Appendix

We provide additional details in Appendix A, Appendix B, Appendix C and Appendix D as well as some additional quantitative results in Appendix E and qualitative visualizations in Appendix F.

A. Self-distillation asymmetry abstraction

In Section 3.1, we state that self-distillation methods avoid collapse by using asymmetry and claim that this asymmetry can be abstracted out using two asymmetric encoders f and f'. Given a distance metric d, a self-distillation objective is made solely out of positive terms of the form:

$$\mathcal{L}_{distil} = d(f(\boldsymbol{x}), f'(\boldsymbol{x}^+)) \tag{2}$$

where $(\boldsymbol{x}, \boldsymbol{x}^+)$ is a positive pair. The total self-distillation objective is Equation (2) summed over all positive pairs $(\boldsymbol{x}, \boldsymbol{x}^+)$. Below, we explicit the form of the encoders and the distance metric, both for SimSiam [12] and DINO [8].

A.1. SimSiam [12]

Using notation from the original paper, SimSiam defines an encoder $f\colon \mathcal{X} \to \mathcal{Z}$ and a predictor $p\colon \mathcal{Z} \to \mathcal{Z}$. Using our notations, we encapsulate both the original encoder f and the predictor p in a single encoder which we also denote by f and define our f' as the f from SimSiam[3]:

$$f' \triangleq f \tag{A1}$$

$$f \triangleq p \circ f \tag{A2}$$

The distance metric used is the negative cosine similarity. Given the above, the self-distillation loss in SimSiam can be written as

$$d(f(\boldsymbol{x}), f'(\boldsymbol{x}^+)) := -\frac{f(\boldsymbol{x})^\top f'(\boldsymbol{x}^+)}{\|f(\boldsymbol{x})\|_2 \, \|f'(\boldsymbol{x}^+)\|_2} \tag{A3}$$

This loss is minimized w.r.t. the weights of f (no gradients are back-propagated through f').

A.2. DINO [8]

Using notation from the original paper, DINO uses a student backbone $g_{\theta_s}\colon \mathcal{X} \to \mathcal{P}$ where $\mathcal{P}$ is the space of discrete probability mass functions. An analogous teacher backbone g_{θ_t} is defined as a smoothed version of g_{θ_s}. At the end of each epoch, the weights of the teacher backbone are updated with $\theta_t \leftarrow \lambda\theta_t + (1-\lambda)\theta_s$ where θ_s and θ_t refer to the weights of the student and teacher backbone, respectively.

[3] the notation on the right side of Equation (A1) and Equation (A2) refers to notation from SimSiam, and the left side refers to our notation

Using our notations, we define both encoders as

$$f \triangleq g_{\theta_s} \tag{A4}$$

and

$$f' \triangleq g_{\theta_t} \tag{A5}$$

The distance metric d is the cross entropy. Given the above, the self-distillation loss of DINO can be written as

$$d(f(\boldsymbol{x}), f'(\boldsymbol{x}^+)) := H(f(\boldsymbol{x}), f'(\boldsymbol{x}^+)) \tag{A6}$$

where H is the cross entropy

$$H(p, q) = -\sum_{i \in \mathcal{I}} p(i) \log q(i) \tag{A7}$$

and $\mathcal{I}$ is the support of the distributions p and q, *i.e.* $\mathcal{I} = [I] = \{1, 2, \cdots, I\}$. I refers to the dimensionality of the output distributions. This loss is minimized w.r.t. to the weights of f (no gradients are back-propagated through f').

B. Implementation details

B.1. SimSiam [12]

We use the code from their official GitHub (link). All 3 runs (baseline, baseline + NN, baseline + Adasim) use the same hyperparameters:

- `arch`: resnet50
- `epochs`: 100
- `batch_size`: 512
- `lr`: 0.05
- `momentum`: 0.9
- `weight_decay`: 0.0001
- `dim`: 2048
- `pred_dim`: 512
- `fix_pred_lr`: True

B.2. DINO [8]

We use the code from their official GitHub (link). For the 3 ResNet-50 runs (baseline, baseline + NN, baseline + Adasim) the same hyperparameters specified on the official GitHub are used, except for `local_crops_number` which we set to 0:

- `arch`: resnet50
- `batch_size_total`: 1024
- `clip_grad`: 0.0
- `drop_path_rate`: 0.1
- `epochs`: 100
- `freeze_last_layer`: 1
- `global_crops_scale`: [0.14, 1.0]
- `local_crops_number`: 0

- `lr`: 0.03
- `lr_linear`: 0.03
- `min_lr`: 1e-05
- `momentum_teacher`: 0.996
- `norm_last_layer`: False
- `optimizer`: sgd
- `out_dim`: 65536
- `seed`: 0
- `teacher_temp`: 0.07
- `use_bn_in_head`: False
- `use_fp16`: False
- `warmup_epochs`: 10
- `warmup_teacher_temp`: 0.04
- `warmup_teacher_temp_epochs`: 30
- `weight_decay`: 0.0001
- `weight_decay_end`: 0.0001

For all runs using the ViT-S/16 backbone, the same hyperparameters specified on the official GitHub are used:

- `arch`: vit_small
- `patch_size`: 16
- `batch_size_total`: 1024
- `clip_grad`: 0.0
- `drop_path_rate`: 0.1
- `epochs`: 800
- `freeze_last_layer`: 1
- `global_crops_scale`: [0.4, 1.0]
- `local_crops_number`: 0
- `lr`: 0.0005
- `min_lr`: 1e-05
- `momentum_teacher`: 0.996
- `norm_last_layer`: False
- `optimizer`: adamw
- `out_dim`: 65536
- `seed`: 0
- `teacher_temp`: 0.07
- `use_bn_in_head`: False
- `use_fp16`: True

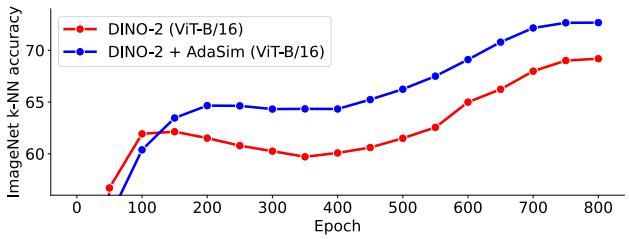

Figure A1: AdaSim benefits from **longer training schedules**.

Table A1: **Runtime analysis of AdaSim per iteration of pretraining.** Run on 4x AMD MI250X GPUs.

method	backbone	batchsize per GPU	time per iter [s]
DINO-2 [8]	ViT-S/16	128	0.256
DINO-2 + AdaSim	ViT-S/16	128	0.270
DINO-2 [8]	ViT-S/16	256	0.480
DINO-2 + AdaSim	ViT-S/16	256	0.488

Table A2: **Transfer to linear segmentation.** A linear layer is trained on top of the frozen spatial features. We report mIoU scores on PVOC12, COCO-Thing, and COCO-Stuff. "-" denotes collapse.

				mIoU scores			
Method	Model	Dataset	Epochs	PVOC12	COCO-Thing	COCO-Stuff	Avg.
DINO-2	ViT-S/16	ImageNet-1k	800	69.0	65.0	**52.4**	62.1
DINO-2 + NN	ViT-S/16	ImageNet-1k	800	-	-	-	-
DINO-2 + AdaSim	ViT-S/16	ImageNet-1k	800	**69.6**	**65.7**	52.0	**62.4** (+ 0.3)
DINO-2	ViT-S/16	COCO	800	49.9	48.1	46.3	48.1
DINO-2 + NN	ViT-S/16	COCO	800	-	-	-	-
DINO-2 + AdaSim	ViT-S/16	COCO	800	**52.9**	**52.2**	**48.1**	**51.1** (+ 3.0)

- `warmup_epochs`: 10
- `warmup_teacher_temp`: 0.04
- `warmup_teacher_temp_epochs`: 30
- `weight_decay`: 0.04
- `weight_decay_end`: 0.4

C. Runtime analysis

We compare the runtime for baseline DINO-2 [8] and DINO-2 + AdaSim. As can be seen in Table A1, AdaSim has a negligible impact on throughput during the pretraining.

D. Scalability

The memory required to store the cache $\mathbf{Z}$ is linear in the size of the dataset. However, its footprint is always much lower than the dataset itself since the cache only stores a representation instead of a full image. When training on very large datasets, workers do not store the whole dataset but only a shard $\mathcal{D}^{(i)}$ with $\mathcal{D} = \{\mathcal{D}^{(1)}, \mathcal{D}^{(2)}, \cdots \mathcal{D}^{(n)}\}$. In such case, the cache $\mathbf{Z}$ can also be split into shards $\mathbf{Z}^{(i)}$ with $\mathbf{Z} = \{\mathbf{Z}^{(1)}, \mathbf{Z}^{(2)}, \cdots \mathbf{Z}^{(n)}\}$ making the algorithm scalable to datasets of arbitrary size.

E. Additional results

We investigate the ability of AdaSim to cope with scene-centric datasets and to produce spatial features aligned with dense downstream tasks. To that end, we train a ViT-S/16 for 800 epochs on COCO [34]. The resulting features are then evaluated via a linear segmentation task on three datasets, namely PVOC12 [21], COCO-Thing, and COCO-Stuff [34]. We rely on the evaluation pipeline of [44] and refer to their work for the implementation details.

Additionally, we examine how the performance of AdaSim evolves throughout training. For that purpose, we report the k-NN accuracy on ImageNet [15] at different epochs. We observe that AdaSim initially performs worse than the baseline (DINO-2 + ViT-B/16), but outperforms it from ≈ 150 epochs on. Indeed, the learned representations must be sufficiently good for bootstrapping to be beneficial. Bootstrapping pairs of nearest neighbors that aren't semantically related hurts the performance and is part of the motivation behind AdaSim. Along the same line, we observe that both larger backbones and longer training schedules contribute to learning better representations.

F. Investigating intriguing neighbors

In Figure 1, we show a visualization of random query images $\boldsymbol{x}_i$ and their corresponding support $\mathcal{S}_{\text{win}}$ ranked by decreasing $p^{\text{win}}(\boldsymbol{x}_j|\boldsymbol{x}_i)$. All 1-NN are the same as the query image ($\boldsymbol{x}_i = \arg\max_{\boldsymbol{x}_j} p^{\text{win}}(\boldsymbol{x}_j|\boldsymbol{x}_i)$), and all other neighbors seem to share semantic content with the query image.

F.1. Nearest neighbor is different from the query

Here, we explicitly search for cases where the nearest neighbor is not the same image as the query to observe border cases of AdaSim. Mathematically, this is the case when $\boldsymbol{x}_i \neq \arg\max_{\boldsymbol{x}_j} p^{\text{win}}(\boldsymbol{x}_j|\boldsymbol{x}_i)$. Such queries are shown in Figure A2 with the corresponding metadata in Table A3. It can be observed that even when the nearest neighbor does not originate from the same image, all nearest neighbors visually share semantic content.

F.2. Nearest neighbor is from a different class as the query

A stronger special case happens when the nearest neighbor is not even from the same class as the query *i.e.* $\texttt{class}(\boldsymbol{x}_i) \neq \texttt{class}(\arg\max_{\boldsymbol{x}_j} p^{\text{win}}(\boldsymbol{x}_j|\boldsymbol{x}_i))$. This is shown in Figure A3 and Table A4. Interestingly, even if the first nearest neighbor is not from the same class, it still looks very similar. This shows evidence of mislabelling. Consider the example of the first row of Figure A3. The query is from class 384 (indri, indris, Indri indri, Indri brevicaudatus) yet the first neighbor is from class 383 (Madagascar cat, ring-tailed lemur, Lemur catta). However, they are very similar and one seems to be a zoomed-in version of the other. Similar conclusions can be drawn for all other query images in Figure A3.

For readability purposes, Table A3 and Table A4 only include the class ids and not the full class names. A mapping from class id to class name can be found below for the subset of classes appearing in Table A3 and Table A4.

- 60: night snake, Hypsiglena torquata
- 66: horned viper, cerastes, sand viper, horned asp, Cerastes cornutus
- 68: sidewinder, horned rattlesnake, Crotalus cerastes
- 80: black grouse
- 82: ruffed grouse, partridge, Bonasa umbellus
- 83: prairie chicken, prairie grouse, prairie fowl
- 138: bustard
- 166: Walker hound, Walker foxhound
- 167: English foxhound
- 206: curly-coated retriever
- 219: cocker spaniel, English cocker spaniel, cocker
- 220: Sussex spaniel
- 221: Irish water spaniel
- 238: Greater Swiss Mountain dog
- 239: Bernese mountain dog
- 240: Appenzeller
- 241: EntleBucher
- 244: Tibetan mastiff
- 337: beaver
- 341: hog, pig, grunter, squealer, Sus scrofa
- 342: wild boar, boar, Sus scrofa
- 343: warthog
- 356: weasel
- 357: mink
- 358: polecat, fitch, foulmart, foumart, Mustela putorius
- 359: black-footed ferret, ferret, Mustela nigripes
- 360: otter
- 365: orangutan, orang, orangutang, Pongo pygmaeus
- 370: guenon, guenon monkey
- 376: proboscis monkey, Nasalis larvatus
- 378: capuchin, ringtail, Cebus capucinus
- 380: titi, titi monkey
- 383: Madagascar cat, ring-tailed lemur, Lemur catta
- 384: indri, indris, Indri indri, Indri brevicaudatus
- 386: African elephant, Loxodonta africana
- 401: accordion, piano accordion, squeeze box
- 420: banjo
- 482: cassette player
- 487: cellular telephone, cellular phone, cellphone, cell, mobile phone
- 546: electric guitar
- 574: golf ball
- 592: hard disc, hard disk, fixed disk
- 605: iPod
- 647: measuring cup
- 745: projector
- 819: stage
- 848: tape player
- 852: tennis ball
- 863: totem pole
- 890: volleyball
- 968: cup

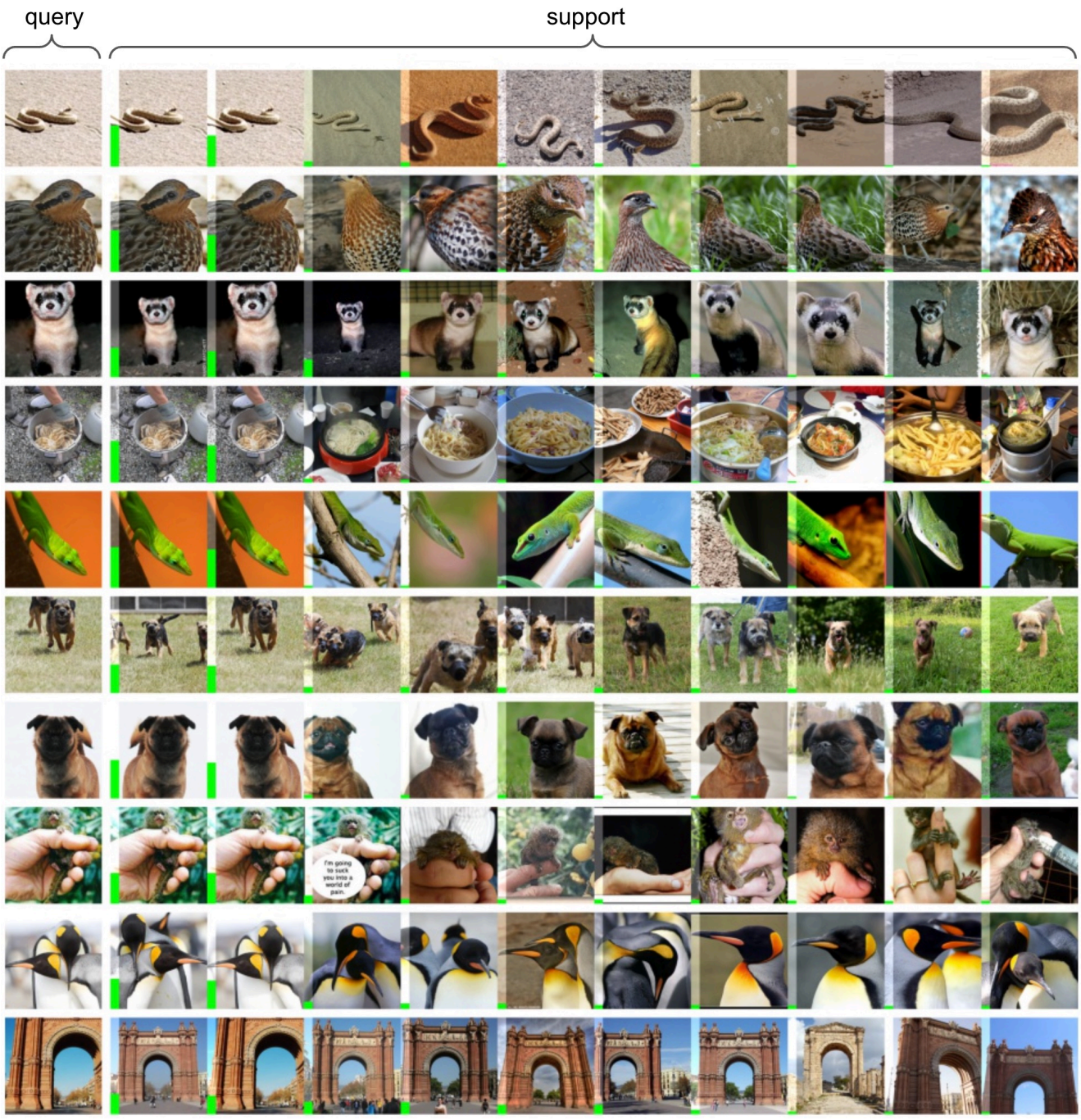

Figure A2: **Query image $\boldsymbol{x}_i$ (left column) and corresponding support $\mathcal{S}_{\text{win}}$ ranked by decreasing $p^{\text{win}}(\boldsymbol{x}_j|\boldsymbol{x}_i)$ (illustrated as a green bar on the bottom left of each image).** The query images are chosen such that the most similar image from the support is not the same as the query ($\boldsymbol{x}_i \neq \arg\max_{\boldsymbol{x}_j} p^{\text{win}}(\boldsymbol{x}_j|\boldsymbol{x}_i)$). Metadata about the images is shown in Table A3.

Table A3: **Metadata corresponding to Figure A2.** Each block corresponds to a row of Figure A2. Within a block, the first row denotes the image id, the second row denotes the class id and the last row denotes the sampling distribution $p^{\text{win}}(\boldsymbol{x}_j|\boldsymbol{x}_i)$.

$\boldsymbol{x}_i$ (query)	1-NN	2-NN	3-NN	4-NN	5-NN	6-NN	7-NN	8-NN	9-NN	10-NN
313037	313303	313037	312150	312728	312960	312198	313277	312158	312230	312903
244	244	244	244	244	244	244	244	244	244	244
	0.41	0.36	0.04	0.04	0.04	0.03	0.02	0.02	0.02	0.02
1105805	1105953	1105805	1105673	1105826	1106178	1106303	1105027	1105739	1105747	1105980
863	863	863	863	863	863	863	863	863	863	863
	0.2	0.16	0.16	0.14	0.13	0.07	0.06	0.04	0.02	0.02
1140581	1139752	1140581	1139966	1139623	1140694	1140389	1139643	1140611	1140717	1139731
890	890	890	890	890	890	890	890	890	890	890
	0.21	0.18	0.14	0.11	0.1	0.09	0.06	0.06	0.03	0.02
213504	214625	213504	214919	213989	214090	213530	214654	214352	214537	214521
166	167	166	167	166	166	166	167	167	167	167
	0.2	0.15	0.13	0.13	0.11	0.11	0.09	0.06	0.02	0.01
482666	483351	482666	482655	482983	482561	482824	483621	482626	482600	482467
376	376	376	376	376	376	376	376	376	376	376
	0.3	0.29	0.1	0.06	0.06	0.05	0.03	0.03	0.03	0.03
439007	437919	439007	439041	440492	440579	496403	496313	179490	440440	439580
342	341	342	342	343	343	386	386	138	343	343
	0.27	0.24	0.13	0.07	0.07	0.05	0.05	0.04	0.04	0.04
304802	307986	304802	305026	308855	308662	305875	305635	307400	305792	307371
238	240	238	238	241	241	239	239	240	239	240
	0.32	0.2	0.11	0.09	0.09	0.08	0.04	0.03	0.03	0.02
86529	88177	86529	88271	88804	78641	88860	88405	86130	89098	86108
66	68	66	68	68	60	68	68	66	68	66
	0.42	0.41	0.03	0.03	0.02	0.02	0.02	0.02	0.01	0.01
738207	1091457	738207	738128	738179	738186	737971	737433	737058	737339	738233
574	852	574	574	574	574	574	574	574	574	574
	0.23	0.21	0.14	0.12	0.08	0.05	0.05	0.05	0.04	0.04
460081	461411	460081	461155	460028	459517	461393	459646	460305	459219	460791
358	359	358	359	358	358	359	358	359	358	359
	0.37	0.35	0.08	0.05	0.04	0.03	0.02	0.02	0.02	0.02

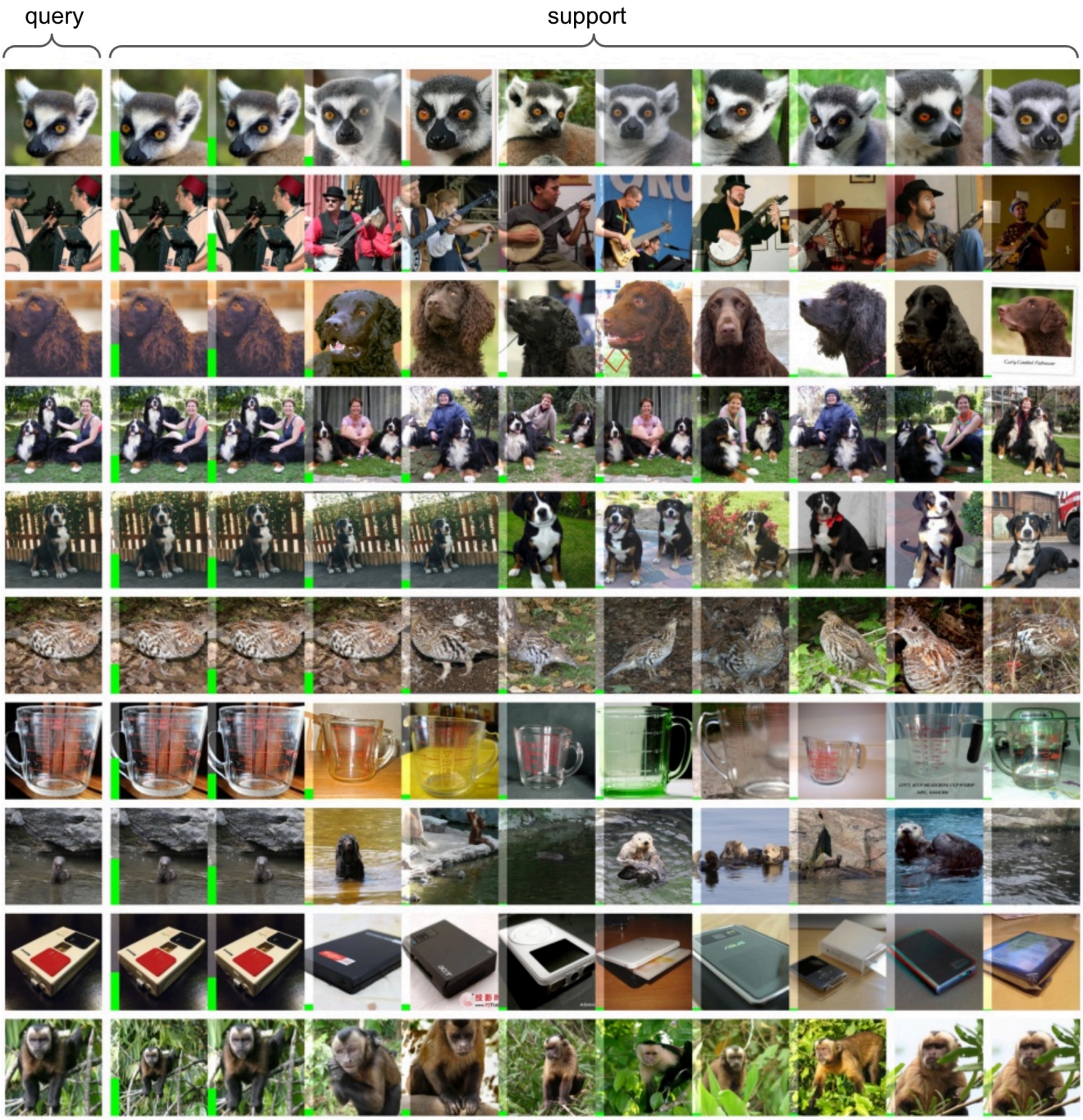

Figure A3: **Query image $\boldsymbol{x}_i$ (left column) and corresponding support $\mathcal{S}_{\text{win}}$ ranked by decreasing $p^{\text{win}}(\boldsymbol{x}_j|\boldsymbol{x}_i)$ (illustrated as a green bar on the bottom left of each image).** The query images are chosen such that the most similar image from the support is not from the same class as the query ($\texttt{class}(\boldsymbol{x}_i) \neq \texttt{class}(\arg\max_{\boldsymbol{x}_j} p^{\text{win}}(\boldsymbol{x}_j|\boldsymbol{x}_i))$). Metadata about the images is shown in Table A4.

Table A4: **Metadata corresponding to Figure A3.** Each block corresponds to a row of Figure A3. Within a block, the first row denotes the image id, the second row denotes the class id and the last row denotes the sampling distribution $p^{\text{win}}(\boldsymbol{x}_j|\boldsymbol{x}_i)$.

$\boldsymbol{x}_i$ (query)	1-NN	2-NN	3-NN	4-NN	5-NN	6-NN	7-NN	8-NN	9-NN	10-NN
493896	492056	493896	492267	493504	492574	492098	491860	493914	493160	492988
384	383	384	383	384	383	383	383	384	384	384
	0.36	0.3	0.1	0.06	0.04	0.04	0.03	0.03	0.03	0.02
514556	540102	514556	539549	539488	539085	1049500	540080	539722	539428	701031
401	420	401	420	420	420	819	420	420	420	546
	0.43	0.4	0.03	0.03	0.02	0.02	0.02	0.02	0.02	0.02
281630	282778	281630	263441	282465	263810	263486	281511	282504	280302	263945
220	221	220	206	221	206	206	220	221	219	206
	0.34	0.29	0.08	0.08	0.06	0.04	0.03	0.03	0.03	0.02
307055	309205	307055	308671	307104	308841	307955	308794	309282	309246	309227
240	241	240	241	240	241	240	241	241	241	241
	0.29	0.23	0.1	0.1	0.07	0.07	0.05	0.04	0.03	0.02
304813	308718	304813	304687	307947	308807	308163	308745	307713	308216	307930
238	241	238	238	240	241	241	241	240	241	240
	0.35	0.32	0.11	0.08	0.03	0.03	0.02	0.02	0.02	0.02
107449	107748	103927	107449	106473	107398	106690	106705	106982	106536	106610
82	83	80	82	82	82	82	82	82	82	82
	0.31	0.26	0.22	0.06	0.05	0.04	0.03	0.02	0.01	0.01
830394	1239824	830394	831098	830660	831378	831425	830417	831372	831192	830232
647	968	647	647	647	647	647	647	647	647	647
	0.43	0.27	0.11	0.05	0.04	0.03	0.02	0.02	0.02	0.02
458411	457058	458411	283078	469050	432009	462780	461777	457794	462065	461770
357	356	357	221	365	337	360	360	357	360	360
	0.47	0.4	0.02	0.02	0.02	0.02	0.01	0.01	0.01	0.01
619775	1086664	619775	760794	954774	777417	1085978	626568	776959	760171	759785
482	848	482	592	745	605	848	487	605	592	592
	0.4	0.34	0.07	0.07	0.03	0.03	0.02	0.02	0.01	0.01
488570	485197	488570	486127	488201	485483	485652	485828	484997	485594	475405
380	378	380	378	380	378	378	378	378	378	370
	0.39	0.28	0.07	0.06	0.05	0.04	0.03	0.03	0.02	0.02