Neural Prototype Trees for Interpretable Fine-grained Image Recognition: Supplementary Material

Meike Nauta[1] Ron van Bree[1] Christin Seifert[1,2]
[1] University of Twente, the Netherlands [2] University of Duisburg-Essen, Germany
m.nauta@utwente.nl, r.j.vanbree@student.utwente.nl, christin.seifert@uni-due.de

S1. Training Details

We train the neural network f and the prototypes of a ProtoTree with Adam [6], and the leaves with our derivative-free algorithm. As shown in Table S1, the prototypes and leaves are only a fraction of the trainable parameters and therefore barely give any overhead. However, note that the number of prototypes and number of leaves will exponentially increase when increasing height h.

Part	Layer	(Output) Shape	Total # Parameters
f	ResNet50 (without avgpool and fc-layer)	(2048, 7, 7)	23,508,032
	1×1 Conv2D	(256, 7, 7)	524,288
$\boldsymbol{P}$		$255 \times 1 \times 1 \times 256$	65,280
$\boldsymbol{c}$		256×200	51,200
Total			24M

Table S1: Trainable parameters in a ProtoTree with height $h = 8$ and $D = 256$, for CUB-200-2011 with 200 classes.

For CUB, we use the backbone of [9] pretrained on iNaturalist2017 for 180 epochs. For CARS, we use a ResNet50 pretrained on ImageNet. This backbone, except for the last convolutional layer, is frozen for some epochs (Table S2). The 1×1 convolutional layer is initialized with Xavier initialization [4]. The prototypes, the last layers of f, and the backbone each have their specified learning rate, as indicated in Table S2.

Data Augmentation For CUB, we crop each training image offline into four corners based on the bounding box annotations, and include the full image resulting in 5 images per original image. We then resize each image to 224×224. Test images are not cropped and resized to 224×224. To make our visualizations comparable to ProtoPNet [1], we select the nearest training image patch for each prototype by considering cropped training images only.

Data	Parameter	Value
All	Batch size	64
	Input image size	224×224
	Output image size	7×7
	H_1	1
	W_1	1
	Learning rate prototypes	0.001
	Learning rate 1×1 conv layer and last conv layer ResNet50	0.001
	Gamma for lr decay	0.5
	Epochs backbone frozen	30
CUB	Learning rate pretrained ResNet50 (except last layer)	$1e-5$
	Pruning threshold τ	0.01
	Number of epochs (after offline data augmentation)	100
	Milestones for lr decay	60,70,80,90,100
CARS	Learning rate pretrained ResNet50 (except last layer)	$2e-4$
	Pruning threshold τ	0.01
	Number of epochs	500
	Milestones for lr decay	250,350,400,425, 450,475,500

Table S2: Parameter values when training ProtoTrees for our experiments.

For CARS, we do not use any annotations. We resize all images to 256×256, apply online data augmentation and then take a random crop of size 224×224. Comparable to ProtoPNet [2], we apply data augmentation including random rotation, shearing, distortion, color jitter and horizontal flipping. Data augmentation details (applied in an online fashion and implemented in PyTorch) are shown in Table S3. More complex training and augmentation techniques, such as AutoAugment [3] and cyclic learning rates [8], are not used to keep a fair comparison, but might

improve accuracy. Similarly, applying more advanced ensemble techniques, such as bagging and boosting, might improve the prediction accuracy of a ProtoTree ensemble.

Data	Augmentation	Value/Scale
All	Brightness jitter	(0.6, 1.4)
	Contrast jitter	(0.6, 1.4)
	Saturation jitter	(0.6,1.4)
	Horizontal flip	$p = 0.5$
	Random shear	(-2, 2)
	Normalization	mean 0.485,0.456,0.406 std 0.229,0.224,0.225
CUB	Hue jitter	(-0.02, 0.02)
	Random rotation	10
	Random translation	(0.05, 0.05)
	Perspective distortion	0.2 ($p = 0.5$)
	Resize	(224×224)
CARS	Hue jitter	(-0.4, 0.4)
	Random rotation	15
	Perspective distortion	0.5 ($p = 0.5$)
	Resize	(256×256)
	Random Crop	(224×224)

Table S3: Online data augmentation. Jitter values are based on [5], except for smaller hue differences since color hue can be discriminative for classes in CUB.

S2. Prototype Visualization with Class Constraints

Prototypes are trainable vectors that, after training, can be replaced with a latent patch of a training image. Equation 6 (main paper) shows that the nearest training patch $\tilde{\boldsymbol{z}}_n^*$ can be found by looping through all images in the training set. Whereas ProtoPNet has class-specific prototypes, our prototypes can be of any class. However, we argue that the perceptual interpretability of a prototype in ProtoTree T can be improved by only considering images that have a certain class label.

Specifically, we require that $\boldsymbol{x}_n^*$ should be from the majority class of any of the leaves reachable by node n. For each internal node n and corresponding prototype $\boldsymbol{p}_n$, we define $T'_n \subset T$ as a full binary subtree of T with n as root node, such that $\mathcal{Y}'_n$ is the corresponding set of class labels $\{\text{argmax}_{\boldsymbol{c}_\ell}$ for all ℓ in $T'_n\}$. $\boldsymbol{\mathcal{T}}'_n \subseteq \boldsymbol{\mathcal{T}}$ is the set of training images with class label $\in \mathcal{Y}'$. Then, Equation 6 from the main paper can be adapted as follows:

$$\boldsymbol{p}_n \leftarrow \tilde{\boldsymbol{z}}_n^{*'}, \quad \tilde{\boldsymbol{z}}_n^{*'} = \underset{\boldsymbol{z} \in \{f(\boldsymbol{x}), \forall \boldsymbol{x} \in \boldsymbol{\mathcal{T}}'_n\}}{\text{argmin}} ||\tilde{\boldsymbol{z}}^* - \boldsymbol{p}_n||. \quad \text{(S1)}$$

We denote by $\boldsymbol{x}_n^*$ the training image corresponding to nearest patch $\tilde{\boldsymbol{z}}_n^*$ when considering all training data, and $\boldsymbol{x}_n^{*'}$ denotes the training image corresponding to nearest patch $\tilde{\boldsymbol{z}}_n^*$ with class restrictions as defined in Equation S1. In our experiments on CUB, we found that the difference in Euclidean distance from $\boldsymbol{p}_n$ to $\tilde{\boldsymbol{z}}_n^{*'}$ with $\boldsymbol{p}_n$ to $\tilde{\boldsymbol{z}}_n^*$ was 5.86×10^{-5} on average, and is therefore negligible. Figure S1 visualizes three prototypes with and without such constraints ($\tilde{\boldsymbol{z}}_n^{*'}$ and $\tilde{\boldsymbol{z}}_n^*$). Both visualization methods (with or without class contraints) also give a similar prediction accuracy, as shown in Table S4. Interestingly, adding the class constraints even slightly improves accuracy.

Visualization method	Accuracy
Without class constraints (Eq. 6)	82.195 ± 0.723
With class constraints (Eq. S1)	82.199 ± 0.726

Table S4: Accuracy of ProtoTree after pruning and visualization for CUB ($h = 9$) across 5 runs.

Thus, adding the restriction that $\boldsymbol{x}_n^*$ should be from the majority class of any of the leaves reachable by node n does not negatively impact the accuracy of the model, but could improve interpretability. Our results in the main paper and Supplementary material are based on prototype replacement with class constraints.

S3. Detailed Results

Table S5 compares the deterministic classification strategies with the soft strategy for a ProtoTree trained on CARS. It shows that, similar to the results for CUB, selecting the leaf with the highest path probability leads to nearly the same prediction accuracy as soft routing, since the fidelity is 1. The greedy strategy performs slightly worse but its fidelity is still close to 1. Interestingly, pruning a ProtoTree of height 11 trained on CARS leads to a much smaller tree, with an average path length of only 8.6.

Figure S2 shows the maximum values of all leaf distributions for trained ProtoTrees on CARS or CUB. It can be seen that almost all leaves learn either one class label, or an almost uniform distribution ($1/K$).

Table S6) presents the detailed results for ProtoTrees of various heights trained on CUB or CARS.

Strategy	Accuracy	Fidelity	Path length
Soft	86.58±0.24	n.a.	n.a.
Max $\boldsymbol{\pi}_\ell$	86.58±0.24	1.000±0.000	8.6 ± 1.7 $(11, 4)$
Greedy	86.43±0.30	0.992±0.002	8.6 ± 1.7 $(11, 4)$

Table S5: Soft vs. deterministic classification strategies at test time. Fidelity is agreement with soft strategy. Min and max path lengths in brackets. ProtoTree on CARS (h=11, pruned and replaced), averaged over 5 runs (mean, stdev).

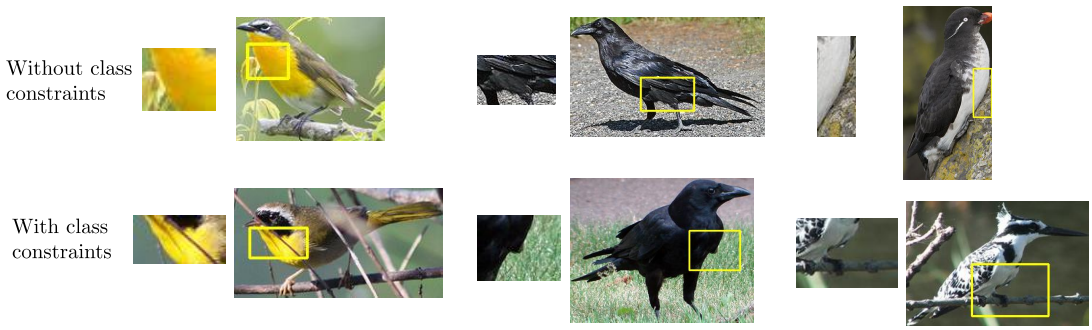
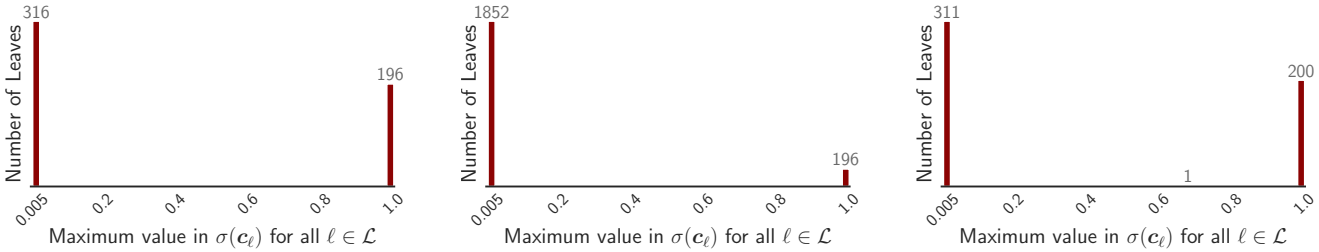


Figure S1: Three prototypes occurring in a ProtoTree trained on CUB. The upper row shows prototypes when considering all images for prototype replacement (Eq. 6). The bottom row shows prototypes when only those images are considered that have a class label that is from the majority class of any of the reachable leaves (Eq. S1). For example, the left column shows that the prototype represents a white belly. For a human classifying a bird similar to the bottom left image, perceptual similarity might be higher for the bottom left prototype than the upper left prototype.



(a) CARS, $h = 9$ (b) CARS, $h = 11$ (c) CUB, $h = 9$

Figure S2: Maximum values of all leaf distributions in a trained ProtoTree.

Dataset	h	Initial Acc	Acc pruned	Acc pruned+vis.	# Prototypes	% Pruned	Distance $\tilde{z}_n^*$
CUB ($K = 200$)	7	41.826 ± 2.776	41.826 ± 2.776	41.798 ± 2.780	127.0 ± 0.0	0.0	0.0027 ± 0.0045
	8	81.046 ± 0.674	81.042 ± 0.676	81.032 ± 0.680	200.4 ± 1.2	21.4	0.0025 ± 0.0047
	9	82.206 ± 0.723	82.192 ± 0.723	82.199 ± 0.726	201.6 ± 1.9	60.5	0.0020 ± 0.0068
	10	82.054 ± 0.517	82.019 ± 0.468	82.019 ± 0.469	203.2 ± 2.0	80.1	0.0018 ± 0.0072
	11	82.370 ± 0.575	82.357 ± 0.580	82.352 ± 0.572	207.0 ± 2.7	89.9	0.0038 ± 0.0313
CARS ($K = 196$)	7	53.842 ± 0.733	53.842 ± 0.733	53.847 ± 0.732	127.0 ± 0.0	0.0	0.0006 ± 0.0018
	8	85.049 ± 0.384	85.007 ± 0.398	85.017 ± 0.393	195.0 ± 0.0	23.5	0.0005 ± 0.0018
	9	85.601 ± 0.361	85.586 ± 0.361	85.586 ± 0.361	195.2 ± 0.4	61.8	0.0027 ± 0.0626
	10	86.064 ± 0.187	86.071 ± 0.191	86.076 ± 0.186	195.8 ± 1.2	80.9	0.0005 ± 0.0017
	11	86.584 ± 0.250	86.576 ± 0.245	86.576 ± 0.245	195.4 ± 0.5	90.5	0.0005 ± 0.0016

Table S6: Mean and standard deviation across 5 runs of: 1) accuracy before pruning and visualization, 2) accuracy after pruning, 3) accuracy after pruning and visualization, 4) number of prototypes after pruning, 5) fraction of prototypes that is pruned and 6) Euclidean distance from each latent prototype to its nearest latent training patch (after pruning).

S4. More Visualized ProtoTrees

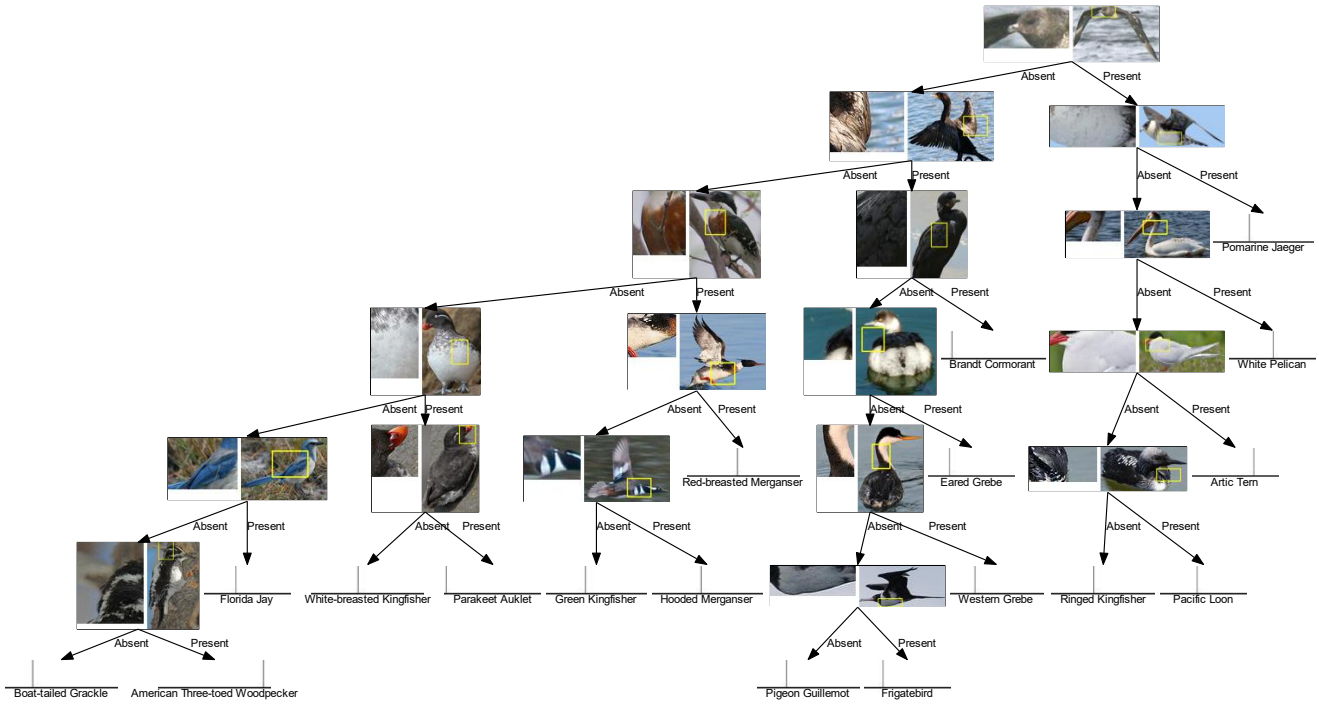


Figure S3: Subtree of a ProtoTree (CUB, $h = 9$). Each internal node contains a prototype (left) and the training image from which it is extracted (right). Each leaf shows the class probability distribution and the label of the class with the highest probability. Prototypes seem to correctly represent distinctive parts. For example, the Green Kingfisher, Hooded Merganser and Red-breasted Merganser all have a red-brown spot. Interpreting the top node is a bit challenging. A local explanation showing the similarity with a test image, or supplementary explanations as presented in [7] could help to clarify this.

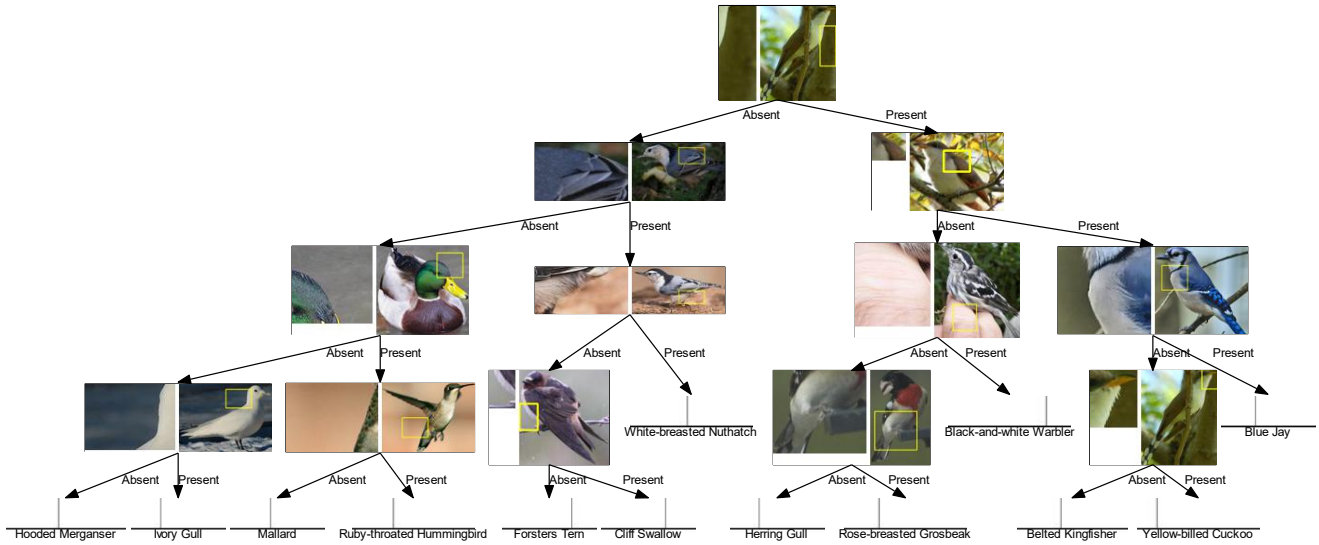


Figure S4: Subtree of an automatically visualized ProtoTree (CUB, $h = 8$). Each internal node contains a prototype (left) and the training image from which it is extracted (right). The Mallard and Ruby Throated Hummingbird share the same green-colored prototype.

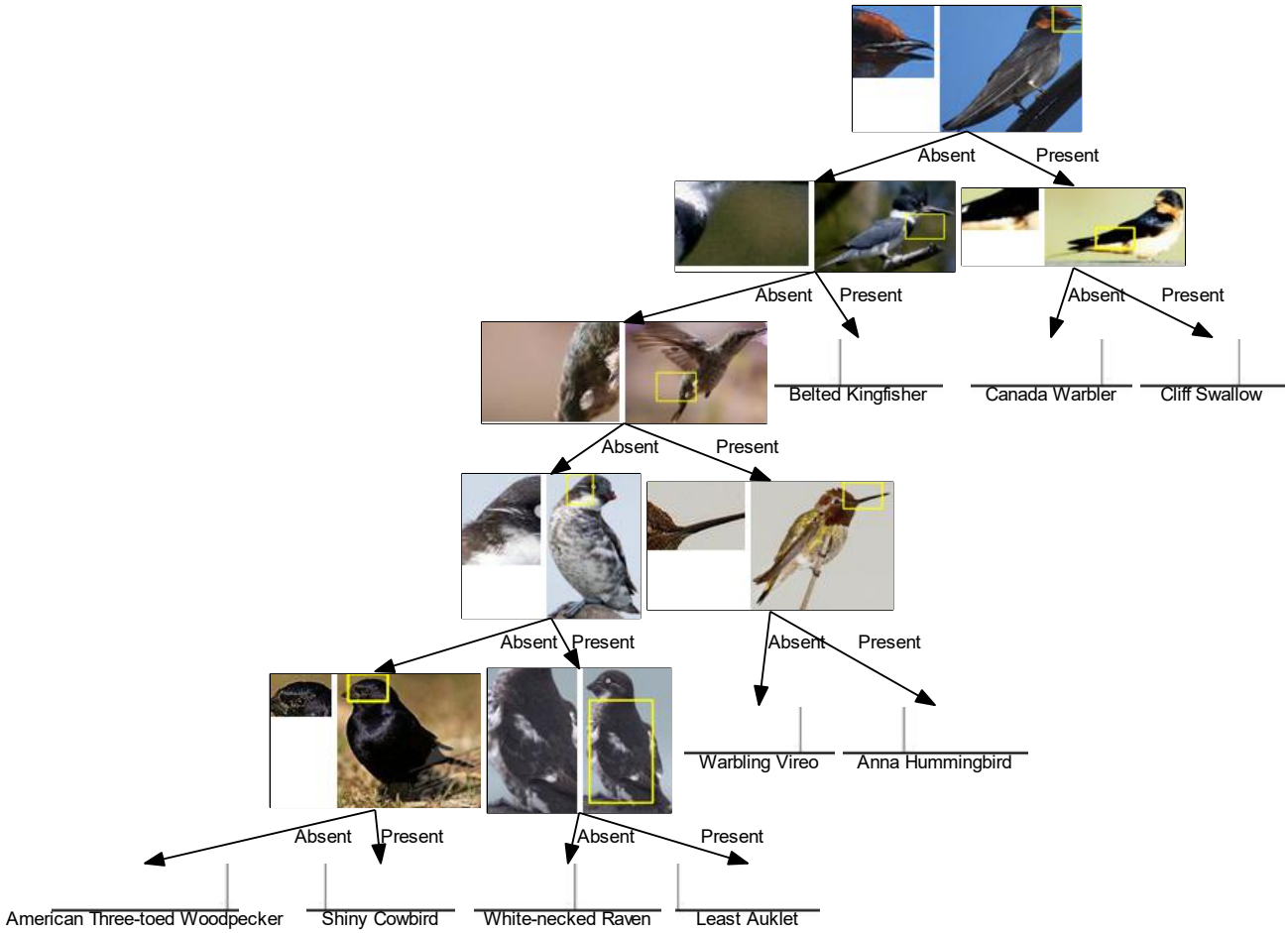


Figure S5: Subtree of a ProtoTree (CUB, $h = 10$). The Anna Hummingbird is recognized by its specific, long bill. Generally, a higher maximum height h results, after pruning, in a less balanced tree.

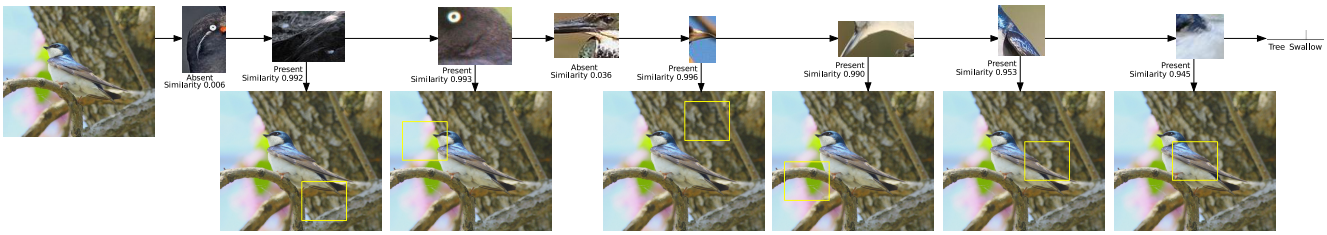


Figure S6: Local explanation for classifying a test image of a Tree Swallow. Interestingly, the 6th prototype could be detected in the test image because of the white-colored chest or because of the similarity with the curved branch. An explanation as presented by [7] to indicate whether color hue or shape is important, could clarify this.

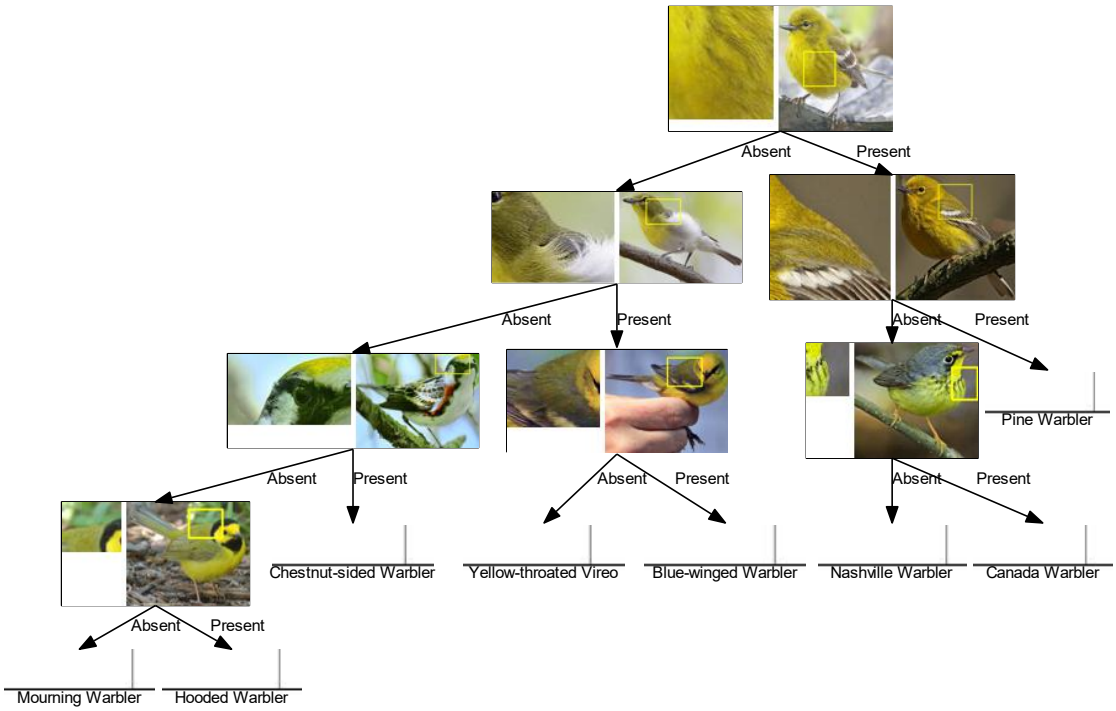


Figure S7: Subtree of an automatically visualized ProtoTree (CUB, $h = 8$). A ProtoTree hierarchically clusters similar classes, in this case Warblers.

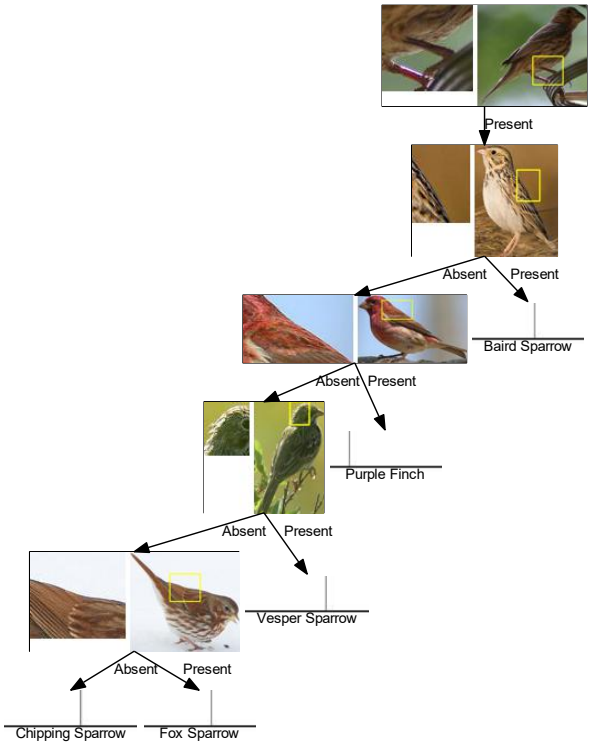


Figure S8: Subtree of a ProtoTree (CUB, $h = 9$). The top node clusters birds with red legs and a light colored abdomen. Pruning can result in a deep, imbalanced tree.

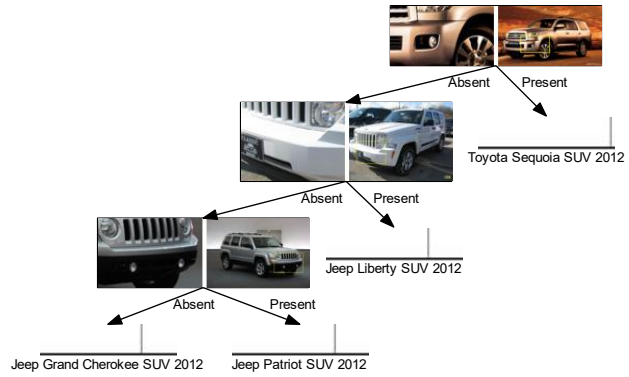


Figure S9: Subtree of a ProtoTree (CARS, $h = 10$) which clusters similar SUV's. Here, pruning results in an imbalanced tree.

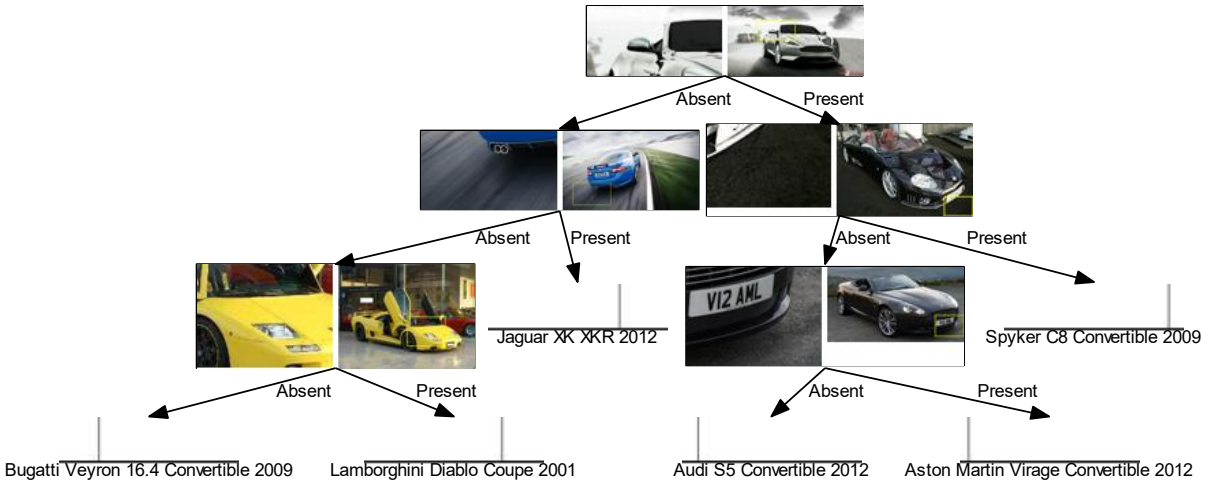


Figure S10: Subtree of a ProtoTree (CARS, $h = 9$) with convertibles clustered on the right. Each internal node contains a prototype (left) and the training image from which it is extracted (right).

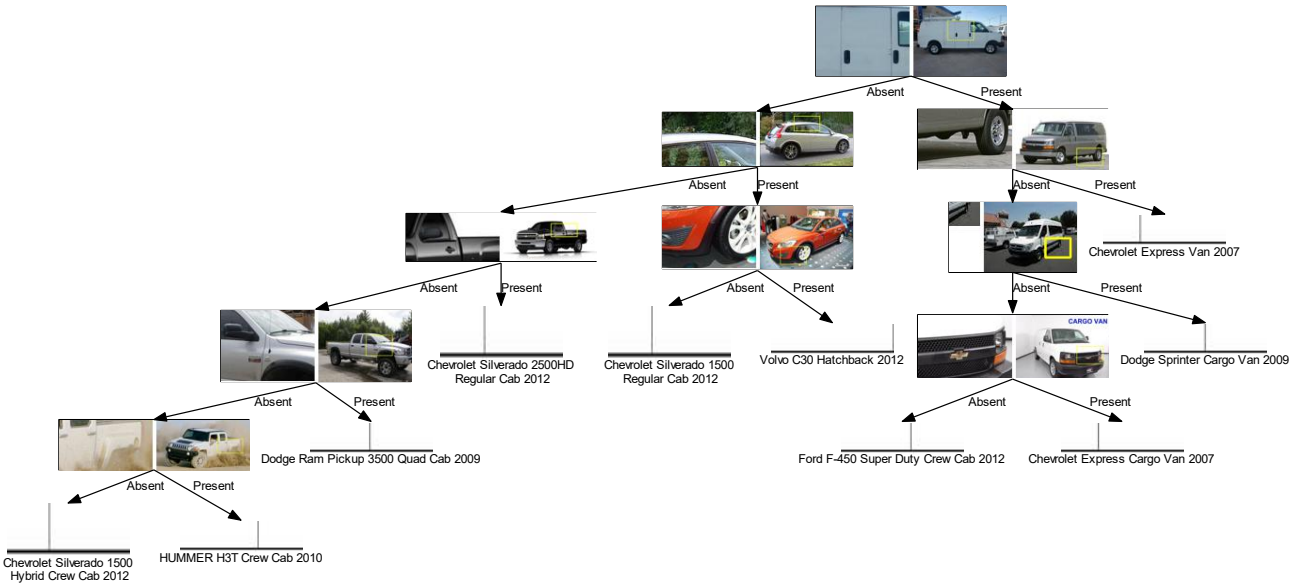


Figure S11: Subtree of a ProtoTree (CARS, $h = 11$). Vans are clustered on the right, and pickup trucks on the left.

Figure S12: Local explanation for classifying a test image of a Dodge Sprinter Cargo Van 2009 (CARS, $h = 11$), recognizable by the black stripe on the side.

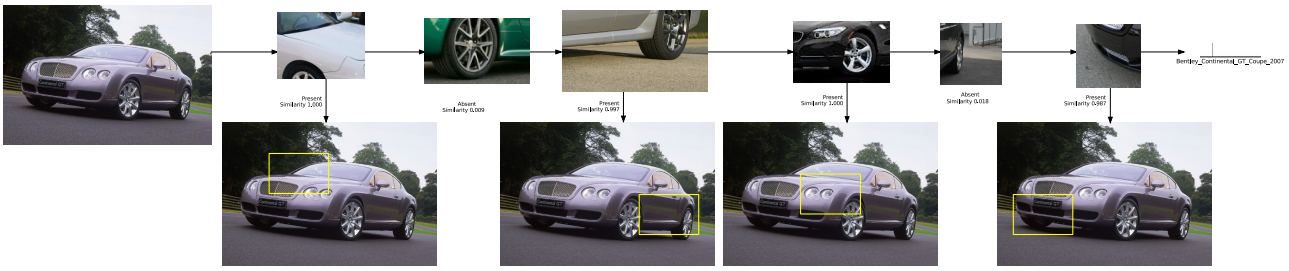


Figure S13: Local explanation for classifying a test image of a Bentley Continental GT Coupe 2007 (CARS, $h = 11$).

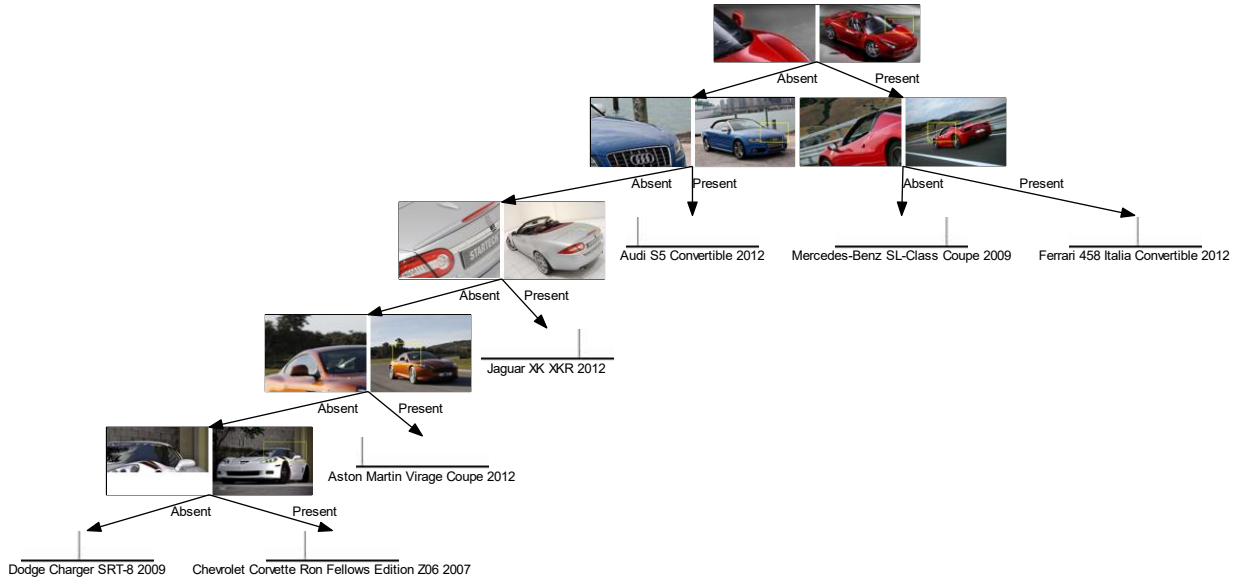


Figure S14: Subtree of a ProtoTree (CARS, $h = 10$). The top node clusters two cars that are styled with similar feature lines on the hood. The Audi is recognized by its logo.

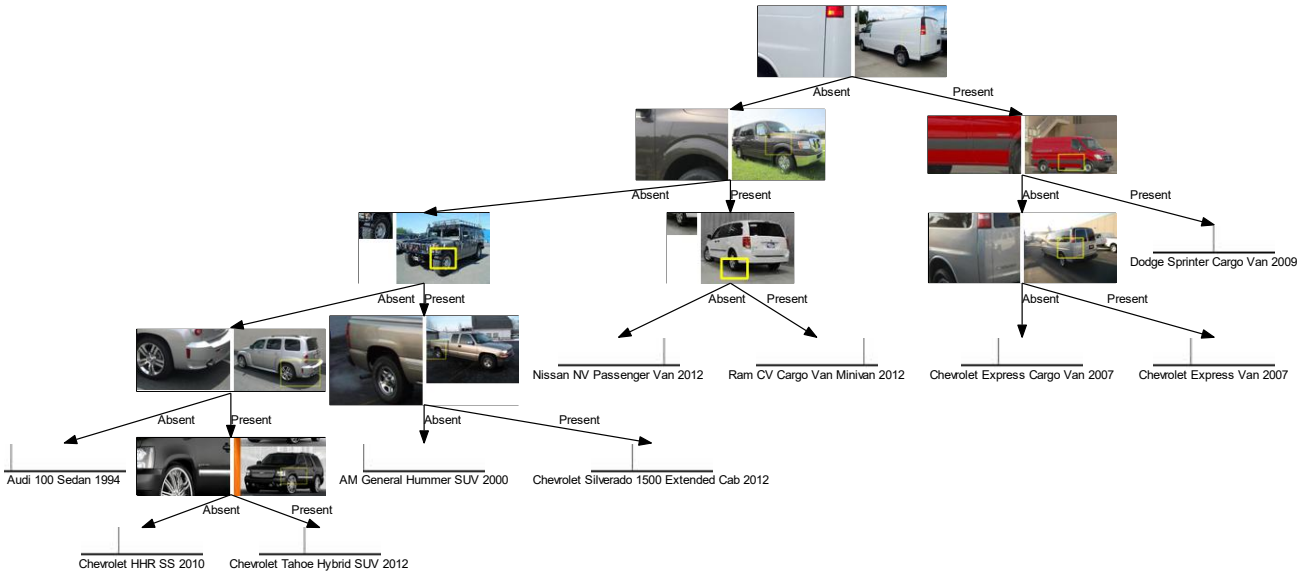


Figure S15: Subtree of a ProtoTree (CARS, $h = 10$). Similar vans are clustered on the right. Chrevrolets are recognized by their distinctive back.

References

[1] Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. This looks like that: Deep learning for interpretable image recognition. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8930–8941. Curran Associates, Inc., 2019. 1

[2] Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. This looks like that: Deep learning for interpretable image recognition. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, volume Supplement S9, pages 8930–8941. Curran Associates, Inc., 2019. 1

[3] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 1

[4] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010. 1

[5] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 558–567, 2019. 2

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1

[7] Meike Nauta, Annemarie Jutte, Jesper Provoost, and Christin Seifert. This looks like that, because ... explaining prototypes for interpretable image recognition, 2020. 4, 5

[8] L. N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472, 2017. 1

[9] Boyan Zhou, Quan Cui, Xiu-Shen Wei, and Zhao-Min Chen. Bbn: Bilateral-branch network with cumulative learning for long-tailed visual recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9719–9728, 2020. 1