seen*	seen	[CS] the most representative subset of unlabeled set	find a small subset competitive to whole dataset
Coreset Selection in CL [1, 65, 78]	continuously learn a sequence of tasks	partially novel	-	seen	[CS] the most representative instances at each task	promote task adaptation with less catastrophic forgetting
Hard Negative Mining in Self-Sup. [55, 70]	assume that hard negatives are helpful	-	target	target	[CS] the hardest contrastive pair instances for SSL	improve SSL performance using core-negative instances
Open-Set Self-Sup. [ours]	utilize open-set in pretraining, which may have irrelevant data	-	target + irrelevant	target	[OS] large-scale unlabeled set [CS] subset of OS sharing the same semantics with target set	improve SSL performance on fine-grained datset via coreset sampling method

Method: SimCore Algorithm

We introduce a **sim**ple **core**set sampling algorithm, coined as **SimCore**.

Algorithm 1: Simple coreset sampling from open-set

Require: E_θ: encoder pretrained on X;
Require: $\mathcal{U}_0$: initial candidate set (open-set);
Require: $\mathcal{B}$, τ: coreset budget, threshold;
initialize $\mathcal{I} \leftarrow \emptyset$, $t \leftarrow 0$;
replace $\hat{X} \leftarrow$ cluster centroids of X;
calculate $z_x, z_u \leftarrow E_\theta(x), E_\theta(u)$ for $\forall x, u \in \hat{X} \times \mathcal{U}_0$;
while $|\mathcal{I}| < \mathcal{B}$ **do**
  set $\mathcal{S}_t^*$ as the elements in $\mathcal{U}_t$ that are closest to each element in $\hat{X}$ (Eq. 2);
  $\mathcal{I} \leftarrow \mathcal{I} \cup \mathcal{S}_t^*,\ \mathcal{U}_{t+1} \leftarrow \mathcal{U}_t \setminus \mathcal{S}_t^*$
  $t \leftarrow t + 1$
  // stopping criterion
  if $\hat{f}(\mathcal{S}_t^*) < \tau \cdot \hat{f}(\mathcal{S}_1^*)$ **then**
    stop sampling;
  end
end
re-initialize θ and pretrain E_θ with $X \cup \mathcal{I}$;

Method: SimCore Algorithm

We introduce a **sim**ple **core**set sampling algorithm, coined as **SimCore**.

Algorithm 1: Simple coreset sampling from open-set

Require: E_θ: encoder pretrained on X; → Retrieval model
Require: $\mathcal{U}_0$: initial candidate set (open-set);
Require: $\mathcal{B}, \tau$: coreset budget, threshold;
initialize $\mathcal{I} \leftarrow \emptyset$, $t \leftarrow 0$;
replace $\hat{X} \leftarrow$ cluster centroids of X;
calculate $z_x, z_u \leftarrow E_\theta(x), E_\theta(u)$ for $\forall x, u \in \hat{X} \times \mathcal{U}_0$;
while $|\mathcal{I}| < \mathcal{B}$ **do**
  set $\mathcal{S}_t^*$ as the elements in $\mathcal{U}_t$ that are closest to each element in $\hat{X}$ (Eq. 2);
  $\mathcal{I} \leftarrow \mathcal{I} \cup \mathcal{S}_t^*,\ \mathcal{U}_{t+1} \leftarrow \mathcal{U}_t \setminus \mathcal{S}_t^*$
  $t \leftarrow t + 1$
  // stopping criterion
  if $\hat{f}(\mathcal{S}_t^*) < \tau \cdot \hat{f}(\mathcal{S}_1^*)$ **then**
    stop sampling;
  end
end
re-initialize θ and pretrain E_θ with $X \cup \mathcal{I}$;

Method: SimCore Algorithm

We introduce a **sim**ple **core**set sampling algorithm, coined as **SimCore**.

Algorithm 1: Simple coreset sampling from open-set

Require: E_θ: encoder pretrained on X;
Require: $\mathcal{U}_0$: initial candidate set (open-set);
Require: $\mathcal{B}$, τ: coreset budget, threshold;
initialize $\mathcal{I} \leftarrow \emptyset$, $t \leftarrow 0$;
replace $\hat{X} \leftarrow$ cluster centroids of X;
calculate $z_x, z_u \leftarrow E_\theta(x), E_\theta(u)$ for $\forall x, u \in \hat{X} \times \mathcal{U}_0$;
while $|\mathcal{I}| < \mathcal{B}$ **do**
  set $\mathcal{S}_t^*$ as the elements in $\mathcal{U}_t$ that are closest to each element in $\hat{X}$ (Eq. 2);
  $\mathcal{I} \leftarrow \mathcal{I} \cup \mathcal{S}_t^*$, $\mathcal{U}_{t+1} \leftarrow \mathcal{U}_t \setminus \mathcal{S}_t^*$
  $t \leftarrow t + 1$
  // stopping criterion
  if $\hat{f}(\mathcal{S}_t^*) < \tau \cdot \hat{f}(\mathcal{S}_1^*)$ **then**
    stop sampling;
  end
end
re-initialize θ and pretrain E_θ with $X \cup \mathcal{I}$;

→ Finding a subset S that maximizes Eq. 2:

$$f(\mathcal{S}) = \sum_{x \in X} \max_{u \in \mathcal{S}} w(x, u), \text{ where } \mathcal{S} \subseteq \mathcal{U}, \ \mathcal{U} \cap X = \emptyset$$

Method: SimCore Algorithm

We introduce a **sim**ple **core**set sampling algorithm, coined as **SimCore**.

Algorithm 1: Simple coreset sampling from open-set

Require: E_θ: encoder pretrained on X;
Require: $\mathcal{U}_0$: initial candidate set (open-set);
Require: $\mathcal{B}, \tau$: coreset budget, threshold;
initialize $\mathcal{I} \leftarrow \emptyset, t \leftarrow 0$;
replace $\hat{X} \leftarrow$ cluster centroids of X;
calculate $z_x, z_u \leftarrow E_\theta(x), E_\theta(u)$ for $\forall x, u \in \hat{X} \times \mathcal{U}_0$;
while $|\mathcal{I}| < \mathcal{B}$ **do**
    set $\mathcal{S}_t^*$ as the elements in $\mathcal{U}_t$ that are closest to each element in $\hat{X}$ (Eq. 2);
    $\mathcal{I} \leftarrow \mathcal{I} \cup \mathcal{S}_t^*,\ \mathcal{U}_{t+1} \leftarrow \mathcal{U}_t \setminus \mathcal{S}_t^*$
    $t \leftarrow t+1$
    // stopping criterion
    if $\hat{f}(\mathcal{S}_t^*) < \tau \cdot \hat{f}(\mathcal{S}_1^*)$ **then**
        stop sampling;
    end
end
re-initialize θ and pretrain E_θ with $X \cup \mathcal{I}$;

→ Finding a subset S that maximizes Eq. 2:

$$\hat{f}(S) = \sum_{x \in \hat{X}} \max_{u \in \mathcal{S}} w(x, u), \text{ where } \mathcal{S} \subseteq \mathcal{U},\ \mathcal{U} \cap X = \emptyset$$

Method: SimCore Algorithm

We introduce a **sim**ple **core**set sampling algorithm, coined as **SimCore**.

Algorithm 1: Simple coreset sampling from open-set

Require: E_θ: encoder pretrained on X;
Require: $\mathcal{U}_0$: initial candidate set (open-set);
Require: $\mathcal{B}$, τ: coreset budget, threshold;
initialize $\mathcal{I} \leftarrow \emptyset$, $t \leftarrow 0$;
replace $\hat{X} \leftarrow$ cluster centroids of X;
calculate $z_x, z_u \leftarrow E_\theta(x), E_\theta(u)$ for $\forall x, u \in \hat{X} \times \mathcal{U}_0$;
while $|\mathcal{I}| < \mathcal{B}$ **do**
    set $\mathcal{S}_t^*$ as the elements in $\mathcal{U}_t$ that are closest to each element in $\hat{X}$ (Eq. 2);
    $\mathcal{I} \leftarrow \mathcal{I} \cup \mathcal{S}_t^*$, $\mathcal{U}_{t+1} \leftarrow \mathcal{U}_t \setminus \mathcal{S}_t^*$
    $t \leftarrow t + 1$
    // stopping criterion
    if $\hat{f}(\mathcal{S}_t^*) < \tau \cdot \hat{f}(\mathcal{S}_1^*)$ **then**
        stop sampling;
    end
end
re-initialize θ and pretrain E_θ with $X \cup \mathcal{I}$;

→ Finding a subset S^* that maximizes Eq. 2:

$$\hat{f}(S) = \sum_{x \in \hat{X}} \max_{u \in \mathcal{S}} w(x, u), \text{ where } \mathcal{S} \subseteq \mathcal{U},\ \mathcal{U} \cap X = \emptyset$$

Method: SimCore Algorithm

We introduce a **sim**ple **core**set sampling algorithm, coined as **SimCore**.

Algorithm 1: Simple coreset sampling from open-set

Require: E_θ: encoder pretrained on X;
Require: $\mathcal{U}_0$: initial candidate set (open-set);
Require: $\mathcal{B}$, τ: coreset budget, threshold;
initialize $\mathcal{I} \leftarrow \emptyset$, $t \leftarrow 0$;
replace $\hat{X} \leftarrow$ cluster centroids of X;
calculate $z_x, z_u \leftarrow E_\theta(x), E_\theta(u)$ for $\forall x, u \in \hat{X} \times \mathcal{U}_0$;
while $|\mathcal{I}| < \mathcal{B}$ **do**
    set $\mathcal{S}_t^*$ as the elements in $\mathcal{U}_t$ that are closest to each element in $\hat{X}$ (Eq. 2);
    $\mathcal{I} \leftarrow \mathcal{I} \cup \mathcal{S}_t^*,\ \mathcal{U}_{t+1} \leftarrow \mathcal{U}_t \setminus \mathcal{S}_t^*$
    $t \leftarrow t + 1$
    // stopping criterion
    if $\hat{f}(\mathcal{S}_t^*) < \tau \cdot \hat{f}(\mathcal{S}_1^*)$ **then**
        stop sampling;
    end
end
re-initialize θ and pretrain E_θ with $X \cup \mathcal{I}$;

→ Iterative coreset sampling

Linear Evaluation Performance

For 11 fine-grained datasets, open-set does not always improve the performance.

		Target dataset (X) and its number of samples										
pretrain	p	Aircraft 6,667	Cars 8,144	Pet 3,680	Birds 5,990	Dogs 12,000	Flowers 2,040	Action 4,000	Indoor 5,360	Textures 3,760	Faces 4,263	Food 13,296
X	-	46.56	55.42	59.23	29.27	49.88	80.14	43.76	54.10	58.78	56.63	87.99
OS	-	41.50	41.86	67.66	33.21	49.94	85.67	60.65	64.46	67.23	52.84	86.14
X+OS	-	39.88	42.92	68.22	32.88	50.42	85.34	60.61	63.66	67.98	52.76	85.90

Linear Evaluation Performance

SimCore with certain sampling ratios has showed the performance gains.

pretrain	p	Aircraft 6,667	Cars 8,144	Pet 3,680	Birds 5,990	Dogs 12,000	Flowers 2,040	Action 4,000	Indoor 5,360	Textures 3,760	Faces 4,263	Food 13,296
		Target dataset (X) and its number of samples										
X	-	46.56	55.42	59.23	29.27	49.88	80.14	43.76	54.10	58.78	56.63	87.99
OS	-	41.50	41.86	67.66	33.21	49.94	85.67	60.65	64.46	67.23	52.84	86.14
X+OS	-	39.88	42.92	68.22	32.88	50.42	85.34	60.61	63.66	67.98	52.76	85.90
X+OS_{rand}	1%	48.24	49.26	64.27	31.90	49.62	83.17	47.25	55.37	61.33	57.37	88.08
X+$OS_{\text{SimCore}^\dagger}$	1%	48.06	58.56	74.82	33.37	57.42	82.12	51.37	57.84	61.76	56.95	90.35
X+OS_{SimCore}	1%	**48.45**	59.00	77.13	36.56	59.83	86.70	52.98	59.18	63.40	58.85	89.78
X+OS_{rand}	5%	45.75	46.03	68.38	33.63	50.24	84.52	57.27	60.71	65.80	56.05	87.75
X+$OS_{\text{SimCore}^\dagger}$	5%	45.57	50.75	80.20	35.56	64.62	85.11	64.53	68.13	66.22	58.93	89.87
X+OS_{SimCore}	5%	47.14	52.22	**81.75**	**39.21**	**66.82**	**87.28**	66.38	70.96	**68.13**	**59.34**	90.74

Linear Evaluation Performance

With a stopping criterion, SimCore automatically sample the enough amount of coreset.

		Target dataset (X) and its number of samples										
pretrain	p	Aircraft 6,667	Cars 8,144	Pet 3,680	Birds 5,990	Dogs 12,000	Flowers 2,040	Action 4,000	Indoor 5,360	Textures 3,760	Faces 4,263	Food 13,296
X	-	46.56	55.42	59.23	29.27	49.88	80.14	43.76	54.10	58.78	56.63	87.99
OS	-	41.50	41.86	67.66	33.21	49.94	85.67	60.65	64.46	67.23	52.84	86.14
X+OS	-	39.88	42.92	68.22	32.88	50.42	85.34	60.61	63.66	67.98	52.76	85.90
X+OS_{rand}	1%	48.24	49.26	64.27	31.90	49.62	83.17	47.25	55.37	61.33	57.37	88.08
X+$OS_{\text{SimCore}\dagger}$	1%	48.06	58.56	74.82	33.37	57.42	82.12	51.37	57.84	61.76	56.95	90.35
X+OS_{SimCore}	1%	**48.45**	59.00	77.13	36.56	59.83	86.70	52.98	59.18	63.40	58.85	89.78
X+OS_{rand}	5%	45.75	46.03	68.38	33.63	50.24	84.52	57.27	60.71	65.80	56.05	87.75
X+$OS_{\text{SimCore}\dagger}$	5%	45.57	50.75	80.20	35.56	64.62	85.11	64.53	68.13	66.22	58.93	89.87
X+OS_{SimCore}	5%	47.14	52.22	**81.75**	**39.21**	**66.82**	**87.28**	66.38	70.96	**68.13**	**59.34**	90.74
Stopping Criterion		1.03%	0.95%	14.4%	13.7%	9.72%	7.96%	15.6%	13.5%	5.89%	0.27%	3.86%
X+OS_{SimCore}	-	48.27	**60.29**	79.66	37.65	66.48	87.04	**67.46**	**71.95**	67.66	59.01	**91.31**

Robustness of SimCore

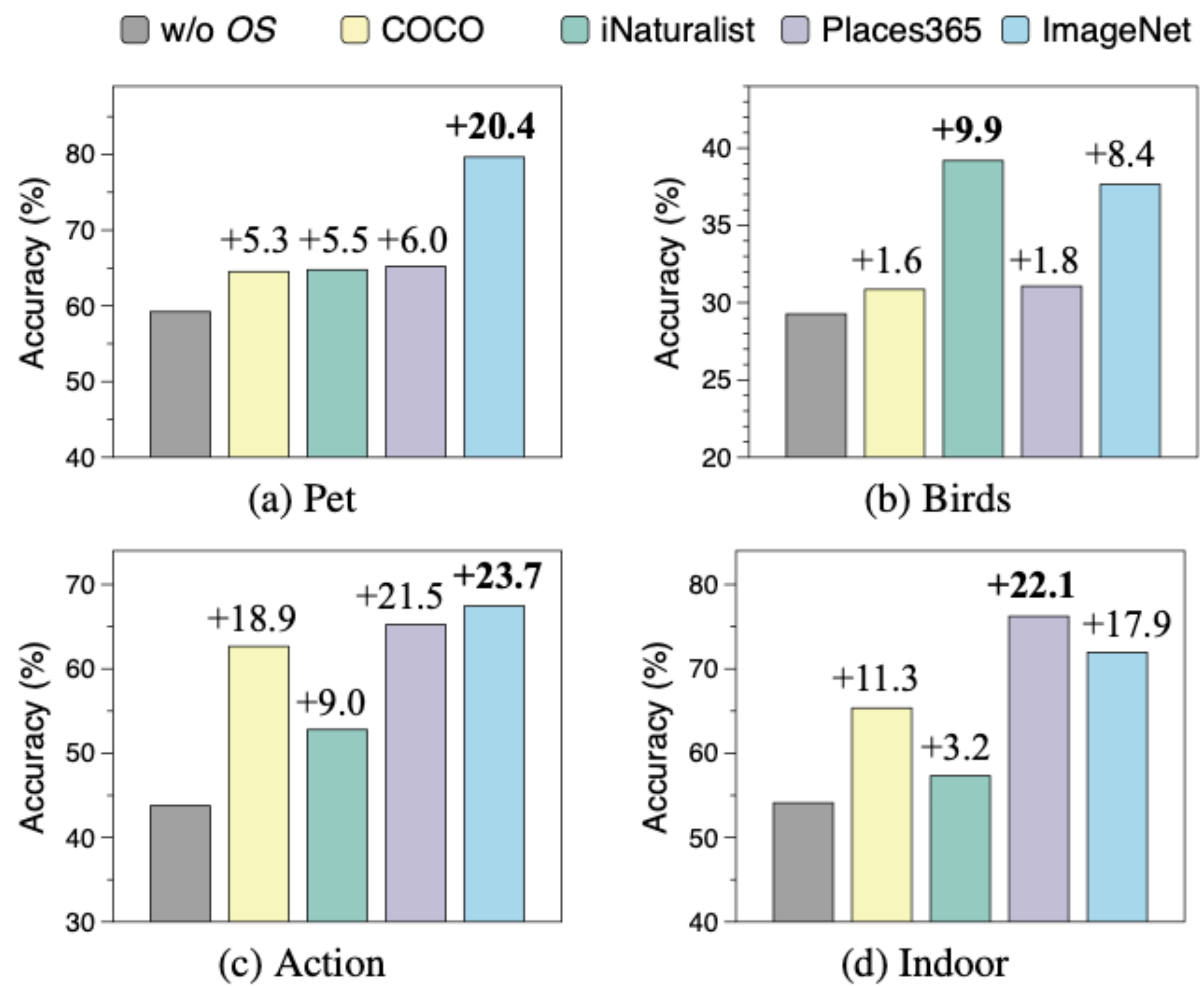
SimCore is robust to various architectures and SSL methods.

method	architecture	pretrain	Aircraft	Cars	Pet	Birds
SimCLR	EfficientNet	*X*	25.5	37.0	58.1	27.8
SimCLR	EfficientNet	*OS*	31.6	29.5	57.8	26.5
SimCLR	EfficientNet	**SimCore**	**41.7**	**52.8**	**69.5**	**29.6**
SimCLR	ResNet18	*X*	43.4	51.9	58.2	25.9
SimCLR	ResNet18	*OS*	33.9	33.1	62.5	27.7
SimCLR	ResNet18	**SimCore**	**44.5**	**55.1**	**72.7**	**31.3**
SimCLR	ResNeXt50	*X*	45.9	56.5	63.4	28.6
SimCLR	ResNeXt50	*OS*	39.2	39.4	68.2	32.6
SimCLR	ResNeXt50	**SimCore**	**49.5**	**59.5**	**81.0**	**37.4**
SimCLR	ResNet101	*X*	49.4	54.5	64.0	29.1
SimCLR	ResNet101	*OS*	40.4	41.9	69.5	34.2
SimCLR	ResNet101	**SimCore**	**50.9**	**58.8**	**83.0**	**39.1**

method	architecture	pretrain	Aircraft	Cars	Pet	Birds
BYOL	ResNet50	*X*	40.6	49.4	56.5	27.6
BYOL	ResNet50	*OS*	46.1	49.6	78.4	44.7
BYOL	ResNet50	**SimCore**	**46.5**	**50.4**	**85.1**	**47.9**
SwAV	ResNet50	*X*	34.5	42.4	49.4	21.6
SwAV	ResNet50	*OS*	33.8	30.0	64.2	27.3
SwAV	ResNet50	**SimCore**	**45.0**	**45.1**	**80.2**	**36.6**
DINO	ViT-Ti/16	*X*	27.3	**48.2**	42.4	28.5
DINO	ViT-Ti/16	*OS*	42.0	39.1	78.4	61.2
DINO	ViT-Ti/16	**SimCore**	**43.2**	47.2	**83.3**	**72.6**
MAE	ViT-B/16	*X*	**55.9**	44.7	56.3	32.2
MAE	ViT-B/16	*OS*	39.8	37.3	68.3	31.4
MAE	ViT-B/16	**SimCore**	48.1	**52.4**	**77.8**	**42.1**

Various Open-Sets

With curated open-sets, SimCore can sample relevant coresets.

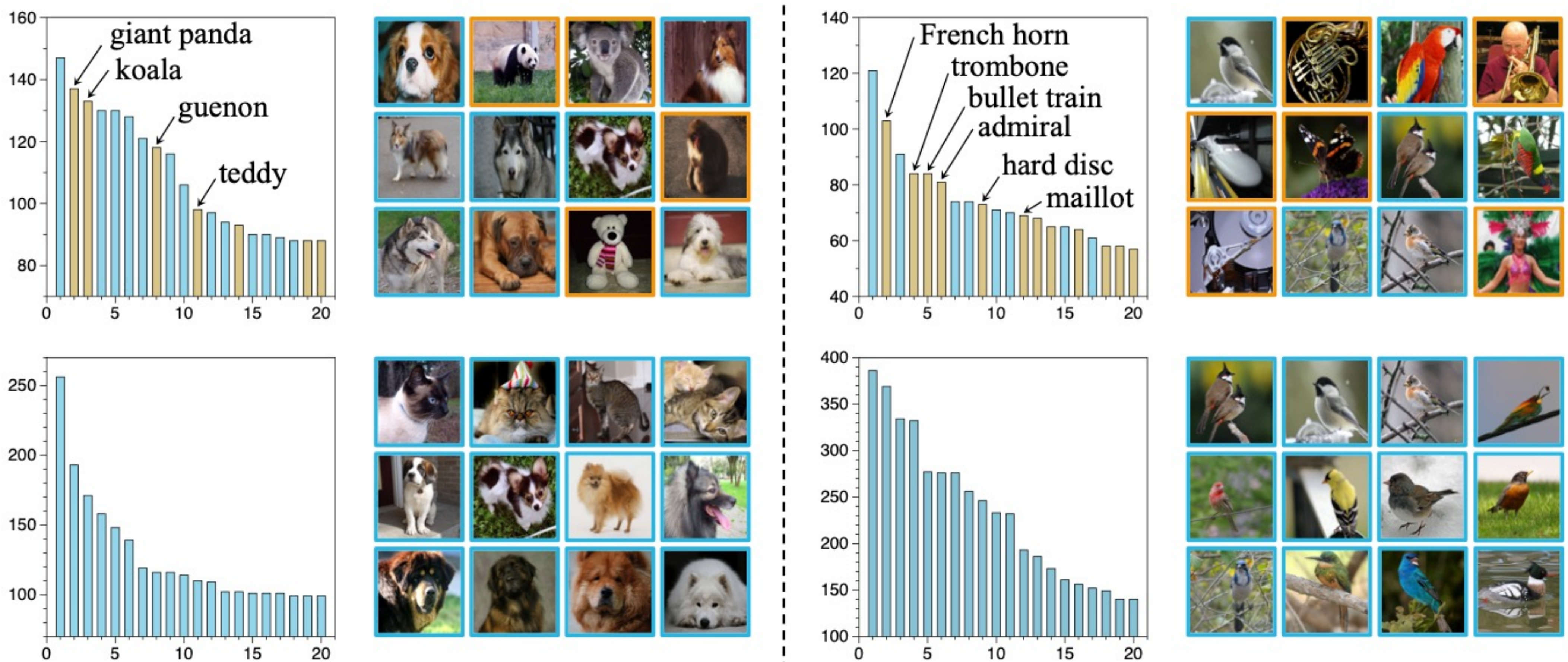


Qualitative Evaluation

We visualized selected samples of Places365 (top) and iNaturalist coreset (bottom).

Qualitative Evaluation

SimCore with larger k centroids could sample the similar classes samples.



(a) OS_{SimCore} with X = Pet and $\{k = 1 \text{ (top)}, k = 100 \text{ (bottom)}\}$

(b) OS_{SimCore} with X = Birds and $\{k = 1 \text{ (top)}, k = 100 \text{ (bottom)}\}$

Downstream Tasks

SimCore outperforms baselines on various downstream tasks.

	Aircraft		Cars		Pet		Birds	
pretrain	20	200	20	200	20	200	20	200
X	36.1	36.7	33.1	34.3	52.0	51.8	**20.7**	**21.8**
OS	19.3	17.7	11.4	10.9	50.4	49.0	13.9	15.1
SimCore	**40.7**	**41.4**	**33.8**	**34.6**	**61.4**	**61.4**	18.3	19.2

(a) kNN classification

	Aircraft			Cars			Pet			Birds		
pretrain	10%	20%	50%	10%	20%	50%	10%	20%	50%	10%	20%	50%
X	29.0	47.6	64.6	25.1	53.5	80.2	47.2	58.7	71.4	**13.3**	25.2	51.2
OS	19.6	34.1	43.9	10.8	35.7	74.1	35.7	62.3	76.9	10.1	21.0	51.2
SimCore	**33.9**	**51.3**	**65.6**	**25.4**	**55.3**	**81.6**	**50.9**	**70.4**	**79.7**	11.9	**25.5**	**55.7**

(b) Semi-supervised learning

	Aircraft		Cars		Pet		Birds	
pretrain	mAP	mAP_{50}	mAP	mAP_{50}	IoU_{fg}	IoU_{bg}	IoU_{fg}	IoU_{bg}
X	10.8	12.7	34.7	40.0	79.1	82.0	65.3	92.6
OS	23.7	27.0	20.8	23.6	79.8	82.8	67.9	93.3
SimCore	**29.6**	**36.8**	**37.6**	**43.2**	**80.0**	**83.1**	**68.4**	**93.4**

(c) Object detection and pixel-wise segmentation

	Aircraft		Cars		Faces			
pretrain	mfr.	family	brand	type	pointy	oval	young	smiling
X	21.1	40.4	67.3	78.0	66.8	83.4	93.1	93.4
OS	17.9	35.2	49.3	61.3	64.9	81.9	92.9	86.3
SimCore	**21.9**	**41.9**	**70.7**	**80.1**	**67.5**	**83.9**	**93.6**	**93.7**

(d) Multi-attribute classification

Comparisons to Open-Set Framework

We have compared our OpenSSL framework to two different frameworks, which utilize unlabeled or noisy-labeled open-sets.

	framework: *Open-Set Semi-Sup.*				framework: *Webly Sup.*			
pretrain	method	Aircraft	Cars	Birds	method	Aircraft	Cars	Birds
SimCore	FT (50%)	73.5	80.1	57.4	FT (100%)	84.3	**89.3**	70.6
✗	SelfTrain	51.9	55.5	35.7	CoTeach	79.3	51.7	70.4
SimCore	SelfTrain	**78.1**	**81.3**	**59.1**	CoTeach	**89.8**	57.0	**78.9**
✗	OpenMatch	70.1	70.2	52.3	DivideMix	82.2	54.4	74.5
SimCore	OpenMatch	**83.5**	**89.5**	**66.4**	DivideMix	**86.5**	56.5	**80.0**

Comparisons to Open-Set Framework

We have compared our OpenSSL framework to two different frameworks, which utilize unlabeled or noisy-labeled open-sets.

	framework: *Open-Set Semi-Sup.*				framework: *Webly Sup.*			
pretrain	method	Aircraft	Cars	Birds	method	Aircraft	Cars	Birds
SimCore	FT (50%)	73.5	80.1	57.4	FT (100%)	84.3	**89.3**	70.6
✗	SelfTrain	51.9	55.5	35.7	CoTeach	79.3	51.7	70.4
SimCore	SelfTrain	**78.1**	**81.3**	**59.1**	CoTeach	**89.8**	57.0	**78.9**
✗	OpenMatch	70.1	70.2	52.3	DivideMix	82.2	54.4	74.5
SimCore	OpenMatch	**83.5**	**89.5**	**66.4**	DivideMix	**86.5**	56.5	**80.0**

1. When the SimCore model is simply fine-tuned on each target, it outperforms others.
2. SimCore can synergize with both frameworks, serving as an effective initialization.

Thank You!

Please check our paper and come to TUE-PM-326 :)

Sungnyun Kim: ksn4397@kaist.ac.kr

Sangmin Bae: bsmn0223@kaist.ac.kr