TRANSFERLAB

Latent space prototype interpretability: Strengths and shortcomings

Giorgia Pitteri

September 28, 2023

Table of contents

Introduction

Interpretability, why?

- Deep Neural Networks are very powerful but **black boxes**
- Why should we trust these models for high stake decisions?
- Decision trees or linear classifiers are intrinsically **interpretable** but not very powerful

The goal is to define a form of interpretability in image processing similar to the way humans describe their thinking in classification tasks.

Interpretability, why?

- Explaining a model with *post-hoc* explanations, such as saliency maps, doesn't work [6]

- Part-based attention methods expose the parts of an input image the network focuses on when making decisions but not the relationship to prototypical cases

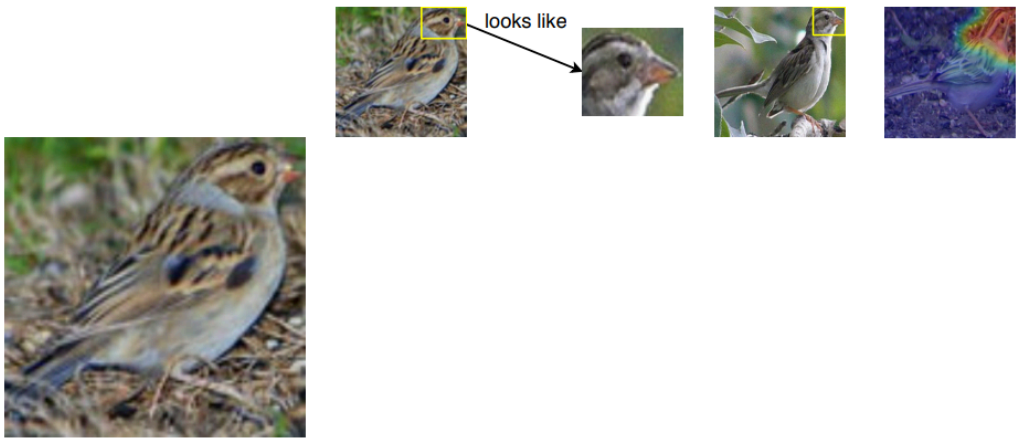
Prototypical Part Network

Proto P-Net [1]

- Stop explaining a black box model $\longrightarrow$ create an interpretable model!
- Focus on **fine-grained image recognition** task
- How would you classify the bird in the picture as a clay colored sparrow?

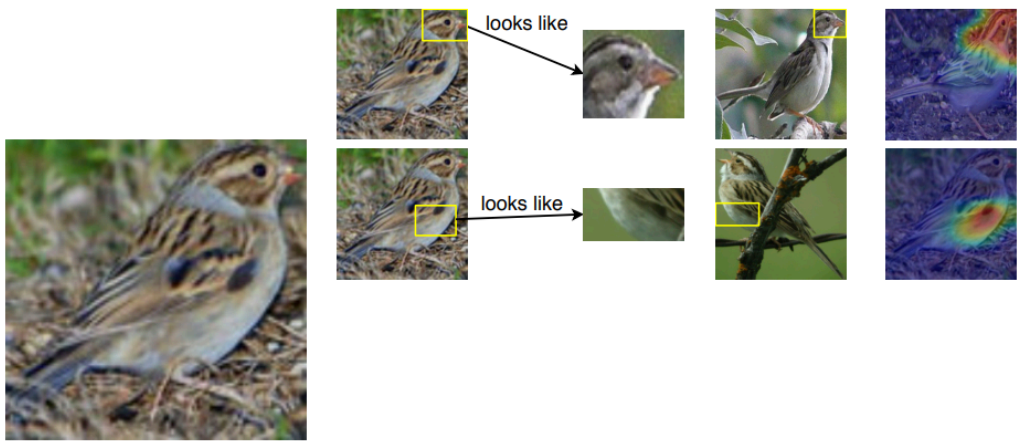
Proto P-Net

- Stop explaining a black box model $\longrightarrow$ create an interpretable model!
- Focus on **fine-grained image recognition** task
- How would you classify the bird in the picture as a clay colored sparrow?



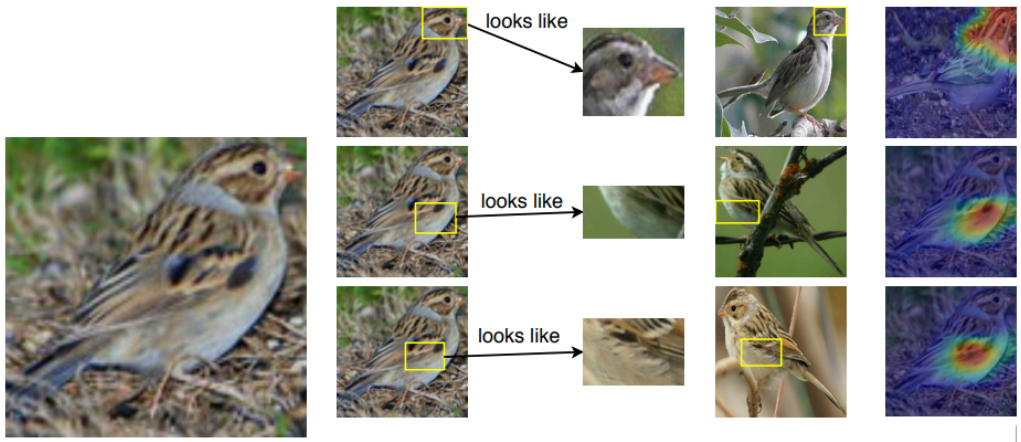
Proto P-Net

- Stop explaining a black box model $\longrightarrow$ create an interpretable model!
- Focus on **fine-grained image recognition** task
- How would you classify the bird in the picture as a clay colored sparrow?



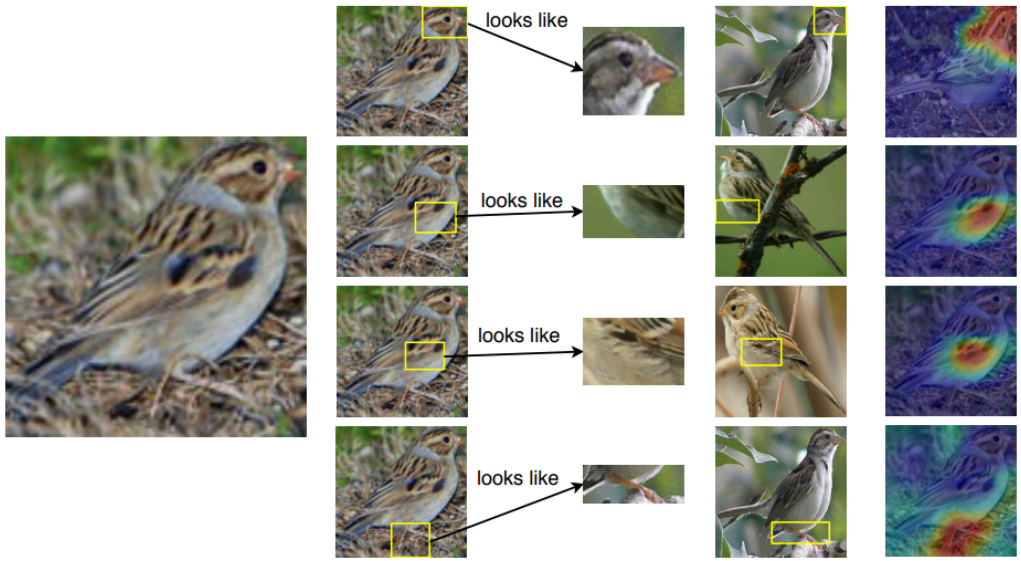
Proto P-Net

- Stop explaining a black box model $\longrightarrow$ create an interpretable model!
- Focus on **fine-grained image recognition** task
- How would you classify the bird in the picture as a clay colored sparrow?



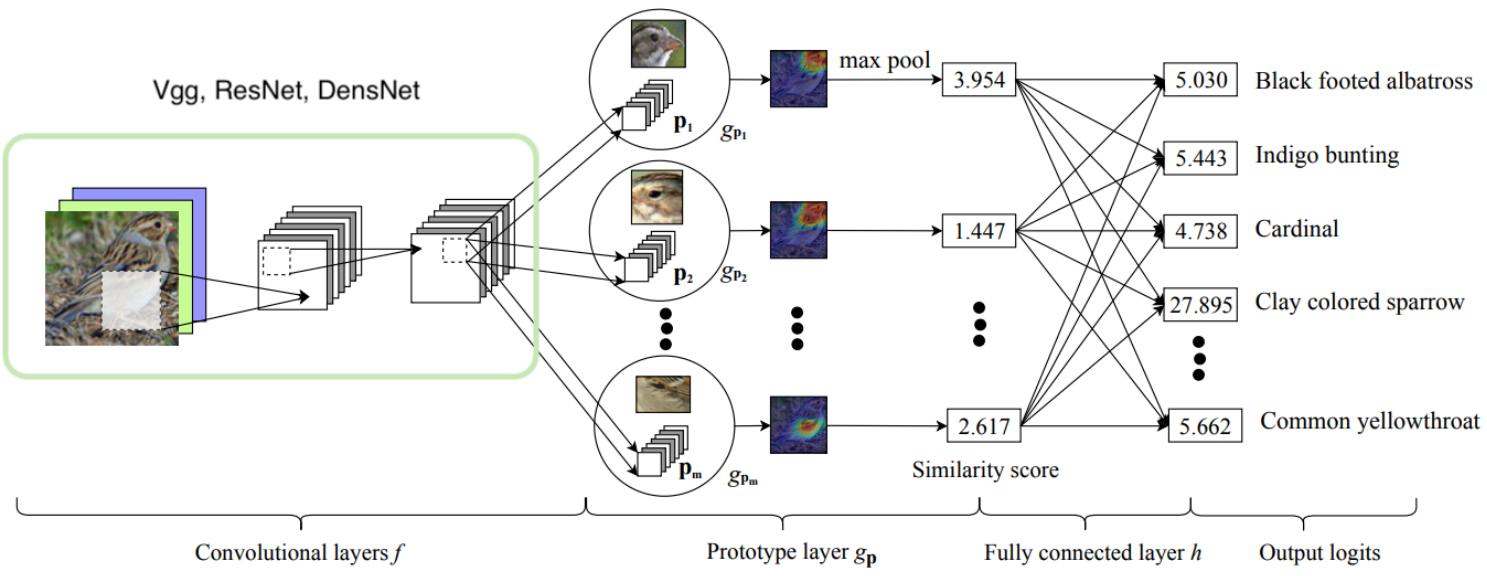
Proto P-Net

- Stop explaining a black box model $\longrightarrow$ create an interpretable model!
- Focus on **fine-grained image recognitionn** task
- How would you classify the bird in the picture as a clay colored sparrow?



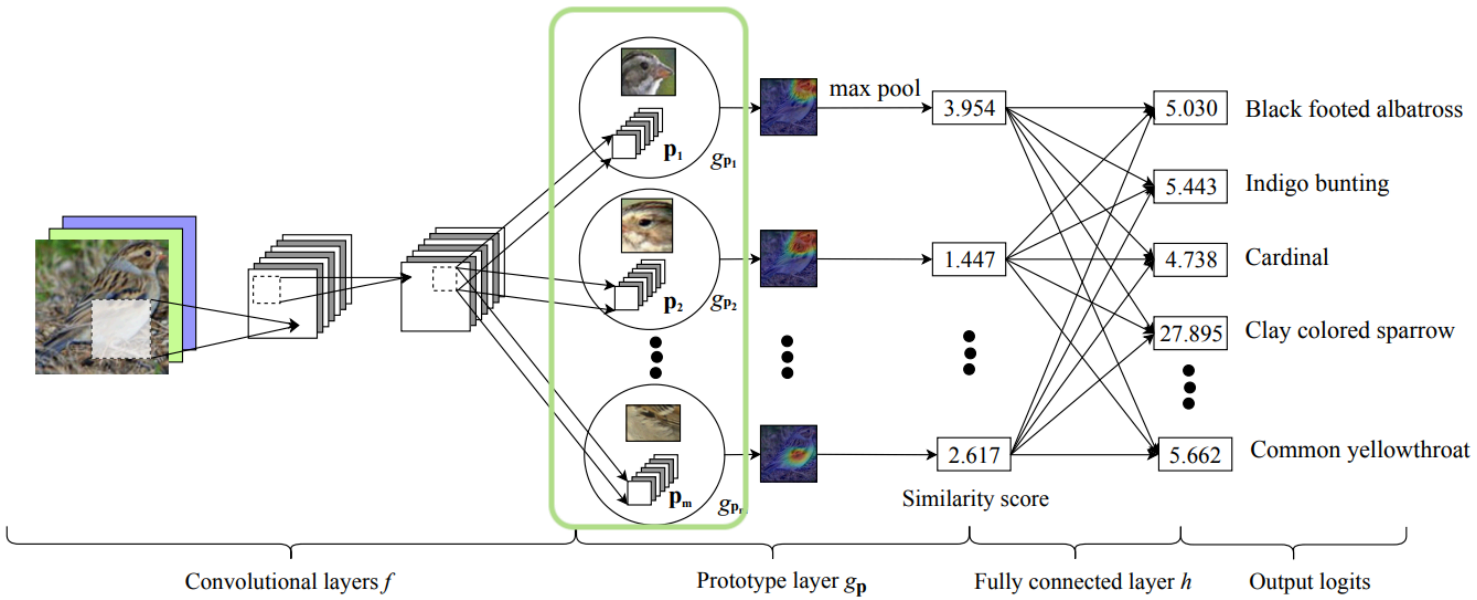
Proto P-Net

- A **regular CNN** with an additional 1x1 convolutional layer on top to extract a latent representation $\mathbf{z} \in \mathbb{R}^{HxWxD}$



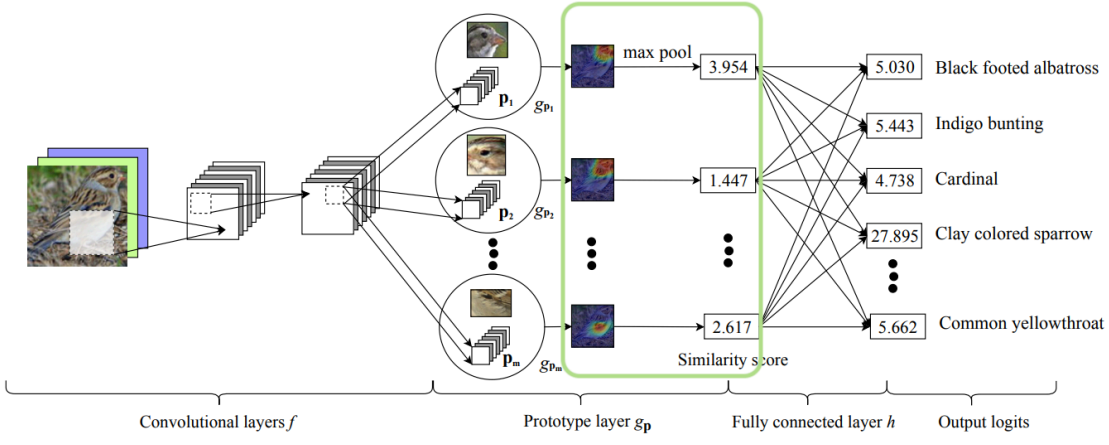
Proto P-Net

- A **prototype layer** g_p is added for interpretability.
- The network learns m prototypes per class of dimensions $H_1 x W_1 x D$, where D is the same dimension as the convolutional output
- A prototype can be seen as the latent representation of an image patch



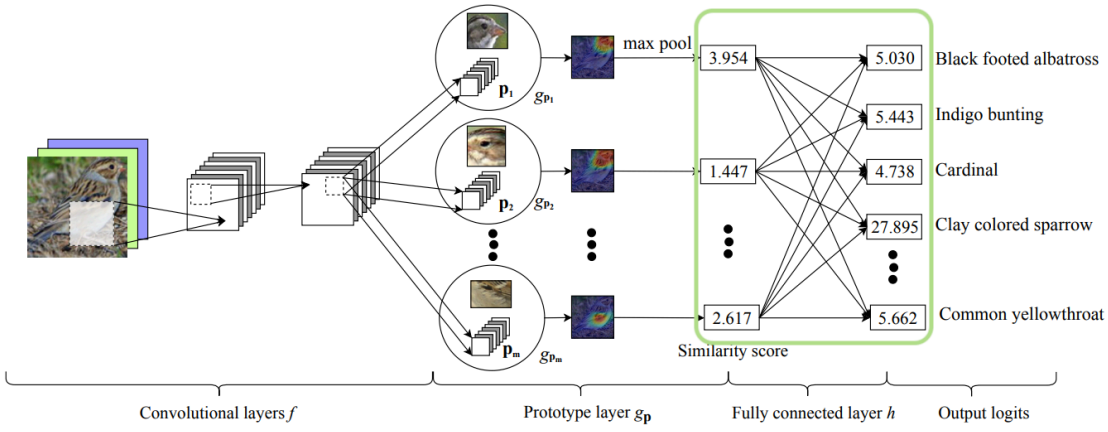
Proto P-Net

- Each unit g_{p_j} in the prototype layer computes the **L2 distances** between all latent patches and prototype $\mathbf{p}_j$ and inverts the distances in similarity scores
- An **activation map of similarity scores** is obtained for each prototype and it indicates how strong a prototypical part is present in the image
- The activation maps are reduced to a single similarity score by max pooling



Proto P-Net

- **A fully connected layer** multiplies each m similarity score with the weight matrix W_h to obtain the logits
- A final softmax layer normalize the logits to get the class probabilities



Training algorithm

The training of ProtoPNet is divided into three steps:

1. **Stochastic Gradient Descend** of layers before the last one
2. **Projection of prototypes**
3. **Convex optimization** of last layer

Training algorithm - 1

- Convolutional layers w_{conv} and prototypes $\mathbf{P} = \{\mathbf{p}_j\}_{j=1}^{m}$ are trained together by optimizing:

$$\min_{\mathbf{P}, w_{conv}} \frac{1}{n} \sum_{i=1}^{n} CrsEnt(h \circ g_{\mathbf{p}} \circ f(\mathbf{x}_i), \mathbf{y}_i) + \lambda_1 Clst + \lambda_2 Sep$$

Training algorithm - 1

- Convolutional layers w_{conv} and prototypes $\mathbf{P} = \{\mathbf{p}_j\}_{j=1}^{m}$ are trained together by optimizing:

$$\min_{\mathbf{P}, w_{conv}} \frac{1}{n} \sum_{i=1}^{n} CrsEnt(h \circ g_{\mathbf{p}} \circ f(\mathbf{x}_i), \mathbf{y}_i) + \lambda_1 Clst + \lambda_2 Sep$$

- **Cluster cost**: each training image should have a latent patch similar to at least one prototype of its own class

$$Clst = \frac{1}{n} \sum_{i=1}^{n} min_{j:\mathbf{p}_j \in \mathbf{P}_{y_i}} \min_{\mathbf{z} \in patches(f(\mathbf{x}_i))} ||\mathbf{z} - \mathbf{p}_j||_2^2$$

Training algorithm - 1

- Convolutional layers w_{conv} and prototypes $\mathbf{P} = \{\mathbf{p}_j\}_{j=1}^{m}$ are trained together by optimizing:

$$\min_{\mathbf{P}, w_{conv}} \frac{1}{n} \sum_{i=1}^{n} CrsEnt(h \circ g_{\mathbf{p}} \circ f(\mathbf{x_i}), \mathbf{y_i}) + \lambda_1 Clst + \lambda_2 Sep$$

- **Cluster cost**:

$$Clst = \frac{1}{n} \sum_{i=1}^{n} min_{j:\mathbf{p}_j \in \mathbf{P}_{y_i}} \min_{\mathbf{z} \in patches(f(\mathbf{x}_i))} ||\mathbf{z} - \mathbf{p}_j||_2^2$$

- **Separation cost**: each training image should have all latent patches away from the prototypes of the other classes

$$Sep = -\frac{1}{n} \sum_{i=1}^{n} min_{j:\mathbf{p}_j \notin \mathbf{P}_{y_i}} \min_{\mathbf{z} \in patches(f(\mathbf{x}_i))} ||\mathbf{z} - \mathbf{p}_j||_2^2$$

Training algorithm - 1

- Convolutional layers w_{conv} and prototypes $\mathbf{P} = \{\mathbf{p}_j\}_{j=1}^{m}$ are trained together by optimizing:

$$\min_{\mathbf{P}, w_{conv}} \frac{1}{n} \sum_{i=1}^{n} CrsEnt(h \circ g_{\mathbf{p}} \circ f(\mathbf{x_i}), \mathbf{y_i}) + \lambda_1 Clst + \lambda_2 Sep$$

- **Cluster cost**:

$$Clst = \frac{1}{n} \sum_{i=1}^{n} min_{j:\mathbf{p}_j \in \mathbf{P}_{y_i}} \min_{\mathbf{z} \in patches(f(\mathbf{x}_i))} ||\mathbf{z} - \mathbf{p}_j||_2^2$$

- **Separation cost**:

$$Sep = -\frac{1}{n} \sum_{i=1}^{n} min_{j:\mathbf{p}_j \notin \mathbf{P}_{y_i}} \min_{\mathbf{z} \in patches(f(\mathbf{x}_i))} ||\mathbf{z} - \mathbf{p}_j||_2^2$$

- The last layer w_h is fixed:

$$w_h^{(k,j)} = 1 \; \forall \; j \; with \; \mathbf{p}_j \in \mathbf{P}_k \; and \; w_h^{(k,j)} = -0.5 \; \forall \; j \; with \; \mathbf{p}_j \notin \mathbf{P}_k$$

Training algorithm - 2

- Each prototype is projected into the nearest latent training patch with the following update step:

$$\mathbf{p}_j \leftarrow argmin_{\mathbf{z} \in Z_j} ||\mathbf{z} - \mathbf{p}_j||_2$$

- A prototype can be visualized as a **training patch**
- This prototype projection step doesn't change the prediction accuracy of ProtoPNet and it is done every 10 epochs during the training

Training algorithm -3

- Last linear classifier optimization:

$$\min_{w_h} \frac{1}{n} \sum_{i=1}^{n} CrsEnt(h \circ g_{\mathbf{p}} \circ f(\mathbf{x}_i), \mathbf{y}_i) + \lambda \sum_{k=1}^{K} \sum_{j:\mathbf{P}_j \notin \mathbf{P}_k} |w_h^{(k,j)}|$$

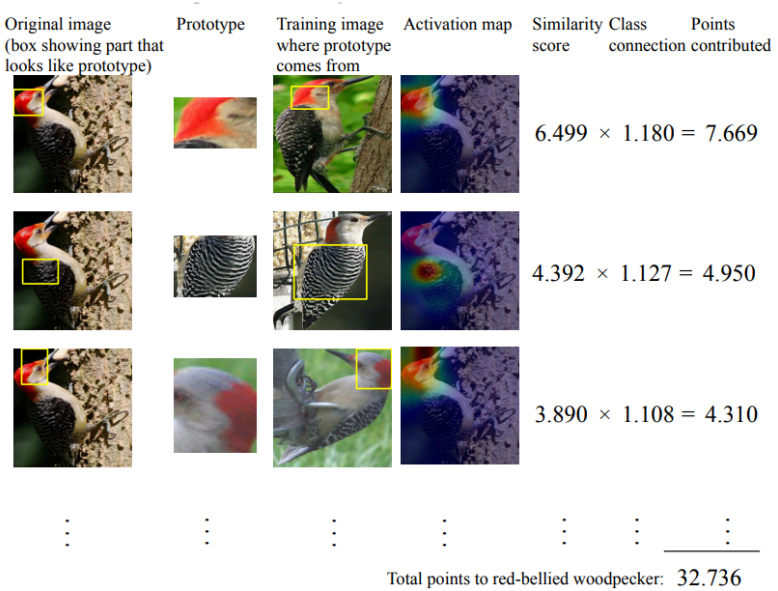
- Parameters from the convolutional layers and prototypes are fixed
- For each class, a L_1 **regularization term** penalizes the prototypes that don't belong to the class.
- The model relies less on "negative" reasoning process thanks to the sparsity property:

$$w_h^{(k,j)} \approx 0 \; for \; k \; and \; j \; with \; \mathbf{p}_j \notin \mathbf{P}_k$$

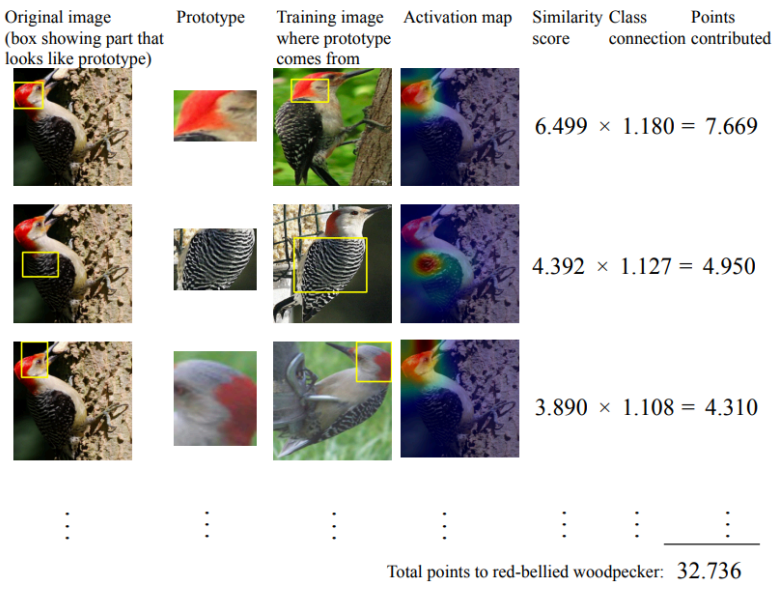
Reasoning process

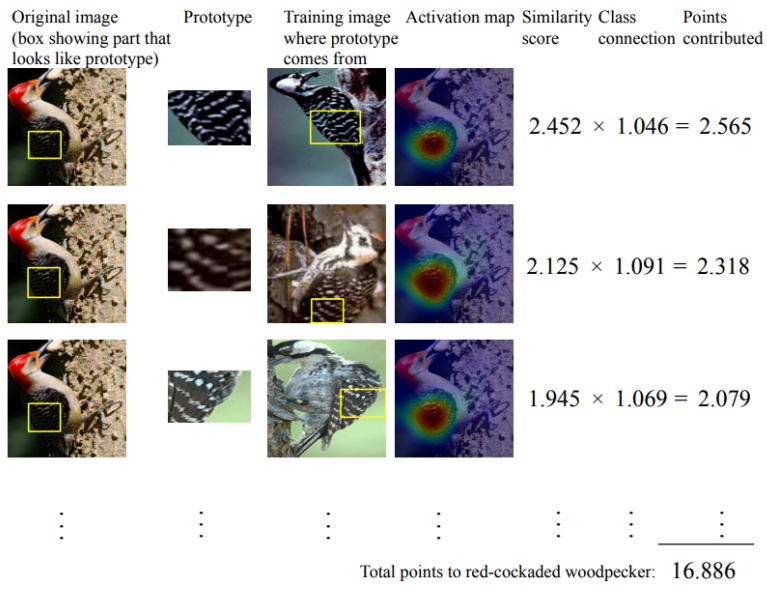
- Why is this bird classified as a red-bellied woodpecker?

Reasoning process



Reasoning process





Results

- Loss of accuracy at most 3.5%

Base	ProtoPNet	Baseline	Base	ProtoPNet	Baseline
VGG16	76.1 ± 0.2	74.6 ± 0.2	VGG19	78.0 ± 0.2	75.1 ± 0.4
Res34	79.2 ± 0.1	82.3 ± 0.3	Res152	78.0 ± 0.3	81.5 ± 0.4
Dense121	80.2 ± 0.2	80.5 ± 0.1	Dense161	80.1 ± 0.3	82.2 ± 0.2

Results

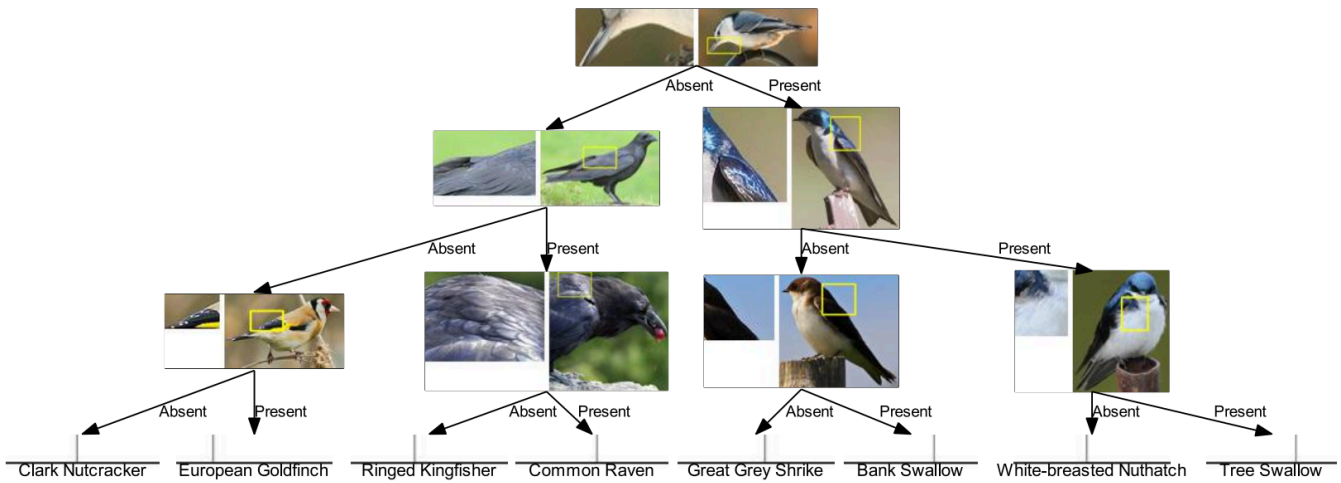
- Loss of accuracy at most 3.5%
- Accuracy can also be improved by combining multiple ProtoPNet models
- Comparison with part-level attentions methods

Interpretability	Model: accuracy
None	**B-CNN**[25]: 85.1 (bb), 84.1 (full)
Object-level attn.	**CAM**[56]: 70.5 (bb), 63.0 (full)
Part-level attention	**Part R-CNN**[53]: 76.4 (bb+anno.); **PS-CNN** [15]: 76.2 (bb+anno.); **PN-CNN** [3]: 85.4 (bb+anno.); **DeepLAC**[24]: 80.3 (anno.); **SPDA-CNN**[52]: 85.1 (bb+anno.); **PA-CNN**[19]: 82.8 (bb); **MG-CNN**[46]: 83.0 (bb), 81.7 (full); **ST-CNN**[16]: 84.1 (full); **2-level attn.**[49]: 77.9 (full); **FCAN**[26]: 82.0 (full); **Neural const.**[37]: 81.0 (full); **MA-CNN**[55]: 86.5 (full); **RA-CNN**[7]: 85.3 (full)
Part-level attn. + prototypical cases	**ProtoPNet** (ours): 80.8 (full, VGG19+Dense121+Dense161-based) 84.8 (bb, VGG19+ResNet34+DenseNet121-based)

Neural Prototype Tree

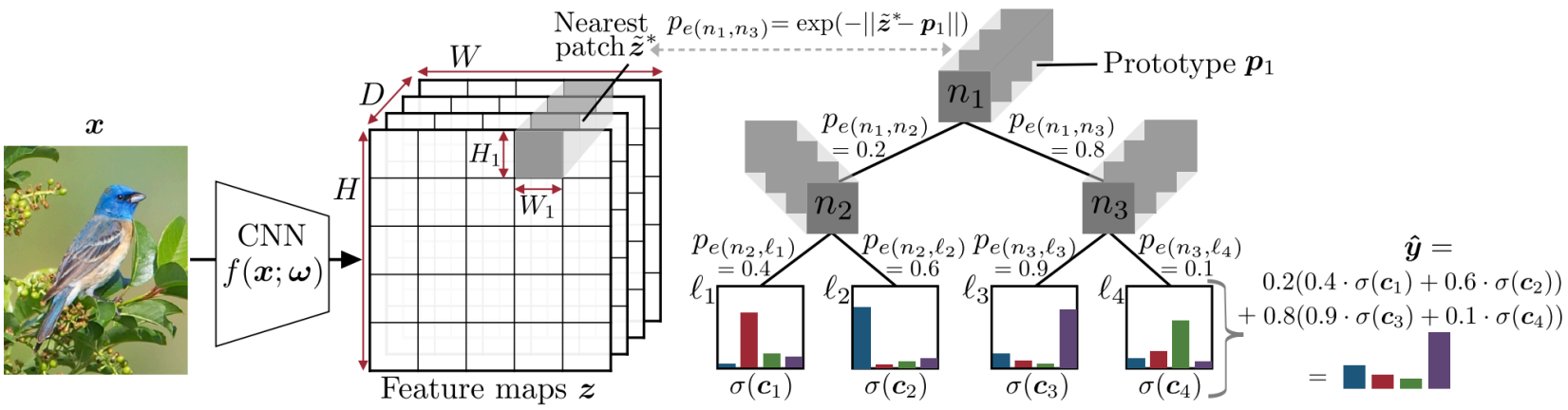
Proto Tree [5]

- A *Neural Prototype Tree* (ProtoTree) combines **prototypes learning** and a **decision binary tree**
- A single prediction is explained by outlining a decision path through the tree (**global explanation**)
- ”Guess who?” game: binary questions to underline visual properties to predict the class of the input image
- ▸ https://github.com/aai-institute/ProtoTree/tree/main



Architecture

- A CNN extracts a feature maps **z** of dimension *HxWxD*
- This latent representation **z** is the input to a ***soft* binary tree** (both children of a node are always visited)

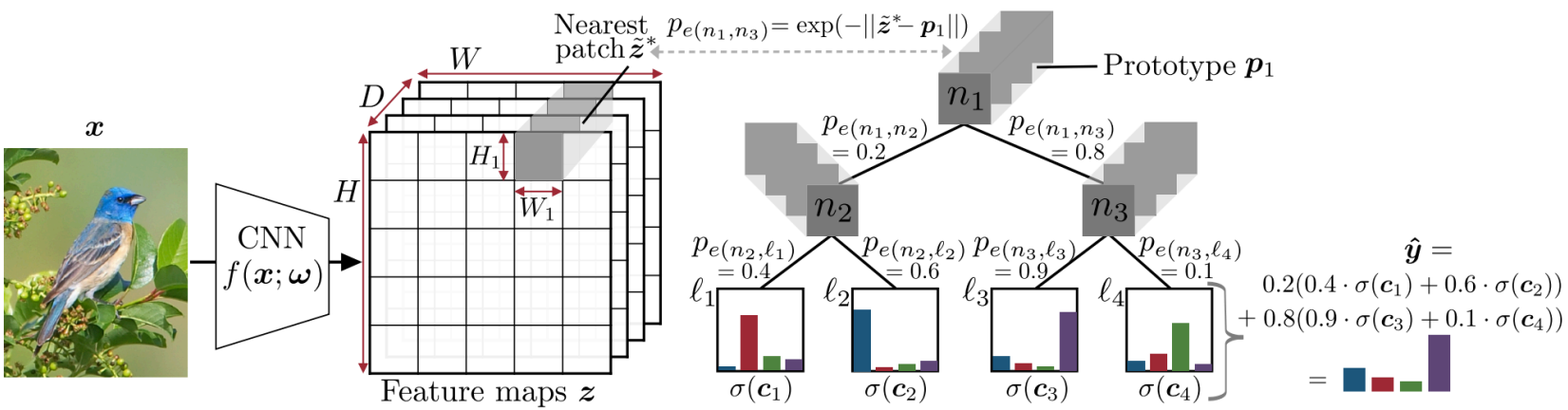


Architecture

- Each **node** of the tree is a **trainable prototype** $\mathbf{p}_n$ of dimension $H_1 x W_1 x D$
- The L_2 **distance** is computed between each prototype and all the patches $\tilde{\mathbf{z}}$ of the latent representation.
- Min pooling operation to select the closest latent patch $\tilde{\mathbf{z}}^*$
- $\mathbf{z}$ is routed through both children with probabilities:

$$p_{e(n,n.right)}(z) = exp(-||\tilde{\mathbf{z}}^* - \mathbf{p}_n||)$$

$$p_{e(n,n_left)} = 1 - p_{e(n,n.right)}$$



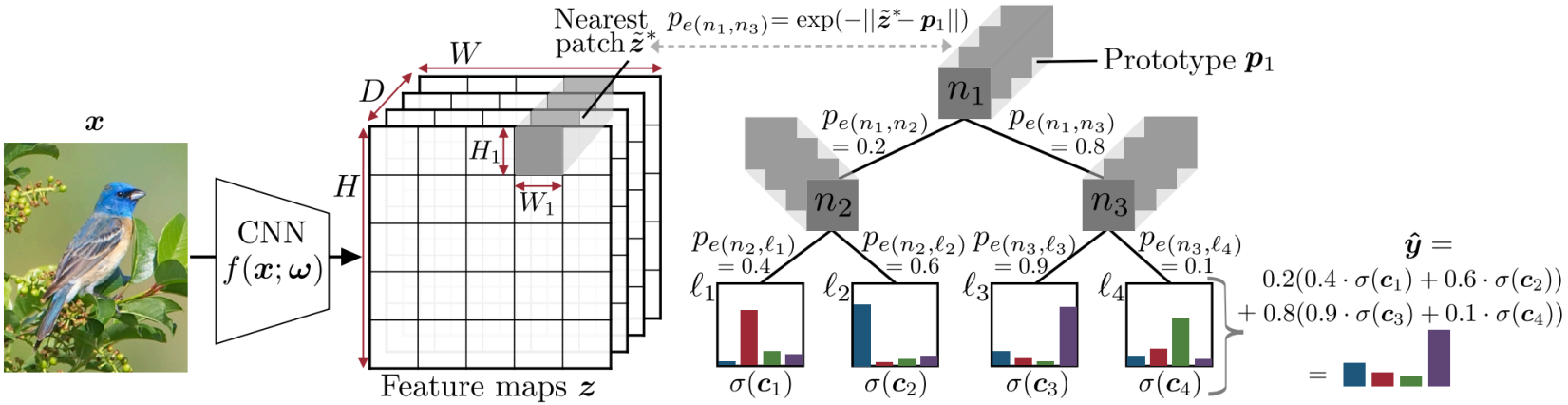
Architecture

- Each leaf is reached with probability equals to the product of probabilities of the edges of the followed path:

$$\pi_l(z) = \Pi_{e \in P_l} p_e((z))$$

- A sigmoid function is applied to the trainable parameter of each leaf to get the final class distribution:

$$\hat{y}(x) = \sum_{l \in L} \sigma(c_l) \cdot \pi(f(x; w))$$



Proto Tree Training

The training of a ProtoTree follows these steps:

- Prototypes $\mathbf{P}$ and parameters w of the CNN are trained together by minimizing the **cross entropy loss** between the class probability distribution $\tilde{\mathbf{y}}$ and the ground-truth $\mathbf{y}$
- $\mathbf{P}$ and w are then updated with gradient descent
- We update leaves distribution with a **derivative-free strategy**:

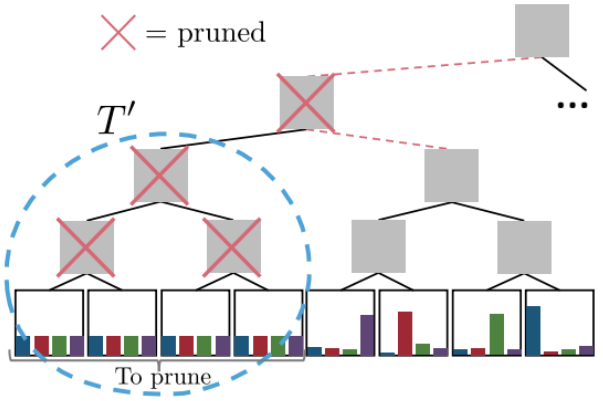
$$\mathbf{c}_l^{(t+1)} = \sum_{\mathbf{x},\mathbf{y} \in T} (\sigma(\mathbf{c}_l^{(t)}) \odot \mathbf{y} \odot \pi_l) \oslash \hat{\mathbf{y}}$$

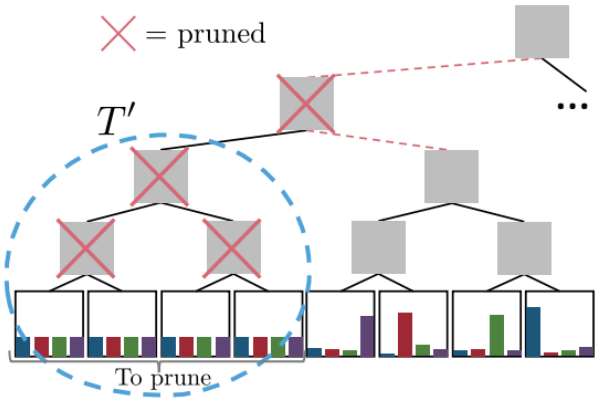
Pruning

- **Pruning operation** to reduce the number of prototypes and augment interpretability
- All leaves with a distribution similar to a uniform one are removed

$$max(\sigma(\mathbf{c}_l)) \leq \tau$$

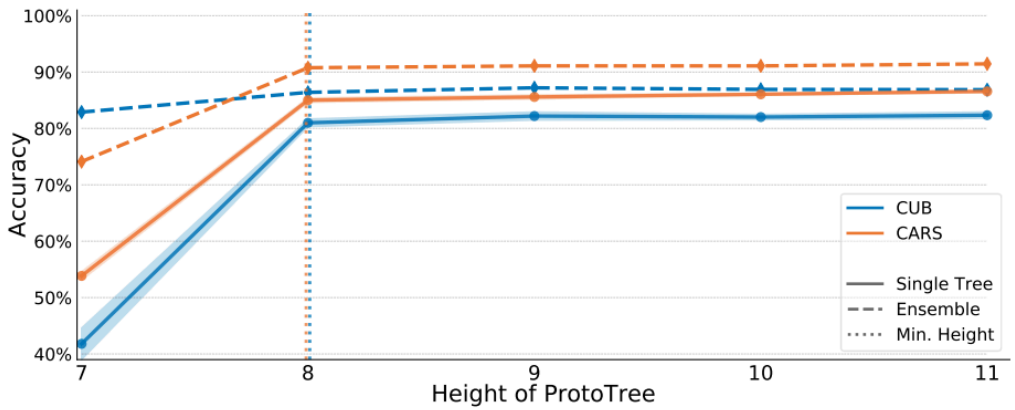
- If all leaves of a subtree are pruned, then the subtree is removed
- The tree is then reorganized by removing the superflous parent of the removed subtree





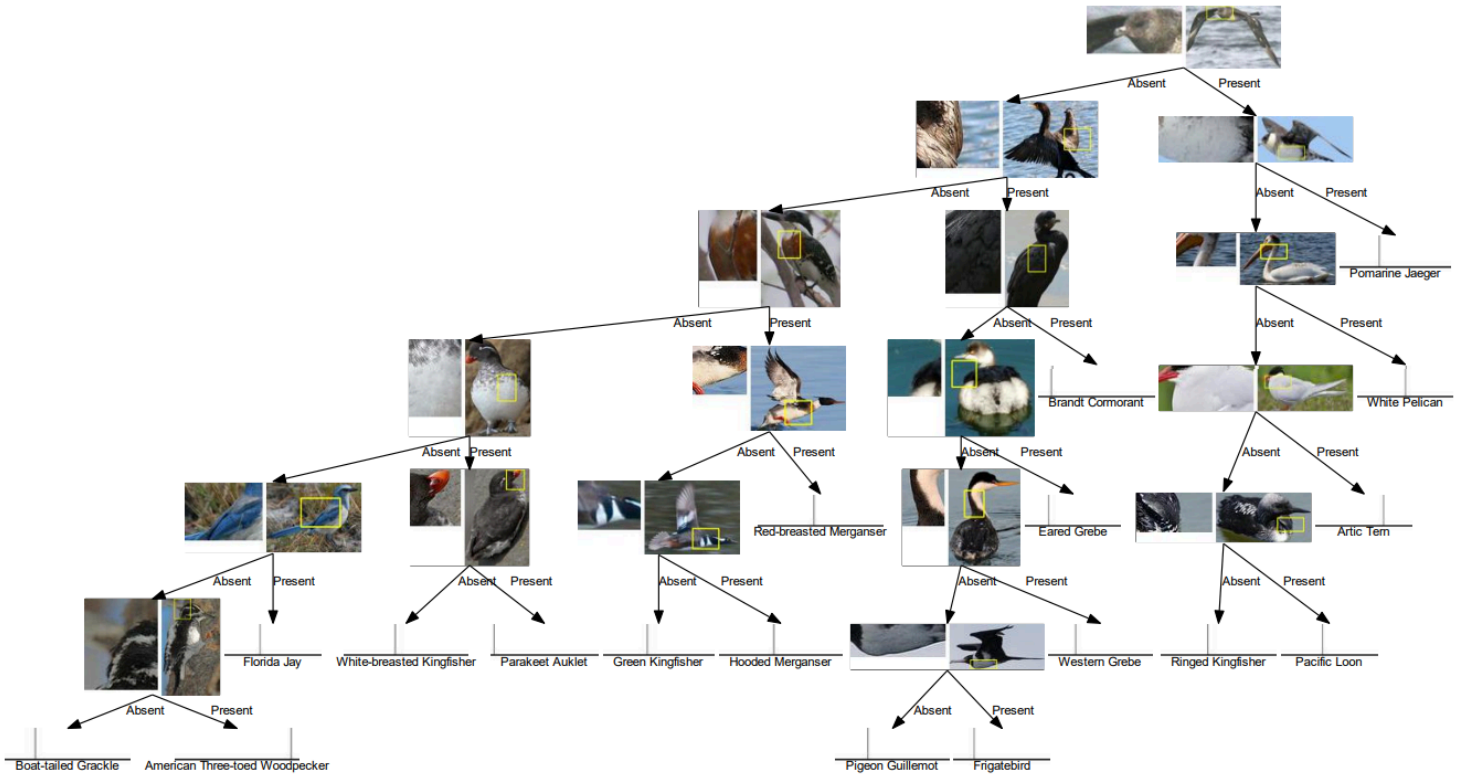
ProtoTree Height

- A model with fewer prototypes is easier to interprete but represent a less complex model with less predictive power
- Need to select a right value for *h*, the height of the tree
- The initial value for *h* such that the number of leaves is at least as large as the number of classes.



Reasoning

- Visualization of a subtree of a ProtoTree
- For each node: the prototype and the image from which the prototype is extracted

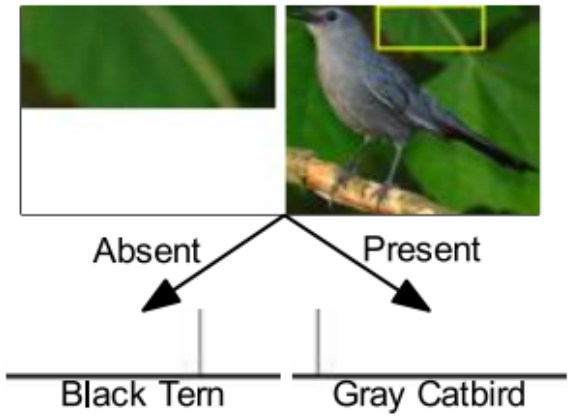


Results

- Higher accuracy then ProtoPNet
- ProtoTree is almost 90% smaller (1 prototype per class against 10 of ProtoPNet) and easier to interpret

Data set	Method	Inter-pret.	Top-1 Accuracy	#Proto types
CUB (224×224)	Triplet Model [34]	-	**87.5**	n.a.
	TranSlider [58]	-	85.8	n.a.
	TASN [57]	o	87.0	n.a.
	ProtoPNet [9]	+	79.2	2000
	ProtoTree h=9 (ours)	++	82.2±0.7	**202**
	ProtoPNet ens. (3) [9]	+	84.8	6000
	ProtoTree ens. (3)	+	86.6	605
	ProtoTree ens. (5)	+	**87.2**	1008

- ProtoTree reveals biases learned by the model, where some prototypes focus on the background.



Shortcomings

Looks like that, does it?

- For a human-interpretable model this statement must hold:

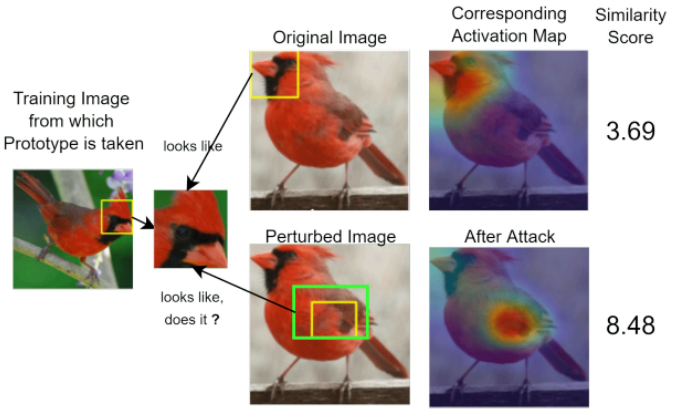
  > Two image patches look similar to a ProtoPNet $\iff$ Two image patches look similar to a human

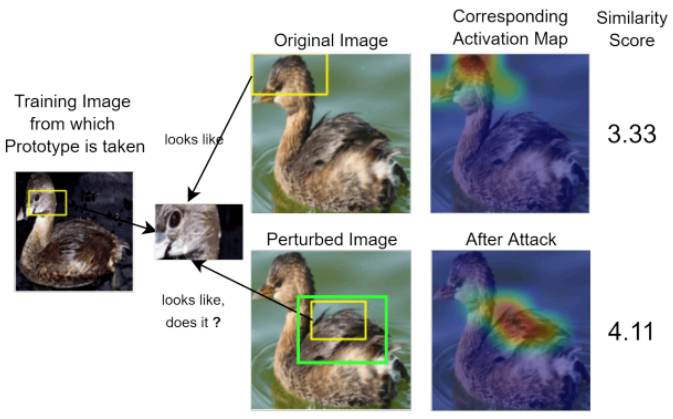
- **Semantic gap** between image space and latent space

Looks like that, Does it? [2]

Head on Stomach experiment

- One of the most activated prototype $\mathbf{p}_l$ is taken into consideration
- The image is perturbed (with very small noise) such that the network finds $\mathbf{p}_l$ in a different location with **high similarity** in a **non-sensical place**

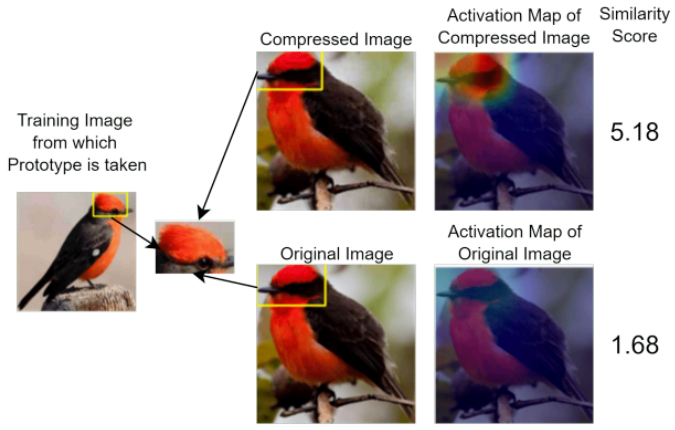


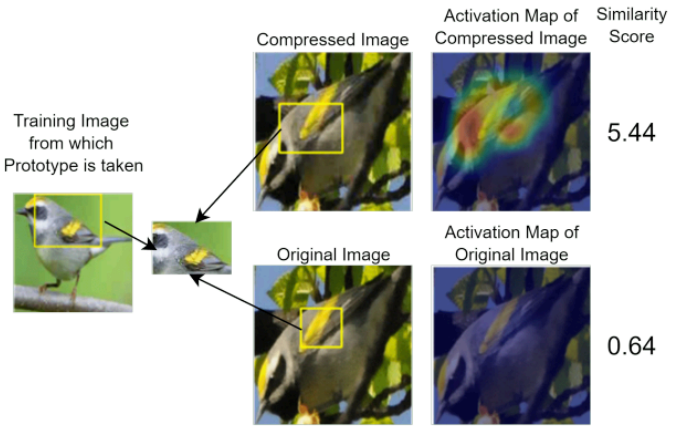


Looks like that, Does it? [2]

JPEG compression experiment

- Training images of half of the classes are compressed
- The similarity score of the compressed version of an image ($\bar{x}$) from a compressed class that is classified correctly by ProtoPNet is recorded
- This score is then compared with the one obtained by passing the original image (x) to ProtoPNet
- If the scores are very different, it means ProtoPNet doesn't consider x and $\bar{x}$ similar as humans would



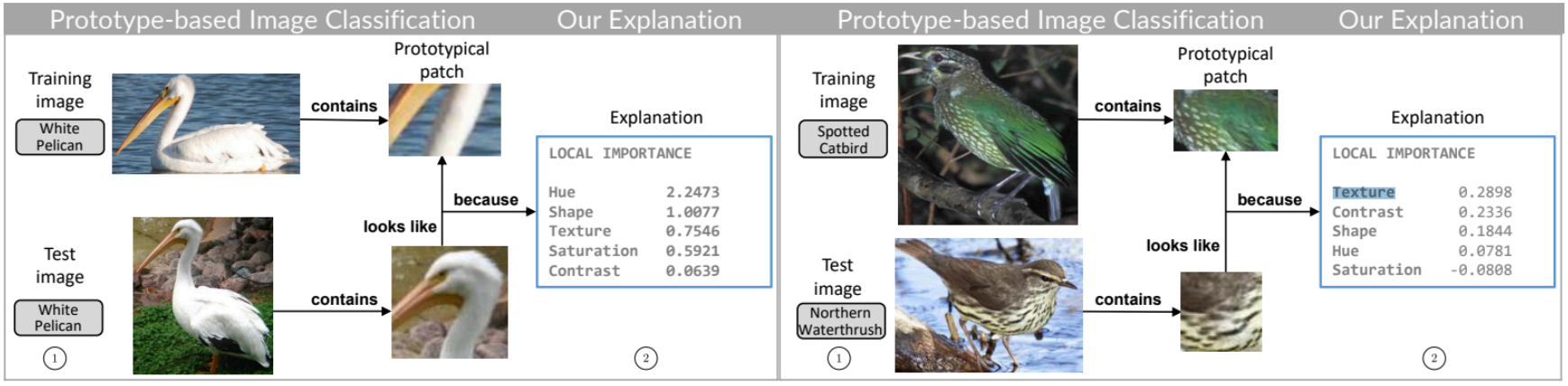


Looks like that, Because... [4]

- Prototypes alone are not enough, they need to be **explained**
- Visual textual information is added to each prototype regarding visual features:

> **hue**, **contrast**, **saturation**, **shape**, **texture**

- Useful to understand misleading prototypes and redundancy



Looks like that, Because... [4]

- A set of modified images is created for each visual feature
- **Local score** for test image k, prototype j and transformation i:

$$\phi_{local}^{i,j,k} = g_{j,k} - \hat{g}_{i,j,k}$$

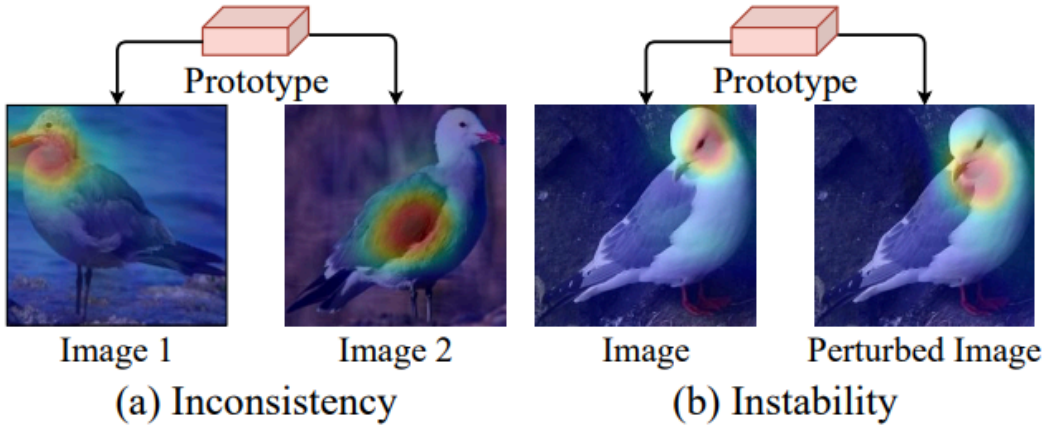
- **Global score** for a prototype j and transformation i (obtained from the training dataset)

$$\phi_{global}^{i,j} = \frac{\sum_{k=1}^{|S_{train}|} \phi_{local}^{i,j,k} \cdot g_{j,k}}{\sum_{k=1}^{|S_{train}|} g_{j,k}}$$

Recent improvement

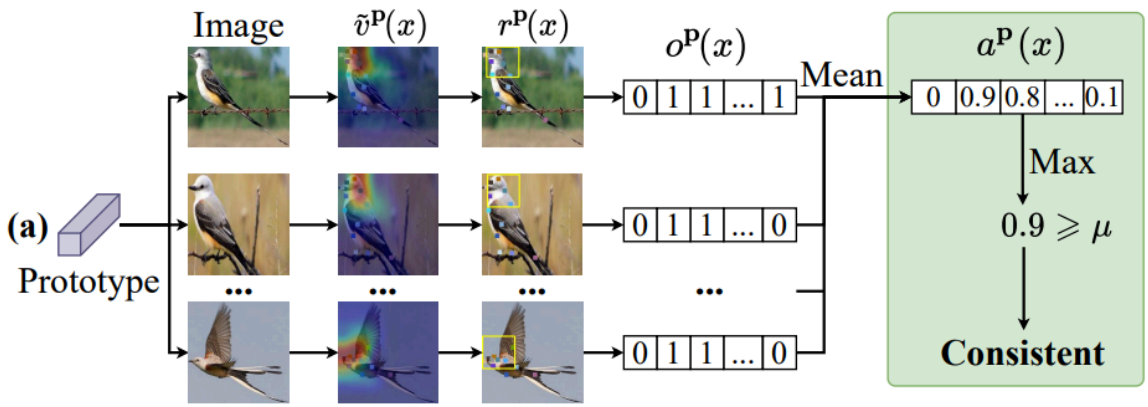
Improvement of Interpretability with SA and SDFA [3]

- Need of a benchmark to quantitatively evaluate the interpretability of prototypes
- Evalutation metrics based on two problems:
  - inconsistency: a prototype may mistakenly correspond to different object parts in different images
  - instability: a prototype may mistakenly correspond to different object parts in the original image and the slightly perturbed image



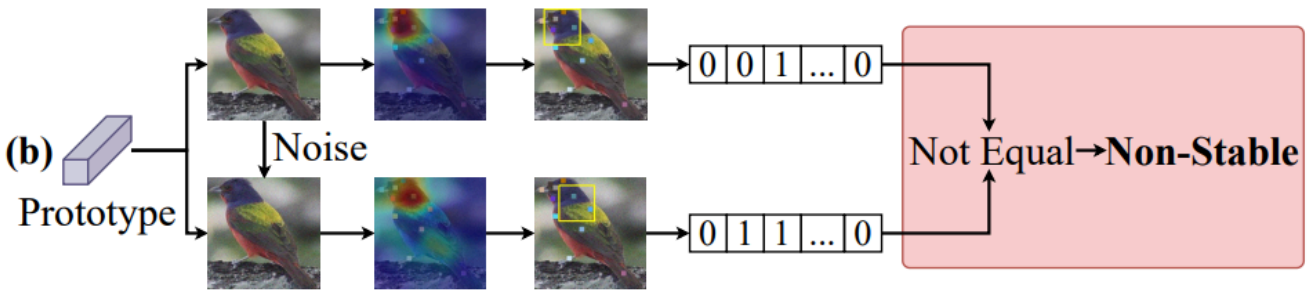
Improvement of Interpretability with SA and SDFA [3]

- The dataset needs to have object parts annotations
- **Consistency score**:
  - Consider a prototype of class k and all the test images of that class
  - For each image, compute a binary vector: 1 if an object part is inside the prototype activation region, 0 otherwise
  - Take the average of all these binary vectors
  - If the maximum element is higher than a threshold the prototype is consistent
  - Take the ratio between the number of consistent prototypes and the total number of prototypes as the consistency score



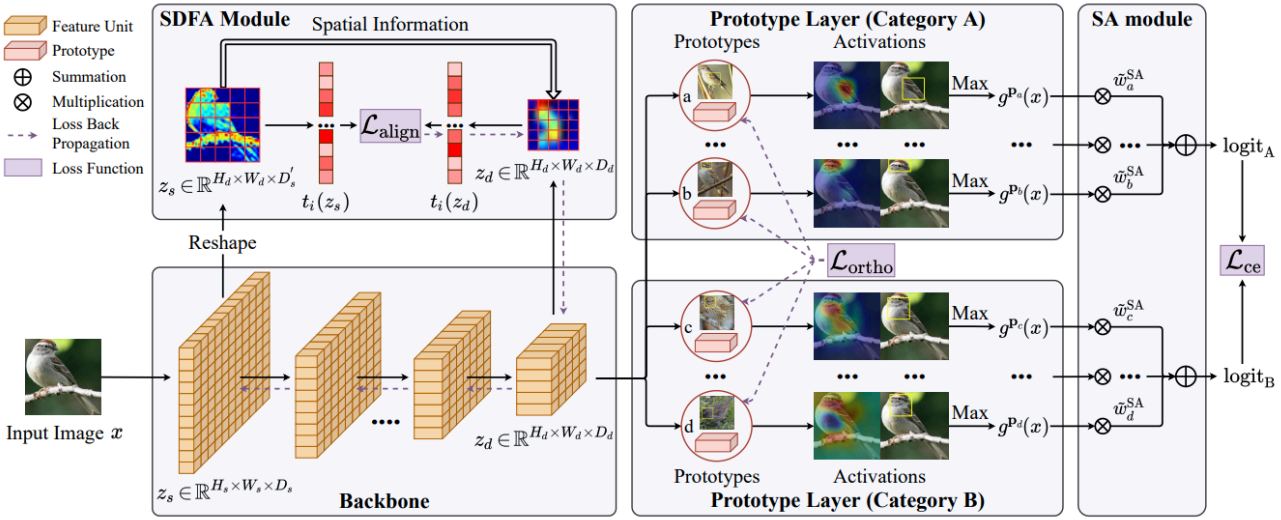
Improvement of Interpretability with SA and SDFA [3]

- **Stability score**:
  - Consider a prototype of class k and all the test images of that class
  - For each image, compute the binary vector for the original version and the perturbeted one
  - If these binary vector are equals, the prototype is stable
  - Take the ratio between the number of stable prototypes and the total number of prototypes as the stability score



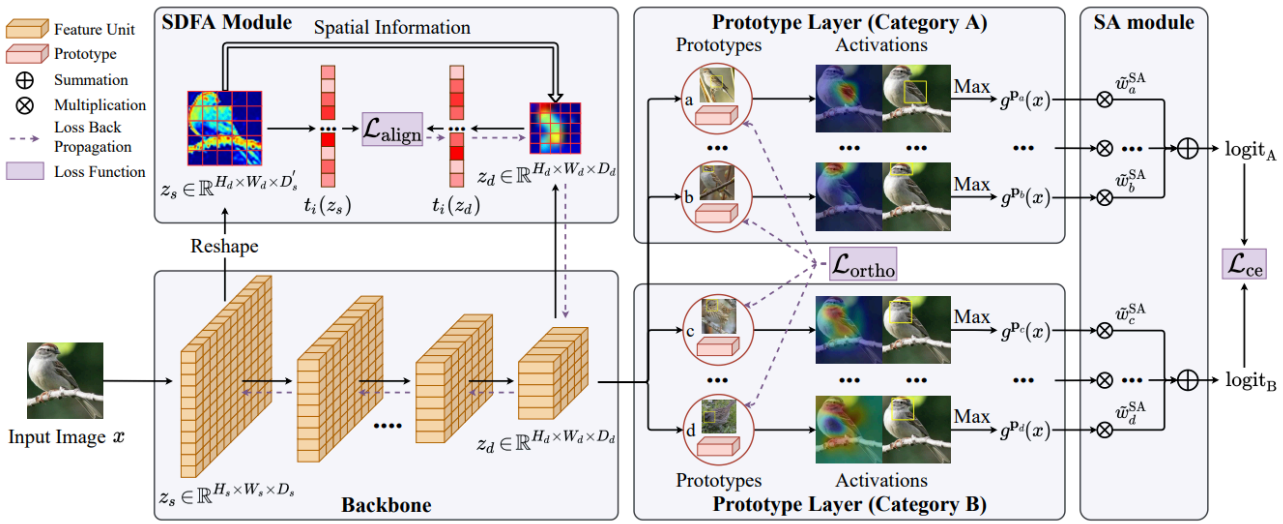
Improvement of Interpretability with SA and SDFA [3]

- **Shallow Deep Features Alignement** (SDFA) module: incorporated **spatial information** from shallow layers is in the deep feature maps
- Similarity structure $t \in \mathbb{R}^{HWxHW}$ of a feature map $z \in \mathbb{R}^{HxWxD}$ can be used to compare two representations
- A loss term is added during the training to enforce these spatial structures to be similar



Improvement of Interpretability with SA and SDFA [3]

- **Score Aggregation** (SA) module: the activation values of prototypes are allocated into their categories
- Because of the last fully connected layer of ProtoPNet, the classification score depends also on the prototypes of other classes
- The similarity scores of the prototypes are aggregated into their classes
- A learnable layer adjust the importance of each prototypes for each class prototypes.



Conclusions

- Start using interpretable models for high stake decisions instead of black boxes!
- Prototypical learning gives a form of interpretability, ***this part looks like that!***
- There still are some shortcomings
- Recent improvement of ProtoPNet and first attempt to create a benchmark for interpretability

Thank you!

References

C. Chen, O. Li, C. Tao, A. J. Barnett, J. Su, and C. Rudin.
This looks like *That*: Deep learning for interpretable image recognition.
NeurIPS, 2019.

A. Hoffmann, C. Fanconi, R. Rade, and J. Kohler.
This looks like that... does it? shortcomings of latent space prototype interpretability in deep networks.
ICMLW, 2021.

Q. Huang, M. Xue, W. Huang, H. Zhang, J. Song, Y. Jing, and M. Song.
Evaluation and improvement of interpretability for self-explainable part-prototype networks.
ArXiv, 2023.

M. Nauta, A. J. J. Provoost, and C. Seifert.
This looks like that, because ... explaining prototypes for interpretable image recognition.
XKDD, 2021.

M. Nauta, R. van Bree, and C. Seifert.
Neural prototype trees for interpretable fine-grained image recognition.
CVPR, 2021.

C. Rudin.
Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead.
Nature, 2019.