Appendix for *Cosine Similarity is Almost All You Need (for Prototypical-Part Models)*

8. Detailed Experimental Setup

8.1. Bayesian Hyperparameter Optimization

Hyperparameter selection is not well documented in the literature even though these models introduce a number of key hyperparameters that control the balance between optimizing performance and interpretability. Because hyperparameter tuning requires many expensive model fits, we leverage Bayesian hyperparameter tuning. Bayesian hyperparameter estimates a posterior distribution over hyperparameter configurations with a cheap-to-evaluate surrogate function. Using this estimate, the optimization regime finds a set of hyperparameters over which to perform a complete training run that will resolve uncertainty while having a high likelihood of success. For a more complete review of Bayesian hyperparameter tuning, see [9]. In practice, we use the optimization regime implemented by Weights and Biases Sweeps [8] under Academic Research Licensing. The specific hyperparameters we optimize are provided in section 8.2.

8.2. Hyperparameters

All training runs were executed on single NVIDIA RTX A5000 GPU. Below, we describe the specific hyperparameters tuned through our Bayesian hyperparameter tuning. In this section, we provide an overview of how the hyperparameters were specified for our optimization. In the following sections, we provide hyperparameter tables and model configurations for each model we trained. Unless otherwise noted, we use the configuration specified by the original publication for each model.

All models except ProtoTree use what we call the ProtoPNet training schedule, which has the following structure:

- **Phase 1: Warm-up Training** Train only the add-on and prototype layers for e_w epochs
- **Phase 2: Joint Training** Train all layers except the prediction head for e_j epochs
- **Phase 3: Prototype Projection** Project prototypes onto the nearest sample patches
- **Phase 4: Iterative Refinement**
  - **4.1.** Train only the last layer of the backbone for e_l epochs
  - **4.2.** Repeat Phase 2 (Joint Training)
  - **4.3.** Repeat Phase 3 (Prototype Projection)
  - **4.4.** Repeat steps 4.1-4.3 until convergence
- **Phase 5: Fine-tuning** Train only the last layer of the backbone for e_l epochs

Deformable ProtoPNet has an additional warming phase, **Warm Offset**, immediately after the first Warm Up phase. ST-ProtoPNet alternates between the Support and Trivial branches on each batch.

ProtoTree has a simpler training schedule:

- **Phase 1: Warm-up Training** Train only the add-on and prototype layers for e_w epochs
- **Phase 2: Joint Training** Train all layers except the prediction head for e_j epochs
- **Phase 3: Prototype Projection** Project prototypes onto the nearest sample patches one time

The following are definitions of hyperparameters that are used in optimizing more than one model:

1. `joint_steps_per_phase_at_lr1`: In the ProtoPNet training schedule (used in all models but ProtoTree), how many times to step the learning rate per joint training step after last-only optimization. See: Learning Rate Scaling.
2. `lr_step_per_joint_phase_2exp`: Number of learning rate steps per joint phase, expressed as a binary exponent. For instance, 1 means 2 steps per joint training phase.
3. `lr_step_gamma` Adam or AdamW optimizer γ parameter.
4. `num_warm_epochs_at_lr1`: In each training schedule, the number of epochs to train only the add-on and prototype layers before progressing to the first joint training epoch. See: Learning Rate Scaling.
5. `num_addon_layers`: The number of additional layers to add between the last layer of the backbone and the prototype layer.
6. `num_prototypes_per_class`: The number of prototypes created for each class during model initialization.
7. `latent_dim_multiplier_exp`: Binary logarithm of the number of dimensions to reduce the channel sizes from the backbone through the add-on layers. For instance, -1 means half the channels.
8. `post_project_phases`: For models aside from ProtoTree, the maximum number of cycles of `project-last-only-joint` to run before stopping training.
9. `pre_project_phase_len`: For models aside from ProtoTree, the number of joint and warm epochs to run before the first `project` phase.
10. `last_only_steps_per_joint_step`: For models aside from ProtoTree, the number of epochs to run last-only optimization after each project before progressing the next joint phase.

The following are coefficients for loss terms. Which terms are used depends on the model:

1. `l1_coef`
2. `orthogonality_loss_coef`
3. `cluster_coef`
4. `separation_coef`
5. `closeness_loss_coef`
6. `descrimination_loss_coef`
7. `grassmannian_loss_coef`

Learning Rate Scaling Learning rates for all models are defaulted to $1e-4$ for the backbone and prediction head,

Table 3. Hyperparameter Ranges for Optimized ProtoPNet.

Name	Distribution	Parameters	Quant	$p5$	$p95$
cluster_coef	Normal	$\mu = -0.8$ $\sigma = 0.5$	—	-1.6224	0.0224
joint_epochs _per_phase	Enumerated	10	—	—	—
joint_lr_step_size	Integer Uniform	[2, 10]	—	2.0	10.0
l1_coef	Log-uniform	[1e-05, 0.001]	—	6e-05	0.00095
last_only_epochs _per_phase	Enumerated	20	—	—	—
latent_dim _multiplier_exp	Integer Uniform	[-4, 1]	—	-4.0	1.0
lr_multiplier	Normal	$\mu = 1.0$ $\sigma = 0.4$	—	0.3421	1.6579
lr_step_gamma	Enumerated	0.1	—	—	—
num_addon_layers	Integer Uniform	[0, 2]	—	0.0	2.0
num_prototypes _per_class	Integer Uniform	[1, 16]	—	1.0	16.0
phase_multiplier	Enumerated	1	—	—	—
post_project_phases	Enumerated	10	—	—	—
pre_project _phase_len	Integer Uniform	[3, 15]	—	3.0	15.0
separation_coef	Normal	$\mu = 0.08$ $\sigma = 0.1$	—	-0.0845	0.2445

and 0.003 for prototypes and add-on layers. Changes to the learning rate are specified in terms of a multiplier. Then, the length of the training schedule is adjusted inversely-cubically to the learning rate. This allows us to express dependency between the learning rate and length of the training in number of epochs so that we do not run extremely long training schedules with very high learning rates.

The length adjustment applies to a specific phase of training (i.e., `warm`, `joint`, `last-only`). The effective length function E_{phase} in epochs is defined as follows:

$$E_{phase}(l_{phase}, m) = \max\left(1, \left\lfloor \frac{l_{phase}}{\sqrt[3]{\text{m}}} \right\rfloor\right)$$

where l_{phase} is the phase length in the model's original training schedule (i.e., 10 epochs for ProtoPNet `joint` phases), and m is the learning rate multiplier.

8.2.1. ProtoPNet (Optimized ProtoPNet Experiments)

The hyperparameters between the accuracy-only and accuracy-prototype score optimization are the same, with difference being the optimization objective. The parameters are in Supplementary Table 3. This is the only setting where we optimize over the number of add-on layers. Analysis of performance in this setting suggested that cosine similarity performed equally well with and without add-on layers, whereas L2 needed them. In the optimizations for other models, we always included add-on layers in fairness to L2.

8.2.2. ProtoPNet - Cosine Ablation

There are no additional ProtoPNet-specific hyperparameters. Hyperparmeters ranges are in Supplementary Table 4.

8.2.3. Deformable ProtoPNet - Cosine Ablation

There are no additional Deformable ProtoPNet-specific hyperparameters outside of its loss term coefficients. Hyperparmeters ranges are in Supplementary Table 5.

8.2.4. TesNet - Cosine Ablation

There are no additional TesNet-specific hyperparameters outside of its loss term coefficients. Hyperparmeters ranges are in Supplementary Table 6.

8.2.5. ST-ProtoPNet - Cosine Ablation

We employ the ST-ProtoPNet shared add-on layer configuration so that the resulting model has a single embedding space. We follow the orthogonality loss configuration of ST-ProtoPNet by setting the loss norm `orth_p_norm` to 1. Hyperparmeters ranges are in Supplementary Table 7.

8.2.6. ProtoTree - Cosine Ablation

Because of ProtoTree's training schedule – namely, that project only happens once at the end – we introduced checkpoints for `Successive Halfing`. they were:
1. After $\frac{1}{3}$ of the warm epochs.
2. After $\frac{2}{3}$ of the warm epochs.

Table 4. Hyperparameter Ranges for ProtoPNet - Cosine Ablation.

Name	Distribution	Parameters	Quant	$p5$	$p95$
cluster_coef	Normal	$\mu = -1.0$ $\sigma = 0.4$	0.2	-1.6579	-0.3421
joint_steps_per _phase_at_lr1	Log-normal	$\mu = 2.3$ $\sigma = 0.3$	3.0	6.09	16.34
l1_coef	Log-uniform	[5e-05, 0.001]	5e-05	9.8e-05	0.000953
last_only_steps _per_joint_step	Uniform	[0.25, 1]	0.25	0.2875	0.9625
latent_dim _multiplier_2exp	Uniform	[-3, 0]	1.0	-2.85	-0.15
latent_dim _multiplier_exp	Enumerated	-2	—	—	—
lr_multiplier	Log-normal	$\mu = -0.6$ $\sigma = 1$	0.05	0.1059	2.843
lr_step_gamma	Enumerated	0.1	—	—	—
lr_step_per _joint_phase_2exp	Uniform	[-2, 2]	1.0	-1.8	1.8
num_addon_layers	Enumerated	2	—	—	—
num_warm_epochs _at_lr1	Log-normal	$\mu = 1.6$ $\sigma = 0.4$	2.0	2.57	9.56
post_project_phases	Enumerated	32	—	—	—
separation_coef	Normal	$\mu = 0.1$ $\sigma = 0.04$	0.02	0.0342	0.1658

Table 5. Hyperparameter Ranges for Deformable ProtoPNet - Cosine Ablation.

Name	Distribution	Parameters	Quant	$p5$	$p95$
cluster_coef	Normal	$\mu = -1.0$ $\sigma = 0.4$	0.2	-1.6579	-0.3421
joint_steps_per _phase_at_lr1	Log-normal	$\mu = 2.3$ $\sigma = 0.3$	3.0	6.09	16.34
l1_coef	Log-uniform	[5e-05, 0.001]	5e-05	9.8e-05	0.000953
last_only_steps _per_joint_step	Uniform	[0.25, 1]	0.25	0.2875	0.9625
lr_multiplier	Log-normal	$\mu = -0.6$ $\sigma = 1$	0.05	0.1059	2.843
lr_step_gamma	Enumerated	0.1	—	—	—
lr_step_per _joint_phase_2exp	Uniform	[-2, 2]	1.0	-1.8	1.8
num_addon_layers	Enumerated	2	—	—	—
num_warm_epochs _at_lr1	Log-normal	$\mu = 1.6$ $\sigma = 0.4$	2.0	2.57	9.56
orthogonality_loss _coef	Log-normal	$\mu = -5.2$ $\sigma = 1.4$	0.0005	0.000552	0.055177
post_project_phases	Enumerated	32	—	—	—
separation_coef	Normal	$\mu = 0.1$ $\sigma = 0.04$	0.02	0.0342	0.1658

3. After $\frac{1}{6}$ of the joint epochs.
4. After $\frac{1}{3}$ of the joint epochs.
5. After $\frac{1}{2}$ of the joint epochs.

As in [39], we configured ProtoTree with gradient-free

Table 6. Hyperparameter Ranges for TesNet - Cosine Ablation.

Name	Distribution	Parameters	Quant	$p5$	$p95$
cluster_coef	Normal	$\mu = -1.0$ $\sigma = 0.4$	0.2	-1.6579	-0.3421
grassmannian_orthogonality _loss_coef	Normal	$\mu = -1e - 07$ $\sigma = 2e - 08$	2e-08	-0.0	-0.0
joint_steps_per _phase_at_lr1	Log-normal	$\mu = 2.3$ $\sigma = 0.3$	3.0	6.09	16.34
l1_coef	Log-uniform	[5e-05, 0.001]	5e-05	9.8e-05	0.000953
last_only_steps _per_joint_step	Uniform	[0.25, 1]	0.25	0.2875	0.9625
lr_multiplier	Log-normal	$\mu = -0.6$ $\sigma = 1$	0.05	0.1059	2.843
lr_step_gamma	Enumerated	0.1	—	—	—
lr_step_per _joint_phase_2exp	Uniform	[-2, 2]	1.0	-1.8	1.8
num_addon_layers	Enumerated	2	—	—	—
num_warm_epochs _at_lr1	Log-normal	$\mu = 1.6$ $\sigma = 0.4$	2.0	2.57	9.56
orthogonality_loss _coef	Normal	$\mu = 0.0001$ $\sigma = 2e - 05$	2e-05	6.7e-05	0.000133
post_project_phases	Enumerated	32	—	—	—
proto_channels	Uniform	[64, 128]	64.0	67.2	124.8
separation_coef	Normal	$\mu = 0.1$ $\sigma = 0.04$	0.02	0.0342	0.1658

leaf weight updates, do not use log probabilities, train the final layer of the backbone during warm epochs, and set the AdamW optimizer parameters ϵ (`adamw_eps`) and weight decay (`adamw_weight_decay`) to $1e-7$ and 0, respectively. In addition, we fix the tree depth `depth` to 9. We split the learning rate multipliers into separate multipliers for the backbone and the add-on and prototype layers, `backbone_lr_multiplier` and `non_backbone_lr_multiplier`, respectively.

We generalized ProtoTree's fixed training schedule. It uses a milestone-based learning rate decay. We parameterized both the start of and number of milestones, `joint_epochs_before_lr_milestones` and `num_lr_milestones`, respectively.

Hyperparmeters ranges are in Supplementary Table 8.

8.2.7. Augmentation Details

Except for when training ProtoTree on CUB200, we perform online augmentation on our training data to prevent overfitting. In particular, when loading each training image, we performed:

- A random rotation between -15 and 15 degrees
- A random distortion with scale 0.2
- A random shear of up to 10 pixels
- A 50% chance of a horizontal flip

Because of this, the effective length of our training epochs is one-tenth of the original ProtoPNet. There is a trade-off here. The extended training schedule likely played a role in allowing ProtoPNet with ℓ_2 to reach reasonable performance. However, the cost of training runs that take a day or longer is not insignificant for a model with many hyperparameters. This is one of the core reasons the experiments we ran needed to have a fixed-GPU cost. We allowed the hyperparameters to adjust the schedule length to ensure that the shorter schedule did not preclude ℓ_2 from being competitive.

When training ProtoTree on CUB-200, we use the offline augmentation scheme from [39], which is described in the supplementary material of that paper.

Table 7. Hyperparameter Ranges for ST-ProtoPNet - Cosine Ablation.

Name	Distribution	Parameters	Quant	$p5$	$p95$
closeness_loss_coef	Normal	$\mu = 1.0$ $\sigma = 0.4$	0.2	0.3421	1.6579
cluster_coef	Normal	$\mu = -1.0$ $\sigma = 0.4$	0.2	-1.6579	-0.3421
discrimination_loss _coef	Normal	$\mu = 1.0$ $\sigma = 0.4$	0.2	0.3421	1.6579
joint_steps_per _phase_at_lr1	Log-normal	$\mu = 2.3$ $\sigma = 0.3$	3.0	6.09	16.34
l1_coef	Log-uniform	[5e-05, 0.001]	5e-05	9.8e-05	0.000953
last_only_steps _per_joint_step	Uniform	[0.25, 1]	0.25	0.2875	0.9625
lr_multiplier	Log-normal	$\mu = -0.6$ $\sigma = 1$	0.05	0.1059	2.843
lr_step_gamma	Enumerated	0.1	—	—	—
lr_step_per _joint_phase_2exp	Uniform	[-2, 2]	1.0	-1.8	1.8
num_addon_layers	Enumerated	1	—	—	—
num_warm_epochs _at_lr1	Log-normal	$\mu = 1.6$ $\sigma = 0.4$	2.0	2.57	9.56
ortho_p_norm	Enumerated	1	—	—	—
orthogonality_loss _coef	Normal	$\mu = 0.001$ $\sigma = 0.4$	0.2	-0.6569	0.6589
post_project_phases	Enumerated	32	—	—	—
support_separation _coef	Normal	$\mu = 0.5$ $\sigma = 0.04$	0.02	0.4342	0.5658
trivial_separation _coef	Normal	$\mu = 0.1$ $\sigma = 0.04$	0.02	0.0342	0.1658

Table 8. Hyperparameter Ranges for ProtoTree - Cosine Ablation.

Name	Distribution	Parameters	Quant	$p5$	$p95$
adamw_eps	Enumerated	1e-07	—	—	—
adamw_weight_decay	Enumerated	0.0	—	—	—
backbone_lr _multiplier	Log-normal	$\mu = 0$ $\sigma = 1.2$	0.2	0.1389	7.1982
depth	Enumerated	9	—	—	—
joint_epochs_before _lr_milestones	Enumerated	30	—	—	—
joint_phase_len _at_lr1	Normal	$\mu = 70$ $\sigma = 10$	5.0	53.55	86.45
log_probabilities	Enumerated	False	—	—	—
lr_step_gamma	Normal	$\mu = 0.5$ $\sigma = 0.2$	0.1	0.171	0.829
lr_weight_decay	Log-normal	$\mu = -10$ $\sigma = 1$	5e-05	9e-06	0.000235
non_backbone _lr_multiplier	Log-normal	$\mu = 0$ $\sigma = 1.2$	0.2	0.1389	7.1982
num_lr_milestones	Enumerated	5	—	—	—
proto_channels	Enumerated	256	—	—	—
warm_up_phase _len_at_lr1	Normal	$\mu = 30$ $\sigma = 5$	5.0	21.78	38.22

9. Deformable Prototype Implementation

In Deformable ProtoPNet [17], each prototype and each latent feature vector is restricted to be of a fixed norm. In particular, if prototypes consist of $H_P W_P$ parts, each part and each latent vector is set to have a 2-norm of $\frac{1}{\sqrt{H_P W_P}}$. This guarantees that the cosine similarity between a prototype $\mathbf{p}_j \in \mathbb{R}^{d \times H_P \times W_P}$ and a group of latent feature vectors $\{\mathbf{z}_k \in \mathbb{R}^d\}_{k=1}^{H_P W_P}$ can be computed directly as (where θ denotes the angle between two vectors):

$$\cos\left(\theta\left(\begin{bmatrix}\mathbf{p}_{j,:,1,1}\\ \mathbf{p}_{j,:,1,2}\\ \cdots \\ \mathbf{p}_{j,:,H_P,W_P}\end{bmatrix}, \begin{bmatrix}\mathbf{z}_1\\ \mathbf{z}_2\\ \cdots \\ \mathbf{z}_{H_P W_P}\end{bmatrix}\right)\right)$$

$$= \begin{bmatrix}\mathbf{p}_{j,:,1,1}\\ \mathbf{p}_{j,:,1,2}\\ \cdots \\ \mathbf{p}_{j,:,H_P,W_P}\end{bmatrix}^T \begin{bmatrix}\mathbf{z}_1\\ \mathbf{z}_2\\ \cdots \\ \mathbf{z}_{H_P W_P}\end{bmatrix}$$

$$= \sum_{h}^{H_P} \sum_{w}^{W_P} \langle \mathbf{p}_{j,:,h,w}, \mathbf{z}_{hw} \rangle$$

Deformations complicate this formulation, since the feature at a fractional location is defined to be an interpolation between the neighboring grid locations. In general, bilinear interpolation does not preserve the norm of the vectors it interpolates between. Deformable ProtoPNet [17] introduces a norm-preserving interpolation method to address this issue, but implementing this interpolation function requires complicated, low-level changes to a standard implementation of deformable convolution. In our code, we opted instead to use standard PyTorch [42] functionality rather than a custom implementation of deformable convolution. This offers two benefits: this approach is less error-prone since the PyTorch functionality is likely to receive support moving forward, and this approach avoids the need to build and install custom C++ code, as is required with the implementation from [17].

Concretely, in our deformable prototype implementation we 1) Predict a set of offsets using convolution, as in [17]. 2) Compute the feature value at each location suggested by these offsets using bilinear interpolation. This is implemented using PyTorch's version of the sampling method from [28]. 3) Re-normalize each interpolated vector to have the desired norm. 4) Compare our prototypes to these interpolated vectors, getting a similarity score at each required location.

Steps 2, 3, and 4 differ from the implementation suggested in [17], but maintain all aspects of the method other than the interpolation function used.

Table 9. **Stanford Dogs Test Accuracy and Model Sparsity.** Comparison of each model across three CNN backbones. *:T* indicates the *Best Test* model, and *:V* the *Best Validation*. #P is number of prototypes. For baselines it is estimated from Cosine Ablation experiment.

	DN161		**RN50**		**VGG19**	
	Acc	#P	Acc	#P	Acc	#P
ST-ProtoPNet	92.7	2000	85.7	1200	80.9	1000
Deformable	83.7	3700	–	–	77.9	4100
BOL2-PPN:V	82.5	706	82.6	1726	74.4	1244
BOL2-PPN:T	83.0	1201	83.1	1845	74.9	1641
BOCos-PPN:V	83.8	318	83.1	1000	75.6	736
BOCos-PPN:T	85.8	246	83.4	836	76.3	484

Table 10. **Stanford Cars Test Accuracy and Model Sparsity.** Comparison of each model across three CNN backbones. *:T* indicates the *Best Test* model, and *:V* the *Best Validation*. #P is number of prototypes. For baselines it is estimated from Cosine Ablation experiment.

	DN161		**RN50**		**VGG19**	
	Acc	#P	Acc	#P	Acc	#P
ST-ProtoPNet	92.7	4300	–	–	91.7	4300
TesNet	92.6	1300	–	–	90.6	1100
ProtoPool	–	–	88.9	195	–	–
BOL2-PPN:V	79.8	973	65.5	2198	80.0	2563
BOL2-PPN:T	79.8	973	65.5	2198	81.2	1353
BOCos-PPN:V	84.5	668	77.9	981	82.3	2511
BOCos-PPN:T	86.1	1409	78.6	1716	84.8	2187

10. Extended Optimization Results

10.1. Stanford Dogs

The Stanford Dogs dataset provides standardized train and test splits; as with CUB-200, we further partitioned the train set into train (90% of samples from the original train set) and validation (the remaining 10%) sets for our experiments. We fit all models on this train set, and used performance on the validation set to optimize hyperparameters. Results are in Supplementary Table 9.

10.2. Stanford Cars

The original Stanford Cars dataset – which was used in experiments by multiple previous ProtoPNet variations – is no longer available [3]. As a result, we constructed a new iteration from a publicly available Kaggle copy of the dataset.[4] Unfortunately, this version of the dataset does not have documented provenance, and there are multiple documented labeling errors in this version of the dataset.[5] Since the original dataset is not available, we are unable to compare the Kaggle version of the dataset to understand what discrepancies may exist.

Results are in Supplementary Table 10.

10.3. Normalized-ℓ_2

10.3.1. Accuracy and Sparsity

There are two major differences between the implementation of ℓ_2-similarity and cosine similarity. The first is the use of the logistic kernel, $\log \frac{\gamma+1}{\gamma+\epsilon}$. The second is the normalization of the input vectors before invoking gamma, which is done for cosine but not for ℓ_2. To ablate the effect of normalization, we applied to normalization to the ℓ_2-similarity yielding normalized-ℓ_2 similarity: $\gamma_{\bar{\ell_2}}(\mathbf{p}_j, \mathbf{z}) := \log(\|\frac{v(\mathbf{z})}{\|v(\mathbf{z})\|} - \frac{v(\mathbf{p}_j)}{\|v(\mathbf{p}_j)\|}\|_2^2+1) - (\log\|\frac{v(\mathbf{z})}{\|v(\mathbf{z})\|} - \frac{v(\mathbf{p}_j)}{\|v(\mathbf{p}_j)\|}\|_2^2+\epsilon)$.

We ran the same training as used in the Cosine Ablation experiments for ProtoPNet using normalized-ℓ_2 and covering DenseNet-161, ResNet-50 (pretrained on inaturalist), and VGG-19 on CUB-200 and Stanford Dogs. The resulting accuracies are in Table 11. The results show that, after optimization, the performance is very similar between the cosine similarity and normalized-ℓ_2 models, with cosine holding a slight edge ($< 1\%$) for all but one case: VGG-19 on CUB-200. This is generally the worst performing backbone-dataset combination across all models, and in this case, normalized-ℓ_2 performs similarity to unnormalized ℓ_2. This demonstrates that normalization is an important factor in the better performance of cosine, but that is not the only factor.

10.3.2. Clustering Structure

To better understand both the training dynamics that lead to the improvements from normalization and why VGG-19 on CUB-200 behaves differently, we compared the ℓ_{clst} and ℓ_{sep} at the time of the first project across similarity metrics when training ProtoPNet on CUB-200. These measures provide insight into the quality of the clustering structure the model has learned - high ℓ_{clst} means that prototypes are close to samples of their own classes. Low ℓ_{sep} indicates the prototypes are not close to samples from other classes. Note there are many incorrect classes and only one correct class.

One concern in making this comparison is that different similarity measures are on different scales. As a result, the same hyperparameters may not be appropriate for each similarity metric, which may confound our ability to compare cluster quality.

[3]https://ai.stanford.edu/~jkrause/cars/car_dataset.html

[4]https://www.kaggle.com/datasets/jessicali9530/stanford-cars-dataset/data

[5]https://www.kaggle.com/datasets/jessicali9530/stanford-cars-dataset/discussion/412690

Table 11. **Impact of Normalizing ℓ_2 on Accuracy and Sparsity.** This table extends the results reported in Table 1 and 2 to include models using normalized-ℓ_2 ($\gamma_{\bar{\ell}_2}$. *Top Section:* CUB-200. *Bottom Section:* Stanford Dogs. *:T* indicates the *Best Test* model, and *:V* the *Best Validation*. Results are from Cosine Ablation optimizations on ProtoPNet with additional optimizations for normalized-ℓ_2. Normalized-ℓ_2 has superior accuracy to ℓ_2 and similar to cosine in all but one case: VGG-19 on CUB-200 (this is generally the worst-performing setting for all models). We note that Cosine still has a minor advantage over normalized-ℓ_2 in other cases. Overall, normalization explains most but not all of the difference between ℓ_2 and cosine.

		DenseNet-161		**ResNet-50**		**VGG-19**	
		Acc	# Proto	Acc	# Proto	Acc	# Proto
CUB-200	**ProtoPNet-L2:V**	69.2	1983	84.5	1682	69.2	2134
	ProtoPNet-L2:T	69.2	1983	84.5	1682	69.2	2134
	ProtoPNet-Cos:V	78.6	1062	86.5	829	73.4	809
	ProtoPNet-Cos:T	78.6	1062	87.0	861	73.4	809
	ProtoPNet-NormalizedL2:V	77.8	713	85.8	1555	67.3	2134
	ProtoPNet-NormalizedL2:T	78.3	951	86.9	1172	69.0	1426
Stanford Dogs	**ProtoPNet-L2:V**	81.5	1390	83.5	507	72.0	1521
	ProtoPNet-L2:T	81.5	1390	83.5	507	72.0	1521
	ProtoPNet-Cos:V	86.3	571	84.7	661	75.2	1057
	ProtoPNet-Cos:T	86.4	706	84.8	538	75.8	1056
	ProtoPNet-NormalizedL2:V	85.2	1381	83.7	1224	75.0	1617
	ProtoPNet-NormalizedL2:T	85.8	1162	84.3	1225	75.3	1530

To control for this, we performed a grid search over learning rates and embedding dimensions for all three backbones (DenseNet-161, ResNet-50, VGG-19) and both datasets (CUB-200, CUB-200 Cropped, Stanford Dogs and Stanford Cars). We considered learning rate multipliers (`lr_multiplier`) of $\{0.01, 0.1, 1.0\}$ and `latent_dim_multiplier_exp` of $\{-4, -2, 0\}$ (see Section 8.2 for a detailed description of hyperparameters).

To evaluate cluster quality, we look at the ratio of cluster-to-separation $\ell_{\text{clst}} : \ell_{\text{sep}} := \frac{\ell_{\text{clst}}}{\ell_{\text{sep}}}$. Values that exceed 1 indicate that the model's prototype clustering structure has separated within-class patches from out-of-class patches, since prototypes tend to be closer to samples of their own class than those of others.

Figure 5 shows the breakout of $\ell_{\text{clst}} : \ell_{\text{sep}}$ for these hyperparameter sweeps. The values yielded by cosine similarity are very stable across all initializations and $\ell_{\text{clst}} : \ell_{\text{sep}}$ is close to 1.0 in all cases. This is superior to unnormalized-ℓ_2 in all settings, which is frequently in the $0.5 - 0.7$ range. The behavior of normalized-ℓ_2 is more complicated. The ratios yielded by normalized-ℓ_2 have higher variance than cosine, but this leads to superior $\ell_{\text{clst}} : \ell_{\text{sep}}$ in some cases and inferior in others. Notably, for VGG-19, normalized-ℓ_2's $\ell_{\text{clst}} : \ell_{\text{sep}}$ ratio is worse than cosine's for all hyperparameter combinations. This is in accordance with eventual accuracy (Table 11).

In total, we find that 1) cosine models produce more consistent and better clustering structures by the first project; 2) a large part of this improvement in clustering structure may be attributed to the normalization inherent in cosine similarity; and 3) that clustering structure at the time of first projection is predictive of accuracy by the completion of training.

10.3.3. Dimensional Collapse

[23] described a representation learning failure mode they termed *dimensional collapse*, where individual features in the latent space demonstrate strong correlation. This leads to an *effective* reduction in the total dimensions in the embedding space, which may limit the expressive power of the model. [29] measured this effect using the distribution of singular values in the latent embeddings (termed the *representation space spectrum*): if the magnitude of the largest singular values constitutes a large proportion of overall magnitude, then much of the variance the model can predict is concentrated in a few principal components. This is dimensional collapse.

We tested whether ProtoPNet models were suffering from dimensional collapse as a possible explanation for differences in performance between similarity metrics. Following [29], we apply the same representation space spectrum analysis to the latent embeddings of the best performing ProtoPNet CUB-200 models from the Cosine Ablation experiments using cosine, ℓ_2, and normalized-ℓ_2 as similarity measures for DenseNet-161, ResNet-50, and VGG-19.

We also aimed to determine the impact of dimensional collapse on accuracy, if any. Even if dimensional collapse was occurring in these models, it was possible that collapse would not inhibit predictive performance.

We generated accuracy insertion curves for the same em-

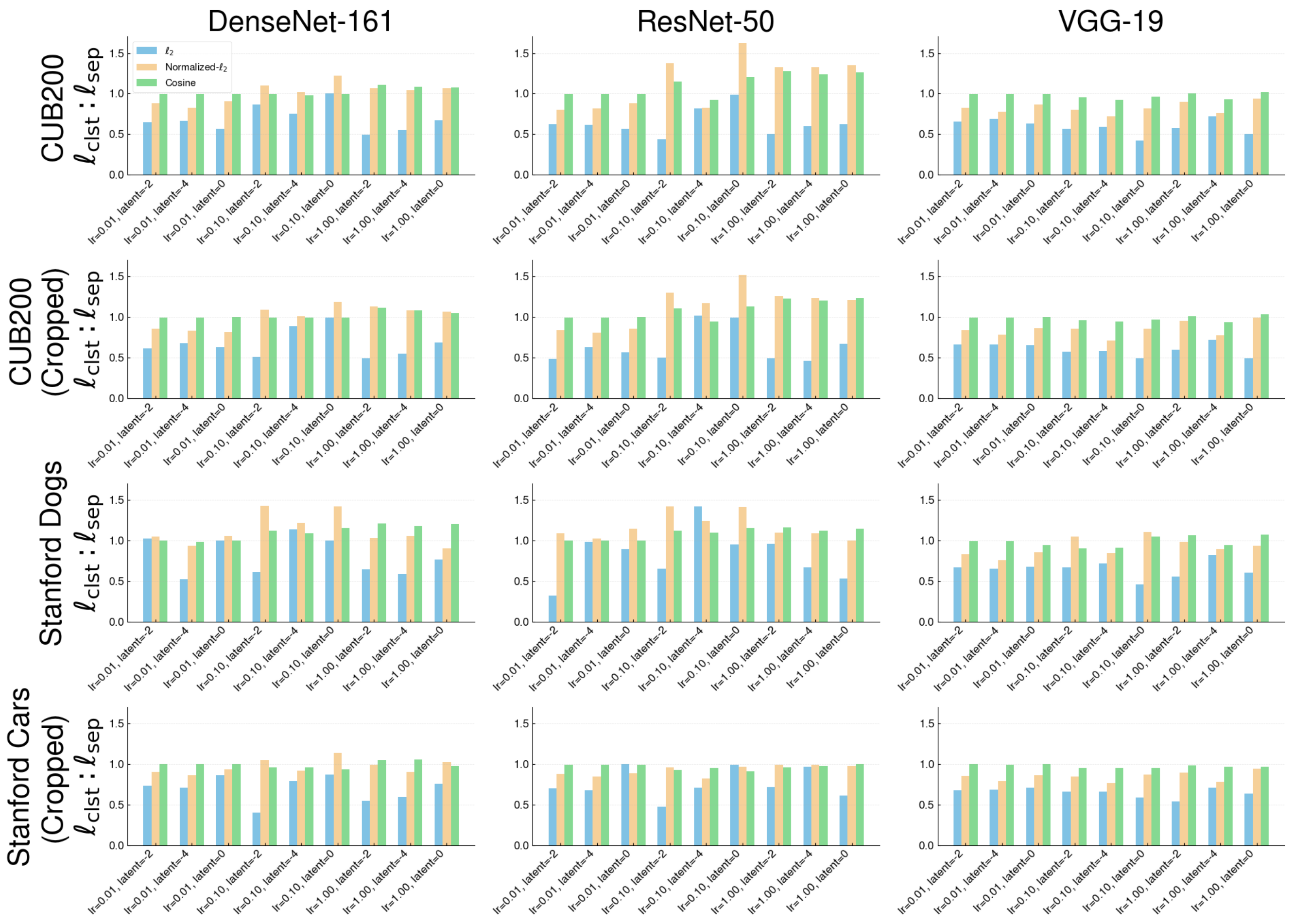

Figure 5. **$\ell_{\text{clst}} : \ell_{\text{sep}}$ at First Project.** Rows are datasets, columns are backbones. Each graph represents $\ell_{\text{clst}} : \ell_{\text{sep}}$ at first project for a combination of embedding dimension and learning rate. High values indicate the model has correctly separated clusters. Cosine models (blue bars) are consistently close to 1. ℓ_2 models are inferior, with $\ell_{\text{sep}} > \ell_{\text{clst}}$ consistently. Normalized-ℓ_2 models have high variance (sensitivity to hyperparameters), sometimes exceeding cosine but consistently outperforming unnormalized-ℓ_2. However, normalization does not close the gap between cosine and ℓ_2 when using VGG-19 (right column). While normalization improves the clustering structure, cosine models have better clustering structure in 34 of 36 cases, which matches the accuracy from Table 11.

beddings with the following logic: We ordered principal components by magnitude of their associated singular values and ran prediction using only the top-k principal components for $k \in 1, D$, where D is the number of channels in the latent embedding for each model tested. The resulting curves are in Figure 6.

These models do indeed experience dimensional collapse, indicated by the differences in magnitude of the maximum singular values and a mean singular values. However, the dimensional collapse is greater for ℓ_2 models than cosine models. ℓ_2 models have greater concentration of explanation of variance in their top few dimensions than Cosine models, which can be seen by the larger downward slope of their singular value curves.

This is also directly correlated with the predictive power of those dimensions. The accuracy insertion curves show that ℓ_2 models have more predictive power in their highest-singular-value principal components than cosine models (the curves rise more quickly than cosine), and less predictive power later in the curve (the curves plateau more quickly than cosine). This is precisely what we would expect if dimensional collapse was concentrating discriminatory information in few dimensions. Cosine models distribute this information more evenly, ultimately having more predictive power when using the whole representational space.

The normalized-ℓ_2 curves generally fall between the cosine and ℓ_2 curves and demonstrate similar patterns. Normalization appears to resolve some dimensional collapse for ResNet-50 and DenseNet-161, but not for VGG-19. This is consistent with the accuracy metrics.

By analyzing dimensional collapse, we found that part of the reason ℓ_2 models perform worse than cosine models may

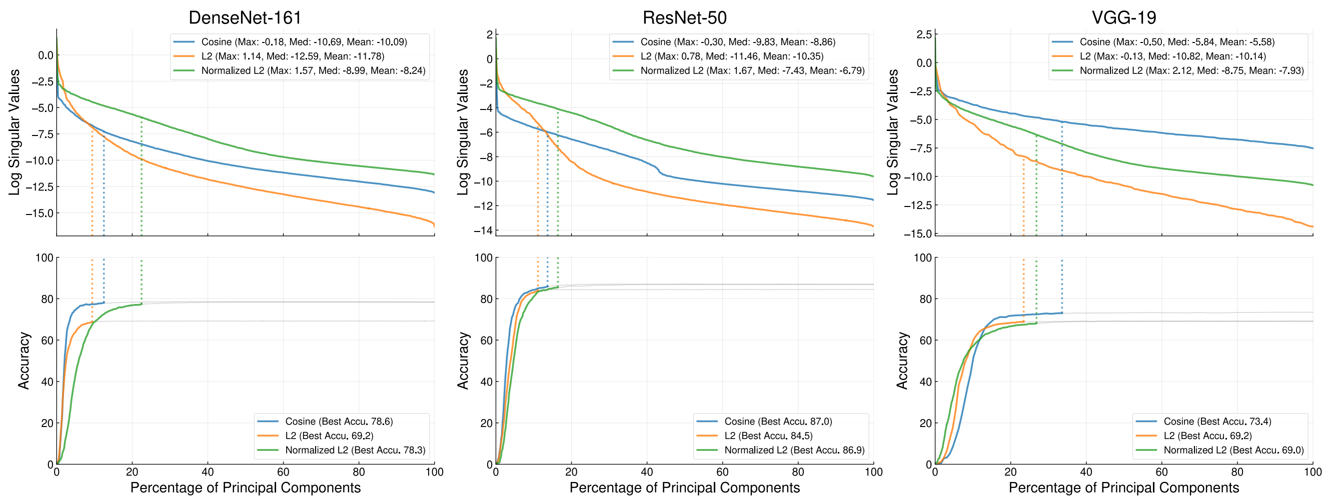

Figure 6. **Dimensional Collapse for ProtoPNet**. *Top Row*: Log-scale singular values for test set embeddings of best performing ProtoPNet models from Cosine Ablation. ℓ_2 models have less distribution in their principal components, indicating greater dimensional collapse. Cosine models have more distributed singular values. Normalized-ℓ_2 models are in-between, with DenseNet-161 and ResNet-50 models have less collapse than cosine, but VGG-19 having collapse like ℓ_2. *Bottom Row*: Accuracy insertion curves by principal components for best performing Cosine Ablation ProtoPNet Models. Lines are grayed out after accuracy converges. Most of the predictive power of ℓ_2 models is contained within their top principal components, indicating only non-collapsed dimensions are useful for prediction. Both cosine and normalized-ℓ_2 models have more distribution of their discriminatory power in lower singular values and better discriminatory power at convergence.

be that they experience a greater degree of dimensional collapse. Like problems in clustering structure, normalization mitigates some of this effect but not its entirety.

10.4. Extended Cosine Ablation

Supplementary Table 12 provides accuracies and sparsities for all optimizations in the Cosine Ablation experiments.

10.5. Distribution of Accuracy and Prototype Metrics

Figure 7 shows that, across both the accuracy-only optimization and the joint accuracy-prototype score optimization, there are a large number of ProtoPNet models that have similar accuracies but materially different prototype quality scores, which provides an empirical justification for how improvement along those axis is possible. The greatest variance is along sparsity, which explains why the joint optimization had such a major effect on sparsity. Even with cosine similarity, there is a negative correlation between prototype stability and accuracy, which suggests a future avenue of improvement.

10.6. Estimating Prototype Consolidation

Latent prototypes are parameters trained jointly with other model parameters during backpropogation. To ensure these represent actual images, a project (or push) step moves these prototypes to the nearest sample patches in latent space. During this operation, multiple prototypes may have the nearest sample patch. For all the ProtoPNet varieties we studied, this leads to *prototype consolidation*, where a single prototype is represented multiple times in the model. As a result, the initial hyperparameter of the number of prototypes in a model is actually an upper-bound on the number of prototypes. This results of this process have not been well-documented previously, with previous work documenting the hyperparameters rather than the resulting prototype distributions. This makes comparisons of model sparsity a challenge.

In our work, for all models we trained, we directly measured the number of prototypical parts (and their source images) to report precise estimates of sparsity. However, we still faced a challenge when comparing to previous work. For instance, ST-ProtoPNet's best results were on models using 40 prototypes-per-class, but we did not practically expect these models to have 8,000 prototypes on CUB-200.

To allow these comparisons, we used the models we trained in our Cosine Ablation to estimate prototype consolidation for previously published models. We took the most accurate 10% of each of the model types by backbone and prototype similarity, and calculated the number of prototypes at the end of the optimization. We expressed this as a percentage of the number of prototypes created during random initialization. This gives us a prototype consolidation coefficient for each model, backbone, similarity triplet. We then multiplied this by the previously report maximum prototype hyperparameters and rounded to the nearest 100. The estimated coefficients are in Supplementary Table 13.

Table 12. **Table of Cosine Ablation Results**. Comparison of each model across three CNN backbones. *:T* indicates the *Best Test* model, *:V* the *Best Validation*, #P is Number of Prototypes and Sp is Prototype Sparsity. These results do not reflect fully optimized versions of these models, but rather show how easy these models are to optimize. Across all models and backbones, cosine optimized models have better accuracy than their ℓ_2 counterparts. Except for Deformable ProtoPNet, they consistently have better sparsity as well. Since ProtoTree shares prototypes between classes, its sparsity actually exceeds 1.0 on CUB-200 - it has less prototypes than classes. As before, the best performing VGG model is Deformable ProtoPNet, which also has the largest number of prototypes. ProtoPNet, ST-ProtoPNet, and TesNet all perform well with with DenseNet-161 and ResNet-50.

		DenseNet-161			**ResNet-50**			**VGG-19**		
		Acc	#P	Sp	Acc	#P	Sp	Acc	#P	Sp
CUB-200	**TesNet-L2:V**	74.0	2633	7.6	86.3	2410	8.3	66.0	2612	7.7
	TesNet-L2:T	74.0	2633	7.6	86.5	1934	10.3	66.4	2264	8.8
	TesNet-Cos:V	77.4	1204	16.6	86.2	1601	12.5	72.5	764	26.2
	TesNet-Cos:T	77.4	1204	16.6	86.9	1588	12.6	72.5	764	26.2
	ST-ProtoPNet-L2:V	76.5	2743	7.3	82.8	2215	9.0	64.8	2789	7.2
	ST-ProtoPNet-L2:T	77.2	2655	7.5	84.4	2680	7.5	65.2	2313	8.6
	ST-ProtoPNet-Cos:V	78.1	2731	7.3	86.5	912	21.9	74.2	2724	7.3
	ST-ProtoPNet-Cos:T	79.6	479	41.8	87.4	881	22.7	75.5	1029	19.4
	ProtoTree-L2:V	40.5	198	101.0	73.6	198	101.0	12.4	190	105.3
	ProtoTree-L2:T	40.5	198	101.0	73.6	198	101.0	14.9	195	102.6
	ProtoTree-Cos:V	64.2	199	100.5	82.5	199	100.5	57.0	200	100.0
	ProtoTree-Cos:T	64.2	199	100.5	83.2	199	100.5	58.9	189	105.8
	ProtoPNet-L2:V	69.2	1983	10.1	84.5	1682	11.9	69.2	2134	9.4
	ProtoPNet-L2:T	69.2	1983	10.1	84.5	1682	11.9	69.2	2134	9.4
	ProtoPNet-Cos:V	78.6	1062	18.8	86.5	829	24.1	73.4	809	24.7
	ProtoPNet-Cos:T	78.6	1062	18.8	87.0	861	23.2	73.4	809	24.7
	Deformable-L2:V	50.0	1744	11.5	76.3	4784	4.2	46.5	9941	2.0
	Deformable-L2:T	50.0	1744	11.5	76.3	4784	4.2	46.5	9941	2.0
	Deformable-Cos:V	76.9	7764	2.6	86.1	7904	2.5	73.4	9775	2.0
	Deformable-Cos:T	77.6	7360	2.7	87.0	7536	2.7	74.5	9019	2.2
Stanford Dogs	**TesNet-L2:V**	82.1	1627	7.4	82.4	1631	7.4	74.1	1471	8.2
	TesNet-L2:T	83.2	844	14.2	82.4	1631	7.4	74.3	1186	10.1
	TesNet-Cos:V	84.4	1289	9.3	83.9	1417	8.5	74.2	765	15.7
	TesNet-Cos:T	85.2	1372	8.7	84.4	1219	9.8	75.5	1334	9.0
	ST-ProtoPNet-L2:V	82.2	1640	7.3	80.2	1585	7.6	70.0	1217	9.9
	ST-ProtoPNet-L2:T	82.6	1660	7.2	81.6	1343	8.9	72.8	1676	7.2
	ST-ProtoPNet-Cos:V	84.1	311	38.6	84.4	420	28.6	75.1	628	19.1
	ST-ProtoPNet-Cos:T	84.1	800	15.0	84.6	322	37.3	75.1	628	19.1
	ProtoTree-L2:V	68.2	121	99.2	72.8	127	94.5	39.9	124	96.8
	ProtoTree-L2:T	69.1	122	98.4	72.8	127	94.5	39.9	124	96.8
	ProtoTree-Cos:V	77.6	122	98.4	77.1	128	93.8	65.1	129	93.0
	ProtoTree-Cos:T	77.6	122	98.4	77.1	128	93.8	65.1	129	93.0
	ProtoPNet-L2:V	81.5	1390	8.6	83.5	507	23.7	72.0	1521	7.9
	ProtoPNet-L2:T	81.5	1390	8.6	83.5	507	23.7	72.0	1521	7.9
	ProtoPNet-Cos:V	86.3	571	21.0	84.7	661	18.2	75.2	1057	11.4
	ProtoPNet-Cos:T	86.4	706	17.0	84.8	538	22.3	75.8	1056	11.4
	Deformable-L2:V	15.9	564	21.3	10.4	1277	9.4	37.4	6475	1.9
	Deformable-L2:T	16.0	1600	7.5	10.4	1277	9.4	37.4	6475	1.9
	Deformable-Cos:V	85.0	2612	4.6	83.9	6132	2.0	77.2	6144	2.0
	Deformable-Cos:T	85.5	5000	2.4	83.9	6132	2.0	77.3	5236	2.3

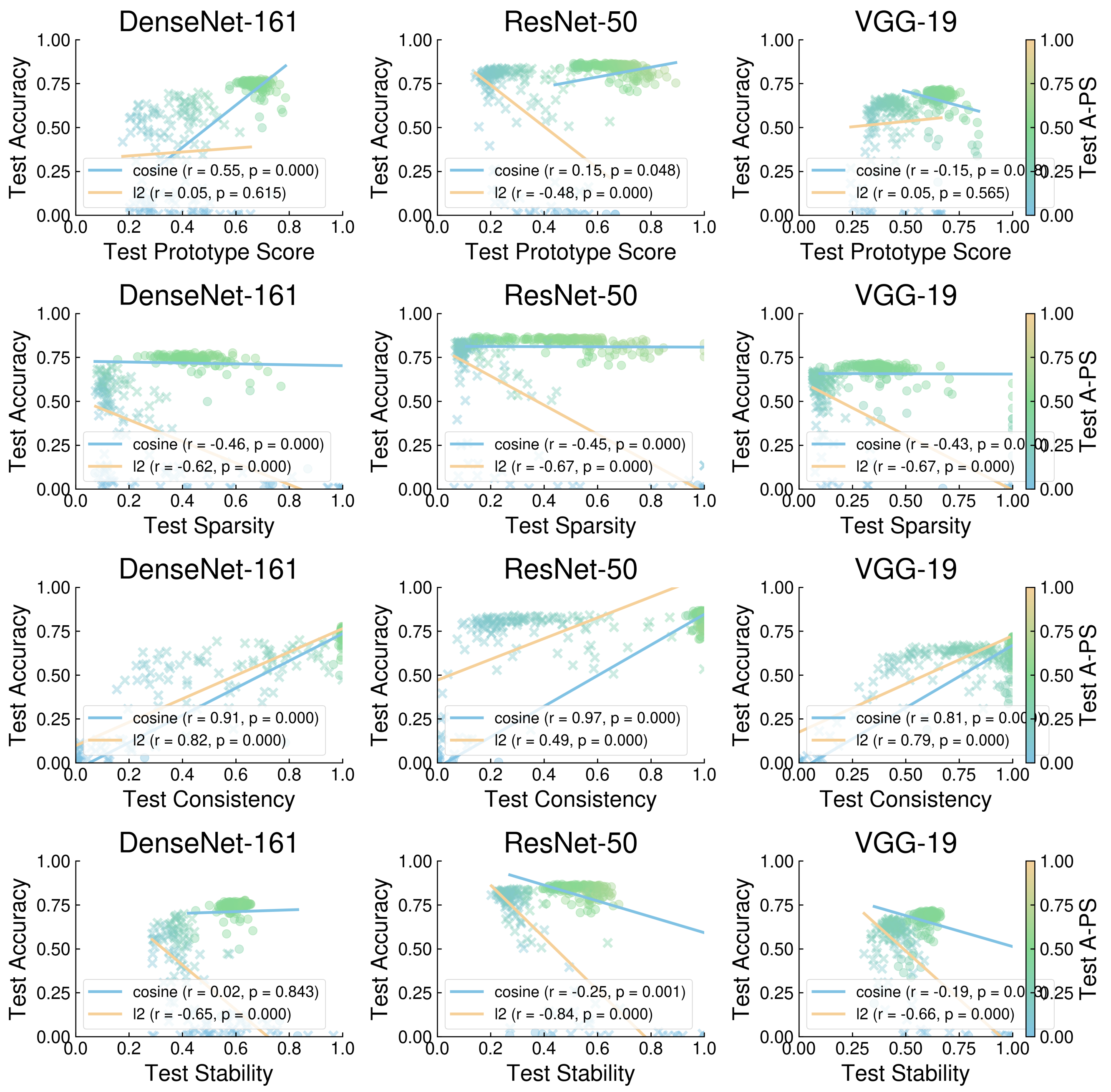

Figure 7. **Model Accuracy and Prototype Score Correlation for Cosine Similarity Models.** *A-PS* is accuracy-prototype score, the joint optimization obj$_{\text{aps}}$. Includes BOL2-ProtoPNet and BOCos-ProtoPNet optimized under the joint objective. *Top Row:* Accuracy and combined Prototype Score. *Second Row:* Accuracy and combined Prototype Sparsity. *Third Row:* Accuracy and combined Prototype Consistency. *Forth Row:* Accuracy and combined Prototype Stability. The clear positive relationship between consistency and accuracy holds for both ℓ_2 and cosine optimized models. These plots show that cosine improves on a strong negative correlation between accuracy and sparsity, but it does not entirely solve it. The relationship with Prototype Stability is more complicated. Stability is much more concentrated than the other two - it does not appear to be particularly sensitive to hyperparameter selection, but cosine also improves the relationship between stability and accuracy.

Table 13. **Prototype Consolidation Coefficients.** Coefficients estimating the percentage of prototypes that remain after consolidation due to projection on high performing models. Estimated using the top 10% of models from the Cosine Ablation experiments. These were used to estimate number of prototypes in Sections 4 and 5 (rounded to the nearest 100 prototypes).

		DenseNet-161		**ResNet-50**		**VGG-19**	
		Cos	ℓ_2	Cos	ℓ_2	Cos	ℓ_2
Deformable	**CUB200**	54.7	18.5	63.9	28.7	89.4	74.3
	Stanford Dogs	76.6	17.6	74.5	23.3	85.3	57.7
ProtoPNet	**CUB200**	28.9	64.8	29.1	63.3	39.0	81.8
	CUB200 (Cropped)	27.7	47.9	29.3	69.0	35.8	75.6
	Stanford Cars (Cropped)	25.6	76.4	31.9	66.3	29.4	81.4
	Stanford Dogs	40.5	74.2	37.2	70.1	72.6	76.2
ProtoTree	**CUB200**	99.3	99.8	100.0	99.9	97.7	97.6
	Stanford Dogs	99.2	100.0	99.2	99.6	98.5	99.2
ST-ProtoPNet	**CUB200**	68.4	97.3	42.2	94.0	88.9	95.0
	Stanford Dogs	41.8	92.9	24.2	86.9	21.8	90.7
TesNet	**CUB200**	52.2	94.6	59.6	83.7	34.5	83.8
	Stanford Dogs	76.3	74.9	72.6	89.1	74.7	79.7

11. Additional Visualizations

In this section, we provide a large set of additional visualizations from the most accurate cosine-based ResNet-50 model that was used in Figure 1.

Supplementary Figure 13 illustrates 15 prototypes from this model and the 10 input images that yielded the highest activation for them. Across all 15 prototypes, we see strong, consistent semantics in what the prototypes look for, further showing that jointly optimizing for prototype metrics improves the semantics of the resulting model.

Additionally, several instances of this model forming a prediction on images from the validation set are presented. We observe that, in all cases examined, the model follows a coherent reasoning process, even in cases where the model was confounded. Supplementary Figure 8 shows a Western Grebe being correctly classified, Supplementary Figure 9 illustrates a Evening Grosbeak being correctly classified, Supplementary Figure 10 shows a Pied Kingfisher being correctly classified, and Figure 11 illustrates a Mallard being correctly classified.

Supplementary Figure 12 illustrates a Chipping Sparrow being incorrectly classified as a Tree Sparrow. In this case, the model confuses the light brown diagonal patch on the Tree Sparrow's breast with the light brown wing pattern of a Tree Sparrow.

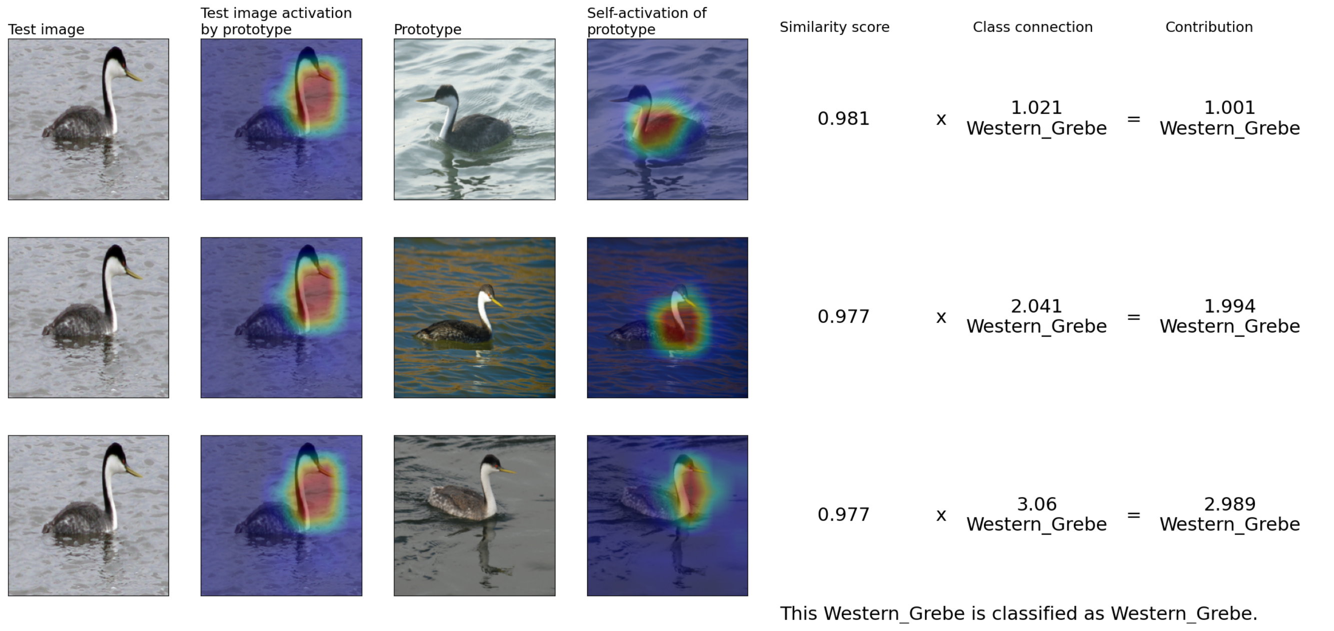
Test image
Test image activation by prototype
Prototype
Self-activation of prototype
Similarity score
Class connection
Contribution
0.981 x 1.021 Western_Grebe = 1.001 Western_Grebe
0.977 x 2.041 Western_Grebe = 1.994 Western_Grebe
0.977 x 3.06 Western_Grebe = 2.989 Western_Grebe
This Western_Grebe is classified as Western_Grebe.

Figure 8. Reasoning process for the most accurate cosine-based model on an image of a Western Grebe. The model correctly classifies this image.

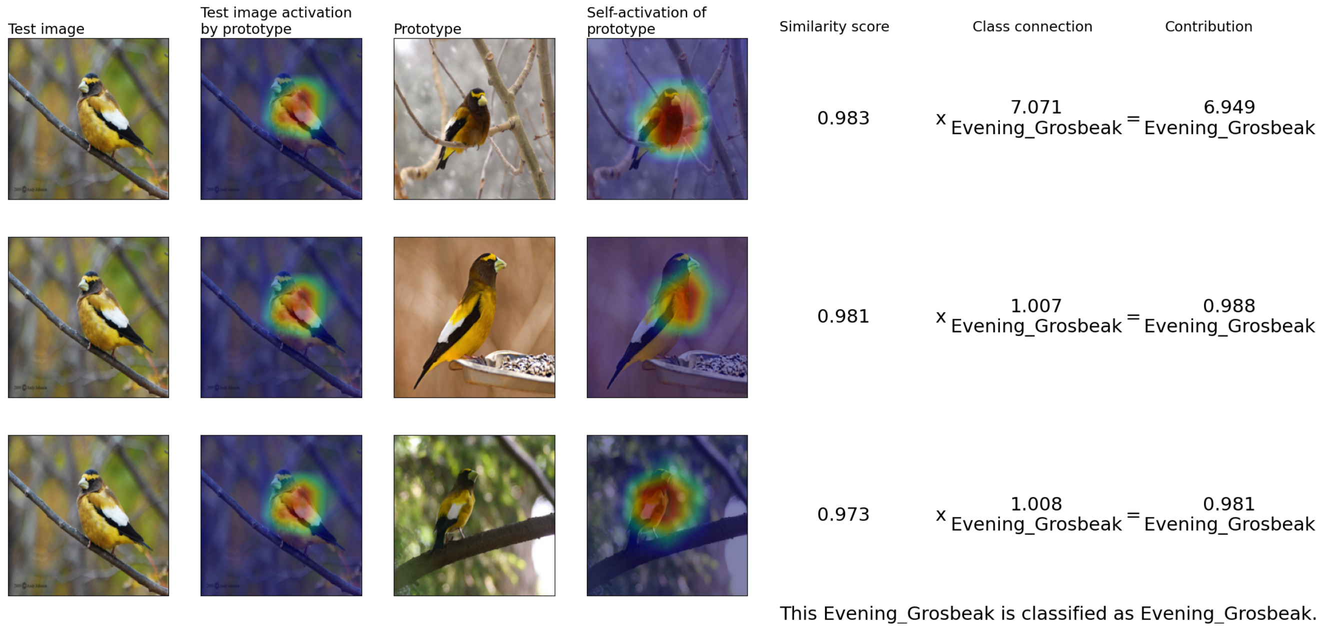
Test image
Test image activation by prototype
Prototype
Self-activation of prototype
Similarity score
Class connection
Contribution
0.983 x 7.071 Evening_Grosbeak = 6.949 Evening_Grosbeak
0.981 x 1.007 Evening_Grosbeak = 0.988 Evening_Grosbeak
0.973 x 1.008 Evening_Grosbeak = 0.981 Evening_Grosbeak
This Evening_Grosbeak is classified as Evening_Grosbeak.

Figure 9. Reasoning process for the most accurate cosine-based model on an image of a Evening Grosbeak. The model correctly classifies this image.

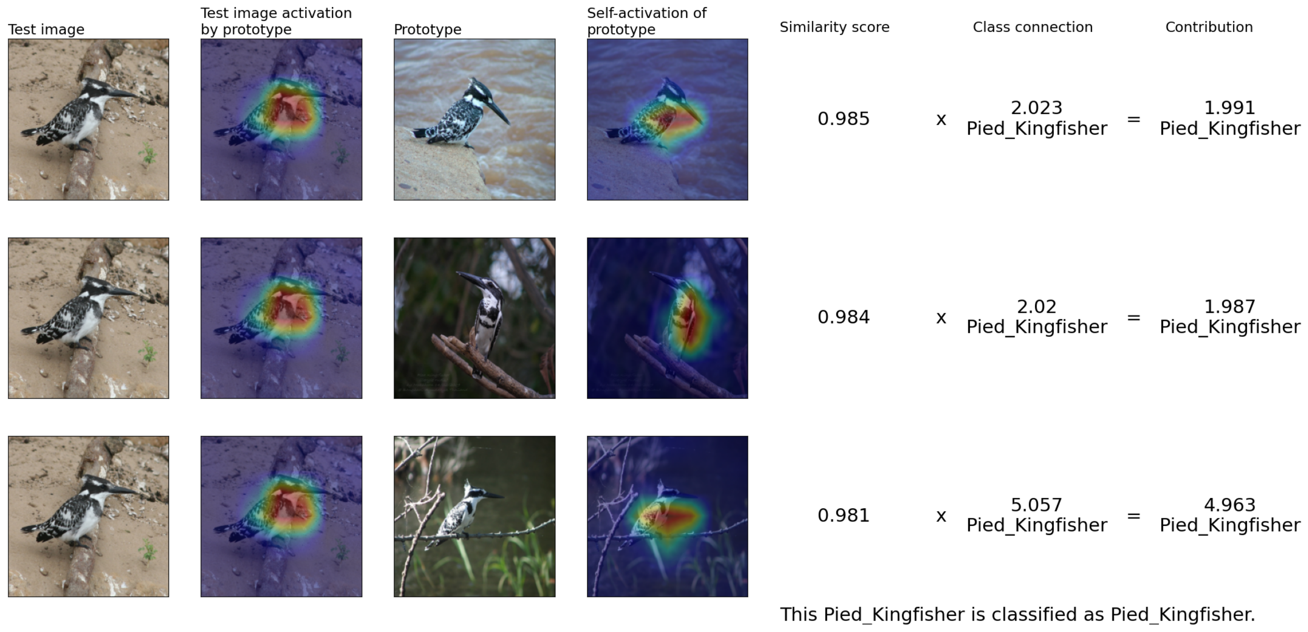
Test image
Test image activation by prototype
Prototype
Self-activation of prototype
Similarity score
Class connection
Contribution
0.985 x 2.023 Pied_Kingfisher = 1.991 Pied_Kingfisher
0.984 x 2.02 Pied_Kingfisher = 1.987 Pied_Kingfisher
0.981 x 5.057 Pied_Kingfisher = 4.963 Pied_Kingfisher
This Pied_Kingfisher is classified as Pied_Kingfisher.

Figure 10. Reasoning process for the most accurate cosine-based model on an image of a Pied Kingfisher. The model correctly classifies this image.

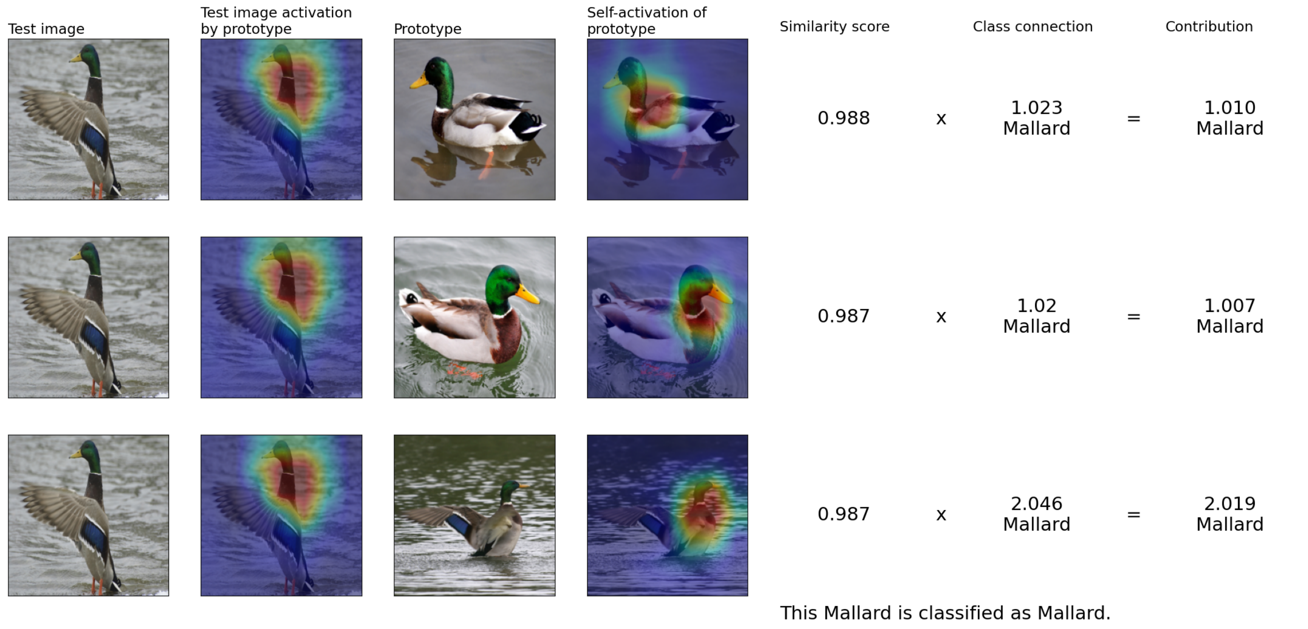
Test image
Test image activation by prototype
Prototype
Self-activation of prototype
Similarity score
Class connection
Contribution
0.988 x 1.023 Mallard = 1.010 Mallard
0.987 x 1.02 Mallard = 1.007 Mallard
0.987 x 2.046 Mallard = 2.019 Mallard
This Mallard is classified as Mallard.

Figure 11. Reasoning process for the most accurate cosine-based model on an image of a Mallard. The model correctly classifies this image.

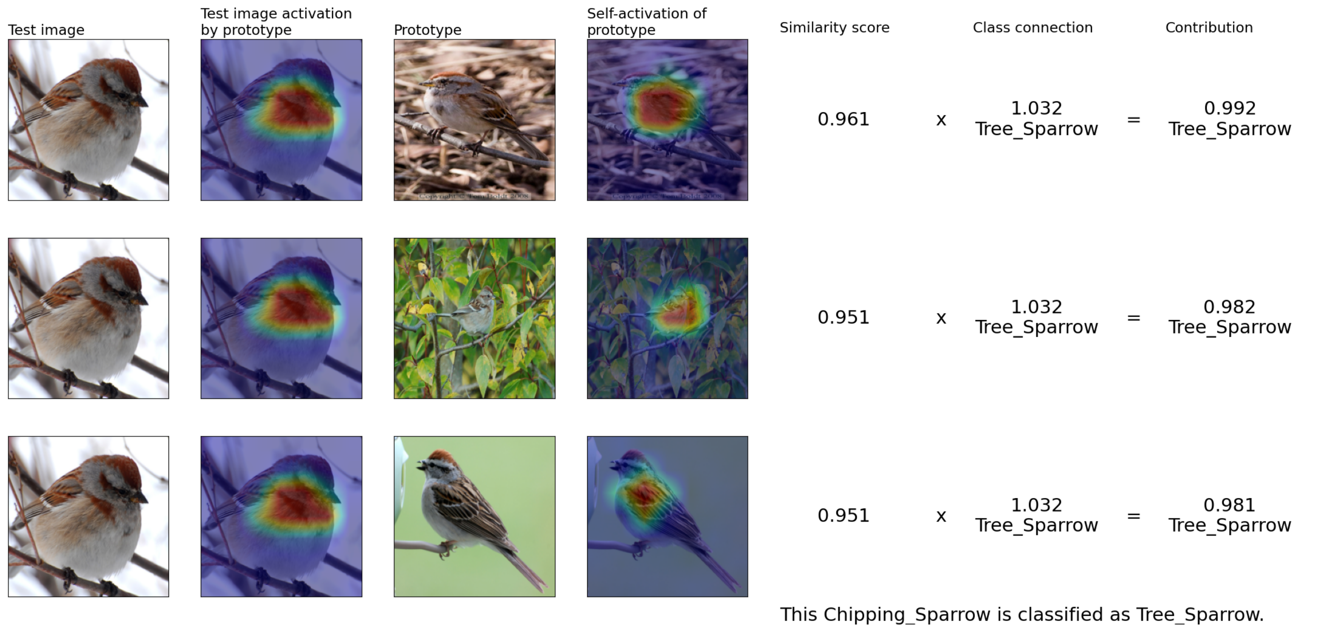
Test image
Test image activation by prototype
Prototype
Self-activation of prototype
Similarity score
Class connection
Contribution
0.961 x 1.032 Tree_Sparrow = 0.992 Tree_Sparrow
0.951 x 1.032 Tree_Sparrow = 0.982 Tree_Sparrow
0.951 x 1.032 Tree_Sparrow = 0.981 Tree_Sparrow
This Chipping_Sparrow is classified as Tree_Sparrow.

Figure 12. Reasoning process for the the most accurate cosine-based model on an image of a Chipping Sparrow. The model incorrectly classifies this image as a Tree Sparrow, apparently confounded by the light brown, diagonal patch on the Chipping Sparrow's breast, which the model considers similar to the light brown diagonal wing pattern of a Tree sparrow.

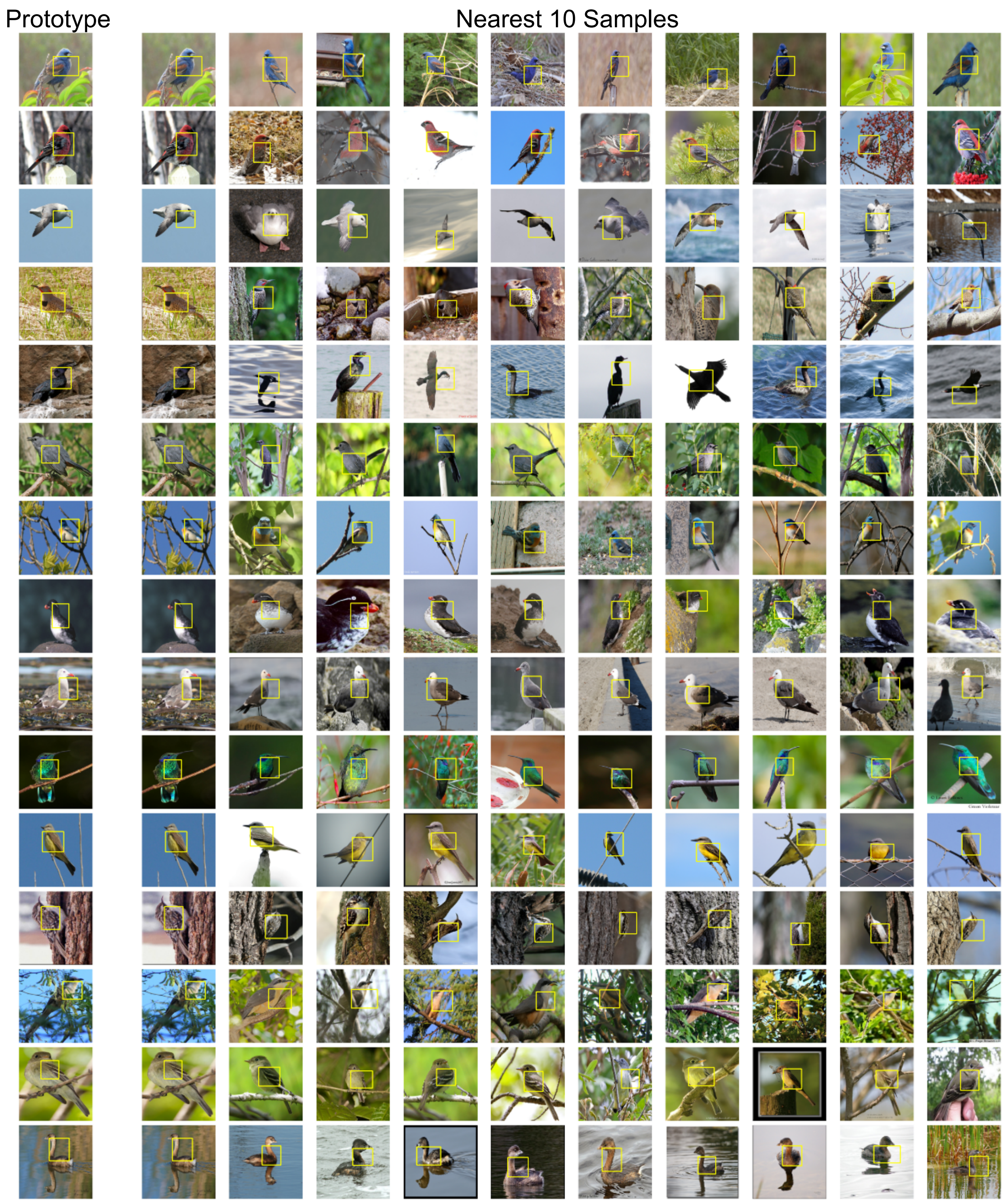

Figure 13. Each row in the figure shows the prototype in the first column, followed by the ten images with the highest activations for that prototype (sorted by activation in descending order). In each image, a yellow bounding box is shown to highlight the image patch match to the prototype. The closest match to the prototype is always itself.