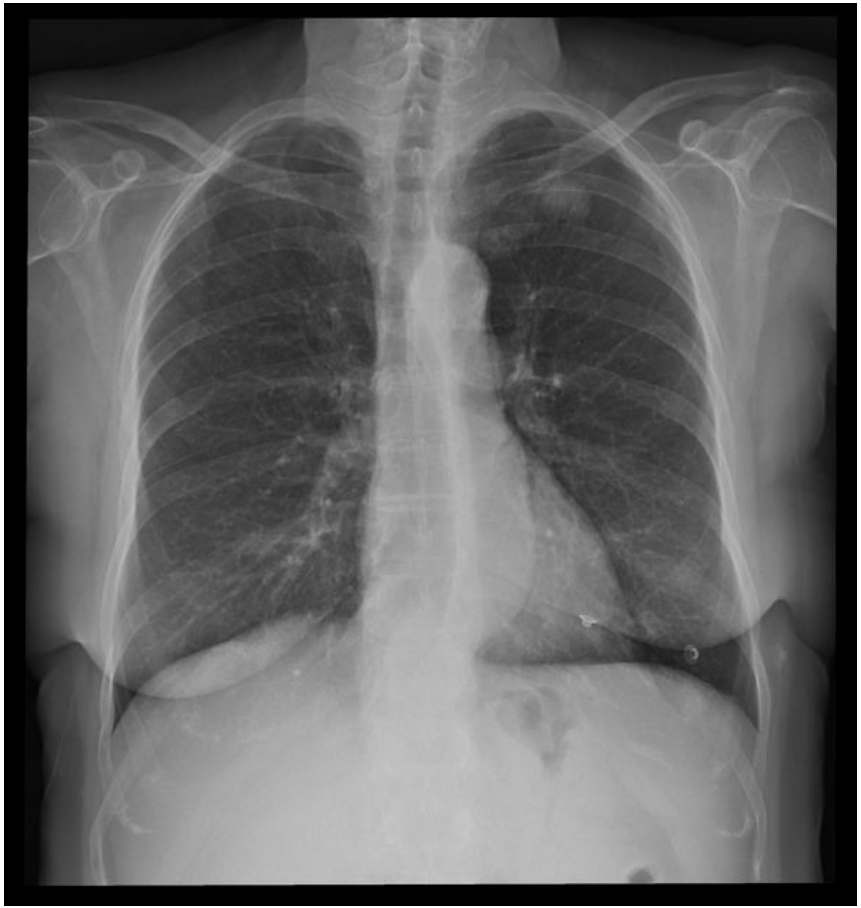
This Looks Like That: Deep Learning for Interpretable Image Recognition

David Demitri Africa
L193: Explainable AI

How would a radiologist look for tumors in a lung x-ray?



You would **not**:

➔ Compare it to every lung in existence

You would **probably**:

➔ Look at a textbook

How would you tell which person will do better in the half marathon?

Enter: the **Prototypical Part Network** (not to be confused with Prototypical Networks which are different)

A Prototypical Part Network (ProtoPNet):

- ➔ Learns representative prototypes of each class
- ➔ Extracts features from an image and compares patches to each learned prototype
- ➔ Calculates similarity and votes

How would we build this?

First, take a standard CNN.

Given an input image **x**, the CNN produces a latent feature representation **f(x)** of size **H × W × D**.

Second, add a prototype layer.

Add **m** learnable prototypes

$$P = \{ p_j \}_{j=1}^{m}$$

where each prototype has dimensions

$\mathbf{H_1 \times W_1 \times D}$

Second, add a prototype layer.

For each prototype p_j and patch z (a subtensor from f(x) with same size as p_j), the network computes the squared L_2-distance.

$$d\left(\tilde{z}, p_j\right) = ||\tilde{z} - p_j||_2^2$$

Second, add a prototype layer.

We then invert the distances to get a similarity score for each patch…

$$g_{p_j}(\tilde{z}) = \log\left(\frac{d(\tilde{z}, p_j) + 1}{d(\tilde{z}, p_j) + \epsilon}\right)$$

Second, add a prototype layer.

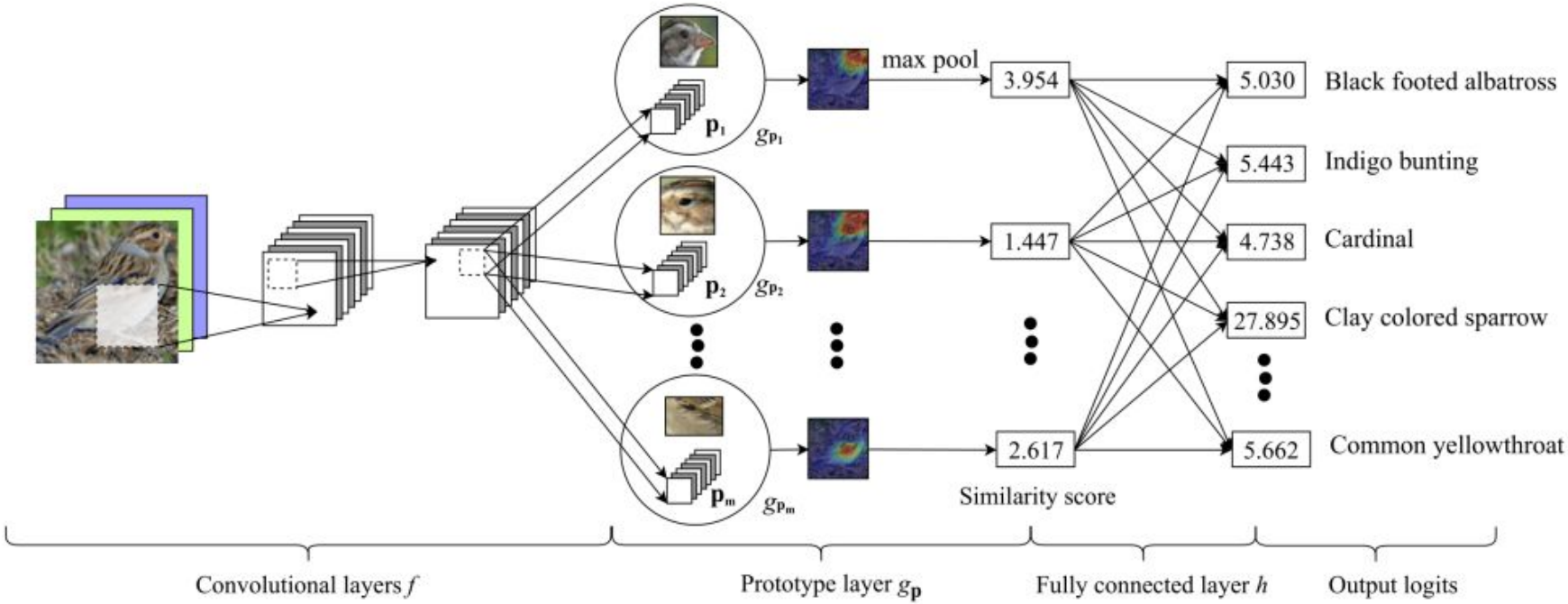
…and reduce it for global max pooling.

$$s_j(x) = \max_{\tilde{z} \in \text{patches}(f(x))} g_{p_j}(\tilde{z})$$

Third, feed it into a fully connected layer…

…to produce logits which we softmax over to get predictions.

$$\text{logits}\,(x) \;=\; W_h \;\cdot\; [s_1\,(x), s_2\,(x), \ldots, s_m\,(x)]^\top$$


max pool
3.954
1.447
2.617
5.030
5.443
4.738
27.895
5.662
Black footed albatross
Indigo bunting
Cardinal
Clay colored sparrow
Common yellowthroat
Similarity score
Convolutional layers f
Prototype layer g_p
Fully connected layer h
Output logits

Okay, how would we train this?

We do SGD on:

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

where

We do SGD on:

where

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

For the last layer, we initialize weights as:

For each class **k,** prototype $\mathbf{p}_j$ assigned to class **k** is given a positive weight while prototypes from other classes are given a negative weight.

For the last layer, we initialize weights as:

Then we do convex optimization to guarantee sparsity

One concern!

Projection of prototypes: To be able to visualize the prototypes as training image patches, we project ("push") each prototype $\mathbf{p}_j$ onto the nearest latent training patch from the *same* class as that of $\mathbf{p}_j$. In this way, we can conceptually equate each prototype with a training image patch. (Section 2.3 discusses how we visualize the projected prototypes.) Mathematically, for prototype $\mathbf{p}_j$ of class k, i.e., $\mathbf{p}_j \in \mathbf{P}_k$, we perform the following update:

$$\mathbf{p}_j \leftarrow \arg\min_{\mathbf{z} \in \mathcal{Z}_j} \|\mathbf{z} - \mathbf{p}_j\|_2, \text{where } \mathcal{Z}_j = \{\tilde{\mathbf{z}} : \tilde{\mathbf{z}} \in \text{patches}(f(\mathbf{x}_i)) \; \forall i \text{ s.t. } y_i = k\}.$$

Sketch of the theorem:

If prototype projection does not move the prototypes by much, the prediction does not change for examples that the model predicted correctly with some confidence before the projection.

Sketch of the proof:

➔ Our activation function is monotonic with respect to distance so a bounded change in distance is bounded in similarity score

Sketch of the proof:

- ➔ We already fine tuned our last layer to have weight 1 for correct class prototypes and 0 else
- ➔ If there are m prototypes per class, then the total change in logits is m times the original prototype change which we call Δ_{max}

Sketch of the proof:

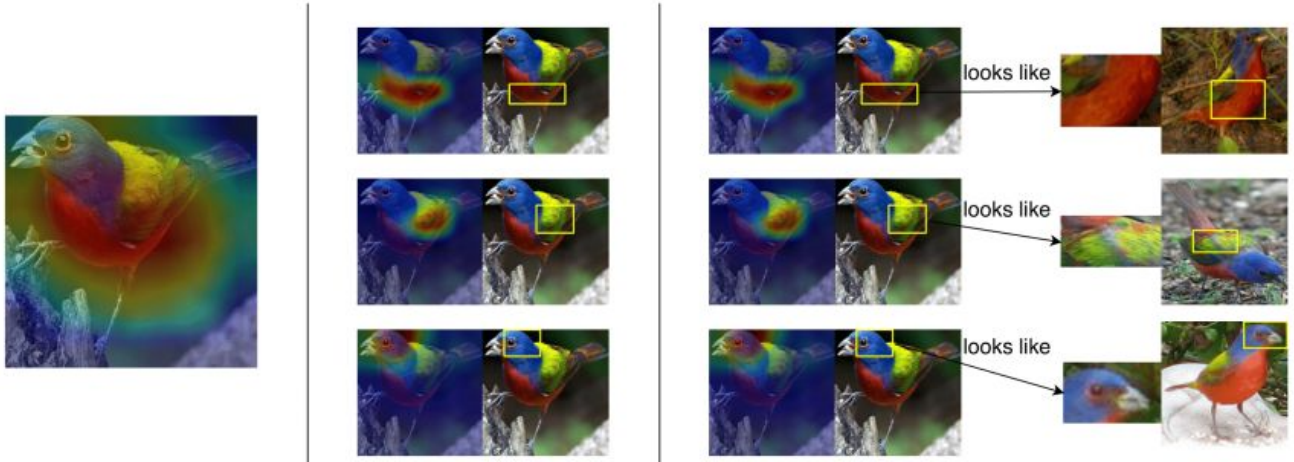
➔ If the difference between the logit of the correct class and that of the closest incorrect class is greater than $2\Delta_{max}$ then our prediction does not change

Results

Table 1: Top: Accuracy comparison on cropped bird images of CUB-200-2011
Bottom: Comparison of our model with other deep models

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

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)



(a) Object attention (class activation map)

(b) Part attention (attention-based models)

(c) Part attention + comparison with learned prototypical parts (our model)

Figure 4: Visual comparison of different types of model interpretability: (a) object-level attention map (e.g., class activation map [56]); (b) part attention (provided by attention-based interpretable models); and (c) part attention with similar prototypical parts (provided by our model).

3 Case study 2: car model identification

In this case study, we apply our method to car model identification. We trained our ProtoPNet on the Stanford Cars dataset [20] of 196 car models, using similar architectures and training algorithm as we did on the CUB-200-2011 dataset. The accuracy of our ProtoPNet and the corresponding baseline model on this dataset is reported in Section S6 of the supplement. We briefly state our performance here: the test accuracy of our ProtoPNet is comparable with that of the corresponding baseline model ($\leq 3\%$ difference), and that of a combined network of a VGG19-, ResNet34-, and DenseNet121-based ProtoPNet can reach 91.4%, which is on par with some state-of-the-art models on this dataset, such as B-CNN [25] (91.3%), RA-CNN [7] (92.5%), and MA-CNN [55] (92.8%).

Differences from other papers

Versus CBMs

- ➔ Does not require fine-grained labelling, image-level is fine
- ➔ Tries to separate the prototypical parts while CBM classes can share concepts

Other Case-based Techniques

➔ Main difference is that it is end-to-end, integrated into a neural network with better feature extraction

Prototypical Networks

- ➔ Does not require a decoder (which also produces unrealistic images in natural settings)
- ➔ More fine-grained comparisons, parts of images

Final thoughts

- ➔ Good paper, cool insight
- ➔ Not enough practical guidance
- ➔ Did not display final results very well
- ➔ Could have explained how to extend it to other domains
- ➔ Didn’t really explore the latent space all that much

Thanks for listening!