More comparison with related interpretable methods (R1, R2, R3): In our paper, we discussed the main difference between our ProtoPNet and related attention models in terms of the **type of explanations** offered: **our ProtoPNet not only offers attention on several parts** (akin to attention models), **but also provides similar prototypical cases to those parts** (which attention models cannot provide) **as built-in justification for classification**. In terms of **how attention is generated**: **some attention models generate attention with auxiliary part-localization models trained with part annotations** (e.g. part-based R-CNN (Zhang et al., ECCV 2014), SPDA-CNN (Zhang et al., CVPR 2016), pose-normalized CNN (Branson et al., 2014), DeepLAC (Lin et al., CVPR 2015), part-stacked CNN (Huang et al., CVPR 2016)); **other attention models generate attention with "black-box" methods** – e.g. RA-CNN (Fu et al., CVPR 2017) uses another neural network (attention proposal network) to decide where to look next; multi-attention CNN (Zheng et al., ICCV 2017) uses aggregated conv-feature maps as "part attentions." There is **no explanation** for why the attention proposal network decides to look at some region over others, or why certain parts are highlighted in those conv-feature maps. In contrast, **our ProtoPNet generates attention based on similarity with learned prototypes**: it requires no part annotations for training, and explains its attention naturally – it is looking at *this* region of input because *this* region is similar to *that* prototypical example. Although other attention models focus on similar regions (e.g. head, wing, etc.) as our ProtoPNet, **they cannot be made into a case-based reasoning model like ours**: the only way to find prototypes on other attention models is to analyze *posthoc* what activates a conv-filter of the model most strongly and think of that as a prototype – however, since such prototypes do not participate in the actual model computation, any explanations produced this way are **not always faithful** to the classification decisions.

Hyperparameter choices (R1, R2, R3): In our experiments, we chose $H_1 = 1$ and $W_1 = 1$: given that the spatial dimension of conv-output for a 224×224 image is only 7×7, a 1×1 prototype is already large enough to represent a significant part of the original image in the pixel space (we want to learn prototypes **focused on specific parts**). The number of prototypes can be chosen with prior domain knowledge or hyperparameter search: we used 10 prototypes per class, which should be enough to capture a variety of bird parts (or different views of a car). Section S8 of supplement also discusses prototype pruning to remove non-essential prototypes. The result of pruning is a model with *fewer* and *different* number of prototypes for various classes. We also performed an experiment to see the effect of changing the number of prototypes per class: test accuracy of VGG16-based ProtoPNet is 72.4% with 5 prototypes per class, 76.1% with 10 prototypes per class, and 76.2% with 15 prototypes per class. This shows that having too few prototypes limits performance, but having too many prototypes does not further improve accuracy.

Training algorithm and time complexity (R2): In the algorithm chart (bottom of page): w_{base} and w_{add} denote the parameters of base and additional conv-layers; N_{SGD} and N_{convex} denote the number of training epochs in stage 1 and 3; L and L_{convex} denote the loss function of stage 1 and 3; $\eta_{\text{base}}^{(t)}$, $\eta_{\text{add}}^{(t)}$, $\eta_{\text{p}}^{(t)}$, η_{convex} are learning rates (t denotes epoch #). Prototypes are initialized randomly from uniform distribution over $[0,1]^{H_1 \times W_1 \times D}$ – the last conv-layer uses sigmoid activation, so the conv-features all lie in $[0,1]$. In our experiments, we set $N_{SGD} = 10$ and $N_{\text{convex}} = 20$. This means that prototype projection happens after every 10 SGD epochs. **Feedforward computation of prototype layer has the same time complexity as that of a regular conv followed by global average pooling**, a configuration common in standard CNNs (e.g. ResNet, DenseNet), because the former takes the max of similarity scores computed over all prototype-sized patches while the latter takes the average of dot-products computed over all filter-sized patches. Similarly, **prototype projection has the same time complexity as feedforward part of SGD on standard conv+pooling**, because the former takes the min distance over all prototype-sized patches, and the latter takes the average of dot-products over all patches. Hence, **using prototype layer (to replace the common conv+pooling in a standard CNN) does not introduce extra time complexity**. For a fixed architecture, time complexity of training/testing ProtoPNet is linear in the number of examples, just like any CNN. Empirically, prototype projection takes < 250 seconds for about 6000 training images, **roughly the same time as an SGD epoch** on the same training set using same hardware (1 GPU).

Other questions: **(1) Similarity scores and prediction confidence (R1):** A higher similarity score with a prototype of the predicted class contributes to a more confident final class prediction. **(2) Domains where finding prototypes are useful in itself (R1):** We are currently using this technique to find prototypical tumors in radiology, which can enhance doctors' understanding. **(3) Choice of L^2 (R3):** We choose L^2 because it is a distance metric that is intuitive to understand, and allows us to easily specify the desired **cluster** and **separation** properties of the latent space.

---

initialize $t_0 \leftarrow 0$; $w_{\text{base}} \leftarrow$ weights pre-trained on ImageNet; $w_{\text{add}} \leftarrow$ Kaiming uniform initialization (He et al., 2015);
    $\forall j$: prototype $\mathbf{p}_j \leftarrow \text{Uniform}([0,1]^{H_1 \times W_1 \times D})$; $\forall k, j$: $w_h^{(k,j)} \leftarrow 1$ if $\mathbf{p}_j \in \mathbf{P}_k$, $w_h^{(k,j)} \leftarrow 0$ if $\mathbf{p}_j \notin \mathbf{P}_k$;
while NOT(converge AND Clst $< -$Sep) **do**
  for SGD training epoch $t = t_0 + 1, ..., t_0 + N_{\text{SGD}}$ **do** `/* Stage 1: SGD of layers before the last */`
    foreach batch $[\overline{\mathbf{X}}, \overline{\mathbf{Y}}]$ from $[\mathbf{X}, \mathbf{Y}]$ **do** `/* see Section S9.3 of supplement for learning rates */`
      $w_{\text{base}} \leftarrow w_{\text{base}} - \eta_{\text{base}}^{(t)} \nabla_{w_{\text{base}}} L(\overline{\mathbf{X}}, \overline{\mathbf{Y}})$; $w_{\text{add}} \leftarrow w_{\text{add}} - \eta_{\text{add}}^{(t)} \nabla_{w_{\text{add}}} L(\overline{\mathbf{X}}, \overline{\mathbf{Y}})$; $\mathbf{P} \leftarrow \mathbf{P} - \eta_{\text{p}}^{(t)} \nabla_{\mathbf{P}} L(\overline{\mathbf{X}}, \overline{\mathbf{Y}})$;
  $t_0 \leftarrow t_0 + N_{SGD}$;
  foreach prototype $\mathbf{p}_j$ **do** `/* Stage 2: projection of prototypes */`
    $k \leftarrow$ class of $\mathbf{p}_j$; $\mathbf{p}_j \leftarrow \arg\min_{\mathbf{z} \in \{\tilde{\mathbf{z}} : \tilde{\mathbf{z}} \in \text{patches}(f(\mathbf{x}))\ \forall (\mathbf{x}, y) \in [\mathbf{X}, \mathbf{Y}] \text{ s.t. } y = k\}} \|\mathbf{z} - \mathbf{p}_j\|_2$;
  for convex training epoch $t' = 1, ..., N_{\text{convex}}$ **do** `/* Stage 3: convex optimization of last layer */`
    foreach batch $[\overline{\mathbf{X}}, \overline{\mathbf{Y}}]$ from $[\mathbf{X}, \mathbf{Y}]$ **do** $w_h \leftarrow w_h - \eta_{\text{convex}} \nabla_{w_h} L_{\text{convex}}(\overline{\mathbf{X}}, \overline{\mathbf{Y}})$;
