

A Appendix

A.1 Adaptive Token Method Descriptions

A.1.1 Token Reduction

Protect Informative Tokens before Merging (PiToMe) [23] builds on Bipartite Soft Matching (BSM), which partitions tokens into two sets and merges pairs across both according to key-based similarity. Instead of considering *all* tokens for merging, PiToMe proposes an energy score—defined as an aggregate similarity of a token with all others—to distinguish redundant (high-energy) from informative (low-energy) tokens. Only the top $2k$ high-energy tokens are considered for merging via BSM, preserving the most informative tokens.

Agglomerative Token Clustering (ATC) [24] replaces BSM with a bottom-up, hierarchical clustering approach for token merging. Each token starts as its own cluster, and clusters are iteratively merged based on a similarity score defined between clusters. At each step, the two most similar (i.e., redundant) clusters are merged, continuing until a target number of tokens remains.

Decoupled Token Embedding for Merging (DTEM) [25] introduces a trainable module that decouples token merging from representation learning. It projects intermediate token embeddings to extract merging-specific features for estimating pairwise token similarities. Unlike traditional *hard* assignment grouping schemes, DTEM uses a differentiable top- k operator [60] to softly group tokens, followed by BSM for merging. Notably, the soft merging retains *all* tokens during training, with discretization applied only during inference for actual token reduction.

A.1.2 Patch Selection

PatchDrop [27] formulates selection as a Reinforcement Learning problem to train a patch selection policy. Given a down-sampled input, the policy returns Bernoulli distributions over high-resolution patches, from which samples are drawn. The policy is initially trained using the REINFORCE [50] algorithm with a *fixed* pretrained classifier, receiving rewards that encourage both classification accuracy and sparse sampling. A joint finetuning stage then updates the policy and classifier together for improved performance.

Differentiable Patch Selection (DPS) [28] casts patch selection as a ranking problem, where a scoring network assigns relevance scores to high-resolution patches based on a low-resolution input. A differentiable top- k module, based on the perturbed maximum method [54], selects the top k scoring patches for downstream processing by feature and patch-aggregation networks. The entire pipeline is trained end-to-end under supervision.

Iterative Patch Selection (IPS) [29] performs ranking-based patch selection directly on the high-res input through sequential glimpses. A scoring module maintains a buffer of the top k most salient patches, which is updated auto-regressively by evaluating I patches at a time in no-gradient mode. The selected patches are then re-embedded with gradients enabled and passed to a cross-attention transformer-based classifier, enabling end-to-end training. This same cross-attention module is shared with the scoring network to guide saliency prediction.

Learning to Rank Patches (LTRP) [30] trains a patch ranking model via self-supervision. Using a pretrained masked autoencoder, the method randomly masks patches and estimates each *visible* patch’s importance by measuring the change in image reconstruction when the patch is removed—yielding a pseudo-relevance score. A ranking model is then trained to predict the resulting ranks such that, following pretraining, they may be used to retain the top most salient patches for downstream tasks.

A.2 Experimental Details

A.2.1 Pretraining Hyperparameters

We pretrain our ViT-S for 400 epochs and ViT-B for 200 epochs on ImageNet-1K [31] using the AdamW optimizer [46]. We use a batch size of 1024, image size of 518×518 px, weight decay of 0.05, learning rate warmup of 10%, peak learning rate of $2e-04$ with cosine decay to $1e-05$, and RandAugment [47] data augmentation.

A.2.2 ImageNet Hyperparameters

We finetune for 30 epochs on ImageNet-1K [31] using the AdamW optimizer [46]. We use a batch size of 128, image size of 224×224 px, weight decay of 0.02, learning rate warmup of 10%, peak learning rate of $2e-5$ with cosine decay to $1e-5$, and 3-Augment [61] data augmentation.

A.2.3 ADE20K Hyperparameters

We finetune for 160K steps on ADE20K [32] using the AdamW optimizer [46]. We use a batch size of 16, image size of 518×518 px, weight decay of 0.02, learning rate warmup of 10%, peak learning rate of $2e-5$ with cosine decay to $1e-5$, and data augmentation: random crop scale $[0.7, 1.0]$, horizontal flip, random choice {gray scale, solarize, gaussian blur}, and color jitter (0.3). We compute all patch-token representations by interpolating sparse representations with the same interpolation method used during pretraining. We use a simple linear head that maps patch-token representations to pixel-wise class predictions.

A.2.4 Traffic Signs Recognition

Traffic Signs Setup. We use the annotated subset of the Swedish Traffic Signs dataset [33], containing 747 training and 684 test images. All images are resized and cropped to square dimensions of 994×994 px for compatibility with DINOv2 ViT backbones pretrained on square images. IPS and DPS use 980×980 px due to compatibility with their ViT patch extraction methods. We finetune all models for 30 epochs using the AdamW optimizer [46]. We sweep learning rates over $\{2e-5, 5e-5, 8e-5, 1e-4\}$ with a batch size of 16, following [29], and report the best final test accuracy achieved by each method. We apply a weight decay of 0.02, a 10% learning rate warmup, with cosine decay to $1e-5$, and data augmentation including random cropping (scale range $[0.8, 1.0]$) and standard RandAugment transformations [47]. DTEM and PiToMe merge tokens after every layer, ATC merges after layers 4, 7, and 10 (following [52]).

Traffic Signs Results in Tabular Format. We report the full experimental results in Table 3 to complement Figure 4.

Table 3: **Traffic Signs results.** LookWhere remains competitive with SoTA token selection methods, being more efficient than IPS and rivaling DPS for speed, memory, and FLOPs. LookWhere-R uses random masking instead of the selector, but still receives the selector’s low-res global tokens for context. DTEM runs out of memory (OOM).

Method	Top-1 Acc. % \uparrow	Memory (GB) \downarrow		FLOPs (G) \downarrow		Speed (im/s) \uparrow	
		Test	Train	Test	Train	Test	Train
DINOv2 [1]	94.0	3.1	21.3	900	2698	24.1	6.8
PiToMe [23] ($r=0.9$)	66.5	3.1	8.6	440	1300	51.4	17.0
ATC [24] ($r=0.7$)	69.3	2.7	10.7	537	1599	1.3	1.2
DTEM [25]	OOM	OOM	OOM	OOM	OOM	OOM	OOM
DPS ($r=0.1$) [28]	83.6	0.8	1.8	49	142	942.4	259.0
IPS ($r=0.1$) [29]	96.3	0.8	1.8	1819	1912	94.1	79.3
LookWhere-R ($k=504$)	70.6	0.8	1.8	53	149	609.8	222.0
LookWhere ($k=504$)	94.6	0.8	1.8	53	149	609.8	222.0
LookWhere ($k=1008$)	95.2	0.9	2.5	110	320	288.1	99.1

A.2.5 Fine-grained Bird Classification

Fine-grained Bird Classification Setup. We finetune for 30 epochs on the Caltech-UCSD Birds (CUB-200-2011) dataset [40] of 200 bird species in 11,788 images. We sweep learning rates over $\{2e-5, 5e-5, 8e-5, 1e-4\}$ using a batch size of 64. Images are resized to 518×518 px by default (588×588 px for IPS and DPS for 98×98 pixel patches fed to the ViT feature extractor). We use a weight decay of 0.02, a 10% learning rate warmup, and cosine decay to a minimum learning rate of $1e-5$. Data augmentation includes random cropping (scale range $[0.8, 1.0]$) and standard RandAugment transformations [47]. DTEM and PiToMe merge tokens after every layer, ATC merges after layers 4, 7, and 10 (following [52]).

586 **Detailed Bird Results in Tabular Format.** We report the full experimental results in Table 4 to
587 complement Figure 4.

Table 4: **Bird results.** LookWhere achieves the Pareto frontier with the fastest speed, and lowest memory and FLOPs among SoTA token reduction and selection methods while retaining competitive accuracy. LookWhere-R uses random masking instead of the selector, but still receives the selector’s low-res global tokens for context.

Method	Top-1 Acc.	Memory (GB) ↓		FLOPs (G) ↓		Speed (im/s) ↑	
	% ↑	Eval	Train	Eval	Train	Eval	Train
DINOv2 [1]	90.5	0.9	3.1	152	455	209.4	67.4
PiToMe [23] ($r=0.9$)	34.3	0.9	1.9	81	239	391.1	128.4
ATC [24] ($r=0.7$)	34.4	0.9	2.1	95	285	21.2	18.2
DTEM [25] ($r=96$)	90.7	1.0	5.5	84	464	261.8	52.3
DPS ($r=0.1$) [28]	85.4	0.7	1.8	19	57	2479.0	697.7
DPS ($r=0.2$)	88.3	0.7	1.8	33	98	1442.2	432.1
IPS ($r=0.1$) [29]	89.0	0.8	1.8	653	690	261.8	213.8
IPS ($r=0.2$)	90.2	0.8	1.8	639	704	241.8	179.4
LookWhere-R ($k=136$)	78.2	0.8	1.7	16	41	2322.4	865.6
LookWhere ($k=136$)	89.0	0.8	1.7	16	41	2322.4	865.6
LookWhere ($k=273$)	89.8	0.8	1.7	29	19	1333.6	479.0

588 A.3 Detailed Ablation Setup and Results

589 We ablate several design choices, including: (1) teacher attention targets for selector maps, (2)
590 distillation loss weights, (3) selector map training schemes, (4) low-resolution conditioning tokens
591 for the extractor, and (5) the input size and depth of the selector. Unless stated otherwise, all ablations
592 are conducted over 400 epochs of pretraining on ImageNet-1K [31] at a resolution of 518×518
593 pixels using the ViT-S architecture. During pretraining, we sample $k \in [16, 128]$ uniformly at
594 random, and report results at test time for fixed values $k \in \{16, 72, 128\}$ for general classification
595 and segmentation downstream tasks. Unless stated otherwise, ablations use our default settings,
596 which are highlighted in blue.

597 To ablate LookWhere efficiently, we use a simple k NN-based evaluation. For classification, we
598 process all ImageNet-HR [48] images with LookWhere and store the resulting class-token representa-
599 tions $\hat{z}_{\text{high}}^{\text{cls}}$. We then compute top-1 accuracy using a leave-one-out k NN approach; specifically, for
600 each ImageNet-HR image we retrieve the 3 nearest neighbors, excluding the query itself and assign
601 the most frequent class among them as the prediction. For segmentation, we process all ADE20K
602 [32] images with LookWhere and store 10 patch-token representations $\hat{z}_{\text{high}}^{\text{pat}}$ per sample. We then
603 compute top-1 accuracy using the validation set as queries and the training set as keys. For each
604 query, we fetch 20 patch tokens and assign the most common class as the predicted label.

605 A.3.1 Teacher Attention Targets for Selection

606 To distill the teacher’s attention, we extract the unnormalized attention among its patch tokens at *select*
 607 layers and query tokens and then average over layers, queries and heads (Fig. 8). This approximates
 608 where the selected tokens contributed to the teacher representation. In this section, we explore
 609 different combinations of *layers* and *query tokens* used for attention aggregation.

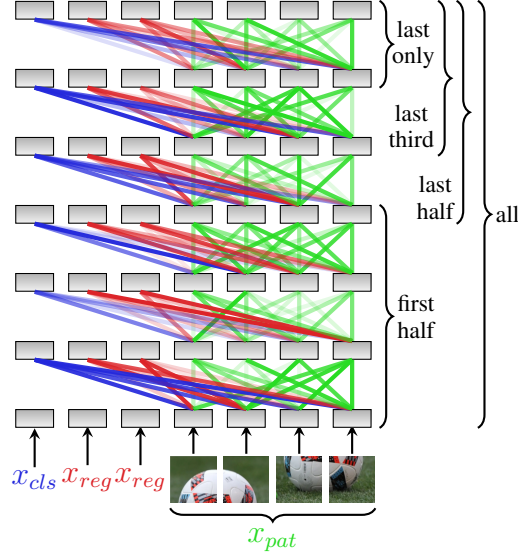


Figure 8: **Attention Aggregation.** We aggregate attention by averaging unnormalized attention scores originating from the **cls**, **reg**, and **patch** queries across different layers (all, first 3, last 3, last 2, and last only). Score maps from each query type are weighted equally when several are used (i.e., we average the maps among tokens of each type first, before averaging across types).

Classification		Layer Aggregation				
Query Aggregation		first half	last half	last third	last only	all layers
	cls	48.5	52.9	54.8	57.6	52.1
	reg	47.8	50.1	50.8	56.5	49.2
	pat	46.9	49.8	50.3	54.1	48.8
	cls+reg	48.5	49.0	53.8	57.2	50.9
	cls+pat	48.9	52.2	54.2	57.7	50.8
	reg+pat	47.5	49.4	51.0	56.3	48.6
	cls+reg+pat	47.9	51.5	52.8	57.3	51.3

Segmentation		Layer Aggregation				
Query Aggregation		first half	last half	last third	last only	all layers
	cls	52.3	51.8	51.6	51.8	52.1
	reg	53.4	53.2	52.7	52.2	53.7
	pat	53.6	54.2	53.6	53.3	54.1
	cls+reg	53.4	52.2	52.5	51.7	53.0
	cls+pat	52.5	52.7	52.6	52.3	52.7
	reg+pat	53.6	53.1	53.6	53.2	53.9
	cls+reg+pat	53.8	53.1	52.1	52.0	53.4

Figure 9: **Teacher Attention for Selection** ($k=16$). We evaluate teacher attention maps as selector maps to choose distillation targets when testing using $k=16$ selected patches.

		Layer Aggregation				
		first half	last half	last third	last only	all layers
Query Aggregation	cls	65.0	68.6	69.2	70.3	67.8
	reg	64.1	66.2	66.8	70.2	65.5
	pat	62.4	69.0	68.9	70.4	67.5
	cls+reg	65.4	66.4	68.8	70.4	67.1
	cls+pat	65.0	68.9	69.6	70.9	68.3
	reg+pat	64.1	67.5	68.4	70.6	66.6
	cls+reg+pat	64.9	68.0	69.2	70.6	67.6

		Layer Aggregation				
		first half	last half	last third	last only	all layers
Query Aggregation	cls	61.4	60.8	60.4	59.4	60.8
	reg	61.1	61.5	61.6	60.5	62.1
	pat	62.1	64.2	63.7	62.7	64.2
	cls+reg	61.2	61.0	61.0	60.1	62.2
	cls+pat	62.0	62.0	61.0	60.9	62.0
	reg+pat	61.7	62.4	62.7	61.1	62.6
	cls+reg+pat	61.5	62.2	61.4	60.5	62.4

Figure 10: **Teacher Attention for Selection ($k=72$)**. We evaluate teacher attention maps as selector maps to choose distillation targets when testing using $k=72$ selected patches.

		Layer Aggregation				
		first half	last half	last third	last only	all layers
Query Aggregation	cls	69.3	72.1	72.4	73.4	71.9
	reg	69.1	70.2	71.3	73.4	69.6
	pat	67.0	71.9	72.0	73.1	71.6
	cls+reg	69.3	70.4	72.7	73.7	71.2
	cls+pat	69.6	72.6	73.0	73.7	72.0
	reg+pat	68.9	71.3	72.1	74.3	70.9
	cls+reg+pat	69.2	72.2	73.0	73.4	71.6

		Layer Aggregation				
		first half	last half	last third	last only	all layers
Query Aggregation	cls	64.3	63.8	63.4	63.4	64.5
	reg	64.5	64.9	65.0	64.1	65.2
	pat	65.5	67.4	67.3	66.2	67.5
	cls+reg	64.5	64.7	64.8	63.2	65.2
	cls+pat	65.4	65.7	64.9	63.7	65.7
	reg+pat	65.1	66.0	66.1	64.6	66.1
	cls+reg+pat	64.9	65.8	65.2	63.8	65.8

Figure 11: **Teacher Attention for Selection ($k=128$)**. We evaluate teacher attention maps as selector maps to choose distillation targets when testing using $k=128$ selected patches.

610 A.3.2 Classification Accuracy vs. Distillation Losses

611 We ablate distillation weights and feature-interpolation parameters with 100 epochs of pretraining.
612 During pretraining (and when using ADE20K downstream), we interpolate the sparse/visible high-res
613 patch-token representations to compute a full 2D grid. We compute the predicted dense patch-token
614 representations as the weighted sum of the N nearest neighbors in 2D space, among the sparse/visible
615 patch tokens. The weights in the weighted sum are the Euclidean distances raised to the pow^{th} power,
616 then normalized s.t. all weights sum to 1. We experiment with $N \in \{5, 16\}$ and $\text{pow} \in \{1, 2\}$, and
617 summarize our results in Tables 5 through 8.

Table 5: **Distillation Losses** ($\text{pow}=1, 5$ neighbors). We evaluate different combinations of distillation loss weights for pretraining the selector-extractor ($\lambda_{cls}, \lambda_{pat}$) and selector map (λ_{map}). We consider spatial patch interpolation using Euclidean distance, with 5 selected neighbours.

λ_{cls}	λ_{pat}	λ_{map}	\mathcal{L}_{map}	cls. (by k)			seg. (by k)		
				16	72	128	16	72	128
1	1	1	KL	47.7	61.7	66.4	51.6	59.6	62.6
1	1	0.1	KL	46.8	61.5	66.7	51.0	59.5	62.3
1	0.1	1	KL	48.1	62.5	67.4	48.2	57.3	60.8
1	0.1	0.1	KL	48.9	63.0	67.4	47.8	56.7	60.7
0.1	1	1	KL	44.9	60.3	66.1	51.6	59.8	63.4
0.1	1	0.1	KL	45.0	60.2	65.3	51.3	59.7	62.9
0.1	0.1	1	KL	46.1	62.2	66.9	51.0	59.5	63.1
0.1	0.1	0.1	KL	48.7	62.0	66.7	50.8	59.8	62.5
1	1	1	MSE	50.2	61.8	66.2	51.2	58.5	62.0

Table 6: **Distillation Losses** ($\text{pow}=1, 16$ neighbors). We evaluate different combinations of distillation loss weights for pretraining the selector-extractor ($\lambda_{cls}, \lambda_{pat}$) and selector map (λ_{map}). We consider spatial patch interpolation using Euclidean distance, with 16 selected neighbours.

λ_{cls}	λ_{pat}	λ_{map}	\mathcal{L}_{map}	cls. (by k)			seg. (by k)		
				16	72	128	16	72	128
1	1	1	KL	48.9	62.2	66.8	49.4	58.1	61.5
1	1	0.1	KL	48.1	61.9	66.9	48.9	57.4	61.0
1	0.1	1	KL	47.4	62.4	67.0	47.4	56.7	60.2
1	0.1	0.1	KL	48.3	62.2	66.2	47.5	55.9	60.2
0.1	1	1	KL	44.6	60.4	65.5	49.8	58.8	62.0
0.1	1	0.1	KL	45.8	60.0	66.0	49.2	58.5	61.6
0.1	0.1	1	KL	46.5	62.6	67.0	49.0	58.2	62.1
0.1	0.1	0.1	KL	47.8	62.5	67.1	49.6	58.4	61.6

Table 7: **Distillation Losses** ($\text{pow}=2, 5$ neighbors). We evaluate different combinations of distillation loss weights for pretraining the selector-extractor ($\lambda_{cls}, \lambda_{pat}$) and selector map (λ_{map}). We consider spatial patch interpolation using squared Euclidean distance, with 5 selected neighbours.

λ_{cls}	λ_{pat}	λ_{map}	\mathcal{L}_{map}	cls. (by k)			seg. (by k)		
				16	72	128	16	72	128
1	1	1	KL	46.7	61.9	67.2	50.7	59.0	62.8
1	1	0.1	KL	47.0	61.5	65.8	50.5	59.6	62.5
1	0.1	1	KL	49.3	62.6	67.0	48.3	56.7	60.6
1	0.1	0.1	KL	48.0	62.6	67.2	48.3	56.7	60.6
0.1	1	1	KL	44.8	60.6	66.0	48.0	56.5	48.0
0.1	1	0.1	KL	43.9	60.3	65.1	51.3	59.8	63.2
0.1	0.1	1	KL	45.7	62.3	67.0	50.6	59.0	62.5
0.1	0.1	0.1	KL	47.6	62.3	66.3	50.7	59.1	62.6

Table 8: **Distillation Losses** (pow=2, 16 neighbors). We evaluate different combinations of distillation loss weights for pretraining the selector-extractor (λ_{cls} , λ_{pat}) and selector map (λ_{map}). We consider spatial patch interpolation using squared Euclidean distance, with 16 selected neighbours.

λ_{cls}	λ_{pat}	λ_{map}	\mathcal{L}_{map}	cls. (by k)			seg. (by k)		
				16	72	128	16	72	128
1	1	1	KL	48.3	62.1	66.9	50.2	58.2	61.7
1	1	0.1	KL	47.6	61.5	66.4	49.8	57.5	61.6
1	0.1	1	KL	48.7	62.6	67.5	47.7	56.7	60.1
1	0.1	0.1	KL	49.1	62.6	66.9	47.6	6.1	59.8
0.1	1	1	KL	44.1	61.0	65.3	50.4	59.5	62.6
0.1	1	0.1	KL	44.2	60.8	65.7	50.4	58.8	61.7
0.1	0.1	1	KL	45.9	62.3	67.2	49.8	58.4	62.2
0.1	0.1	0.1	KL	47.8	62.5	67.1	50.3	58.7	62.2

A.3.3 Selector Map Training

In addition to distilling the teacher’s attention map, we explore differentiable learning of the selector map to better leverage the extractor’s signal for optimal patch selection. To enable differentiable selection, we experiment with (1) Gumbel Top-K [49] and (2) REINFORCE [50].

In both approaches, the selector map is treated as logits defining a softmax probability distribution over patches; we sample the map using the Gumbel Top-K trick. We make sampling differentiable through the straight-through estimator (i.e., differentiable Gumbel Top-K) and sweep over learning rates and sampling temperatures. With REINFORCE, we model pretraining as a one-step (contextual) bandit problem: low-resolution images define the state, actions are binary masks selecting k patches (as in PatchDrop [27]), and the reward is derived from the loss. Specifically, we compute an advantage as the difference in distillation loss between a greedy top- k policy and the sampled action. We backpropagate through the extractor for both samples. We explore various learning rates, weightings for distillation and policy gradient losses, and the use of entropy bonuses.

We observe that training fails to converge without a map distillation loss (which effectively acts analogous to a KL regularizing term with respect to the teacher’s “policy”). Table 9 reports the best results across all configurations, consistently showing that teacher distillation alone performs best. Gumbel Top-K pretrains for 400 epochs, while REINFORCE pretrains for 300 epochs (to roughly balance compute across all methods since REINFORCE backpropagates through both the policy sample and greedy baseline).

Table 9: **Selector Map Training Schemes**. We consider different training schemes for the selector map *in addition to distillation*. Stop grad refers to no additional scheme, while Gumbel Top-K and REINFORCE leverage sampling to estimate a gradient from the extractor’s signal. We find that distillation, alone, is optimal.

case	cls. (by k)			seg. (by k)		
	16	72	128	16	72	128
none (stop grad)	50.9	63.0	66.5	51.1	60.2	62.9
Gumbel Top-K [49]	44.5	61.4	66.2	49.1	58.9	62.5
REINFORCE [50]	47.9	61.8	66.0	50.7	59.6	62.8

A.3.4 Extractor Conditioning on Low-Resolution Global Tokens

To provide the extractor with additional *global* image context, we experiment with initializing its class token x_{cls} and/or register tokens x_{reg} by the selector’s final representations for each: z_{low}^{cls} and z_{low}^{reg} , respectively. Table 10 summarizes our results; we find that the addition of global tokens generally helps, although performance interestingly degrades for classification when selecting $k=128$ patches specifically.

Table 10: **Extractor Conditioning.** We experiment with feeding the extractor different combinations of global low-resolution tokens resulting from the selector’s processing. We find that giving *both* the cls token and all register generally helps performance.

case	cls. (by k)			seg. (by k)		
	16	72	128	16	72	128
none	28.2	60.7	68.4	39.7	56.9	61.1
cls-only	48.8	62.2	67.2	49.8	58.7	61.8
reg-only	49.4	62.0	67.3	51.0	59.8	62.4
cls+reg	50.9	63.0	66.5	51.1	60.2	62.9

643 A.3.5 Extractor Input Size and Depth

644 We experiment with varying the extractor’s size by trading off network depth against input resolution,
 645 while keeping computational cost relatively low. Our results are summarized in Table 11.

Table 11: **Extractor Sizes.** Both shallow networks with large inputs and deeper networks with smaller inputs result in suboptimal performance. The best results are achieved by balancing depth and input resolution.

depth	res.	cls. (by k)			seg. (by k)		
		16	72	128	16	72	128
1	252^2	43.7	66.2	70.8	49.0	60.5	64.4
3	252^2	65.8	72.1	73.7	56.1	64.7	67.2
3	154^2	58.7	68.8	71.2	53.9	62.7	65.1
6	154^2	65.0	69.7	72.1	54.4	63.0	65.4
6	98^2	50.9	63.0	66.5	51.1	60.2	62.9
12	98^2	65.8	69.6	70.9	53.0	61.0	63.3
12	42^2	25.5	47.3	57.1	43.8	54.6	58.1

646 A.4 Selector Maps

647 We find that our selector generalizes to diverse downstream images and tasks. In particular, we
 648 visualize general examples for recognition of Birds (Fig. 12) and traffic signs (Fig. 13).

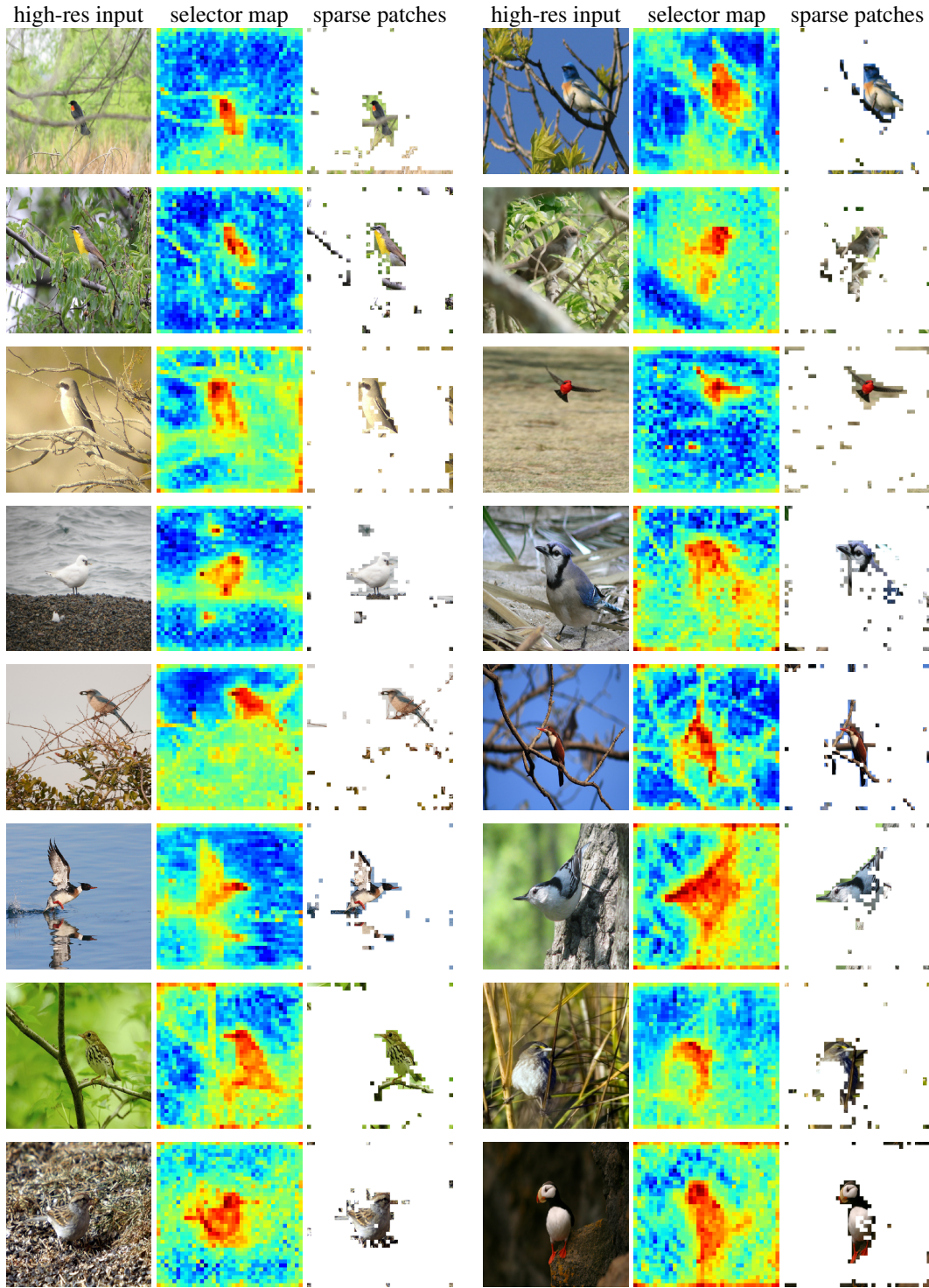


Figure 12: **Adaptive Computation for Recognizing Birds.** We visualize the selector’s prediction of *where* to compute and the extractor’s sparse input for *what* to see. We do *not* finetune the selector; each pair shows different generalization scenarios, specifically for bird recognition on CUB [40].

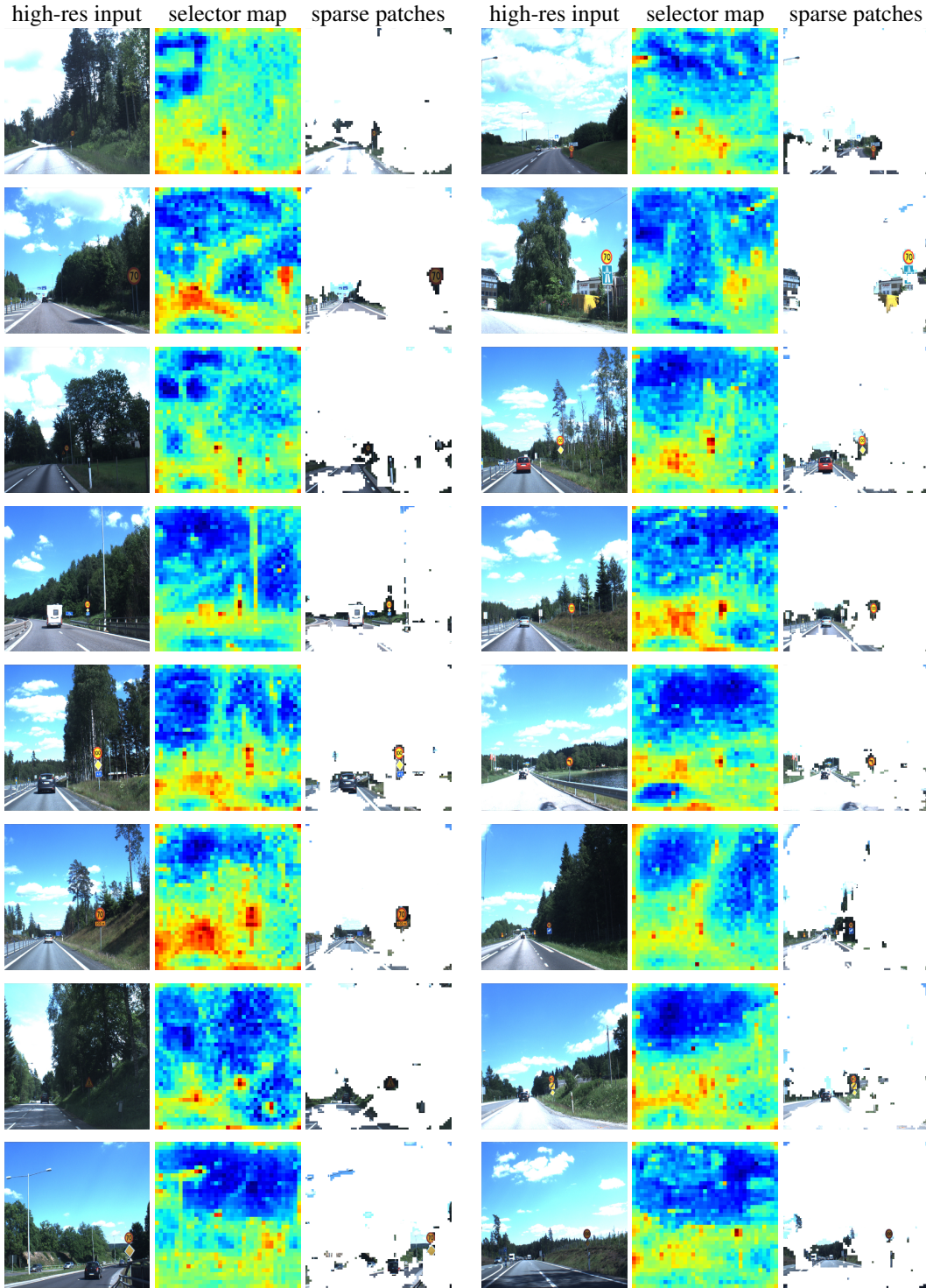


Figure 13: **Adaptive Computation for Traffic Sign Recognition.** We visualize the selector’s prediction of *where* to compute and the extractor’s sparse input for *what* to see. We do *not* finetune the selector; each pair shows different generalization scenarios, specifically for recognizing traffic signs on the Swedish Traffic Signs dataset [33].

651 **A.5.1 Impact Statement**

652 This work presents new designs and empirical results for deep network architectures for more
653 computationally efficient modeling applied to visual recognition. This general topic does not have
654 more specific societal consequences aside from those inherited, good or bad, from the adoption of
655 machine learning.